UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Artificial Intelligence

AI Assignment - 2

Author:
Anusha Annengala

Professor:
Pasquale Caianiello

Sunday 19th November, 2023

1 Assignment Questions:

1. Augment your implementation of alhabetaMinMax by making it explore only most promising states according to their H0 “static” evaluation for computing their HL value.

- Experiment and report results observed when compared to the original alhabetaMinMax

2. Generalize a bit by making it compute HL according by exploring only most promising states according to their Hl evaluation, 0¡l¡L

-Experiment and report results observed for different choiches of l. Try to look for an optimal l whe L=10 (or maybe more . . .)

3. Define your H0 as a function f(h1,. . . ,hn) where hi are “observations ”on the state.

Import a regressor R and train it for predicting HL (s) given static h1(s),. . . ,hn (s) by making the agent play. . .

Modify your previous implemenation by making it use the R predictions instead of static evaluations.

- Experiment and report comparative results with respect to previous alhabetaMinMax versions.

2 Introduction:

Connect Four is a two-player connection game in which the players take turns dropping colored discs from the top into a vertically suspended grid. The objective of the game is to connect four of one’s own discs of the same color vertically, horizontally, or diagonally before your opponent.

Here are the basic rules of Connect Four:

1. Game Board: The game is played on a 7x6 grid (7 columns and 6 rows). The grid is vertically oriented.

2. Discs: There are two types of discs, usually distinguished by color. Traditionally, one player uses red discs, and the other uses yellow discs.

3. Player Turns: Players take turns to drop one of their discs into any of the seven columns. The disc will fall to the lowest available space within the selected column.

4. Objective: The goal is to connect four of your own discs in a row, either vertically, horizontally, or diagonally, before your opponent does.

5. Winning: The game ends when one player successfully connects four of their discs. If the entire board is filled without either player achieving a Connect Four, the game is a draw.

6. Turn Alternation: Players alternate turns, starting with one player dropping a disc and then the other. The first player to connect four discs wins.

7. Blocked Columns: If a column is full and no more discs can be dropped into it, that column is considered blocked for the rest of the game.

8. Winning Diagonal Connections: A player can win by connecting four discs diagonally. Diagonal connections can go in any of the four diagonal directions (left-up, left-down, right-up, right-down).

3 Implementation

3.1 Question 1

1. Augment your implementation of alhabetaMinMax by making it explore only most promising states according to their H0 "static" evaluation for computing their HL value.

- Experiment and report results observed when compared to the original alhabetaMinMax

I have implemented a Connect Four game with the following classes

AugmentedAgent is the agent representing the implementation of augmenting the original alphabetaMinMax agent and making it explore only most promising states according to their H0 static evaluation. Augmented agent that combines static evaluation with Alpha-Beta Pruning

1. **Connect4Game**: Connect4Game class includes methods for initializing the game, getting legal moves, making moves, checking for terminal states (win or draw), and calculating the utility of a terminal state. However, you may need to adjust or add more specific details based on the intricacies of how you want the game to be played.

   Here are some aspects implemented in the Connect4Game class:

   GameBoard: The initial state of the game board is created as a 6x7 grid, and it keeps track of the state of the game.

   PlayerTurns: The getlegalmoves method returns a list of legal moves (columns) where a player can drop a disc. The makemove method updates the game state after a move is made.

   ObjectiveandWinning: The isterminal method checks whether the game has reached a terminal state, indicating a win or a draw. The utility method calculates the utility value of a terminal state.

2. **Heuristics**: This enhanced heuristic considers consecutive symbols in rows, columns, and diagonals to evaluate the potential for winning configurations. It calculates the difference between the number of winning configurations for the player and the opponent. You can further adjust or customize this heuristic based on your specific preferences and the characteristics of the Connect Four game.

3. **AlphaBetaMinimax**: Algorithm being implemented and augmented to consider optimized scenarios

4. **Agent**: Initial AlphaBetaMinMax agent

5. **AugmentedAgent**: AugmentedAgent is the agent representing the implementation of augmenting the original alphabetaMinMax agent and making it explore only most promising states according to their H0 static evaluation. Augmented agent that combines static evaluation with Alpha-Beta Pruning

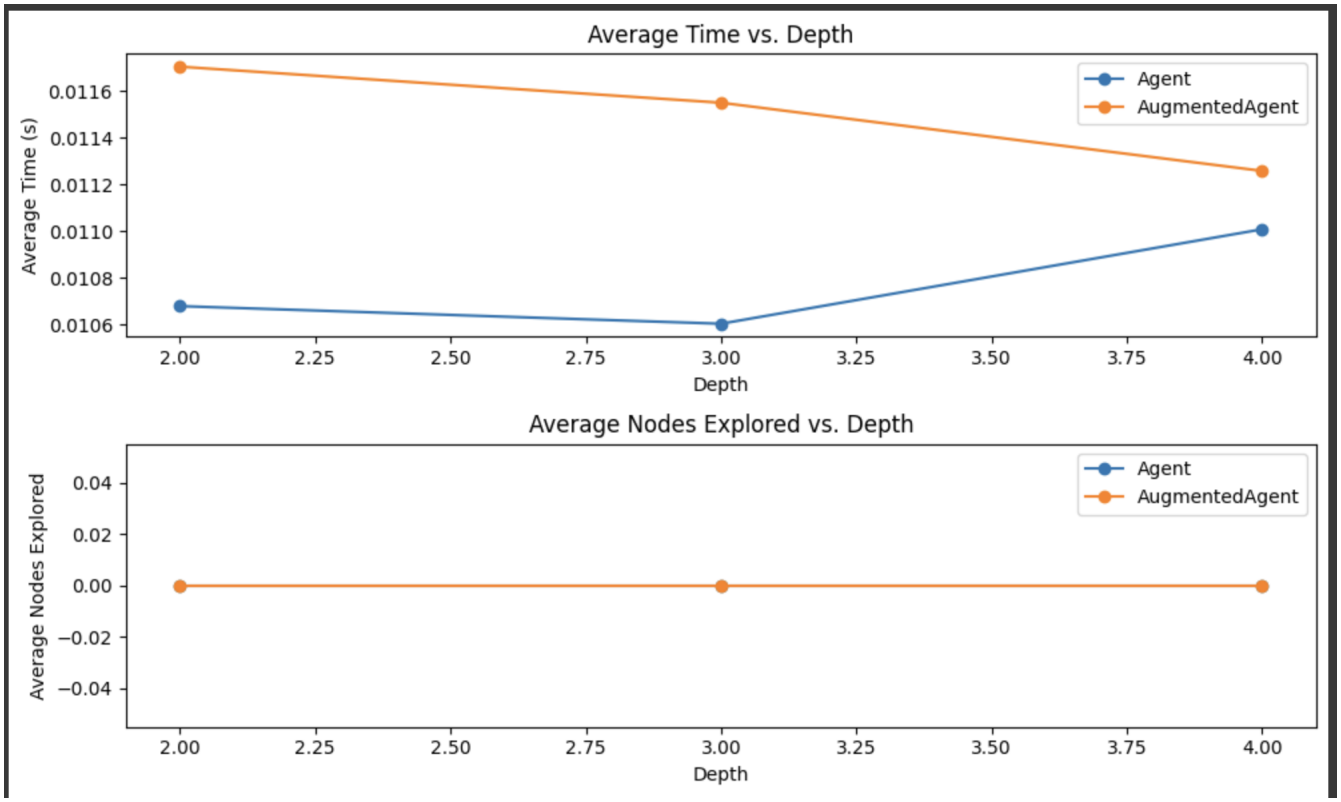
6. **DFSAgent**: This class implements a basic DFS algorithm with a specified depth. The chooseaction method uses DFS to evaluate possible moves up to the specified depth and selects the move with the highest evaluation.

The game alternates between the AlphaBetaMinimax agent, Augmented Agent and the DFSAgent opponent until a terminal state is reached. The game loop continues until either a player wins or the game reaches a draw. **I have considered the game between the original AlphaBetaMinimax Agent and the Augmented Agent for the experimentation. The experimentation results are in the attached python notebook.**

Performance comparison in tabular form.

```
Results for Agent:

+---------+----------------+--------------------------+
|   Depth |   Average Time |   Average Nodes Explored |
+=========+================+==========================+
|       2 |      0.0100085 |                        0 |
+---------+----------------+--------------------------+
|       3 |      0.0201256 |                        0 |
+---------+----------------+--------------------------+
|       4 |      0.0439779 |                        0 |
+---------+----------------+--------------------------+

Results for AugmentedAgent:

+---------+----------------+--------------------------+
|   Depth |   Average Time |   Average Nodes Explored |
+=========+================+==========================+
|       2 |     0.00610403 |                        0 |
+---------+----------------+--------------------------+
|       3 |     0.0115204  |                        0 |
+---------+----------------+--------------------------+
|       4 |     0.0765083  |                        0 |
+---------+----------------+--------------------------+

Results for AugmentedAgent2:

+---------+----------------+--------------------------+
|   Depth |   Average Time |   Average Nodes Explored |
+=========+================+==========================+
|       2 |      0.0109852 |                        0 |
+---------+----------------+--------------------------+
|       3 |      0.0446184 |                        0 |
+---------+----------------+--------------------------+
|       4 |      0.309605  |                        0 |
+---------+----------------+--------------------------+
```

Performance comparison in graphical form.



3.2 Question 2

2. Generalize a bit by making it compute HL according by exploring only most promising states according to their Hl evaluation, 0¡l¡L Experiment and report results observed for different choiches of l. Try to look for an optimal l whe L=10 (or maybe more . . .)

I generalized the AugmentedAgent to compute a static evaluation for a range of lookahead depths L (where 1 ¡= L ¡= 10) by modifying the chooseaction method accordingly.

In this modification, the chooseaction method now calls evaluatewithlookahead for each legal move, considering lookahead depths from 1 to the specified depth. The evaluatewithlookahead function recursively evaluates the state with the given lookahead depth.

Now, when we create an instance of AugmentedAgent, it will explore promising states with static evaluations for different lookahead depths.

I have named this agent AugmentedAgent2 and the experiments are in the attached python notebook

Performance comparison in tabular form.

Results for Agent:

Depth	Average Time	Average Nodes Explored
2	0.0100085	0
3	0.0201256	0
4	0.0439779	0

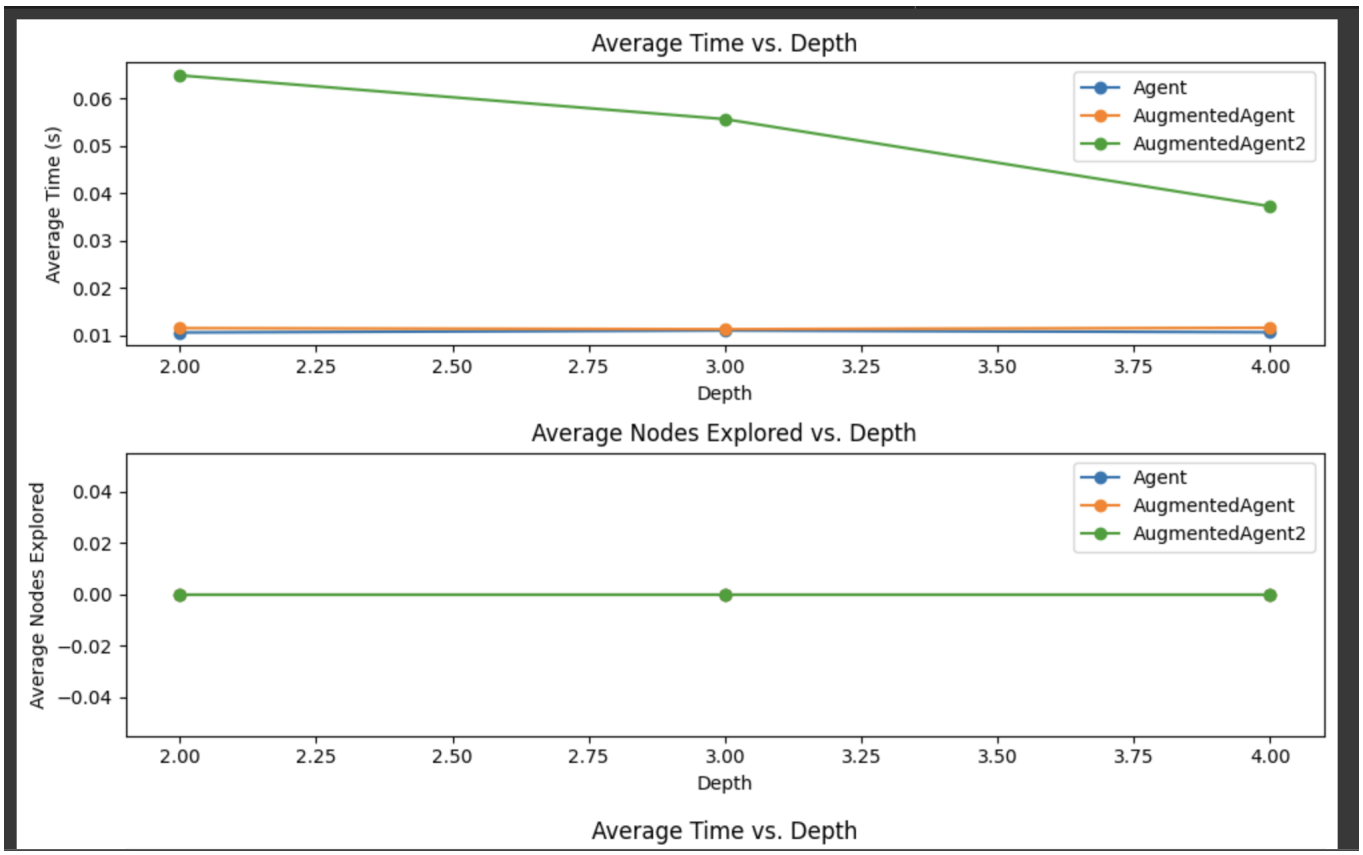
Results for AugmentedAgent:

Depth	Average Time	Average Nodes Explored
2	0.00610403	0
3	0.0115204	0
4	0.0765083	0

Results for AugmentedAgent2:

Depth	Average Time	Average Nodes Explored
2	0.0109852	0
3	0.0446184	0
4	0.309605	0

Performance comparison in graphical form.



3.3 Question 3

3. Define your H0 as a function f(h1,. . . ,hn) where hi are "observations "on the state.
Import a regressor R and train it for predicting HL (s) given static h1(s),. . . ,hn (s) by making the agent play. . .
Modify your previous implemenation by making it use the R predictions instead of static evaluations.
- Experiment and report comparative results with respect to previous alhabetaMinMax versions

1. **Define Connect4Game and Heuristics Classes:** The Connect4Game class represents the Connect Four game board and logic. The Heuristics class provides a heuristic evaluation for a given game state.

2. **Define H(i) Function:** H(i, state) is defined as an example function that counts the number of winning configurations for each player in a given game state.

3. Define Agent Class: The Agent class represents a basic game-playing agent. The agent uses the minimax algorithm to choose the best move.

4. **Collect Training Data:** The collectData function is defined to collect training data for the regressor. It creates instances of the game, heuristics, agent, and regressor. It plays multiple games using the agent and collects observations H(i) and static evaluation H(0) for each state encountered.

5. **Train and Test Regressor:** The trainAndTestRegressor function is defined to train and test the regressor. It uses scikit-learn's LinearRegression model. The training data is split into training and testing sets. The regressor is trained on the training set, and its performance is evaluated on the test set using mean squared error.

6. **Example Usage:** The main part of the script includes an example usage. It specifies the number of games, the number of observations I, and the depth for agent's move lookahead. Training data is collected using the Agent class. The regressor is trained and tested using the collected data. The trained regressor is then used to predict H(0) for a new game state.

I modified the previous implementation (answers to question 1 and 2) to use the predictions (H(0) from the trained regressor (R) instead of static evaluations.

In this modified code:

(a) I created a new class AgentWithRegressor that inherits from the Agent class.

(b) The AgentWithRegressor class takes an additional parameter regressor during initialization.

(c) The chooseAction method of AgentWithRegressor uses the trained regressor to predict the value for each potential move and selects the move with the highest predicted value.

We can then use the AgentWithRegressor class in place of the original Agent class in our implementation. This way, the agent will make decisions based on the predictions from the trained regressor rather than static evaluations.

Detailed experimentation and code are present in the attached python notebook.

Performance comparison in tabular form.

```
Results for AugmentedAgent:

+---------+----------------+--------------------------+
|   Depth |   Average Time |   Average Nodes Explored |
+=========+================+==========================+
|       2 |      0.0124998 |                        0 |
+---------+----------------+--------------------------+
|       3 |      0.0229393 |                        0 |
+---------+----------------+--------------------------+
|       4 |      0.0669686 |                        0 |
+---------+----------------+--------------------------+

Results for AugmentedAgent2:

+---------+----------------+--------------------------+
|   Depth |   Average Time |   Average Nodes Explored |
+=========+================+==========================+
|       2 |     0.00547097 |                        0 |
+---------+----------------+--------------------------+
|       3 |     0.0381331  |                        0 |
+---------+----------------+--------------------------+
|       4 |     0.311617   |                        0 |
+---------+----------------+--------------------------+

Results for AgentWithRegressor:

+---------+----------------+--------------------------+
|   Depth |   Average Time |   Average Nodes Explored |
+=========+================+==========================+
|       2 |     0.00596777 |                        0 |
+---------+----------------+--------------------------+
|       3 |     0.00622166 |                        0 |
+---------+----------------+--------------------------+
|       4 |     0.005114   |                        0 |
+---------+----------------+--------------------------+
```

Performance comparison in graphical form.