UMass CMPSCI 383 (AI) HW2: Chapters 4-6

Yifu Liu

Assigned: Sep 21 2017; Due: Oct 3 2017 @ 11:55 PM EST

Abstract

Submit a (.zip) file to Moodle containing your latex (.tex) file, rendered pdf, and code. All written HW responses should be done in latex (use sharelatex.com or overleaf.com). All code should be written in Python and should run on the edlab machines ([yourusername]@elnux[1,2,...].cs.umass.edu).

1 Gradient Descent

Recall that gradient ascent is a form of hill-climbing to find maxima of a function using the update $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha \nabla f(\mathbf{x}_k)$ with $\alpha > 0$. Note that $\mathbf{x}$ is a vector of real numbers, ie., $\mathbf{x} = [x_1, \ldots, x_n]$. Here, we will use gradient descent to find minima of a function using a similar update:

$$\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - \alpha \nabla f(\mathbf{x}_k), \tag{1}$$

where $\mathbf{x}_k$ is gradient descent's k-th guess at the vector that minimizes $f(\mathbf{x})$. We can expand $\mathbf{x}_k$ as $[x_{1,k}, \ldots, x_{n,k}]$.

1.1 Bottom of a Bowl

As a warm up, consider minimizing the function, $f(x_1, x_2) = x_1^2 + x_2^2$. This function can be written succinctly as $f(x) = ||x||^2$ and is known as the L_2-norm of x. It appears often in machine learning problems. The global minimum of this function is at the origin, $\mathbf{x} = [0, 0]$.

(5 pts) Starting with $\mathbf{x_0} = [2, -1]$, and $\alpha = 0.1$, report the first 3 steps of gradient descent. Does the function appear to be approaching the global minimum at the origin? If not, what's the problem and what's causing it?

$$x_1 = \textbf{[2, -1] - 0.1 * [0.4, -0.2]} \qquad = [1.6, -0.8] \tag{2}$$
$$x_2 = \textbf{[1.6, -0.8] - 0.1 * [3.2, -1.6]} \qquad = [1.28, -0.64] \tag{3}$$
$$x_3 = \textbf{[1.28, -0.64] - 0.1 * [2.56, -1.28]} \qquad = [1.024, -0.512] \tag{4}$$

The function will go to the minimum point, which is (0,0) based on the $\mathbf{x_1}$, $\mathbf{x_2}$, are approaching to 0.

(5 pts) Starting with $\mathbf{x_0} = [2, -1]$, and $\alpha = 1.5$, report the first 3 steps of gradient descent. Does the function appear to be approaching the global minimum at the origin? If not, what's the problem and what's causing it?

$$x_1 = \textbf{[2, -1] - 1.5*[4, -2]} \qquad = [-4, 2] \tag{5}$$
$$x_2 = \textbf{[-4, 2] - 1.5*[-8, 4]} \qquad = [8, -4] \tag{6}$$
$$x_3 = \textbf{[8, -4] - 1.5*[16, -8]} \qquad = [-16, 8] \tag{7}$$

The function will not return to the (0,0), because $\mathbf{x_1}$, $\mathbf{x_2}$ are getting far away from (0,0)

1.2 Non-convex (messy) Function

Now, consider minimizing the 1-d function, $f(x) = 2 + x^2 - 2\cos(10x)$. Note that the argument to cosine is interpreted as radians, not degrees (e.g., $\cos(\pi/2) = 0$). The global minimum of this function is at $x = 0$. When you compute the updates, round $\frac{\partial f}{\partial x}$ to two decimal places. Also round each new x_k to two decimal places after each iteration. In other words,

$$\mathbf{x}_{k+1} \leftarrow \mathit{Round}(\mathbf{x}_k - \alpha \mathit{Round}(\nabla f(\mathbf{x}_k), 2), 2). \tag{8}$$

(5 pts) Starting with $\mathbf{x_0} = 100$, and $\alpha = 0.1$, report the first 3 steps of gradient descent. Does the function appear to be approaching the global minimum at zero? If not, what's the problem and what's causing it?

$$\frac{\partial f}{\partial x} = \mathbf{2x + 20sin(10x)} \tag{9}$$

$$x_1 = \mathbf{100 - 0.1*(216.53)} \quad = 78.35 \tag{10}$$
$$x_2 = \mathbf{78.35 - 0.1*(137.76)} \quad = 64.57 \tag{11}$$
$$x_3 = \mathbf{64.57 - 0.1(109.25)} \quad = 53.65 \tag{12}$$

(5 pts) Starting with $\mathbf{x_0} = 10$, and $\alpha = 0.01$, report the first 3 steps of gradient descent. Does the function appear to be approaching the global minimum at zero? If not, what's the problem and what's causing it?

$$x_1 = \mathbf{10 - 0.01(20 + 20sin(100))} \quad = 9.90 \tag{13}$$
$$x_2 = \mathbf{9.90 - 0.01*(9.90 * 2 -20sin(10 *9.90))} \quad = 9.90 \tag{14}$$
$$x_3 = \mathbf{9.90 - 0.01*(9.90 * 2 -20sin(10 *9.90))} \quad = 9.90 \tag{15}$$

The function will not return to 0, because the result kept resulting in 9.90 (5 pts) Starting with $\mathbf{x_0} = 2.3$, and $\alpha = 0.01$, report the first 3 steps of gradient descent. Does the function appear to be approaching the global minimum at zero? If not, what's the problem and what's causing it?

$$x_1 = \mathbf{2.3 - 0.01(2 * 0.01 + 20sin(10 * 2.3))} \quad = 2.42 \tag{16}$$
$$x_2 = \mathbf{2.42 - 0.01(2 * 0.01 + 20sin(10 * 2.42))} \quad = 2.53 \tag{17}$$
$$x_3 = \mathbf{2.53 - 0.01(2 * 0.01 + 20sin(10 * 2.53))} \quad = 2.44 \tag{18}$$

The function will not return to 0, because the result was in fluctuating around 2.44

2 Constraint Satisfaction Problems

Implement a constraint satisfaction solver for solving Sudoku puzzles. We have provided a `my_hw1.py` template and a Sudoku class in `sudoku.py` with convenient methods (e.g, loading a Sudoku puzzle from a text file). We've given you 6 Sudoku puzzles to debug your code on. We will be testing your code on 10 held-out Sudoku puzzles. You will receive points for getting the correct solution as well as the correct number of guesses needed for your CSP solver to solve the held-out puzzles. Partial credit may be given if you correctly solve the 6 puzzles we've given you, but your solver fails on the held-out puzzles.

2.1 Binary Constraints

(10pts) Show that the number of binary constraints required to represent a Sudoku problem is 810 (Hint: recall that the numbers on each column and row must be distinct, as well as on each 3x3 square).
Each cell has one binary constraint with the other cell.

In the puzzle, there is one constraint between two cells, which makes them cannot take the same value. It can be treated like each cell has 20 small constraints and 8 of them are in the same box and rest of 2*6 are for row and column. Since there are 81 cells, and each of them have two constraints each other, so there are 810 constraints.
The function should be 20 * 81 / 2 = 810

2.2 Backtracking

(20pts) Implement the backtracking algorithm described in Figure 6.5 of the textbook. Your CSP solver should return a list of lists representation of the solved Sudoku puzzle as well as the number of guesses (assignments) required to solve the problem. Please see the comments in the code for more details. In your backtracking search implementation, order the Sudoku tile variables by the order they would be read on a page (left to right, up to down) and order the domain values by ascending order. Include the number of guesses required for each of the 6 practice puzzles below. Also include a folder of your solved Sudoku puzzles as text files in your submission.

puzzle	# of guesses
001	541
026	20079
051	104112
076	45179
090	46719
100	206891

2.3 Heuristics

(15pts) Augment the backtracking algorithm with the MRV (minimum-remaining-value) heuristic, and observe the resulting savings in terms of number of guesses compared to plain backtracking. Include the number of guesses required for each of the 6 practice puzzles below. Also include a folder of your solved Sudoku puzzles as text files in your submission.

puzzle	# of guesses
001	199
026	405
051	909
076	827
090	1035
100	350

3 Alpha-Beta Pruning

Consider the tic-tac-toe problem, where players take turns in placing O's or X's until one of the players has a three-in-a-row (or column or diagonal).

3.1 State Representation

(5pts) Explain how symmetries in the placement of X's and O's on the tic-tac-toe board can be exploited to dramatically shrink the search tree. Draw the resulting search tree down to a depth of 2 given the start state and `TicTacToe-SearchTree-Template` provided at HWs Public/HW2; make a copy and move the copy to your own Google Drive to edit it. You will fill out the figure with minimax values in the next question.

When the 'X' is in the left top corner of the board, when we rotate the board counter clock-wisely for 90/180/270 degrees, the results of the board will have the same results with when you add 'X' in the left-bottom, right-bottom and right-top of the corner. So, that's why the boards can be shrink to the search tree. Same when we put 'X' in the middle of the first row.

3.2 Minimax with Evaluation Heuristic

(10pts) For tic-tac-toe, there are a little less than 9! (362,880) leaf nodes (possible endings of the game). This is too many to draw on a page, so let's consider using an evaluation function, Eval(node), down to a depth of 2. Eval(node) returns N(Player X) - N(Player O) where N(Player i) gives the number of complete rows, columns, and diagonals that are still open for player i. A depth of 2 is too shallow to observe a winning move, but for completeness, define Eval(node) to be +infinity when X wins, and -infinity when O wins. Compute the minimax values for each node in the tree you drew above using the evaluation function, Eval(node). Include the search tree with the minimax values here.

See Figure 1.

3.3 Alpha-Beta Pruning

(15pts) Show the upper and lower bounds for each node that are found using alpha-beta pruning. Show which nodes in the tree will be pruned. Explain why pruning depends on the node ordering. You can use the `TicTacToe-AlphaBetaTree-Template` provided at HWs Public/HW2; make a copy and move the copy to your own Google Drive to edit it.

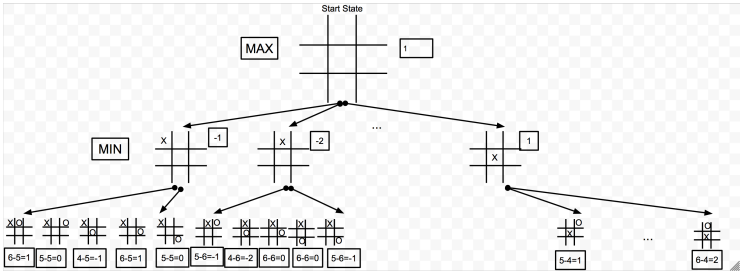

Figure 1: Tic-Tac-Toe Minimax Search Tree

The reason is, in this board, Max always considers the greatest value from Min. In this case, when Max finds that the largest value of middle node is -2, then Max does not have to consider continuously. So when the order of nodes changed, Max may find the larger values first than lower values, so Max has to keep considering the next, however, Max will eventually find the value -2, which lower than the -1. So, Max does not have to consider the next node when he found -2. But the nodes will be pruned based on when -2 appears.

See Figure 2.

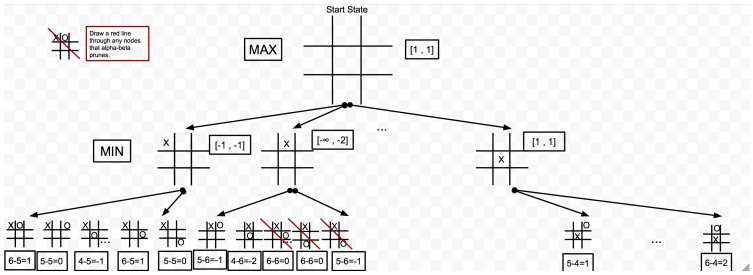

Figure 2: Tic-Tac-Toe AlphaBeta Search Tree