COSI 101A

Machine Problem 3

A Nannon-Schmorgas-board of Machine Learning

Due May 1 11:55 PM via Latte

Backgammon has a storied history in machine learning, but the game is unfortunately quite complex. Even Tesauro found that TD-Gammon needed the addition of handcrafted “expert” features before it reached superhuman levels of play. Nannon, as a simplified version of backgammon, is easier to work with—but machine learning is still a challenge!

The goal of this assignment is for you to gain experience with multiple machine learning approaches. We offer you a number of algorithms presented over the course of the class. You have the choice of three to implement. (You may also try using a different approach, but let us know first.) Broad categories include (1) learning a useful value function or value table for the game, (2) implementing a game-playing neural network, and (3) using reinforcement learning to boost learning.

You should build upon the code you wrote for MP2, but some base code will be provided in Lisp and Python in case you were unable to finish.

This assignment will require some research and reading on your part to fill out your knowledge. Initial resources can be found on Latte in the folder titled “Background Readings for MP3.”

Part 1: Basic Framework (25%)

In addition to your gameplay and tournament functions, you should implement the following:

- *VALUEPLAYER* uses a hash table with estimated values (between -1 and 1) for positions in the game. To use the valueplayer, look up the values after each legal move and choose the best. We will supply a medium-strength player (62% against random) you can use to start.
- *NEUROPLAYER* uses a simple 3-layer network to estimate values (sigmoid returns between 0 and 1, so you will have to scale the output to -1 to 1).
- *EXPECTIMAX* is minimax for Markovian games (with dice). A tournament between the valueplayer and the expectimax 1 or 2 ply player should show some improvement.

Part 2: Machine Learning (60%)

Three out of six is required (each worth 20%). Implement two more for 10% each extra credit.

A) *Pollack & Blair Hillclimbing*. Starting with a neural net with 0 weights, play a tournament between the current network and the network plus small uniform random noise. Pick the winner, and repeat. You may, however, experiment with different mating and scheduling strategies to achieve better learning over generations. (Use of tensorflow or pytorch is OK. A basic neural framework will be provided in Lisp as well.)

B) *Matchbox Reinforcement.* The earliest reinforcement player was MENACE. Use Michie style matchbox reinforcement on all positions stored in a hash table. Play a game and modify the positions in both winner and loser.

C) *Neurogammon-style Backpropagation*. Learn on 50% of the given positions, test sum square error on remaining 50%.

D) *TD-Gammon-style Reinforcement Learning*. Following Tesauro 95, use self-play on a value function expressed as a neural network. Then update the weights backward from the win.

E) *Bellman's Algorithm*. Generate a value table update using expectimax, scanning the whole value table and updating expected values.

F) *AlphaZero.* This learning algorithm was recently used to create a Go engine that played above grandmaster level. Starting with an untrained neural net or random value table, use Monte-Carlo Tree search to look ahead and improve your player through the update process.

Part 3: Writeup (15%)

Provide a 2-3 page discussion of your work including:

- Key details from your research
- Concise descriptions of your implementations.
- Tournament results of your new players benchmarked against the random player
- Observations of individual algorithm performance with different parameter configurations
    - For example, some agents might perform better as the number of training episodes are increased.
    - When are plateaus reached? I.e., at what point does use of CPU power result in negligible improvement?
    - Any interesting results from varying algorithm-specific parameters (e.g., λ in TD).
- Comparisons of the algorithms. Which learn the fastest? Which perform best overall?
- Results of a round-robin tournament including all agents, including those from MP2.

Deliverables

As always, hand in (a) your discussion paper, (b) documented source code, and (c) an edited log, demonstrating the results of running your code. If you use other's packages or libraries, make sure to document that fact.