CS 678 Machine Learning
Programming Assignment #5

Q-Learning for Intelligent Game-playing

Specification

The basic idea is to write an autonomous and adaptive agent capable of learning how to play a more intelligent game of Tic-Tac-Toe. The improvement in the agent's performance must occur via the mechanism of Reinforcement Learning.

Background

The game of Tic-Tac-Toe is simple and familiar, and has a small, well-understood set of rules. Hence it represents a good problem on which to demonstrate the implementation of a program with the capability of learning.

The focus of this project is Reinforcement Learning, the process of discovering which actions yield the highest rewards. The notions of trial-and-error search and maximization of (deferred) long-term reward characterize these types of programs.

Reinforcement learning involves an agent programmed to "explore" its environment, and able to evaluate a reward, i.e. the numerical desirability of a state. Over time, the agent "learns" the tendencies that form a successful policy. Mathematically, we say that the estimated probabilities of success correlated with each state have converged.

One of the methods of implementing this type of learning is called the Temporal Difference method. Let $V(s_t)$ represent an estimate of the value of a state *s* at time *t*, and $V(s_{t+1})$ the estimated value of the temporally *next* state. To learn, the estimate of the original state is updated by a small fraction (η) of the difference between the values of the two states:

$$V(s_t) \leftarrow V(s_t) + \eta[V(s_{t+1}) - V(s_t)]$$

If η, the learning parameter, is reduced slowly over time, then this method converges to the accurate probabilities of winning from each individual state. In subsequent game play, called "exploit" mode, a greedy algorithm can direct the agent to follow a policy that always moves to the state with the highest probability of maximum reward.

A specific form of TD learning is called Q Learning, useful when an environment provides non-deterministic rewards (such as playing a game in which the other player's moves cannot be predicted in advance). In the following equation, a_t represents the action taken at time *t*, *r* is the reward received, and γ is the discount factor (the weight given to new information):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \eta[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Discounting implements the notion of allotting less weight (or reward) for desirable states which are in the distant future, hence have greater uncertainty. The rest of the algorithm operates as a Temporal Difference method: during game play, as states are visited and revisited, the average desirability of each state/action is continuously refined as the expected value.

Implementation

In practice, there is a good bit of variability in the implementation of these algorithms. Design decisions include:

- how should reward the reward function be configured (i.e. weights, punishment, relative value)?
- when should the update function be applied: after each move? or after the entire game has been played and the result is known? The former represents a fairly literal implementation of the equations. The latter implies maintaining a "trace" of the played game states, followed by backpropagation of values.
- should a table containing the expected value of each state be maintained? or should the current probabilities be calculated dynamically? The former uses more storage as it requires storing a value for each possible state of the game.
- how is the learning agent trained? by playing a more rigid (fixed probabilities) version of itself? or by playing a non-learning (random move) version of itself? or by playing itself?

Requirements:

When implementing your solution:

- □ You must use python
- □ You must implement some form of Reinforcement Learning.
- □ Demonstrate the learning of your agent, as represented by improved performance vs. a non-learning opponent.
- □ Experiment with different agent learning strategies, tell us what you discovered.
- □ Submit your report to github (including visualizations, analysis, etc.). If your report does not include your code, be sure to add your code separately.