TD-Gammon

CompSci 590.11

Ron Parr

Duke University

Some game background

- Backgammon is two player, alternating move, zero-sum game
- For now:
  - This class of problems is very similar to MDPs
  - Assume other player makes best move for them (worst move for us)
  - MDP-like algorithms will converge to ***minimax optimal*** strategy
  - Maximizes worst case outcome for us

- AI history: Before RL existed as a term in AI, Arthur Samuel (also from IBM) made a checkers player that learned from experience – used ideas similar to what we call RL today

How most (computer) game players work

- Construct a ***game tree*** of possible moves
- Most interesting games cannot be searched to the end of the game (unless we are already very close to the end)
- Reason: Exponential growth in size of tree
- Players construct a partial game tree
- Use evaluation function to estimate result of searching to game end

- Evaluation function ~ value function
- Reward for winning and/or cost for losing
- Tune this with RL

Some Backgammon background

- Backgammon has dice, so has randomness
- Large state space: 10^{20}
- High branching factor: several hundred (much higher than chess)
- Deep search is impractical – can only do very shallow searches

Previous approaches

- Neuro-gammon viewed backgammon as supervised learning
- Trained NN on database of expert games
- Achieved “strong intermediate” level of play

- Limitations:
  - Experts may not be optimal
  - Experts may be contradictory
  - Nothing to enforce consistency
  - Expert games may see only a fraction of the state space

A note about λ

- When we introduced TD, we considered evaluating entire trajectories before updating vs. updating after each transition
- What if wanted to interpolate between these in some way?
- TD(λ) is an approach that does updates based upon multiple steps
  - TD(1) = Monte Carlo evaluation based on entire trajectories
  - TD(0) = standard TD algorithm
  - TD(λ) – ($0<\lambda<1$) combines both, with lower values closer to standard TD, and higher values closer to Monte Carlo
- Picking good λ = more data efficiency, but doesn’t change the fixed point
- RP:
  - Not always clear how to tweak λ
  - Would rather focus on better features/better algorithm than tweak λ

Training in TD-Gammon

- Initial feature representation was a raw encoding of board positions
- NN was simple by today's standards – 40 hidden nodes
- Main training paradigm was "self play"
- TD-Gammon played both sides

- Achieved "strong intermediate" play after 200K games
- Parity with neuro-gammon, but neuro-gammon had carefully engineered features (Tesauro is a good backgammon player)

TD-Gammon 2.X

- Added 2-ply search
- Expert features from neuro-gammon
- 1.5M games of self play
- Played at master level!

Building on TD-Gammon

- Quite difficult to replicate this success in other domains
- For other games, NN diverged or just didn't play well (e.g. chess, go)

- What's special about backgammon?
  - Tesauro's expert features
  - Possible to do well with linear, suggesting an "on ramp" for the NN
  - Smoothness introduced by randomness
  - Maybe people aren't very good at backgammon?