CS 287: Advanced Robotics
Fall 2009

Lecture 14: Reinforcement Learning with Function Approximation and TD
Gammon case study

Pieter Abbeel
UC Berkeley EECS

Assignment #1

- One basis function, 20, that maps state to the number of ‘holes’ in the state!
- 22 features aka basis functions \( \phi_i \)
  - Ten basis functions, 0, … , 9, mapping the state to the height of each of the ten columns.
  - Nine basis functions, 10, … , 18, each mapping the state to the absolute difference between heights of successive columns: \( |h_{k+1} - h_k| \), \( k = 1, \ldots , 9 \).
  - One basis function, 19, that maps state to the maximum column height: \( max(h) \).
  - One basis function, 20, that maps state to the number of ‘holes’ in the state.
  - One basis function, 21, that is equal to 1 in every state.

Example: tetris

- State: board configuration + shape of the falling piece \( \rightarrow 2^{40} \) states!
- Action: rotation and translation applied to the falling piece
- \( V(s) = \sum_{i=1}^{22} \theta_i \phi_i(s) \)
- 22 features aka basis functions \( \phi_i \)
  - Ten basis functions, 0, … , 9, mapping the state to the height of each of the ten columns.
  - Nine basis functions, 10, … , 18, each mapping the state to the absolute difference between heights of successive columns: \( |h_{k+1} - h_k| \), \( k = 1, \ldots , 9 \).
  - One basis function, 19, that maps state to the maximum column height: \( max(h) \).
  - One basis function, 20, that maps state to the number of ‘holes’ in the state.
  - One basis function, 21, that is equal to 1 in every state.

Recap RL so far

- When model is available:
  - VI, PI, GPI, LP
- When model is not available:
  - Model-based RL: collect data, estimate model, run one of the above methods for estimated model
  - Model-free RL: learn \( V, Q \) directly from experience:
    - TD(\(\lambda\)), sarsa(\(\lambda\)): on-policy updates
    - Q off-policy updates
- What about large MDPs for which we cannot represent all states in memory or cannot collect experience from all states?
  \( \rightarrow \) Function Approximation

Objective

- A standard way to find \( \theta \) in supervised learning, optimize MSE:
  \[ \min_{\theta} \sum P(s) (V(s) - V_{\theta}(s))^2 \]
- Is this the correct objective?
Evaluating the objective

- When performing policy evaluation, we can obtain samples by simply executing the policy and recording the empirical (discounted) sum of rewards

\[ \text{sample sum of rewards at state } s = \left( V^\pi(s) - V^\pi(s) \right)^2 \]

- TD methods: use \( v_t = r_t + \gamma V^\pi(s_{t+1}) \) as a substitute for \( V^\pi(s) \)

\[ \theta_{t+1} = \theta_t + \alpha (r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)) \nabla \theta V^\pi(s_t) \]

TD(\( \lambda \)) with function approximation

- TD(\( \lambda \)) with function approximation: \( [\text{Takahara and Van Roy, 1997}] \)

- \( \lambda \) values are generated from traces of execution of the policy \( \pi \), and for appropriate choice of step-sizes \( \alpha \), TD(\( \lambda \)) converges and at convergence: \( |D - V^\pi| \leq \lambda |\lambda \theta V^\pi - V^\pi| \) (under \( \pi \) and \( \lambda \) assumptions)

\[ |V^\pi - V^\pi| \leq \lambda |\lambda \theta V^\pi - V^\pi| \]

Guarantees

- Monte Carlo based evaluation:
  - Provides unbiased estimates under current policy
  - Will converge to true value of current policy

- Temporal difference based evaluations:
  - TD(\( \lambda \)) with linear function approximation: \( [\text{Takahara and Van Roy, 1997}] \)
  - \( Q \) value under certain assumptions, including: \( |\lambda \theta V^\pi - V^\pi| \leq \lambda \theta V^\pi - V^\pi| \leq 1 \)
  - Same as TD

- Sarsa(\( \lambda \)) with linear function approximation: \( [\text{Melo and Ribeiro, 2007}] \) (Convergence 

Off-policy counterexamples

- Baird's counterexample for off-policy updates:

\[ \theta_{t+1} = \theta_{t+1} + \alpha (r_{t+1} + \gamma \lambda V^\pi(s_{t+2}) - V^\lambda s_{t+1}(s_{t+1})) \]

Combined:

\[ \theta_{t+1} = \theta_{t+1} + \alpha (r_{t+1} + \gamma \lambda V^\pi(s_{t+2}) - V^\lambda s_{t+1}(s_{t+1})) \]

Page 2
Off-policy counterexamples

- Tsitsiklis and Van Roy counterexample: complete back-up with “off-policy” linear regression [i.e., uniform least squares, rather than weighted by state visitation rates]

Intuition behind TD(0) with linear function approximation guarantees

- Stochastic approximation of the following operations:
  - Back-up: \((T^0 V)(s) = \sum_{(s', \pi(s), s')} T(s, \pi(s), s') \pi(s') V(s')\)
  - Weighted linear regression: minimize \(\sum_s D(s)(T^0 V(s) - \theta^T \phi(s))^2\) with solution: \(\theta = (\Phi^T \Phi)^{-1} \Phi^T T^0 \Phi\)

- Key observations:
  \[\forall \lambda_1, \lambda_2: \|T^0 \lambda_1 - T^0 \lambda_2\|_D \leq \gamma \|\lambda_1 - \lambda_2\|_D\]
  \[\forall \lambda_1, \lambda_2: \|\Pi \lambda_1 - \Pi \lambda_2\|_D \leq \|\lambda_1 - \lambda_2\|_D\]

Intuition behind TD(\(\lambda\)) guarantees

- Bellman operator:
  \((T^\lambda J)(s) = \sum_{(s', \pi(s), s')} \pi(s') P(s'|s, \pi(s)) [J(s') + \gamma J(s') - J(s)]\)

- \(T^\lambda\) operator:
  \(T^\lambda J(s) = (1 - \lambda) \sum_{n=0}^\infty \lambda^n \left[ \sum_{a \in A} P(a|s) + \sum_{n=0}^\infty \gamma^{n+1} J(s_{n+1}) \right]\)

- \(T^\lambda\) operator is contraction w.r.t. \(\|\cdot\|_D\) for all \(\lambda \in [0, 1]\)

Should we use TD than well Monte Carlo?

- At convergence: \(\|v - v^\lambda\|_D \leq \frac{1}{1 - \gamma} \|v_0 - v^\lambda\|_D\)

Empirical comparison

(See, Sutton&Barto p.221 for details)

Backgammon

- 15 pieces, try go reach “other side”
- Move according to roll of dice
- If hitting an opponent piece: it gets reset to the middle row
- Cannot hit points with two or more opponent pieces
Backgammon

- 30 pieces, 24+2 possible locations
- For typical state and dice roll: often 20 moves

TD Gammon [Tesauro 92,94,95]

- Reward = 1 for winning the game
- = 0 other states
- Function approximator: 2 layer neural network

Neural net

- Each hidden unit computes:
  \[ h(j) = \sigma(\sum_i w_{ij} \phi(i)) = \frac{1}{1+\exp(-\sum_i w_{ij} \phi(i))} \]
- Output unit computes:
  \[ o = \sigma(\sum_j w_{o} h(j)) = \frac{1}{1+\exp(-\sum_j w_{o} h(j))} \]
- Overall:
  \[ o = f(\phi(1), \ldots, \phi(198); w) \]

Neural nets

- Popular at that time for function approximation / learning in general
- Derivatives/Gradients are easily derived analytically
- Turns out they can be computed through backward error propagation --- name “error backpropagation”
- Susceptible to local optima!

Input features

- For each point on the backgammon board, 4 input units indicate the number of white pieces as follows:
  - 1 piece \( \rightarrow \) unit1=1;
  - 2 pieces \( \rightarrow \) unit1=1, unit2=1;
  - 3 pieces \( \rightarrow \) unit1=1, unit2=1, unit3=1;
  - \( n > 3 \) pieces \( \rightarrow \) unit1=1, unit2=1, unit3=1, unit4 = (n-3)/2
- Similarly for black
  [This already makes for \( 2^4 \times 24 = 192 \) input units.]
- Last six: number of pieces on the bar (w/b), number of pieces that completed the game (w/b), white’s move, black’s move

Learning

- Initialize weights randomly
- TD(\( \lambda \)) \( [\lambda = 0.7, \alpha = 0.1] \)
- Source of games: self-play, greedy w.r.t. current value function [results suggest game has enough stochasticity built in for exploration purposes]
Results

- After 300,000 games as good as best previous computer programs
  - Neurogammon: highly tuned neural network trained on large corpus of exemplary moves
- TD Gammon 1.0: add Neurogammon features
  - Substantially better than all previous computer players; human expert level
- TD Gammon 2.0, 2.1: selective 2-ply search
- TD Gammon 3.0: selective 3-ply search, 160 hidden units