Least Squares Policy Iteration

Ronald Parr
CPS 270

Joint work with: Michail Lagoudakis

Overview

- Motivation
- LSPI
  - Derivation from LSTD
  - Experimental results

Why We *Love* RL

- Ideally, RL agents:
  - Learn continuously by trial and error
  - Correctly attribute credit and blame when causes and effects are not co-temporal
  - Converge to optimal behavior
- RL connects to *beautiful* theory
  - Markov Decision Processes (MDPs)
  - Convergence of stochastic estimators

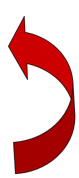
Why We *Hate* RL

- Use for real problems often frustrates
- Reasons:
  - Real problems have huge state spaces
    - Impossible to visit every state
    - Impossible to represent solution exactly
  - Approximation methods are dodgy
    - Require human intervention
    - May not converge
    - Sloooooowwwwww debug cycle

The RL World

- For practical problems RL often involves an "outer loop" with a clever grad student in control:

1. Choose an approximation architecture
2. Run experiments
   - Convergence/Oscillation
   - Good performance/Bad performance
3. Refine approximation architecture

Consequence: RL rarely applied "live"

Example: TD-Gammon

- Brilliant success for RL
  - Plays at level of best human players
  - Inspired a generation of RL researchers
- But…
  - Required hand crafted features
  - Required about 1.5 million games of experience
  - Hard to reproduce:
    - For other implementations
    - For other games

What can we do to help?

- Get more/better grad students **(hard)**
- Automatic approximation architecture selection

- Shorten the cycle
  - Provide more stable RL algorithm **(LSPI)**
  - Reduce data dependence **(LSPI)**

LSPI Teaser

- LSPI is stable and efficient
  - Never diverges or gives meaningless answers
  - Uses efficient linear algebra routines

- LSPI reuses data
  - Remembers past experiences
  - All past experiences relevant to all policies

Optimal Value Function, Policy

Optimal value function, policy satisfy *Bellman* equation:

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

$$\pi^*(s) = \arg\max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

- If P,R are known, solve MDP:
  - VI, PI, LP
  - Poly time in number of states
- Otherwise, we use RL

Intuitions for VFA

- Leverage generalization power of machine learning to produce approximate values for all states while considering only a tiny fraction

- Dramatic success in some areas
  - Backgammon
  - Elevator scheduling

- Dramatically frustrating in others…

Implementing VFA

- Can't represent Value Function as a big vector
- Use (parametric) function approximator
  - Neural network
  - Linear regression (least squares)
  - Nearest neighbor (with interpolation)
- (Typically) sample a subset of the the states
- Use function approximation to "generalize"

Approximate Solutions

- The standard Bellman equation:

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s')$$

- With approximation

$$\hat{V}^*(s) = \Pi\left(\max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) \hat{V}^*(s')\right)$$

- Π is a *projection* operator
  - Projects into space of representable value functions
  - Often implicit

Problem 1: Stability

- Exact value iteration, Q-learning stability ensured by contraction of:

$$V^{i+1}(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^i(s')$$

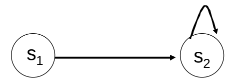
- Is this a contraction:

$$\hat{V}^{i+1}(s) = \Pi\left(\max_a R(s,a) + \gamma \sum_{s'} P(s'|s,a) \hat{V}^i(s')\right)$$

?

Stability Problem

Problem: Most VFA methods are unstable

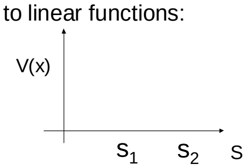


No rewards, $\gamma = 0.9$: $V^* = 0$

Example: Bertsekas & Tsitsiklis 1996

Least Squares Approximation

Restrict V to linear functions:

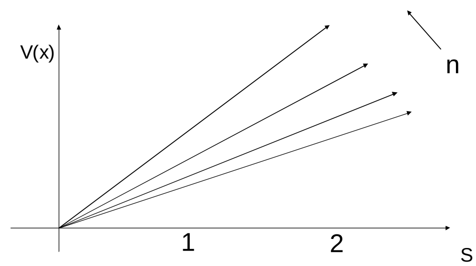


Find θ s.t. $V(s_1) = \theta$, $V(s_2) = 2\theta$

Counterintuitive Result: If we do a least squares fit of θ

$\theta^{t+1} = 1.08\, \theta^t$

Unbounded Growth of V



Understanding the Problem

- What went wrong?
  - VI reduces error in maximum norm
  - Least squares (= projection) non-expansive in L_2
  - May increase maximum norm distance
  - Grows max norm error at faster rate than VI
- Can't this be fixed by sampling trajectories?
  - Yes (VI is also a projection in weighted L_2)
  - Dubious usefulness for policy improvement!

Problem 2: Efficiency

- Most RL methods are gradient based
- Q-learning:

$$Q^{i+1}(s,a) = (1-\alpha)Q^i(s,a) + \alpha\left(r + \gamma V^i(s',a)\right)$$
$$V^i(s',a) = \max_a Q^i(s,a)$$

- Convergence requires:
  - Small steps (small α)
  - Visiting every state infinitely often

Overview

- Motivation
- LSPI
  - Derivation from LSTD
  - Experimental results

How does LSPI fix these?

- LSPI is based on LSTD
- Policy evaluation alg. by Bratdke & Barto 96
- Stability:
  - LSTD directly solves for the fixed point of the approximate Bellman equation
  - With SVD, this is always well defined
- Data efficiency
  - LSTD finds best solution for any finite data set
  - Single pass over data
  - Can be implemented incrementally

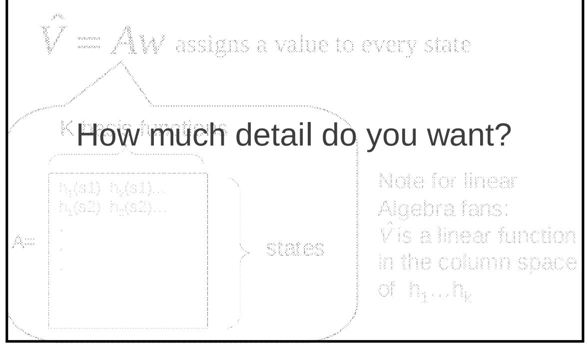
OK, What's LSTD?

- Least Squares Temporal Difference Learning
- Linear value function approximation

$$\hat{V}(s) = \sum_k w_k h_k(s)$$

- NOT necessarily linear in state variables
- Each h_k can be an arbitrary function
- Compare with neural nets

Deriving LSTD



How much detail do you want?

Suppose we know V*

- Want:

$$Aw \approx V^*$$

- Projection minimizes squared error

$$w = \underbrace{(A^T A)^{-1} A^T}V^*$$

Textbook least squares projection

But we don't know V*…

- Require consistency:

$$\hat{V}^* = \Pi\left(R(s,a) + \gamma P \hat{V}^*\right)$$

- Substituting least squares projection

$$Aw = A(A^T A)^{-1} A^T \left(R(s,a) + \gamma P A w\right)$$

- Solving for w

$$w = (A^T A - A^T P A)^{-1} A^T R$$

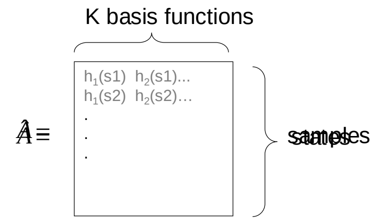
Almost there…

$$w = (A^T A - A^T PA)^{-1} A^T R$$

- Matrix to invert is only k x k
- But…
  - Expensive to construct matrix
  - We don't know P
  - We don't know R

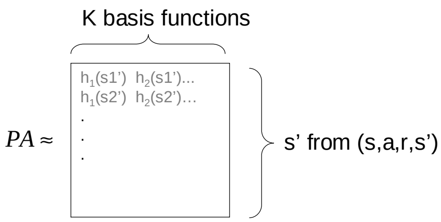
Using Samples for A

Idea: Replace enumeration of states with sampled states



Using Samples for PA

Idea: Replace expectation over next states with sampled next states.



Putting it Together

- LSTD needs to compute:

$$w = (A^T A - A^T PA)^{-1} A^T R$$

- The hard part of which is the kxk matrix:

$$B = A^T A - A^T PA$$

- For each (s,a,r,s') sample:

$$B_{ij} \leftarrow B_{ij} + h_i(s)h_j(s) + h_i(s)h_j(s')$$

LSTD Summary

- Does O(k²) work per datum
- Approaches model-based answer in limit
- Finding fixed point requires inverting matrix

- Fixed point almost always exists
- Can use SVM if B is singular

- Stable; efficient

Policy Iteration with LSTD

Guess $\hat{V}_i(s,\mathbf{w})$

π_{i+1} = greedy($\hat{V}_i(s,\mathbf{w})$)

$\hat{V}_{i+1}(s,\mathbf{w})$= value of acting on π_{i+1}

Increment i
Repeat until???

Use LSTD here?

What Breaks?

- No way to pick actions
- Approximation is biased by current policy
  - We only approximate values of states we see
  - LSTD is a *weighted* approximation
- Learn-forget cycle of policy iteration
  - Drive off the road; learn that it's bad
  - New policy never does this; forgets that it's bad

LSPI

- LSPI makes LSTD suitable for Policy Iteration
- LSTD: state -> state
- LSPI: (state, action) -> (state, action)
- Similar to Q learning
- Implementation is subtle
- Has deep consequences:
  - Disconnects policy evaluation from data collection
  - Permits reuse of data across iterations

Implementing LSPI

- Both LSTD and LSPI must compute:

$$B = A^T A - A^T PA$$

- But LSPI has a factor of (#$\mathcal{A}$) more basis fns
- Duplicate basis functions for each action:
  - $h_i^{a1}(s) = h_i(s)$ if a_1 taken, 0 otherwise,
  - $h_i^{a2}(s) = h_i(s)$ if a_2 taken, 0 otherwise,etc
- For each (s,a,r,s') sample:

$$B_{ij} \leftarrow B_{ij} + h_i^a(s) h_j^a(s) - h_i^a(s) h_j^{\pi(s')}(s')$$

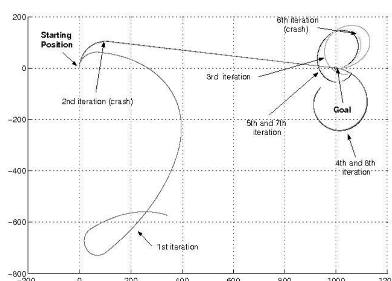
Running LSPI

- Start w/random weights (= random policy)
- Collect a database of (s,a,r,s') experiences
- Repeat
  - Evaluate current policy against database
    - Run LSPI to generate new set of weights
    - New weights imply new policy
  - Replace current weights with new weights
- Until convergence (or ε weight change)

Results: Bicycle Riding

- Randlov and Alstrom simulator
- Watch random controller operate bike
- Collect ~60,000 (s,a,r,s') samples
- Pick 20 simple basis functions (×5 actions)
- Make 5-10 passes over data (PI steps)
- Result:
  Controller that balances and rides to goal

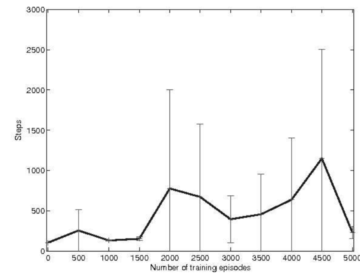
Bicycle Trajectories



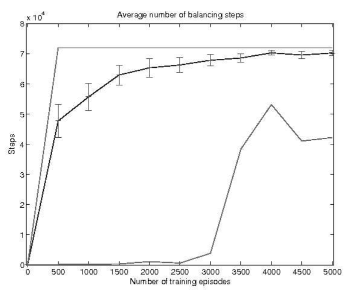
What about Q-learning?

- Bicycle "solved" using CMAC
  - CMAC is very expressive
  - Trajectories were not that tight
- Compare with same architecture
- Use experience replay for data efficiency

Q-learning Results



LSPI Robustness



So, what's the bad news?

- (k (#A))^2 can sometimes be big
  - Lots of storage
  - Matrix inversion can be expensive
- Linear VFA is "weak"
- Bicycle needed shaping
- Still haven't solved
  - Feature selection
  - Exploration vs. Exploitation