Least Squares Policy Iteration

Ronald Parr
CPS 271

Joint work with: Michail Lagoudakis

Overview

- Motivation
- LSPI
  - Derivation from LSTD
  - Experimental results

The RL Story

- MDPs, Decision theory tell us how to act optimally
- Beautiful theory – hard to use in practice
- Problem: Satisfying the Markov property means that there are usually way too many states
- Q: How can machine learning come to the rescue?

Need for Function Approximation

- MDPs
  - State space with |S| states
  - n state variabes (fluents) imply $|S|=2^n$
  - Need to assign actions to all |S| states
  - Continuous state spaces are problematic
- Many MDP/RL algorithms use value functions
- How can we use our expertise in machine learning to extrapolate values for the entire state even if we have visited only a small fraction of it?

Example: TD-Gammon

- Used a neural network to represent value function
- 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 million games of experience
  - Hard to reproduce:
    - For other implementations
    - For other games

Standard RL Approaches

- Reinforcement often presented as stochastic gradient descent
- Agent observes (s,a,r,s')
- Adjusts value function representation to make v(s) closer to $r+\gamma v(s')$
- Surprisingly, these approaches can diverge or oscillate when standard stochastic gradient does not
- We diverge from the standard view and present RL from a linear regression viewpoint

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

Terminology

- S: state space
- s: individual states
- R: reward
- γ: discount
- V: state value
- Q: state-action value
- Policy: $\pi(s) \rightarrow a$

Objective: *Maximize expected, discounted return*

$$\mathrm{E}\sum_{t=0}^{\infty} \gamma^t r_t$$

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

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:

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

- "Fixed Point" Bellman Equation With approximation

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

- 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:

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

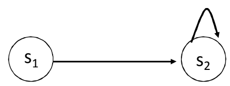
- Is this a contraction:

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

?

Stability Problem

Problem: Most VFA methods are unstable

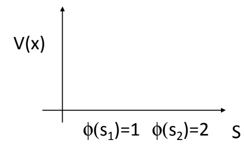


No rewards, $\gamma = 0.9$: $V^* = 0$

Example from Bertsekas & Tsitsiklis 1996

Least Squares Approximation

Restrict V to linear functions:

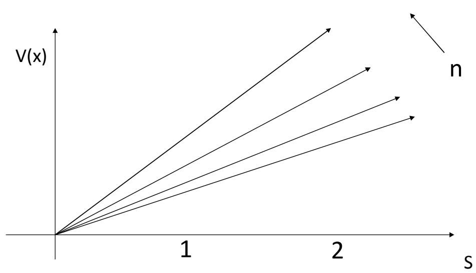


Find θ s.t. $V(s_1) = \theta$, $V(s_2) = 2\theta$

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
- Conclusion: Alternating value iteration and regression is risky business

Problem 2: Efficiency

- Most RL methods can be viewed as stochastic gradient descent of some kind
- 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
  - Makes a single pass over the data for each policy
  - Can be implemented incrementally

OK, What's LSTD?

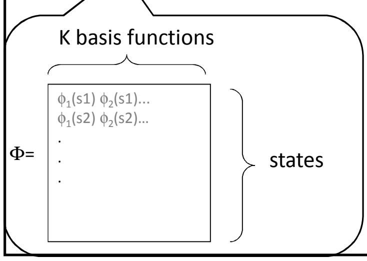
- Least Squares Temporal Difference Learning
- Linear value function approximation

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

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

Deriving LSTD

$\hat{V} = \Phi w$ assigns a value to every state



$\hat{V}$ is a linear function in the column space of $\phi_1 \ldots \phi_k$

Suppose we know V*

- Want:

$$\Phi w \approx V^*$$

- Projection minimizes squared error

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

Textbook least squares projection

But we don't know V*…

- Require consistency:

$$\hat{V}^* = \prod\left(R + \gamma P \hat{V}^*\right)$$

- Substituting least squares projection

$$\Phi w = \Phi(\Phi^T \Phi)^{-1} \Phi^T \left(R + \gamma P \Phi w\right)$$

- Solving for w

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

Almost there…

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

- 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 Φ

Idea: Replace enumeration of states with sampled states

K basis functions

$$\hat{\Phi} \equiv \begin{bmatrix} \phi_1(s1)\ \phi_2(s1)\ldots \\ \phi_1(s2)\ \phi_2(s2)\ldots \\ \vdots \end{bmatrix}$$

samples states

Using Samples for PΦ

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

K basis functions

$$P\Phi \approx \begin{bmatrix} \phi_1(s1')\ \phi_2(s1')\ldots \\ \phi_1(s2')\ \phi_2(s2')\ldots \\ \vdots \end{bmatrix}$$

s' from (s,a,r,s')

Putting it Together

- LSTD needs to compute:

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

- The hard part of which is the kxk matrix:

$$B = \Phi^T \Phi - \Phi^T P\Phi$$

- This can be done incrementally, for each (s,a,r,s') sample:

$$B_{ij} \leftarrow B_{ij} + \phi_i(s)\phi_j(s) + \phi_i(s)\phi_j(s')$$

LSTD Summary

- Does O(k^2) 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:

$$B = \Phi^T \Phi - \Phi^T P \Phi$$

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

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

Running LSPI

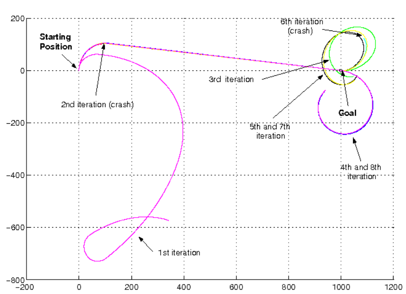
- Start w/random weights (= random policy)
- Collect a database of (s,a,r,s') experiences
- Repeat
  - Evaluate current policy against database
    - Run LSPI to generate new set of weights
    - New weights imply new policy
  - Replace current weights with new weights
- Until convergence (or e weight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

Results: Bicycle Riding

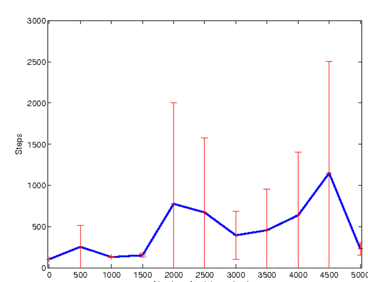
- Randlov and Alstrom simulator
- Watch random controller operate bike
- Collect ~40,000 (s,a,r,s') samples
- Pick 20 simple basis functions (×5 actions)
- Make 5-10 passes over data (PI steps)

- Result:
  Controller that balances and rides to goal

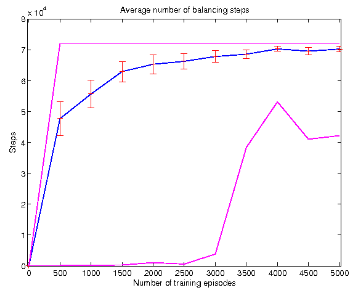
Bicycle Trajectories



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" rewards
- Still haven't solved
  - Feature selection (issue for all machine learning, but RL seems even more sensitive)
  - Exploration vs. Exploitation

Conclusion

- Reinforcement learning combines decision theory with machine learning techniques
- Key idea: Avoid covering the large state space imposed by adherence to Markov property
- Key challenges:
  - Stability
  - Non-linearity introduced by max in Bellman equation
  - Feature/model selection
  - Exploration vs. Exploitation
- Many methods exist for RL
- LSTD/LSPI represent one family of methods closely tied to linear regression