Batch Reinforcement Learning (LSTD and LSPI)

CPS 271
Ron Parr

With thanks to Alan Fern for feedback on slides
LSPI is joint work with Michail Lagoudakis
Equivalence between the linear model and LSTD is joint work
with Li, Littman, Painter-Wakefield and Taylor

Online versus Batch RL

• Online RL:
  – Concurrent data collection and optimization
  – Example: TD and Q-learning

• Batch RL:
  – Decouples data collection and optimization
  – Generate database of experiences in the environment
  – Use the fixed set of experiences to optimize/learn a policy

• Online vs. Batch:
  – Batch algorithms are often more “data efficient” and stable
  – Batch algorithms ignore the exploration-exploitation problem, and do their best with the data they have
  – Not necessarily a crisp distinction: Batch algorithms can be interleaved with exploration to augment database
Model Learning

• We considered model learning as our first “straw man” approach for RL
• Can learn reward, distributions over next states for each state
• Discarded as impractical because it required too much data and storage
• Idea: What if we learned linear approximations, i.e., what if we used our old trick, regression?

Linear Model Approximation
(Policy Evaluation Case)

• Linearly independent features \( \Phi=(\phi_1\ldots\phi_k) \) \((n \times k)\)

• Want \( R_\Phi = \text{reward model} \) \((k \times 1)\) w/smallest \( L_2 \) error:
  \[
  \Phi R_\Phi \approx R
  \]
  \[
  R_\Phi = \left( \Phi^T \Phi \right)^{-1} \Phi^T R = \Pi_\Phi R
  \]

• Want \( P_\Phi = \text{feature} \times \text{feature model} \) \((k \times k)\) w/ smallest \( L_2 \) error

\[
\Phi P_\Phi \approx R \Phi = \Pi_\Phi P(= \Pi_\Phi \Phi')
\]

\[
P_\Phi = \Pi_\Phi P(= \Pi_\Phi \Phi')
\]

\(\approx\) Expected (or sampled)
next feature values \((n \times k)\)
State Values Under Linear Model

- Bellman equation:
  \[ V(s) = R(s) + \gamma \sum_{s'} P(s'|s)V(s') \]

- Bellman equation in terms of features
  \[ V(\phi(s)) = R(\phi(s)) + \gamma V(\phi(s')) \]
  \[ = \phi(s)^T R_\phi + \gamma V(\phi(s)P_\phi) \]
  \[ = \phi(s)^T R_\phi + \gamma \left( \phi(s)^T P_\phi R_\phi + \gamma V(\phi(s)^T P'_{\phi}) \right) \]
  \[ = \sum_{i=0}^{\infty} \gamma^i \phi(s)^T P'_i R_\phi = \phi(s)^T \left( \sum_{i=0}^{\infty} \gamma^i P'_i \right) R_\phi = (I - \gamma P_\phi)^{-1} R_\phi \]

Value Function of the Linear Model

- Value function is in \( \text{span}(\Phi) \)
- Can express value functions as \( \Phi w \)
- If \( V \) is bounded, then:
  \[ w = (I - \gamma P_\phi)^{-1} R_\phi \]

  \( (k \times 1) \) \( (k \times k) \) \( (k \times 1) \)

- Note similarity to conventional solution:
  \[ V^* = (I - \gamma P)^{-1} R \]

  \( (n \times 1) \) \( (n \times n) \) \( (n \times 1) \)
Solving for the Value Function Directly

- LSTD [Bradtke & Barto] aims to estimate the value function directly from samples
- Give $\Phi, \Phi'$, and sampled $R$, LSTD computes

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

- Where does this come from?

Deriving LSTD

- Recall fixed-point equation for policies

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

- For linear value function approximation:

$$V = R + \gamma \sum_{s'} P(s'|s) \phi w$$

- Problem: This might not be in $\text{span}(\Phi)$
- Solution project back into $\text{span}(\Phi)$

$$\hat{V}^\pi = \Phi w = \prod_{\phi} (R + \gamma P \Phi w) \approx \prod_{\phi} (R + \gamma \Phi' w)$$

- Substituting least squares projection into this gives:

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

$$w = (\Phi^T \Phi - \gamma \Phi^T P \Phi)^{-1} \Phi^T R = (\Phi^T \Phi - \gamma \Phi' \Phi')^{-1} \Phi^T R$$
LSTD Solution = Linear Model Solution

- LSTD solution: \( w = (\Phi^T \Phi - \gamma \Phi^T \Phi')^{-1} \Phi^T R \)

- Linear model solution:
  \[
  w = \left( I - \gamma (\Phi^T \Phi)^{-1} \Phi^T \Phi' \right)^{-1} (\Phi^T \Phi)^{-1} \Phi^T R
  = (\Phi^T \Phi - \gamma \Phi^T \Phi')^{-1} \Phi^T R
  \]

- Conclusion: LSTD and solution to linear model approximation are identical
- Historical note: People did not initially realize this

Approximate Policy Iteration with LSTD

**Policy Iteration**: iterates between policy improvement and policy evaluation

**Idea**: use LSTD for approximate policy evaluation in PI

Start with random weights \( w \) (i.e. value function)
Repeat Until Convergence

- \( \pi(s) = \text{greedy} (\Phi w) \)
- Evaluate \( \pi \) using LSTD
  - Generate sample trajectories of \( P\pi \)
  - Use LSTD to produce new weights \( w \) (\( w \) gives an approx. value function of \( \pi \) )
What Breaks?

- No way to execute greedy policy without a model

- Approximation is biased by current policy
  - We only approximate values of states we see when executing the current policy
  - LSTD is a weighted approximation toward those states

- Can result in 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 is similar to previous loop by replaces LSTD with a new algorithm LSTDQ
- LSTD: produces a value function
  - Requires sample from policy under consideration
- LSTDQ: produces a Q-function ($Q_\pi$, not necessarily $Q^*$)
  - Can learn Q-function for policy from any (reasonable) set of samples---sometimes called an off-policy method
  - No need to collect samples from current policy
- Disconnects policy evaluation from data collection
  - Permits reuse of data across iterations
Computing Q-functions w/LSTDQ

- Suppose we have samples of form \((s, a, r, s')\)
- Expand our state space to include the actions taken as part of the state
- View our samples as \(([s, a], r, [s', \pi(s')])\)
  - Q: Is this valid if we didn’t really take \(\pi(s')\) in \(s'\)?
  - A: Yes, because the right hand side of the Bellman equation just uses our linear approximation:

\[
Q_\pi(s, a) = R(s) + \sum_{s'} P(s' \mid s, a) V(s') = R(s) + \sum_{s'} P(s' \mid s, a) Q_\pi(s', \pi(s'))
\approx R(s) + \sum_{s'} P(s' \mid s, a) \phi_{a_1}(s') V_{s'}(s')
\]

Implementing LSPI

- Implement LSTDQ as LSTD with expanded feature set:
  - For A actions, make A copies of our features
  - Features are replaced with product of features and indicators
  - \(\phi(s) \rightarrow \phi_{a_1}(s) = \phi(s)(a_1), \phi_{a_2}(s) = \phi(s)(a_2), \ldots\)
- Each time we run LSTDQ, our Q values encode the next policy, i.e.

\[
\pi_{i+1}(s) = \arg\max_a Q_{\pi_i}(s, a)
\]
Running LSPI

• There is a Matlab implementation available!

1. Collect a database of \((s,a,r,s')\) experiences
2. Start \(w/\text{random weights} (= \text{random policy})\)

3. Repeat
   • Evaluate current policy against database
     • Run LSTDQ to generate new set of weights
     • New weights imply new Q-function and hence new policy
   • Replace current weights with new weights

• Until convergence
  (for some notion of convergence, usually small change in weights)

Nota bene: LSPI may not converge!

What’s under the hood?

• At each iteration, we’re basically doing LSTD:

\[
w = (\Phi^T \Phi - \gamma \Phi^T \Phi')^{-1} \Phi^T R
\]

• What changes at each iteration?

\[
w = (\Phi^T \Phi - \gamma \Phi^T \Phi')^{-1} \Phi^T R
\]

When the policy changes, the \(\pi(s')\) part of \([s',\pi(s')]\), changes. That’s it!
In practice, all that changes is which indicators are active in \(\Phi'\).
Everything else can be cached.
Example 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
What about Q-learning?

• Bicycle “solved” using CMAC
  – CMAC is very expressive
  – Trajectories were not that tight

• We compare with same architecture (linear)
• Use experience replay for data efficiency

Q-learning Results
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