Batch Reinforcement Learning
(Emphasizing LSTD and LSPI)

CompSci 590
Duke University
Ronald 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-learning 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
  – Example: Fitted Value Iteration

• Online vs. Batch:
  – Batch algorithms are often more “data efficient” and stable
  – Batch algorithms typically 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
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

\[
\Phi P_\Phi \approx P_\Phi (\approx \Phi')
\]

\[
P_\Phi = \Pi_\Phi \rho \Phi (\approx \Pi_\Phi \Phi')
\]

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

\[
V(\phi(s)) = R(\phi(s)) + \gamma V(\phi(s))
\]

\[
= \phi(s)^T R_\Phi + \gamma V(\phi(s)^T P_\Phi)
\]

\[
= \phi(s)^T R_\Phi + \gamma \left( \phi(s)^T P_\Phi R_\Phi + \gamma V(\phi(s)^T P_\Phi) \right)
\]

\[
= \sum_{t=0}^{\infty} \gamma^t \phi(s)^T P_\Phi R_\Phi = \phi(s) \left( \sum_{t=0}^{\infty} \gamma^t P_\Phi \right) R_\Phi = \phi(s) (I - \gamma P_\Phi)^{-1} R_\Phi
\]
Value Function of the Linear Model

• Value function is in \text{span}(\Phi)
• Can express value functions as \Phi w
• If \text{V} is bounded, then:

$$w = (I - \gamma P_\Phi)^{-1} R_\Phi$$

\( (k\times1) \hspace{1cm} (k\times k) \hspace{1cm} (k\times1) \)

• Note similarity to conventional solution:

$$V^* = (I - \gamma P)^{-1} R$$

\( (n\times1) \hspace{1cm} (n\times n) \hspace{1cm} (n\times1) \)

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?
Linear Fixed Point

- \( \Pi_\Phi V = \text{weights of projection of } V \text{ into } \text{span}(\Phi) \)

- LSTD solves for the linear fixed point:
  \[
  w_\phi = \Pi_\Phi \tau \Phi w_\phi
  \]

Deriving LSTD

- Recall fixed-point equation for policies
  \[
  V'(s) = R(s, \pi(s)) + \gamma \sum_s P(s \mid s, \pi(s)) V'(s)
  \]
- For linear value function approximation:
  \[
  V = R + \gamma \sum_s P(s \mid s, \pi(s)) \Phi w
  \]
- Problem: This might not be in \( \text{span}(\Phi) \)
- Solution project back into \( \text{span}(\Phi) \)
  \[
  \hat{V} = \Phi w = \Pi_\Phi (R + \gamma P \Phi w) = \Pi_\Phi (R + \gamma \Phi' w)
  \]
- Substituting least squares projection into this gives:
  \[
  \Phi w = \Phi (\Phi^T \Phi)^{-1} \Phi^T (R + \gamma P \Phi w)
  \]
  \[
  w = (\Phi^T \Phi - \gamma \Phi^T P \Phi)^{-1} \Phi^T R = (\Phi^T \Phi - \gamma \Phi^T \Phi')^{-1} \Phi^T R
  \]
LSTD Solution = Linear Model Solution

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

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

• Conclusion: LSTD and solution to linear model approximation are identical
• Historical notes:
  – Indicator function case was clear from the beginning, but
  – People did not initially realize this equivalence in full generality

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 \( \mathbf{w} \) (i.e. value function)
Repeat Until Convergence
  \( \pi(s) = \text{greedy}(\Phi \mathbf{w}) \)
  Evaluate \( \pi \) using LSTD
  • Generate sample trajectories of \( P\pi \)
  • Use LSTD to produce new weights \( \mathbf{w} \)
    (\( \mathbf{w} \) 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 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 \(\left([s,a],r,[s',\pi(s')]\right)\)
  - 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:

\[
Q(s,a) = R(s) + \sum_{s'} P(s'|s,a)V(s') = R(s) + \sum_{s'} P(s'|s,a)Q(s', \pi(s))
\]
\[
= R(s) + \sum_{s'} P(s'|s,a)\phi_{\pi(s)}(s)w_{\pi(s)}
\]

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
  - \(\phi(s)\rightarrow\phi_{a_1}(s) = \phi(s)(a_1), \phi_{a_2}(s) = \phi(s)(a_2),...\)
- Each time we run LSTDQ, our Q-values encode the next policy, i.e.

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

1. Collect a database of \((s,a,r,s')\) experiences
2. Start \(w/\)random weights (= random policy)
3. Repeat until convergence*
   - 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

*Nota bene: LSPI may not converge; need some notion of “convergence”

What’s under the hood?

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

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

• What changes at each iteration?

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

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

Bicycle Trajectories
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
LSPI as an Instance of Approximate Policy Iteration

- Analysis of approximate policy iteration from Neuro-Dynamic Programming, Bertsekas and Tsitsiklis, 1996

\[ \lim_{k \to \infty} \sup \| V^{\pi_k} - V^* \|_\infty \leq \frac{\delta + 2 \gamma \varepsilon}{(1 - \gamma)^2} \]

- \( \varepsilon = \text{worst max-norm value function error} \)

\[ \varepsilon = \max_k \| V^{\pi_k} - V^{\tilde{\pi}_k} \|_\infty \]

- \( \delta = \text{max-norm policy update error} \)

\[ \delta = \max_k \| TV^{\pi_k} - \tilde{T}V^{\tilde{\pi}_k} \|_\infty \]

LSPI Error Bounds in Practice

- No guarantee matrix is invertible (though almost always is)
- No practical guarantee on \( \varepsilon \)
- \( \delta = 0 \)
Fitted Q-Iteration

- Can we apply the Q-function trick to fitted value iteration?
- Yes!
- Fitted Q-iteration algorithm:
  - Randomly initialize approximate Q function $Q_0$
  - $i=0$
  - Repeat until done*
    - Sample states $s^1...s^m$
    - $\pi = \text{greedy}(Q_i)$
    - Fit $Q_{i+1}$ on $T^\pi Q_i(s^1)...T^\pi Q_i(s^m)$
    - $i=i+1$
- Achieves value iteration w/o a model

LSPI vs. Fitted Q-iteration

<table>
<thead>
<tr>
<th></th>
<th>Representation Expressiveness</th>
<th>Can Diverge</th>
<th>Can Oscillate</th>
<th>Reuse Training Data Across Policies</th>
<th>Underlying MDP Solution Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSPI</td>
<td>linear</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Policy Iteration</td>
</tr>
<tr>
<td>Fitted Q-Iteration</td>
<td>Arbitrary</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Value Iteration</td>
</tr>
</tbody>
</table>
Final Thoughts

• By using Q-functions, batch methods can do policy improvement with model

• Does not explicitly address data collection (exploration) – both a pro and a con

• In practice, performance depends heavily upon:
  – Choice of features (both LSPI and fitted Q)
  – Choice of approximation architecture (fitted Q)
  – Distribution of training samples – how exploration is taken into account