Batch RL Via
Least Squares Policy Iteration

Michail G. Lagoudakis and Ronald Parr

Alan Fern

* Based on slides by Ronald Parr
Overview

• Motivation

• LSPI
  ▲ Derivation from LSTD
  ▲ Experimental results
Online versus Batch RL

• **Online RL**: integrates data collection and optimization
  ▲ Select actions in environment and at the same time update parameters based on each observed \((s, a, s', r)\)

• **Batch RL**: decouples data collection and optimization
  ▲ First generate experience in the environment giving a data set of state-action-reward-state pairs \(\{(s_i, a_i, r_i, s'_i)\}\)
  ▲ Use the fixed set of experience 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
Batch RL Motivation

• There are many applications that naturally fit the batch RL model

• **Medical Treatment Optimization:**
  
  \(\text{Input:}\) collection of treatment episodes for an ailment giving sequence of observations and actions including outcomes
  
  \(\text{Output:}\) a treatment policy, ideally better than current practice

• **Emergency Response Optimization:**
  
  \(\text{Input:}\) collection of emergency response episodes giving movement of emergency resources before, during, and after 911 calls
  
  \(\text{Output:}\) emergency response policy
LSPI

• LSPI is a model-free batch RL algorithm
  ▲ Learns a linear approximation of Q-function
  ▲ stable and efficient
  ▲ Never diverges or gives meaningless answers

• LSPI can be applied to a dataset regardless of how it was collected
Terminology

- $S$: state space, $s$: individual states
- $R(s,a)$: reward for taking action $a$ in state $s$
- $\gamma$: discount factor
- $V$: state value
- $P(s' | s,a) = T(s,a,s')$: transition function
- $Q$: state-action value
- Policy: $\pi(s) \rightarrow a$

Objective: Maximize expected, discounted return

$$E \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right]$$
Projection Approach to Approximation

• Recall the standard Bellman equation:

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

or equivalently \( V^* = T[V^*] \) where \( T[.] \) is the Bellman operator

• Recall from value iteration, the sub-optimality of a value function can be bounded in terms of the Bellman error:

\[ \| V - T[V] \|_\infty \]

• This motivates trying to find an approximate value function with small Bellman error
Projection Approach to Approximation

• Suppose that we have a space of representable value functions
  ▲ E.g. the space of linear functions over given features

• Let $\Pi$ be a projection operator for that space
  ▲ Projects any value function (in or outside of the space) to “closest” value function in the space

• “Fixed Point” Bellman Equation with approximation
  $$\hat{V}^* = \Pi(T[\hat{V}^*])$$
  ▲ Depending on space this will have a small Bellman error

• LSPI will attempt to arrive at such a value function
  ▲ Assumes linear approximation and least-squares projection
**Projected Value Iteration**

- **Naïve Idea**: try computing projected fixed point using VI
- **Exact VI**: (iterate Bellman backups)
  \[ V^{i+1} = T[V^i] \]

- **Projected VI**: (iterated projected Bellman backups):
  \[ \hat{V}^{i+1} = \prod \left( T[\hat{V}^i] \right) \]

Projects exact Bellman backup to closest function in our restricted function space

exact Bellman backup (produced value function)
Example: Projected Bellman Backup

Restrict space to linear functions over a single feature $\phi$:

$$\hat{V}(s) = w \cdot \phi(s)$$

Suppose just two states $s_1$ and $s_2$ with: $\phi(s_1)=1$, $\phi(s_2)=2$

Suppose exact back of $V^i$ gives:

$$T[\hat{V}^i](s_1) = 2, \quad T[\hat{V}^i](s_2) = 2$$

Can we represent this exact backup in our linear space?
Example: Projected Bellman Backup

Restrict space to linear functions over a single feature $\phi$: 
\[ \hat{V}(s) = w \cdot \phi(s) \]

Suppose just two states $s_1$ and $s_2$ with: $\phi(s_1)=1$, $\phi(s_2)=2$

Suppose exact back of $V^i$ gives:
\[ T[\hat{V}^i](s_1) = 2, \; T[\hat{V}^i](s_2) = 2 \]

The backup can’t be represented via our linear function:
\[ \hat{V}^{i+1} = \prod(T[\hat{V}^i]) \]

\[ \hat{V}^{i+1}(s) = 1.333 \cdot \phi(s) \]

Projected backup is just least-squares fit to exact backup.
Problem: Stability

- Exact value iteration stability ensured by contraction property of Bellman backups:

\[ V^{i+1} = T[V^i] \]

- Is the “projected” Bellman backup a contraction:

\[ \hat{V}^{i+1} = \prod(T[\hat{V}^i]) \]
Example: Stability Problem [Bertsekas & Tsitsiklis 1996]

**Problem:** Most projections lead to backups that are not contractions and unstable

Rewards all zero, $\gamma = 0.9$: $V^* = 0$

Consider linear approx. w/ single feature $\phi$ with weight $w$.

$$\hat{V}(s) = w \cdot \phi(s)$$

Optimal $w = 0$
since $V^*=0$
Example: Stability Problem

From $V^i$ perform projected backup for each state

$T[\hat{V}^i](s_1) = \gamma \hat{V}^i(s_2) = 1.8w^i$

$T[\hat{V}^i](s_2) = \gamma \hat{V}^i(s_2) = 1.8w^i$

Can’t be represented in our space so find $w^{i+1}$ that gives least-squares approx. to exact backup

After some math we can get: $w^{i+1} = 1.2w^i$

What does this mean?
Example: Stability Problem

Each iteration of Bellman backup makes approximation worse!
Even for this simple problem “projected” VI diverges.
Understanding the Problem

• What went wrong?
  ▲ Exact Bellman backups reduces error in maximum norm
  ▲ Least squares (= projection) non-expansive in $L_2$ norm
  ▲ May increase maximum norm distance

• **Conclusion**: Alternating value iteration and function approximation is risky business
Overview

- Motivation

- LSPI
  - Derivation from Least-Squares Temporal Difference Learning
  - Experimental results
How does LSPI fix these?

• LSPI performs approximate policy iteration
  ▲ PI involves policy evaluation and policy improvement
  ▲ Uses a variant of least-squares temporal difference learning (LSTD) for **approx. policy evaluation** [Bratdke & Barto ‘96]

• Stability:
  ▲ LSTD directly solves for the fixed point of the approximate Bellman equation for policy values
  ▲ With singular-value decomposition (SVD), this is always well defined

• Data efficiency
  ▲ LSTD finds best approximation 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
- Assumes linear value function approximation of K features
  \[ \hat{V}(s) = \sum_k w_k \phi_k(s) \]
- The \( \phi_k \) are arbitrary feature functions of states
- Some vector notation

\[
\begin{align*}
\hat{V} &= \begin{bmatrix}
\hat{V}(s_1) \\
\vdots \\
\hat{V}(s_n)
\end{bmatrix} \\
w &= \begin{bmatrix}
w_1 \\
\vdots \\
w_k
\end{bmatrix} \\
\phi_k &= \begin{bmatrix}
\phi_k(s_1) \\
\vdots \\
\phi_k(s_n)
\end{bmatrix} \\
\Phi &= \begin{bmatrix}
\phi_1 & \cdots & \phi_K
\end{bmatrix}
\end{align*}
\]
Deriving LSTD

\[ \hat{V} = \Phi \mathbf{w} \]
assigns a value to every state

\[ \Phi = \begin{bmatrix} \phi_1(s1) & \phi_2(s1) \\ \phi_1(s2) & \phi_2(s2) \\ \vdots & \vdots \\ \phi_1(sN) & \phi_2(sN) \end{bmatrix} \]
K basis functions

\[ \hat{V} \] is a linear function in the column space of \( \phi_1 \ldots \phi_k \), that is,

\[ \hat{V} = w_1 \cdot \phi_1 + \cdots + w_K \cdot \phi_K \]
Suppose we know value of policy

- Want: \( \Phi \mathcal{W} \approx \mathcal{V}^{\pi} \)

- Least squares weights minimizes squared error
  \[
  \mathcal{W} = \left( \Phi^T \Phi \right)^{-1} \Phi^T \mathcal{V}^{\pi}
  \]
  Sometimes called pseudoinverse

- Least squares projection is then
  \[
  \hat{\mathcal{V}} = \Phi \mathcal{W} = \Phi \left( \Phi^T \Phi \right)^{-1} \Phi^T \mathcal{V}^{\pi}
  \]
  Textbook least squares projection operator
But we don’t know $V$...

- Recall fixed-point equation for policies
  \[ V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} P(s' | s, \pi(s)) V^\pi(s') \]

- Will solve a projected fixed-point equation:
  \[ \hat{V}^\pi = \prod \left( R + \gamma P \hat{V}^\pi \right) \]

\[ R = \begin{bmatrix}
R(s_1, \pi(s_1)) \\
\vdots \\
R(s_n, \pi(s_n))
\end{bmatrix}, \quad P = \begin{bmatrix}
P(s_1 | s_1, \pi(s_1)) & \cdots & P(s_n | s_1, \pi(s_1)) \\
\vdots & \ddots & \vdots \\
P(s_1 | s_n, \pi(s_n)) & \cdots & P(s_1 | s_n, \pi(s_n))
\end{bmatrix} \]

- Substituting least squares projection into this gives:
  \[ \Phi w = \Phi (\Phi^T \Phi)^{-1} \Phi^T \left( R + \gamma P \Phi w \right) \]

- Solving for $w$:
  \[ w = (\Phi^T \Phi - \gamma \Phi^T P \Phi)^{-1} \Phi^T R \]
Almost there...

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

• Matrix to invert is only $K \times K$

• But...
  
  ▲ Expensive to construct matrix (e.g. $P$ is $|S| \times |S|$)
  
  ▲ We don’t know $P$
  
  ▲ We don’t know $R$
Using Samples for $\Phi$

Suppose we have state transition samples of the policy running in the MDP: $\{(s_i, a_i, r_i, s_i')\}$

Idea: Replace enumeration of states with sampled states

$$\hat{\Phi} = \begin{bmatrix} \phi_1(s_1) & \phi_2(s_1) \\ \phi_1(s_2) & \phi_2(s_2) \\ \vdots & \vdots \\ \vdots & \vdots \\ \phi_1(s_i) & \phi_2(s_i) \\ \phi_1(s_i') & \phi_2(s_i') \end{bmatrix}$$
Using Samples for R

Suppose we have state transition samples of the policy running in the MDP: \( \{(s_i, a_i, r_i, s'_i)\} \)

Idea: Replace enumeration of reward with sampled rewards
Using Samples for $\mathbb{P}\Phi$

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

\[ P\Phi \approx \{ \phi_1(s_1'), \phi_2(s_1'), \ldots, \phi_1(s_2'), \phi_2(s_2'), \ldots \} \]

\[ \text{s' from (s, a, r, s')} \]
Putting it Together

• LSTD needs to compute:

\[ w = (\Phi^T \Phi - \gamma \Phi^T \Phi \Phi)^{-1} \Phi^T R = B^{-1} b \]

\[ B = \Phi^T \Phi - \gamma \Phi^T (P \Phi) \]

\[ b = \Phi^T R \]

from previous slide

• The hard part of which is \( B \) the kxk matrix:

• Both \( B \) and \( b \) can be computed incrementally for each \((s,a,r,s')\) sample: (initialize to zero)

\[ B_{ij} \leftarrow B_{ij} + \phi_i(s)\phi_j(s) - \gamma \phi_i(s)\phi_j(s') \]

\[ b_i \leftarrow b_i + r \cdot \phi_i(s) \]
LSTD Algorithm

- Collect data by executing trajectories of current policy
- For each \((s,a,r,s')\) sample:
  \[
  B_{ij} \leftarrow B_{ij} + \phi_i(s)\phi_j(s) - \gamma \phi_i(s)\phi_j(s') \\
  b_i \leftarrow b_i + r \cdot \phi_i(s, a) \\
  w \leftarrow B^{-1}b
  \]
LSTD Summary

• Does $O(k^2)$ work per datum
  ▲ Linear in amount of data.

• Approaches model-based answer in limit

• Finding fixed point requires inverting matrix

• Fixed point almost always exists

• Stable; efficient
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

$$\pi(s) = \text{greedy}(\hat{V}(s, \mathbf{w}))$$  // policy improvement

Evaluate $\pi$ using LSTD

- Generate sample trajectories of $\pi$
- 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
- Not truly a batch method
  - Data must be collected from current policy for LSTD
LSPI

- LSPI is similar to previous loop but replaces LSTD with a new algorithm LSTDQ

- LSTD: produces a value function
  - Requires sample from policy under consideration

- LSTDQ: produces a Q-function
  - 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!
  - Truly a batch method.
Implementing LSTDQ

- Both LSTD and LSTDQ compute: $B = \Phi^T \Phi - \lambda \Phi^T (P \Phi)$
- But LSTDQ basis functions are indexed by actions

$$\hat{Q}_w(s, a) = \sum_k w_k \cdot \phi_k(s, a)$$
defines greedy policy: $\pi_w(s) = \arg \max_a \hat{Q}_w(s, a)$

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

$$B_{ij} \leftarrow B_{ij} + \phi_i(s, a)\phi_j(s, a) - \lambda \phi_i(s, a)\phi_j(s', \pi_w(s'))$$

$$b_i \leftarrow b_i + r \cdot \phi_i(s, a)$$

$$w \leftarrow B^{-1}b$$

$\arg \max_a \hat{Q}_w(s', a)$
Running LSPI

• There is a Matlab implementation available!

1. Collect a database of \((s,a,r,s')\) experiences (this is the magic step)

2. Start w/random weights (= 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
Results: Bicycle Riding

- Watch random controller operate bike
- Collect ~40,000 \((s,a,r,s')\) samples
- Pick 20 simple basis functions \((\times5\text{ actions})\)
- Make 5-10 passes over data (PI steps)
- Reward was based on distance to goal + goal achievement

- Result:
  Controller that balances and rides to goal
Bicycle Trajectories
What about Q-learning?

- Ran Q-learning with same features
- Used experience replay for data efficiency
Q-learning Results
LSPI Robustness

![Graph showing the average number of balancing steps against the number of training episodes. The x-axis represents the number of training episodes ranging from 0 to 5000, while the y-axis represents steps ranging from 0 to 8 x 10^4. The graph includes error bars indicating variability.](image-url)
Some key points

• LSPI is a batch RL algorithm
  ▲ Can generate trajectory data anyway you want
  ▲ Induces a policy based on global optimization over full dataset

• Very stable with no parameters that need tweaking
So, what’s the bad news?

- LSPI does not address the exploration problem
  - It decouples data collection from policy optimization
  - This is often not a major issue, but can be in some cases

- $k^2$ can sometimes be big
  - Lots of storage
  - Matrix inversion can be expensive

- Bicycle needed “shaping” rewards

- Still haven’t solved
  - Feature selection (issue for all machine learning, but RL seems even more sensitive)