Learning Rules

… where do they come from?
Recall: Bellman

\[ V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} T(s'|s, \pi(s)) V^\pi(s') \]

We know that Bellman must hold.
Converse: if Bellman holds we’re good.
Objective Function

So let’s form an objective function:

- *Minimize summed squared Bellman error.*

\[
\min \sum_s \left( V^\pi(s) - R(s, \pi(s)) - \gamma \sum_{s'} T(s'|s, \pi(s)) V^\pi(s') \right)^2
\]

This assumes known \( T, R \) - full backups.
Objective Function

Sample-based version:

$$\min \sum_{i=1}^{n} \left( V^\pi(s_i) - r_i - \gamma V^\pi(s'_i) \right)^2$$

... assuming n samples of form \((s_i, r_i, s'_i)\)
Objective Function Minimization

Classical approach:

*Stochastic gradient descent*

As each data point comes in

- compute gradient of objective w.r.t. data point
- descend gradient a little bit
- this reduces the objective (ascend to increase)
Value Function Approximation

We need to pick a parametrized form for $V$.

Linear VFA:

$$V(s) = w \cdot \phi(s) = w^T \phi(s)$$

Objective function:

$$\min \sum_{i=1}^{n} \left( w^T \phi(s) - r_i - \gamma w^T \phi(s_i') \right)^2$$
Gradient

For each weight \( w_i \):

\[
\frac{\partial}{\partial w_i} \sum_{i=1}^{n} \left( w^T \phi(s) - r_i - \gamma w^T \phi(s'_i) \right)^2
\]

\[
= 2 \sum_{i=1}^{n} \left( w^T \phi(s) - r_i - \gamma w^T \phi(s'_i) \right) \phi^T(s)
\]

so for each \( s_i \) the contribution is:

\[
= \left( w^T \phi(s) - r_i - \gamma w^T \phi(s'_i) \right) \phi^T(s)
\]

therefore, make a step:

\[
w_{t+1} = w_t + \alpha \left( w^T_t \phi(s) - r_i - \gamma w^T_t \phi(s'_i) \right) \phi^T(s)
\]
Gradient

For each weight $w_i$:

$$\frac{\partial}{\partial w_i} \sum_{i=1}^{n} \left( w^T \phi(s) - r_i - \gamma w^T \phi(s'_i) \right)^2$$

$$= 2 \sum_{i=1}^{n} \left( w^T \phi(s) - r_i - \gamma w^T \phi(s'_i) \right) \phi^T(s)$$

so for each $s_i$ the contribution is:

$$= \left( w^T \phi(s) - r_i - \gamma w^T \phi(s'_i) \right) \phi^T(s)$$

therefore, make a step:

$$w_{t+1} = w_t + \alpha \left( w_t^T \phi(s) - r_i - \gamma w_t^T \phi(s'_i) \right) \phi^T(s)$$

TD error
Therefore

TD learning with LVFA:

\[ w_{t+1} = w_t + \alpha \left( w_t^T \phi(s) - r_i - \gamma w_t^T \phi(s'_i) \right) \phi^T(s) \]

Remember that this works for discrete VFA as well.

- Indicator functions.
But Wait!

Why stochastic gradient descent?
  • Linear
  • No “memory”
  • Fast
  • Easily adapt to changing policy

What if we just kept all the data and minimized the Bellman error?
Least-Squares TD

Minimize:

$$\min \sum_{i=1}^{n} \left( w^T \phi(s) - r_i - \gamma w^T \phi(s'_i) \right)^2$$

Error function has a bowl shape, so unique minimum. Just go right there!
Derivative set to zero:

\[
\sum_{i=1}^{n} (w^T \phi(s) - r_i - \gamma w^T \phi(s'_i)) \phi^T(s) = 0
\]

\[
w^T \sum_{i=1}^{n} (\phi(s_i) - \phi(s'_i)) \phi^T(s_i) = \sum_{i=1}^{n} r_i \phi^T(s_i)
\]

\[w^T A = b\]

\[w = A^{-1} b^T\]

\[A = \sum_{i=1}^{n} (\phi(s_i) = \phi(s'_i)) \phi(s_i)^T\]

\[b = \sum_{i=1}^{n} r_i \phi(s_i)^T\]
LSTD

A and $b$ are summary statistics - they store all you need to know to obtain $w$.

$$A = \sum_{i=1}^{n} (\phi(s_i) = \phi(s'_i))\phi(s_i)^T$$

$$b = \sum_{i=1}^{n} r_i \phi(s_i)^T$$

At each timestep, update $A$ and $b$.

Complexity:

- $O(k^2)$ for $A$
- $O(k)$ for $b$
- $O(k^3\text{-ish})$ for inverting $A$. 
Can derive the least-squares version of $\text{LSTD}(\lambda)$ in this way. Try it at home!

- Write down the objective function …
- Sample $r_i$ replaced by complex reward estimate.
- The rest is the same.
- You will get a trace vector if you do some clever algebra.
- Trace vector is the same size as $w$. 

**Note!**
LSTD

One inversion solves for $w$!

But:

• Computationally expensive.

• $A$ may not be invert-able.

• Least-squares behavior sometimes unstable outside of data.
Policy Iteration

That takes care of learning $V/Q$, but how do we do policy iteration?

Easy!
- Estimate $V/Q$ with LSTD
- Update policy to be greedy

But:
- LSTD does not forget: uses all data equally.
- Data is from old policy.
- Need off-policy learning.
LSPI

Least-Squares Policy Iteration

Objective function:

$$\min \sum_{i=1}^{n} \left( w^T \phi(s_i, a_i) - r_i - \gamma w^T \max_{a'} \phi(s'_i, a') \right)^2$$

Equation:

$$w = A^{-1} b^T$$

$$A = \sum_{i=1}^{n} \left( \phi(s_i, a_i) - \max_{a'} \phi(s'_i, a') \right) \phi(s_i, a_i)^T$$

$$b = \sum_{i=1}^{n} r_i \phi(s_i)^T$$  (as before)
LSPI

Advantages:
- Fast learning
- No learning rate

Disadvantages:
- Must *recompute* $A$ every new policy.
- Time to compute: $O(nk^2 + k^3)$
- Large matrix inversions: numerical stability issues.
- Batch …

$$A = \sum_{i=1}^{n} \left( \phi(s_i, a_i) - \max_{a'} \phi(s'_i, a') \right) \phi(s_i, a_i)^T$$
Fitted Q Iteration:
Essentially LSPI but using any function approximator.
  • Get a batch of data.
  • Repeat:
    • Learn Q by minimizing Bellman error.
    • Get new policy.
    • Get more data.

Very many state-of-the-art RL algorithms are LSPI/FQI-style.