Review: Bayesian learning

- Bayesian learning: \( P(\theta \mid D) = \frac{P(D \mid \theta) P(\theta)}{Z} \)
  - \( P(\theta) \): prior over parametric model class
  - \( P(D \mid \theta) \): likelihood
- or, \( P(\theta \mid X, Y) = \frac{P(Y \mid \theta, X) P(\theta)}{Z} \) as long as \( X \perp \theta \)
- Predictive distribution
Review: Bayesian learning

- Exact Bayes w/ conjugate prior, or numerical integration—this example: logistic regression
- Or, MLE/MAP
Review: MDPs

- Sequential decision problem under uncertainty
- States, actions, costs, transitions, discounting
- Policy, execution trace
- State-value ($J$) and action-value ($Q$) function
  - $(1-\gamma) \times$ immediate cost + $\gamma \times$ future cost
Review: MDPs

- Tree search
- Receding horizon tree search w/ heuristic
- Dynamic programming (value iteration)
- Pruning (once we realize a branch is bad, or subsampling scenarios)
- Curse of dimensionality
Alternate algorithms for “small” systems—policy evaluation

\[ Q^\pi(s, a) = (1 - \gamma)C(s, a) + \gamma \mathbb{E}[J^\pi(s') | s' \sim T(\cdot | s, a)] \]
\[ J^\pi(s) = \mathbb{E}[Q^\pi(s, a) | a \sim \pi(\cdot | s)] \]

- Linear equations: so, Gaussian elimination, biconjugate gradient, Gauss-Seidel iteration, …
  - VI is essentially the Jacobi iterative method for matrix inversion
- Stochastic-gradient-descent-like
  - TD(\lambda), Q-learning
Alternate algorithms for “small” systems—policy optimization

- Policy iteration: alternately
  - use any above method to evaluate current $\pi$
  - replace $\pi$ with **greedy** policy: at each state $s$, $\pi(s) := \arg\max_a Q(s,a)$

- Actor-critic: like policy iteration, but **interleave** solving for $J^\pi$ and updating $\pi$
  - e.g., run biconjugate gradient for a few steps
  - warm start: each $J^\pi$ probably similar to next

- SARSA = AC w/ TD($\lambda$) critic, $\epsilon$-greedy policy
Alternate algorithms for “small” systems—policy optimization

- (Stochastic) policy gradient
  - pick a parameterized policy class \( \pi_\theta(a \mid s) \)
  - compute or estimate \( g = \nabla_\theta J_\pi(s_1) \)
  - \( \theta \leftarrow \theta - \eta g \), repeat

- More detail:
  - can estimate \( g \) quickly by simulating a few trajectories
  - can also use *natural* gradient to get faster convergence
Alternate algorithms for “small” systems—policy optimization

- Linear programming
  - analogy: use an LP to compute $\min(3, 6, 5)$
  - note min v. max

$$\begin{align*}
\text{max } J & \quad \text{s.t.} \\
J & \leq 3 \\
J & \leq 6 \\
J & \leq 5
\end{align*}$$
Linear programming

\[
\begin{align*}
\text{max } & \quad J(s_1) \quad \text{s.t.} \\
Q(s, a) & = (1 - \gamma)C(s, a) + \gamma \mathbb{E}[J(s')] \mid s' \sim T(\cdot \mid s, a)] \\
J(s) & \leq Q(s, a) \quad \forall s, a
\end{align*}
\]

- Variables \(J(s)\) and \(Q(s, a)\) for all \(s, a\)
- Note: dual of this LP is interesting
  - generalizes single-source shortest paths
Model requirements

- What we have to know about the MDP in order to plan?
  - full model
  - simulation model
  - no model: only the real world
Model requirements

- VI and LP require full model
- PI and actor-critic inherit requirements of policy-evaluation subroutine
- TD(\(\lambda\)), SARSA, policy gradient: OK with simulation model or no model
  - horribly data-inefficient if used directly on real world with no model—don’t do this!
  - note: model can be just \{ all of my data \}
A word on performance measurements

- Multiple criteria we might care about:
  - data (from real world)
  - runtime
  - calls to model (under some API)

- Measure convergence rate of:
  - $J(s)$ or $Q(s, a)$
  - $\pi(s)$
  - actual (expected total discounted) cost
Building a model

- How to handle lack of model without horrible data inefficiency? Build one!
  - hard inference problem; getting it wrong is bad
  - this is why \{ all of my data \} is a popular model

- What do we do with posterior over models?
  - just use MAP model ("certainty equivalent")
  - compute posterior over $\pi^*$: slow, still wrong
  - even slower: $\max_{\pi} \mathbb{E}(J^{\pi}(s) \mid \text{data, model class})$
    - except policy gradient (Ng’s helicopter)
Algorithms for large systems

- Policy gradient: no change
- Any value-based method: can’t even write down $J(s)$ or $Q(s,a)$
- So,

$$J(s) = \sum_i w_i \phi_i(s)$$

$$Q(s, a) = \sum_i w_i \phi_i(s, a)$$
Algorithms for large systems

- **Evaluation:** TD(\(\lambda\)), LSTD

- **Optimization:**
  - policy iteration or actor-critic
    - e.g., LSTD \(\rightarrow\) LSPI
  - approximate LP
  - value iteration: only special cases, e.g., finite-element grid
Least-squares temporal differences (LSTD)

\[ Q^\pi(s, a) = (1 - \gamma)C(s, a) + \gamma \mathbb{E}[J^\pi(s') \mid s' \sim T(\cdot \mid s, a)] \]
\[ J^\pi(s) = \mathbb{E}[Q^\pi(s, a) \mid a \sim \pi(\cdot \mid s)] \]

- Data: \( T = (s_1, a_1, c_1, s_2, a_2, c_2, \ldots) \sim \pi \)
- Want \( Q(s_t, a_t) \approx (1 - \gamma)c_t + \gamma Q(s_{t+1}, a_{t+1}) \)
  - \( w^T\Phi(s_t, a_t) \approx (1 - \gamma)c_t + \gamma w^T\Phi(s_{t+1}, a_{t+1}) \)
  - \( \Phi = \text{vector of } k \text{ features}, \ w = \text{weight vector} \)
LSTD

- \( w^T \Phi(s_t, a_t) \approx (1-\gamma)c_t + \gamma w^T \Phi(s_{t+1}, a_{t+1}) \)

- Vector notation:
  - \( Fw \approx (1-\gamma)c_t + \gamma F_1 w \)

- Overconstrained: multiply both sides by \( F \)
  - \( F^TFw = (1-\gamma)F^Tc_t + \gamma F^TF_1 w \)
LSTD: example

- 100 states in a line; move left or right at cost 1 per state; goals at both ends; discount 0.99
LSTD: example

- 100 states in a line; move left or right at cost 1 per state; goals at both ends; discount 0.99
LSPI
LSPI

\[ \phi_i(s) \]

\[ J(s) \]
LSPI
LSPI

\[ \phi_i(s) \]

\[ J(s) \]
LSPI

\[ \phi_i(s) \]

\[ J(s) \]