fitted Value function methods & fitted Dynamic Programming

Lucas Janson and Sham Kakade

CS/Stat 184: Introduction to Reinforcement Learning

Fall 2022

Today

- Recap + Overview of PG

- Today:
  1. Fitted Policy Evaluation
  2. Fitted Dynamic Programming Methods
     1. Fitted Policy Evaluation
     2. Fitted Q-Value Iteration

Recap + Overview of PG

Recap: Policy Parameterization

Recall that we consider parameterized policy $\pi_\theta(\cdot \mid s) \in \Delta(A), \forall s$

1. Softmax linear Policy

Feature vector $\phi(s, a) \in \mathbb{R}^d$, and parameter $\theta \in \mathbb{R}^d$

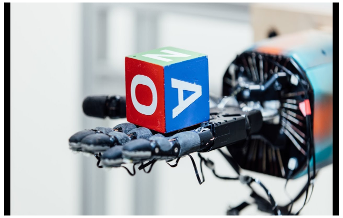
$$\pi_\theta(a \mid s) = \frac{\exp(\theta^\top \phi(s, a))}{\sum_{a'} \exp(\theta^\top \phi(s, a'))}$$

2. Neural Policy:

Neural network $f_\theta : S \times A \mapsto \mathbb{R}$

$$\pi_\theta(a \mid s) = \frac{\exp(f_\theta(s, a))}{\sum_{a'} \exp(f_\theta(s, a'))}$$

Policy Parameterization Example for “Controls”



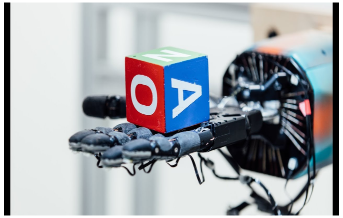
Suppose $a \in R^k$, as it might be for a control problem.

2. Neural Policy:

Neural network
$f_\theta : S \times A \mapsto \mathbb{R}$

$$\pi_\theta(a \mid s) = \frac{\exp(f_\theta(s,a))}{\sum_{a'} \exp(f_\theta(s,a'))}$$

Policy Parameterization Example for “Controls”



Suppose $a \in R^k$, as it might be for a control problem.

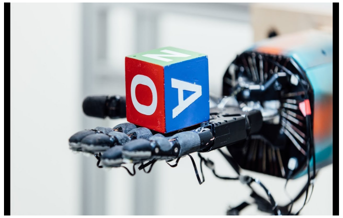
2. Neural Policy:

Neural network
$f_\theta : S \times A \mapsto \mathbb{R}$

$$\pi_\theta(a \mid s) = \frac{\exp(f_\theta(s, a))}{\sum_{a'} \exp(f_\theta(s, a'))}$$

3. Example: Neural policy for continuous action case

Policy Parameterization Example for "Controls"



Suppose $a \in R^k$, as it might be for a control problem.

2. Neural Policy:

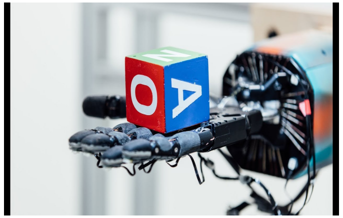
Neural network
$f_\theta : S \times A \mapsto \mathbb{R}$

$$\pi_\theta(a \mid s) = \frac{\exp(f_\theta(s, a))}{\sum_{a'} \exp(f_\theta(s, a'))}$$

3. Example: Neural policy for continuous action case

- Neural network $g_\theta : S \mapsto \mathbb{R}^k$
- Parameters: $\theta \in R^d, \sigma \in R^+$

Policy Parameterization Example for "Controls"



Suppose $a \in R^k$, as it might be for a control problem.

2. Neural Policy:

Neural network
$f_\theta : S \times A \mapsto \mathbb{R}$

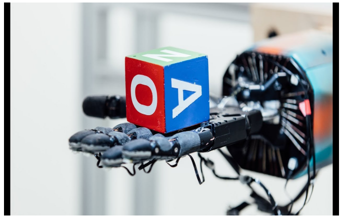
$$\pi_\theta(a \mid s) = \frac{\exp(f_\theta(s, a))}{\sum_{a'} \exp(f_\theta(s, a'))}$$

3. Example: Neural policy for continuous action case

- Neural network $g_\theta : S \mapsto \mathbb{R}^k$
- Parameters: $\theta \in R^d$, $\sigma \in R^+$
- Policy: sample action from a (multivariate) Normal with mean $g_\theta(s)$ and variance $\sigma^2 I$, i.e.

$$\pi_{\theta,\sigma}(a \mid s) = \mathcal{N}(g_\theta(s), \sigma^2 I)$$

Policy Parameterization Example for "Controls"



Suppose $a \in R^k$, as it might be for a control problem.

2. Neural Policy:

Neural network

$f_\theta : S \times A \mapsto \mathbb{R}$

$$\pi_\theta(a \mid s) = \frac{\exp(f_\theta(s,a))}{\sum_{a'} \exp(f_\theta(s,a'))}$$

3. Example: Neural policy for continuous action case

- Neural network $g_\theta : S \mapsto \mathbb{R}^k$
- Parameters: $\theta \in R^d$, $\sigma \in R^+$
- Policy: sample action from a (multivariate) Normal with mean $g_\theta(s)$ and variance $\sigma^2 I$, i.e.

$$\pi_{\theta,\sigma}(a \mid s) = \mathcal{N}(g_\theta(s), \sigma^2 I)$$

- Implicitly, this is the same functional form as 2:

$$f_{\theta,\sigma}(s,a) = \frac{\|a - g_\theta(s)\|^2}{2\sigma^2 k}$$

The Advantage Function (finite horizon)

The Advantage Function (finite horizon)

- The Advantage function is defined as:

$$A_h^\pi(s, a) = Q_h^\pi(s, a) - V_h^\pi(s)$$

The Advantage Function (finite horizon)

- The Advantage function is defined as:

$$A_h^\pi(s,a) = Q_h^\pi(s,a) - V_h^\pi(s)$$

- We have that:

$$E_{a\sim\pi(\cdot|s)}\left[A_h^\pi(s,a)\,\middle|\,s,h\right] = \sum_a \pi(a\,|\,s)A_h^\pi(s,a) = \;0$$

The PG: baseline and advantage versions

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta(\tau)} \left[\sum_{h=0}^{H-1} \nabla_\theta \ln \pi_\theta(a_h | s_h) \left(\left(\sum_{t=h}^{H-1} r_t \right) - b_h(s_h) \right) \right]$$

$$= \mathbb{E}_{\tau \sim \rho_\theta(\tau)} \left[\sum_{h=0}^{H-1} \nabla_\theta \ln \pi_\theta(a_h | s_h) \left(Q_h^{\pi_\theta}(s_h, a_h) - b_h(s_h) \right) \right]$$

$$= \mathbb{E}_{\tau \sim \rho_\theta(\tau)} \left[\sum_{h=0}^{H-1} \nabla_\theta \ln \pi_\theta(a_h | s_h) A_h^{\pi_\theta}(s_h, a_h) \right]$$

nicest for sampling

- The second step follows by choosing $b_h(s) = V_h^\pi(s)$.
- The most common approach is to use $b_h(s)$ to approximate $V_h^\pi(s)$.
- The REINFORCE version is not used in practice.

PG for the (softmax) linear policies

- We can simplify this to:

$$\nabla J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta(\tau)} \left[\sum_{h=0}^{H-1} A_h^{\pi_\theta}(s_h, a_h) \phi(s_h, a_h) \right]$$

“Review”

- For a random variable $y \in R$, what is:

$$\arg\min_c E_{y \sim D}[(c-y)^2] = ??$$

- Now let us look at the “function” case where we have a distribution over (x, y) pairs

$$f^\star = \arg\min_{f \in \mathscr{F}} E_{(x,y) \sim D}[(f(x)-y)^2]$$

(where $\mathscr{F}$ is the class of all possible functions)

What is $f^\star(x) = ??$ $f(x) \approx E\{y/x\}$

Recap + Overview of PG

A PG procedure:

(this is sometimes referred to as actor-critic approach)

1. Initialize θ_0, samples sizes M,N, parameters: $\eta_1, \eta_2, \ldots$
2. For t = 0, … :
   1. [Policy Eval Subroutine]

      Using N sampled trajectories, $\tau_1, \ldots \tau_N \sim \rho_{\theta_t}$, try to learn a $\widetilde{b}$ s.t.

      $\widetilde{b}(s) \approx V_h^{\pi_{\theta_t}}(s)$
   2. [Mini-Batch PG Update]

      Init $g = 0$ and do M times:

      Obtain a trajectory $\tau \sim \rho_{\theta_t}$

      Set $g = g + \sum_{h=0}^{H-1} \nabla \ln \pi_{\theta_t}(a_h | s_h)\big(R_h(\tau) - \tilde{b}(s_h)\big)$

      Set $\widetilde{\nabla}_\theta J(\theta_t) := \frac{1}{M} g$

   3. Update: $\theta_{t+1} = \theta_t + \eta_t \widetilde{\nabla}_\theta J(\theta_t)$

$R_h(\tau) = \sum_{t=h}^{H-1} r_t$

Baseline/Value Function Parameterizations

Now let us consider parameterized classes of functions $\mathscr{F}$, where for each $f \in \mathscr{F}$, $f : S \to R$

1. Linear Functions

Feature vector $\psi(s) \in \mathbb{R}^k$, and parameter $w \in \mathbb{R}^k$

$$f_w(s) = w^\top \psi(s)$$

2. Neural Policy:

Neural network $f_w : S \mapsto \mathbb{R}$

Let's assume the current time in the episode is contained in the state. $s \leftarrow (s, h)$
(e.g. you can always add the time into the "list" that specifies the state).

Example [Policy Eval Subroutine]:
Directly fit unbiased estimates of $V^\pi(s)$

input: policy π, sample size N

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_\pi$

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1}, \})$
2. Construct an *empirical loss function*:

$$\widetilde{L}(w) = \frac{1}{N} \sum_{i=1}^{N} \sum_{s_h \in \tau_i} \Big(f_w(s_h) - R_h(\tau_i) \Big)^2$$

$$= \mathbb{E}_{\tau \sim \rho_\pi} \left[\sum_{h=1}^{H-1} \left(f_w(s_h) - R_h(\tau) \right)^2 \right]$$

3. (approximately) find a minimizer

$$\widetilde{w} \approx \arg\min_{w} \widetilde{L}(w)$$

   (often done with SGD)
4. Return the function $\widetilde{b} = f_{\widetilde{w}}$

Today:

Fitted Value Function & Dynamic Programming Methods

Outline:

1. Fitted Policy Evaluation
2. Fitted Dynamic Programming Methods
    1. Fitted Policy Evaluation
    2. Fitted Q-Value Iteration

Implications of appending the timestep h to the state s

Implications of appending the timestep h to the state s

- Option 1 (without appending h):
  - Try to find parameters H parameters $w^0, \ldots w^{H-1}$, with each in $w_i \in R^k$ s.t. $f_{w^h}(s) \approx V_h^\pi(s)$.
  - This is means building H models (or neural nets) so we have $H \cdot k$ parameters.

Implications of appending the timestep h to the state s

- Option 1 (without appending h):
  - Try to find parameters H parameters $w^0, \ldots w^{H-1}$, with each in $w_i \in R^k$ s.t. $f_{w^h}(s) \approx V_h^\pi(s)$.
  - This is means building H models (or neural nets) so we have $H \cdot k$ parameters.

- Option 2 (building just one model):
  - try to learn a single $w \in R^k$ s.t. $f_w(s, h) \approx V_h^\pi(s), \forall h$.
  - Let us assume: $s \leftarrow (s, h)$ (i.e. that h is implicitly contained in s)

Implications of appending the timestep h to the state s

- Option 1 (without appending h):
  - Try to find parameters H parameters $w^0, \ldots w^{H-1}$, with each in $w_i \in R^k$ s.t. $f_{w^h}(s) \approx V^\pi_h(s)$.
  - This is means building H models (or neural nets) so we have $H \cdot k$ parameters.

- Option 2 (building just one model):
  - try to learn a single $w \in R^k$ s.t. $f_w(s, h) \approx V^\pi_h(s), \forall h$.
  - Let us assume: $s \leftarrow (s, h)$ (i.e. that h is implicitly contained in s)

- We can implicitly consider this to be an infinite horizon problem,
  but one which happens to terminate in H steps (i.e. at state $(s, H-1)$ the trajectory ends).

Implications of appending the timestep h to the state s

- Option 1 (without appending h):
  - Try to find parameters H parameters $w^0, \ldots w^{H-1}$, with each in $w_i \in R^k$ s.t. $f_{w^h}(s) \approx V_h^\pi(s)$.
  - This is means building H models (or neural nets) so we have $H \cdot k$ parameters.

- Option 2 (building just one model):
  - try to learn a single $w \in R^k$ s.t. $f_w(s, h) \approx V_h^\pi(s), \forall h$.
  - Let us assume: $s \leftarrow (s, h)$ (i.e. that h is implicitly contained in s)

with $\gamma = 1$

- We can implicitly consider this to be an infinite horizon problem, but one which happens to terminate in H steps (i.e. at state $(s, H-1)$ the trajectory ends).

- It is helpful to use with the discounted algorithms work — Iterative PE, Value Iteration, Policy Iteration — because our parameter w should be effective for all h.

Is there an iterative version of Policy Evaluation?

(that is faster, but approximate?)

Is there an iterative version of Policy Evaluation?

(that is faster, but approximate?)

Algorithm (Iterative PE):

1. Initialization: $V^0 : \|V^0\|_\infty \in \left[0, \frac{1}{1-\gamma}\right]$
2. Iterate until convergence: $V^{t+1} \leftarrow R + \gamma P V^t$

Is there an iterative version of Policy Evaluation?

(that is faster, but approximate?)

Algorithm (Iterative PE):

1. Initialization: $V^0 : \|V^0\|_\infty \in \left[0, \frac{1}{1-\gamma}\right]$
2. Iterate until convergence: $V^{t+1} \leftarrow R + \gamma P V^t$

   Equivalently,

$$\forall s, V^{t+1}(s) = r(s, \pi(s)) + \gamma \mathbb{E}_{s' \sim P(s,\pi(s))} V^t(s')$$

Is there an iterative version of Policy Evaluation?

(that is faster, but approximate?)

Algorithm (Iterative PE):

1. Initialization: $V^0 : \|V^0\|_\infty \in \left[0, \frac{1}{1-\gamma}\right]$
2. Iterate until convergence: $V^{\cancel{t}k+1} \leftarrow R + \gamma P V^{\cancel{t}k}$

   Equivalently,

$$\forall s, V^{\cancel{t}k+1}(s) \overset{\approx}{=} r(s,\pi(s)) + \gamma \mathbb{E}_{s' \sim P(s,\pi(s))} V^{\cancel{t}k}(s')$$

$f_{w_k} \longleftrightarrow V^k$

This is a "fixed point" algorithm trying to enforce Bellman consistency:

$$\forall s, V^\pi(s) = r(s,\pi(s)) + \gamma \mathbb{E}_{s' \sim P(s,\pi(s))} V^\pi(s')$$

$\gamma = 1$, $s \leftarrow (s,h)$

The Bellman consistency for the finite horizon case:

$$\forall s, V_h^\pi(s) = r(s,\pi(s)) + \mathbb{E}_{s \sim P(s,\pi(s))} V_{h+1}^\pi(s')$$

same consistency conditions

Another [Policy Eval Subroutine]:

Fit $V^{\pi}(s)$ using the iterative policy evaluation alg.

Another [Policy Eval Subroutine]:
Fit $V^\pi(s)$ using the iterative policy evaluation alg.

[Iterative Policy Eval Subroutine/TD]

input: policy π, sample size N

Another [Policy Eval Subroutine]:
Fit $V^{\pi}(s)$ using the iterative policy evaluation alg.

[Iterative Policy Eval Subroutine/TD]

input: policy π, sample size N

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_{\pi}$

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1},\}$)

Another [Policy Eval Subroutine]:
Fit $V^\pi(s)$ using the iterative policy evaluation alg.

[Iterative Policy Eval Subroutine/TD]

input: policy π, sample size N

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_\pi$

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1},\}$)
2. For k = $0,\ldots,K$:

Another [Policy Eval Subroutine]:
Fit $V^\pi(s)$ using the iterative policy evaluation alg.

[Iterative Policy Eval Subroutine/TD]

input: policy π, sample size N – init w_0

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_\pi$

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1},\}$)

2. For k = $0,\ldots,K$:
   1. Construct an *empirical loss function*:

$$\widetilde{L}_k(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{(s_h,r_h,s_{h+1})\in\tau_i}\Big(f_w(s_h) - \big(r_h + f_{w_k}(s_{h+1})\big)\Big)^2$$

$E\,[f_{w_k}(s')]$

$s' \sim P(s_h, a_h)$

Another [Policy Eval Subroutine]:
Fit $V^{\pi}(s)$ using the iterative policy evaluation alg.

[Iterative Policy Eval Subroutine/TD]

input: policy π, sample size N

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_{\pi}$

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1},\}$)
2. For k = $0,\ldots,K$:
   1. Construct an *empirical loss function*:

   $$\widetilde{L}_k(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{(s_h,r_h,s_{h+1})\in\tau_i}\Big(f_w(s_h) - \underbrace{\big(r_h + f_{\widetilde{w}_k}(s_{h+1})\big)}_{R_h(\tau_i)}\Big)^2$$

   2. Update:

   $$\widetilde{w}_{k+1} \approx \arg\min_w \widetilde{L}_k(w)$$

Another [Policy Eval Subroutine]:
Fit $V^\pi(s)$ using the iterative policy evaluation alg.

[Iterative Policy Eval Subroutine/TD]

input: policy π, sample size N

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_\pi$

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1},\}$)
2. For k = $0,\ldots,K$:
   1. Construct an *empirical loss function*:

   $$\widetilde{L}_k(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{(s_h,r_h,s_{h+1})\in\tau_i}\Big(f_w(s_h) - \big(r_h + f_{w_k}(s_{h+1})\big)\Big)^2$$

   2. Update:

   $$\widetilde{w}_{k+1} \approx \arg\min_w \widetilde{L}_k(w)$$
3. Return the function $\widetilde{b} = f_{\widetilde{w}}$

Another [Policy Eval Subroutine]:
Fit $V^\pi(s)$ using the iterative policy evaluation alg.

[Iterative Policy Eval Subroutine/TD]

input: policy π, sample size N, end time K

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_\pi$

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1}, \})$
2. For k = $0, \ldots, K$:
   1. Construct an *empirical loss function*:

   $$\widetilde{L}_k(w) = \frac{1}{N} \sum_{i=1}^{N} \sum_{(s_h, r_h, s_{h+1}) \in \tau_i} \Big(f_w(s_h) - (r_h + f_{w_k}(s_{h+1})) \Big)^2$$

   2. Update:

   $$\widetilde{w}_{k+1} \approx \arg\min_w \widetilde{L}_k(w)$$
3. Return the function $\widetilde{b} = f_{\widetilde{w}_K}$

Temporal Difference Learning (TD) is an online variant to do the above.

Outline:

1. Fitted Policy Evaluation
2. Fitted Dynamic Programming Methods
    1. Fitted Policy Evaluation
    2. Fitted Q-Value Iteration

This approach leads us to fitted dynamic programming…

This approach leads us to fitted dynamic programming…

Direct Policy Optimization:

- With PG, we tried to directly learn a good (parameterized) policy π_θ, for $\theta \in R^d$
  - Learning means we used only sampled trajectories (we didn't assume the MDP is known).
  - Fitted value functions were introduced for variance reduction.

This approach leads us to fitted dynamic programming…

Direct Policy Optimization:

- With PG, we tried to directly learn a good (parameterized) policy π_θ, for $\theta \in R^d$
  - Learning means we used only sampled trajectories (we didn’t assume the MDP is known).
  - Fitted value functions were introduced for variance reduction.

Fitted Dynamic Programming:

- Can we instead use a learning (fitting) approach to approximate dynamic programming?

Outline:

1. Fitted Policy Evaluation
2. Fitted Dynamic Programming Methods
    1. Fitted Policy Evaluation
    2. Fitted Q-Value Iteration

Policy Iteration (PI)

- Initialization: choose a policy $\pi^0 : S \mapsto A$
- For $t = 0, 1, \ldots$
  1. **Policy Evaluation**: compute $Q^{\pi^t}(s, a)$
  2. **Policy Improvement**: set

$$\pi^{t+1}(s) := \arg\max_a Q^{\pi^t}(s, a)$$

Fitted Policy Iteration:

Fitted Policy Iteration:

1. Initialize staring policy π_0, samples size M

Fitted Policy Iteration:

1. Initialize staring policy π_0, samples size M
2. For t = 0, … :

Fitted Policy Iteration:

1. Initialize staring policy π_0, samples size M
2. For t = 0, … :
    1. [Q-Evaluation Subroutine]

       Using N sampled trajectories, $\tau_1, \ldots \tau_N \sim \rho_{\pi_t}$, try to learn a $\widetilde{b}$

       $\widetilde{Q}^t(s,a) \approx Q_h^{\pi_t}(s,a)$

Fitted Policy Iteration:

1. Initialize staring policy π_0, samples size M
2. For t = 0, … :
   1. [Q-Evaluation Subroutine]
      Using N sampled trajectories, $\tau_1, \ldots \tau_N \sim \rho_{\pi_t}$, try to learn a $\widetilde{b}$
      $\widetilde{Q}^t(s,a) \approx Q_h^{\pi_\theta t}(s,a)$
   2. Policy Update
      $\pi_{t+1}(s) := \arg\max_a \widetilde{Q}^{\pi_t}(s,a)$

function of π

LQR

$Q_h^{\pi}(s,a) = a^\top M a$

$a^\top g_\theta(s)\, a$

Q-Function Parameterizations

(nothing new at this point)

Now, for our parameterized classes of functions $\mathscr{F}$, we have for $f \in \mathscr{F}, f : S \times A \to R$

1. Linear Functions

Feature vector $\psi(s, a) \in \mathbb{R}^k$, and parameter $w \in \mathbb{R}^k$

$$f_w(s, a) = w^\top \psi(s, a)$$

2. Neural Policy:

Neural network $f_w : S \times A \mapsto \mathbb{R}$

Example [Q-Eval Subroutine]:
Directly fit unbiased estimates of $Q^\pi(s, a)$

Example [Q-Eval Subroutine]:
Directly fit unbiased estimates of $Q^\pi(s, a)$

input: policy π, sample size N

Example [Q-Eval Subroutine]:
Directly fit unbiased estimates of $Q^\pi(s,a)$

input: policy π, sample size N

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_\pi$

Example [Q-Eval Subroutine]:

Directly fit unbiased estimates of $Q^\pi(s,a)$

input: policy π, sample size N

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_\pi$
2. Construct an *empirical loss function*:

$$\widetilde{L}(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{(s_h,a_h)\in\tau_i}\left(f_w(s_h,a_h) - R_h(\tau_i)\right)^2$$

Example **[Q-Eval Subroutine]**:
Directly fit unbiased estimates of $Q^\pi(s, a)$

$E[R_h(\tau) | s_h, a_h]$

$= Q_h^\pi(s_h, a_h)$

input: policy π, sample size N

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_\pi$
2. Construct an *empirical loss function*:
$$\widetilde{L}(w) = \frac{1}{N} \sum_{i=1}^{N} \sum_{(s_h, a_h) \in \tau_i} \left(f_w(s_h, a_h) - R_h(\tau_i) \right)^2$$
3. (approximately) find a minimizer
$$\widetilde{w} \approx \arg\min_w \widetilde{L}(w)$$
(often done with SGD)

Example [Q-Eval Subroutine]:
Directly fit unbiased estimates of $Q^\pi(s, a)$

input: policy π, sample size N

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_\pi$
2. Construct an *empirical loss function*:
$$\widetilde{L}(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{(s_h,a_h)\in\tau_i}\Big(f_w(s_h, a_h) - R_h(\tau_i)\Big)^2$$
3. (approximately) find a minimizer
$$\widetilde{w} \approx \arg\min_w \widetilde{L}(w)$$
(often done with SGD)
4. Return the function $\widetilde{b} = f_{\widetilde{w}}$

Example [Q-Eval Subroutine]:
Directly fit unbiased estimates of $Q^\pi(s,a)$

input: policy π, sample size N

1. Sample trajectories $\tau_1, \ldots \tau_N \sim \rho_\pi$
2. Construct an *empirical loss function*:
$$\widetilde{L}(w) = \frac{1}{N}\sum_{i=1}^{N} \sum_{(s_h,a_h)\in\tau_i} \Big(f_w(s_h, a_h) - R_h(\tau_i)\Big)^2$$
3. (approximately) find a minimizer
$$\widetilde{w} \approx \arg\min_w \widetilde{L}(w)$$
(often done with SGD)
4. Return the function $\widetilde{b} = f_{\widetilde{w}}$

As with the [V-Eval Subroutine], there is also an iterative (TD) approach to this.

Outline:

1. Fitted Policy Evaluation
2. Fitted Dynamic Programming Methods
    1. Fitted Policy Evaluation
    2. Fitted Q-Value Iteration

Alternative Version: Bellman Operator $\mathcal{T}$ on Q

(HW2 Q2 is the Q-version of the Bellman Equations)

- Given a function $Q : S \times A \mapsto \mathbb{R}$, define $\mathcal{T}Q : S \times A \mapsto \mathbb{R}$ as

$$\big(\mathcal{T}Q\big)(s,a) := \underbrace{r(s,a) + \gamma \mathbb{E}_{s' \sim P(s,a)} \max_{a' \in A} Q(s',a') = Q(s,a)}_{\text{B.E. for } Q.}$$

- (Bellman equations for Q)
Q is equal to $Q^\star$ if and only if $\mathcal{T}Q = Q$.

- V is eq. to V^* iff $\tilde{\mathcal{T}} V = V$

$$\tilde{\mathcal{T}} V(s) = \max_a \left[r(s,a) + \gamma \mathbb{E}_{s' \sim P(s,a)} V(s') \right]$$

Q-Value Iteration Algorithm:

$\pi_k(s) = \arg\max_a Q_k(s,a)$

1. Initialization: $Q^0 : \|Q^0\|_\infty \in \left[0, \frac{1}{1-\gamma}\right]$
2. Iterate until convergence: $Q_{k+1} \leftarrow \mathcal{T} Q_k$

i.e. $\forall s, a$

$$Q_{k+1}(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(s,a)} \max_{a' \in A} Q_k(s', a')$$

Q-Value Iteration Algorithm:

1. Initialization: $Q^0 : \|Q^0\|_\infty \in \left[0, \frac{1}{1-\gamma}\right]$
2. Iterate until convergence: $Q_{k+1} \leftarrow \mathcal{T} Q_k$

$$Q_{k+1}(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(s,a)} \max_{a' \in A} Q_k(s',a')$$

- Guarantees of Q-VI: analogous contraction properties to VI.
- What about a fitted version of this algorithm?

Fitted Q-Iteration

Fitted Q-Iteration

1. Initialize: Q_0
2. For $k = 0,1,\ldots$:
    $\mathcal{T} Q_k$
    1. Approximately try to estimate $\cancel{\mathcal{T} f_{w_k}}$ with samples

$$Q_{k+1}(s,a) \approx r(s,a) + \gamma \mathbb{E}_{s' \sim P(s,a)} \max_{a' \in A} Q_k(s',a')$$

Fitted Q-Iteration

1. Initialize: Q_0
2. For $k = 0,1,\dots$:
    1. Approximately try to estimate $\mathcal{T} f_{w_k}$ with samples

$$Q_{k+1}(s,a) \approx r(s,a) + \gamma \mathbb{E}_{s' \sim P(s,a)} \max_{a' \in A} Q_k(s',a')$$

What distribution should use to for this fitting??

The Offline Learning Setting:

We don't know the MDP and our data collection is under some fixed distribution.

The Offline Learning Setting:

We don't know the MDP and our data collection is under some fixed distribution.

The Finite Horizon, Offline Learning Setting:

- We have N trajectories $\tau_1, \ldots \tau_N \sim \rho_{\pi_{data}}$

The Offline Learning Setting:

We don't know the MDP and our data collection is under some fixed distribution.

The Finite Horizon, Offline Learning Setting:

- We have N trajectories $\tau_1, \ldots \tau_N \sim \rho_{\pi_{data}}$
- π_{data} is often referred to as our data collection policy.

$\pi_{data}(a|s)$

(we $\overset{(1)}{\text{may}}$ or $\underbrace{\overset{(2)}{\text{may not}}}$ know π_{data})

can't do IS

Fitted Q-Iteration

Fitted Q-Iteration

input: **offline dataset** $\tau_1, \ldots \tau_N \sim \rho_{\pi_{data}}$

1. Initialize parameter w_0

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1},\}$)

Fitted Q-Iteration

input: **offline dataset** $\tau_1, \ldots \tau_N \sim \rho_{\pi_{data}}$

1. Initialize parameter w_0

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1}, \})$

2. For $k = 0,1,\ldots K$:
   1. Construct an *empirical loss function*:

$r_h(s,a) + \mathbb{E}_{s' \sim P(s,a)} \left[\max_{a} Q_k(s', a')\right]$

$$\widetilde{L}_k(w) = \frac{1}{N} \sum_{i=1}^{N} \sum_{(s_h, a_h, r_h, s_{h+1}) \in \tau_i} \left(f_w(s_h, a_h) - \left(r_h + \max_a f_{w_k}(s_{h+1}, a) \right) \right)^2$$

Fitted Q-Iteration

input: **offline dataset** $\tau_1, \ldots \tau_N \sim \rho_{\pi_{data}}$

1. Initialize parameter w_0

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1},\}$)

2. For $k = 0,1,\ldots K$:
   1. Construct an *empirical loss function*:

      $$\widetilde{L}_k(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{(s_h,a_h,r_h,s_{h+1})\in\tau_i}\left(f_w(s_h, a_h) - \left(r_h + \max_a f_{w_k}(s_{h+1}, a)\right)\right)^2$$

   2. Update:

      $$\widetilde{w}_{k+1} \approx \arg\min_w \widetilde{L}_k(w)$$

Fitted Q-Iteration

input: **offline dataset** $\tau_1, \ldots \tau_N \sim \rho_{\pi_{data}}$

1. Initialize parameter w_0

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1},\}$)

2. For $k = 0, 1, \ldots K$:
   1. Construct an *empirical loss function*:

      $$\widetilde{L}_k(w) = \frac{1}{N}\sum_{i=1}^{N} \sum_{(s_h, a_h, r_h, s_{h+1}) \in \tau_i} \Big(f_w(s_h, a_h) - \big(r_h + \max_a f_{w_k}(s_{h+1}, a)\big)\Big)^2$$

   2. Update:

      $$\widetilde{w}_{k+1} \approx \arg\min_w \widetilde{L}_k(w)$$

3. Return the function $f_{\widetilde{w}_K}$ as an estimate of $Q^\star$

Fitted Q-Iteration

input: **offline dataset** $\tau_1, \ldots \tau_N \sim \rho_{\pi_{data}}$

1. Initialize parameter w_0

   (each trajectory is of the form $\tau_i = \{s_0, a_0, r_0, \ldots s_{H-1}, a_{H-1}, r_{H-1},\}$)

2. For $k = 0,1,\ldots K$:
   1. Construct an *empirical loss function*:

      $$\widetilde{L}_k(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{(s_h,a_h,r_h,s_{h+1})\in\tau_i}\Big(f_w(s_h, a_h) - \big(r_h + \max_a f_{w_k}(s_{h+1}, a)\big)\Big)^2$$

   2. Update:

      $$\widetilde{w}_{k+1} \approx \arg\min_w \widetilde{L}_k(w)$$

3. Return the function $f_{\widetilde{w}_K}$ as an estimate of $Q^\star$

Q-Learning is an online variant to do the above.

Summary:

1. Fitted Policy Evaluation
2. Fitted Dynamic Programming Methods
   1. Fitted Policy Evaluation
   2. Fitted Q-Value Iteration

Next up: fitted DP methods or PG methods?
TRPO and Natural PG connects these two ideas

1-minute feedback form: https://bit.ly/3RHtlxy