Approximate Reinforcement Learning

Alessandro LAZARIC (*Facebook AI Research* / on leave *Inria Lille*)

ENS Cachan - Master 2 MVA

Approximate Reinforcement Learning

Approximate Reinforcement Learning

Approximate Value Iteration

Approximate Policy Iteration

From Exact to Approximate RL

- Dynamic programming algorithms require an *explicit* definition of
  - transition probabilities $p(\cdot|x, a)$
  - reward function $r(x, a)$

From Exact to Approximate RL

- Dynamic programming algorithms require an *explicit* definition of
  - transition probabilities $p(\cdot|x, a)$
  - reward function $r(x, a)$
- This knowledge is often *unavailable* (i.e., wind intensity, human-computer-interaction).

From Exact to Approximate RL

- Dynamic programming algorithms require an *explicit* definition of
  - transition probabilities $p(\cdot|x,a)$
  - reward function $r(x,a)$
- This knowledge is often *unavailable* (i.e., wind intensity, human-computer-interaction).
- ***Can we rely on samples?*** *(partially addressed by RL)*

From Exact to Approximate RL

- Dynamic programming algorithms require an *exact* representation of value functions and policies

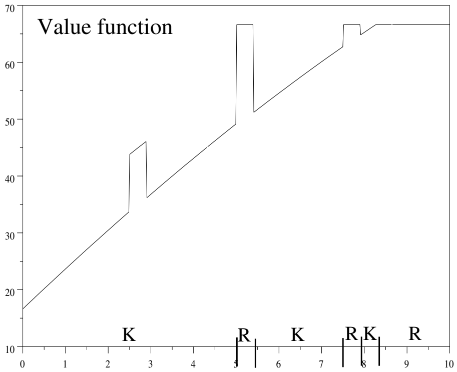
From Exact to Approximate RL

- Dynamic programming algorithms require an *exact* representation of value functions and policies
- This is often *impossible* since their shape is too “complicated” (e.g., large or continuous state space).

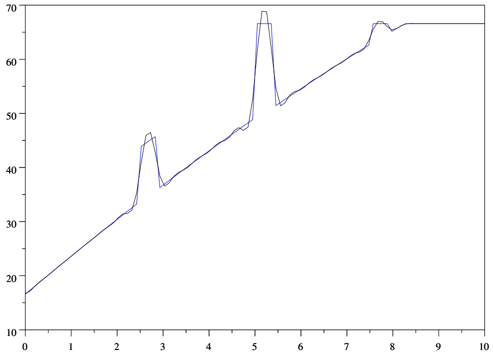
From Exact to Approximate RL

- Dynamic programming algorithms require an *exact* representation of value functions and policies
- This is often *impossible* since their shape is too “complicated” (e.g., large or continuous state space).
- ***Can we use approximations?***

From Exact to Approximate RL



From Exact to Approximate RL



Approximated by a Fourier basis expansion

The Objective

Find a policy π such that

the *performance loss* $\|V^* - V^\pi\|$ is as small as possible

Approximate Reinforcement Learning

Approximate Value Iteration

Approximate Policy Iteration

From Approximation Error to Performance Loss

Question: if V is an approximation of the optimal value function V^* with an error

$$\text{error} = \|V - V^*\|$$

From Approximation Error to Performance Loss

Question: if V is an approximation of the optimal value function V^* with an error

$$\text{error} = \|V - V^*\|$$

how does it translate to the (loss of) performance of the *greedy policy*

$$\pi(x) \in \arg\max_{a \in A} \sum_y p(y|x,a)\big[r(x,a,y) + \gamma V(y)\big]$$

From Approximation Error to Performance Loss

Question: if V is an approximation of the optimal value function V^* with an error

$$\text{error} = \|V - V^*\|$$

how does it translate to the (loss of) performance of the *greedy policy*

$$\pi(x) \in \arg\max_{a \in A} \sum_y p(y|x,a)\big[r(x,a,y) + \gamma V(y)\big]$$

i.e.

$$\text{performance loss} = \|V^* - V^\pi\|$$

From Approximation Error to Performance Loss

Proposition

Let $V \in \mathbb{R}^N$ be an approximation of V^* and π its corresponding greedy policy, then

$$\underbrace{\|V^* - V^\pi\|_\infty}_{\textit{performance loss}} \leq \frac{2\gamma}{1-\gamma} \underbrace{\|V^* - V\|_\infty}_{\textit{approx. error}}.$$

Furthermore, there exists $\epsilon > 0$ such that if $\|V - V^*\|_\infty \leq \epsilon$, then π is *optimal*.

From Approximation Error to Performance Loss

Proof.

$$\begin{aligned}
\|V^* - V^\pi\|_\infty &\le \|\mathcal{T}V^* - \mathcal{T}^\pi V\|_\infty + \|\mathcal{T}^\pi V - \mathcal{T}^\pi V^\pi\|_\infty \\
&\le \|\mathcal{T}V^* - \mathcal{T}V\|_\infty + \gamma\|V - V^\pi\|_\infty \\
&\le \gamma\|V^* - V\|_\infty + \gamma(\|V - V^*\|_\infty + \|V^* - V^\pi\|_\infty) \\
&\le \frac{2\gamma}{1-\gamma}\|V^* - V\|_\infty.
\end{aligned}$$

■

From Approximation Error to Performance Loss

Question: how do we compute a *good* V?

From Approximation Error to Performance Loss

Question: how do we compute a *good* V?

Problem: unlike in standard approximation scenarios (see supervised learning), we have a *limited access* to the target function, i.e. V^*.

From Approximation Error to Performance Loss

Question: how do we compute a *good* V?

Problem: unlike in standard approximation scenarios (see supervised learning), we have a *limited access* to the target function, i.e. V^*.

Solution: value iteration tends to learn functions which are *close to the optimal value function* V^*.

Value Iteration: the Idea

1. Let Q_0 be *any* action-value function
2. At each iteration $k = 1, 2, \ldots, K$
   - Compute
     $$Q_{k+1}(x,a) = \mathcal{T} Q_k(x,a) = r(x,a) + \sum_y p(y|x,a) \gamma \max_b Q_k(y,b)$$
3. Return the *greedy* policy

$$\pi_K(x) \in \arg\max_{a \in A} Q_K(x,a).$$

Value Iteration: the Idea

1. Let Q_0 be *any* action-value function
2. At each iteration $k = 1, 2, \ldots, K$
    - Compute
      $$Q_{k+1}(x,a) = \mathcal{T} Q_k(x,a) = r(x,a) + \sum_y p(y|x,a) \gamma \max_b Q_k(y,b)$$
3. Return the *greedy* policy

$$\pi_K(x) \in \arg\max_{a \in A} Q_K(x,a).$$

- ***Problem***: how can we approximate $\mathcal{T} Q_k$?
- ***Problem***: if $Q_{k+1} \neq \mathcal{T} Q_k$, does (approx.) value iteration still work?

Linear Fitted Q-iteration: the Approximation Space

Linear space to approximate action–value functions

$$\mathcal{F} = \Big\{ f(x,a) = \sum_{j=1}^{d} \alpha_j \varphi_j(x,a), \;\; \alpha \in \mathbb{R}^{d} \Big\}$$

Linear Fitted Q-iteration: the Approximation Space

Linear space to approximate action–value functions

$$\mathcal{F} = \left\{ f(x,a) = \sum_{j=1}^{d} \alpha_j \varphi_j(x,a), \;\; \alpha \in \mathbb{R}^d \right\}$$

with features (*alternative for discrete actions*: duplicate state features)

$$\varphi_j : X \times A \to [0, L] \qquad \phi(x,a) = [\varphi_1(x,a) \dots \varphi_d(x,a)]^\top$$

Linear Fitted Q-iteration: the Samples

Assumption: access to a ***generative model***, that is a black-box simulator *sim()* of the environment is available.
Given (x, a),

$$\mathrm{sim}(x,a) = \{y, r\}, \qquad \text{with } y \sim p(\cdot|x,a), r = r(x,a)$$

Linear Fitted Q-iteration

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Linear Fitted Q-iteration

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Initial function $\widehat{Q}_0 \in \mathcal{F}$

Linear Fitted Q-iteration

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

```
Initial function $\widehat{Q}_0 \in \mathcal{F}$
For $k = 1, \dots, K$
```

Linear Fitted Q-iteration

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

```
Initial function $\widehat{Q}_0 \in \mathcal{F}$
For $k = 1, \dots, K$
    1. Draw $n$ samples $(x_i, a_i) \overset{\text{i.i.d}}{\sim} \rho$
```

Linear Fitted Q-iteration

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Initial function $\widehat{Q}_0 \in \mathcal{F}$
For $k = 1, \ldots, K$

1. Draw n samples $(x_i, a_i) \overset{\text{i.i.d}}{\sim} \rho$
2. Sample $x_i' \sim p(\cdot|x_i, a_i)$ and $r_i = r(x_i, a_i)$

Linear Fitted Q-iteration

```
**Input**: space $\mathcal{F}$, iterations $K$, sampling distribution $\rho$, num of samples $n$

Initial function $\widehat{Q}_0 \in \mathcal{F}$
For $k = 1, \dots, K$
    1. Draw $n$ samples $(x_i, a_i) \overset{\text{i.i.d}}{\sim} \rho$
    2. Sample $x'_i \sim p(\cdot|x_i, a_i)$ and $r_i = r(x_i, a_i)$
    3. Compute $y_i = r_i + \gamma \max_a \widehat{Q}_{k-1}(x'_i, a)$
```

Linear Fitted Q-iteration

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Initial function $\widehat{Q}_0 \in \mathcal{F}$
For $k = 1, \dots, K$

1. Draw n samples $(x_i, a_i) \overset{\text{i.i.d}}{\sim} \rho$
2. Sample $x_i' \sim p(\cdot|x_i, a_i)$ and $r_i = r(x_i, a_i)$
3. Compute $y_i = r_i + \gamma \max_a \widehat{Q}_{k-1}(x_i', a)$
4. Build training set $\big\{\big((x_i, a_i), y_i\big)\big\}_{i=1}^n$

Linear Fitted Q-iteration

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Initial function $\widehat{Q}_0 \in \mathcal{F}$
For $k = 1, \ldots, K$

1. Draw n samples $(x_i, a_i) \overset{\text{i.i.d}}{\sim} \rho$
2. Sample $x_i' \sim p(\cdot|x_i, a_i)$ and $r_i = r(x_i, a_i)$
3. Compute $y_i = r_i + \gamma \max_a \widehat{Q}_{k-1}(x_i', a)$
4. Build training set $\big\{\big((x_i, a_i), y_i\big)\big\}_{i=1}^n$
5. Solve the *least squares problem*

$$f_{\hat{\alpha}_k} = \arg\min_{f_\alpha \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \big(f_\alpha(x_i, a_i) - y_i\big)^2$$

Linear Fitted Q-iteration

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Initial function $\widehat{Q}_0 \in \mathcal{F}$
For $k = 1, \ldots, K$

1. Draw n samples $(x_i, a_i) \overset{\text{i.i.d}}{\sim} \rho$
2. Sample $x_i' \sim p(\cdot|x_i, a_i)$ and $r_i = r(x_i, a_i)$
3. Compute $y_i = r_i + \gamma \max_a \widehat{Q}_{k-1}(x_i', a)$
4. Build training set $\big\{\big((x_i, a_i), y_i\big)\big\}_{i=1}^n$
5. Solve the *least squares problem*
$$f_{\hat{\alpha}_k} = \arg\min_{f_\alpha \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \big(f_\alpha(x_i, a_i) - y_i\big)^2$$
6. Return $\widehat{Q}_k = f_{\hat{\alpha}_k}$ *(truncation may be needed)*

Linear Fitted Q-iteration

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Initial function $\widehat{Q}_0 \in \mathcal{F}$
For $k = 1, \dots, K$

1. Draw n samples $(x_i, a_i) \overset{\text{i.i.d}}{\sim} \rho$
2. Sample $x'_i \sim p(\cdot|x_i, a_i)$ and $r_i = r(x_i, a_i)$
3. Compute $y_i = r_i + \gamma \max_a \widehat{Q}_{k-1}(x'_i, a)$
4. Build training set $\big\{\big((x_i, a_i), y_i\big)\big\}_{i=1}^n$
5. Solve the *least squares problem*
$$f_{\hat{\alpha}_k} = \arg\min_{f_\alpha \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \big(f_\alpha(x_i, a_i) - y_i\big)^2$$
6. Return $\widehat{Q}_k = f_{\hat{\alpha}_k}$ *(truncation may be needed)*

Return $\pi_K(\cdot) = \arg\max_a \widehat{Q}_K(\cdot, a)$ (*greedy policy*)

Linear Fitted Q-iteration: Sampling

1. Draw n samples $(x_i, a_i) \overset{\text{i.i.d}}{\sim} \rho$
2. Sample $x_i' \sim p(\cdot|x_i, a_i)$ and $r_i = r(x_i, a_i)$

Linear Fitted Q-iteration: Sampling

1. Draw n samples $(x_i, a_i) \overset{\text{i.i.d}}{\sim} \rho$
2. Sample $x_i' \sim p(\cdot|x_i, a_i)$ and $r_i = r(x_i, a_i)$

- In practice it can be done *once* before running the algorithm
- The sampling distribution ρ should cover the state-action space in all *relevant* regions
- If not possible to choose ρ, a *database* of samples can be used

Linear Fitted Q-iteration: The Training Set

4. Compute $y_i = r_i + \gamma \max_a \widehat{Q}_{k-1}(x_i', a)$
5. Build training set $\big\{\big((x_i, a_i), y_i\big)\big\}_{i=1}^n$

Linear Fitted Q-iteration: The Training Set

4. Compute $y_i = r_i + \gamma \max_a \widehat{Q}_{k-1}(x'_i, a)$
5. Build training set $\big\{\big((x_i, a_i), y_i\big)\big\}_{i=1}^{n}$

- Each sample y_i is an unbiased sample, since

$$\begin{aligned}\mathbb{E}[y_i|x_i, a_i] = \mathbb{E}[r_i + \gamma \max_a \widehat{Q}_{k-1}(x'_i, a)] &= r(x_i, a_i) + \gamma \mathbb{E}[\max_a \widehat{Q}_{k-1}(x'_i, a)] \\ &= r(x_i, a_i) + \gamma \int_X \max_a \widehat{Q}_{k-1}(x', a) p(dy|x, a) = \mathcal{T}\widehat{Q}_{k-1}(x_i, a_i)\end{aligned}$$

- The problem "reduces" to standard *regression*
- It should be recomputed at each iteration

Linear Fitted Q-iteration: The Regression Problem

6. Solve the *least squares problem*

$$f_{\hat{\alpha}_k} = \arg\min_{f_\alpha \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \big(f_\alpha(x_i, a_i) - y_i\big)^2$$

7. Return $\widehat{Q}_k = f_{\hat{\alpha}_k}$ *(truncation may be needed)*

Linear Fitted Q-iteration: The Regression Problem

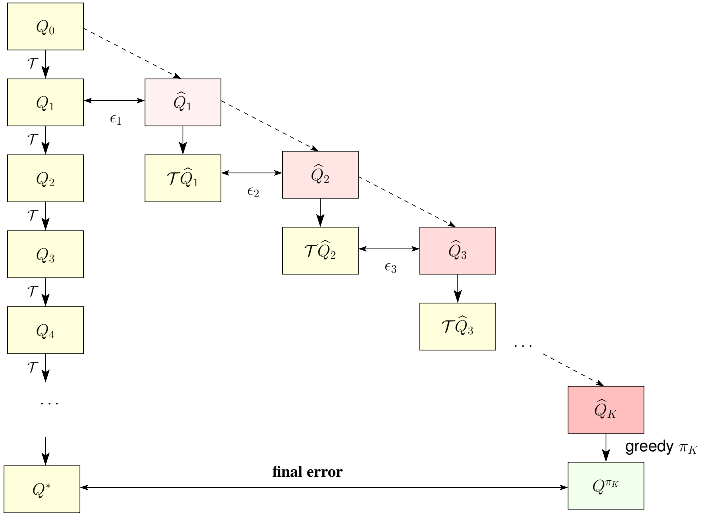
6. Solve the *least squares problem*

$$f_{\hat{\alpha}_k} = \arg\min_{f_\alpha \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \big(f_\alpha(x_i, a_i) - y_i\big)^2$$

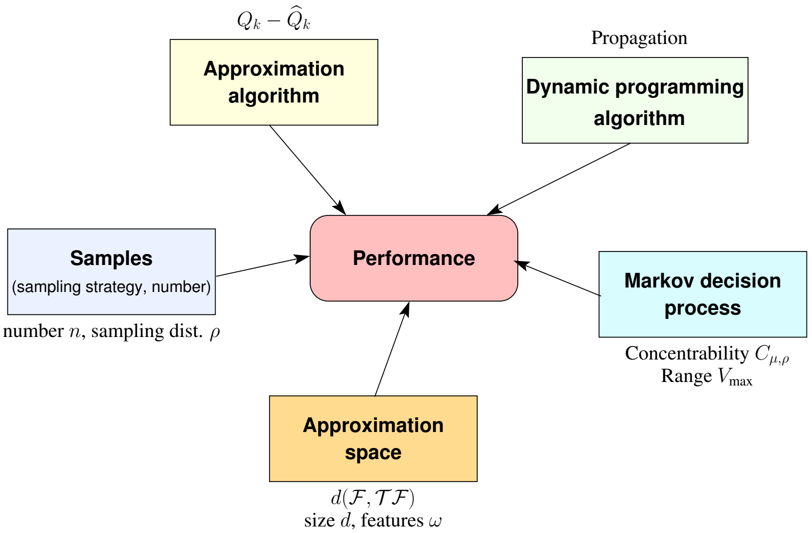
7. Return $\widehat{Q}_k = f_{\hat{\alpha}_k}$ *(truncation may be needed)*

- Thanks to the linear space we can solve it as
  - Build matrix $\Phi = \big[\phi(x_1, a_1)^\top \dots \phi(x_n, a_n)^\top\big]$
  - Compute $\hat{\alpha}^k = (\Phi^\top \Phi)^{-1} \Phi^\top y$ (*least–squares solution)*
- Truncation to $[-V_{\max}; V_{\max}]$ (with $V_{\max} = R_{\max}/(1-\gamma)$)

Sketch of the Analysis



Summary



$Q_k - \widehat{Q}_k$
Approximation algorithm
Propagation
Dynamic programming algorithm
Samples
(sampling strategy, number)
number n, sampling dist. ρ
Performance
Markov decision process
Concentrability $C_{\mu,\rho}$
Range $V_{\max}$
Approximation space
$d(\mathcal{F}, \mathcal{T}\mathcal{F})$
size d, features ω

The Final Bound

Theorem (see e.g., Munos,'03)

LinearFQI with a space $\mathcal{F}$ of d features, with n samples at each iteration returns a policy π_K after K iterations such that

$$\|Q^* - Q^{\pi_K}\|_\mu \leq \frac{2\gamma}{(1-\gamma)^2}\sqrt{C_{\mu,\rho}}\Bigg(4d(\mathcal{F},\mathcal{T}\mathcal{F}) + O\Big(V_{\max}\big(1+\frac{L}{\sqrt{\omega}}\big)\sqrt{\frac{d\log n/\delta}{n}}\Big)\Bigg) + O\Big(\frac{\gamma^K}{(1-\gamma)^3}{V_{\max}}^2\Big)$$

Other implementations

Replace the *regression* step with

- K-nearest neighbour
- Regularized linear regression with L_1 or L_2 regularisation
- Neural network
- Support vector regression
- Trees

Other implementations

Replace the *regression* step with

- K-nearest neighbour
- Regularized linear regression with L_1 or L_2 regularisation
- Neural network
- Support vector regression
- Trees

Remark: we need to solve the approximation problem *efficiently*

Approximate Reinforcement Learning

Approximate Value Iteration

Approximate Policy Iteration

Policy Iteration: the Idea

1. Let π_0 be *any* stationary policy
2. At each iteration $k = 1, 2, \dots, K$
   - *Policy evaluation* given π_k, compute $V_k = V^{\pi_k}$.
   - *Policy improvement*: compute the *greedy* policy

$$\pi_{k+1}(x) \in \arg\max_{a \in A} \Big[r(x,a) + \gamma \sum_y p(y|x,a) V^{\pi_k}(y) \Big].$$

3. Return the last policy π_K

Policy Iteration: the Idea

1. Let π_0 be *any* stationary policy
2. At each iteration $k = 1, 2, \dots, K$
    - *Policy evaluation* given π_k, compute $V_k = V^{\pi_k}$.
    - *Policy improvement*: compute the *greedy* policy

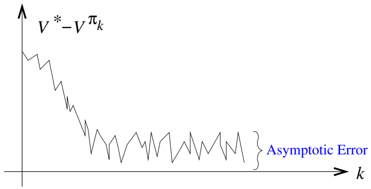
$$\pi_{k+1}(x) \in \arg\max_{a \in A}\Big[r(x,a) + \gamma \sum_y p(y|x,a) V^{\pi_k}(y)\Big].$$

3. Return the last policy π_K

- ***Problem***: how can we approximate V^{π_k}?
- ***Problem***: if $V_k \neq V^{\pi_k}$, does (approx.) policy iteration still work?

Approximate Policy Iteration: performance loss

Problem: the algorithm is no longer guaranteed to converge.



Proposition

The asymptotic performance of the policies π_k generated by the API algorithm is related to the approximation error as:

$$\limsup_{k\to\infty} \underbrace{\|V^* - V^{\pi_k}\|_\infty}_{\text{performance loss}} \leq \frac{2\gamma}{(1-\gamma)^2} \limsup_{k\to\infty} \underbrace{\|V_k - V^{\pi_k}\|_\infty}_{\text{approximation error}}$$

Least-Squares Policy Iteration (LSPI)

LSPI uses

- Linear space to approximate value functions*

$$\mathcal{F} = \Big\{ f(x) = \sum_{j=1}^{d} \alpha_j \varphi_j(x), \;\; \alpha \in \mathbb{R}^{d} \Big\}$$

Least-Squares Policy Iteration (LSPI)

LSPI uses

- Linear space to approximate value functions*

$$\mathcal{F} = \Big\{ f(x) = \sum_{j=1}^{d} \alpha_j \varphi_j(x), \;\; \alpha \in \mathbb{R}^d \Big\}$$

- Least-Squares Temporal Difference (LSTD) algorithm for *policy evaluation*.

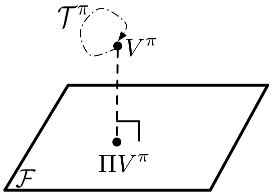
*In practice we use approximations of action-value functions.

Least-Squares Temporal-Difference Learning (LSTD)

- V^π may not belong to $\mathcal{F}$ $\quad V^\pi \notin \mathcal{F}$
- Best approximation of V^π in $\mathcal{F}$ is

$$\Pi V^\pi = \arg\min_{f \in \mathcal{F}} ||V^\pi - f||$$

(Π is the projection onto $\mathcal{F}$)



Least-Squares Temporal-Difference Learning (LSTD)

- V^π is the fixed-point of $\mathcal{T}^\pi$

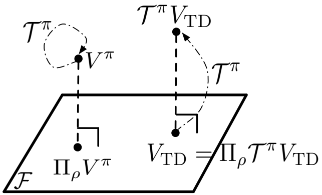
$$V^\pi = \mathcal{T}^\pi V^\pi = r^\pi + \gamma P^\pi V^\pi$$

- LSTD searches for the fixed-point of $\Pi_{2,\rho}\mathcal{T}^\pi$

$$\Pi_{2,\rho}\, g = \arg\min_{f\in\mathcal{F}} ||g - f||_{2,\rho}$$

- ***When*** the fixed-point of $\Pi_\rho \mathcal{T}^\pi$ exists, we call it the LSTD solution

$$V_{\text{TD}} = \Pi_\rho \mathcal{T}^\pi V_{\text{TD}}$$


$\mathcal{T}^\pi$
V^π
$\mathcal{T}^\pi V_{\text{TD}}$
$\mathcal{T}^\pi$
$\Pi_\rho V^\pi$
$V_{\text{TD}} = \Pi_\rho \mathcal{T}^\pi V_{\text{TD}}$
$\mathcal{F}$

Least-Squares Temporal-Difference Learning (LSTD)

$$V_{\text{TD}} = \Pi_\rho \mathcal{T}^\pi V_{\text{TD}}$$

$$\Downarrow$$

$$\underbrace{\langle r^\pi, \varphi_i \rangle_\rho}_{b_i} - \sum_{j=1}^{d} \underbrace{\langle (I - \gamma P^\pi)\varphi_j, \varphi_i \rangle_\rho}_{A_{i,j}} \alpha_{TD,j} = 0$$

$$\Downarrow$$

$$A\alpha_{TD} = b$$

Least-Squares Policy Iteration (LSPI)

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Least-Squares Policy Iteration (LSPI)

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Initial policy π_0

Least-Squares Policy Iteration (LSPI)

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

```
Initial policy $\pi_0$
For $k = 1, \ldots, K$
```

Least-Squares Policy Iteration (LSPI)

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

```
Initial policy $\pi_0$
For $k = 1, \ldots, K$
  1. Generate a trajectory of length $n$ from the stationary dist. $\rho^{\pi_k}$
```

$$(x_1, \pi_k(x_1), r_1, x_2, \pi_k(x_2), r_2, \ldots, x_{n-1}, \pi_k(x_{n-1}), r_{n-1}, x_n)$$

Least-Squares Policy Iteration (LSPI)

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Initial policy π_0
For $k = 1, \dots, K$

1. Generate a trajectory of length n from the stationary dist. ρ^{π_k}

$$(x_1, \pi_k(x_1), r_1, x_2, \pi_k(x_2), r_2, \dots, x_{n-1}, \pi_k(x_{n-1}), r_{n-1}, x_n)$$

2. Compute the empirical matrix $\widehat{A}_k$ and the vector $\widehat{b}_k$

$$[\widehat{A}_k]_{i,j} = \frac{1}{n}\sum_{t=1}^{n}(\varphi_j(x_t) - \gamma\varphi_j(x_{t+1})\varphi_i(x_t) \approx \langle (I - \gamma P^{\pi})\varphi_j, \varphi_i \rangle_{\rho^{\pi_k}}$$

$$[\widehat{b}_k]_i = \frac{1}{n}\sum_{t=1}^{n}\varphi_i(x_t) r_t \approx \langle r^{\pi}, \varphi_i \rangle_{\rho^{\pi_k}}$$

3. Solve the linear system $\alpha_k = \widehat{A}_k^{-1}\widehat{b}_k$

Least-Squares Policy Iteration (LSPI)

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

Initial policy π_0
For $k = 1, \dots, K$

1. Generate a trajectory of length n from the stationary dist. ρ^{π_k}
$$(x_1, \pi_k(x_1), r_1, x_2, \pi_k(x_2), r_2, \dots, x_{n-1}, \pi_k(x_{n-1}), r_{n-1}, x_n)$$
2. Compute the empirical matrix $\widehat{A}_k$ and the vector $\widehat{b}_k$
$$[\widehat{A}_k]_{i,j} = \frac{1}{n}\sum_{t=1}^{n}(\varphi_j(x_t) - \gamma\varphi_j(x_{t+1})\varphi_i(x_t) \approx \langle (I - \gamma P^{\pi})\varphi_j, \varphi_i \rangle_{\rho^{\pi_k}}$$
$$[\widehat{b}_k]_i = \frac{1}{n}\sum_{t=1}^{n}\varphi_i(x_t) r_t \approx \langle r^{\pi}, \varphi_i \rangle_{\rho^{\pi_k}}$$
3. Solve the linear system $\alpha_k = \widehat{A}_k^{-1}\widehat{b}_k$
4. Compute the greedy policy π_{k+1} w.r.t. $\widehat{V}_k = f_{\alpha_k}$

Least-Squares Policy Iteration (LSPI)

Input: space $\mathcal{F}$, iterations K, sampling distribution ρ, num of samples n

```
Initial policy $\pi_0$
For $k = 1, \ldots, K$
  1. Generate a trajectory of length $n$ from the stationary dist. $\rho^{\pi_k}$
```

$$(x_1, \pi_k(x_1), r_1, x_2, \pi_k(x_2), r_2, \ldots, x_{n-1}, \pi_k(x_{n-1}), r_{n-1}, x_n)$$

```
  2. Compute the empirical matrix $\widehat{A}_k$ and the vector $\widehat{b}_k$
```

$$[\widehat{A}_k]_{i,j} = \frac{1}{n}\sum_{t=1}^{n}(\varphi_j(x_t) - \gamma\varphi_j(x_{t+1})\varphi_i(x_t) \approx \langle (I - \gamma P^{\pi})\varphi_j, \varphi_i\rangle_{\rho^{\pi_k}}$$

$$[\widehat{b}_k]_i = \frac{1}{n}\sum_{t=1}^{n}\varphi_i(x_t)r_t \approx \langle r^{\pi}, \varphi_i\rangle_{\rho^{\pi_k}}$$

```
  3. Solve the linear system $\alpha_k = \widehat{A}_k^{-1}\widehat{b}_k$
  4. Compute the greedy policy $\pi_{k+1}$ w.r.t. $\widehat{V}_k = f_{\alpha_k}$
```

Return the last policy π_K

Least-Squares Policy Iteration (LSPI)

1. Generate a trajectory of length n from the stationary dist. ρ^{π_k}

$$(x_1, \pi_k(x_1), r_1, x_2, \pi_k(x_2), r_2, \ldots, x_{n-1}, \pi_k(x_{n-1}), r_{n-1}, x_n)$$

- The first few samples may be *discarded* because not actually drawn from the *stationary* distribution ρ^{π_k}
- *Off-policy* samples could be used with *importance weighting*
- In practice i.i.d. states drawn from an arbitrary distribution (but with actions π_k) may be used

Least-Squares Policy Iteration (LSPI)

4. Compute the greedy policy π_{k+1} w.r.t. $\widehat{V}_k = f_{\alpha_k}$

- Computing the greedy policy from $\widehat{V}_k$ is difficult, so move to LSTD-Q and compute

$$\pi_{k+1}(x) = \arg\max_a \widehat{Q}_k(x, a)$$

Least-Squares Policy Iteration (LSPI)

For $k = 1, \ldots, K$

Least-Squares Policy Iteration (LSPI)

For $k = 1, \ldots, K$

1. Generate a trajectory of length n from the stationary dist. ρ^{π_k}

$$(x_1, \pi_k(x_1), r_1, x_2, \pi_k(x_2), r_2, \ldots, x_{n-1}, \pi_k(x_{n-1}), r_{n-1}, x_n)$$

...

4. Compute the greedy policy π_{k+1} w.r.t. $\widehat{V}_k = f_{\alpha_k}$

Problem: This process may be unstable because π_k ***does not cover*** the state space *properly*

LSTD Algorithm

When $n \to \infty$ then $\widehat{A} \to A$ and $\widehat{b} \to b$, and thus,

$$\widehat{\alpha}_{\mathrm{TD}} \to \alpha_{\mathrm{TD}} \text{ and } \widehat{V}_{\mathrm{TD}} \to V_{\mathrm{TD}}$$

Proposition (LSTD Performance)

If LSTD is used to estimate the value of π with an ***infinite*** number of samples drawn from the stationary distribution ρ^π then

$$||V^\pi - V_{\mathrm{TD}}||_{\rho^\pi} \leq \frac{1}{\sqrt{1-\gamma^2}} \inf_{V \in \mathcal{F}} ||V^\pi - V||_{\rho^\pi}$$

LSTD Algorithm

When $n \to \infty$ then $\widehat{A} \to A$ and $\widehat{b} \to b$, and thus,

$$\widehat{\alpha}_{\text{TD}} \to \alpha_{\text{TD}} \text{ and } \widehat{V}_{\text{TD}} \to V_{\text{TD}}$$

Proposition (LSTD Performance)

If LSTD is used to estimate the value of π with an ***infinite*** number of samples drawn from the stationary distribution ρ^π then

$$||V^\pi - V_{\text{TD}}||_{\rho^\pi} \leq \frac{1}{\sqrt{1-\gamma^2}} \inf_{V \in \mathcal{F}} ||V^\pi - V||_{\rho^\pi}$$

Problem: we don't have an infinite number of samples...

LSTD Algorithm

When $n \to \infty$ then $\widehat{A} \to A$ and $\widehat{b} \to b$, and thus,

$$\widehat{\alpha}_{\text{TD}} \to \alpha_{\text{TD}} \text{ and } \widehat{V}_{\text{TD}} \to V_{\text{TD}}$$

Proposition (LSTD Performance)

If LSTD is used to estimate the value of π with an ***infinite*** number of samples drawn from the stationary distribution ρ^π then

$$||V^\pi - V_{\text{TD}}||_{\rho^\pi} \leq \frac{1}{\sqrt{1-\gamma^2}} \inf_{V \in \mathcal{F}} ||V^\pi - V||_{\rho^\pi}$$

Problem: we don't have an infinite number of samples...
Problem 2: V_{TD} is a fixed point solution and not a standard machine learning problem...

LSPI Error Bound

Theorem (LSPI Error Bound)

If LSPI is run over K iterations, then the performance loss policy π_K is

$$||V^* - V^{\pi_K}||_\mu \leq \frac{4\gamma}{(1-\gamma)^2} \left\{ \sqrt{CC_{\mu,\rho}} \left[E_0(\mathcal{F}) + O\left(\sqrt{\frac{d \log(dK/\delta)}{n\, \nu_\rho}} \right) \right] + \gamma^K R_{\max} \right\}$$

with probability $1 - \delta$.

Approximate Reinforcement Learning

Approximate Temporal Difference / Q-Learning

TD as a Gradient Algorithm

- *Ideal* regression problem: given functions V_θ and distribution $\mathcal{D}$

$$\min_\theta L(\theta) = \min_\theta \mathbb{E}_{\mathcal{D}}\Big[\big(V^\pi(x) - V_\theta(x)\big)^2\Big]$$

TD as a Gradient Algorithm

- *Ideal* regression problem: given functions V_θ and distribution $\mathcal{D}$

$$\min_\theta L(\theta) = \min_\theta \mathbb{E}_\mathcal{D}\Big[\big(V^\pi(x) - V_\theta(x)\big)^2\Big]$$

- Gradient descent

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta L(\theta) = -\alpha\mathbb{E}_\mathcal{D}\Big[\big(V^\pi(x) - V_\theta(x)\big)\nabla_\theta V_\theta(x)\Big]$$

TD as a Gradient Algorithm

- *Ideal* regression problem: given functions V_θ and distribution $\mathcal{D}$

$$\min_\theta L(\theta) = \min_\theta \mathbb{E}_{\mathcal{D}}\Big[\big(V^\pi(x) - V_\theta(x)\big)^2\Big]$$

- Gradient descent

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta L(\theta) = -\alpha\mathbb{E}_{\mathcal{D}}\Big[\big(V^\pi(x) - V_\theta(x)\big)\nabla_\theta V_\theta(x)\Big]$$

- Gradient descent (sample x from distribution $\mathcal{D}$)

$$\Delta\theta = -\alpha\big(V^\pi(x) - V_\theta(x)\big)\nabla_\theta V_\theta(x)$$

TD as a Gradient Algorithm

- Replace *unknown* V^π by its one-step estimate

$$\Delta\theta = -\alpha\big(V^\pi(x) - V_\theta(x)\big)\nabla_\theta V_\theta(x)$$

$$\Rightarrow \Delta\theta_t = -\alpha\big(r_t + \gamma V_\theta(x_{t+1}) - V_\theta(x_t)\big)\nabla_\theta V_\theta(x_t)$$

TD as a Gradient Algorithm

- Replace *unknown* V^π by its one-step estimate

$$\Delta\theta = -\alpha\big(V^\pi(x) - V_\theta(x)\big)\nabla_\theta V_\theta(x)$$

$$\Rightarrow \Delta\theta_t = -\alpha\big(r_t + \gamma V_\theta(x_{t+1}) - V_\theta(x_t)\big)\nabla_\theta V_\theta(x_t)$$

- Converges if samples are obtained *on-policy* and *linear* approximation (may *diverge* with off-policy samples)
- Improved convergence guarantees obtained with *Bellman residual* variants (GTD2, TDC)

Q-learning as a Gradient Algorithm

- Regression problem *(ideal)*: given functions $V_\theta(x)$

$$\min_\theta L(\theta) = \min_\theta \mathbb{E}_{\mathcal{D}}\Big[\big(Q^*(x,a) - Q_\theta(x,a)\big)^2\Big]$$

Q-learning as a Gradient Algorithm

- Regression problem *(ideal)*: given functions $V_\theta(x)$

$$\min_\theta L(\theta) = \min_\theta \mathbb{E}_{\mathcal{D}}\Big[\big(Q^*(x,a) - Q_\theta(x,a)\big)^2\Big]$$

- Gradient descent

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta L(\theta) = -\alpha\mathbb{E}_{\mathcal{D}}\Big[\big(Q^*(x,a) - Q_\theta(x,a)\big)\nabla_\theta Q_\theta(x,a)\Big]$$

Q-learning as a Gradient Algorithm

- Regression problem *(ideal)*: given functions $V_\theta(x)$

$$\min_\theta L(\theta) = \min_\theta \mathbb{E}_{\mathcal{D}}\Big[\big(Q^*(x,a) - Q_\theta(x,a)\big)^2\Big]$$

- Gradient descent

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta L(\theta) = -\alpha\mathbb{E}_{\mathcal{D}}\Big[\big(Q^*(x,a) - Q_\theta(x,a)\big)\nabla_\theta Q_\theta(x,a)\Big]$$

- Gradient descent (sample x from distribution $\mathcal{D}$)

$$\Delta\theta = -\alpha\big(Q^*(x,a) - Q_\theta(x,a)\big)\nabla_\theta Q_\theta(x,a)$$

Q-learning as a Gradient Algorithm

- Replace *unknown* Q^* by its one-step estimate

$$\Delta\theta = -\alpha\big(Q^*(x,a) - Q_\theta(x,a)\big)\nabla_\theta Q_\theta(x,a)$$

$$\Rightarrow \Delta\theta_t = -\alpha\big(r_t + \gamma \max_b Q_\theta(x_{t+1},b) - Q_\theta(x_t,a_t)\big)\nabla_\theta Q_\theta(x_t,a_t)$$

Q-learning as a Gradient Algorithm

- Replace *unknown* Q^* by its one-step estimate

$$\Delta\theta = -\alpha\big(Q^*(x,a) - Q_\theta(x,a)\big)\nabla_\theta Q_\theta(x,a)$$

$$\Rightarrow \Delta\theta_t = -\alpha\big(r_t + \gamma \max_b Q_\theta(x_{t+1},b) - Q_\theta(x_t,a_t)\big)\nabla_\theta Q_\theta(x_t,a_t)$$

- May *diverge* even with a linear approximator

Deep Q-Network (DQN)

aka Semi-batch Q-learning / semi-online fitted value iteration

- Construct a memory $D = \{(x_i, a_i, x_i', r_i)\}_{i=1}^n$
- Sample a mini-batch D_{mini} at random from D
- Compute the desired output (for all i in D_{mini})

$$y_i = r_i + \gamma \max_b Q(x_i', b)$$

- Minimize (e.g., with SGD) (as in FVI+approxQL)

$$L_{\text{mini}}(\theta) = \mathbb{E}_{i \sim \mathcal{D}_{\text{mini}}}\left[\left(y_i - Q_\theta(x_i, a_i)\right)^2\right]$$

Deep Q-Network (DQN)

aka Semi-batch Q-learning / semi-online fitted value iteration

- Construct a memory $D = \{(x_i, a_i, x'_i, r_i)\}_{i=1}^n$
- Sample a mini-batch D_{mini} at random from D
- Compute the desired output (for all i in D_{mini})

$$y_i = r_i + \gamma \max_b Q(x'_i, b)$$

- Minimize (e.g., with SGD) (as in FVI+approxQL)

$$L_{\text{mini}}(\theta) = \mathbb{E}_{i \sim \mathcal{D}_{\text{mini}}}\left[\left(y_i - Q_\theta(x_i, a_i)\right)^2\right]$$

No proof of convergence but mini-batch training (and other "tricks") improve stability

Extensions

Alternative algorithms

- TD(λ) (*better sample efficiency*)
- GTD, GTD2, GQ (*stronger convergence guarantees with linear approximators*)
- Use "stable" function approximators (e.g., averagers)
- Use off-policy data

Improvements: if TD/QL are gradient descent algorithms we can apply all the machinery from gradient descent literature (e.g., variance reduction)

Approximate Reinforcement Learning

Policy Gradient Methods

The Objective Function

- Define a parameterized (and differentiable) policy π_θ (*stochastic in general*)
- Define a desired distribution ρ over $\mathcal{X}$
- Objective function

$$J(\theta) = \mathbb{E}_{x\sim\rho}\big[V^{\pi_\theta}(x)\big]$$

The Objective Function

- Define a parameterized (and differentiable) policy π_θ (*stochastic in general*)
- Define a desired distribution ρ over $\mathcal{X}$
- Objective function

$$J(\theta) = \mathbb{E}_{x \sim \rho}\big[V^{\pi_\theta}(x)\big]$$

Idea1: use *global optimizers* or gradient by *finite-difference* methods
$\Rightarrow$ *Policy search / Black-box policy optimization*

The Objective Function

- Define a parameterized (and differentiable) policy π_θ (*stochastic in general*)
- Define a desired distribution ρ over $\mathcal{X}$
- Objective function

$$J(\theta) = \mathbb{E}_{x\sim\rho}\big[V^{\pi_\theta}(x)\big]$$

Idea1: use *global optimizers* or gradient by *finite-difference* methods
$\Rightarrow$ *Policy search / Black-box policy optimization*

Idea2: compute the *gradient* $\nabla_\theta J(\theta)$ and follow gradient *ascent* on policies
$\Rightarrow$ *(white-box) policy gradient*

From Policy Iteration to Policy Search

Approximate policy iteration

$$\pi_{\theta_{k+1}} = \arg\max_{\pi_\theta} Q^{\pi_{\theta_k}}(x, \pi_\theta(x))$$

Policy gradient

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k)$$

From Policy Iteration to Policy Search

Approximate policy iteration

$$\pi_{\theta_{k+1}} = \arg\max_{\pi_\theta} Q^{\pi_{\theta_k}}(x, \pi_\theta(x))$$

Big jumps $\rightarrow$ *fast but unstable*

Policy gradient

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k)$$

Small shift $\rightarrow$ *slow but stable*

From Policy Iteration to Policy Search

Approximate policy iteration

$$\pi_{\theta_{k+1}} = \arg\max_{\pi_\theta} Q^{\pi_{\theta_k}}(x, \pi_\theta(x))$$

Big jumps $\rightarrow$ *fast but unstable*

Policy gradient

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k)$$

Small shift $\rightarrow$ *slow but stable*

How do we compute $\nabla_\theta J$?

Policy Gradient Theorem

Theorem

For any differentiable policy $\pi_\theta(a|s)$ and objective function J, the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log\left(\pi_\theta(a|x)\right) Q^{\pi_\theta}(x,a)\right]$$

Policy Gradient Theorem

Theorem

For any differentiable policy $\pi_\theta(a|s)$ and objective function J, the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log\big(\pi_\theta(a|x)\big) Q^{\pi_\theta}(x,a)\Big]$$

Expectation w.r.t. policy (states from stationary distribution)

$$\nabla_\theta J(\theta) = \sum_{x\in X} \rho^{\pi_\theta}(x)\Big[\nabla_\theta \log\big(\pi_\theta(a|x)\big) Q^{\pi_\theta}(x,a)\Big]$$

Policy Gradient Theorem: Rough Idea

Let $\tau = (x_1, a_1, r_1, \ldots, x_T)$ a trajectory and $R(\tau)$ its return (i.e., sum of rewards)

$$J(\theta) = \sum_\tau \mathbb{P}(\tau|\pi_\theta) R(\tau)$$

Gradient of J

$$\begin{aligned}\nabla J(\theta) &= \nabla_\theta \Big(\sum_\tau \mathbb{P}(\tau|\pi_\theta) R(\tau) \Big) = \sum_\tau \nabla_\theta \mathbb{P}(\tau|\pi_\theta) R(\tau) \\ &= \sum_\tau \mathbb{P}(\tau|\pi_\theta) \nabla_\theta \log \big(\mathbb{P}(\tau|\pi_\theta)\big) R(\tau) \\ &= \mathbb{E}_{\tau|\pi_\theta}\Big[\nabla_\theta \log \big(\mathbb{P}(\tau|\pi_\theta)\big) R(\tau)\Big]\end{aligned}$$

Policy Gradient Theorem: Rough Idea

Likelihood of a trajectory

$$\mathbb{P}(\tau|\pi_\theta) = \rho(x_1)\prod_{t=1}^{T} p(x_{t_1}|x_t, a_t)\pi_\theta(a_t|x_t)$$

$$\log \mathbb{P}(\tau|\pi_\theta) = \log\big(\rho(x_1)\big) + \sum_{t=1}^{T}\log\big(\pi_\theta(a_t|x_t)\big) + \sum_{t=1}^{T}\log\big(p(x_{t_1}|x_t, a_t)\big)$$

$$\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta) = \cancelto{0}{\nabla_\theta \log\big(\rho(x_1)\big)} + \sum_{t=1}^{T}\nabla_\theta\log\big(\pi_\theta(a_t|x_t)\big) \cancelto{0}{\sum_{t=1}^{T}\nabla_\theta\log\big(p(x_{t_1}|x_t, a_t)\big)}$$

Gradient of J

$$\nabla J(\theta) = \mathbb{E}\Big[\nabla_\theta \log\big(\mathbb{P}(\tau|\pi_\theta)\big)R(\tau)\Big]$$

$$= \mathbb{E}_{\tau|\pi_\theta}\Big[\sum_{t=1}^{T}\nabla_\theta\log\big(\pi_\theta(a_t|x_t)\big)R(\tau)\Big]$$

REINFORCE

Policy gradient theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log\big(\pi_\theta(a|x)\big) Q^{\pi_\theta}(x,a)\Big]$$

REINFORCE algorithm

- For each trajectory $\tau_k = (x_1, a_1, r_1, x_2, \ldots, x_{T-1}, a_{T-1}, r_T) \sim \pi_\theta$
- For $t = 1, \ldots, T$
    - Compute Monte-Carlo estimate

$$R(x_t, a_t) = \sum_{s=t}^{T} r_t$$

    - Update policy

$$\theta = \theta + \alpha \nabla_\theta \log\big(\pi_\theta(a_t|x_t)\big) R(x_t, a_t)$$

Reinforce

Issues

- $R(x,a)$ is a MC (unbiased) estimation of $Q^{\pi_\theta}(x,a)$
- $R(x,a)$ has possibly a very large variance
- $\Rightarrow$ Reinforce needs many samples to converge

REINFORCE

Issues

- $R(x, a)$ is a MC (unbiased) estimation of $Q^{\pi_\theta}(x, a)$
- $R(x, a)$ has possibly a very large variance
- $\Rightarrow$ REINFORCE needs many samples to converge

Possible solutions

- Define an alternative estimator for $Q^{\pi_\theta}(x, a) \Rightarrow$ actor-critic
- Subtract a baseline function to $R(x, a) \Rightarrow$ advantage function

Actor-Critic

Use $\mathrm{TD}(0)$ to estimate Q^{π_θ} using functions Q_w

Actor-Critic algorithm

- For each trajectory $\tau_k = (x_1, a_1, r_1, x_2, \ldots, x_{T-1}, a_{T-1}, r_T) \sim \pi_\theta$
- For $t = 1, \ldots, T$
  - Compute temporal difference

$$\delta_t = r_t + \gamma Q_w(x_{t+1}, a_{t+1})$$

  - Update Q estimate

$$w = w + \beta \delta_t \nabla Q_w(x_t, a_t)$$

  - Update policy

$$\theta = \theta + \alpha \nabla_\theta \log\left(\pi_\theta(a_t|x_t)\right) Q_w(x_t, a_t)$$

Actor-Critic

Issues

- $Q_w(x, a)$ is a biased estimator of $Q^{\pi_\theta}(x, a)$
- The update of θ may not follow the gradient $\nabla_\theta J$ anymore

Actor-Critic

Issues

- $Q_w(x, a)$ is a biased estimator of $Q^{\pi_\theta}(x, a)$
- The update of θ may not follow the gradient $\nabla_\theta J$ anymore

Possible solutions

- Choose the approximation space Q_w “carefully” $\Rightarrow$ compatibility between Q_w and π_θ

Actor-Critic: compatible function approximation

Theorem

An action value function space Q_w is "compatible" with a policy space π_θ if

$$Q_w(x,a) = w^\top \nabla_\theta \log\big(\pi_\theta(a|x)\big).$$

If w is minimizing the squared Bellman residual

$$w = \arg\min_w \mathbb{E}_{\pi_\theta}\Big[\big(Q^{\pi_\theta}(x,a) - Q_w(x,a)\big)^2\Big].$$

Then

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log\big(\pi_\theta(a|x)\big) Q_w(x,a)\Big]$$

Actor-Critic: compatible function approximation

Theorem

An action value function space Q_w is "compatible" with a policy space π_θ if

$$Q_w(x,a) = w^\top \nabla_\theta \log\big(\pi_\theta(a|x)\big).$$

If w is minimizing the squared Bellman residual

$$w = \arg\min_w \mathbb{E}_{\pi_\theta}\Big[\big(Q^{\pi_\theta}(x,a) - Q_w(x,a)\big)^2\Big].$$

Then

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log\big(\pi_\theta(a|x)\big) Q_w(x,a)\Big]$$

$$\Rightarrow \theta = \theta + \alpha \nabla_\theta \log\big(\pi_\theta(a_t|x_t)\big) Q_w(x_t, a_t)$$

Actor-Critic with a baseline

Theorem

Let $b(x)$ an arbitrary baseline function, then

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log\big(\pi_\theta(a|x)\big)\big(Q^{\pi_\theta}(x,a) - b(x)\big)\Big]$$

$\Rightarrow$ use $b(s)$ to reduce the variance of the estimates

Actor-Critic with a baseline

Theorem

Let $b(x)$ an arbitrary baseline function, then

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log\big(\pi_\theta(a|x)\big)\big(Q^{\pi_\theta}(x,a) - b(x)\big)\Big]$$

$\Rightarrow$ use $b(s)$ to reduce the variance of the estimates

$\Rightarrow$ the choice that minimize the variance is V^{π_θ}!

ACTOR-CRITIC with a baseline

Theorem

Let $b(x)$ an arbitrary baseline function, then

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log\big(\pi_\theta(a|x)\big)\big(Q^{\pi_\theta}(x,a) - b(x)\big)\Big]$$

$\Rightarrow$ use $b(s)$ to reduce the variance of the estimates

$\Rightarrow$ the choice that minimize the variance is V^{π_θ}!

$\Rightarrow$ $A^{\pi_\theta}(x,a) = Q^{\pi_\theta}(x,a) - V^{\pi_\theta}(x,a)$ is the advantage function

Actor-Critic with Advantage Function

Use TD(0) to estimate Q^{π_θ} using functions Q_w and V^{π_theta} using functions V_v

Actor-Critic algorithm

- For each trajectory $\tau_k = (x_1, a_1, r_1, x_2, \ldots, x_{T-1}, a_{T-1}, r_T) \sim \pi_\theta$
- For $t = 1, \ldots, T$
  - Compute temporal differences

$$\delta_t^Q = r_t + \gamma Q_w(x_{t+1}, a_{t+1}); \quad \delta_t^V = r_t + \gamma V_v(x_{t+1})$$

  - Update Q and V estimates

$$w = w + \beta \delta_t^Q \nabla Q_w(x_t, a_t); \quad v = v + \eta \delta_t^V \nabla V_v(x_t)$$

  - Update policy

$$\theta = \theta + \alpha \nabla_\theta \log\big(\pi_\theta(a_t|x_t)\big)\big(Q_w(x_t, a_t) - V_v(x_t)\big)$$

Actor-Critic with advantage function

Issues

- $Q_w(x,a) - V_v(x)$ is a very biased and *unstable* estimator of $A^{\pi_\theta}(x,a)$
- The update of θ may be too fast w.r.t. w and v

Actor-Critic with advantage function

Issues

- $Q_w(x,a) - V_v(x)$ is a very biased and *unstable* estimator of $A^{\pi_\theta}(x,a)$
- The update of θ may be too fast w.r.t. w and v

Possible solutions

- Consider the “exact” temporal difference in x, a

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(x') - V^{\pi_\theta}(x)$$

- δ^{π_θ} is an unbiased estimator of the advantage

$$\mathbb{E}\Big[\delta^{\pi_\theta}\Big] = \mathbb{E}\Big[r + \gamma V^{\pi_\theta}(x')\Big|x,a\Big] - V^{\pi_\theta}(x) = Q^{\pi_\theta}(x,a) - V^{\pi_\theta}(x)$$

- $\Rightarrow$ use only the TD(0) estimator

Actor-Critic with Advantage Function and TD(0)

Actor-Critic algorithm

- For each trajectory $\tau_k = (x_1, a_1, r_1, x_2, \dots, x_{T-1}, a_{T-1}, r_T) \sim \pi_\theta$
- For $t = 1, \dots, T$
  - Compute temporal difference

$$\delta_t = r_t + \gamma V_v(x_{t+1})$$

  - Update V estimate

$$v = v + \eta \delta_t^V \nabla V_v(x_t)$$

  - Update policy

$$\theta = \theta + \alpha \nabla_\theta \log\big(\pi_\theta(a_t|x_t)\big)\big(\delta_t - V_v(x_t)\big)$$

Actor-Critic with Advantage Function and TD(0)

Issues

- Properly setting the learning rates η and α is difficult
- All samples need to be generated by the current policy (*on-policy* learning)

ACTOR-CRITIC with Advantage Function and TD(0)

Issues

- Properly setting the learning rates η and α is difficult
- All samples need to be generated by the current policy (*on-policy* learning)

Possible solutions

- Consider a "conservative" optimization algorithm
- Use importance weighting

Conservative Policy Iteration Algorithms

Relationship between *current policy* π and *candidate policy* $\widetilde{\pi}$

$$J(\widetilde{\pi}) = J(\pi) + \sum_{x \in X} \rho_\gamma^{\tilde{\pi}}(x) \sum_a \tilde{\pi}(a|x) A^\pi(x,a)$$

with $\rho_\gamma^{\tilde{\pi}}(x) = \sum_{t=0}^\infty \gamma^t \mathbb{P}_{\tilde{\pi}}\left[x_t = x\right]$ (discounted stationary distribution)

Conservative Policy Iteration Algorithms

Relationship between *current policy* π and *candidate policy* $\widetilde{\pi}$

$$J(\widetilde{\pi}) = J(\pi) + \sum_{x \in X} \rho_\gamma^{\tilde{\pi}}(x) \sum_a \tilde{\pi}(a|x) A^\pi(x,a)$$

with $\rho_\gamma^{\tilde{\pi}}(x) = \sum_{t=0}^\infty \gamma^t \mathbb{P}_{\tilde{\pi}}\big[x_t = x\big]$ (discounted stationary distribution)

Issue: $\rho_\gamma^{\tilde{\pi}}(x)$ is difficult to compute/estimate for any possible $\tilde{\pi}$

Conservative Policy Iteration Algorithms

Surrogate function

$$J(\widetilde{\pi}) = J(\pi) + \sum_{x \in X} \rho_\gamma^{\tilde{\pi}}(x) \sum_a \tilde{\pi}(a|x) A^\pi(x,a)$$

$$L_\pi(\widetilde{\pi}) = J(\pi) + \sum_{x \in X} \rho_\gamma^{\pi}(x) \sum_a \tilde{\pi}(a|x) A^\pi(x,a)$$

Conservative Policy Iteration Algorithms

Surrogate function

$$J(\widetilde{\pi}) = J(\pi) + \sum_{x \in X} \rho_\gamma^{\tilde{\pi}}(x) \sum_a \tilde{\pi}(a|x) A^\pi(x,a)$$

$$L_\pi(\widetilde{\pi}) = J(\pi) + \sum_{x \in X} \rho_\gamma^{\pi}(x) \sum_a \tilde{\pi}(a|x) A^\pi(x,a)$$

Properties

- $L_\pi(\pi) = J(\pi)$
- If parametrized policy $\pi = \pi_\theta$ then $\nabla_\theta L_{\pi_\theta}(\pi_\theta) = \nabla_\theta J(\pi_\theta)$

$\Rightarrow$ In an interval ***close*** to π, L_π is a good surrogate for J

Conservative Policy Iteration Algorithms

Let measure the "distance" between two policies as

$$D_{\mathrm{TV}}^{\max}(\pi, \tilde{\pi}) = \max_x D_{\mathrm{TV}}\big(\pi(\cdot|x) \| \tilde{\pi}(\cdot|s)\big)$$

Then for any two policies π, $\tilde{\pi}$, such tat $D_{\mathrm{TV}}^{\max}(\pi, \tilde{\pi}) = \alpha$ and $\epsilon = \max_{x,a} |A^\pi(x,a)|$

$$J(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$

Conservative Policy Iteration Algorithms

Let measure the "distance" between two policies as

$$D_{\mathrm{TV}}^{\max}(\pi,\tilde{\pi}) = \max_x D_{\mathrm{TV}}\big(\pi(\cdot|x)\|\tilde{\pi}(\cdot|s)\big)$$

Then for any two policies π, $\tilde{\pi}$, such tat $D_{\mathrm{TV}}^{\max}(\pi,\tilde{\pi}) = \alpha$ and $\epsilon = \max_{x,a}|A^{\pi}(x,a)|$

$$J(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$

New policy improvement scheme = conservative policy iteration

$$\max_{\tilde{\pi}} L_{\pi}(\tilde{\pi}) - C D_{\mathrm{TV}}^{\max}(\pi,\tilde{\pi})$$

Conservative Policy Iteration Algorithms

Let measure the "distance" between two policies as

$$D_{\text{TV}}^{\max}(\pi, \tilde{\pi}) = \max_x D_{\text{TV}}\big(\pi(\cdot|x) \| \tilde{\pi}(\cdot|s)\big)$$

Then for any two policies π, $\tilde{\pi}$, such tat $D_{\text{TV}}^{\max}(\pi, \tilde{\pi}) = \alpha$ and $\epsilon = \max_{x,a} |A^\pi(x,a)|$

$$J(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$

New policy improvement scheme = conservative policy iteration

$$\max_{\tilde{\pi}} L_\pi(\tilde{\pi}) - C D_{\text{TV}}^{\max}(\pi, \tilde{\pi})$$

$\Rightarrow$ difficult to optimize...

Conservative Policy Iteration Algorithms

Alternative measure of distance

$$D^{\rho}_{\mathrm{KL}}(\pi,\tilde{\pi}) = \mathbb{E}_{x\sim\rho}\big[D_{\mathrm{KL}}\big(\pi(\cdot|x)\|\tilde{\pi}(\cdot|s)\big)\big]$$

Alternative policy improvement scheme (regularized version)

$$\max_{\tilde{\pi}} L_{\pi}(\tilde{\pi}) - C D^{\rho^{\pi}_{\gamma}}_{\mathrm{KL}}(\pi,\tilde{\pi})$$

Alternative policy improvement scheme (constrained version)

$$\max_{\tilde{\pi}} L_{\pi}(\tilde{\pi})$$
$$s.t. \quad D^{\rho^{\pi}_{\gamma}}_{\mathrm{KL}}(\pi,\tilde{\pi}) \leq \delta$$

Conservative Policy Iteration Algorithms

Towards an actual algorithm (1)

- Importance weighting with a sampling distribution $q(a|x)$

$$\sum_a \tilde{\pi}(a|x) A^\pi(x,a) \Rightarrow \sum_a q(a|x)\frac{\tilde{\pi}(a|x)}{q(a|x)} A^\pi(x,a) = \mathbb{E}_{q(\cdot|x)}\left[\frac{\tilde{\pi}(a|x)}{q(a|x)} A^\pi(x,a)\right]$$

- Replace A^π with Q^π and remove $J(\pi)$ (constant shifts)

$$\max_{\tilde{\pi}} \mathbb{E}_{x\sim\rho_\gamma^\pi} \mathbb{E}_{a\sim q(\cdot|x)}\left[\frac{\tilde{\pi}(a|x)}{q(a|x)} Q^\pi(x,a)\right]$$
$$s.t. \quad D_{\mathrm{KL}}^{\rho_\gamma^\pi}(\pi,\tilde{\pi}) \le \delta$$

Towards an actual algorithm (2)

- Estimate $\mathbb{E}$ by executing π and q
- Estimate Q^π by rollouts

$\Rightarrow$ Trust region policy optimization (TRPO)

Summary

Policy gradient methods are ***successful*** because

- Easy to integrate a NN architecture into the scheme
- Effective in simulation environments (large amount of rollouts can be generated)
- A lot of "tricks" from optimization can be integrated

Policy gradient methods are ***difficult*** because

- Stochastic policies may not be desirable
- No convergence guarantees
- A zoo of more or less explicit / heuristic variants

Bibliography I

Reinforcement Learning

Alessandro Lazaric
alessandro.lazaric@inria.fr
sequel.lille.inria.fr