facebook
Artificial Intelligence Research

How to solve an MDP incrementally: Approximate algorithms

Alessandro Lazaric
Facebook AI Research

Learning in an MDP

Goal: learning the optimal policy $\pi^\star$ of an MDP

- Tabular MDP, known dynamics
- Tabular MDP, unknown dynamics
- Large or Continuous MDP, known dynamics
- Large or Continuous MDP, unknown dynamics

Learning in an MDP

Goal: learning the optimal policy $\pi^\star$ of an MDP

- Tabular MDP, known dynamics
  Dynamic Programming
- Tabular MDP, unknown dynamics
- Large or Continuous MDP, known dynamics
- Large or Continuous MDP, unknown dynamics

Learning in an MDP

Goal: learning the optimal policy $\pi^\star$ of an MDP

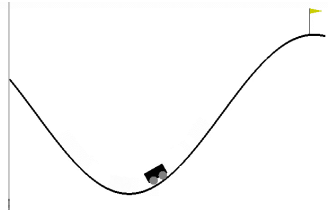
- Tabular MDP, known dynamics
  Dynamic Programming
- Tabular MDP, unknown dynamics
  Q-Learning, SARSA
- Large or Continuous MDP, known dynamics
- Large or Continuous MDP, unknown dynamics

Learning in an MDP

Goal: learning the optimal policy $\pi^\star$ of an MDP

- Tabular MDP, known dynamics
  Dynamic Programming
- Tabular MDP, unknown dynamics
  Q-Learning, SARSA
- Large or Continuous MDP, known dynamics
  ?
- Large or Continuous MDP, unknown dynamics
  ?

Example: Mountain Car



State : $(x, \dot{x}) \in [-1.2; 0.6] \times [-0.07; 0.07]$

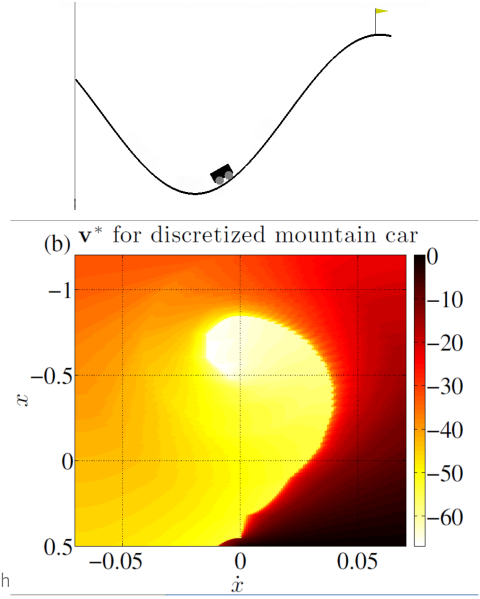
Actions : $\mathcal{A} = \{-1, 0, 1\}$: full throttle reverse / zero throttle / full throttle forward

Reward : always -1 except in the terminal (goal) state $x_\star = 0.6$

Dynamics : when doing action a_t in state $s_t = (x_t, v_t)$, the next state $s_{t+1} = (x_{t+1}, v_{t+1})$ is

$$\begin{cases} v_{t+1} &= \max\{\min\{v_t + \epsilon_t + 0.001 a_t - 0.0025\cos(3x_t), 0.07\}, -0.07\}, \\ x_{t+1} &= \max\{\min\{x_t + v_t, 0.6\}, -1.2\}. \end{cases}$$

Example: Mountain Car



What to approximate?

- Value Function

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s\right]$$

$$Q^{\pi}(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a\right]$$

- Policy

$$\pi : \mathcal{S} \to \Delta(\mathcal{A})$$

How? Value function approximation

From an estimate of $V^\star$ to an estimate of $Q^\star$

$$Q^\star \rightarrow V^\star(s) = \max_a Q^\star(s,a) \qquad \text{easy}$$

$$V^\star \rightarrow Q^\star(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^\star(s') \qquad \text{possibly complicated}$$

Policy Computation

$$\pi(s) = \arg\max_a Q(s,a)$$

$$\pi(s) = \arg\max_a r(s,a) + \gamma \sum_{s'} p(s'|s,a) V^\star(s')$$

☞ decide when to approximate $V^\star$ or $Q^\star$
($Q^\star$ **is more handy to get a policy, but more parameter to learn**)

How? Value function approximation

Problem: Often $\mathcal{S}$ is too large to store a vector V or a table Q in memory...

Solution: look for estimates V (resp. Q) of $V^\star$ (resp. $Q^\star$) in an approximation space $\mathcal{F}_V$ (resp. $\mathcal{F}_Q$)

Parametric approximation

$$\mathcal{F}_V = \{s \mapsto V_\theta(s) | \theta \in \Theta\} \qquad \mathcal{F}_Q = \{(s,a) \mapsto Q_\theta(s,a) | \theta \in \Theta\}$$

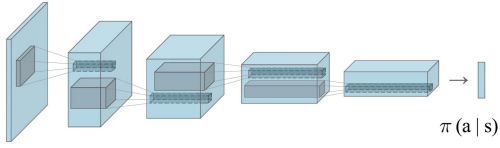
only requires to store a parameter θ (typically $\theta \in \mathbb{R}^d$, $d << S$)

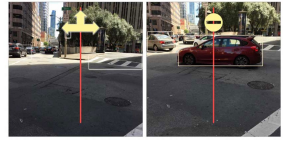
☞ Smooth parameterization if $\nabla_\theta V_\theta(s)$ (resp. $\nabla_\theta Q_\theta(s,a)$) can be computed

How? Policy approximation

$$\mathcal{F}_{\Pi} = \left\{ (s,a) \mapsto \pi_\theta(a|s) \mid \theta \in \Theta \right\}$$

- deterministic vs. stochastic policy
- discrete actions vs. continuous actions





How? Policy approximation

Normal Policy

$$\pi(a|s) = \frac{1}{\sigma_\omega(s)\sqrt{2\pi}} e^{-\frac{(a-\mu_\theta(s))^2}{2\sigma_\omega^2(s)}}$$

with

$$\nabla_\theta \log \pi(a|s) = \frac{(a-\mu_\theta(s))}{\sigma_\omega^2(s)} \nabla_\theta \mu_\theta(s), \qquad \nabla_\omega \log \pi(a|s) \;\; = \frac{(a-\mu_\theta(s))^2 - \sigma_\omega^2(s)}{\sigma_\omega^3(s)} \nabla_\omega \sigma_\omega(s)$$

Softmax Policy (κ inverse temperature)

$$\pi(a|s) = \frac{e^{\kappa Q_\theta(s,a)}}{\sum_{a' \in \mathcal{A}} e^{\kappa Q_\theta(s,a')}}$$

with

$$\nabla_\theta \log \pi(a|s) = \kappa \nabla_\theta Q_\theta(s,a) - \kappa \sum_{a' \in \mathcal{A}} \pi(a'|s) \nabla_\theta Q_\theta(s,a')$$

Outline

Approximate TD As *Pseudo*-Gradient Descent

- Run π over a single trajectory $(s_0, r_0, s_1, r_1, s_2, r_2, \ldots, s_n, r_n)$
- TD loss using **bootstrapped** target

$$\widetilde{L}(s_t, \widetilde{R}_t; \theta) = (V_\theta(s_t) - \widetilde{R}_t)^2 = (V_\theta(s_t) - r_t - \gamma V_{\theta_t}(s_{t+1}))^2$$

- TD *online* update with learning rate α_t

$$\begin{aligned}\widehat{\theta}_{t+1} &= \widehat{\theta}_t - \alpha_t \nabla_\theta \widetilde{L}(s_t, \widetilde{R}_t; \theta_t) \\ &= \widehat{\theta}_t - \alpha_t \big(V_{\theta_t}(s_t) - r_t - \gamma V_{\theta_t}(s_{t+1})\big) \nabla_\theta V_{\theta_t}(s_t)\end{aligned}$$

Approximate TD As *Pseudo*-Gradient Descent

- Run π over a single trajectory $(s_0, r_0, s_1, r_1, s_2, r_2, \ldots, s_n, r_n)$
- TD loss using **bootstrapped** target

$$\widetilde{L}(s_t, \widetilde{R}_t; \theta) = (V_\theta(s_t) - \widetilde{R}_t)^2 = (V_\theta(s_t) - r_t - \gamma V_{\theta_t}(s_{t+1}))^2$$

- TD *online* update with learning rate α_t

$$\begin{aligned}\widehat{\theta}_{t+1} &= \widehat{\theta}_t - \alpha_t \nabla_\theta \widetilde{L}(s_t, \widetilde{R}_t; \theta_t) \\ &= \widehat{\theta}_t - \alpha_t \big(V_{\theta_t}(s_t) - r_t - \gamma V_{\theta_t}(s_{t+1})\big) \nabla_\theta V_{\theta_t}(s_t)\end{aligned}$$

👎 Not really a gradient method...

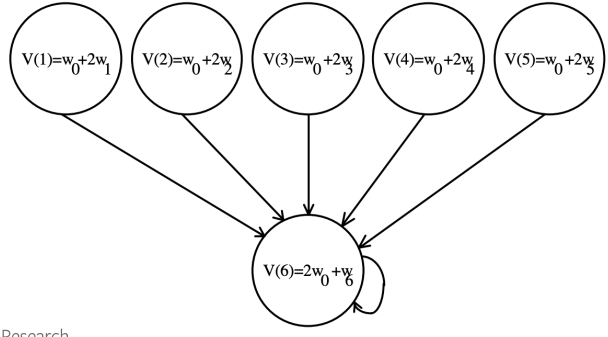
Approximate TD

Approximate TD **converges** if

- Linear approximation and states s_i are obtained by following the policy under evaluation *(on-policy learning)*

Approximate TD **may not converge** (i.e., it might diverge) if

- Linear approximation but states s_i are obtained by following a different policy *(off-policy learning)*
- *Non-linear approximation* and states s_i are obtained by following π



Approximate QL As *Pseudo*-Gradient Descent

- Run π over a single trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_n, a_n, r_n)$
- QL loss using **bootstrapped** target

$$\widetilde{L}(s_t, a_t, \widetilde{R}_t; \theta) = (Q_\theta(s_t, a_t) - \widetilde{R}_t)^2 = \big(Q_\theta(s_t, a_t) \underbrace{- r_t - \gamma \max_{a'} Q_{\theta_t}(s_{t+1}, a')}_{\text{target}}\big)^2$$

- QL *online* update with learning rate α_t

$$\begin{aligned}\widehat{\theta}_{t+1} &= \widehat{\theta}_t - \alpha_t \nabla_\theta \widetilde{L}(s_t, a_t, \widetilde{R}_t; \theta_t) \\ &= \widehat{\theta}_t - \alpha_t \big(Q_{\theta_t}(s_t, a_t) - r_t - \gamma \max_{a'} Q_{\theta_t}(s_{t+1}, a')\big) \nabla_\theta Q_{\theta_t}(s_t, a_t)\end{aligned}$$

Online Approximate Q-Learning

Input: T, s_0, Q_0
$Q = Q_0$, $s = s_0$
for $t = 1, \dots, T$ **do**
  Execute action $a_t \sim \pi_t(s_t)$
  Observe r_t and next state s_{t+1}
  Update

$$\widehat{\theta}_{t+1} = \widehat{\theta}_t - \alpha_t \big(Q_{\theta_t}(s_t, a_t) - r_t - \gamma \max_{a'} Q_{\theta_t}(s_{t+1}, a') \big) \nabla_\theta Q_{\theta_t}(s_t, a_t)$$

end
return $\widehat{Q}$, greedy($\widehat{Q}$)

Online Approximate Q-Learning

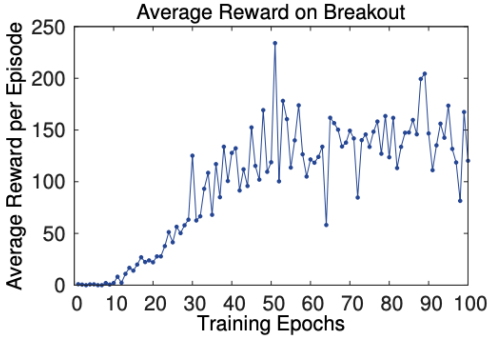
Input: T, s_0, Q_0
$Q = Q_0$, $s = s_0$
for $t = 1, \dots, T$ **do**
  Execute action $a_t \sim \pi_t(s_t)$
  Observe r_t and next state s_{t+1}
  Update

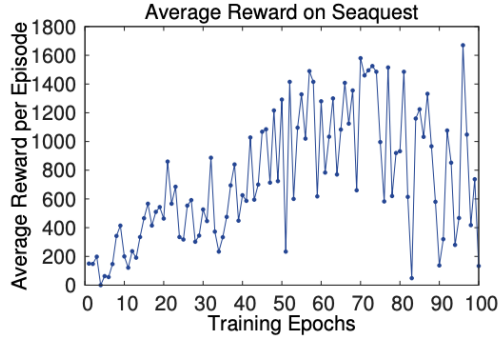
$$\widehat{\theta}_{t+1} = \widehat{\theta}_t - \alpha_t \big(Q_{\theta_t}(s_t, a_t) - r_t - \gamma \max_{a'} Q_{\theta_t}(s_{t+1}, a')\big) \nabla_\theta Q_{\theta_t}(s_t, a_t)$$

end
return $\widehat{Q}$, greedy($\widehat{Q}$)

👎 It may diverge even with linear function approximation...

Approximate QL As *Pseudo*-Gradient Descent





Towards DQN

Practical challenges in making approximate QL "more" stable

- Sequential updates $\Rightarrow$ *correlated samples*
- From Q-values to policy, from policy to Q-values, ... $\Rightarrow$ *oscillations*
- Scale of Q-values unknown $\Rightarrow$ *gradients with different scales*
- QL update using $\max_{a'} Q(s, a') \Rightarrow$ *over-estimation*

Towards DQN

Practical challenges in making approximate QL "more" stable

- Sequential updates $\Rightarrow$ *correlated samples*
  $\Rightarrow$ *experience replay*
- From Q-values to policy, from policy to Q-values, ... $\Rightarrow$ *oscillations*
- Scale of Q-values unknown $\Rightarrow$ *gradients with different scales*
- QL update using $\max_{a'} Q(s, a')$ $\Rightarrow$ *over-estimation*

Towards DQN

Practical challenges in making approximate QL "more" stable

- Sequential updates $\Rightarrow$ *correlated samples*
  $\Rightarrow$ *experience replay*
- From Q-values to policy, from policy to Q-values, ... $\Rightarrow$ *oscillations*
  $\Rightarrow$ *target network*
- Scale of Q-values unknown $\Rightarrow$ *gradients with different scales*
- QL update using $\max_{a'} Q(s, a')$ $\Rightarrow$ *over-estimation*

Towards DQN

Practical challenges in making approximate QL "more" stable

- Sequential updates $\Rightarrow$ *correlated samples*
  $\Rightarrow$ *experience replay*
- From Q-values to policy, from policy to Q-values, ... $\Rightarrow$ *oscillations*
  $\Rightarrow$ *target network*
- Scale of Q-values unknown $\Rightarrow$ *gradients with different scales*
  $\Rightarrow$ *reward normalization*
- QL update using $\max_{a'} Q(s, a') \Rightarrow$ *over-estimation*

Towards DQN

Practical challenges in making approximate QL "more" stable

- Sequential updates $\Rightarrow$ *correlated samples*
  $\Rightarrow$ *experience replay*
- From Q-values to policy, from policy to Q-values, ... $\Rightarrow$ *oscillations*
  $\Rightarrow$ *target network*
- Scale of Q-values unknown $\Rightarrow$ *gradients with different scales*
  $\Rightarrow$ *reward normalization*
- QL update using $\max_{a'} Q(s, a')$ $\Rightarrow$ *over-estimation*
  $\Rightarrow$ *double Q-learning*

Experience Replay

- To help remove correlations, store dataset $\mathcal{D}$ from prior experience
- QL online with *replay buffer*
  - Sample experience from the dataset

$$(s, a, r, s') \sim \mathcal{D}$$

  - Online update

$$\widehat{\theta}_{t+1} = \widehat{\theta}_t - \alpha_t \big(Q_{\theta_t}(s, a) \underbrace{- r - \gamma \max_{a'} Q_{\theta_t}(s', a')}_{\text{target}} \big) \nabla_\theta Q_{\theta_t}(s, a)$$

  - Execute policy (e.g., ϵ-greedy or softmax)
  - Add new sample to dataset

Target Network

Issue: weights are updated and the target changes $\implies$ non-stationarity

- To help improve stability, fix the target weights used in the target calculation for multiple updates
- Target network uses a different set of weights than the weights being updated
- Let $\overline{\theta}$ be the parameters of the target network

Target Network

- QL online with *replay buffer* and *target network*
  - Sample experience from the dataset

$$(s, a, r, s') \sim \mathcal{D}$$

  - Compute target

$$y_t = r + \gamma \max_{a'} Q_{\overline{\theta}}(s', a')$$

  - Online update

$$\widehat{\theta}_{t+1} = \widehat{\theta}_t - \alpha_t \big(Q_{\theta_t}(s, a) - y_t\big) \nabla_\theta Q_{\theta_t}(s, a)$$

  - Execute policy (e.g., ϵ-greedy or softmax)
  - Add new sample to dataset
- *Update target network* $\overline{\theta}$ every C steps

* it is possible to do also a smooth update of the target network $\overline{\theta} = \tau\overline{\theta} + (1 - \tau)\theta_t$ with $\tau \approx 1$. Less used than full updates.

Mini-batch Update

Issue: online update is inefficient with modern tools (e.g., NN)

Perform update on a *mini-batch* $\mathcal{D}_{\text{mini}}$ sampled from $\mathcal{D}$

- Let $\overline{\theta}$ the *target function*
- Mini-batch loss

$$\widetilde{L}_{\mathcal{D}_{\text{mini}}}(\theta) = \mathbb{E}_{(s_i,a_i,s_{i+1},r_i)\sim\mathcal{D}}\left[\left(Q_\theta(s_i,a_i) - r_i - \gamma \max_{a'} Q_{\overline{\theta}}(s_{i+1},a')\right)^2\right]$$

- Update θ using SGD on $\widetilde{L}_{\mathcal{D}_{\text{mini}}}(\theta)$

Mini-Batch Update

- Sample m transitions from replay buffer $\mathcal{D}$

$$\Lambda_t = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^{m}$$

- Compute *loss*

$$L(\theta|\Lambda_t, \overline{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\left(Q_\theta(s,a) - r_i - \gamma \max_{a'} Q_{\overline{\theta}}(s'_i, a')\right)^2$$

- Update by SGD

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta L(\theta|\Lambda_t, \overline{\theta})$$

DQN

Input: T, s_0, Q_0
$Q = Q_0$, $s = s_0$
for $t = 1, \ldots, T$ **do**
  Execute action $a_t \sim \pi_t(s_t)$
  Observe r_t and next state s_{t+1}
  Store transition $(s_{t,i}, a_{t,i}, s_{t+1,i}, r_{t,i})$ into an experience replay buffer $\mathcal{D}$
  Perform update *of* θ on a mini-batch $\mathcal{D}_{\text{mini}}$ sampled from $\mathcal{D}$ *using target* $\overline{\theta}$

$$\widehat{\theta} = \widehat{\theta} - \alpha \frac{1}{m} \sum_{\tau=1}^{m} \left(Q_\theta(s_\tau, a_\tau) - r_\tau - \gamma \max_{a'} Q_{\overline{\theta}}(s_{\tau+1}, a') \right) \nabla_\theta Q_\theta(s_\tau, a_\tau)$$

  Every C **steps** $\overline{\theta} \leftarrow \theta$
end
return $\widehat{Q}$, greedy$(\widehat{Q})$

DQN – Atari

Image preprocessing: grey-scale, crop to 84x84

$$\phi\left(\quad\right)$$

DQN – Atari

State definition

$$s_t = \left\{ \text{  } \; a_{t-1}, r_{t-1}, \; \dots \; a_{t-4}, r_{t-4} \right\}$$

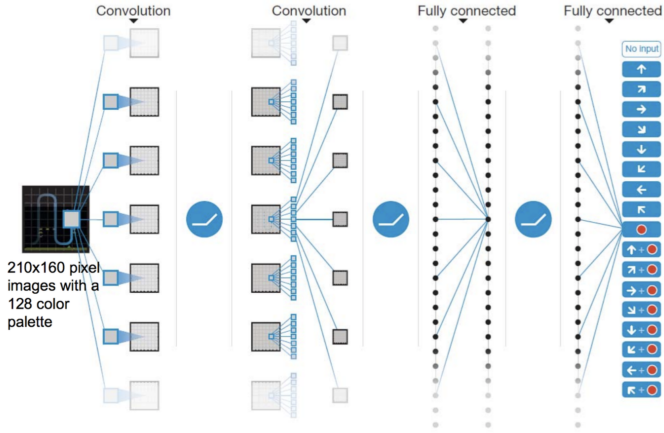
DQN – Atari

Time definition: 4 last frames

$$s_t = \left\{ \text{  } \; a_{t-1}, r_{t-1}, \; \dots \; a_{t-4}, r_{t-4} \right\}$$

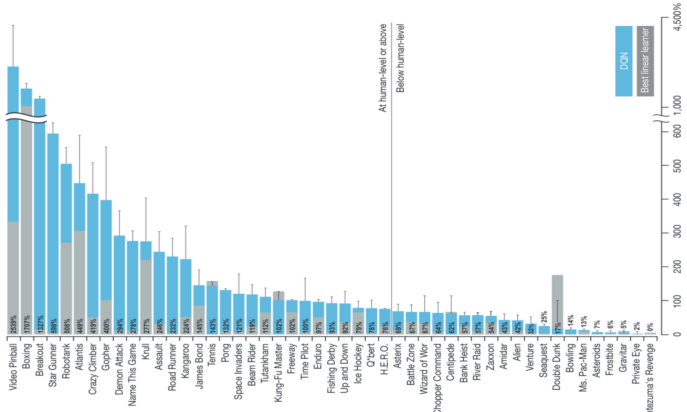
DQN – Atari

Action-value function: deepNet with as many heads as actions



DQN – Atari

Performance



DQN – Atari

Ablation

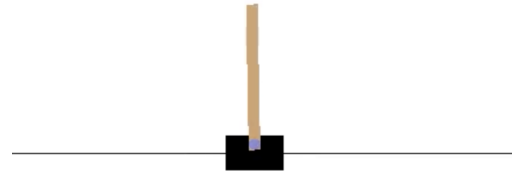
DQN

Game	With replay, with target Q	With replay, without target Q	Without replay, with target Q	Without replay, without target Q
Breakout	316.8	240.7	10.2	3.2
Enduro	1006.3	831.4	141.9	29.1
River Raid	7446.6	4102.8	2867.7	1453.0
Seaquest	2894.4	822.6	1003.0	275.8
Space Invaders	1088.9	826.3	373.2	302.0

DQN

Hands-on session [link] – [20min]

- How does the exploration parameter influence the performance?



Outline

Policy Gradient: Finite-Horizon*

Given an MDP $M = (\mathcal{S}, \mathcal{A}, p, r, H, \rho)$ and a policy π

$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{H} r_t | \pi, M\right] = \mathbb{E}_{\tau \sim \mathbb{P}(\tau|\pi,M)}\left[\mathcal{R}(\tau)\right]$$

where $\tau = (s_1, a_1, r_1, \ldots, s_{H+1})$ is a trajectory and $R(\tau)$ its return (sum of returns).

Policy Gradient: Finite-Horizon*

Given an MDP $M = (\mathcal{S}, \mathcal{A}, p, r, H, \rho)$ and a policy π

$$J(\pi) = \mathbb{E}\left[\sum_{t=1}^{H} r_t | \pi, M\right] = \mathbb{E}_{\tau \sim \mathbb{P}(\tau|\pi,M)}\left[\mathcal{R}(\tau)\right]$$

where $\tau = (s_1, a_1, r_1, \ldots, s_{H+1})$ is a trajectory and $R(\tau)$ its return (sum of returns).

Stochastic policy $\pi : \mathcal{S} \to \mathcal{D}(\mathcal{A})$

*everything extends to infinite-horizon discounted

Policy Gradient as Policy Update

Approximate Policy Iteration

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

Unstable (fast)

Policy Gradient

$$\theta_{k+1} = \theta_k + \alpha_k \nabla J(\theta_k)$$

Smooth, fine control (slow)

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

Policy Gradient Theorem

Theorem

For any finite-horizon MDP $M = (\mathcal{S}, \mathcal{A}, p, r, H, \rho)$ *and differentiable policy* π_θ

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathbb{P}(\cdot|\pi, M)} \left[R(\tau) \sum_{t=1}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

REINFORCE

1. Let π_{θ_1} be an arbitrary policy
2. At each iteration $k = 1, \dots, K$
   - Sample m trajectory $\tau_i = (s_1, a_1, r_1, s_2, \dots, s_T, a_T, r_T, s_{T+1})$ following π_k
   - Compute unbiased gradient estimate

$$\widehat{\nabla_\theta J}(\pi_{\theta_k}) = \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=1}^{H} r_t^i \right) \left(\sum_{t=1}^{H} \nabla_\theta \log \pi_{\theta_k}(s_t, a_t) \right)$$

   - Update parameters

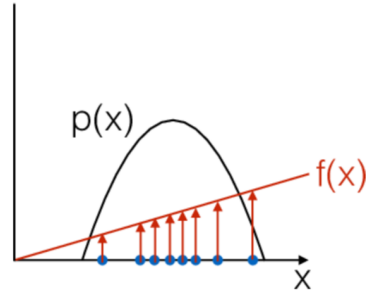
$$\theta_{k+1} = \theta_k + \alpha_k \widehat{\nabla_\theta J}(\pi_{\theta_k})$$

3. Return last policy π_{θ_K}

REINFORCE as *Supervised Learning*

$$\widehat{g}_i = R(\tau_i)\nabla_\theta \log \mathbb{P}(\tau_i|\pi_\theta, M)$$

- $R(\tau_i)$ measures how *good* is sample τ_i
- Moving in the direction of $\widehat{g}_i$ pushes up the log probability of the sample, in proportion to how good it is

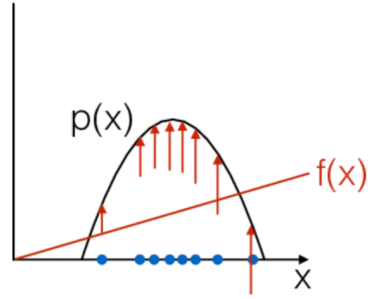


(?)

REINFORCE as *Supervised Learning*

$$\widehat{g}_i = R(\tau_i)\nabla_\theta \log \mathbb{P}(\tau_i|\pi_\theta, M)$$

- $R(\tau_i)$ measures how *good* is sample τ_i
- Moving in the direction of $\widehat{g}_i$ pushes up the log probability of the sample, in proportion to how good it is

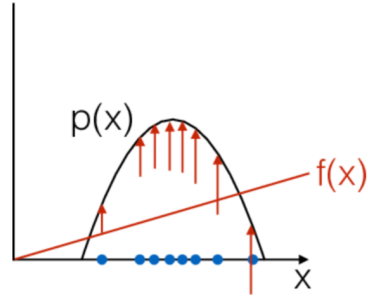


(?)

REINFORCE as *Supervised Learning*

$$\widehat{g}_i = R(\tau_i)\nabla_\theta \log \mathbb{P}(\tau_i|\pi_\theta, M)$$

- $R(\tau_i)$ measures how *good* is sample τ_i
- Moving in the direction of $\widehat{g}_i$ pushes up the log probability of the sample, in proportion to how good it is



(?)

Interpretation: uses good trajectories as supervised examples

- *Like maximum likelihood* in supervised learning
- good stuff are made more likely while bad less
- Trial and Error approach

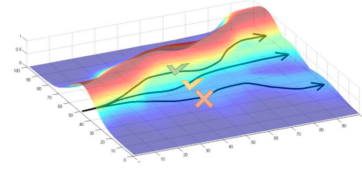


image from "CS 294-112: Deep Reinforcement Learning" slides by S. Levine

REINFORCE

Pros

- Easy to compute
- *Does not use Markov property!*
- Can be used in partially observable MDPs without modification

REINFORCE

Pros

- Easy to compute
- *Does not use Markov property!*
- Can be used in partially observable MDPs without modification

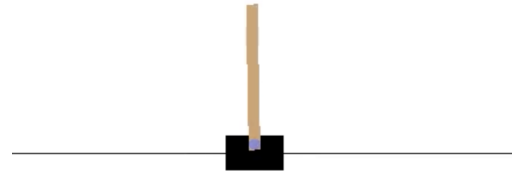
Issues

- Use an MC estimate of $q(s, a)$
- It has possibly a *very large variance*
- Needs many samples to converge

REINFORCE

Hands-on session [link] – [20min]

- How does the exploration parameter influence the performance?



Outline

Policy Gradient Theorem

Theorem

For an infinite horizon MDP (average or discounted), the policy gradient is

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}} \mathbb{E}_{a\sim\pi_\theta(s,\cdot)} \left[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right]$$

- d^π is the stationary distribution
- Q^π is the state-action value function

SARSA

Input: T, s_0, Q_0
$Q = Q_0$, $s = s_0$, $a = \mathcal{U}(\mathcal{A})$
for $t = 1, \dots, T$ **do**
  Execute action $a_t \sim \pi_{t,Q}(\cdot|s_t)$
  Observe r_t and next state s_{t+1}
  $a_{t+1} \sim \pi_{t,Q}(\cdot|s_{t+1})$
  Update

$$\widehat{Q}(s_t, a_t) = \widehat{Q}(s_t, a_t) + \alpha(s_t)\big(r_t + \gamma \widehat{Q}(s_{t+1}, a_{t+1}) - \widehat{Q}(s_t, a_t)\big)$$

end
return $\widehat{Q}$, π_Q

Examples

- $\pi_{t,Q}(a|s) = \dfrac{\exp(Q(s,a)/\tau_t}{\sum_b \exp(Q(s,b)/\tau_t)}$
- $\pi_{t,Q}(a|s) = (1-\epsilon_t)\mathbb{1}\left(a_t = \arg\max_b Q(s,b)\right) + \epsilon_t/A$

SARSA

Input: T, s_0, Q_0
$Q = Q_0$, $s = s_0$, $a = \mathcal{U}(\mathcal{A})$
for $t = 1, \dots, T$ **do**
  Execute action $a_t \sim \pi_{t,Q}(\cdot|s_t)$
  Observe r_t and next state s_{t+1}
  $a_{t+1} \sim \pi_{t,Q}(\cdot|s_{t+1})$
  Update

$$\widehat{Q}(s_t, a_t) = \widehat{Q}(s_t, a_t) + \alpha(s_t)\big(r_t + \gamma \widehat{Q}(s_{t+1}, a_{t+1}) - \widehat{Q}(s_t, a_t)\big)$$

  Update

$$\pi_{t,Q}(a|s) = \frac{\exp(\widehat{Q}(s,a)/\tau_t}{\sum_b \exp(\widehat{Q}(s,b)/\tau_t)}$$

end
return $\widehat{Q}$, π_Q

Actor-Critic

Actor-critic algorithms maintain two sets of parameters:

- Policy parameters $\theta \mapsto \pi$
- Action-value function parameters $\omega \mapsto q^{\pi}$

and use two different algorithms to update them

- Policy gradient to update θ
- TD/Sarsa to update ω

Actor-Critic

for $t = 1, \dots, T$ **do**

$a_t \sim \pi_\theta(s_t, \cdot)$ and observer r_t and s_{t+1}

Compute temporal difference

$$\delta_t = r_t + \gamma q_\omega(s_{t+1}, a_{t+1}) - q_\omega(s_t, a_t)$$

Update action-value

$$\omega = \omega + \beta \delta_t \nabla_\omega q_\omega(x_t, a_t)$$

Update policy

$$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) q_\omega(s_t, a_t)$$

end

Actor-Critic

Issues:

- $q_\omega(s, a)$ is a biased estimate of $Q^{\pi_\theta}(s, a)$
- The update of θ may not follow the gradient of $\nabla_\theta J(\pi_\theta)$

Solution:

- Choose the approximation space $q_\omega(s, a)$ carefully
  $\Longrightarrow$ ***compatible function*** *approximation between* q_ω *and* π_θ

Compatible Function Approximation

Theorem

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

$$q_\omega(s,a) = \omega^\mathsf{T} \nabla_\theta \log \pi_\theta(s,a)$$

If ω minimizes the squared Bellman residual

$$\omega = \arg\min_\omega \mathbb{E}_{s\sim d^{\pi_\theta}} \left[\sum_a \pi_\theta(s,a)(Q^{\pi_\theta}(s,a) - q_\omega(s,a))^2 \right]$$

Then

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}} \mathbb{E}_{a\sim\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s,a) q_\omega(s,a) \right]$$

Actor-Critic with a baseline

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}}\left[\sum_a \nabla_\theta \log(\pi_\theta(s,a))(Q^{\pi_\theta}(s,a) - b(s))\right]$$

- $b(s)$ minimizes the variance
- $V^\pi(s)$ is a good choice as baseline
  - it *minimizes the variance* in average reward
- $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ is the advantage function

Actor-Critic with advantage function (A2C)

- It is possible to estimate V^π and Q^π *independently* (e.g., by TD(0))

Actor-Critic with advantage function (A2C)

- It is possible to estimate V^π and Q^π *independently* (e.g., by TD(0))
- $A^\pi = Q_\omega - V_\nu$ is a **biased** and **unstable** estimate

Actor-Critic with advantage function (A2C)

- It is possible to estimate V^π and Q^π *independently* (e.g., by TD(0))
- $A^\pi = Q_\omega - V_\nu$ is a **biased** and **unstable** estimate

Solution:

- Consider the temporal difference error

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

Actor-Critic with advantage function (A2C)

- It is possible to estimate V^π and Q^π *independently* (e.g., by TD(0))
- $A^\pi = Q_\omega - V_\nu$ is a **biased** and **unstable** estimate

Solution:

- Consider the temporal difference error

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

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

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

Actor-Critic with advantage function (A2C)

for $t = 1, \ldots, T$ **do**

$a_t \sim \pi_\theta(s_t, \cdot)$ and observer r_t and s_{t+1}

Compute temporal difference

$$\delta_t = r_t + \gamma v_\nu(s_{t+1}) - v_\nu(s_t)$$

Update v estimate

$$\nu = \omega + \beta \delta_t \nabla_\nu v_\nu(s_t)$$

Update policy

$$\theta = \theta + \alpha \delta_t \nabla_\theta \log \pi_\theta(s_t, a_t)$$

end

From online to batch actor-critic

- So far we have observed fully online actor-critic approaches
  - The policy is updated at each step
- In some case it can be *inefficient* (e.g., for training approximators)

  $\Longrightarrow$ *batching* as in supervised learning

From online to batch actor-critic

- So far we have observed fully online actor-critic approaches
  - The policy is updated at each step
- In some case it can be *inefficient* (e.g., for training approximators)

$\Longrightarrow$ *batching* as in supervised learning

— Batched Policy Evaluation —

1. Sample m trajectories $\tau_i = \{s_1, a_1, r_1, \ldots, s_{T+1}\}$ using π_θ

$$\hat{v}(s_{i,t}) = \sum_{k=t}^{t+p} \gamma^{k-t} r_{i,k} + \gamma^p v_\nu(s_{i,t+p+1}) \qquad \textbf{bootstrapping}$$

From online to batch actor-critic

- So far we have observed fully online actor-critic approaches
  - The policy is updated at each step
- In some case it can be *inefficient* (e.g., for training approximators)

$\Longrightarrow$ *batching* as in supervised learning

— Batched Policy Evaluation —

1. Sample m trajectories $\tau_i = \{s_1, a_1, r_1, \ldots, s_{T+1}\}$ using π_θ

$$\hat{v}(s_{i,t}) = \sum_{k=t}^{t+p} \gamma^{k-t} r_{i,k} + \gamma^p v_\nu(s_{i,t+p+1}) \qquad \textbf{bootstrapping}$$

2. Use supervised regression on $D = \{(s_{i,t}, \hat{v}(s_{i,t}))\}$, for all i, t

$$\arg\min_\nu \frac{1}{2} \sum_{(s,\hat{v}) \in D} (v_\nu(s) - \hat{v})^2$$

From online to batch actor-critic

— Batched Policy Update —

1. Sample m trajectories $\tau_i = \{s_1, a_1, r_1, \ldots, s_{T+1}\}$ using π_θ
2. Compute an estimate of the gradient

$$\widehat{g} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T_i} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t}) \delta_{i,t}$$

where $\delta_{i,t} = r_{i,t} + \gamma v_\nu(s_{i,t+1}) - v_\nu(s_{i,t})$

Batched A2C

for $k = 1, 2, \dots$ **do**

Generate m trajectories (τ_i) using policy π_{θ_k}

Update v (usually $p = 1$)

$$\hat{v}(s_{i,t}) = \sum_{k=t}^{t+p} \gamma^{k-t} r_{i,k} + \gamma^p v_\nu(s_{i,t+p+1}), \quad \nu_k = \arg\min_\nu \frac{1}{2} \sum_{(s,\hat{v}) \in D} \left(v_\nu(s) - \hat{v}\right)^2$$

with $\mathcal{D} = (s_{i,t}, \hat{v}(s_{i,t}))_{i,t}$

Update policy

$$\delta_{i,t} = r_{i,t} + \gamma v_{\nu_k}(s_{i,t+1}) - v_{\nu_k}(s_{i,t}), \quad \widehat{g} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{T_i} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t}) \delta_{i,t}$$

$$\theta_{k+1} = \theta_{k+1} + \alpha \widehat{g}$$

end

Sample Efficiency in Actor-Critic

Issues:

- Sample efficiency is pretty poor
- All samples need to be generated by the current policy (*on-policy learning*)
- Samples are *discarded* after a single update

Sample Efficiency in Actor-Critic

Issues:

- Sample efficiency is pretty poor
- All samples need to be generated by the current policy (*on-policy learning*)
- Samples are *discarded* after a single update

Solutions

- Use samples from other policies via *importance sampling* (*not very stable*)
- *Conservative approaches*
- Variance reduction techniques
- Newton or Quasi-newton methods

Outline

Relative Performance

Issues:

- We would like to exploit past samples
- We do not know how much to trust them
- Depends on the distribution over trajectories induced by different policies

Relative Performance

Issues:

- We would like to exploit past samples
- We do not know how much to trust them
- Depends on the distribution over trajectories induced by different policies

Performance-Difference Lemma

For any policies $\pi, \pi' \in \Pi^{\mathsf{SR}}$

$$J(\pi') - J(\pi) = \sum_{s,a} d^{\pi'}(s,a) A^{\pi}(s,a)$$
$$= \sum_{s} d^{\pi'}(s) \sum_{a} \pi'(s,a) A^{\pi}(s,a)$$

Optimization step

$$\max_{\pi'} J(\pi')$$

Optimization step

$$\max_{\pi'} J(\pi') = \max_{\pi'} J(\pi') - J(\pi)$$

Issue: as before, cannot be directly estimated using information from π

Optimization step

$$\max_{\pi'} J(\pi') = \max_{\pi'} J(\pi') - J(\pi)$$
$$= \max_{\pi'} \mathbb{E}_{(s,a)\sim d^{\pi'}} \left[A^{\pi}(s,a)\right]$$

Issue: as before, cannot be directly estimated using information from π

Optimization step

$$J(\pi') - J(\pi) = \mathbb{E}_{s\sim d^{\pi}}\left[\sum_a \pi'(s,a)A^{\pi}(s,a)\right] + \sum_s (d^{\pi'}(s) - d^{\pi}(s)) \sum_a \pi'(s,a)A^{\pi}(s,a)$$

Optimization step

$$J(\pi') - J(\pi) = \mathbb{E}_{s\sim d^{\pi}}\left[\sum_a \pi'(s,a)A^{\pi}(s,a)\right] + \sum_s \underbrace{(d^{\pi'}(s) - d^{\pi}(s))}_{?} \sum_a \pi'(s,a)A^{\pi}(s,a)$$

$$\geq \mathbb{E}_{s\sim d^{\pi}}\left[\sum_a \pi'(s,a)A^{\pi}(s,a) - \frac{\gamma\varepsilon}{(1-\gamma)^2} D_{TV}(\pi'\|\pi)[s]\right]$$

where $\varepsilon = \max_s \left|\mathbb{E}_{a\sim\pi'}[A^{\pi}(s,a)]\right|$ and

$$D_{TV}(\pi'\|\pi)[s] = \sum_a |\pi'(s,a) - \pi(s,a)|$$

Surrogate Loss

$$L_\pi(\pi') = J(\pi) + \sum_s d^\pi(s) \sum_a \pi'(s,a) A^\pi(s,a)$$

- $L_\pi(\pi) = J(\pi)$
- If parametric policies $\pi = \pi_\theta$, $\nabla_\theta L_{\pi_\theta}(\pi_\theta) = \nabla_\theta J(\pi_\theta)$

! in an interval close to π, L_π is a good surrogate for J

$\Longrightarrow$ *Conservative Policy Iteration*

Surrogate Loss

$$L_\pi(\pi') = J(\pi) + \sum_s d^\pi(s) \sum_a \pi'(s,a) A^\pi(s,a) \quad - \sum_s d^\pi(s) \frac{\gamma\varepsilon}{(1-\gamma)^2} D_{TV}(\pi' \| \pi)[s]$$

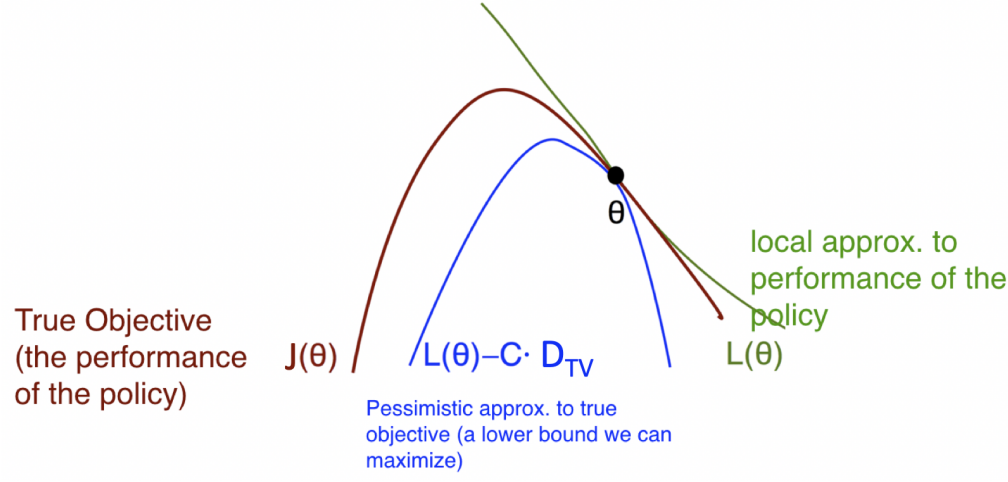
- $L_\pi(\pi) = J(\pi)$
- If parametric policies $\pi = \pi_\theta$, $\nabla_\theta L_{\pi_\theta}(\pi_\theta) = \nabla_\theta J(\pi_\theta)$

also with this

! in an interval close to π, L_π is a good surrogate for J

$\Longrightarrow$ *Conservative Policy Iteration*

Surrogate Loss Cont'd



Conservative Policy Iteration

- *New policy improvement schema*
  - Give current policy π_k solve

$$\max_{\pi'} \left\{ L_{\pi_k}(\pi') - C \, \mathbb{E}_{s \sim d^{\pi_k}} \left[D_{TV}(\pi' \| \pi_k)[s] \right] \right\}$$

Conservative Policy Iteration

- *New policy improvement schema*
  - Give current policy π_k solve

$$\max_{\pi'} \left\{ L_{\pi_k}(\pi') - \boldsymbol{C}\ \mathbb{E}_{s\sim d^{\pi_k}} \left[D_{TV}(\pi' \| \pi_k)[s] \right] \right\} \geq 0$$

Conservative Policy Iteration

- *New policy improvement schema*
  - Give current policy π_k solve

$$\boldsymbol{J(\pi') - J(\pi_k)} \geq \max_{\pi'} \left\{ L_{\pi_k}(\pi') - \boldsymbol{C}\, \mathbb{E}_{s\sim d^{\pi_k}} \left[D_{TV}(\pi' \| \pi_k)[s] \right] \right\} \geq \mathbf{0}$$

Conservative Policy Iteration

- *New policy improvement schema*
  - Give current policy π_k solve

$$\boldsymbol{J(\pi') - J(\pi_k)} \geq \max_{\pi'} \left\{ L_{\pi_k}(\pi') - \boldsymbol{C}\, \mathbb{E}_{s\sim d^{\pi_k}} \left[D_{TV}(\pi' \| \pi_k)[s] \right] \right\} \geq \mathbf{0}$$

$\Longrightarrow$ *Monotonic performance improvement*

Conservative Policy Iteration

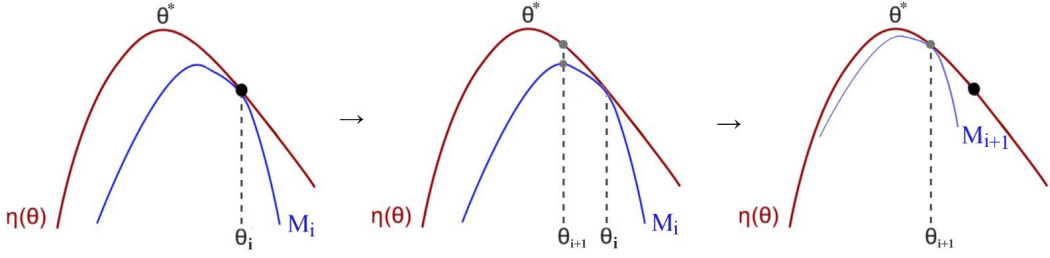
- *New policy improvement schema*
  - Give current policy π_k solve

$$\boldsymbol{J(\pi') - J(\pi_k)} \geq \max_{\pi'} \left\{ L_{\pi_k}(\pi') - \boldsymbol{C}\, \mathbb{E}_{s\sim d^{\pi_k}} \left[D_{TV}(\pi' \| \pi_k)[s] \right] \right\} \geq \mathbf{0}$$

$\Longrightarrow$ *Monotonic performance improvement*

Several approaches have been proposed [e.g., Kakade and Langford, 2002, Perkins and Precup, 2002, Gabillon et al., 2011, Wagner, 2011, 2013, Pirotta et al., 2013b, Scherrer et al., 2015, Schulman et al., 2015]

Idea



$\eta(\theta) = \mathbb{E}\left[\sum_{t=1}^{\infty} r_t | \pi_\theta\right]$ and M is the lower bound

Source

Approximate Monotone Improvement

- The objective can be estimated using rollouts from the most recent policy
- Updates respect a notion of distance in the policy space!

This is the basis for many algorithms!

Toward Practical Algorithm

- Optimizing the total variation $D_{TV}(\pi' \| \pi)$ may be *difficult*
- Relax the problem using *Pinsker's inequality* (**?**)

$$D_{TV}(\pi' \| \pi) \leq \sqrt{2 D_{KL}(\pi' \| \pi)}$$

Further Steps toward Practical Algorithms

- C provided by theory is quite high (*too conservartive*)
- Replace regularization with constraint (*trust region*) (e.g., REPS (?))

$$\pi_{k+1} = \arg\max_{\pi'} L_{\pi}(\pi')$$
$$\text{s.t. } \mathbb{E}_{s\sim d^{\pi}}[D_{KL}(\pi'\|\pi)] \leq \delta$$

Further Steps toward Practical Algorithms

- C provided by theory is quite high (*too conservartive*)
- Replace regularization with constraint (*trust region*) (e.g., REPS (?))

$$\pi_{k+1} = \arg\max_{\pi'} L_{\pi}(\pi')$$
$$\text{s.t. } \mathbb{E}_{s\sim d^{\pi}}[D_{KL}(\pi'\|\pi)] \leq \delta$$

- Importance weighting

$$\mathbb{E}_{s\sim d^{\pi}}\mathbb{E}_{a\sim\pi'}[A^{\pi}(s,a)] = \mathbb{E}_{s\sim d^{\pi}}\mathbb{E}_{a\sim z}\left[\frac{\pi'(s,a)}{z(s,a)}A^{\pi}(s,a)\right]$$

Further Steps toward Practical Algorithms

- C provided by theory is quite high (*too conservartive*)
- Replace regularization with constraint (*trust region*) (e.g., REPS (?))

$$\pi_{k+1} = \arg\max_{\pi'} L_{\pi}(\pi')$$
$$\text{s.t. } \mathbb{E}_{s\sim d^{\pi}}[D_{KL}(\pi'\|\pi)] \leq \delta$$

- Importance weighting

$$\mathbb{E}_{s\sim d^{\pi}}\mathbb{E}_{a\sim\pi'}[A^{\pi}(s,a)] = \mathbb{E}_{s\sim d^{\pi}}\mathbb{E}_{a\sim z}\left[\frac{\pi'(s,a)}{z(s,a)}A^{\pi}(s,a)\right]$$

$\Longrightarrow$ Natural Policy Gradient (NPG)
$\Longrightarrow$ Trust-Region Policy Optimization (TRPO)

Thank you!

facebook
Artificial Intelligence Research