690S: Human-Centric Machine Learning

Reinforcement Learning Overview

Prof. Scott Niekum

Reinforcement Learning:

Learning from experience in sequential decision problems

BC: (s, a^*) vs. RL: (s, a, s', r)

MDP: States: S

Actions: A

Transition probabilities: $T(s, a, s') = p(s' \mid s, a)$

Reward function $r(s, a)$

Start state distribution: d_0

Discount factor: γ

Markov Property: $p(s_t \mid s_{t-1}, a_{t-1}, \ldots, s_0, a_0) = p(s_t \mid s_{t-1}, a_{t-1})$

Objective : find an optimal policy π^* that maximizes expected return

Return : $R = \sum_{t=0}^{\infty} \gamma^t r_t$

Expected Return on MDP : $\mathbb{E}_{\pi, s_0 \sim d_0}[R]$

Why discounting ? Why geometric ?

Side note : Sometimes average reward optimized for instead.

Why is RL hard?

- Delayed effects / credit assignment
- Curse of dimensionality / horizon
- Exploration vs. Exploitation
- Continuous states + actions
- No fixed dataset
- Generalization / distribution shift / nonstationarity
- Designing state representation
- Defining reward
- Hyperparameter tuning
- Reproducibility

⋮

RL landscape

Value function-based

- Model-based
- Model-Free
  - Monte Carlo
  - Backup-based

- "critic-only": how good is a state?
- sample efficient
- deals poorly w/ partial observability, bad features
- usually discrete action

Policy Search

- Policy Gradient
- Black-box optimization

- "actor-only"
- high variance, low efficiency
- robust to partial observability, bad features
- naturally works with continuous action

Actor-critic

combines VF + PS

- often best of both worlds
- efficient, robust
- continuous action

Value functions:

$$V_\pi(s) = E_\pi\left[R_t \mid s_t = s\right] \quad \text{where } R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

$$= \sum_a \pi(s,a) \sum_{s'} T(s,a,s')\left[r(s,a) + \gamma V_\pi(s')\right]$$

$$Q_\pi(s,a) = E_\pi\left[R_t \mid s_t = s,\ a_t = a\right]$$

$$= \sum_{s'} T(s,a,s')\left[r(s,a) + \gamma V_\pi(s')\right]$$

Bellman optimality equations:

$$V^\star = \max_a Q^\star(s,a)$$

$$Q^\star(s,a) = \sum_{s'} T(s,a,s')\left[r(s,a) + \gamma V^\star(s')\right]$$

$$V^\star(s) = \max_a \sum_{s'} T(s,a,s')\left[r(s,a) + \gamma V^\star(s')\right]$$

Optimal policy:

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

Value function methods

Model-based : Dynamic programming to compute Q^* / V^*

Model-free :

Monte-Carlo : Average return samples

On the i^{th} visit to (s,a), record return after : $R^i_{s,a}$

$$\hat{Q}(s,a) = \frac{1}{N} \sum_{i=0}^{N} R^i_{s,a}$$

Q-learning : $\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \Big[\underbrace{r_t + \gamma \max_{a'} \hat{Q}(s',a')}_{\text{"observation"}} - \underbrace{\hat{Q}(s,a)}_{\text{prediction}} \Big]$

"Bootstrapping"

off-policy vs. on-policy

Q-learning : $\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \left[r_t + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a) \right]$

Q-learning is off-policy. Actions can come from any policy, but $\hat{Q}$ will still converge to Q^* (in the tabular case)

SARSA is on-policy. Learns and improves Q^π based on current policy π

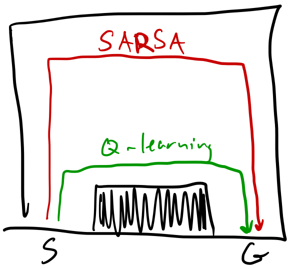
$$\hat{Q}(s,a) \leftarrow \hat{Q}(s,a) + \alpha \left[r_t + \gamma \hat{Q}(s', \pi(s')) - \hat{Q}(s,a) \right]$$

e.g. where $\pi \sim$ epsilon-greedy $(\hat{Q}(s',\cdot))$

π must become greedy over time for on-policy to converge to Q^* !

otherwise

Converged policies: Cliff world with ϵ-greedy exploration



why would you use on-policy algorithms?

→ Better convergence guarantees under function approximation!

Policy Search

Parameterized policy $f_\theta(s) \to a$ or $f_\theta(s,a) \to \mathbb{R}$

Example with linear function approx. w/ features ϕ:

$$f_\theta(s) = \theta_1 \phi_1(s) + \theta_2 \phi_2(s) + \ldots + \theta_n \phi_n(s)$$

Methods:

1. Policy gradient: approximate $\frac{dR}{d\theta}$ and use to update parameters

$$J(\theta) = E_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

$$\nabla_\theta J(\theta) = \sum_s \mu_\pi(s) \sum_a Q^\pi(s,a) \nabla_\theta \pi_\theta(s,a)$$

Need to approximate

REINFORCE does Monte Carlo approx of Q^π

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta J(\theta)$$

2. Black-box optimization

- CMA-ES
- perturb + hill-climb

Actor - Critic

Like policy search, directly parameterizes policy

Like value function methods, also computes VF

Example:

- REINFORCE (policy search) uses MC estimation of $Q(s,a)$ when estimating $\nabla_\theta J(\theta)$

  e.g. $\hat{Q}(s_t, a_t) = r_t + \gamma r_{t+1} + \ldots + \gamma^N r_{t+N}$

- Critic can reduce variance by using:

  $\hat{Q}(s_t, a_t) = r_t + \gamma \hat{V}(s_{t+1})$

✓ Continuous action

✓ Lower variance than policy search alone

✗ More complex, sometimes annoying in practice

CS 690: Human-Centric Machine Learning

Prof. Scott Niekum

Trust region methods

First-order optimization can be dangerous

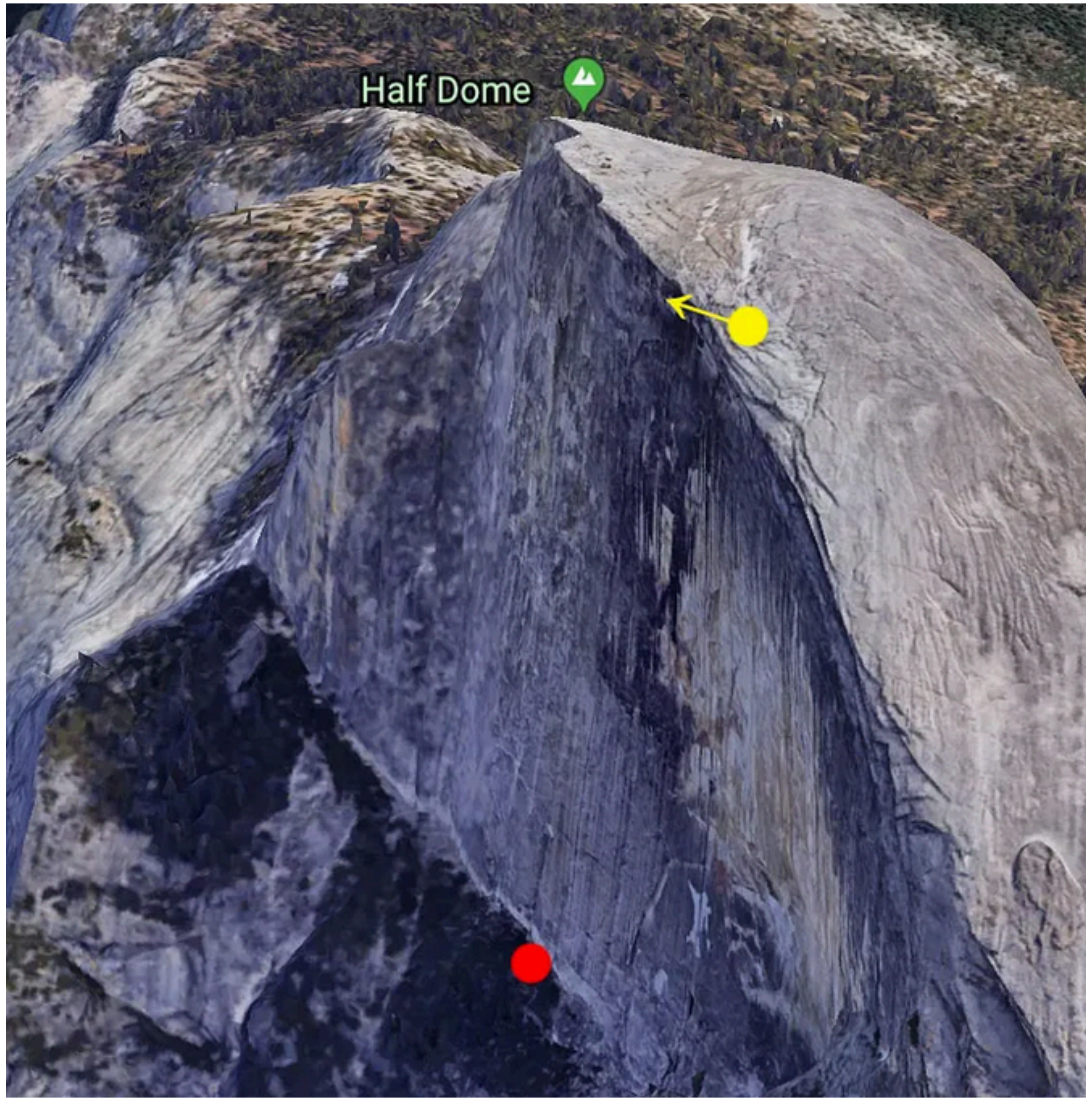


Image credit: https://medium.com/@jonathan_hui/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9

Trust regions

Line search
(like gradient ascent)

Trust region

Image credit: https://medium.com/@jonathan_hui/rl-trust-region-policy-optimization-trpo-explained-a6ee04eeeee9

Sensitivity to parameterization

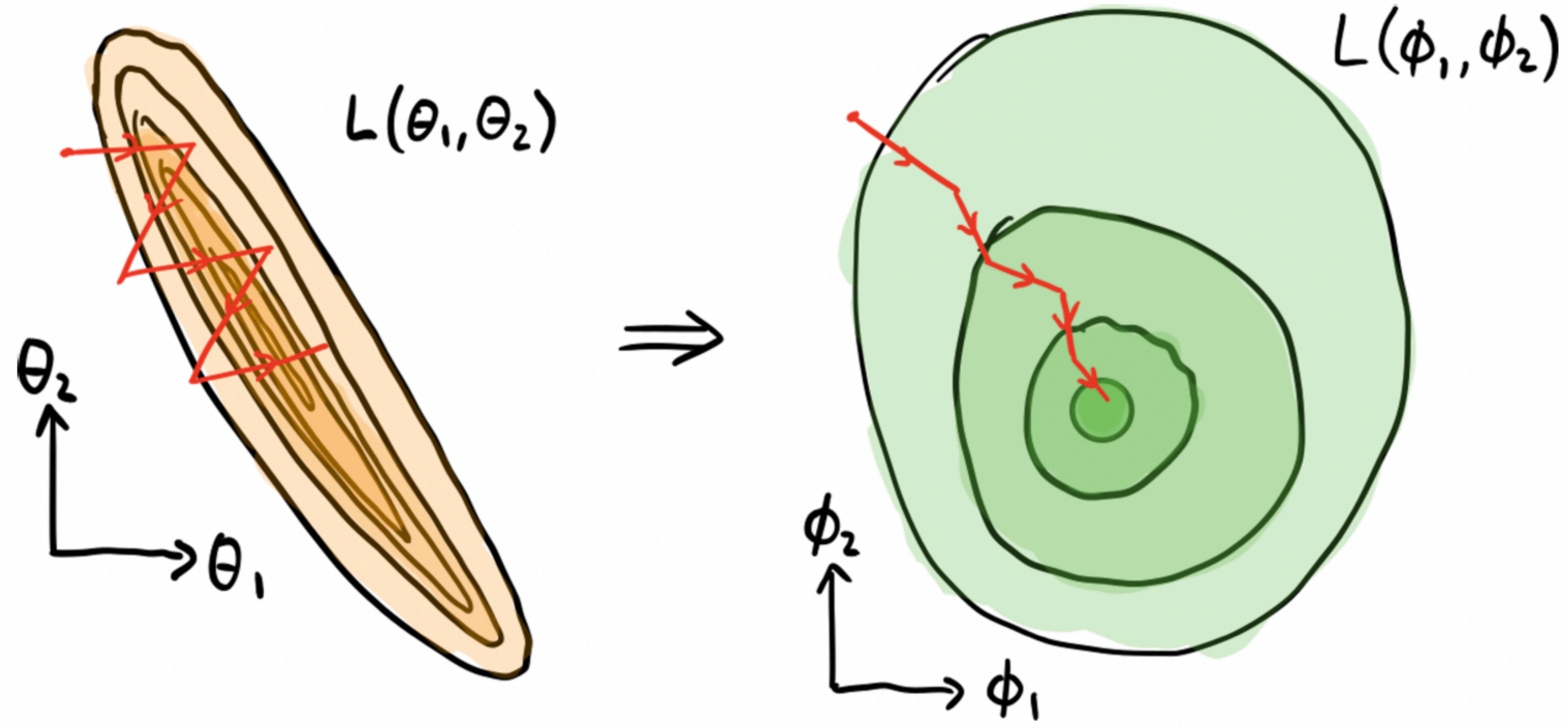


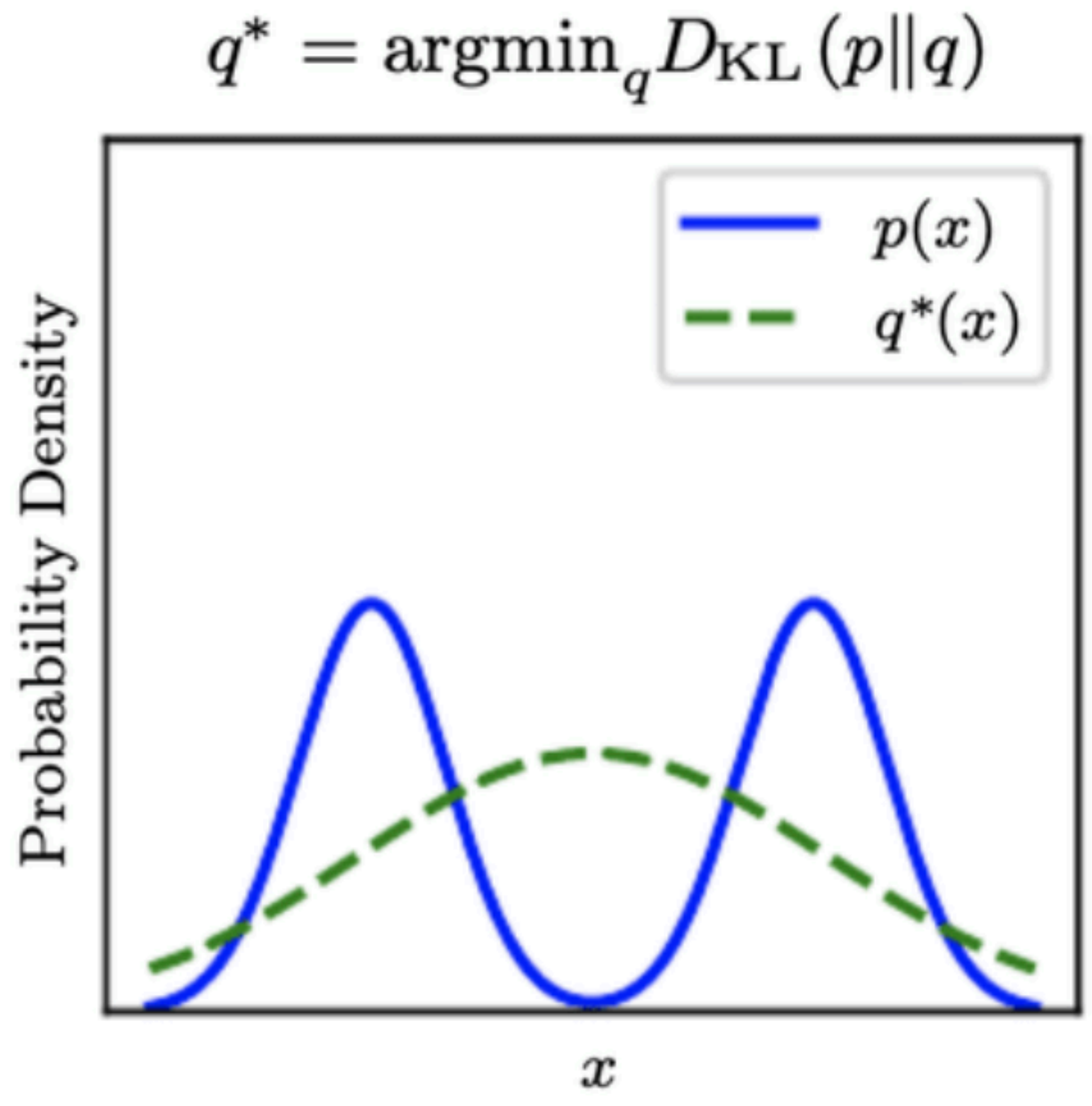
Image credit: https://pennylane.ai/qml/demos/tutorial_quantum_natural_gradient/

KL Divergence

$$D_{KL}(P \parallel Q) = \mathbb{E}_{x \sim P}\left[\log \frac{P(X)}{Q(X)}\right]$$

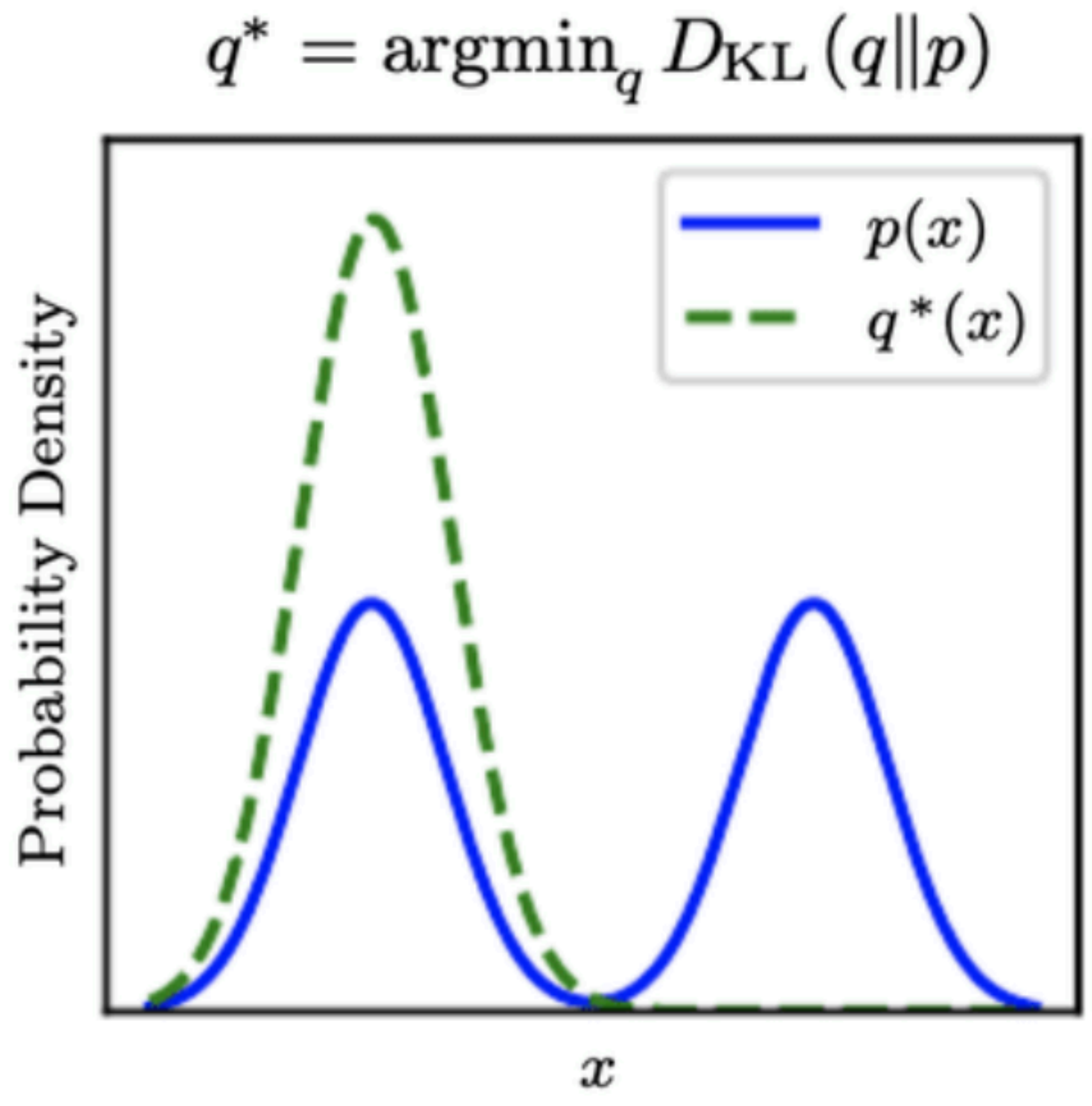
Forward KL

Mean-seeking



Reverse KL

Mode-seeking



TRPO basic idea

Algorithm 1 Policy iteration algorithm guaranteeing non-decreasing expected return η

Initialize π_0.
for $i = 0, 1, 2, \ldots$ until convergence **do**
  Compute all advantage values $A_{\pi_i}(s, a)$.
  Solve the constrained optimization problem

$$\pi_{i+1} = \arg\max_{\pi} \left[L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$$

  where $C = 4\epsilon\gamma/(1-\gamma)^2$

  and $L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$

end for

$$\underset{\theta}{\text{maximize}}\ L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } D_{\mathrm{KL}}^{\max}(\theta_{\text{old}}, \theta) \le \delta.$$

$$\underset{\theta}{\text{maximize}}\ L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } \boxed{\overline{D}_{\mathrm{KL}}^{\rho_{\theta_{\text{old}}}}}(\theta_{\text{old}}, \theta) \le \delta.$$

Quadratic approximation:
Fisher information matrix

PPO

On-policy actor-critic

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L(s,a,\theta_k,\theta) \right],$$

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s,a)) \right),$$

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

Soft actor-critic

Off-policy actor-critic

- PPO can't learn from ***offline data*** or reuse data, since it is on policy
- PPO policies tend to get more deterministic over time, leading exploration to collapse and learning to stagnate
- SAC can reuse past experience or offline data since it is off policy
- SAC uses entropy maximization to learn the most random policy possible that performs well, leading to natural exploration and robustness to estimation errors