Reinforcement Learning: Continuous Control with Off-Policy Learning (ESE 3990)

Instructor:
Antonio Loquercio

Fall 2026

University of Pennsylvania

Categorizing RL agents

Axis 1: How to make decisions

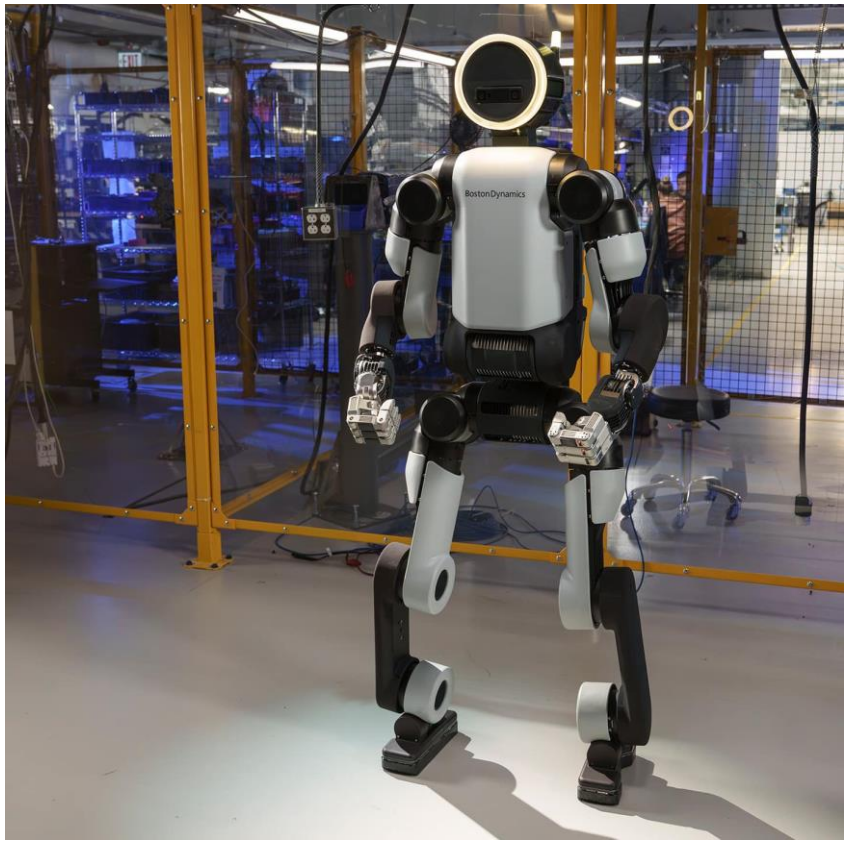
- Value Based
  - No Policy (Implicit)
  - Value/Q Function
- Policy Based
  - Policy
  - No Value/Q Function
- Actor Critic
  - Policy
  - Value Function

Today

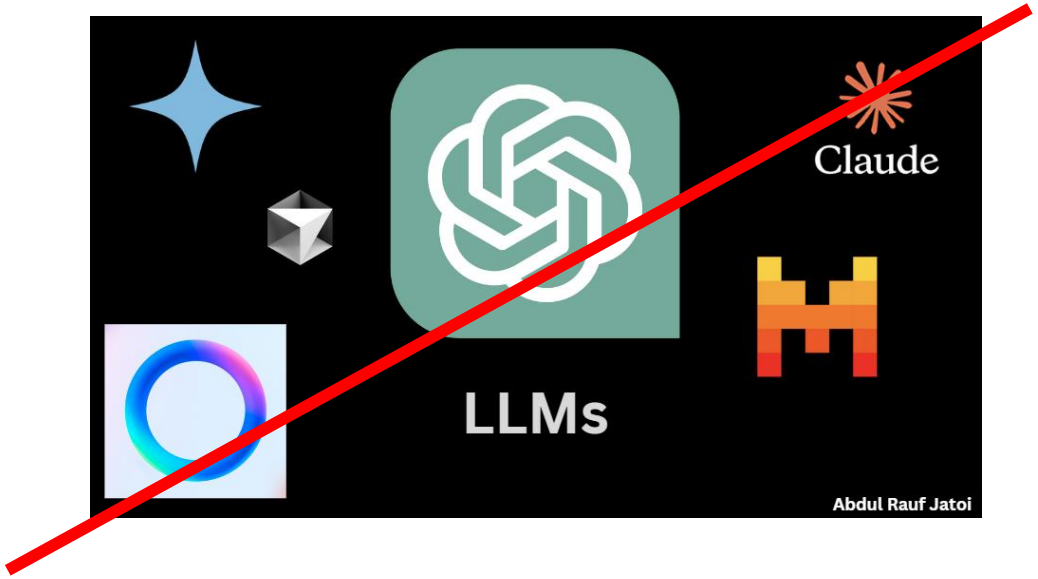
Axis 2: Environment Representation

- Model-based
  - Policy and/or Value/Q Function
  - Model
- Model-free
  - Policy and/or Value/Q Function
  - No Model

Continuous Control







The Challenges of Continuous Control

- Cannot visit “all” states, we need a way to generalize from seen states to unseen ones
  - → Function Approximation of Value Functions and Policies

- Every learning step requires a maximization process
  - Target value $y_i = r(s,a) + \gamma Q\left(s', \arg\max_{a'} Q(s', a', w)\,;\, \bar{w}\right)$
  - → Stochastic optimization or Learning an Optimizer

Learning an Optimizer

- **Idea:** $\pi_\theta(s)$ should approximate a greedy policy: $\pi_\theta(s) \approx \arg\max_a Q(s, a, w)$

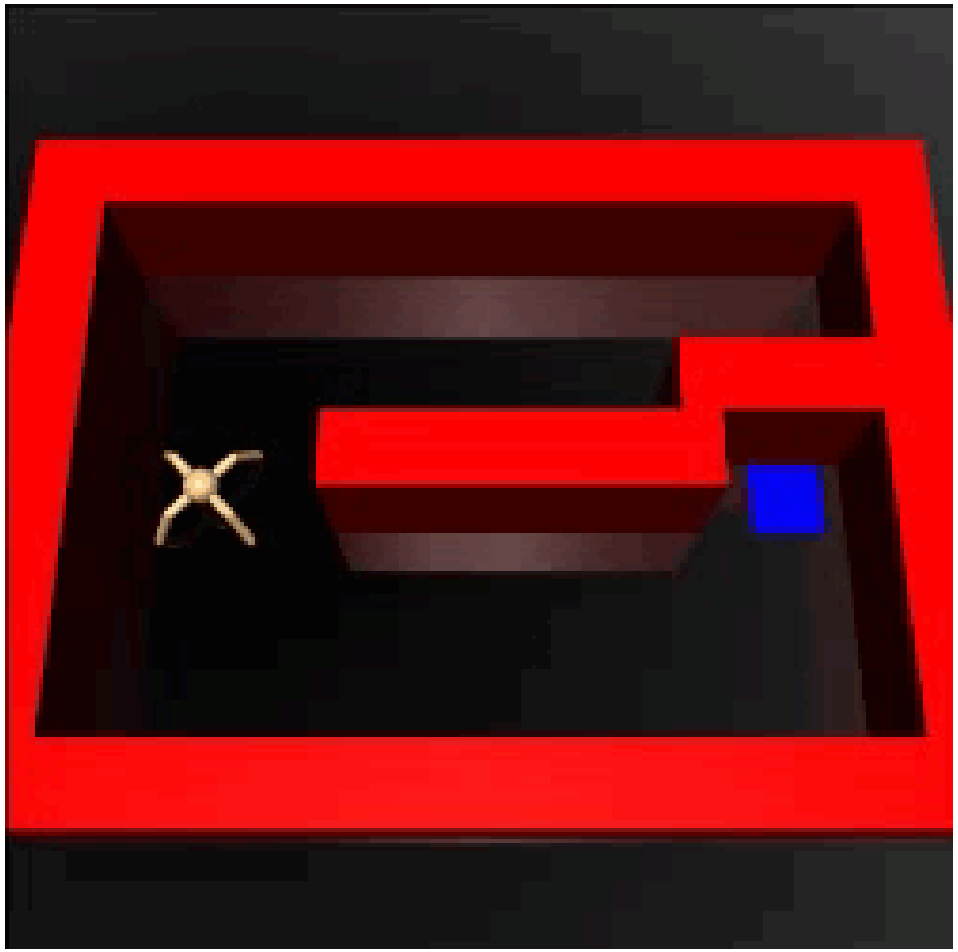
- Train the policy to maximize $\mathbb{E}_D[Q(s, \pi_\theta(s), w)] = \mathbb{E}_D[V^{\pi,w}(s)]$

- Do this maximization with one step of gradient ascent:
- $g = \nabla_\theta \mathbb{E}_D[Q(s, \pi_\theta(s), w)] = \mathbb{E}_D[\nabla_\theta Q(s, \pi_\theta(s), w)]$
- $\hat{g} = \sum_i \nabla_\theta Q(s_i, \pi_\theta(s_i), w)$ (approximate by sampling)
- $\hat{g} = \sum_i \frac{dQ(s_i, \pi_\theta(s_i), w)}{d\,\pi_\theta(s_i)} \frac{d\pi_\theta(s_i)}{d\theta}$ (chain rule)
- $\theta^{new} \leftarrow \theta^{old} + \alpha \hat{g}$

Deep Deterministic Policy Gradients (DDPG)

- Init w,θ
- Collect one episode with *the policy* π_θ, add it to the replay buffer D
  - For K iterations:
    - Sample a batch of transitions $\{(s_i, a_i, s_i', r_i)\}^B$ from D
    - Calculate target (using target networks $\bar{w}$ and $\bar{\theta}$): $y_i = r_i + \gamma Q(s_i', a' \sim \pi_{\bar{\theta}}(s'), \bar{w})$
    - Compute Critic Loss: $L_Q(w) = \frac{1}{B}\sum_i \big(y_i - Q(s_i, a_i, w)\big)^2$
    - Compute Policy Loss: $L_\pi(\theta) = \frac{1}{B}\sum_i Q(s_i, \pi(s_i), w)$
    - Make one gradient *descent* step for the critic: $w \leftarrow w - \frac{dL_q}{dw}$
    - Make one gradient *ascent* step for the policy: $\theta \leftarrow \theta + \frac{dL_\pi}{d\theta}$
    - $\bar{w} \leftarrow m\bar{w} + (1-m)w \qquad (m = 0.999)$
    - $\bar{\theta} \leftarrow m\bar{\theta} + (1-m)\theta \qquad (m = 0.999)$

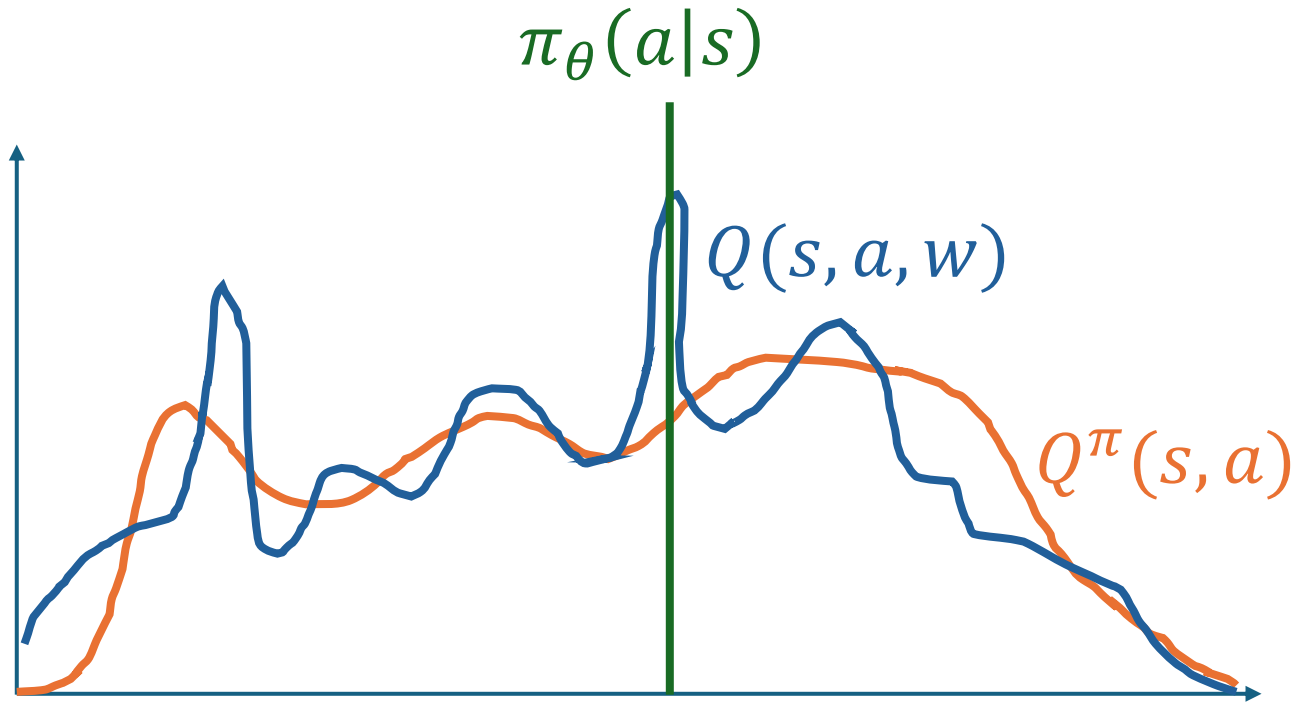
DDPG in Action



Exploration Problems

- The policy is deterministic.
- At the beginning of training, it would probably not explore a wide variety of actions to find useful signals.
- To improve exploration, we add noise to the actions:
  - $a = \pi_\theta(s) + \epsilon$, with $\epsilon \sim N(0, \sigma)$
- Gradually anneal σ as the training proceeds.
- Works OK, but we can do better (more on this later).

Exploitation Problems



The policy $\pi_\theta(s)$ will exploit the overestimation errors of the critic $Q(s,a,w)$

Twin Delayed DDPG (TD3)

- Aims to address the limitations of Q-function exploitation

- It adds three different ideas to DDPG to do so:

- Target Policy Smoothing
- Clipped Double Q-Learning (Twins)
- Delayed Policy Updates (Delayed)

Target Policy Smoothing

- Target calculation in DDPG:

$$y_i = r_i + \gamma Q(s_i', a' \sim \pi_{\overline{\theta}}(s'), \overline{w})$$

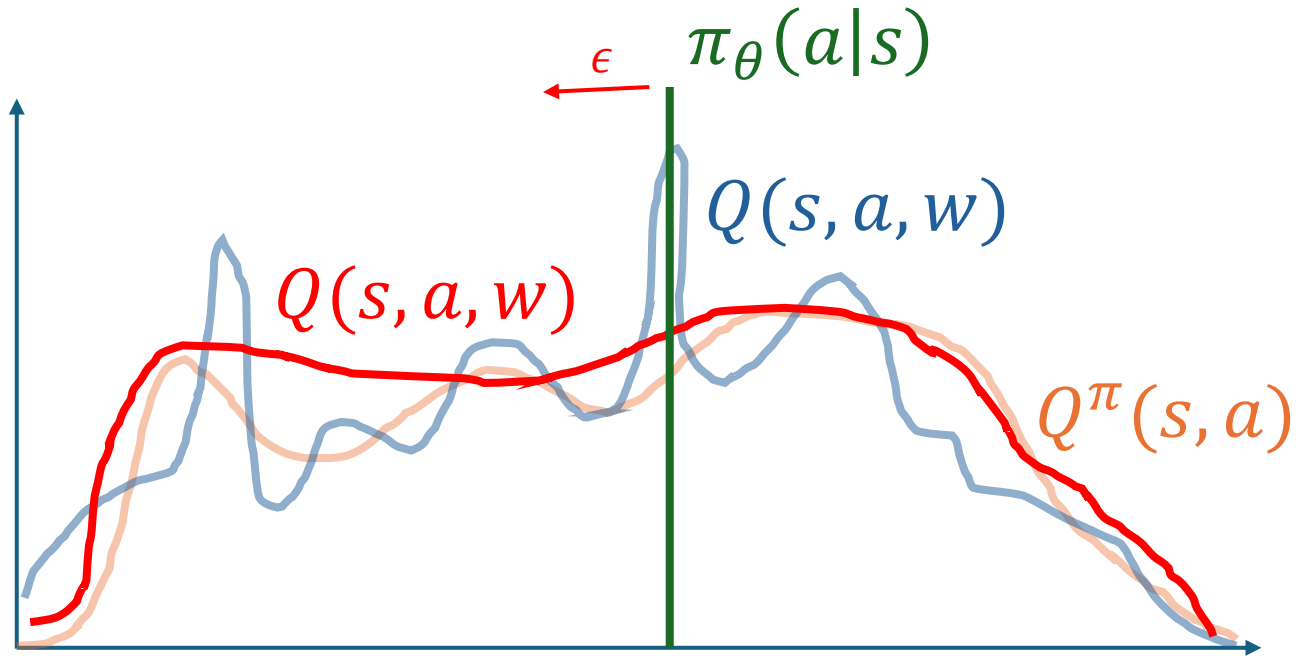
- Add some noise to the target action:

$$a' = \pi_{\overline{\theta}}(s') + \epsilon \qquad \epsilon \sim N(0, \sigma)$$

- Use this action for target calculation

$$y_i = r_i + \gamma Q(s_i', a' \sim \pi_{\overline{\theta}}(s') + \epsilon, \overline{w})$$

Target Policy Smoothing



Clipped Double Q-Learning

- As in Double Q-Learning, use two Q-functions: $Q(\cdot,\cdot,w_1)$, $Q(\cdot,\cdot,w_2)$
- They are trained on the same data, but starting from different initializations
  - Technically wrong, but random init + SGD seems to work in practice
- When computing the target, pick the minimum between the two:

$$y_i = r_i + \gamma \min_k Q(s_i', a', \overline{w_k})$$

Delayed Policy Updates

- Updating the policy at every step makes it challenging to learn its Q function.
- Proposed solution: Update the policy at a slower rate
- Seems to help, but follow-on papers did not adopt this idea, which probably indicates it is not too important.

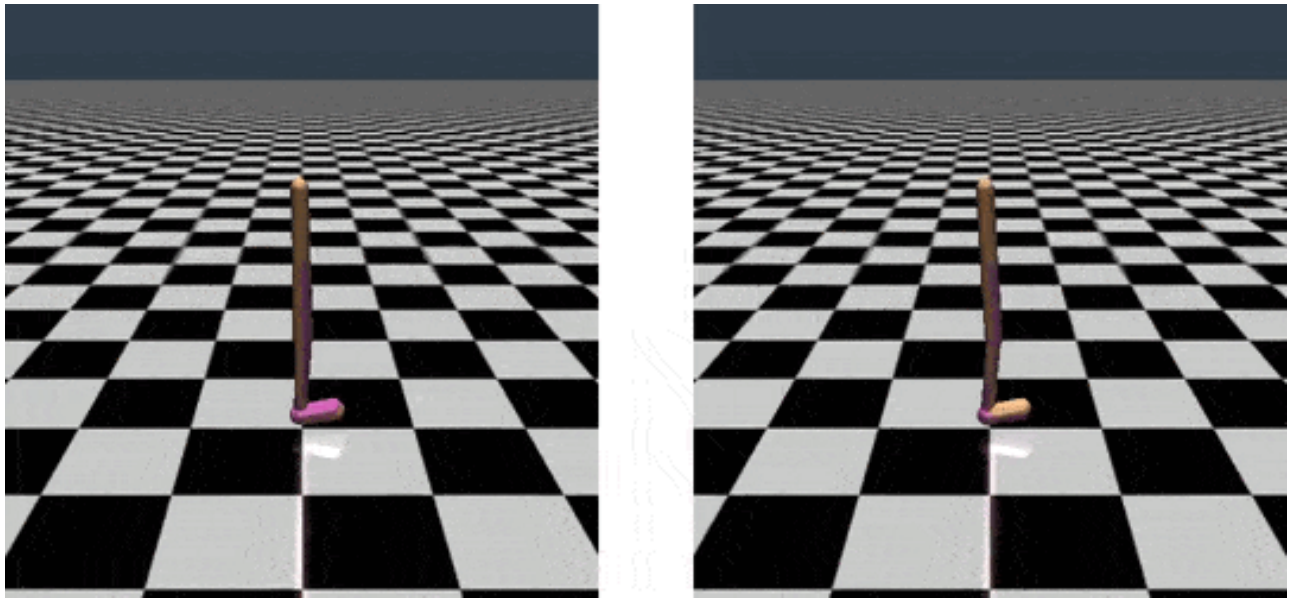


TD3 Algorithm

- Init w_1, w_2, θ
- Collect one episode with *the policy* π_θ, add it to the replay buffer D
  - For K iterations:
    - Sample a batch of transitions $\{(s_i, a_i, s_i', r_i)\}^B$ from D
    - Calculate target (using target networks $\overline{w}$ and $\bar{\theta}$): $y_i = r_i + \gamma \min_k Q(s_i', a' \sim \pi_{\bar{\theta}}(s') + \epsilon, \overline{w_k})$
    - Compute Critic Losses: $L_Q(w_k) = \frac{1}{B}\sum_i \left(y_i - Q(s_i, a_i, w_k)\right)^2$
    - (every n-th iteration): Compute Policy Loss: $L_\pi(\theta) = \frac{1}{B}\sum_i Q(s_i, \pi(s_i), w)$
    - Make one gradient *descent* step for the critics: $w_k \leftarrow w_k - \frac{dL_q(w_k)}{dw_k}$
    - (every n-th iteration): Make one gradient *ascent* step for the policy: $\theta \leftarrow \theta + \frac{dL_\pi}{d\theta}$
    - $\overline{w} \leftarrow m\overline{w} + (1-m)w \qquad (m = 0.999)$
    - $\bar{\theta} \leftarrow m\bar{\theta} + (1-m)\theta \qquad (m = 0.999)$

Going Back To Exploration

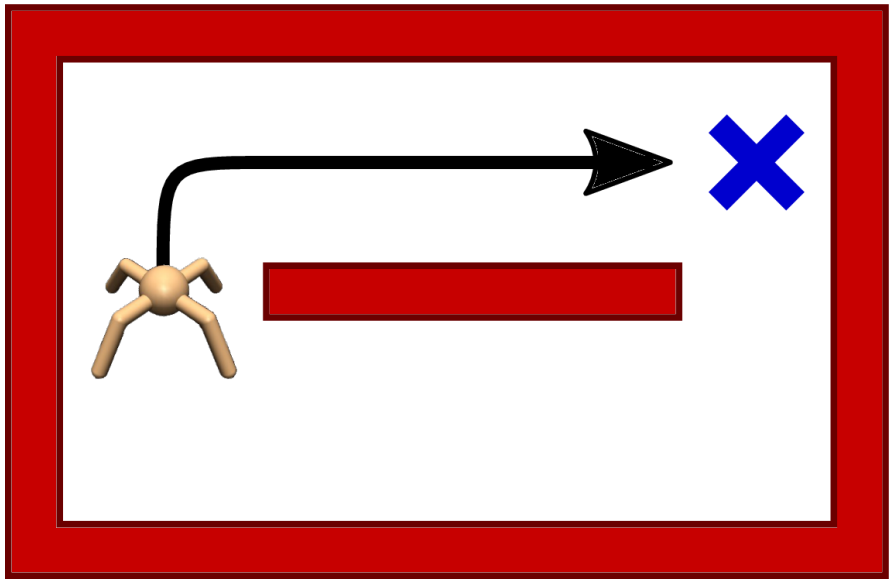
- Easy to get stuck in a local minimum.
- RL algorithms stick to whatever solution they initially find.

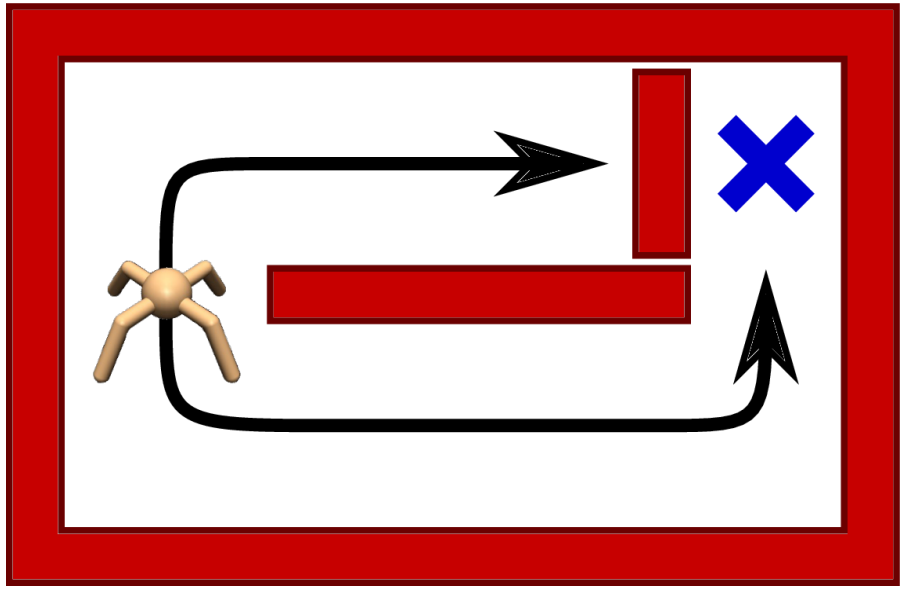


[credit: John Schulman and Patrick Coady (OpenAI Gym)]

Why is this a problem?

- Learning different ways to accomplish a task makes the agent more robust to environmental variations.

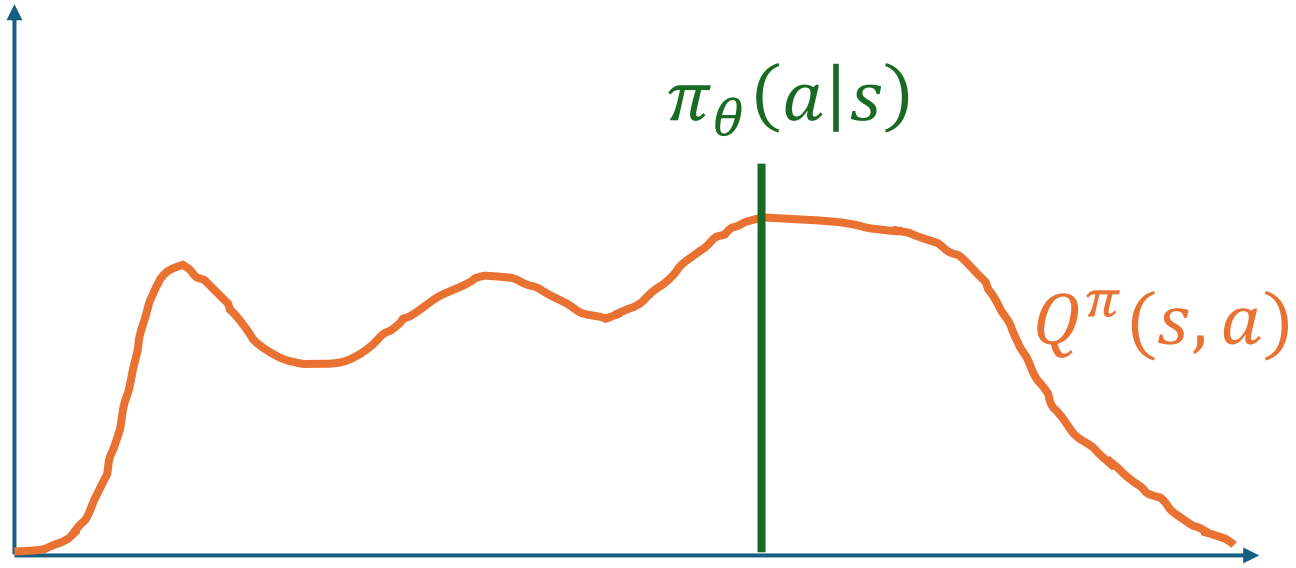




Source

The Problem of Deterministic Policies

- A deterministic policy focuses on the region where the Q function is (expected to be) highest.
- But what if the other parts are not much worse?

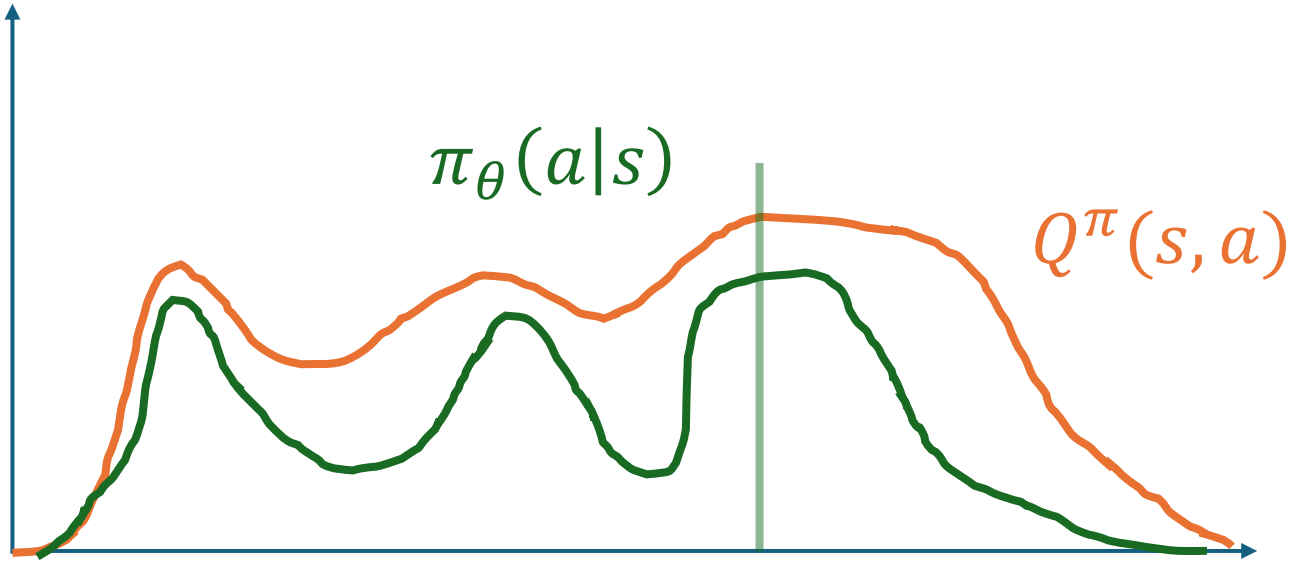


The Problem of Deterministic Policies

- A simple way to capture the multi-modality of the distribution is:

$$\pi(a|s) \propto \exp\left(Q^{\pi}(s,a)\right)$$

- This makes the actor "soft"

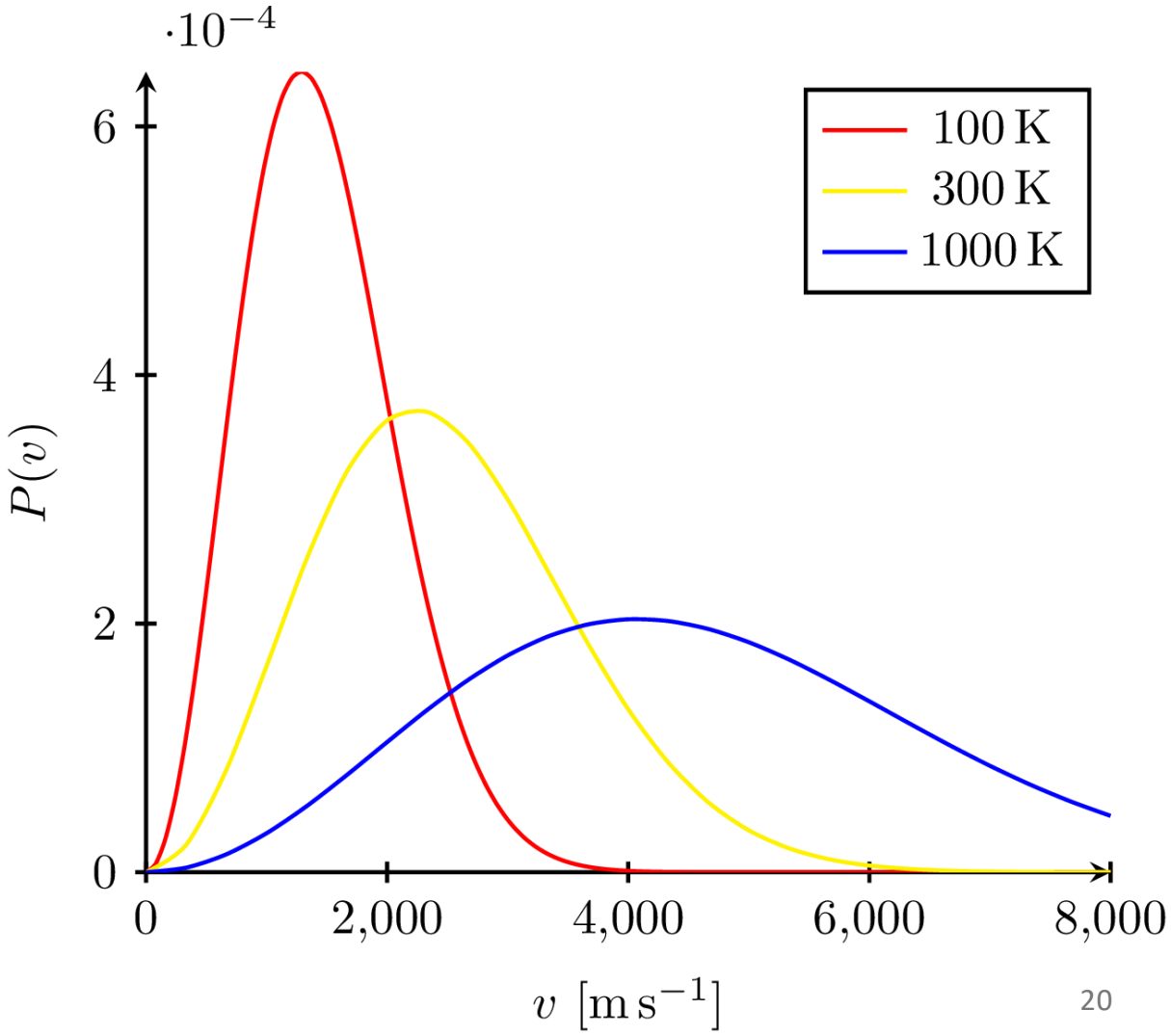


Aside: Boltzmann Distribution

- Probability that a system is in a certain state as a function of the state's energy e_i and the system's temperature T.

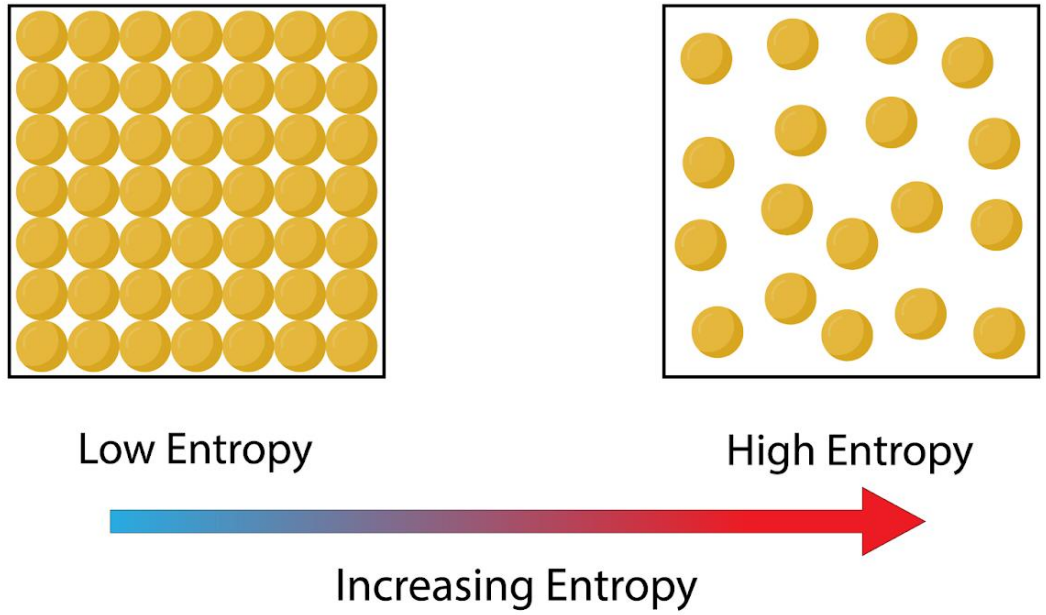
$$p_i \propto \exp\left(-\frac{e_i}{K_b T}\right)$$

- Maxwell-Boltzmann Distribution: Probability of the velocity of particles of an ideal gas at different temperatures.



Aside: Entropy

- Entropy measures the amount of "disorder" of a system.
- Shannon Entropy: $\mathcal{H}(p) = \mathbb{E}_{x \sim p(x)}[-\log p(x)]$
- The higher the entropy, the harder it is to predict the system's state.



Aside: Boltzmann Distribution and Entropy

$$p_i \propto \exp\left(-\frac{e_i}{K_b T}\right)$$

Boltzmann Distribution

Maximizes

$$\mathcal{H}(p) = \mathbb{E}_{x\sim p(x)}[-\log p(x)]$$

Shannon Entropy

- For systems with *Fixed Maximum Energy* $E = \sum_i e_i$:
  - If a system's state probability distribution is Boltzmann, then the system has maximum entropy.
  - (Opposite): If a system has maximum entropy, then its state probability distribution is Boltzmann.

What does it mean for policies?

- Given a state s, define the *fixed* energy of a policy as $\mathrm{E}(s) = \int_a -Q^\pi(s,a)$
- Therefore, a "Boltzmann" policy $\pi(a|s) \propto \exp\big(Q^\pi(s,a)\big)$
- Has maximum entropy!

$$\pi^*_{MaxEnt} = \arg\max_\pi \mathbb{E}_{\tau\sim p(\tau)}\left[\sum_t r_t + \alpha\mathcal{H}\left(\pi(\cdot\,|s_t)\right)\right] \propto \exp\left(\frac{Q^\pi(s,a)}{\alpha}\right)$$

- Where α (*temperature*), controls how "spiky" the distribution is.
  - The smaller the temperature, the more deterministic the policy.
  - For details, see *Modeling Interaction with the principle of maximum causal entropy (Ziebart, 2010)*.

Entropy In Reinforcement Learning

- We can cover multiple modes of the Q function by training policies to maximize entropy.
  - But what if I don't care? My estimated Q-function is anyway noisy…
- MaxEntropy has practical implications for RL even when the Q function (or its estimate) is unimodal.
- Maximizing entropy alongside performance pushes a policy to explore better!
  - And helps prevent getting stuck early in local minima.

Entropy-Regularized Reinforcement Learning

- We change the objective of the optimization problem to be:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau \sim p(\tau)} \left[\sum_t \gamma^t \left(r_t + \alpha \mathcal{H}\left((\pi(\cdot \,|s_t)) \right) \right) \right]$$

- According to this definition, we further define:

$$V^{\pi}(s) = \mathbb{E}_{\tau \sim p(\tau|s_0=s)} \left[\sum_t \gamma^t (r_t + \alpha \mathcal{H}(\pi(\cdot \,|s_t))) \right]$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\tau \sim p(\tau|s_0=s,a_0=a)} \left[\sum_{t=0} \gamma^t r_t + \alpha \sum_{t=1} \gamma^t \mathcal{H}(\pi(\cdot \,|s_t)) \right]$$

Entropy-Regularized Reinforcement Learning

- From the prior definitions:

$$V^{\pi}(s) = \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a)] + \alpha\mathcal{H}(\pi(\cdot\,|s))$$

- And the respective Bellman Equation is:

- $Q^{\pi}(s,a) = \mathbb{E}_{s'\sim P(s'|s,a)}[r(s,a) + \gamma V^{\pi}(s')]$
- $Q^{\pi}(s,a) = \mathbb{E}_{s'\sim p(s'|s,a)}[r(s,a) + \gamma(\mathbb{E}_{a\sim\pi}[Q^{\pi}(s',a)] + \alpha\mathcal{H}(\pi(\cdot\,|s')))]$
- $Q^{\pi}(s,a) = \mathbb{E}_{s'\sim p(s'|s,a),a'\sim\pi(s')}[r(s,a) + \gamma(Q^{\pi}(s',a') - \alpha\log\pi(a'|s'))]$

Entropy-Regularized Reinforcement Learning

- Learning Q functions in traditional reinforcement learning:
  - Target: $y_i = r_i + \gamma Q(s_i', a' \sim \pi_{\bar{\theta}}(s'), \bar{w})$
  - Critic Loss: $L_Q(w) = \frac{1}{B}\sum_i \big(y_i - Q(s_i, a_i, w)\big)^2$

- Learning Q functions in Entropy-regularized RL
  - Target: $y_i = r_i + \gamma Q(s_i', a', \bar{w}) \; - \alpha\gamma \log \pi(a'|s')$, with $a' \sim \pi(\cdot\,|s')$
  - Critic Loss: $L_Q(w) = \frac{1}{B}\sum_i \big(y_i - Q(s_i, a_i, w)\big)^2$

Policy Loss in Entropy-Regularized RL

- We want the policy to maximize the value
- $V^{\pi}(s) = \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a)] + \alpha\mathcal{H}(\pi(\cdot\,|s)) = \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a) - \alpha\log\pi(a|s)]$

- I use the same idea as before, and do one step of gradient ascent:

- $g = \nabla_{\theta}V^{\pi}(s) = \nabla_{\theta}\mathbb{E}_{a\sim\pi_{\theta}}[Q^{\pi}(s,a) - \alpha\log\pi_{\theta}(a|s)]$

- Is this equal to $\mathbb{E}_{a\sim\pi_{\theta}}[\nabla_{\theta}(Q^{\pi}(s,a) - \alpha\log\pi_{\theta}(a|s))]$?

Policy Loss in Entropy-Regularized RL

- I cannot put the gradient inside, since the expectation depends on the parameter
  - This is the same problem we encountered when deriving policy gradients!

- We can address this problem with the *reparametrization trick*:

$$\pi_\theta(a|s) = \mathcal{N}\big(\mu_\theta(s), \sigma_\theta(s)\big)$$

- where μ_θ and σ_θ are neural nets
- I can reparametrize $\pi_\theta(a|s)$ as:

$$\pi_\theta(a|s) = \mu_\theta(s) + \sigma_\theta(s)\mathcal{N}(0,1)$$

Policy Loss in Entropy-Regularized RL

- $V^{\pi}(s) = \mathbb{E}_{a\sim\pi}[Q^{\pi}(s,a) - \alpha \log \pi(a|s)]$

- With the reparameterization trick, I can write this as:

$$a(\epsilon, \theta) = \mu_{\theta}(s) + \epsilon \sigma_{\theta}(s)$$

$$V^{\pi}(s) = \mathbb{E}_{\epsilon\sim N(0,1)}\left[Q^{\pi}\left(s, a(\epsilon,\theta)\right) - \alpha \log \pi(a(\epsilon,\theta)|s)\right]$$

- Now I can easily do gradient ascent
- $g = \nabla_{\theta} V^{\pi}(s) = \mathbb{E}_{\epsilon\sim N(0,1)}\left[\nabla_{\theta}\left(Q^{\pi}\left(s, a(\epsilon,\theta)\right) - \alpha \log \pi(a(\epsilon,\theta)|s)\right)\right]$
- $\hat{g} = \nabla_{\theta}(Q^{\pi}\left(s, a(\epsilon_i,\theta)\right) - \alpha \log \pi(a(\epsilon_i,\theta)|s))$ (sample approx.)

The Soft-Actor Critic (SAC) Algorithm

- Init w_1, w_2, θ, α
- Collect one episode with *the policy* π_θ, add it to the replay buffer D
  - For K iterations:
    - Sample a batch of transitions $\{(s_i, a_i, s_i', r_i)\}^B$ from D
    - Calculate target (using target networks $\overline{w}$): $y_i = r_i + \gamma \min_k Q(s_i', a', \overline{w_k}) - \alpha\gamma \log \pi(a'|s')$
    - Compute Critic Losses: $L_Q(w_k) = \frac{1}{B}\sum_i \left(y_i - Q(s_i, a_i, w_k)\right)^2$
    - Compute Policy Loss: $L_\pi(\theta) = \frac{1}{B}\sum_i Q(s_i, \pi(s_i, \epsilon_i), w) - \alpha \log \pi(s_i, \epsilon_i)$
    - Make one gradient *descent* step for the critics: $w_k \leftarrow w_k - \frac{dL_q(w_k)}{dw_k}$
    - Make one gradient *ascent* step for the policy: $\theta \leftarrow \theta + \frac{dL_\pi}{d\theta}$
    - $\overline{w} \leftarrow m\overline{w} + (1-m)w \qquad (m = 0.999)$

What about the temperature α?

- It seems a very important hyperparameter.
- In practice, it is very challenging to tune.

- But we can optimize it if we change the problem to a constrained optimization one:
- $\max_{\pi} \mathbb{E}_{\tau \sim p(\tau)}[\sum_t r_t]$, s.t. $\mathcal{H}(\pi(\cdot\,|s_t)) \geq \mathcal{H}_{min}$
- $\max_{\pi} \mathbb{E}_{\tau \sim p(\tau)}[\sum_t r_t]$, s.t. $\mathbb{E}_{(s_t,a_t)\sim p(\tau)}[-log\pi(a_t|s_t)] \geq \mathcal{H}_{min}$

What about the temperature α?

- It is possible (though a bit challenging) to show that this formulation also provides a way to learn the parameter α!
  - While leaving the updates for the actor and critic unchanged and making a few (somewhat debatable) assumptions along the way.
- The loss to adapt the temperature is:
  - $L(\alpha) = -\alpha \log \pi(s_i, \epsilon_i) - \alpha \mathcal{H}_{min}$ [Haarnoja et al, 2018]
- Which we optimize with gradient descent:
  - $\alpha_{new} = \alpha_{old} - \frac{dL(\alpha)}{d\alpha}$
- It turns out that, in practice, tuning $\mathcal{H}_{min}$ is simpler than tuning α!

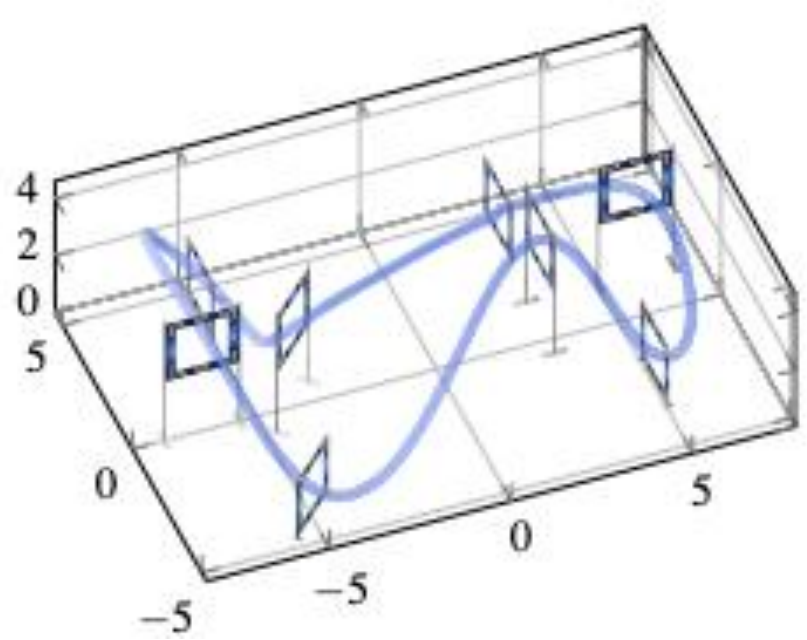
The Soft-Actor Critic (SAC) Algorithm (Revised)

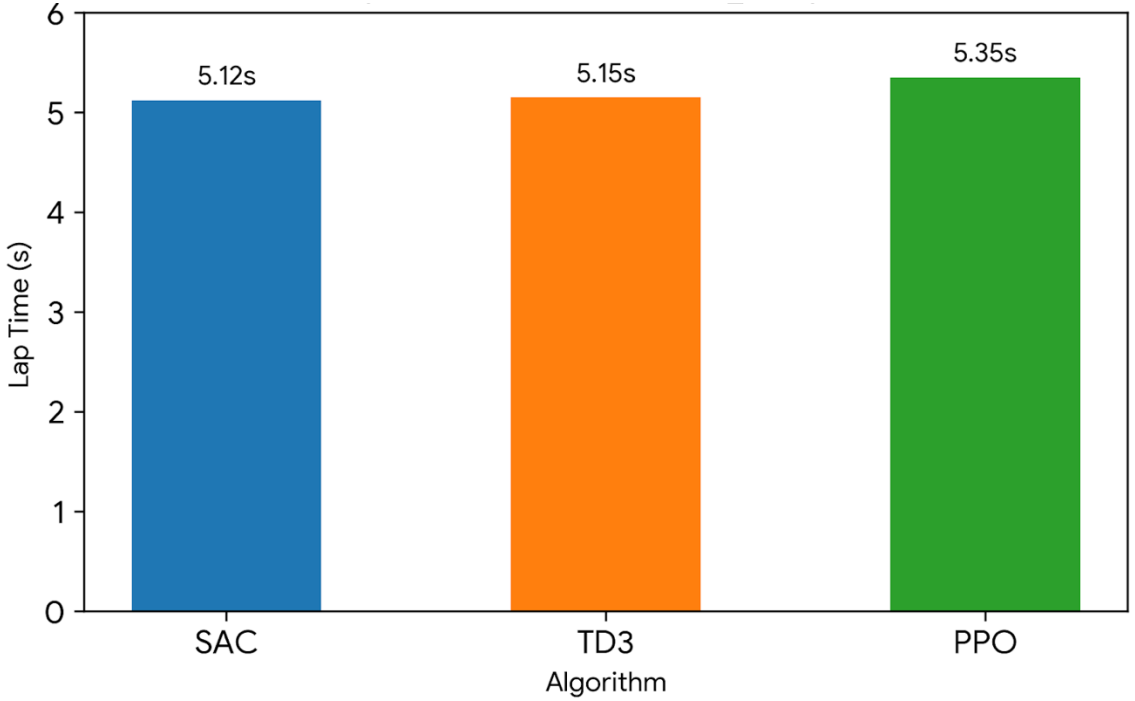
- Init $w_1, w_2, \theta, \alpha, \mathcal{H}_{min}$
- Collect one episode with *the policy* π_θ, add it to the replay buffer D
  - For K iterations:
    - Sample a batch of transitions $\{(s_i, a_i, s_i', r_i)\}^B$ from D
    - Calculate target (using target networks $\overline{w}$): $y_i = r_i + \gamma \min_k Q(s_i', a', \overline{w_k}) - \alpha\gamma \log \pi(a'|s')$
    - Compute Critic Losses: $L_Q(w_k) = \frac{1}{B}\sum_i \left(y_i - Q(s_i, a_i, w_k)\right)^2$
    - Compute Policy Loss: $L_\pi(\theta) = \frac{1}{B}\sum_i Q(s_i, \pi(s_i, \epsilon_i), w) - \alpha \log \pi(s_i, \epsilon_i)$
    - Compute Temperature Loss: $L(\alpha) = \sum_i -\alpha \log \pi(s_i, \epsilon_i) - \alpha\mathcal{H}_{min}$
    - Make one gradient *descent* step for the critics: $w_k \leftarrow w_k - \frac{dL_q(w_k)}{dw_k}$
    - Make one gradient *ascent* step for the policy: $\theta \leftarrow \theta + \frac{dL_\pi}{d\theta}$
    - Make one gradient *descent* step for the temperature: $\alpha_k \leftarrow \alpha_k - \frac{dL(\alpha)}{d\alpha}$
    - $\overline{w} \leftarrow m\overline{w} + (1-m)w \qquad (m = 0.999)$

Notable Differences Between SAC and TD3

- SAC policies balance entropy with the task objective (through the temperature α). TD3 only optimizes the task objective.
- SAC does not use a target policy. It always samples from the current policy
  - Otherwise, entropy management would be a disaster.
- SAC does not have the target policy smoothness trick
  - Since it accomplishes that by making the policy stochastic.
- SAC does not use the delayed policy trick

How Much Does this Matter In Practice?





- Algorithms have an impact on performance.
- Understanding them will help you predict when and how much.
- We’re still missing something to unify them!

Connection Between SAC and Policy Gradient

- Q-Learning methods and Policy Gradient methods are connected.
- Prior work [Shulman et al, 2017] has shown that, when
  - Restricting the actor class to a Boltzmann policy
  - Optimizing an entropy-regularized RL Loss
  - Learning on-policy

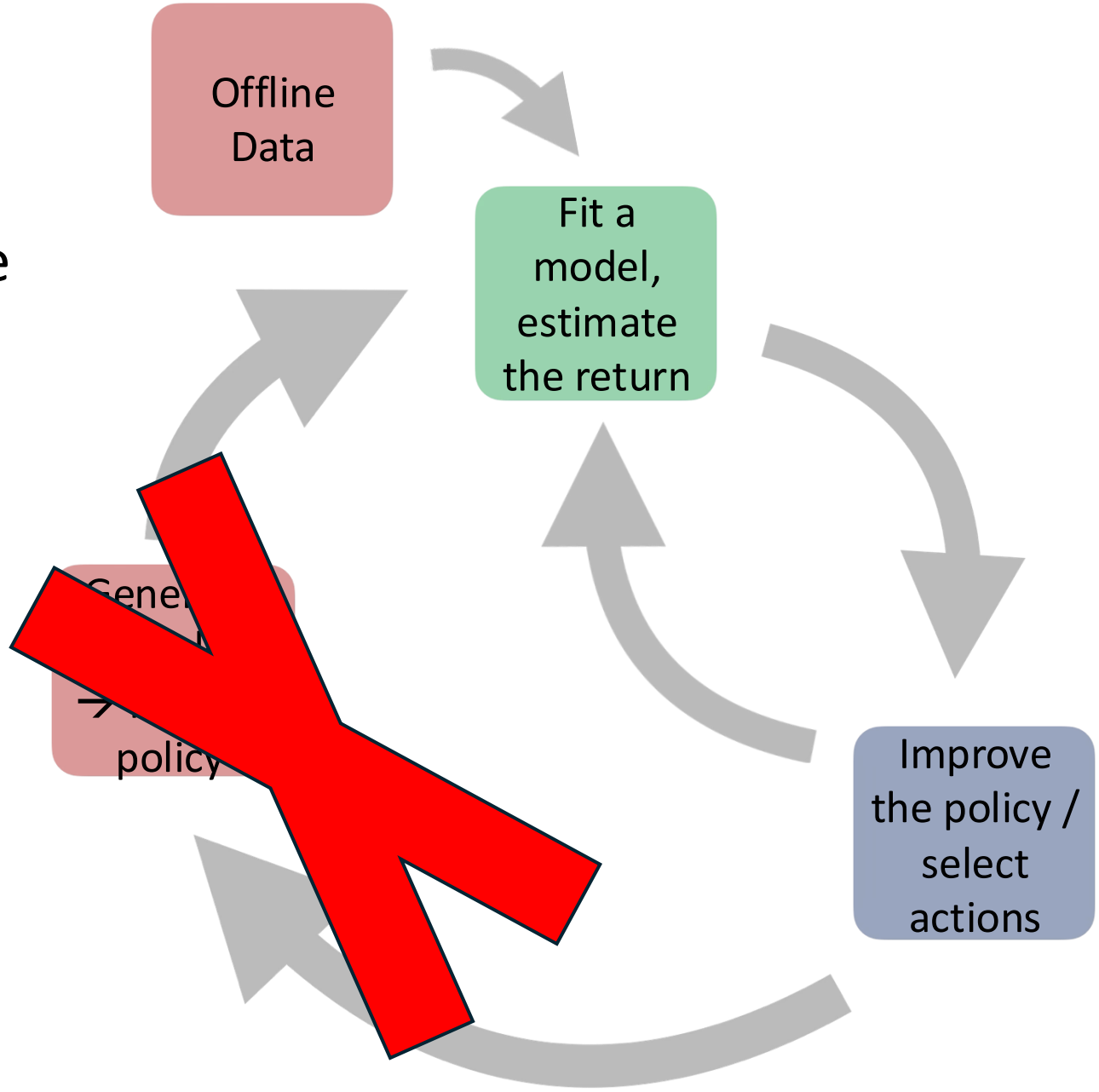
- Then the gradient updates of Q-learning/SARSA and Policy gradient are equivalent:

$$\nabla_\theta \mathbb{E}_{t,s_t,a_t\sim\pi}\left[\frac{1}{2}\left\|Q_\theta(s_t,a_t)-y_t\right\|^2\right]\Big|_{\pi=\pi_\theta}$$

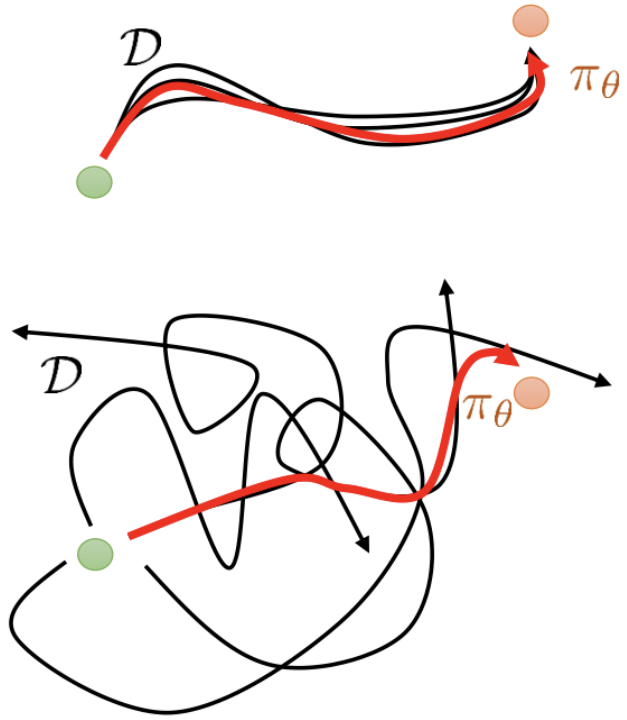
$$=\mathbb{E}_{t,s_t,a_t\sim\pi}\Big[\underbrace{-\tau\nabla_\theta\log\pi_\theta(a_t\mid s_t)\Delta_t+\tau^2\nabla_\theta D_{\mathrm{KL}}\left[\pi_\theta\parallel\overline{\pi}\right](s_t)}_{\text{policy grad}}+\underbrace{\nabla_\theta\tfrac{1}{2}\left\|V_\theta(s_t)-\hat{V}_t\right\|^2}_{\text{value function grad}}\Big]\Big|_{\pi=\pi_\theta}$$

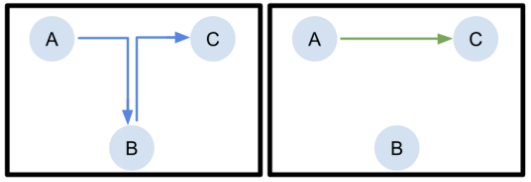
Offline RL

The most extreme example of off-policy learning!



What do we expect Offline RL methods to do?

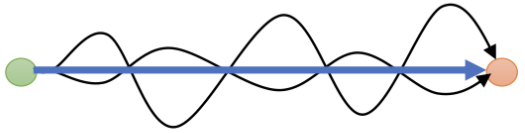




"Macro-scale" stitching

But this is just the clearest example!

"Micro-scale" stitching:

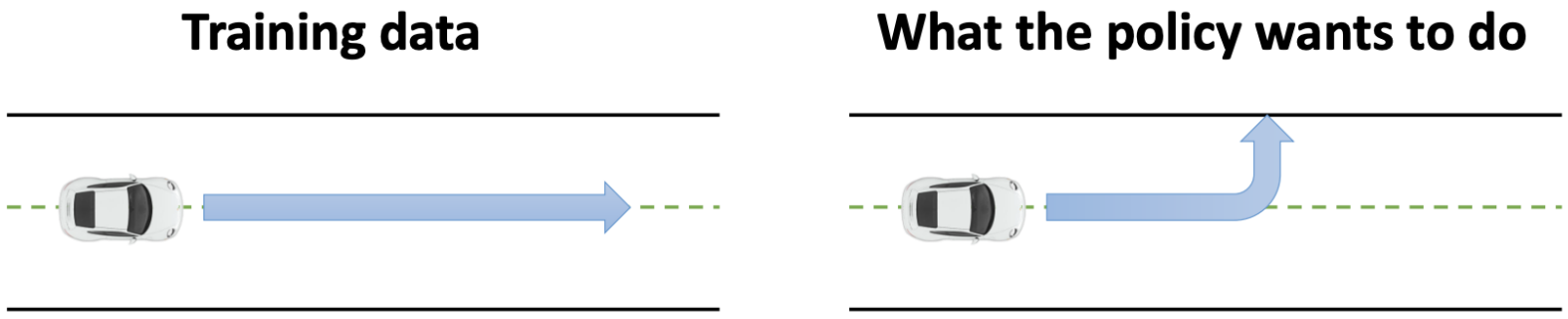


This can be useful for initializing a policy and then quickly fine-tuning it with online experience!

Slide from Sergey Levine

Fundamental Problem of Offline RL

- Counterfactual queries:



- How do we know if this is good or bad if we *did not see it in the data*?
  - In online RL (on-policy or off-policy), this is addressed by running the policy in the environment.

Slide from Sergey Levine

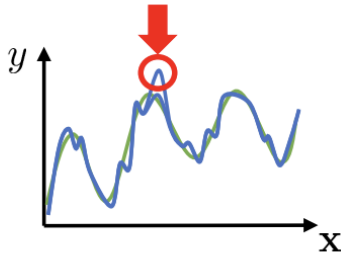
Addressing the Problem with Policy Constraints

- There is a large literature on offline RL.
  - See [Levine et. al, 2020] for a nice tutorial and (slightly out of date) survey.
- Most methods rely on one of these ideas:

$$D_{\mathrm{KL}}(\pi(\mathbf{a}|\mathbf{s})\|\pi_\beta(\mathbf{a}|\mathbf{s})) \leq \epsilon$$

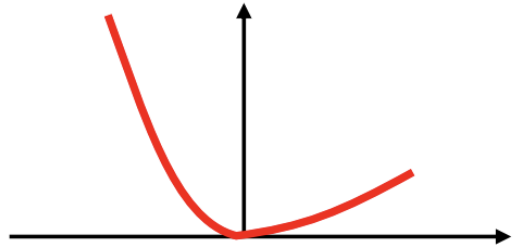
policy constraints

train the policy so that it stays close to data



pessimism (e.g., CQL)

train the Q-function so that it doesn't overestimate



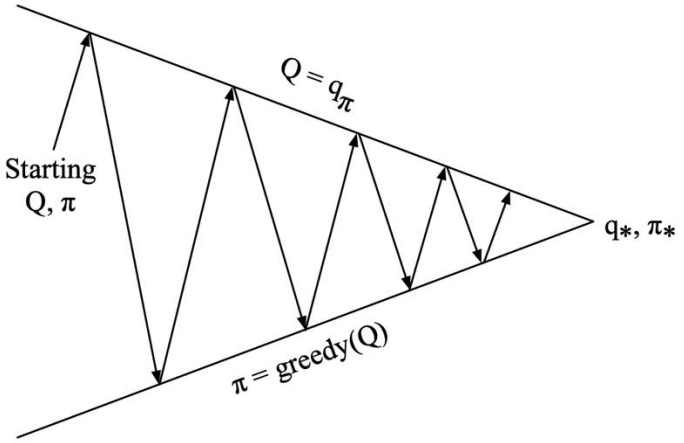
avoid OOD actions in updates (e.g., IQL)

set up the whole algorithm to avoid out-of-sample actions

Slide from Sergey Levine

Tips for Effective Learning with Q Functions

- Q functions can be challenging to stabilize, particularly when using neural networks.
  - Start with simple tests to make sure the implementation is correct.
- Large replay buffers help improve stability
  - It then becomes similar Q iteration



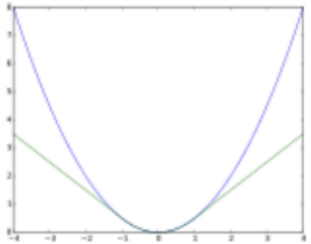
- Q functions often take a lot of experience/time to stabilize. It helps to be patient.

Slide from John Shulman

Tips for Effective Learning with Q Functions

- The gradient of the TD error can be big;
  - Clip gradients or use Huber loss.

$$L(x) = \begin{cases} x^2/2 & \text{if } |x| \leq \delta \\ \delta|x| - \delta^2/2 & \text{otherwise} \end{cases}$$



- Often, it helps a lot to schedule the hyperparameters (learning rates, exploration noise, etc.)

- Run *multiple seeds.* It can be very inconsistent between runs.

Slide from John Shulman

Summary: Off-Policy Learning with Q fn

- Extending Q-Learning/Sarsa to function approximation introduces new challenges related to instability and efficiency.

- Most methods (particularly for the continuous case) rely on SGD of the Bellman Error, which is, in general, not learnable.
  - And they account for it with heuristics to balance exploration and exploitation

- Empirically, more sample-efficient than off-policy policy gradient methods (e.g., PPO)
  - *Maybe* because of the ability to use a larger replay buffer of old policies.
  - But final performance could be similar.
  - And, if using a fast simulator, off-policy PG can converge in lower wall-clock time

Summary: Off-Policy Learning

- It holds great potential because it decouples *learning* from *behavior*.
- This whole area is relatively new and unsettled.
- The best way to achieve it is still a mystery.