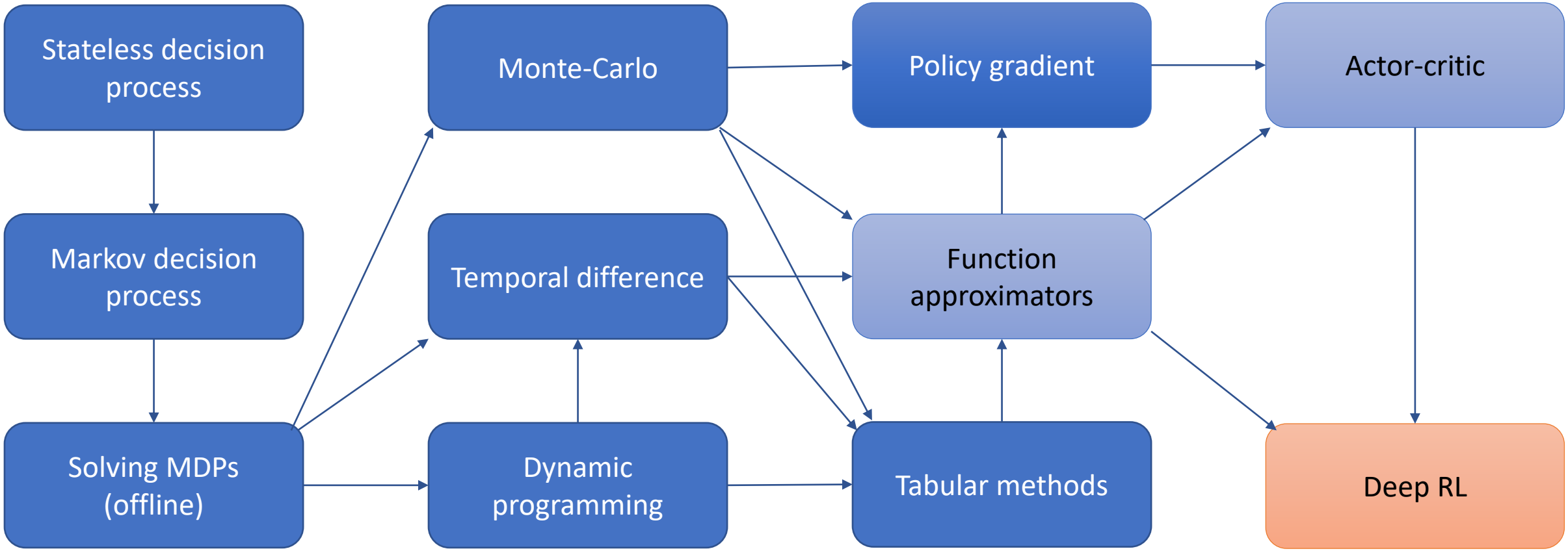
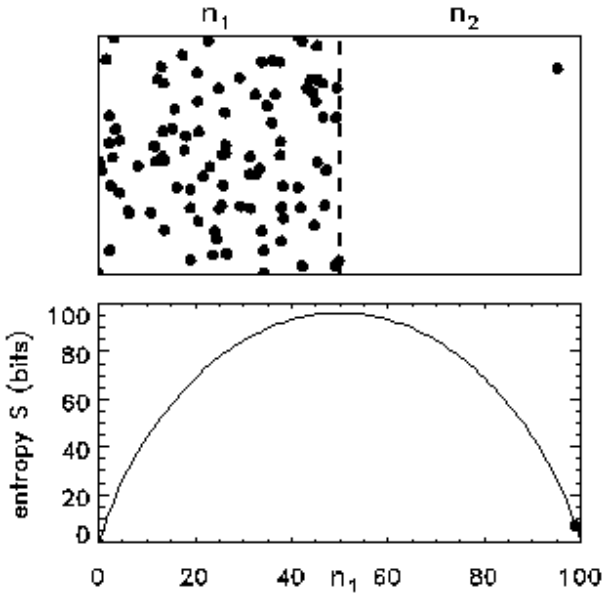
CSCE-642 Reinforcement Learning
Soft Actor-Critic

Instructor: Guni Sharon

CSCE-689, Reinforcement Learning



Entropy



en·tro·py

/ˈentrəpē/

noun

1. PHYSICS

   a thermodynamic quantity representing the unavailability of a system's thermal energy for conversion into mechanical work, often interpreted as the degree of disorder or randomness in the system.

2. lack of order or predictability; gradual decline into disorder.

   "a marketplace where entropy reigns supreme"

   synonyms: deterioration, degeneration, crumbling, decline, degradation, decomposition, breaking down, collapse; More

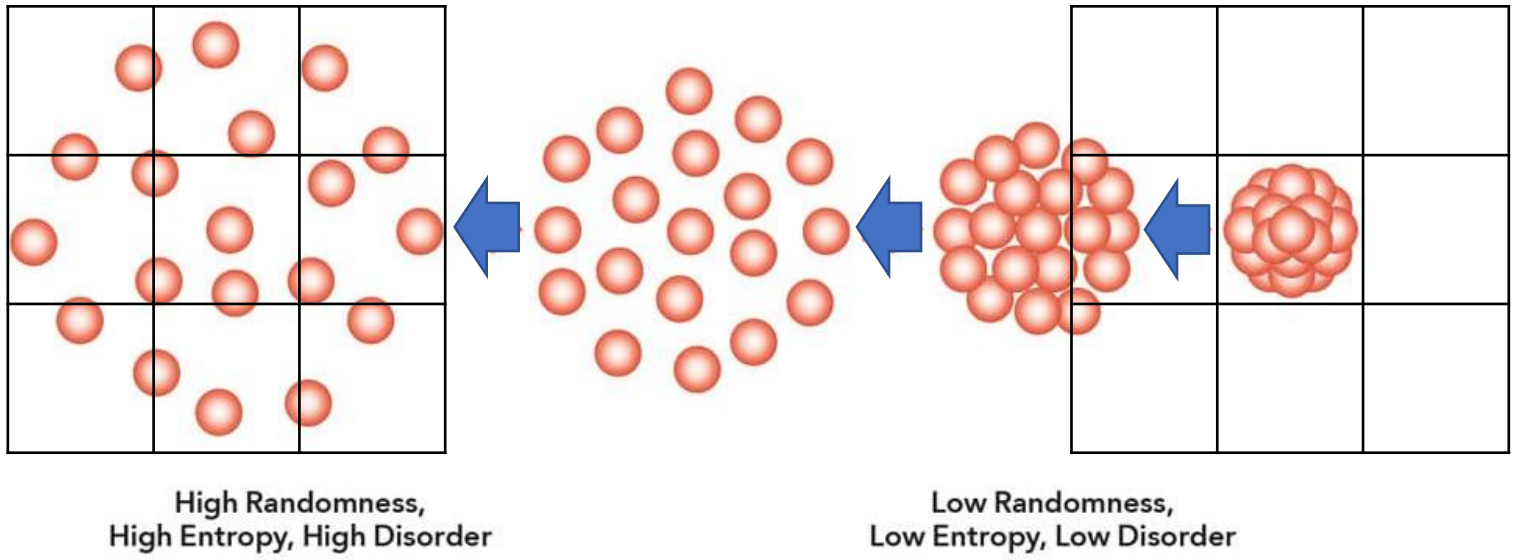
Disorder is the most probable state

- What is the probability of getting a specific order of cards after shuffling a deck?
  - 1 over all possible configurations (52!)
    - 52!=80,658,175,170,943,878,571,660,636, 856,403,766,975,289,505,440, 883,277,824,000,000,000,000
- What is the probability of getting a configuration where cards are ordered based on value?
  - Number of configurations where cards are ordered over (52!)
- Why is it not probable to shuffle cards and get ordered values?

Entropy

- The 2nd law of thermodynamics states that the total entropy of an isolated system can never decrease over time. Isolated systems spontaneously evolve towards thermodynamic equilibrium, i.e., the state with maximum entropy

How much information is required to represent each state?



Entropy in Information Theory

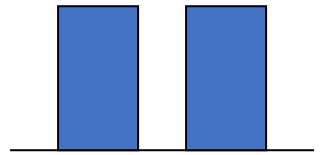
- Information answers questions
  - The more uncertain about the answer initially, the more information in the answer
  - Scale: bits
    - Answer to Boolean question with prior <1/2, 1/2>? — 1 bit
    - Answer to 4-way question with prior <1/4, 1/4, 1/4, 1/4>? — 2 bits
    - Answer to 4-way question with prior <0, 0, 0, 1>? — 0 bits
    - Answer to 3-way question with prior <1/3, 1/3, 1/3>? — 1+0.667x1=1.667 bits
    - Answer to 3-way question with prior <1/2, 1/4, 1/4>? — 1+0.5x1=1.5 bits

- High entropy *p*-way question:
  - A uniform distribution with probability 1/p
  - Requires $O(\log_2 1/p)$ bits

Entropy

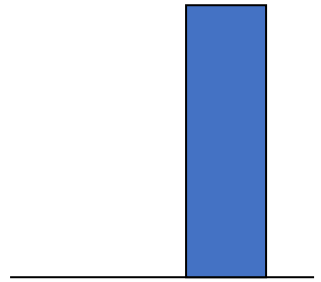
- General answer: if prior is <$p_1, \ldots, p_n$>:
  - Information is the expected number of bits

$$H(\langle p_1, \ldots, p_n \rangle) = E_p \log_2 1/p_i = \sum_{i=1}^{n} -p_i \log_2 p_i$$



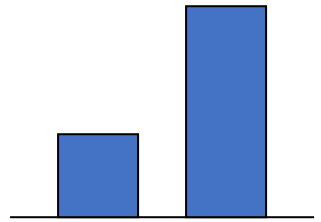
1 bit

- Also called the entropy of the distribution
  - More uniform = higher entropy
  - More values = higher entropy
  - More peaked = lower entropy



0 bits

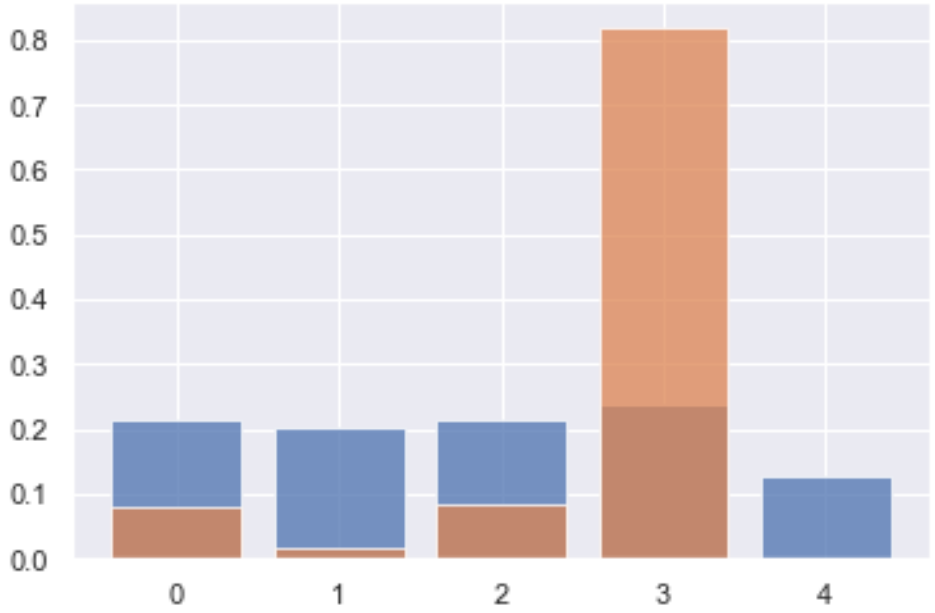
- Can also be computed over a continuous distribution
  - $\mathcal{H}(P) = -\int p_i \log p_i \, di$

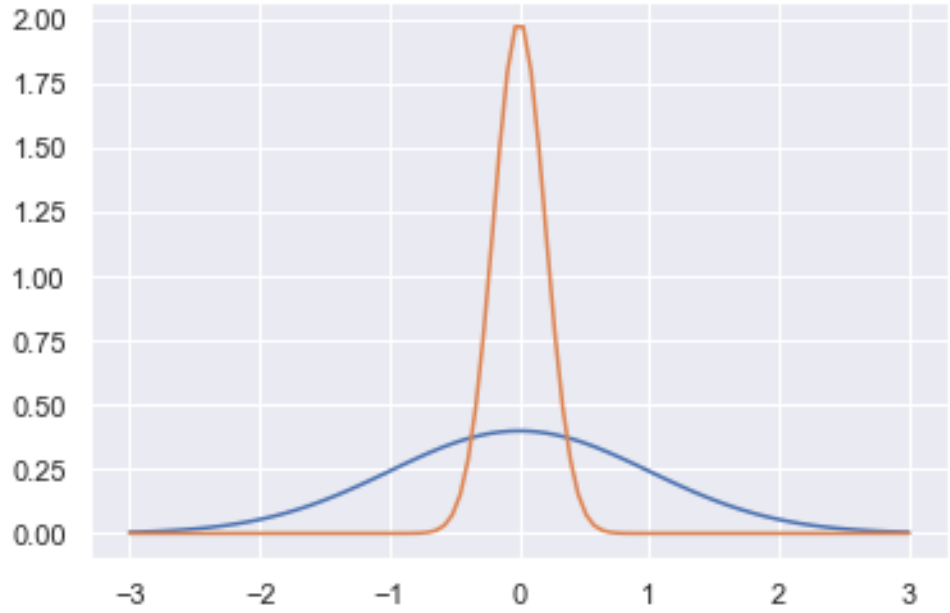


0.5 bit

Entropy

- Categorical (left), and Gaussian (Right) distributions. Orange shows low-entropy distributions, while blue shows high-entropy distributions.





Picture credit: Arthur Juliani

Maximum Entropy RL objective

- $\pi^* = \arg\max_{\pi} \mathbb{E}_{s_t, a_t \sim \pi}[\sum_{t=1}^{T} R(s_t, a_t)] \;\; S.T. \; \mathbb{E}_{s_t \sim \pi}\left[\mathcal{H}\left(\pi(\cdot, s_t)\right)\right] > b$
- $= \arg\max_{\pi} \mathbb{E}_{s_t, a_t \sim \pi}\left[\sum_{t=1}^{T} R(s_t, a_t) + \alpha\mathcal{H}\left(\pi(\cdot, s_t)\right)\right]$

- Several conceptual and practical advantages
  - The policy is incentivized to explore more widely, while giving up on clearly unpromising avenues
  - The policy can capture multiple modes of near-optimal behavior
  - In problem settings where multiple actions seem equally attractive, the policy will commit equal probability mass to those actions
- *"adding the entropy of the policy π to the objective function improved exploration by discouraging premature convergence to suboptimal deterministic policies. This technique was originally proposed by (Williams & Peng, 1991)"* *Mnih et al., Asynchronous Methods for Deep Reinforcement Learning*

Soft Actor Critic [https://arxiv.org/pdf/1812.05905.pdf]

- Defines an off-policy actor-critic algorithm based on the maximum entropy principal
- Attempts to address two major challenges in model-free deep RL: high sample complexity and brittleness to hyperparameters
- The actor aims to simultaneously maximize expected return and policy entropy; that is, to succeed at the task while acting as randomly as possible

1. An actor-critic architecture with separate policy and value function networks
2. Off-policy formulation that enables reuse of previously collected data for efficiency
3. Policy entropy maximization to encourage stability and exploration

Soft Actor Critic

- Off-policy actor-critic training with a stochastic actor $\pi(a|s)$
- SAC is not a true actor-critic algorithm: the Q-function is estimating the optimal Q-function, and the actor does not directly affect the Q-function except through the data distribution
  - This fact allows It to be “off-policy”
- Objective function:
  - $\pi^* = \text{argmax}_\pi \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}\left[R(s_t,a_t) + \alpha\mathcal{H}\left(\pi(\cdot\,|s_t)\right)\right]$
- α is the temperature parameter that determines the relative importance of the entropy term versus the reward, and thus controls the stochasticity of the optimal policy
  - The conventional objective can be recovered in the limit as $\alpha \to 0$

Soft policy iteration

- Policy iteration: interleave between *policy evaluation* and *policy improvement*
- **Policy evaluation**: compute state, action values
  - $Q(s_t, a_t) = r_t + \gamma \mathbb{E}_{s_{t+1} \sim p}[V(s_{t+1})]$
  - $V(s_{t+1}) = \mathbb{E}_{a_{t+1} \sim \pi}[Q(s_{t+1}, a_{t+1})] + \alpha \mathbb{E}_{a_{t+1} \sim \pi}[-\log \pi(a_{t+1}|s_{t+1})]$

Entropy

CONVERGENCE TO soft-Q_π GUARANTEED

  - $= \mathbb{E}_{a_{t+1} \sim \pi}[Q(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1})]$

*for the tabular case and assuming all state-action pairs are visited infinitely often

Soft policy iteration

Softmax: $p_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$

- **Policy improvement**: update the policy towards the exponential of the old soft Q values (softmax)
- Guaranteed to result in an improved policy in terms of its soft value
- $\pi_{\text{new}} = \arg\min_{\pi'} \mathrm{D}_{\mathrm{KL}}\left(\pi'(\cdot\,|s_t) \parallel \frac{\exp\left(\frac{1}{\alpha} Q^{\pi_{\text{old}}}(s_t,\cdot)\right)}{Z^{\pi_{\text{old}}}(s_t)}\right)$
- That is, the desired new policy should be as similar as possible (can't equal due to approximation limits) to the softmax distribution over the Q values from the policy evaluation step of the prev policy
- The partition function, $Z^{\pi_{\text{old}}}(s_t)$, normalizes the distribution
  - Does not contribute to the gradient with respect to the new policy
- Soft policy iteration (Policy evaluation ⇄ Policy improvement) provably converge to the optimal maximum entropy policy
  - For the tabular case

Soft Actor Critic

- = Soft policy iteration + function approximators for both the soft Q-function and the policy
  - $Q(s, a; \theta)$ is a DNN with state features and action as input and action value as output
  - $\pi(a|s; \phi)$ is a DNN with state features as input and the output is the mean vector and covariance matrix for all action variables
    - Multivariate **Gaussian** distribution
    - An action is defined by a set of continues variables
- Instead of running evaluation and improvement to convergence, optimize both networks with stochastic gradient descent

Soft Critic

- The soft Q-function parameters are trained to minimize the squared loss of the soft Bellman residual
- $L(\theta) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q(s_t,a_t;\theta) - \left(r(s_t,a_t) + \gamma\mathbb{E}_{(s_{t+1})\sim p}[V(s_{t+1};\bar{\theta})]\right)\right)^2\right]$
  - $V(s_{t+1};\bar{\theta}) = \mathbb{E}_{a_{t+1}\sim\pi}[Q(s_{t+1},a_{t+1};\bar{\theta}) - \alpha\log\pi(a_{t+1}|s_{t+1};\phi)]$
- $\nabla_\theta L(\theta) = \nabla_\theta Q(s_t,a_t;\theta)\left(Q(s_t,a_t;\theta) - \left(r(s_t,a_t) + \gamma\left(Q(s_{t+1},a_{t+1};\bar{\theta}) - \alpha\log\pi(a_{t+1}|s_{t+1};\phi)\right)\right)\right)$
- The update makes use of a target soft Q-function with parameters $\bar{\theta}$
  - Exponentially moving average of the soft Q-function weights
  - Stabilizes training

Soft actor

- The policy parameters can be learned by directly minimizing the expected KL-divergence from the Q-based policy π'
  - $\mathrm{D}_{\mathrm{KL}}\left(\pi(\cdot\,|s) \parallel \pi'(\cdot\,|s)\right) = \int \pi(a|s) \log\left(\frac{\pi(a|s)}{\pi'(a|s)}\right) da$
  - $\arg\min_{\phi} \int \pi(a|s) \log\left(\frac{\pi(a|s)}{\pi'(a|s)}\right) da = \arg\min_{\phi} \mathbb{E}_{a\sim\pi}\left[\log\left(\frac{\pi(a|s)}{\pi'(a|s)}\right)\right]$

$p(a)$

- Minimize D_{KL} over all visited states or equivalently over the expected state
  - $\arg\min_{\phi} \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}\left[\log\left(\frac{\pi(a|s)}{\pi'(a|s)}\right)\right]\right]$

Actor training

The target policy π' is a softmax over Q

- $\arg\min_{\phi} \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}\left[\log\left(\frac{\pi(a|s)}{\pi'(a|s)}\right)\right]\right]$
  - $= \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}[\log\pi(a|s) - \log\pi'(a|s)]\right]$
- $\pi'(a|s) = \frac{\exp\left(\frac{1}{\alpha}Q(s,a)\right)}{Z^{\pi_{\text{old}}}(s_t)}$

The constant log-partition function can be ignored with no impact on the argmin

  - $= \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}\left[\log\pi(a|s) - \left(\cancel{\log\exp}\left(\frac{1}{\alpha}Q(s,\ a)\right) - \log Z(s_t)\right)\right]\right]$
  - $= \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}\left[\log\pi(a|s) - \frac{1}{\alpha}Q(s,\ a)\right]\right]$
- Multiply by constant $\alpha > 0$ won't affect the argmin
- $= \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}[\alpha\log\pi(a|s) - Q(s,\ a)]\right]$

Actor training

- How should we minimize the (non-convex) actor loss:
  - $L(\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}[\alpha \log \pi(a|s) - Q(s,a)]\right]$
- SGD!
  - $\phi = \phi - \alpha\nabla\hat{L}(\phi)$
- π outputs a distribution (e.g., multivariate Gaussian), specifically for Gaussian it outputs a mean vector $\mu_\phi(s)$ and a covariance matrix $\sigma_\phi(s)$
- We know how to compute gradients of the loss function with respect to μ_θ and σ_θ (see 14.Actor-critic.pptx, slides 30-33)
- However, these gradients tend to be very noisy due to a noisy sampling process ($\pi(s)\sim\mathcal{N}\left(\mu_\phi(s), \sigma_\phi(s)\right)$)

Reparameterization trick

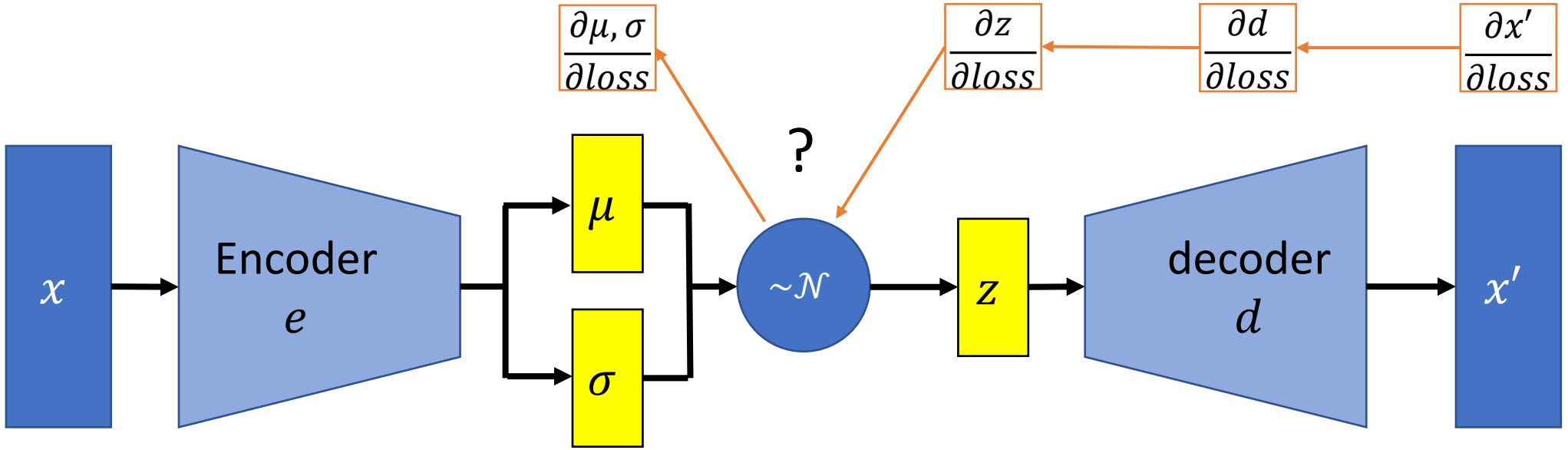
- $L(\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}[\alpha \log \pi(a|s) - Q(s,a)]\right]$
- Define a hard policy function: $a_t = f_\phi(\epsilon_t; s_t) = \mu_\phi(s_t) + \sigma_\phi(s_t)\epsilon_t$
- So now $L(\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[\alpha \log \pi\left(f_\phi(\epsilon; s)|s\right) - Q\left(s, f_\phi(\epsilon; s)\right)\right]$
- And, $\widehat{\nabla}_\phi L(\phi) = \nabla_\phi \alpha \log\left(\pi_\phi(a_t|s_t)\right) + \left(\nabla_{a_t} \alpha \log\left(\pi_\phi(a_t|s_t)\right) - \nabla_{a_t} Q(s_t, a_t)\right)\nabla_\phi f_\phi(\epsilon_t; s_t)$
- Where a_t is defined by $f_\phi(\epsilon_t; s_t)$, π_ϕ is defined implicitly by f_ϕ, ϵ_t is a noise vector sampled from some fixed distribution
- Assuming a fixed noise instead of a randomly sampled action results in a lower variance gradient estimator for both $\mu_\phi(s_t)$ and $\sigma_\phi(s_t)$

Training a VAE

Reconstruction loss

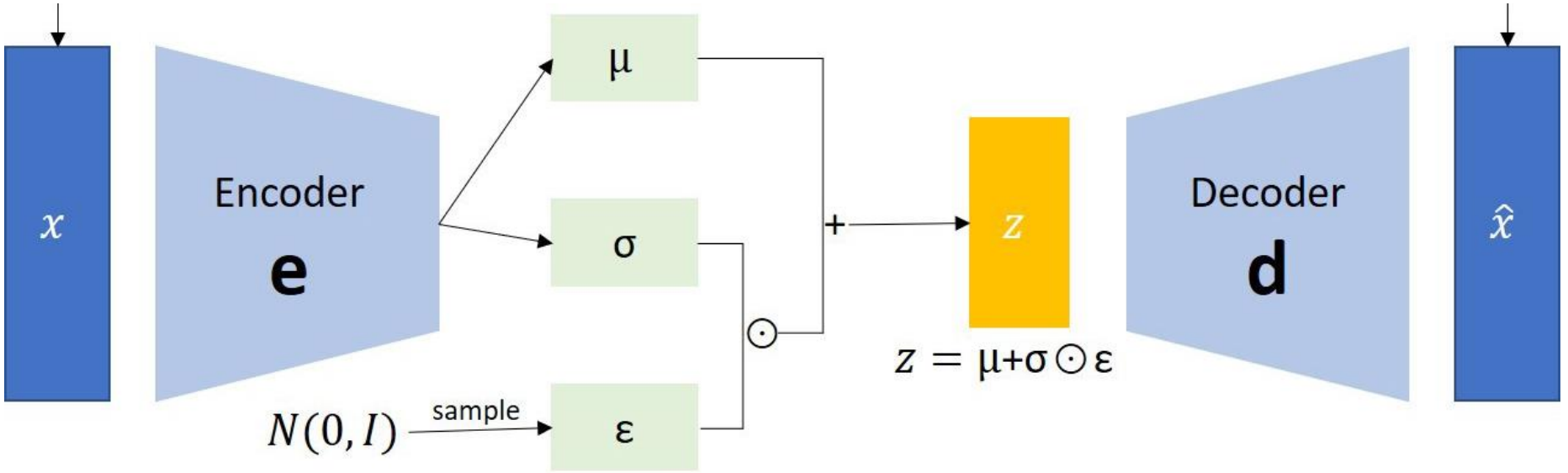
Regularization term

- $l(x) = (x - x')^2 + KL\big(\mathcal{N}(\mu_x, \sigma_x), \mathcal{N}(0, I)\big)$
  - Differentiable loss function
- Backprop!
- Derivative of sampling????



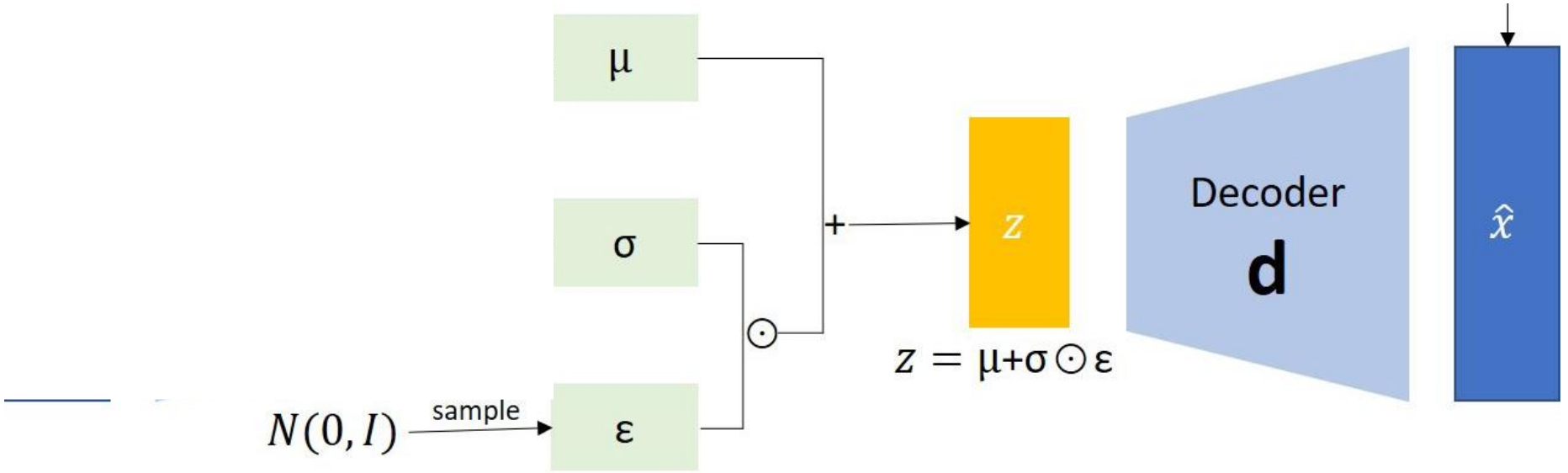
Reparameterization trick

- We cannot backpropagate through a sampling operation so we define a random noise input variable while keeping mu and sigma as constants



Generating new data

- Once we trained mu, sigma, and the decoder, we can create new samples of x
- Simply sample noise from $\mathcal{N}(0, I)$ to get a fabricated x



Auto entropy weight adjustment

- Choosing the optimal temperature (α) is non-trivial, and the temperature needs to be tuned for each task
- The magnitude of the reward differs not only across tasks, but it also depends on the policy, which improves over time during training
- The policy should be free to explore more in regions where the optimal action is uncertain, but remain more deterministic in states with a clear distinction between good and bad actions
- Automatically adjust the temperature on each update:
  - $\alpha \leftarrow \alpha - \lambda \widehat{\nabla}_{\alpha} J(\alpha)$
  - For derivation of $\widehat{\nabla}_{\alpha} J(\alpha)$ see https://arxiv.org/pdf/1812.05905.pdf

Algorithm 1 Soft Actor-Critic

Input: θ_1, θ_2, ϕ ▷ Initial parameters
$\bar{\theta}_1 \leftarrow \theta_1, \bar{\theta}_2 \leftarrow \theta_2$ ▷ Initialize target network weights
$\mathcal{D} \leftarrow \emptyset$ ▷ Initialize an empty replay pool
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ ▷ Sample action from the policy
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ ▷ Sample transition from the environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$ ▷ Store the transition in the replay pool
  end for
  for each gradient step **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ ▷ Update the Q-function parameters
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ Update policy weights
    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ Adjust temperature
    $\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i$ for $i \in \{1, 2\}$ ▷ Update target network weights
  end for
end for
Output: θ_1, θ_2, ϕ ▷ Optimized parameters

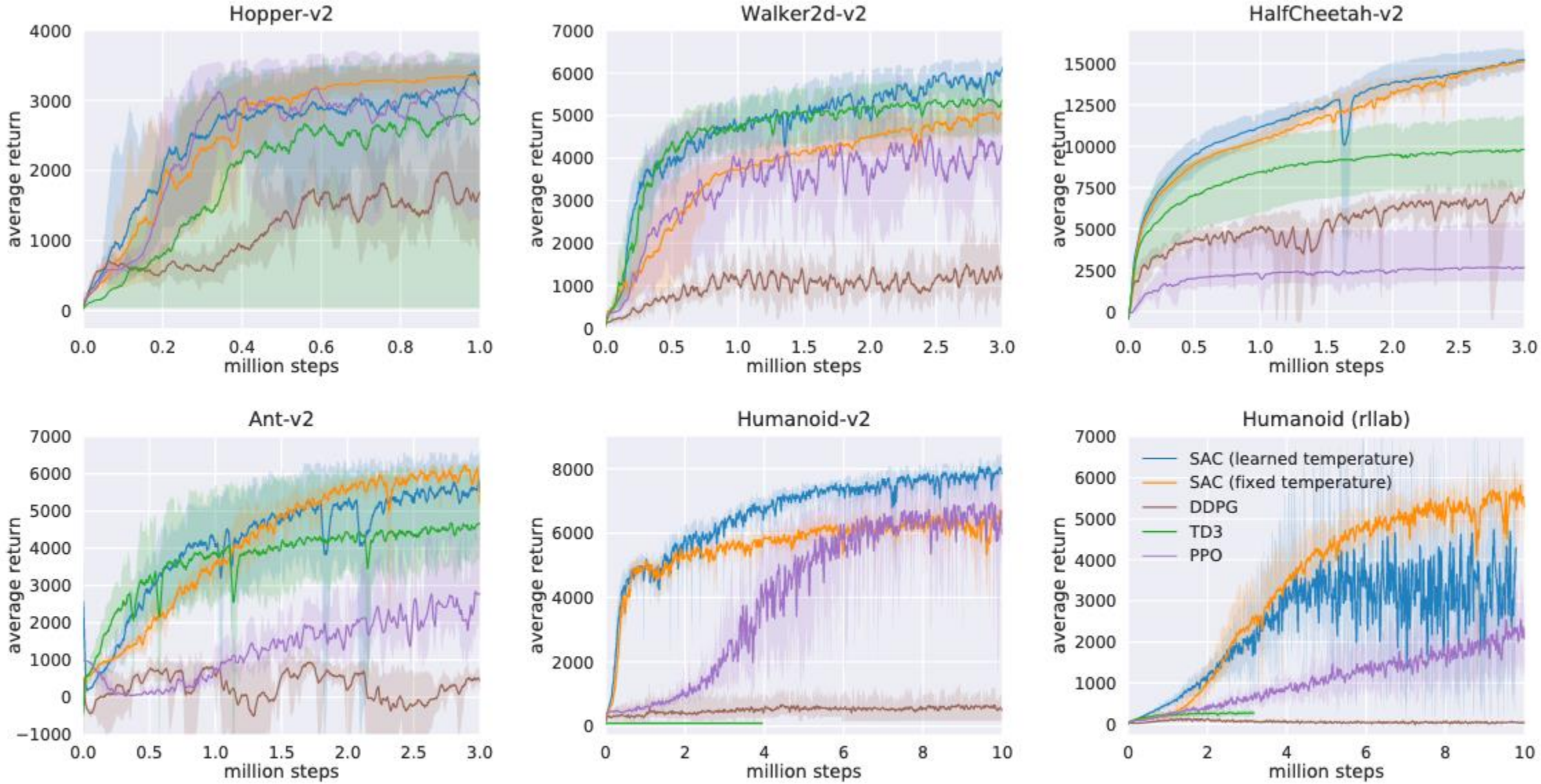


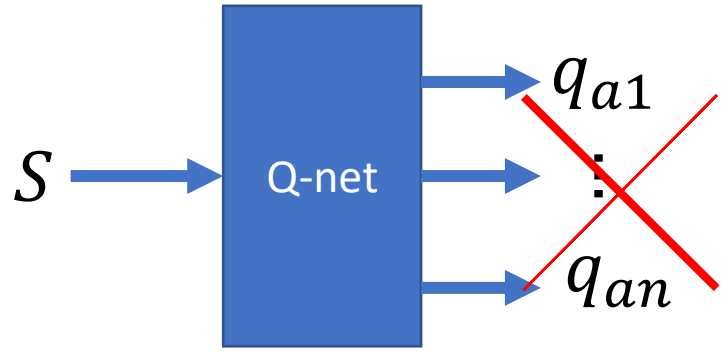
Figure 1: Training curves on continuous control benchmarks. Soft actor-critic (blue and yellow) performs consistently across all tasks and outperforming both on-policy and off-policy methods in the most challenging tasks.

Generalizing DQN to continuous actions

- DQN for discrete action space:

$$\pi(s) = \arg_a \max Q(s, a)$$

Train with squared TD loss



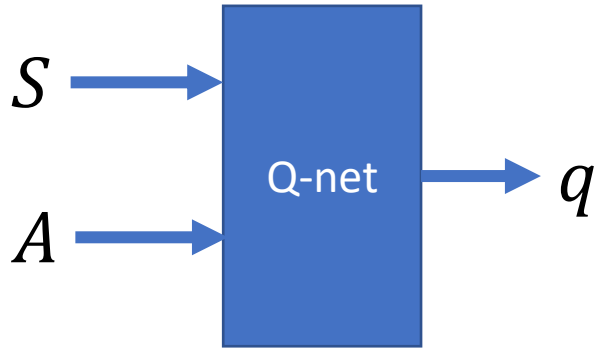
$$L(s, a) = (q - y)^2$$

$$y_i = r_i + \gamma \max_a Q(s_{i+1}, a)$$

How can we define a Q net for continues actions?

Q net for continues actions

- We can simply set the action as an input
- This raises two questions:



1. Training: $y_i = r_i + \gamma \max_a Q(s_{i+1}, a)$
2. Acting: $\pi(s) = \arg_a \max Q(s, a)$

- Deep Deterministic Policy Gradient [Lillicrap et al. 2016]

Train a deterministic actor $\pi(s) = \arg_a \max Q(s, a)$

$y_i = r_i + \gamma \max_a Q(s_{i+1}, a)$ becomes $y_i = r_i + \gamma Q\big(s_{i+1}, \pi(s)\big)$

- But how can we train $\pi(s)$?

Deep Deterministic Policy Gradient

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

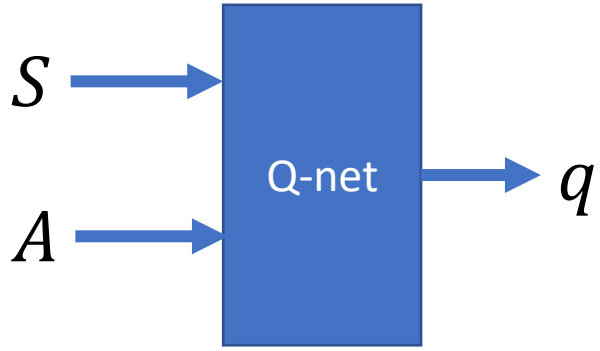
$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

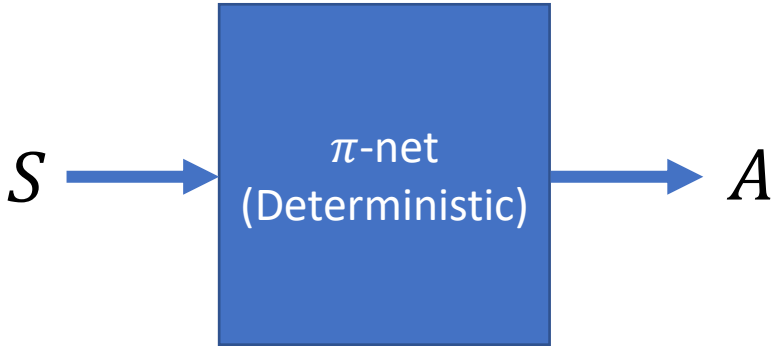
$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Train with squared TD loss



Train policy?



Deep Deterministic Policy Gradient

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

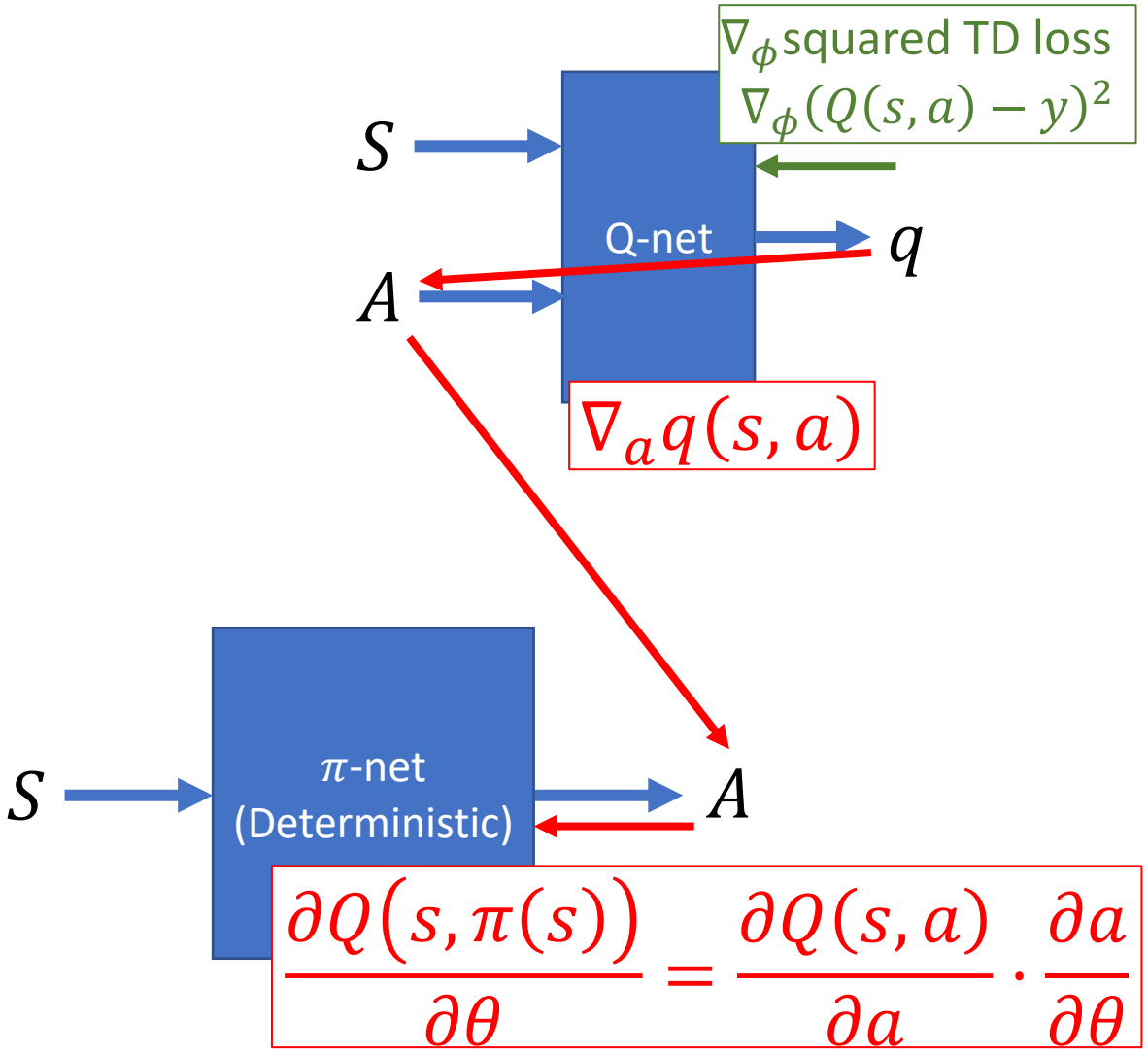
      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence



Deep Deterministic Policy Gradient

Algorithm 1 Deep Deterministic Policy Gradient

1: Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
3: **repeat**
4: Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5: Execute a in the environment
6: Observe next state s', reward r, and done signal d to indicate whether s' is terminal
7: Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
8: If s' is terminal, reset environment state.
9: **if** it's time to update **then**
10: **for** however many updates **do**
11: Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12: Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

13: Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

14: Update policy by one step of gradient ascent using

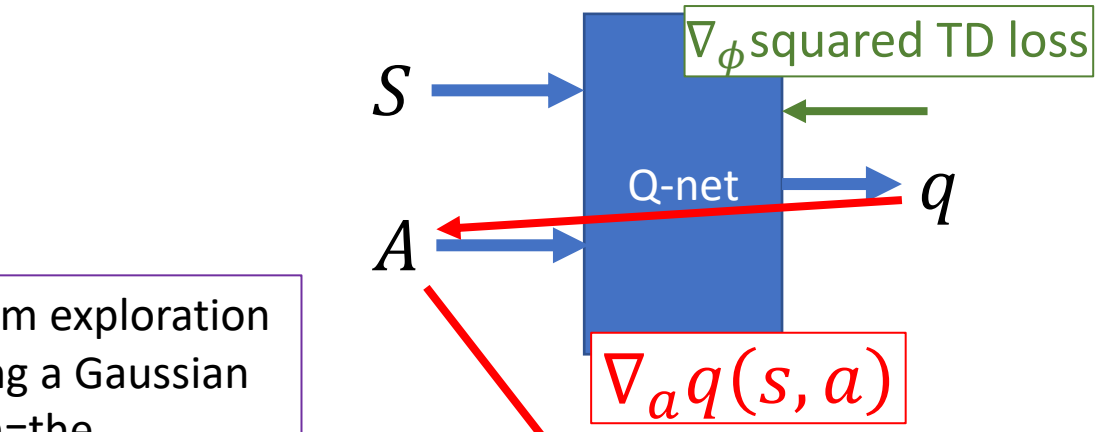
$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

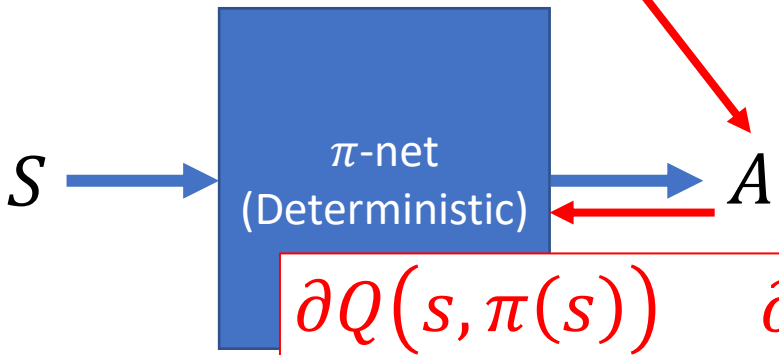
15: Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

16: **end for**
17: **end if**
18: **until** convergence

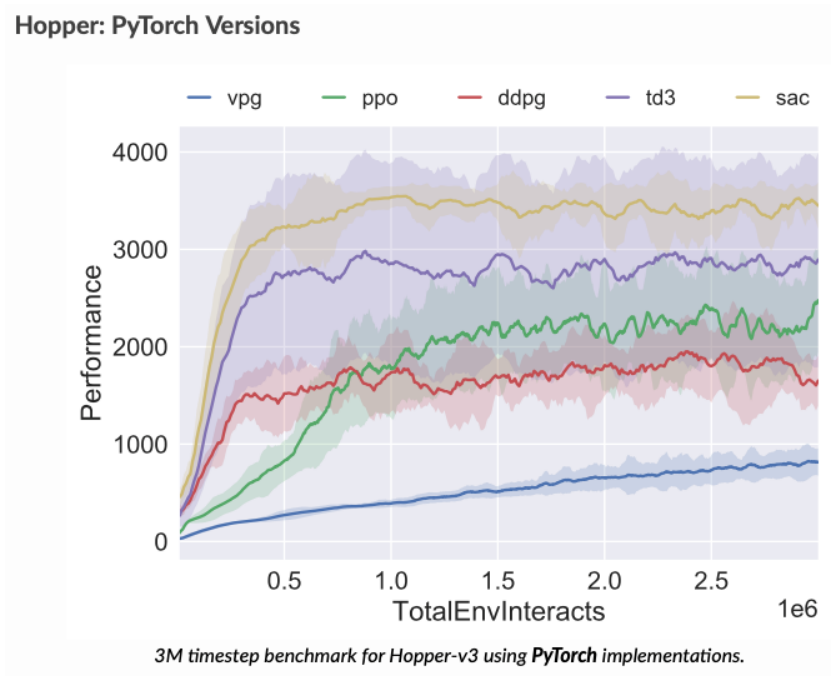
Add random exploration by sampling a Gaussian with mean=the deterministic action

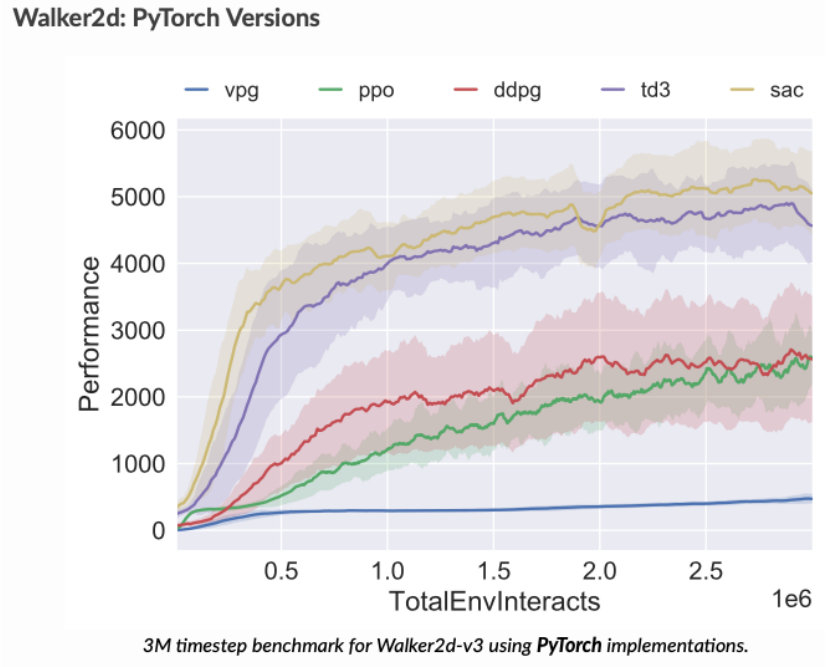


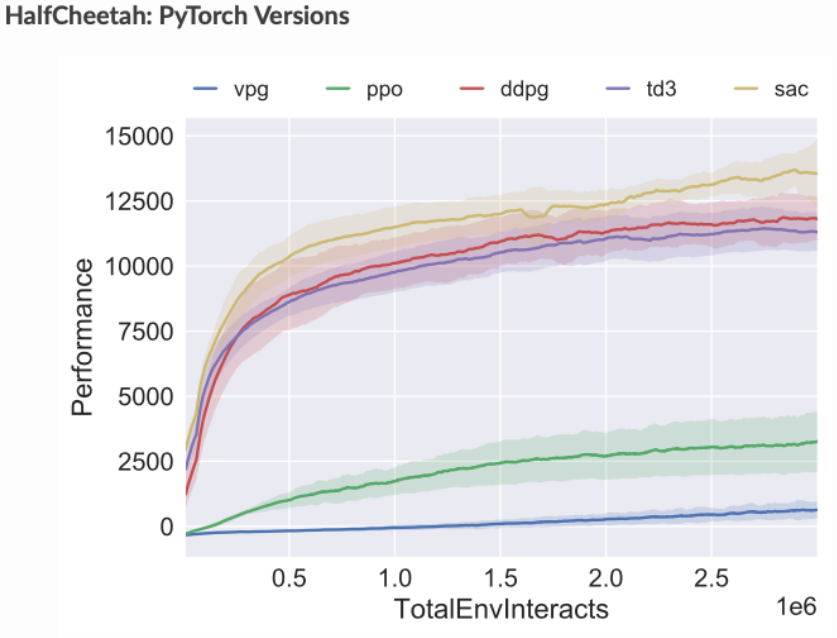


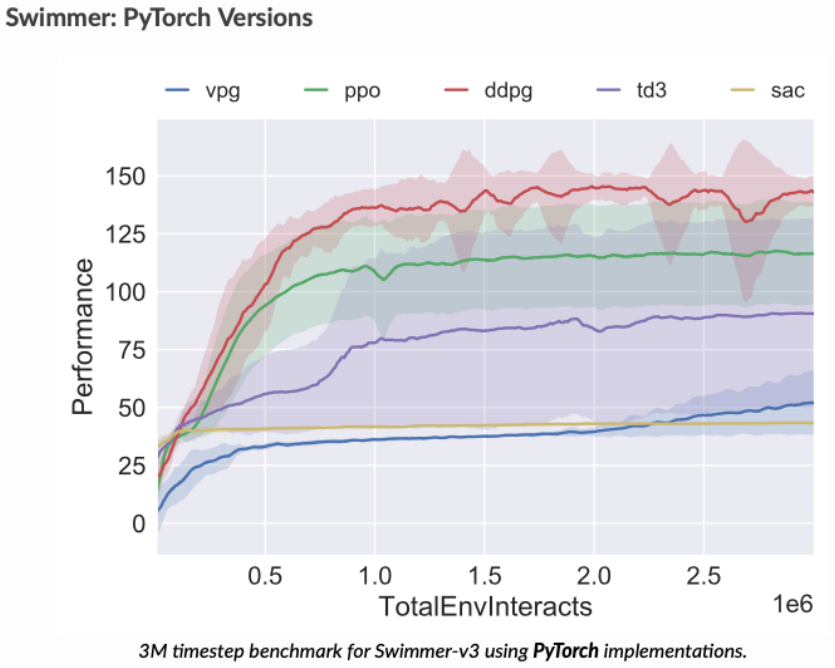
$$\frac{\partial Q\left(s, \pi(s)\right)}{\partial \theta} = \frac{\partial Q(s, a)}{\partial a} \cdot \frac{\partial a}{\partial \theta}$$

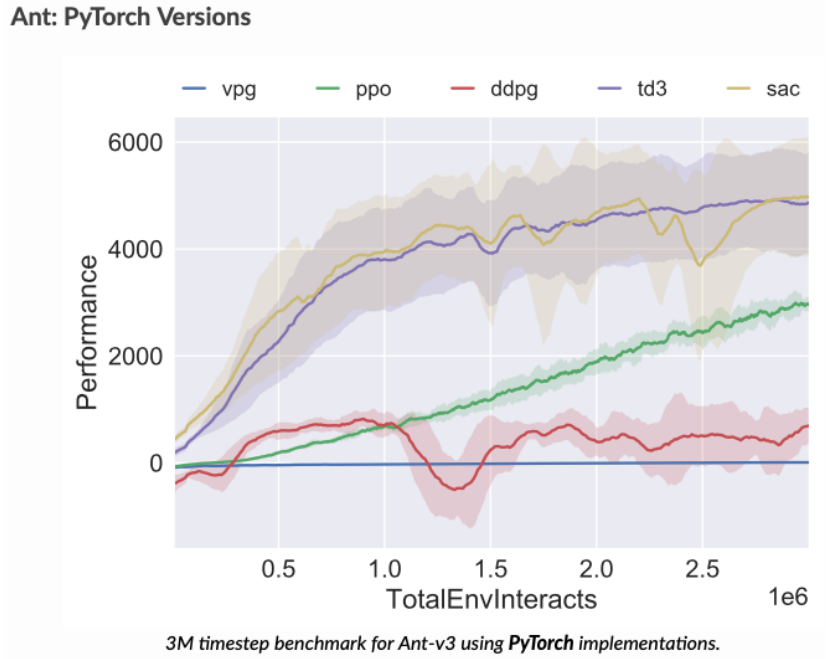
Results (from https://spinningup.openai.com/en/latest/spinningup/bench.html)



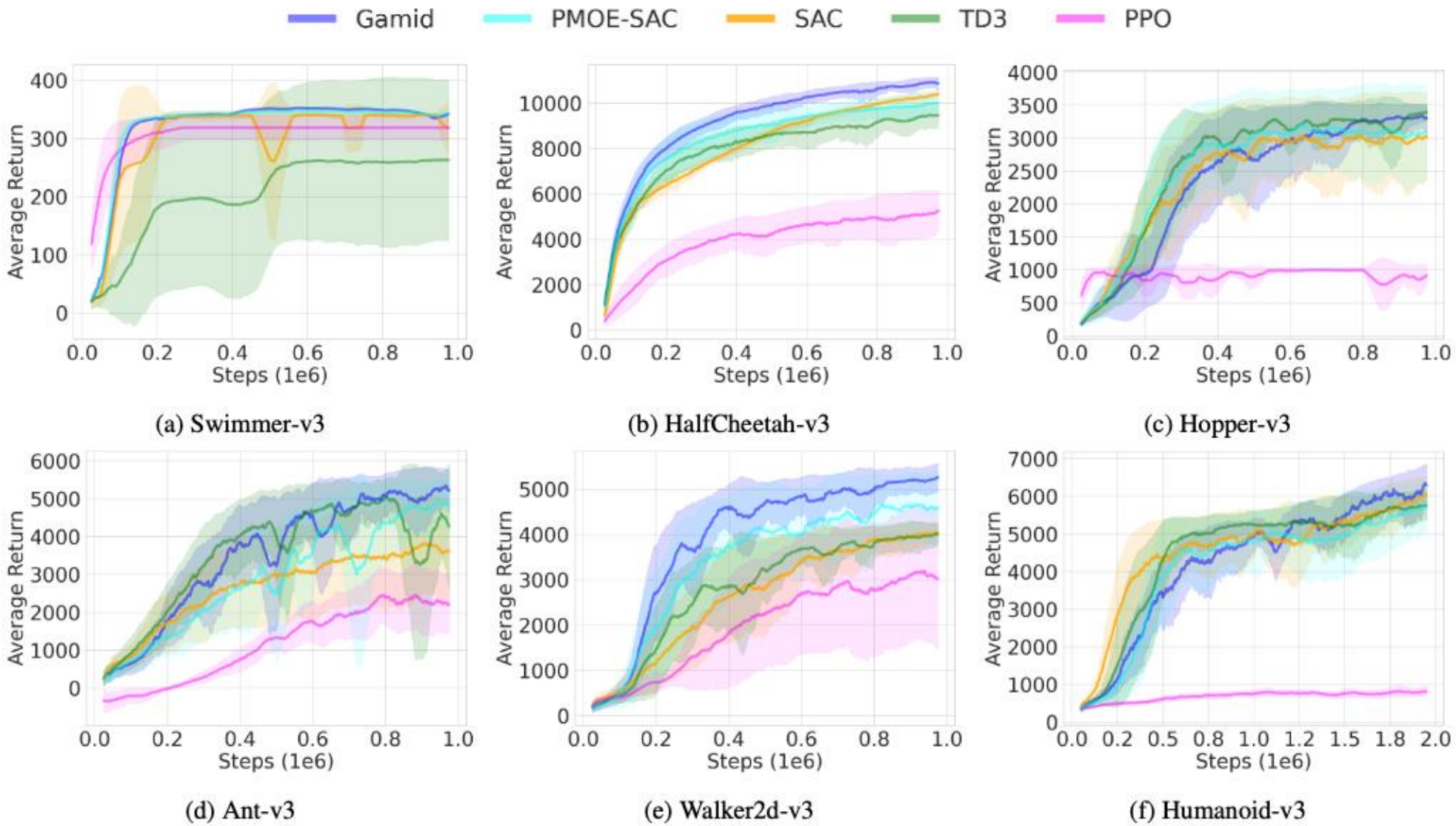








Gaussian Mixture Deterministic Policy Gradient



What did we learn (SAC)?

- Higher entropy for a policy -> more exploration
- More exploration -> higher chances of overcoming local optima
- It is worthwhile to encourage the policy to be of high entropy even at the cost of lower expected return
- Sampling a soft policy for an action results in a noisy gradient estimate
  - Instead, sample noise from a fixed distribution as input for a hard policy

What did we learn (DDPG)?

- Generalizing DQN to continuous action space is challenging
- Can’t enumerate over all possible actions so can’t compute argmax_a
- DDPG: “train a deterministic policy that approximates argmax_a”
- Train the deterministic policy using gradient ascend with respect to the approximated Q output

Extra reading

- https://arxiv.org/pdf/1812.05905.pdf
- https://spinningup.openai.com/en/latest/algorithms/sac.html
- https://towardsdatascience.com/soft-actor-critic-demystified-b8427df61665
- https://arxiv.org/pdf/1312.6114.pdf - **Reparameterization trick**

What next?

- **Lecture**: Imitation learning
- **Assignments**:
  - DDPG, by
  - A2C, by
  - REINFORCE, by
  - Deep Q-Learning, by
- **Quiz (on Canvas)**:
  - Soft Actor-Critic, by
- **Project**:
  - Literature survey, by