西湖大學
WESTLAKE UNIVERSITY

Deep Reinforcement Learning and Applications to Robotics

Donglin Wang

April 1st, 2024

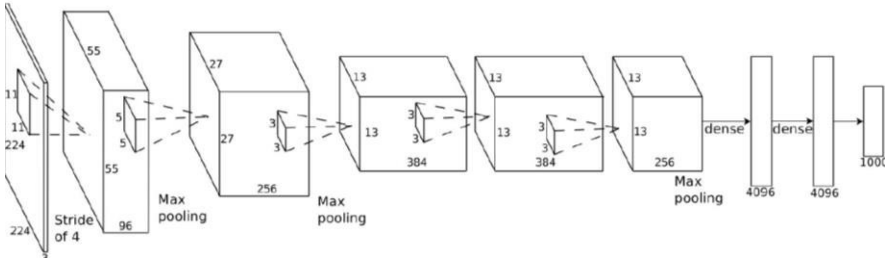
Outline

- ❑ Deep Reinforcement Learning
- ❑ Applications to Robotics

Deep Reinforcement Learning (DRL)

- Model-free DRL
- Model-based DRL
- Inverse Reinforcement Learning
- Offline Reinforcement Learning
- Large Pre-training DRL Model

Terminology & Notation of DRL

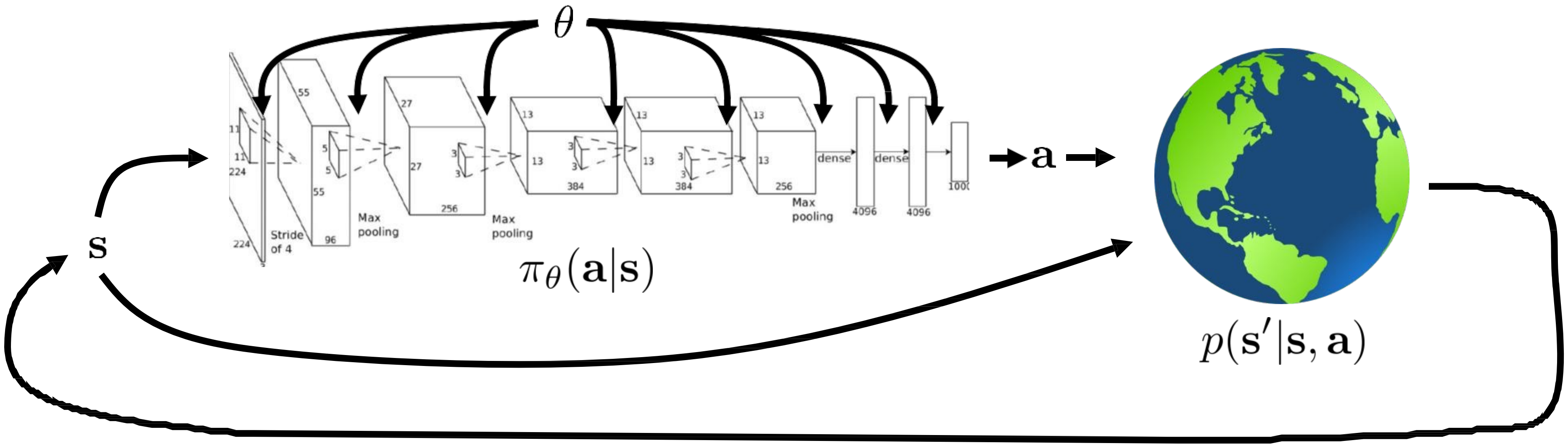


$\mathbf{o}_t$ $\quad$ $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ $\quad$ $\mathbf{a}_t$

$\mathbf{s}_t$ – state
$\mathbf{o}_t$ – observation
$\mathbf{a}_t$ – action

$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ – policy
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ – policy (fully observed)

Goal of DRL



$$\underbrace{p_\theta(\mathbf{s}_1,\mathbf{a}_1,\ldots,\mathbf{s}_T,\mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

Policy Gradients

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\left(\underbrace{\sum_{t'=t}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})}_{\text{"reward to go" } \hat{Q}_{i,t}}\right)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$

Baseline: $V(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q(\mathbf{s}_t,\mathbf{a}_t)]$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \left(Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) - V(\mathbf{s}_{i,t})\right)$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) A(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$

Value function Fitting

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})A^\pi(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$

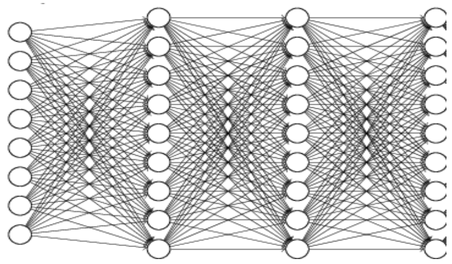
$$A^\pi(\mathbf{s}_t,\mathbf{a}_t) = Q^\pi(\mathbf{s}_t,\mathbf{a}_t) - V^\pi(\mathbf{s}_t)$$

fit *what* to *what*?

$$Q^\pi(\mathbf{s}_t,\mathbf{a}_t) \approx r(\mathbf{s}_t,\mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1})$$

$$A^\pi(\mathbf{s}_t,\mathbf{a}_t) \approx r(\mathbf{s}_t,\mathbf{a}_t) + V^\pi(\mathbf{s}_{t+1}) - V^\pi(\mathbf{s}_t)$$

$\mathbf{s}$



$\hat{V}^\pi(\mathbf{s})$

parameters ϕ

I: $y_{i,t} \approx \sum_{t'=t}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})$

II: $y_{i,t} \approx r(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) + \hat{V}_\phi^\pi(\mathbf{s}_{i,t+1})$

$$\mathcal{L}(\phi) = \frac{1}{2}\sum_i \left\|\hat{V}_\phi^\pi(\mathbf{s}_i) - y_i\right\|^2$$

Actor-Critic Algorithm (with Discount)

batch actor-critic algorithm:

1. sample $\{\mathbf{s}_i, \mathbf{a}_i\}$ from $\pi_\theta(\mathbf{a}|\mathbf{s})$ (run it on the robot)
2. fit $\hat{V}^\pi_\phi(\mathbf{s})$ to sampled reward sums
3. evaluate $\hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i) = r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \hat{V}^\pi_\phi(\mathbf{s}'_i) - \hat{V}^\pi_\phi(\mathbf{s}_i)$
4. $\nabla_\theta J(\theta) \approx \sum_i \nabla_\theta \log \pi_\theta(\mathbf{a}_i|\mathbf{s}_i) \hat{A}^\pi(\mathbf{s}_i, \mathbf{a}_i)$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}^\pi_\phi$ using target $r + \gamma \hat{V}^\pi_\phi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}^\pi_\phi(\mathbf{s}') - \hat{V}^\pi_\phi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Run policy:
1. Collect data
2. Fit value function

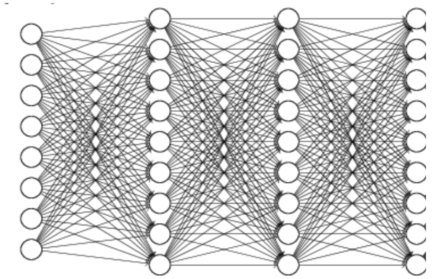
Update policy:
1. Evaluate Advantage A
2. Gradient Descent

"Max" Trick to Remove Policy Gradient

Max Trick for Q-Value

$K\times$

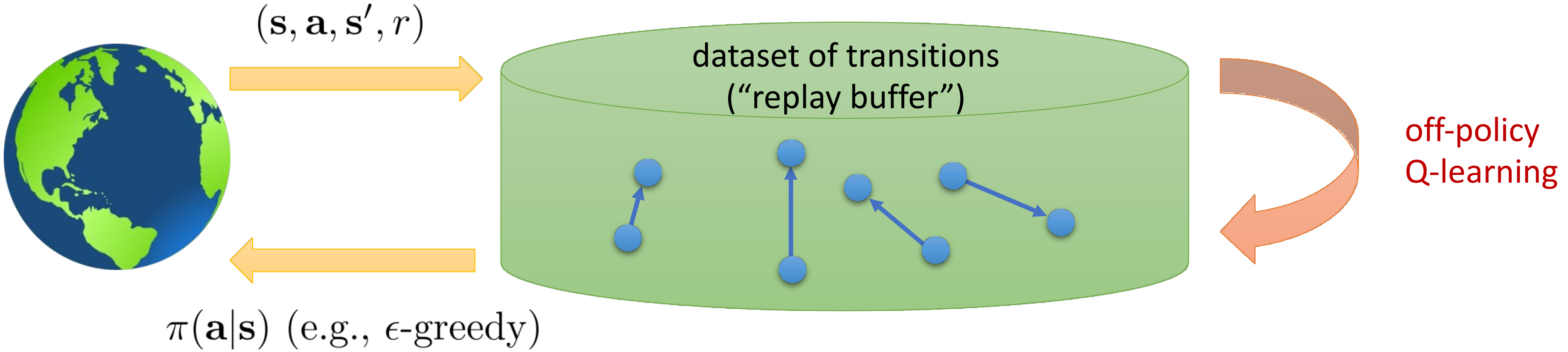
1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy
2. set $\mathbf{y}_i \leftarrow r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \underline{\max_{\mathbf{a}'_i}} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)$
3. set $\phi \leftarrow \arg\min_\phi \frac{1}{2} \sum_i \|Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{y}_i\|^2$



$Q_\phi(\mathbf{s}, \mathbf{a})$

Problem 1): Correlated samples;

Solution: Replay Buffer;



"Max" Trick to Remove Policy Gradient

Problem 2): Q-learning is *not* gradient descent!

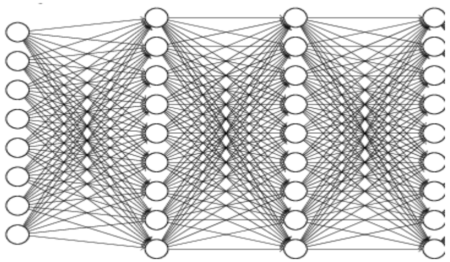
$$\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_\phi(\mathbf{s}'_i, \mathbf{a}'_i)])$$

no gradient through target value

Idea?

$\mathbf{s}$

$\mathbf{a}$

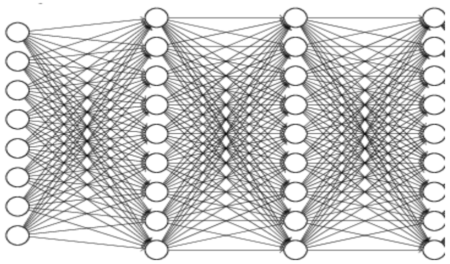


$Q_\phi(\mathbf{s}, \mathbf{a})$

parameters ϕ

$\mathbf{s}$

$\mathbf{a}$



$Q_{\phi'}$

Target network

Targets don't change in inner loop! But regression is more stable.

Q-Learning with Replay Buffer and Target Network

$N\times$ $K\times$

1. save target network parameters: $\phi' \leftarrow \phi$
2. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
3. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$
4. $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)(Q_\phi(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \gamma \max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_i, \mathbf{a}'_i)])$

Supervised Regression

Targets don't change in inner loop!

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$ using *target* network $Q_{\phi'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. update ϕ': copy ϕ every N steps

$K = 1$

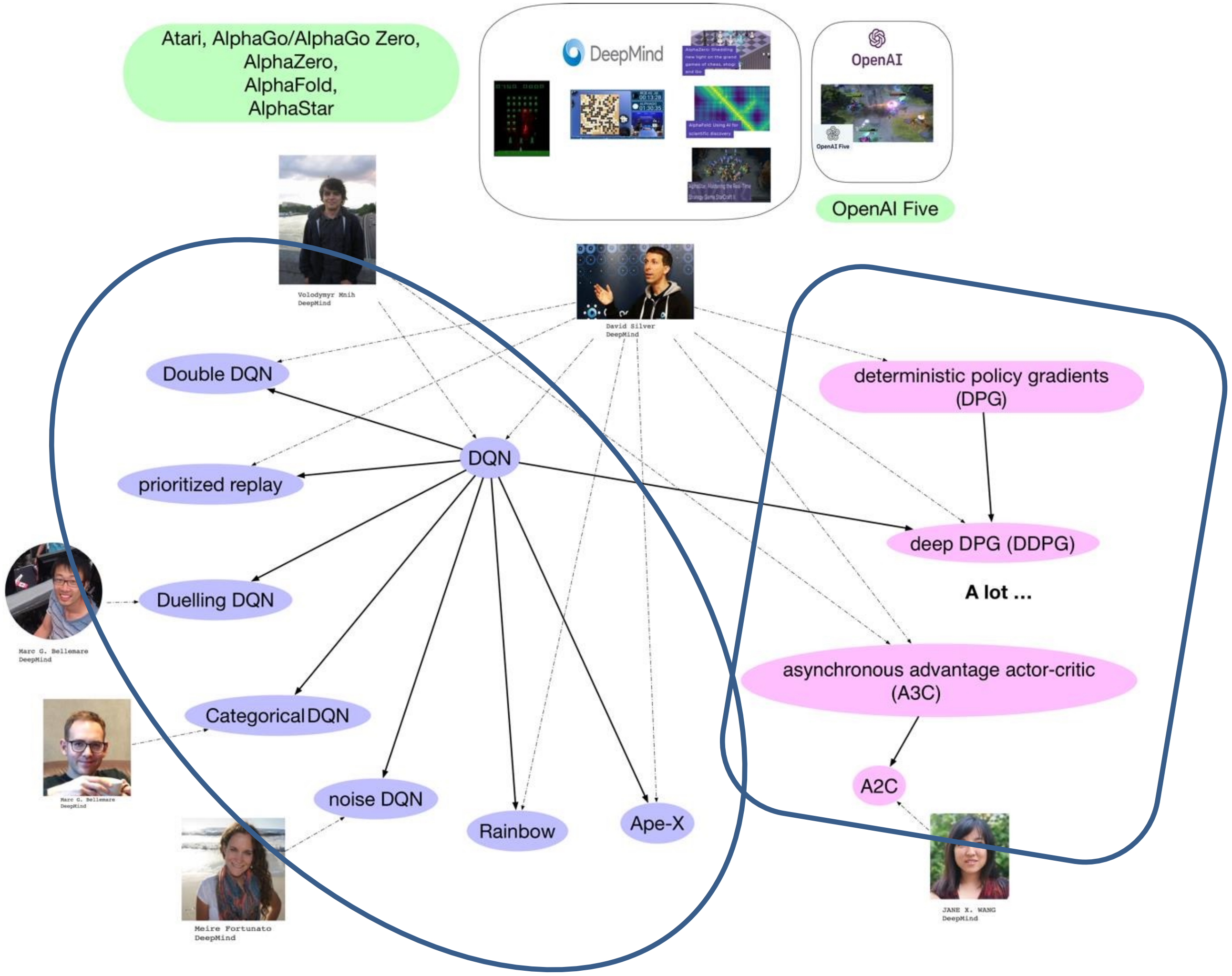
Target: copy value network!

5. update ϕ': $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$

$\tau = 0.999$ works well

DRL Algorithms

- DQN (Deep Q Network):
  - Use NN to estimate Q;
  - Experience replay and fixed target network for convergence;
  - Variants: Double DQN, Prioritized replay, Dueling DQN, Categorical DQN, Noise DQN, Rainbow
- PG and Actor-Critic:
  - Variants: AC, A2C, A3C and SAC
- DPG and DDPG:
  - Deterministic policy gradients
- TRPO and PPO

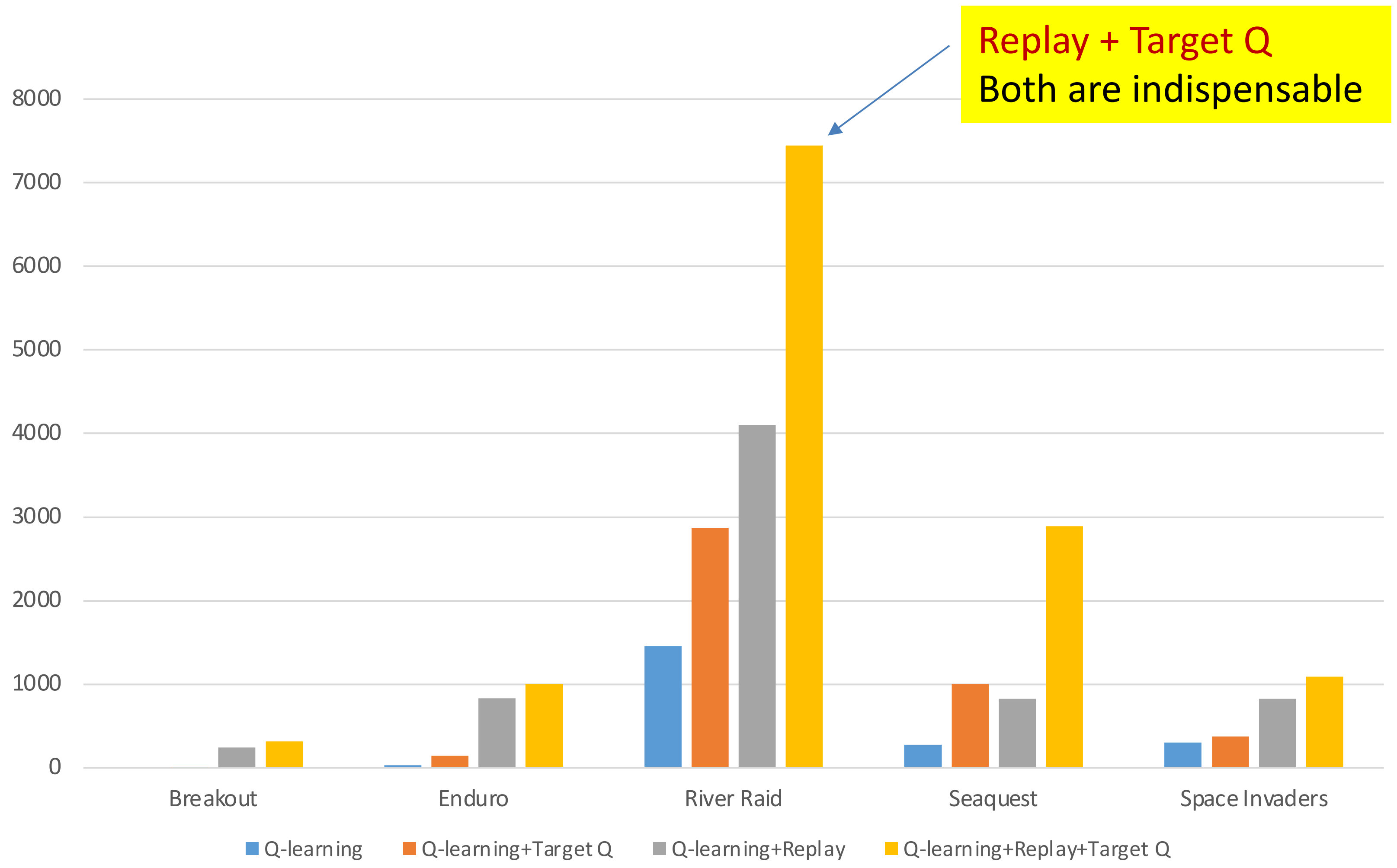


Deep Q Network

- DQN: Use deep NN to compute Q, which is a value-based method
- Before DQN, all attempts failed due to the instability:
  - Unknown reward scale of Q-value, leading to the failure of gradient BP;
  - Strong correlation between continuous-input data or image, leading to easy convergence;
  - Even a fine variation on Q-value causes a huge change on policy (From one end to another).
- Solution from DeepMind
  - Clip rewards or normalize network adaptively to sensible range;
  - Experience Reply (NIPS): store experience (s,a,r,s'); randomly drawn;
  - Freeze target Q-network:

$$loss = \left(r + \gamma \max_{a'} Q(s', a', \underline{\mathbf{w}^-}) - Q(s, a, \underline{\mathbf{w}})\right)^2$$

Deep Q Network



Overestimation of Q-learning in DQN

Overestimation: target value $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$

this last term is the problem

$Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ is not perfect – it looks "noisy"

hence $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}')$ *overestimates* the next value!

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

value *also* comes from $Q_{\phi'}$ — action selected according to $Q_{\phi'}$

Double DQN

note that $\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}') = Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

value *also* comes from $Q_{\phi'}$ — action selected according to $Q_{\phi'}$

if the noise in these is decorrelated, the problem goes away!

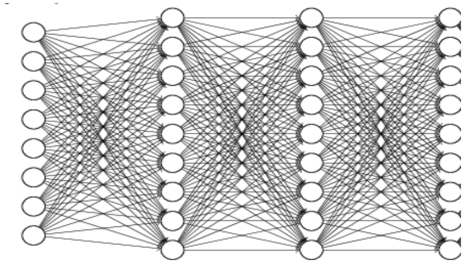
idea: don't use the same network to choose the action and evaluate value!

standard Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}', \mathbf{a}'))$

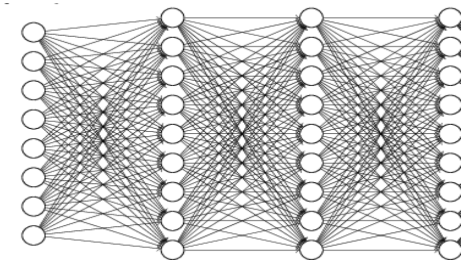
double Q-learning: $y = r + \gamma Q_{\phi'}(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi}(\mathbf{s}', \mathbf{a}'))$

just use current network (not target network) to evaluate action

still use target network to evaluate value!

$\mathbf{s}$ $\mathbf{a}$  $Q_\phi(\mathbf{s}, \mathbf{a})$

Current network: action

$\mathbf{s}$ $\mathbf{a}$  $Q_{\phi'}$

Target network: value

Multi-Step Returns

Q-learning target: $y_{j,t} = r_{j,t} + \gamma \max_{\mathbf{a}_{j,t+1}} Q_{\phi'}(\mathbf{s}_{j,t+1}, \mathbf{a}_{j,t+1})$

these are the only values that matter if $Q_{\phi'}$ is bad!

these values are important if $Q_{\phi'}$ is good

can we construct multi-step targets, like in actor-critic?

$$y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N})$$

N-step return estimator

+ less biased target values when Q-values are inaccurate

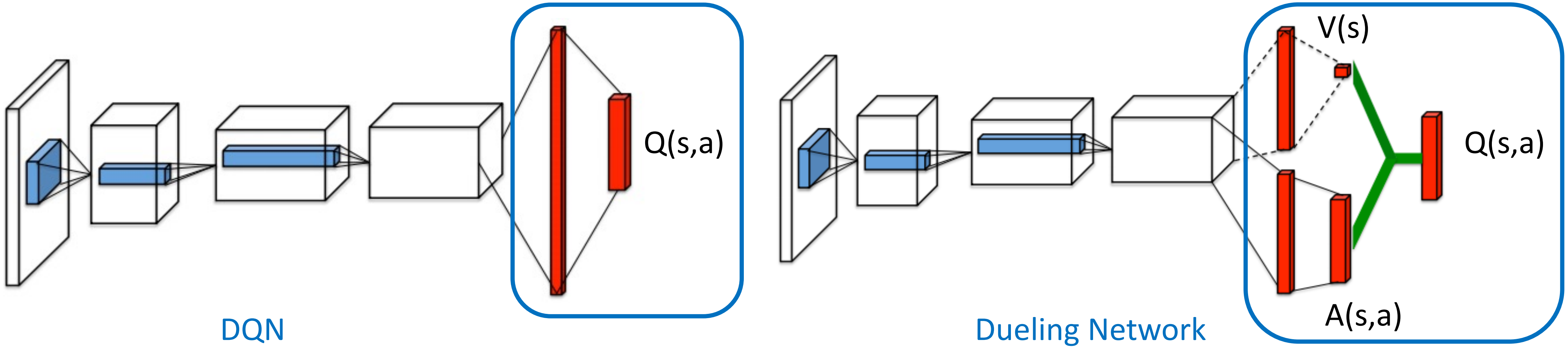
this is supposed to estimate $Q^\pi(\mathbf{s}_{j,t}, \mathbf{a}_{j,t})$ for π

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

Dueling DQN

- Dueling Network: Q=V+A separate state value V and advantage A
  - Value function V measures the value how good it is to be in a particular state s.
  - However, Q function measures the value of choosing a particular action when in this state.
  - Advantage function A obtains a relative measure of the importance of each action.

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

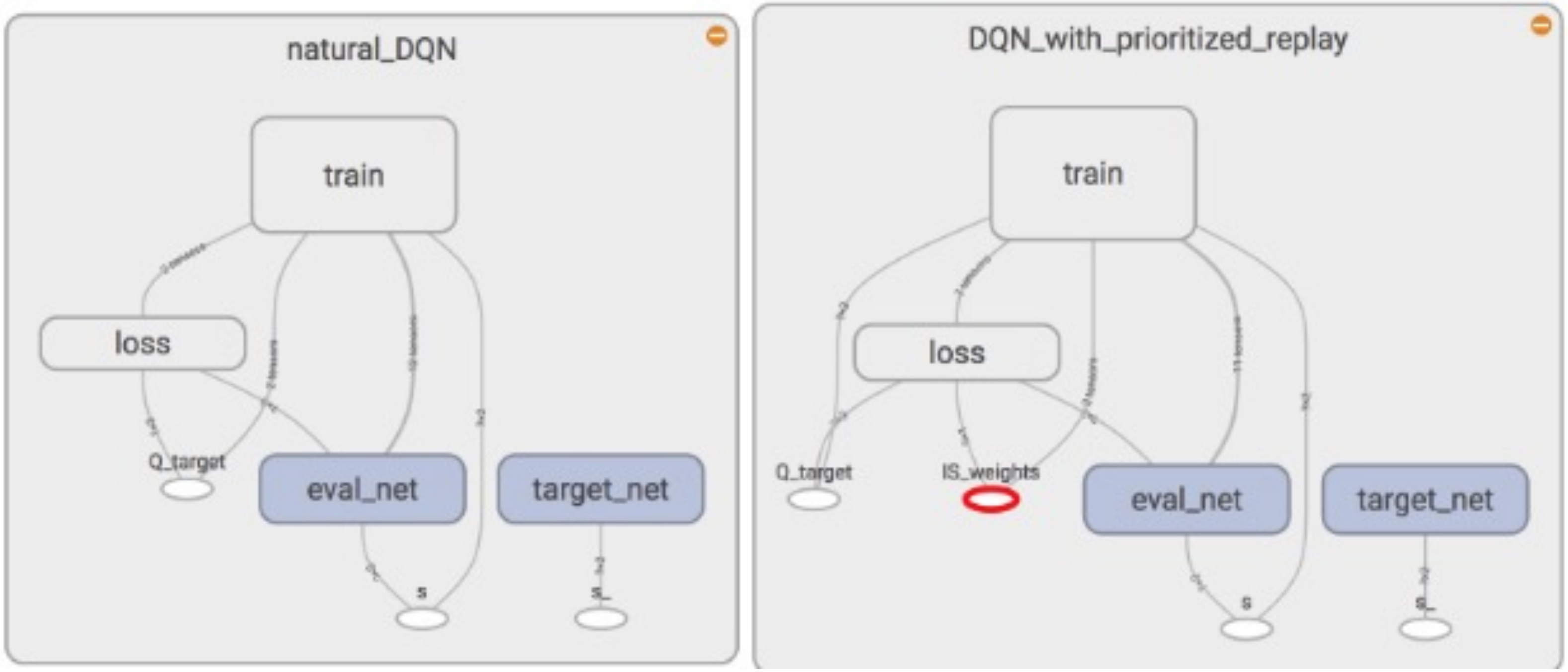


DQN

Dueling Network

Prioritized Replay

- Use priority queue to weight experiences in experience Memory based on their error (surprise) in DQN
- The bigger the TD error, the higher the priority.

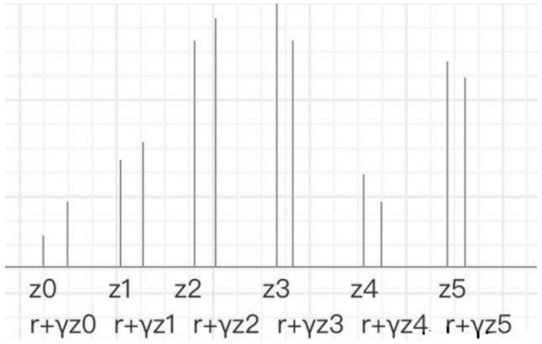


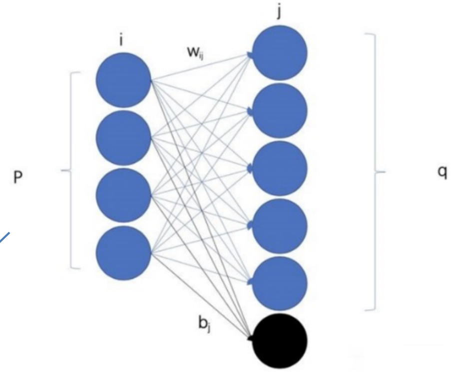
Rainbow

- Rainbow is a model-free, off-policy, value-based and discrete DRL method.
- Rainbow combines all 6 improvements in DQN, including
  - Double Q-learning
  - Multi-step learning
  - Dueling networks
  - Prioritized replay
  - Distributional RL: Q value becomes Q distribution (more stable)

$$r + \gamma \max_a Q(s_{t+1}, a)$$

$$Q(s_{t+1}, a) = \sum_i z_i p_i(s_{t+1}, a)$$





  - Noisy Nets

    1) independent Gaussian Noise: add noise on weights and No. is p*(q+1).

    2) Factorized Gaussian noise: add noise on neurons and No. is p+q

Soft Actor-Critic

- Standard RL maximizes the expected sum of rewards:

$$\sum_t \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[r(\mathbf{s}_t,\mathbf{a}_t)\right]$$

- SAC favors stochastic policies by augmenting the objective with the expected entropy of the policy:

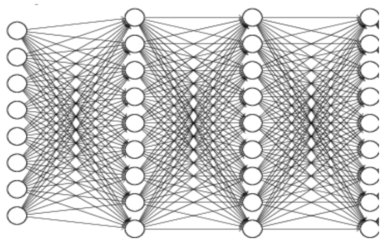
$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[r(\mathbf{s}_t,\mathbf{a}_t) + \alpha\mathcal{H}(\pi(\,\cdot\,|\mathbf{s}_t))\right]$$

- Soft state value function:

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{a}_t\sim\pi}\left[Q(\mathbf{s}_t,\mathbf{a}_t) - \log\pi(\mathbf{a}_t|\mathbf{s}_t)\right]$$

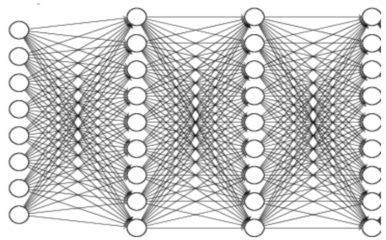
Soft Actor-Critic

Value Network $\mathbf{s}_t$



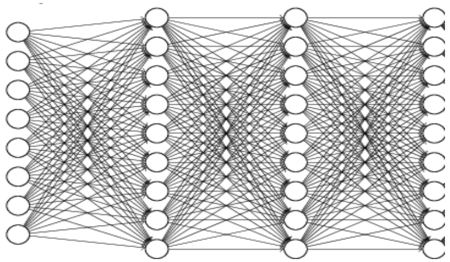
$V_\psi(\mathbf{s}_t)$

Value Target $\mathbf{s}_t$



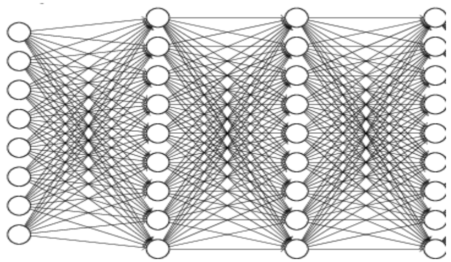
$V_{\bar{\psi}}(\mathbf{s}_t)$

Q Network $\mathbf{s}_t$ $\mathbf{a}_t$



$Q_\theta(\mathbf{s}_t, \mathbf{a}_t)$

Policy Network $\mathbf{s}_t$



$\pi_\phi(\cdot|\mathbf{s}_t)$

- Soft value function V (MSE):

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t \sim \pi_\phi} \left[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) \right] \right)^2 \right]$$

$$\hat{\nabla}_\psi J_V(\psi) = \nabla_\psi V_\psi(\mathbf{s}_t) \left(V_\psi(\mathbf{s}_t) - \underline{Q_\theta(\mathbf{s}_t, \mathbf{a}_t)} + \underline{\log \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)} \right)$$

Soft Actor-Critic

- Soft Q-function parameters Q (MSE):

$$J_Q(\theta) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}(\mathbf{s}_t, \mathbf{a}_t) \right)^2 \right]$$

$$\hat{Q}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\, \mathbb{E}_{\mathbf{s}_{t+1} \sim p} \left[V_{\bar{\psi}}(\mathbf{s}_{t+1}) \right]$$

$$\hat{\nabla}_\theta J_Q(\theta) = \nabla_\theta Q_\theta(\mathbf{a}_t, \mathbf{s}_t) \left(Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - r(\mathbf{s}_t, \mathbf{a}_t) - \gamma V_{\bar{\psi}}(\mathbf{s}_{t+1}) \right)$$

- Policy parameters learned by minimizing expected KL-divergence:

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\mathrm{D}_{\mathrm{KL}} \left(\pi_\phi(\,\cdot\,|\mathbf{s}_t) \,\middle\|\, \frac{\exp\left(Q_\theta(\mathbf{s}_t,\,\cdot\,)\right)}{Z_\theta(\mathbf{s}_t)} \right) \right]$$

- Target value network (for overestimate): moving average of value network weight

$$\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$$

Soft Actor-Critic

Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$ → Update value V
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$ → Update Q
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ → Update Policy
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$ → Update target value network
  end for
end for

use the minimum of Q-functions for the value gradient

Q-learning with continuous actions

What's the problem with continuous actions?

$$\pi(\mathbf{a}_t|\mathbf{s}_t) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi(\mathbf{s}_t, \mathbf{a}_t) \\ 0 \text{ otherwise} \end{cases}$$

this max

$$\text{target value } y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j)$$

this max particularly problematic

How do we perform the max?

DDPG-Learn an Approximate Maximizer

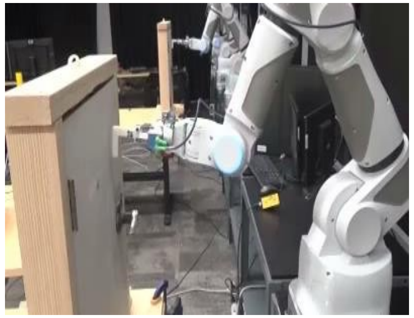
$\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}) = Q_\phi(\mathbf{s}, \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a}))$

idea: train another network $\mu_\theta(\mathbf{s})$ such that $\mu_\theta(\mathbf{s}) \approx \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s},\mathbf{a})$

how? just solve $\theta \leftarrow \arg\max_\theta Q_\phi(\mathbf{s}, \mu_\theta(\mathbf{s}))$ $\qquad \frac{dQ_\phi}{d\theta} = \frac{d\mathbf{a}}{d\theta}\frac{dQ_\phi}{d\mathbf{a}}$

new target $y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_\theta(\mathbf{s}'_j)) \approx r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \arg\max_{\mathbf{a}'} Q_{\phi'}(\mathbf{s}'_j, \mathbf{a}'_j))$

DDPG:



1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$ using *target* nets $Q_{\phi'}$ and $\mu_{\theta'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$ Value Network
5. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(\mathbf{s}_j)\frac{dQ_\phi}{d\mathbf{a}}(\mathbf{s}_j, \mu(\mathbf{s}_j))$ Policy Network; deterministic
6. update ϕ' and θ' (e.g., Polyak averaging)

Trust Region Policy Optimization (TRPO)

Recall:

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Problem: *unstable!*

Bad α may cause terrible policy π_θ!

Question: *How to make policy monotonic improved? (always cause better policy?)*

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})\underbrace{\left(\hat{Q}_{i,t} - \mathbb{E}_a \hat{Q}_{i,t}\right)}_{\hat{A}_{i,t}} \rightarrow \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\hat{A}_t\right]$$

$$\text{s.t.} \quad \underbrace{D_{\mathrm{KL}}^{\max}(\theta_{\text{old}}, \theta) \le \delta.}_{\text{"Trust Region"}}$$

$$+ \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right]$$

Proximal Policy Optimization (PPO)

Off-policy Policy Gradient

$$\nabla_{\theta'} J(\theta') = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\pi_{\theta'}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})}\nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})\hat{Q}_{i,t}$$

Variance Reducing 2

$$\nabla_{\theta'} J(\theta') = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\pi_{\theta'}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})}\nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})\underbrace{\left(\hat{Q}_{i,t} - \mathbb{E}\hat{Q}_{i,t}\right)}_{\text{“advantage” } \hat{A}_{i,t}}$$

“advantage ” $\hat{A}_{i,t}$

How to introduce trust region efficiently? ***CLIP:*** $\operatorname{clip}(x, l, u) = \min(\max(x, l), u)$

$$\nabla_{\theta'} J(\theta') = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\underbrace{\operatorname{clip}\left(\frac{\pi_{\theta'}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})}{\pi_{\theta}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})}, 1-\epsilon, 1+\epsilon\right)}_{\text{“Trust Region by Clip”}}\nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t})\left(\hat{Q}_{i,t} - \mathbb{E}\hat{Q}_{i,t}\right)$$

“Trust Region by Clip”

TRPO and PPO

Policy Gradient Methods

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \Big[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \Big]$$

$$\hat{g} = \hat{\mathbb{E}}_t \Big[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \Big]$$

Trust Region Methods (TRPO)

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right]$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t [\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta.$$

Proximal Policy Optimization (PPO)

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t) \Big]$$

Model-based DRL

model-based reinforcement learning version 1.0:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$ → Model
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions → Planning
4. execute those actions and add the resulting data $\{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_j\}$ to $\mathcal{D}$

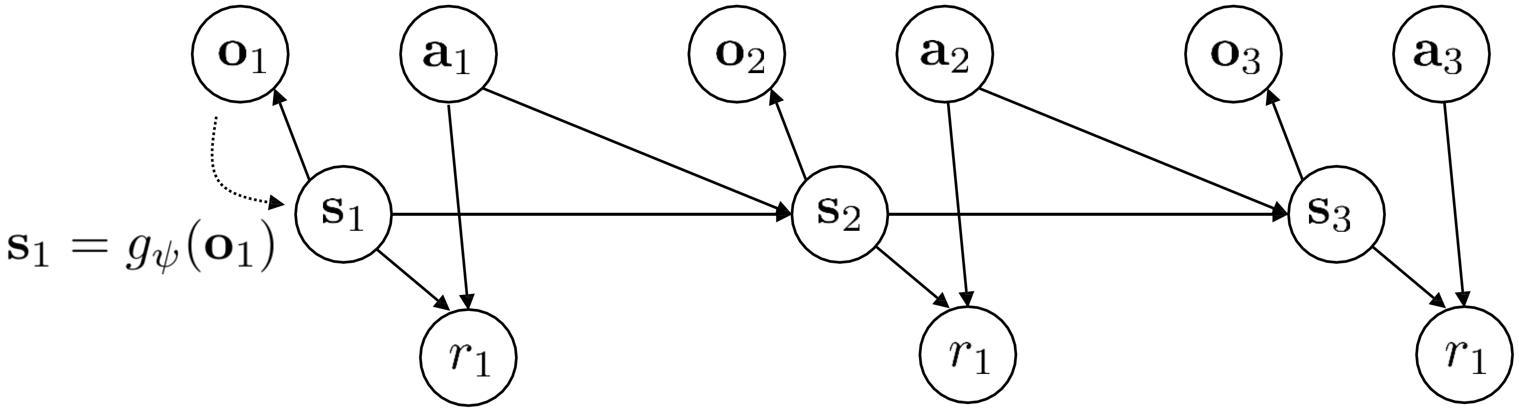
model-based reinforcement learning version 1.5:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$ → Model
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions → Planning
4. execute the first planned action, observe resulting state $\mathbf{s}'$ (MPC)
5. append $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ to dataset $\mathcal{D}$

Replanning

every N steps

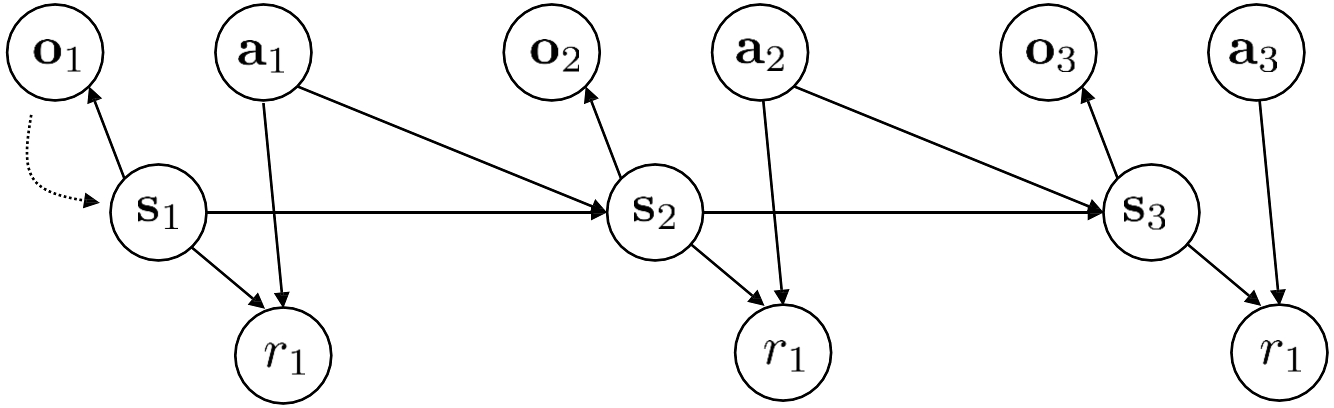
Model-based DRL with Latent Space Models



$$\max_{\phi,\psi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \underbrace{\log p_\phi(g_\psi(\mathbf{o}_{t+1,i}) | g_\psi(\mathbf{o}_{t,i}), \mathbf{a}_{t,i})}_{\text{Latent Space Dynamics}} + \underbrace{\log p_\phi(\mathbf{o}_{t,i} | g_\psi(\mathbf{o}_{t,i}))}_{\text{Image Reconstruction}} + \underbrace{\log p_\phi(r_{t,i} | g_\psi(\mathbf{o}_{t,i}))}_{\text{Reward Model}}$$

Many practical methods consider a Stochastic Encoder for Model Uncertainty.

Model-based DRL with Latent Space Models

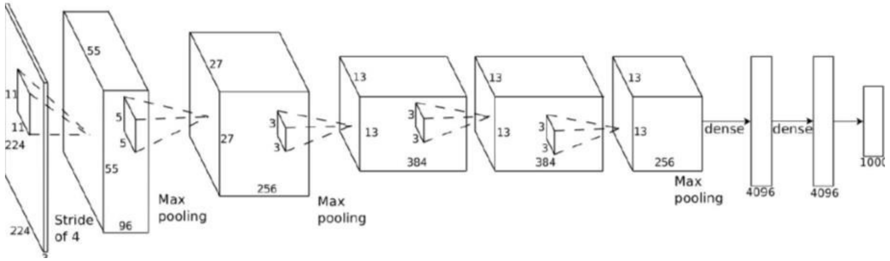


model-based reinforcement learning with latent state:

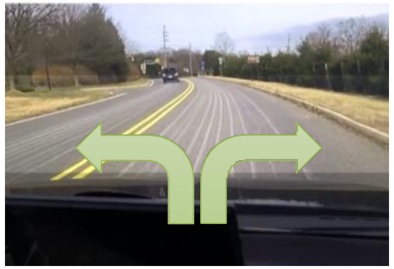
1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{o}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{o}, \mathbf{a}, \mathbf{o}')_i\}$
2. learn $p_\phi(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, $p_\phi(r_t|\mathbf{s}_t)$, $p(\mathbf{o}_t|\mathbf{s}_t)$, $g_\psi(\mathbf{o}_t)$
3. plan through the model to choose actions
4. execute the first planned action, observe resulting $\mathbf{o}'$ (MPC)
5. append $(\mathbf{o}, \mathbf{a}, \mathbf{o}')$ to dataset $\mathcal{D}$

Imitation Learning

$\mathbf{o}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$



$\mathbf{a}_t$

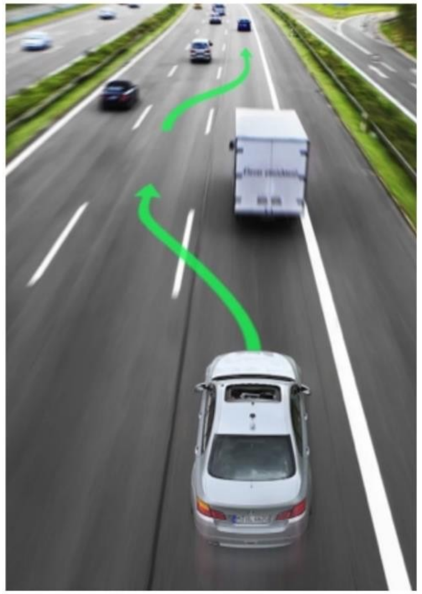
$\mathbf{o}_t$
$\mathbf{a}_t$

→ training data → supervised learning → $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$

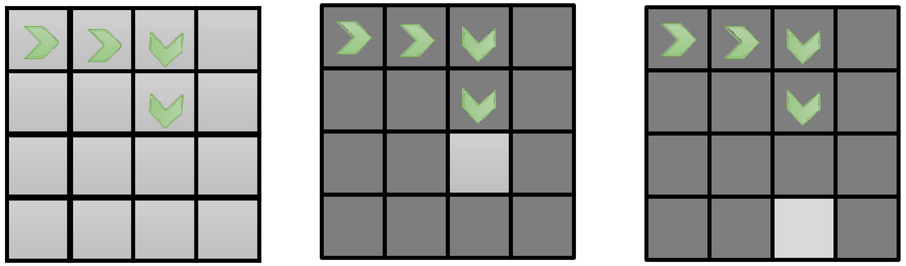
Behavioral Cloning

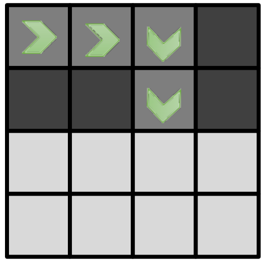
Inverse Reinforcement Learning (IRL)

Infer reward functions from demonstrations



$$r(\mathbf{s}, \mathbf{a})$$





Various Reward Functions

- Underspecified problem
- Many reward functions can explain the same behavior

Inverse Reinforcement Learning (IRL)

"forward" reinforcement learning

given:

states $\mathbf{s} \in \mathcal{S}$, actions $\mathbf{a} \in \mathcal{A}$

(sometimes) transitions $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$

reward function $r(\mathbf{s}, \mathbf{a})$

learn $\pi^\star(\mathbf{a}|\mathbf{s})$

Inverse reinforcement learning

given:

states $\mathbf{s} \in \mathcal{S}$, actions $\mathbf{a} \in \mathcal{A}$

(sometimes) transitions $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$

samples $\{\tau_i\}$ sampled from $\pi^\star(\tau)$

learn $r_\psi(\mathbf{s}, \mathbf{a})$ ← Reward Parameters

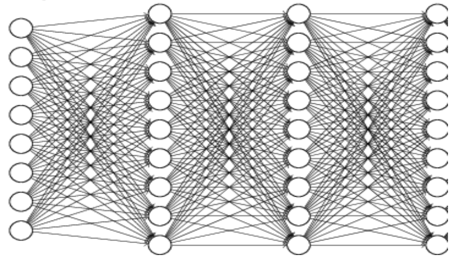
...and then use it to learn $\pi^\star(\mathbf{a}|\mathbf{s})$

linear reward function:

$$r_\psi(\mathbf{s}, \mathbf{a}) = \sum_i \psi_i f_i(\mathbf{s}, \mathbf{a}) = \psi^T \mathbf{f}(\mathbf{s}, \mathbf{a})$$

neural net reward function:

$\mathbf{s}$, $\mathbf{a}$



$r_\psi(\mathbf{s}, \mathbf{a})$

Learn Optimality Variable

given:

samples $\{\tau_i\}$ sampled from $\pi^\star(\tau)$

$$p(\tau|\mathcal{O}_{1:T},\psi) \propto p(\tau) \exp\left(\sum_t r_\psi(\mathbf{s}_t,\mathbf{a}_t)\right)$$

can ignore (independent of ψ)

$$Z = \int p(\tau)\exp(r_\psi(\tau))d\tau$$

maximum likelihood learning: $$\max_\psi \frac{1}{N}\sum_{i=1}^N \log p(\tau_i|\mathcal{O}_{1:T},\psi) = \max_\psi \frac{1}{N}\sum_{i=1}^N r_\psi(\tau_i) - \log Z$$

partition function

$$\nabla_\psi \mathcal{L} = \frac{1}{N}\sum_{i=1}^N \nabla_\psi r_\psi(\tau_i) - \underbrace{\frac{1}{Z}\int p(\tau)\exp(r_\psi(\tau))\nabla_\psi r_\psi(\tau)d\tau}_{p(\tau|\mathcal{O}_{1:T},\psi)}$$

$$\nabla_\psi \mathcal{L} = E_{\tau\sim\pi^\star(\tau)}[\nabla_\psi r_\psi(\tau_i)] - E_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\tau)]$$

estimate with expert samples

soft optimal policy under current reward

Estimation of Expectation

$$\nabla_\psi \mathcal{L} = E_{\tau\sim\pi^\star(\tau)}[\nabla_\psi r_\psi(\tau_i)] - E_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\tau)]$$

$$E_{\tau\sim p(\tau|\mathcal{O}_{1:T},\psi)}\left[\nabla_\psi \sum_{t=1}^{T} r_\psi(\mathbf{s}_t,\mathbf{a}_t)\right]$$

$$= \sum_{t=1}^{T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p(\mathbf{s}_t,\mathbf{a}_t|\mathcal{O}_{1:T},\psi)}[\nabla_\psi r_\psi(\mathbf{s}_t,\mathbf{a}_t)]$$

$$p(\mathbf{a}_t|\mathbf{s}_t,\mathcal{O}_{1:T},\psi)p(\mathbf{s}_t|\mathcal{O}_{1:T},\psi)$$

backward message $\beta_t(\mathbf{s}_t,\mathbf{a}_t) = p(\mathcal{O}_{t:T}|\mathbf{s}_t,\mathbf{a}_t)$

$$= \frac{\beta(\mathbf{s}_t,\mathbf{a}_t)}{\beta(\mathbf{s}_t)}$$

$$\propto \alpha(\mathbf{s}_t)\beta(\mathbf{s}_t)$$

forward message $\alpha_t(\mathbf{s}_t) = p(\mathbf{s}_t|\mathcal{O}_{1:t-1})$

$$p(\mathbf{a}_t|\mathbf{s}_t,\mathcal{O}_{1:T},\psi)p(\mathbf{s}_t|\mathcal{O}_{1:T},\psi) \propto \beta(\mathbf{s}_t,\mathbf{a}_t)\alpha(\mathbf{s}_t)$$

IRL Algorithm: MaxEnt

let $\mu_t(\mathbf{s}_t, \mathbf{a}_t) \propto \beta(\mathbf{s}_t, \mathbf{a}_t)\alpha(\mathbf{s}_t)$

$$\nabla_\psi \mathcal{L} = E_{\tau \sim \pi^\star(\tau)}[\nabla_\psi r_\psi(\tau_i)] - \underbrace{E_{\tau \sim p(\tau|\mathcal{O}_{1:T}, \psi)}[\nabla_\psi r_\psi(\tau)]}$$

$$\sum_{t=1}^{T} \int \int \mu_t(\mathbf{s}_t, \mathbf{a}_t) \nabla_\psi r_\psi(\mathbf{s}_t, \mathbf{a}_t) d\mathbf{s}_t d\mathbf{a}_t = \sum_{t=1}^{T} \vec{\mu}_t^T \nabla_\psi \vec{r}_\psi$$

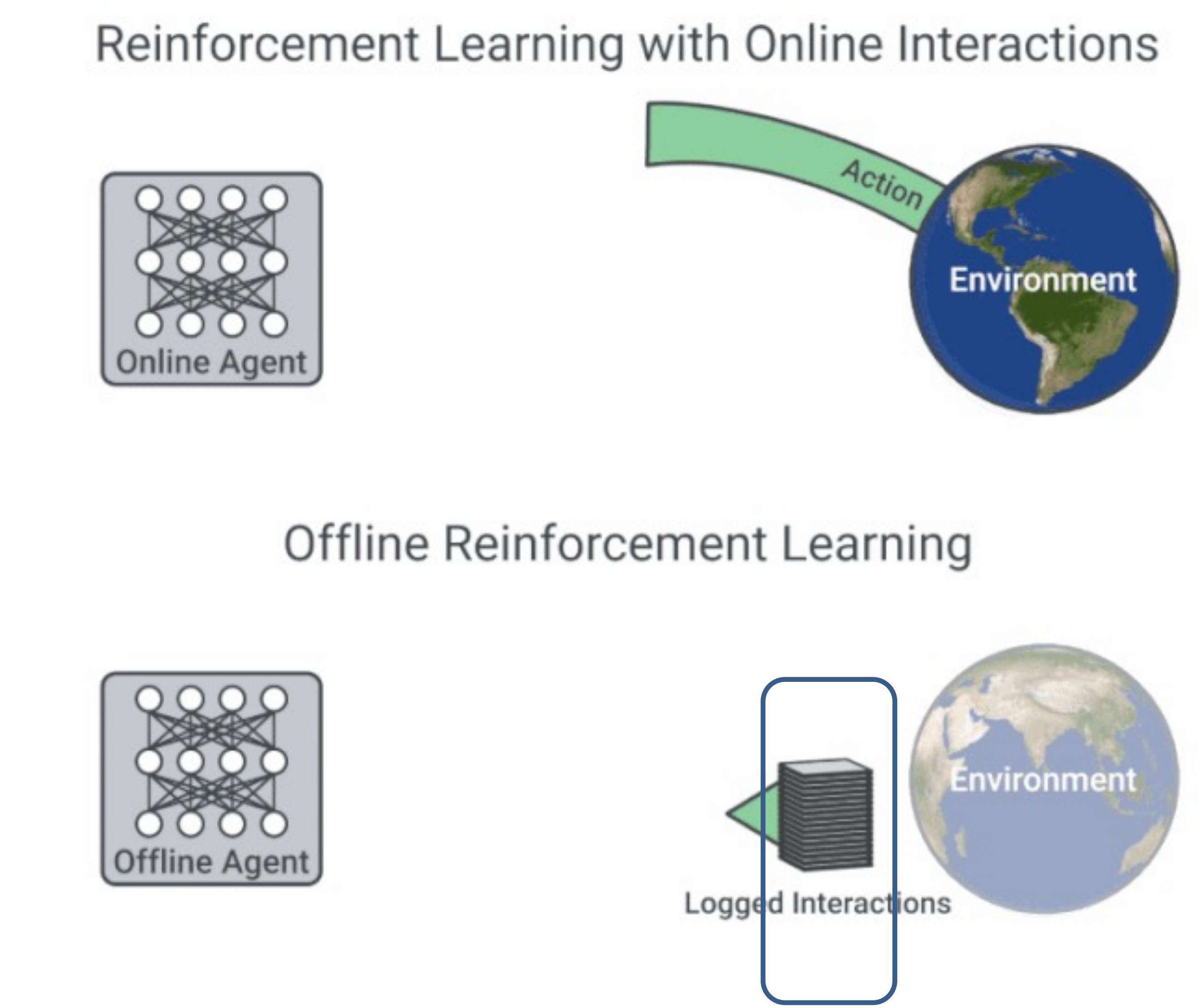
state-action visitation probability for each $(\mathbf{s}_t, \mathbf{a}_t)$

Visitation Frequency

MaxEnt:

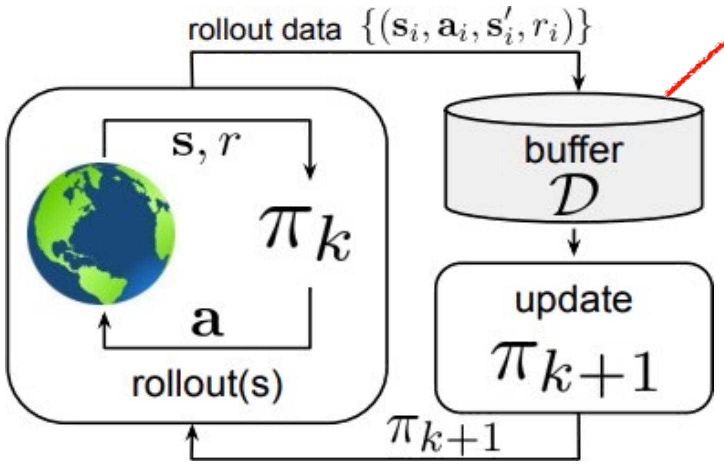
1. Given ψ, compute backward message $\beta(\mathbf{s}_t, \mathbf{a}_t)$
2. Given ψ, compute forward message $\alpha(\mathbf{s}_t)$
3. Compute $\mu_t(\mathbf{s}_t, \mathbf{a}_t) \propto \beta(\mathbf{s}_t, \mathbf{a}_t)\alpha(\mathbf{s}_t)$
4. Evaluate $\nabla_\psi \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\psi r_\psi(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - \sum_{t=1}^{T} \int \int \mu_t(\mathbf{s}_t, \mathbf{a}_t) \nabla_\psi r_\psi(\mathbf{s}_t, \mathbf{a}_t) d\mathbf{s}_t d\mathbf{a}_t$
5. $\psi \leftarrow \psi + \eta \nabla_\psi \mathcal{L}$

Offline Reinforcement Learning

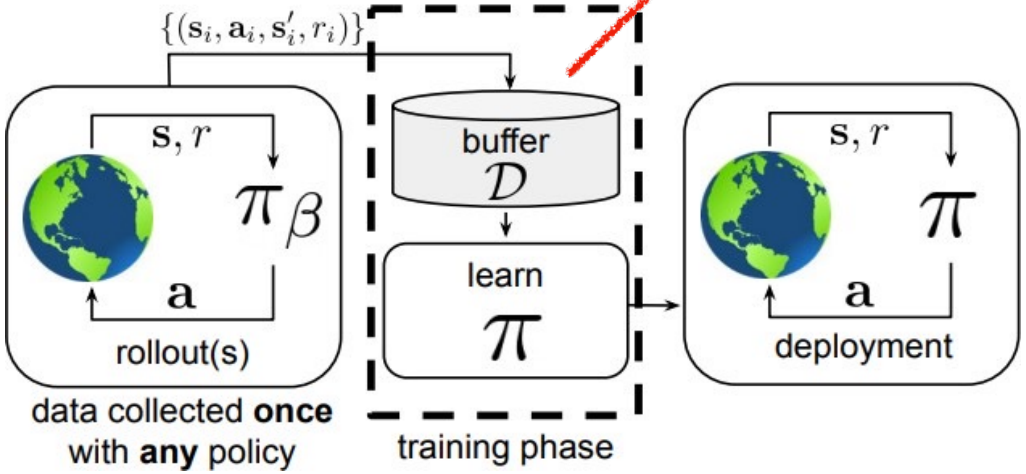


Off-policy and Offline DRL

"Off-Policy" buffer from past policies



"Off-Policy" buffer from some unknown policies



- Off-Policy DRL Algorithms
- Offline DRL Algorithms

Offline Reinforcement Learning

Supervised Learning

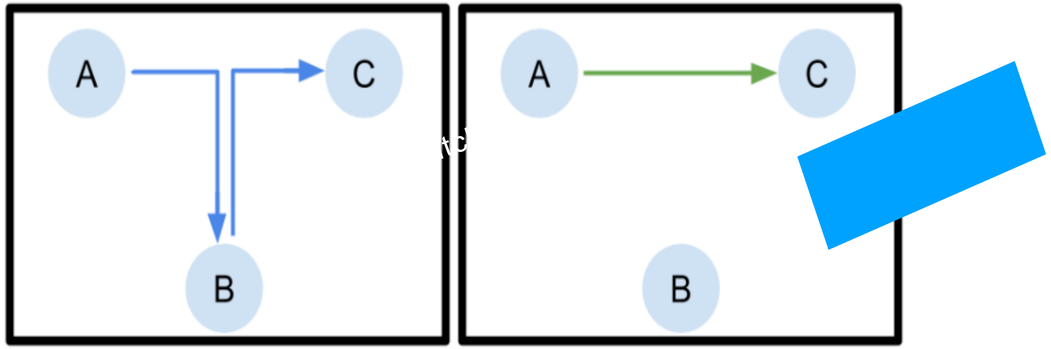
Can do as good as the dataset!

Offline Reinforcement Learning

Can do **better** than the dataset!

Dog?

Cat?



Can show that Q-learning recovers optimal policy from random data.

Conservative Q-Learning (CQL)

- Conservative Q-learning (CQL): aims to address these limitations by learning a conservative Q-function such that the expected value of a policy under this Q-function lower-bounds its true value.

- To prevent overestimation: learn a conservative, lower-bound Q-function by additionally minimizing Q-values alongside Bellman error objective.

Kumar, Zhou, Tucker, Levine. Conservative Q-Learning for Offline RL. NeurIPS 2020.

Conservative Q-Learning (CQL)

- Policy Evaluation:

Minimize big Q-values

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \; \alpha \, \mathbb{E}_{\mathbf{s} \sim \mathcal{D}, \mathbf{a} \sim \mu(\mathbf{a}|\mathbf{s})} \left[Q(\mathbf{s}, \mathbf{a}) \right] + \frac{1}{2} \mathbb{E}_{\mathbf{s}, \mathbf{a} \sim \mathcal{D}} \left[\left(Q(\mathbf{s}, \mathbf{a}) - \hat{\mathcal{B}}^{\pi} \hat{Q}^{k}(\mathbf{s}, \mathbf{a}) \right)^2 \right]$$

- Furthermore, improve the bound by introducing an additional Q-value maximization term.

Minimize big Q-values

Maximize Data Q-values

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \; \alpha \cdot \left(\mathbb{E}_{\mathbf{s} \sim \mathcal{D}, \mathbf{a} \sim \mu(\mathbf{a}|\mathbf{s})} \left[Q(\mathbf{s}, \mathbf{a}) \right] - \mathbb{E}_{\mathbf{s} \sim \mathcal{D}, \mathbf{a} \sim \hat{\pi}_{\beta}(\mathbf{a}|\mathbf{s})} \left[Q(\mathbf{s}, \mathbf{a}) \right] \right) + \frac{1}{2} \mathbb{E}_{\mathbf{s}, \mathbf{a}, \mathbf{s}' \sim \mathcal{D}} \left[\left(Q(\mathbf{s}, \mathbf{a}) - \hat{\mathcal{B}}^{\pi} \hat{Q}^{k}(\mathbf{s}, \mathbf{a}) \right)^2 \right]$$

Conservative Q-Learning (CQL)

- How should we utilize this for policy optimization?
  - ➢ Alternate between performing full off-policy evaluation for each policy iterate, and one-step of policy improvement.

Minimize big Q-values

Maximize Data Q-values

$$\min_{Q} \max_{\mu} \; \alpha \left(\mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\mu(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right] - \mathbb{E}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\hat{\pi}_{\beta}(\mathbf{a}|\mathbf{s})}\left[Q(\mathbf{s},\mathbf{a})\right]\right)$$
$$+\frac{1}{2}\,\mathbb{E}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\left(Q(\mathbf{s},\mathbf{a}) - \hat{\mathcal{B}}^{\pi_k}\hat{Q}^{k}(\mathbf{s},\mathbf{a})\right)^2\right] + \mathcal{R}(\mu) \quad (\mathrm{CQL}(\mathcal{R}))$$

Standard Bellman Error

Regularization

CQL Algorithm:

1. Learn $\hat{Q}^{\pi}_{\mathrm{CQL}}$ using offline data $\mathcal{D}$.
2. Optimize policy w.r.t. $\hat{Q}^{\pi}_{\mathrm{CQL}}$: $\pi \leftarrow \arg\max_{\pi} \; \mathbb{E}_{\pi}[\hat{Q}^{\pi}_{\mathrm{CQL}}]$.

Model-based Offline Policy Optimization (MOPO)

- Standard model-based methods: designed for the online setting, do NOT provide an explicit mechanism to avoid the distributional shift issue.

- MOPO: modify the existing model-based RL by applying them with rewards artificially penalized by the uncertainty of the dynamics.

Model-based Offline Policy Optimization (MOPO)

- MOPO: modify the existing model-based RL by considering such rewards artificially penalized by the uncertainty of the dynamics.

Algorithm 1 Framework for Model-based Offline Policy Optimization (MOPO) with Reward Penalty

Require: Dynamics model $\widehat{T}$ with admissible error estimator $u(s,a)$; constant λ.

1: Define $\tilde{r}(s,a) = r(s,a) - \lambda u(s,a)$. Let $\widetilde{M}$ be the MDP with dynamics $\widehat{T}$ and reward $\tilde{r}$.

2: Run any RL algorithm on $\widetilde{M}$ until convergence to obtain $\hat{\pi} = \mathrm{argmax}_{\pi} \eta_{\widetilde{M}}(\pi)$

- Maximum standard deviation of the learned models in the ensemble:

$$u(s,a) = \max_{i=1}^{N} \|\Sigma_{\phi}^{i}(s,a)\|_{\mathrm{F}}$$

$$\tilde{r}(s,a) = \hat{r}(s,a) - \lambda \max_{i=1,\ldots,N} \|\Sigma_{\phi}^{i}(s,a)\|_{\mathrm{F}}$$

Model-based Offline Policy Optimization (MOPO)

Algorithm 1 Framework for Model-based Offline Policy Optimization (MOPO) with Reward Penalty

Require: Dynamics model $\widehat{T}$ with admissible error estimator $u(s,a)$; constant λ.

1: Define $\tilde{r}(s,a) = r(s,a) - \lambda u(s,a)$. Let $\widetilde{M}$ be the MDP with dynamics $\widehat{T}$ and reward $\tilde{r}$.

2: Run any RL algorithm on $\widetilde{M}$ until convergence to obtain $\hat{\pi} = \text{argmax}_\pi \eta_{\widetilde{M}}(\pi)$

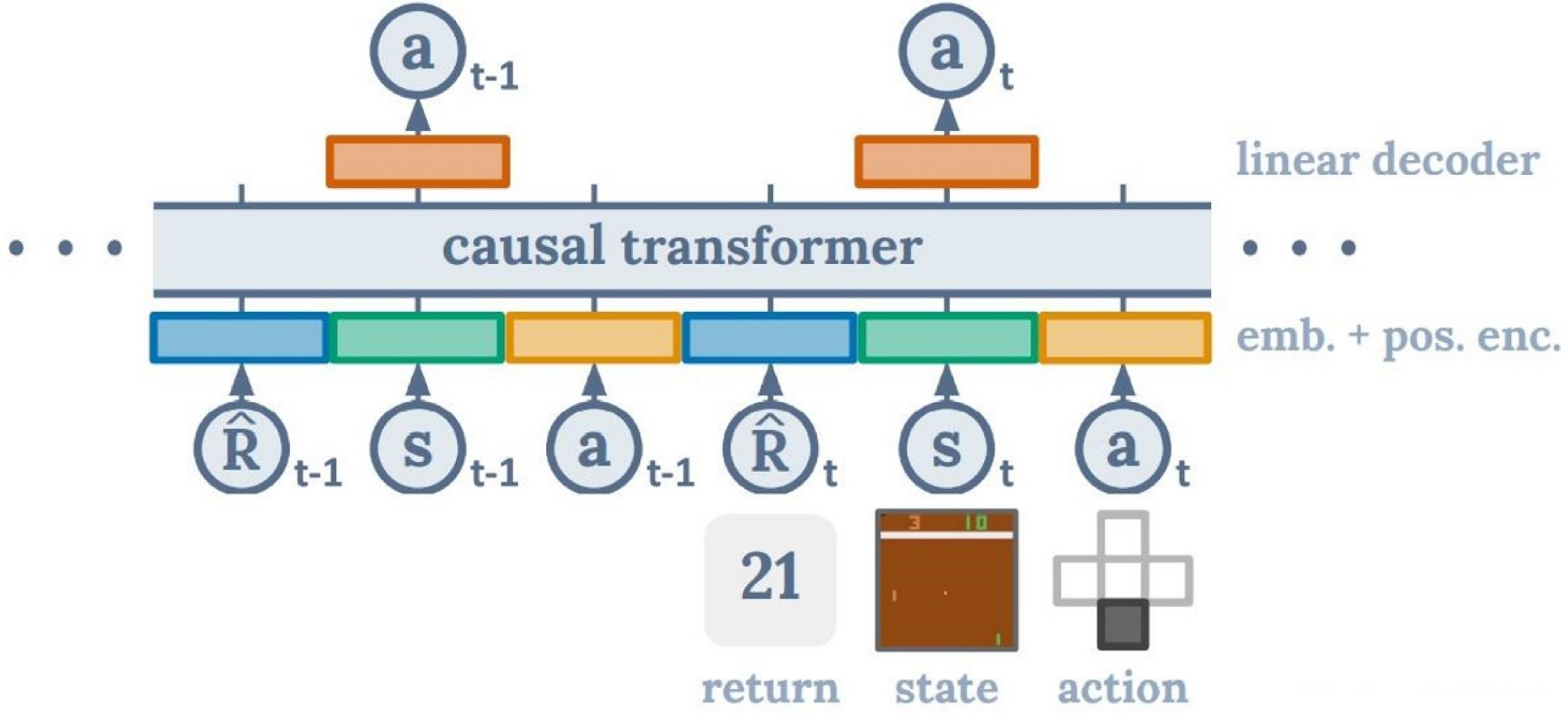
- Model the dynamics using a neural network that outputs a Gaussian distribution over the next state and reward:

$$\widehat{T}_{\theta,\phi}(s_{t+1}, r | s_t, a_t) = \mathcal{N}(\mu_\theta(s_t, a_t), \Sigma_\phi(s_t, a_t))$$

- We learn an ensemble of N dynamics models, with each model trained independently via maximum likelihood.

$$\{\widehat{T}^i_{\theta,\phi} = \mathcal{N}(\mu^i_\theta, \Sigma^i_\phi)\}_{i=1}^N$$

Decision Transformer



Reinforcement Learning via Sequence Modeling

Decision Transformer

- Reinforcement Learning via Sequence Modeling, where the input is

$$\tau = (\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \cdots, \hat{R}_T, s_T, a_T)$$

$$\{\hat{R}_t, S_t, a_t\}_{t=0}^{T} \qquad \hat{R}_t = \sum_{t'=t}^{T} r_{t'}$$

- Via autoregression, the generated output is

$$\{a_t\}_{t=0}^{T}$$

- The architecture of network is decoder only, masked multi-head self-attention.
- Position embedding: one timestep corresponds to three tokens (r,s,a)
- Embedding = embedding + position embedding

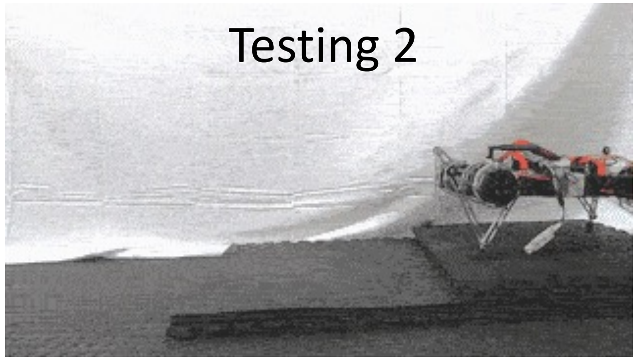
Outline

- ❑ Deep Reinforcement Learning
- ❑ Applications to Robotics

Applications to Robotics

- SAC for Robot Walking
- Policy Learning for Footed Robot
- Robot Manipulation

Soft Actor-Critic



Algorithm 1 Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  end for
end for

Update value V

Update Q

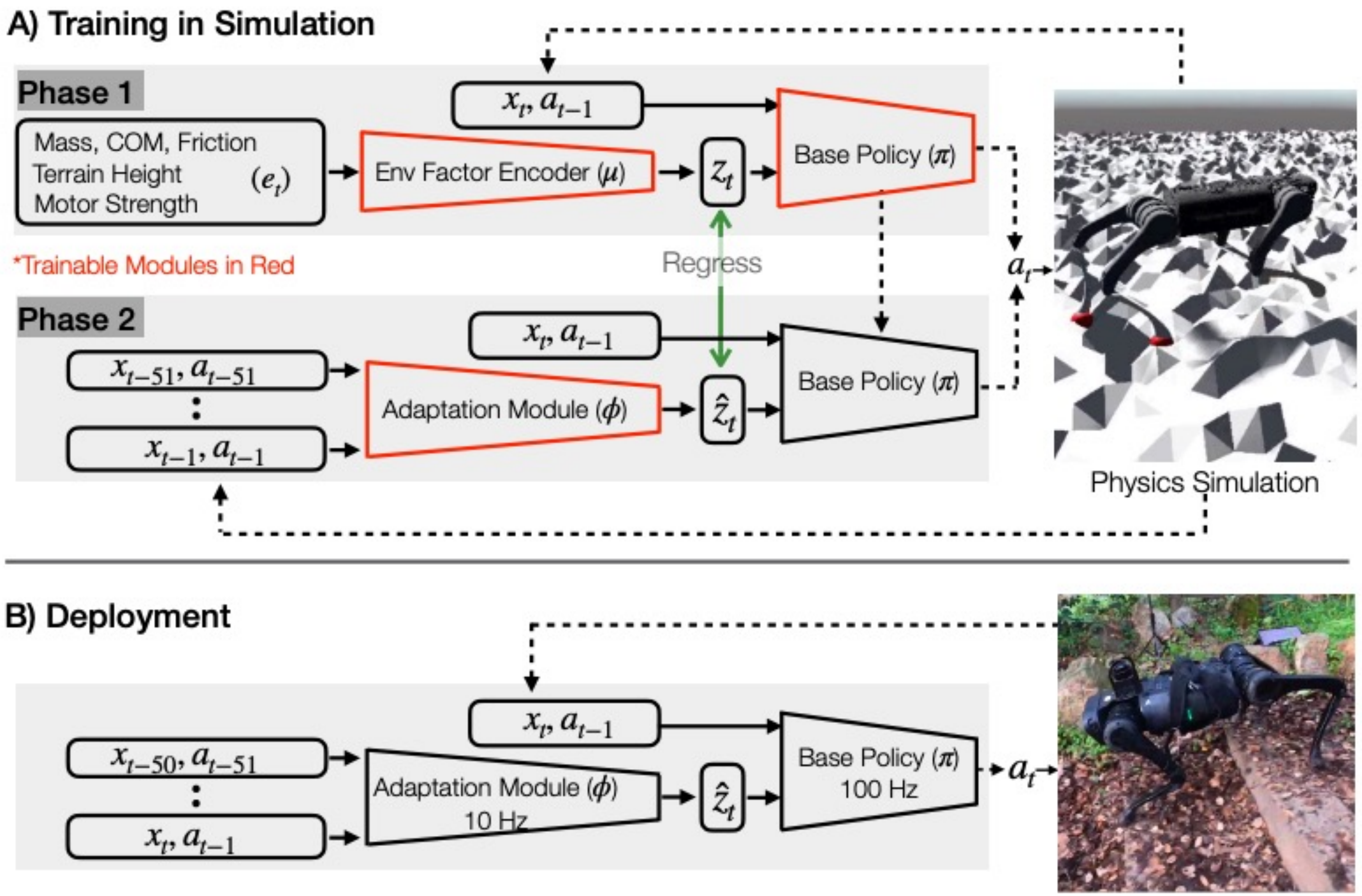
Update Policy

Update target value network

use the minimum of Q-functions for the value gradient

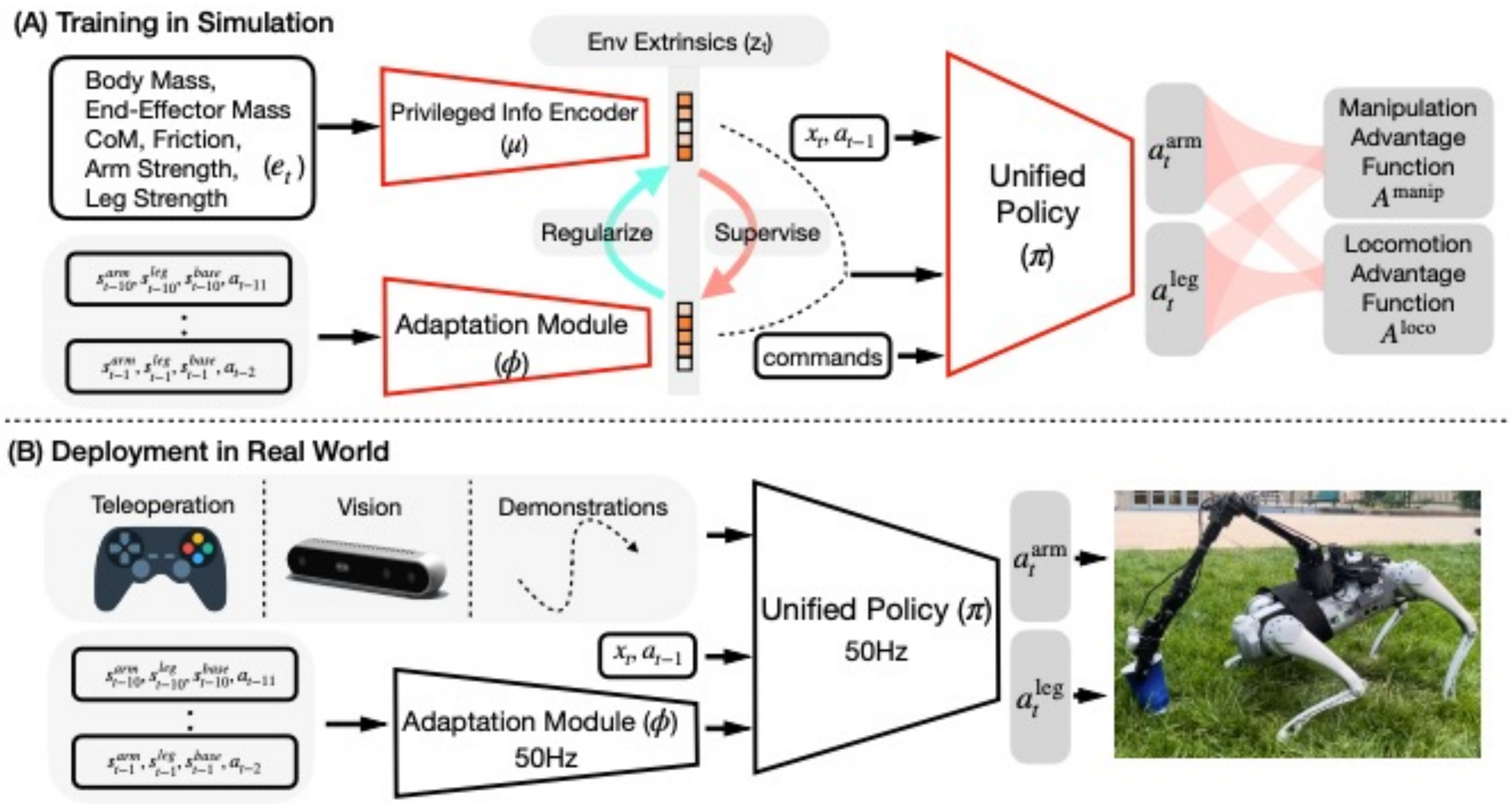
Domain Adaptation for Quadruped Robot

- **Unobservable Privileged Information**
  - a base policy
  - an adaptation module
- Trained on a varied terrain (simulated) generator using bioenergetics-inspired rewards.
- Deployed on a variety of difficult terrains.



Domain Adaptation for Quadruped Robot

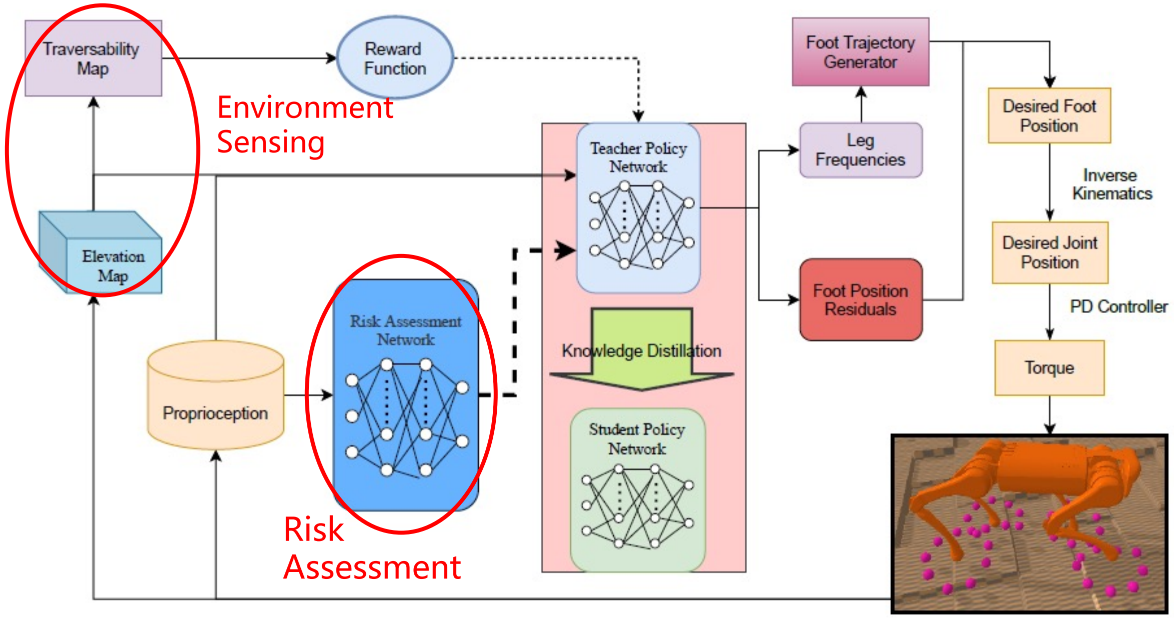
- **Unobservable Privileged Information**
  - a base policy
  - an adaptation module



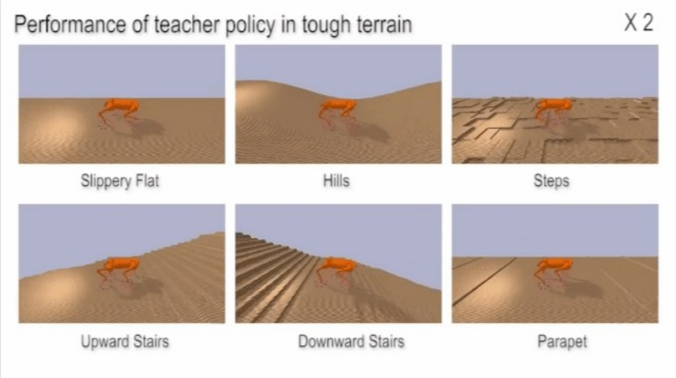
- Mobile Manipulation,
Whole-Body Control,
Legged Locomotion

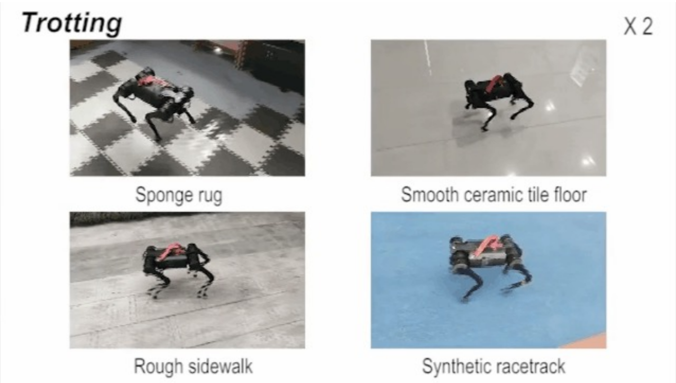
DRL-based Decision Strategy

Risk Assessment Network(RAN) in DRL for safety locomotion



Result

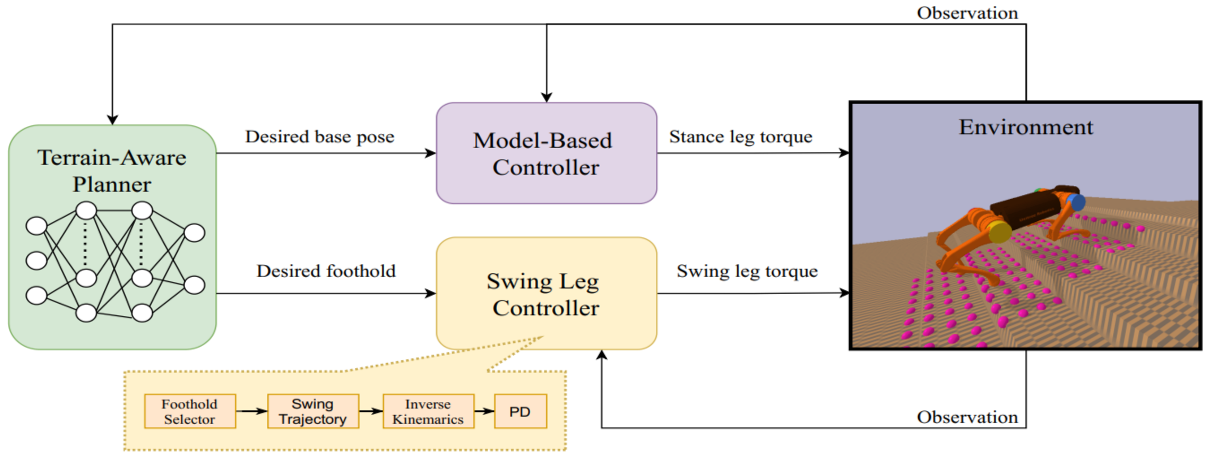


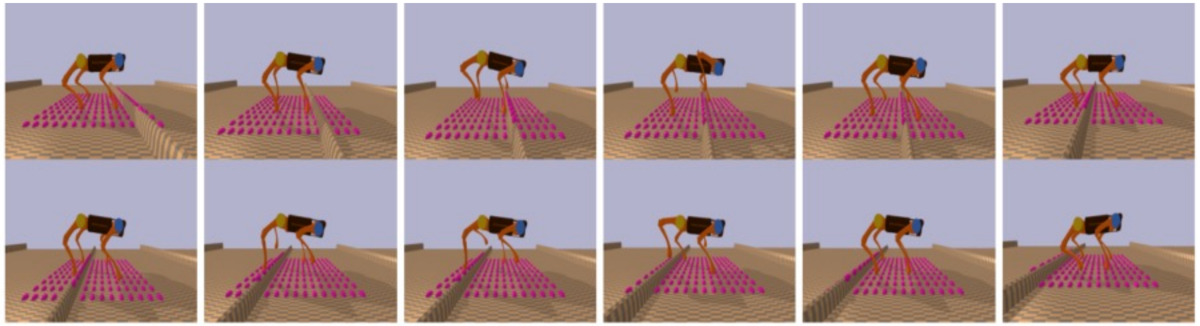


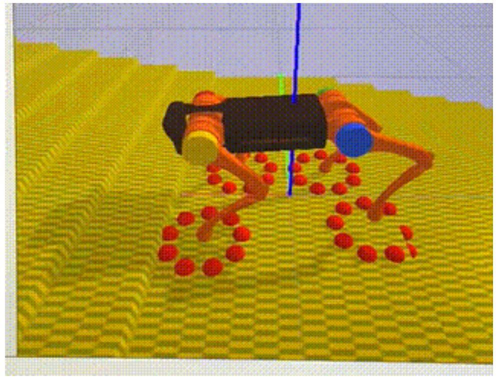
The RAN is incorporated into the model-free RL (e.g. SAC algorithm) as a penalty item δ to the loss function of the value and policy function.

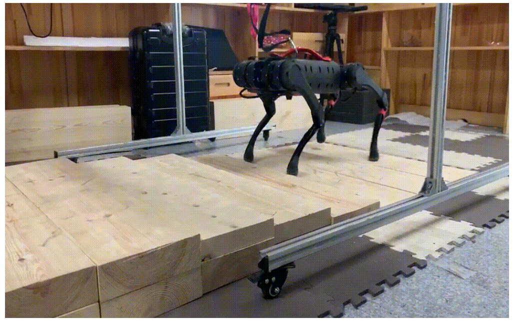
Hierarchical RL for Quadruped Robot

Hierarchical Reinforcement Learning Control Strategy



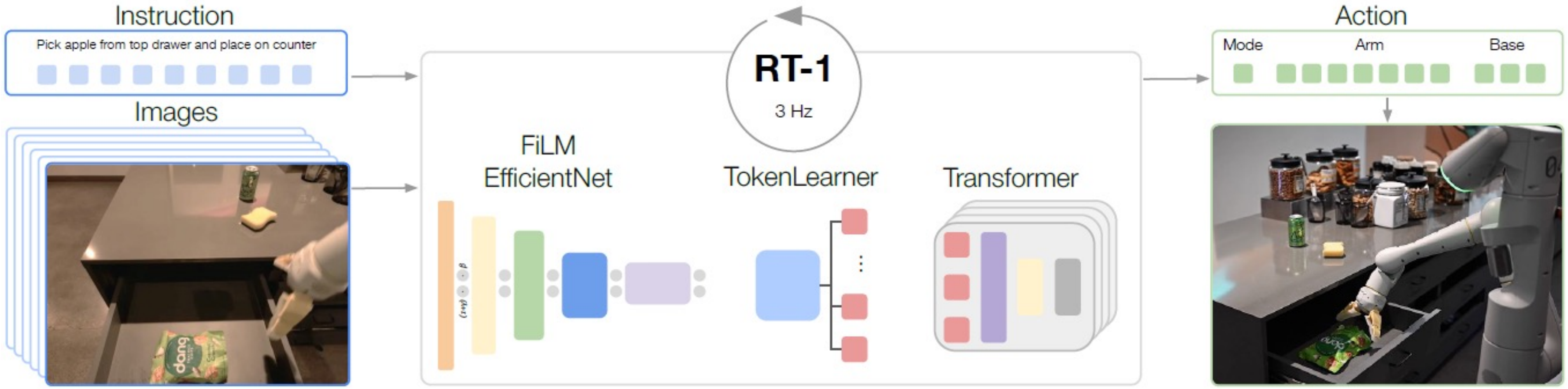






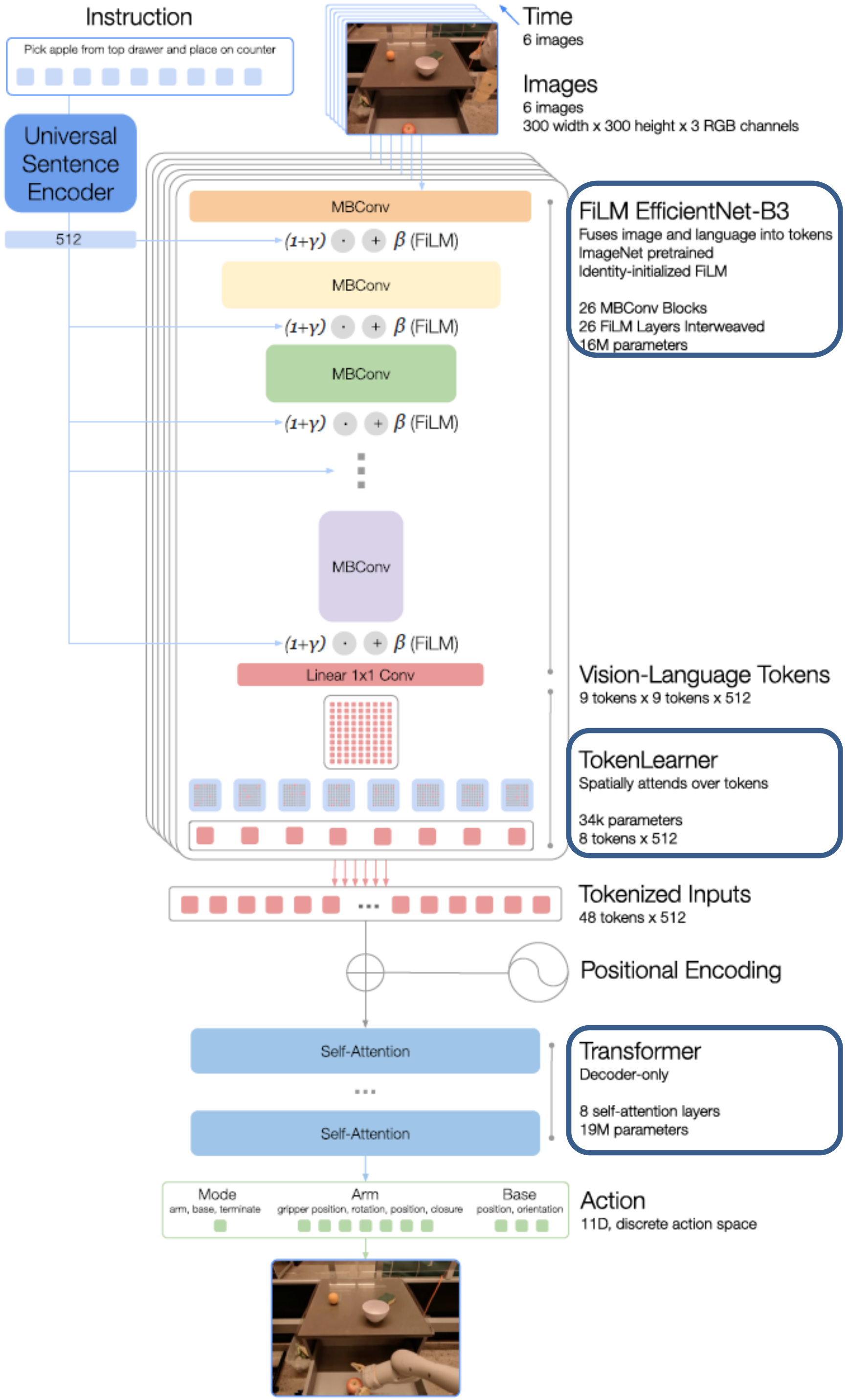
Quadruped **adjust the posture adaptively** varying the terrain changes

Robotics Transformer (RT-1)

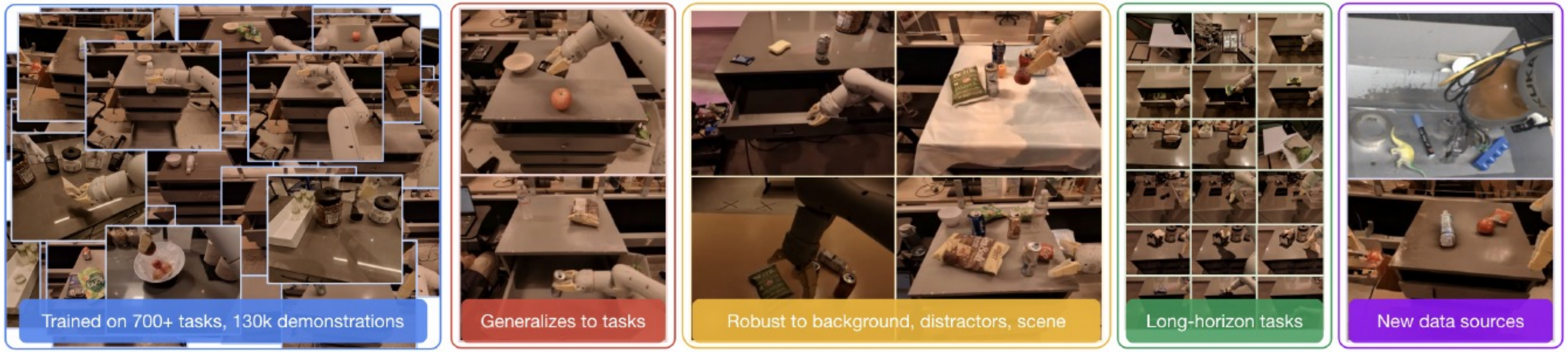


- RT-1 takes images and natural language instructions and outputs discretized base and arm actions.
- Despite its size (35M parameters), it does this at 3 Hz.
- Efficient yet high-capacity architecture:
  - A FiLM (Perez et al., 2018) conditioned EfficientNet (Tan & Le, 2019)
  - A TokenLearner (Ryoo et al., 2021)
  - A Transformer (Vaswani et al., 2017).

Robotics Transformer (RT-1)

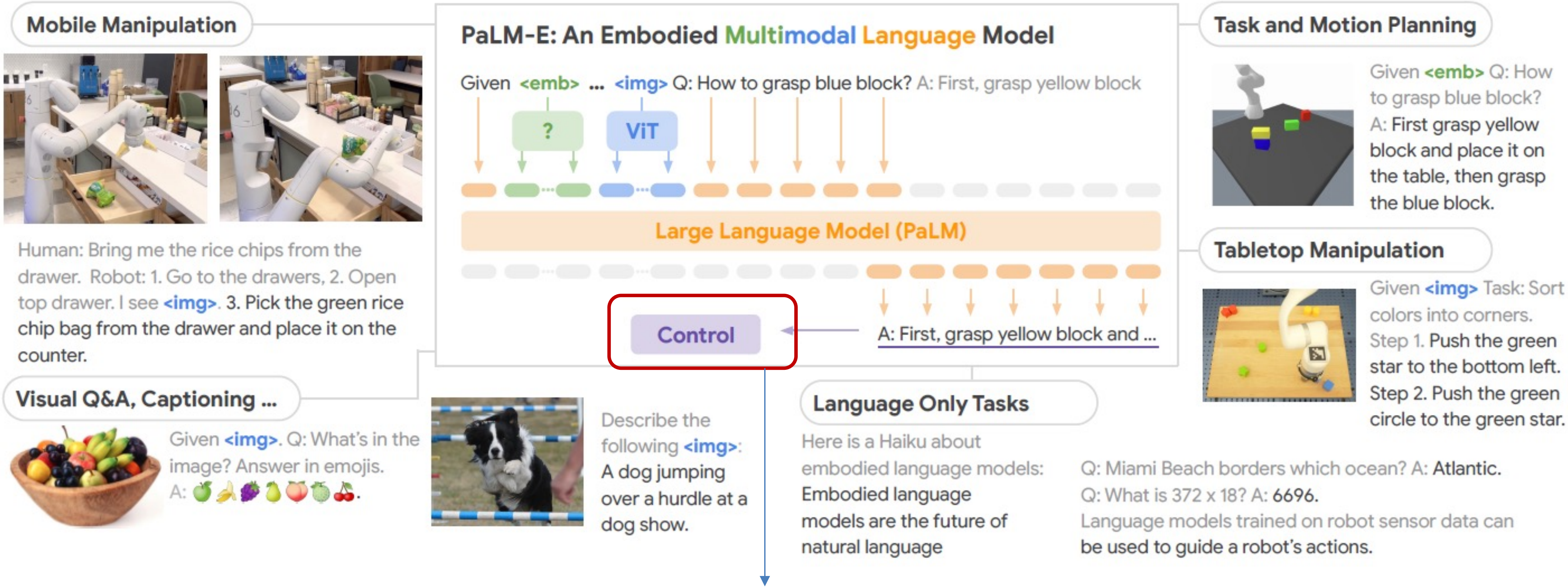


Robotics Transformer (RT-1)



- RT-1's large-scale, real-world training (130k demonstrations) and evaluation (3000 real-world trials)
- Impressive generalization, robustness, and ability to learn from diverse data

PaLM-E



Robotics Transformer (RT-1)

RT-2

- LLM + RL: RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control

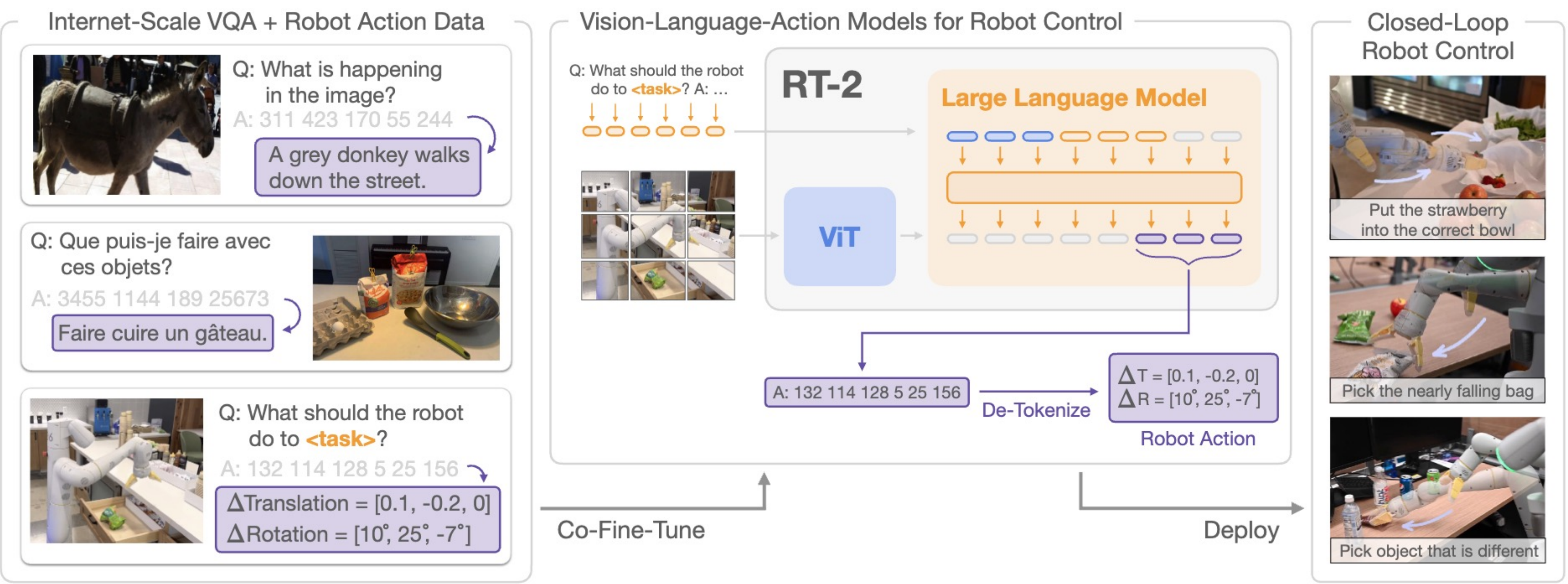
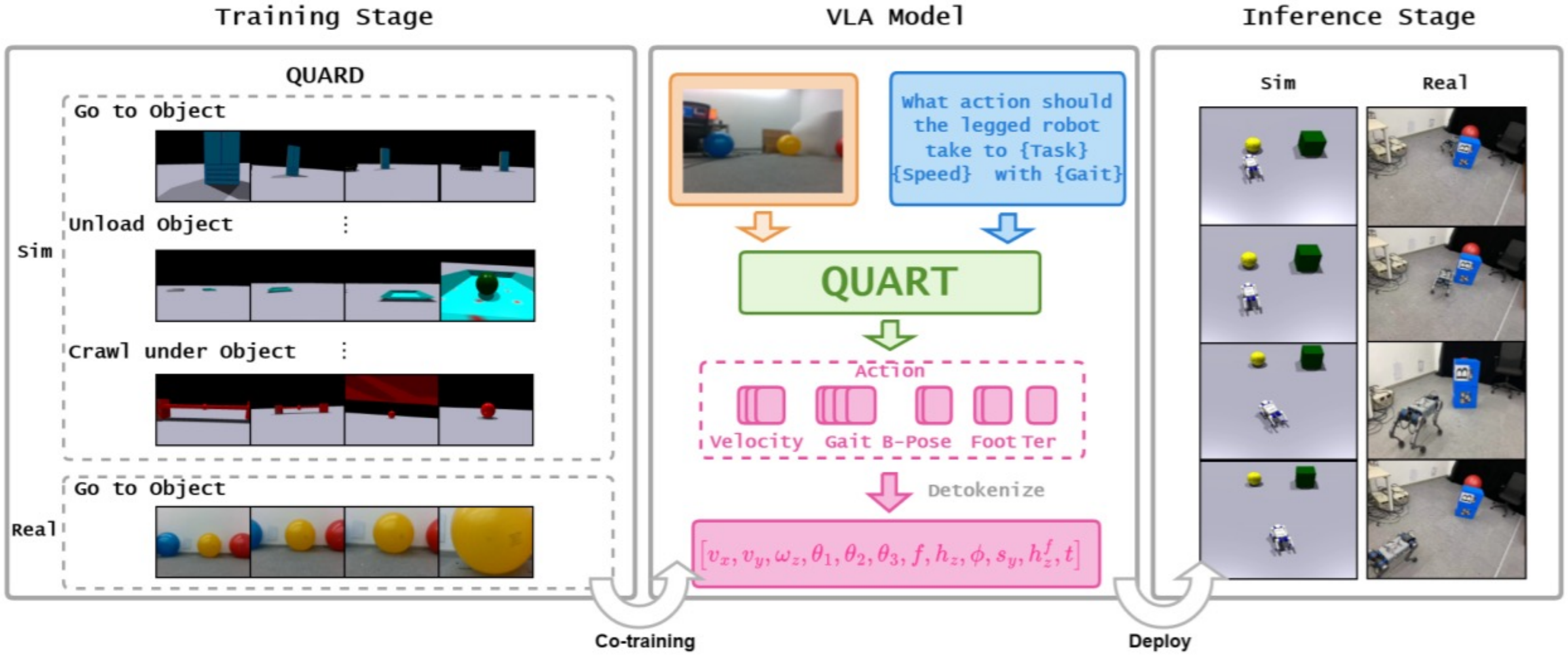


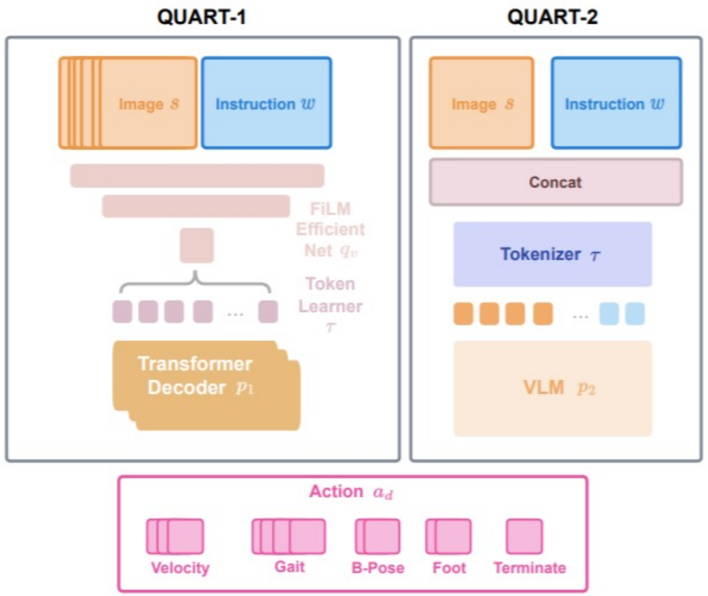
Figure 1 | RT-2 overview: we represent robot actions as another language, which can be cast into text tokens and trained together with Internet-scale vision-language datasets. During inference, the text tokens are de-tokenized into robot actions, enabling closed loop control. This allows us to leverage the backbone and pretraining of vision-language models in learning robotic policies, transferring some of their generalization, semantic understanding, and reasoning to robotic control. We demonstrate examples of RT-2 execution on the project website: robotics-transformer2.github.io.

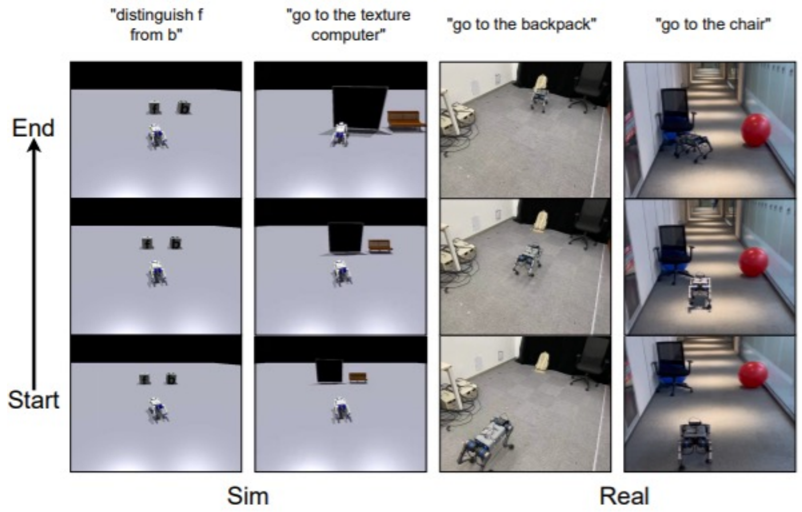

hint

QUART



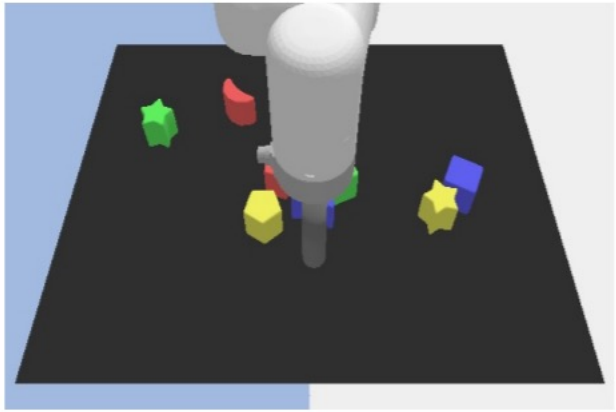
Architecture of QUART-1 and QUART-2





- By combining FiLM, TokenLearner, and a transformer decoder, QUART-1 generates discretized action tokens.
- In contrast, QUART-2 is designed to leverage the scene comprehension capability of a pretrained VLM.

Cobra



LLaVA v1.5

MobileVLM v2

Cobra

Can you describe the image in detail?

The image features a black and white checkered floor with a variety of colorful toys scattered across it. ...... Additionally, there are two toy cars, one located towards the left side of the floor and the other towards the right side. ......

The image presents a 3D rendering of a game board, which is the central focus of the image. ...... In the top right corner of the board, there's a white ball, which appears to be the central object of the game. ......

The image shows a computer-generated scene with a white, cylindrical object, possibly a piece of machinery or a robot, surrounded by various colored blocks. The blocks are of different shapes and colors, including red, yellow, and blue. The scene appears to be a computer-generated image, possibly a 3D model or a digital artwork.The image is a close-up view of the object, emphasizing its cylindrical shape and the surrounding blocks.

Conclusion

- DRL basics and Model-free DRL
- Model-based DRL
- Inverse Reinforcement Learning
- Offline Reinforcement Learning
- Large Pre-training DRL Model
- Applications to Robotics: Robot Arm and Footed Robot