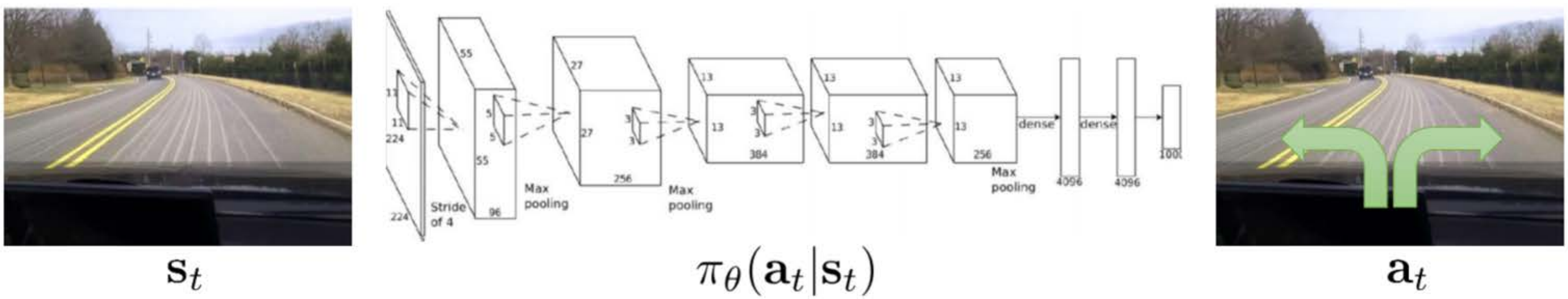
Policy Gradient Methods



Sources: [Stanford CS 231n](), [Berkeley Deep RL course](), [David Silver's RL course]()

[Image source]()

Policy Gradient Methods

- Instead of indirectly representing the policy using Q-values, it can be more efficient to parameterize and learn it directly
  - Especially in large or continuous action spaces

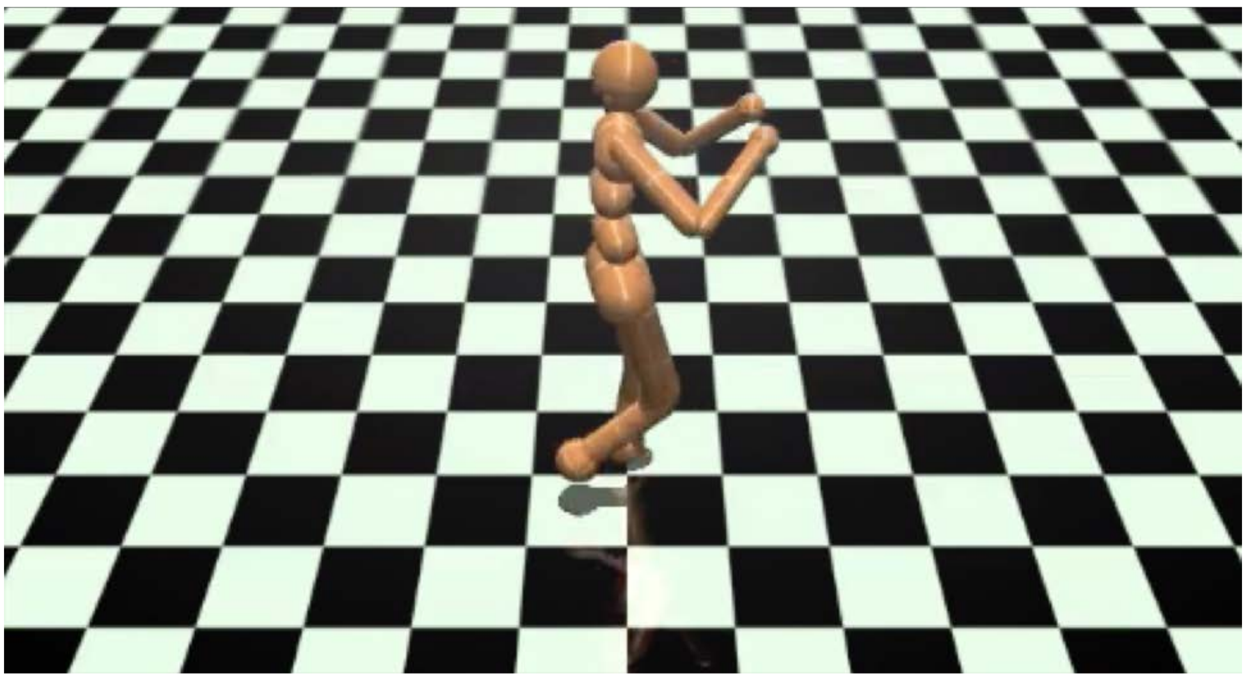


Image source: [OpenAI Gym]

Stochastic policy representation

- Learn a function giving the probability distribution over actions from the current state:

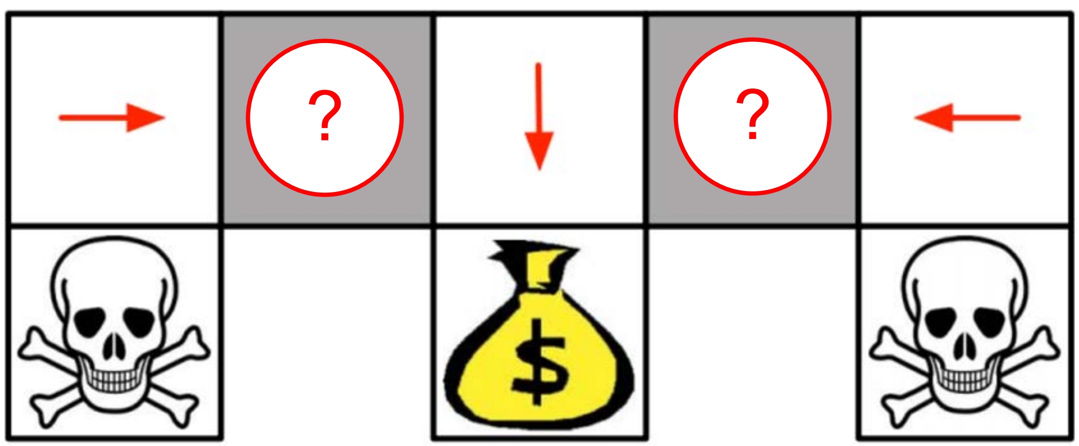
$$\pi_\theta(s, a) \approx P(a|s)$$

Stochastic policy representation

- Learn a function giving the probability distribution over actions from the current state:

$$\pi_\theta(s,a) \approx P(a|s)$$

- Why stochastic policies?
  - There are examples even of grid world scenarios where only a stochastic policy can reach optimality



The agent can’t tell the difference between the gray cells

Source: [D. Silver]

Stochastic policy representation

- Learn a function giving the probability distribution over actions from the current state:

$$\pi_\theta(s,a) \approx P(a|s)$$

- Why stochastic policies?
  - It's mathematically convenient!
    - Softmax policy:

$$\pi_\theta(s,a) = \frac{\exp(f_\theta(s,a))}{\sum_{a'} \exp(f_\theta(s,a'))}$$

    - Gaussian policy (for continuous action spaces):

$$\pi_\theta(s,a) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a - f_\theta(s))^2}{2\sigma^2}\right)$$

Expected value of a policy

$$J(\theta) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t \, r_t \mid \pi_\theta\right]$$

$$= \mathbb{E}_\tau[r(\tau)]$$

Expectation of return over *trajectories* $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots)$

$$= \int_\tau r(\tau)\underbrace{p(\tau;\theta)}_{\substack{\text{Probability of trajectory } \tau \\ \text{under policy with} \\ \text{parameters } \theta}}d\tau$$

Finding the policy gradient

$$J(\theta) = \int_\tau r(\tau) p(\tau;\theta) d\tau$$

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) d\tau$$

$$= \int_\tau r(\tau) p(\tau;\theta) \boxed{\frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)}} d\tau$$

$$= \int_\tau r(\tau) p(\tau;\theta) \nabla_\theta \log p(\tau;\theta) \, d\tau$$

$$= \mathbb{E}_\tau [r(\tau) \nabla_\theta \log p(\tau;\theta)]$$

Finding the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau\left[r(\tau)\nabla_\theta \log \underbrace{p(\tau;\theta)}_{\substack{\text{Probability of trajectory} \\ \tau = (s_0, a_0, s_1, a_1, \ldots)}}\right]$$

$$p(\tau;\theta) = \prod_{t\geq 0} \pi_\theta(s_t, a_t) P(s_{t+1}|s_t, a_t)$$

$$\log p(\tau;\theta) = \sum_{t\geq 0}\left[\log \pi_\theta(s_t, a_t) + \log P(s_{t+1}|s_t, a_t)\right]$$

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \underbrace{\nabla_\theta \log \pi_\theta(s_t, a_t)}_{\text{The } \textit{score function}}$$

Score function $\nabla_\theta \log \pi_\theta(s,a)$

- For softmax policy:

$$\pi_\theta(s,a) = \frac{\exp(f_\theta(s,a))}{\sum_{a'} \exp(f_\theta(s,a'))}$$

$$\nabla_\theta \log \pi_\theta(s_t,a_t) = \nabla_\theta f_\theta(s,a) - \sum_{a'} \pi_\theta(s,a')\, \nabla_\theta f_\theta(s,a')$$

- For Gaussian policy:

$$\pi_\theta(s,a) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a - f_\theta(s))^2}{2\sigma^2}\right)$$

$$\nabla_\theta \log \pi_\theta(s_t,a_t) = \frac{\left(a - f_\theta(s)\right)}{\sigma^2} \nabla_\theta f_\theta(s) - \text{const.}$$

Finding the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(s_t, a_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau\left[\underbrace{\left(\sum_{t\geq 0}\gamma^t r_t\right)}_{\text{Return of trajectory } \tau}\underbrace{\left(\sum_{t\geq 0}\nabla_\theta \log \pi_\theta(s_t, a_t)\right)}_{\text{Gradient of log-likelihood of actions under current policy}}\right]$$

- How do we estmate the gradient in practice?

Finding the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau[r(\tau)\nabla_\theta \log p(\tau;\theta)]$$

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t\geq 0} \nabla_\theta \log \pi_\theta(s_t, a_t)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau\left[\left(\sum_{t\geq 0}\gamma^t r_t\right)\left(\sum_{t\geq 0}\nabla_\theta \log \pi_\theta(s_t, a_t)\right)\right]$$

- Stochastic approximation: sample N trajectories $\tau_1, \ldots, \tau_N$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{T_i}\gamma^t r_{i,t}\right)\left(\sum_{t=0}^{T_i}\nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})\right)$$

REINFORCE algorithm

1. Sample N trajectories τ_i using current policy π_θ

2. Estimate the policy gradient:

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} r(\tau_i)\left(\sum_{t=0}^{T_i} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t})\right)$$

3. Update parameters by gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

Williams et al. [Simple statistical gradient-following algorithms for connectionist reinforcement learning](). Machine Learning, 8(3):229-256, 1992

REINFORCE: Single-step version

1. In state s, sample action a using current policy π_θ, observe reward r

2. Estimate the policy gradient:
$$\nabla_\theta J(\theta) \approx r \, \nabla_\theta \log \pi_\theta(s, a)$$

3. Update parameters by gradient ascent:
$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

- What effect does this update have?
  - Push up the probability of good actions, push down probability of bad actions

Reducing variance

- Gradient estimate (for a single trajectory):

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau)\, \nabla_\theta \log \pi_\theta(s_t, a_t)$$

- **General problem**: rewards of sampled trajectories are too noisy and lead to unreliable policy gradients

Reducing variance

- Gradient estimate (for a single trajectory):

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\, \nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Observation: it seems bad to weight each action in a trajectory by the return of the entire trajectory. In particular, rewards obtained *before* an action was taken should not be used to weight that action
  - Instead, for each action, consider only the cumulative *future* reward:

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t'\geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(s_t, a_t)$$

Reducing variance

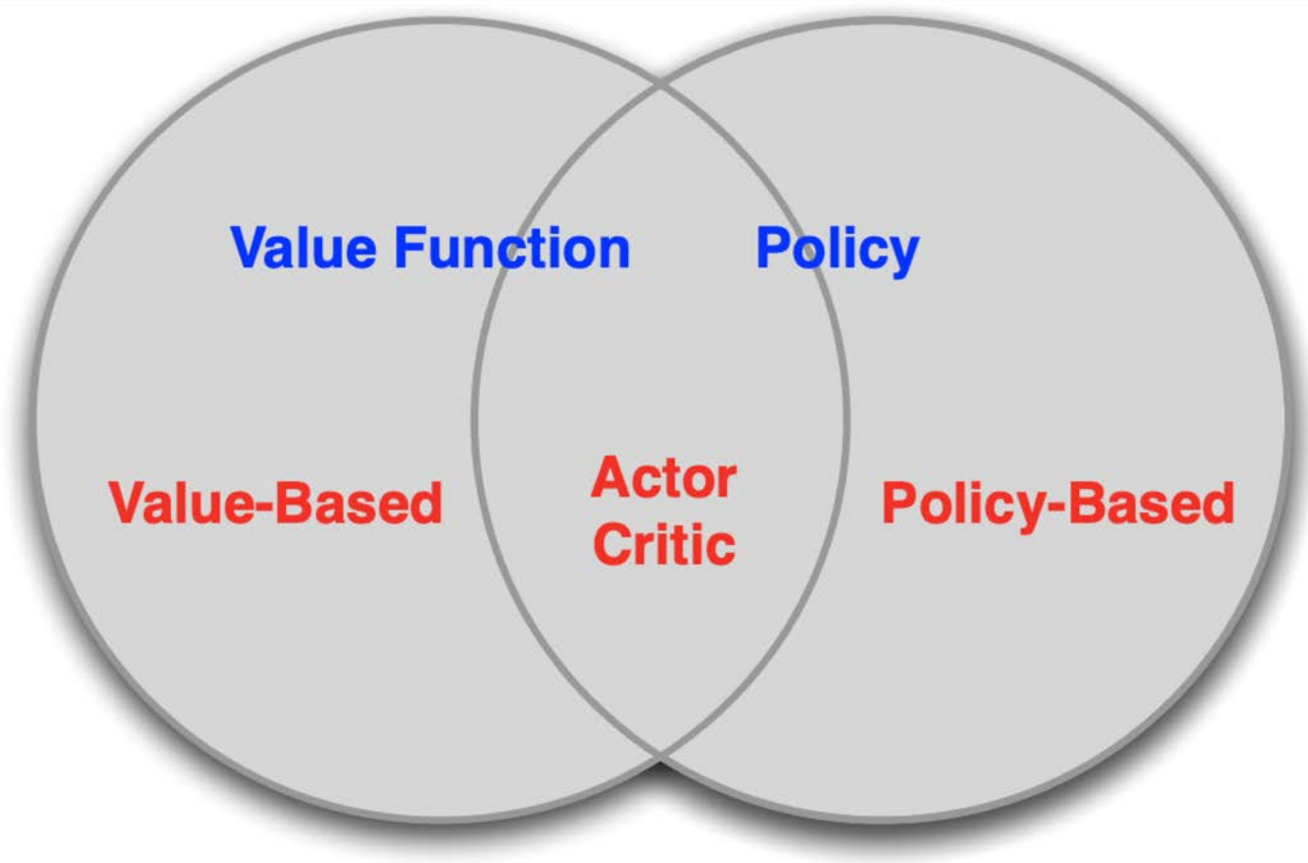
$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \underbrace{\left(\sum_{t'\geq t} \gamma^{t'-t} r_{t'}\right)}_{\text{Observed cumulative reward after taking action } a_t \text{ in state } s_t} \nabla_\theta \log \pi_\theta(s_t, a_t)$$

- But then, why not use *expected* cumulative reward?

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} Q^{\pi_\theta}(s_t, a_t)\, \nabla_\theta \log \pi_\theta(s_t, a_t)$$

Actor-Critic algorithm

- Combine policy gradients and Q-learning by simultaneously training an *actor* (the policy) and a *critic* (the Q-function)



Source: [D. Silver]

Reducing variance

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} Q^{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(s_t, a_t)$$

- Next observation: the raw Q-values are not the most useful. If all Q-values are good, we will try to push up the probabilities of all the actions
  - Instead, introduce a *baseline function* to represent whether the current value is better or worse than what we expect to get

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \underbrace{\left(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)\right)}_{\text{Advantage function}} \nabla_\theta \log \pi_\theta(s_t, a_t)$$

Estimating the advantage function

- Advantage function:

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

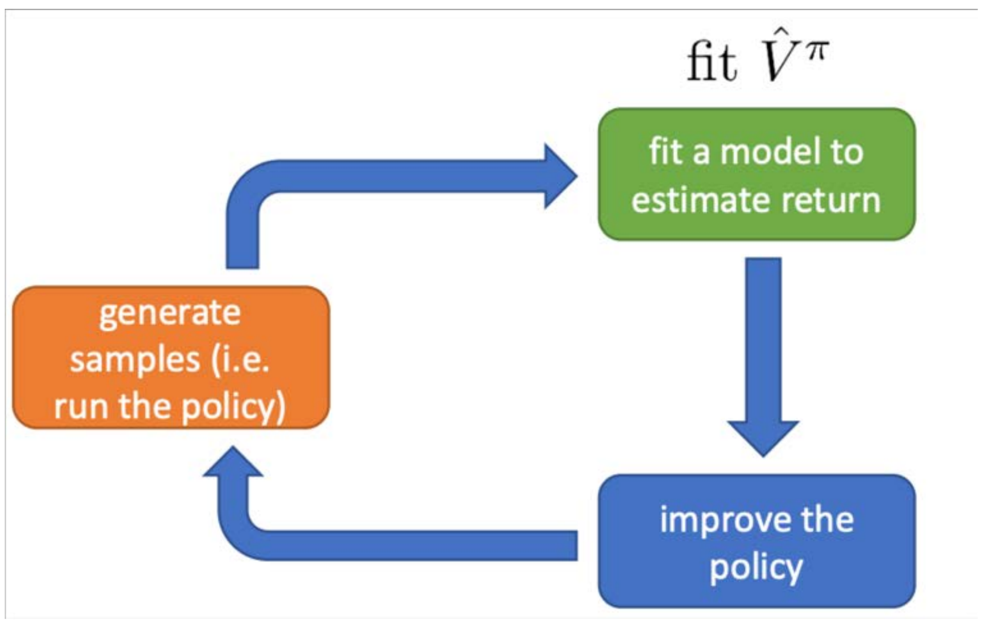
(from a single transition)

- Therefore, it is sufficient to learn the value function:

$$V^{\pi_\theta}(s) \approx V_w(s)$$

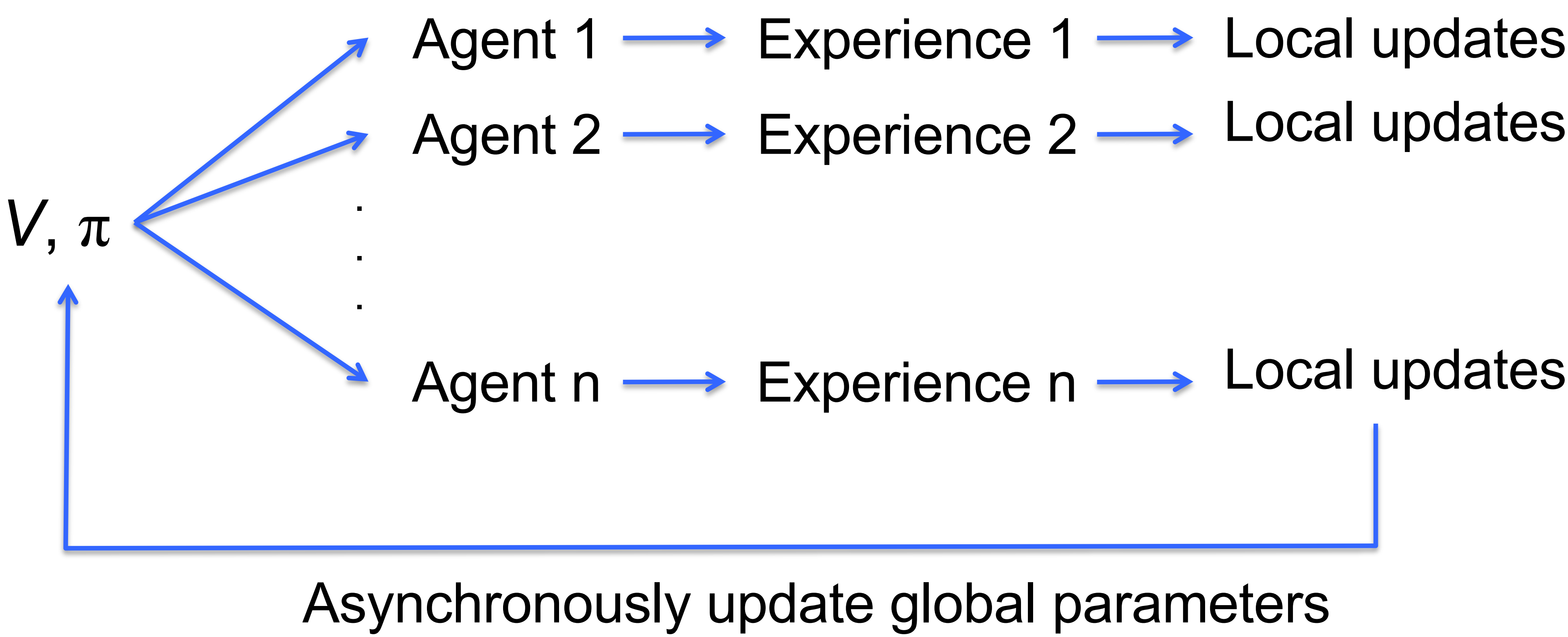
Online actor-critic algorithm

1. Sample action a using current policy, observe reward r, successor state s'
2. Update $V_w(s)$ towards target $r + \gamma V_w(s')$
3. Estimate $A^{\pi_\theta}(s,a) = r + \gamma V_w(s') - V_w(s)$
4. Estimate $\nabla_\theta J(\theta) = A^{\pi_\theta}(s,a) \nabla_\theta \log \pi_\theta(s,a)$
5. Update policy parameters: $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$



Source: Berkeley RL course

Asynchronous advantage actor-critic (A3C)



Mnih et al. [Asynchronous Methods for Deep Reinforcement Learning](). ICML 2016

Asynchronous advantage actor-critic (A3C)

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Mean and median human-normalized scores over 57 Atari games

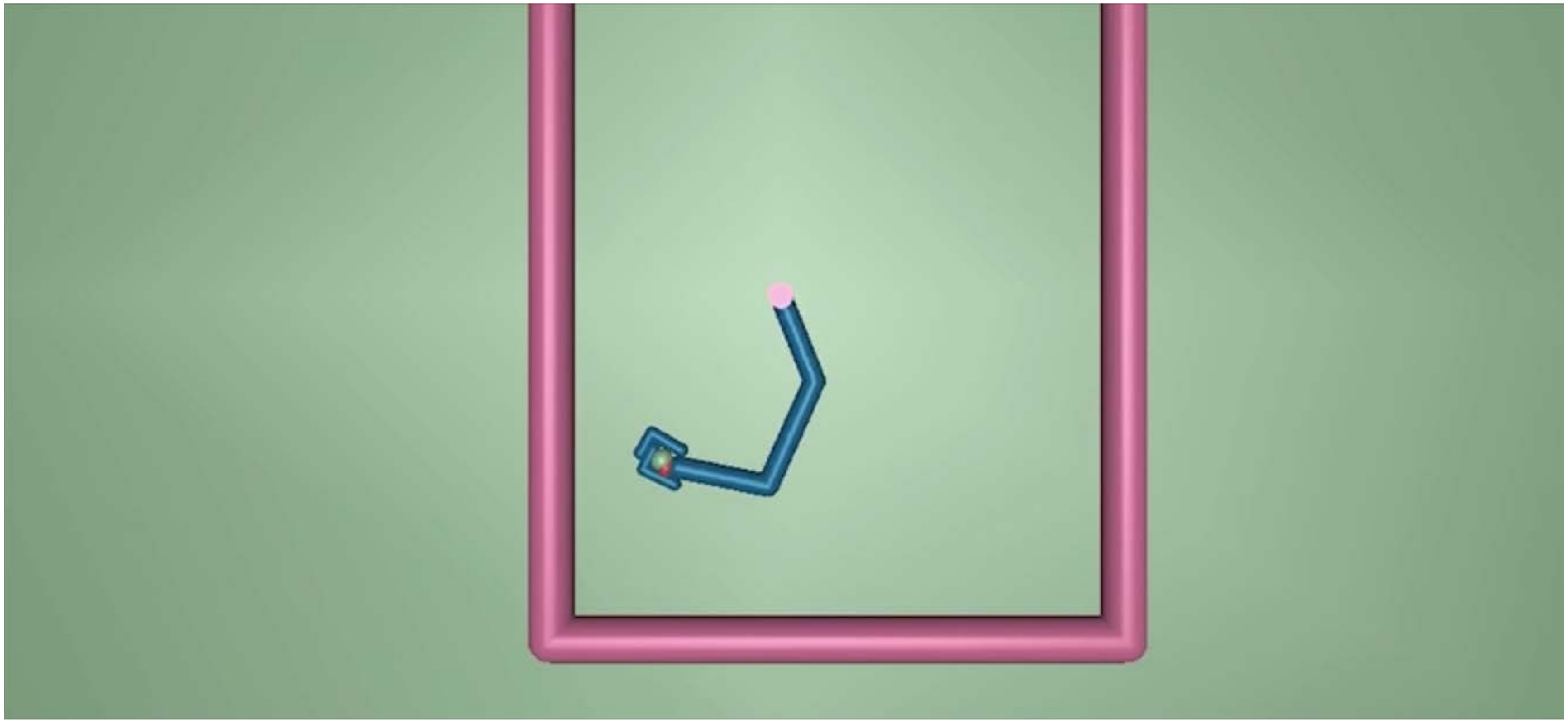
Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Asynchronous advantage actor-critic (A3C)

TORCS car racing simulation video

Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Asynchronous advantage actor-critic (A3C)

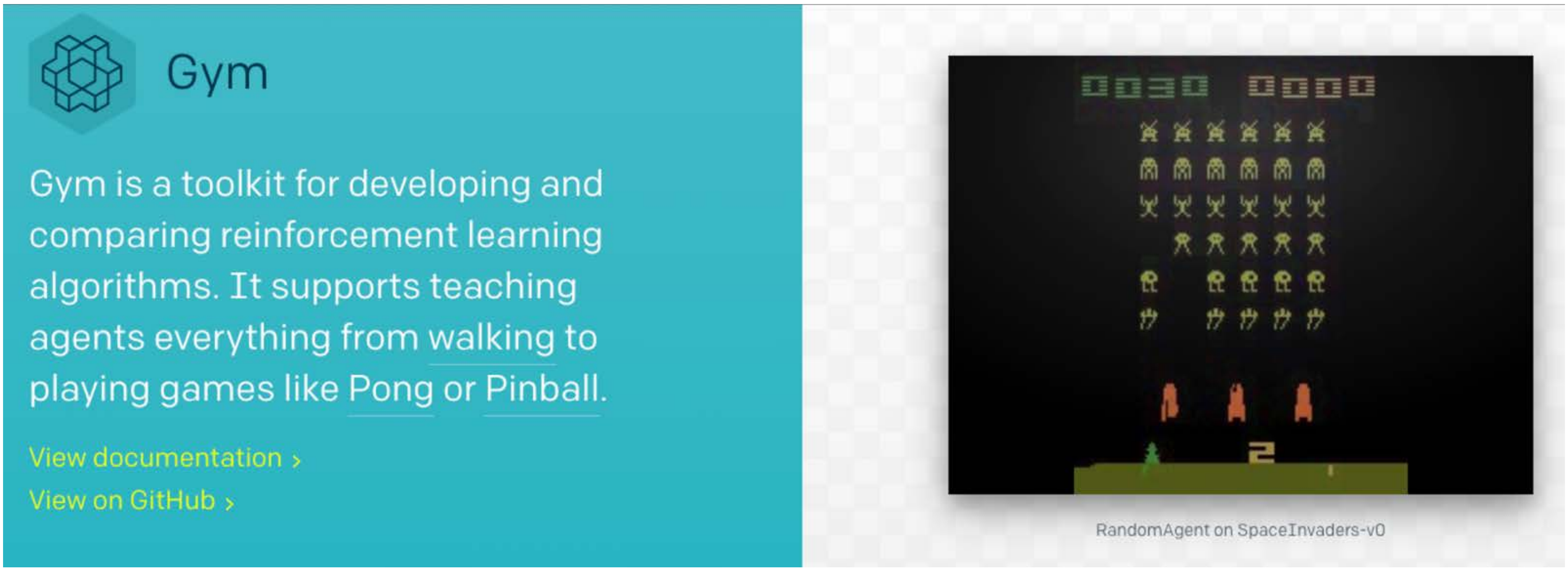


Motor control tasks video

Mnih et al. Asynchronous Methods for Deep Reinforcement Learning. ICML 2016

Benchmarks and environments for Deep RL

OpenAI Gym



https://gym.openai.com/