

Deep Reinforcement Learning and Control

# Policy gradients III

CMU 10-403

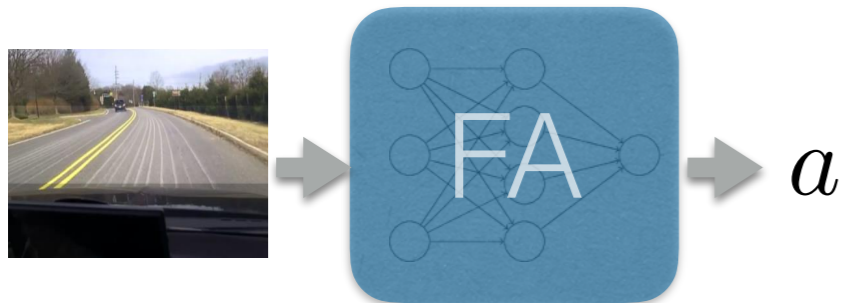
Katerina Fragkiadaki



Revision

# Policy function approximators - this lecture

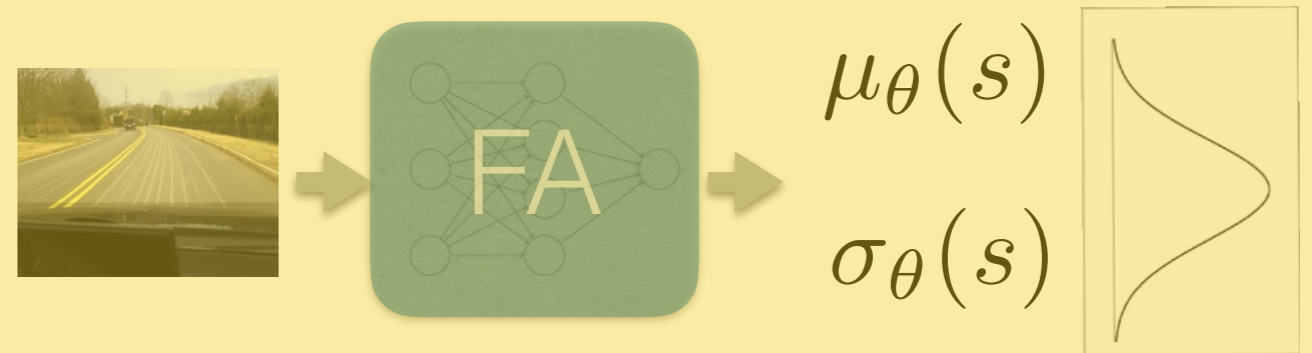
deterministic continuous policy



$$a = \pi_{\theta}(s)$$

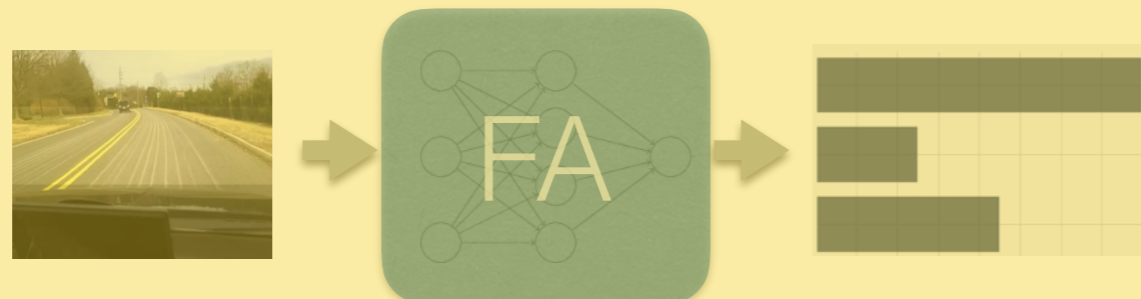
e.g. outputs a steering angle directly

stochastic continuous policy



$$a \sim \mathcal{N}(\mu_{\theta}(s), \sigma_{\theta}^2(s))$$

(stochastic) discrete actions

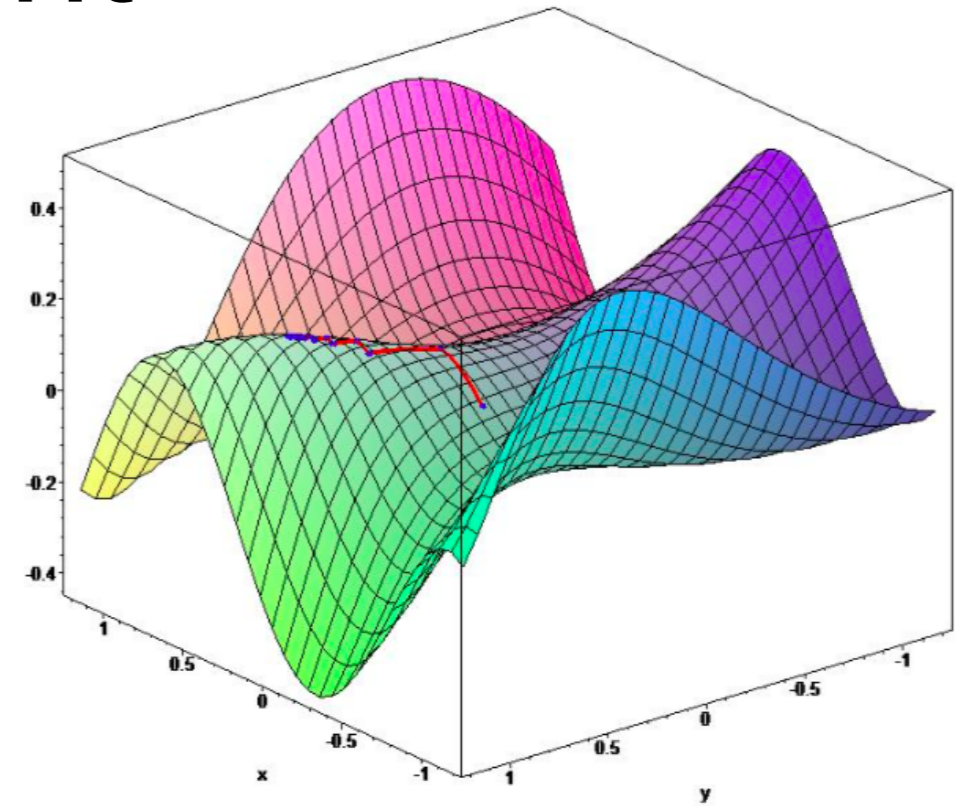


go left  
go right  
press brake

Outputs a distribution over a discrete set of actions

# Policy Gradient

- ▶ Let  $U(\theta)$  be any policy **objective function**
- ▶ Policy gradient algorithms search for a **local** maximum in  $U(\theta)$  by ascending the gradient of the policy, w.r.t. parameters  $\theta$



$$\theta_{new} = \theta_{old} + \Delta\theta$$

$$\Delta\theta = \alpha \nabla_{\theta} U(\theta)$$

$\alpha$  is a step-size parameter (learning rate)

$\nabla_{\theta} U(\theta)$  is the **policy gradient**

$$\nabla_{\theta} U(\theta) = \begin{pmatrix} \frac{\partial U(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial U(\theta)}{\partial \theta_n} \end{pmatrix}$$

Policy gradient: **the gradient of the policy objective w.r.t. the parameters of the policy**

# Gradients of expectations

Computing derivatives of expectations w.r.t. variables that parameterize the distribution, not the quantity inside the expectation. Conditioned on the samples, we can estimate the expectation without knowing theta

$$\max_{\theta} \cdot \mathbb{E}_{x \sim P(x; \theta)} f(x)$$

Later today

$$y = P_{\theta}(x) \\ \max_{\theta} \cdot f(P_{\theta}(x))$$

Assumptions:

- P is a probability density function that is continuous and differentiable
- P is easy to sample from

$$\max_{\theta} \cdot \mathbb{E}_{\tau \sim P_{\theta}(\tau)} [R(\tau)]$$

# Likelihood ratio gradient estimator

$$\max_{\theta} . U(\theta) = \mathbb{E}_{\tau \sim P_{\theta}(\tau)} [R(\tau)]$$

$$\begin{aligned} \nabla_{\theta} U(\theta) &= \nabla_{\theta} \mathbb{E}_{\tau \sim P_{\theta}(\tau)} [R(\tau)] \\ &= \mathbb{E}_{\tau \sim P_{\theta}(\tau)} [\nabla_{\theta} \log P_{\theta}(\tau) R(\tau)] \end{aligned}$$

An unbiased estimator of this gradient:

$$\hat{g} = \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} \log P_{\theta}(\tau^{(i)}) R(\tau^{(i)}) = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) R(\tau^{(i)})$$

$$\mathbb{E}[\hat{g}] = \nabla_{\theta} U(\theta)$$

# Temporal structure

$$\hat{g} = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) \left( \sum_{k=t}^H R(s_k^{(i)}, a_k^{(i)}) \right)$$

Each action takes the blame for the trajectory that comes after it

We can call this the return from t onwards  $G_t$

# Likelihood ratio gradient estimator

- ▶ Let's analyze the update:

$$\Delta\theta_t = \alpha G_t \nabla_{\theta} \log \pi_{\theta}(s_t, a_t)$$

- ▶ Let's us rewrite is as follows:

$$\theta_{t+1} \doteq \theta_t + \alpha \gamma^t G_t \frac{\nabla_{\theta} \pi(A_t | S_t, \theta)}{\pi(A_t | S_t, \theta)}$$

- ▶ Update is proportional to:
  - the product of a return  $G_t$  and
  - the gradient of the probability of taking the action actually taken,
  - divided by the probability of taking that action.



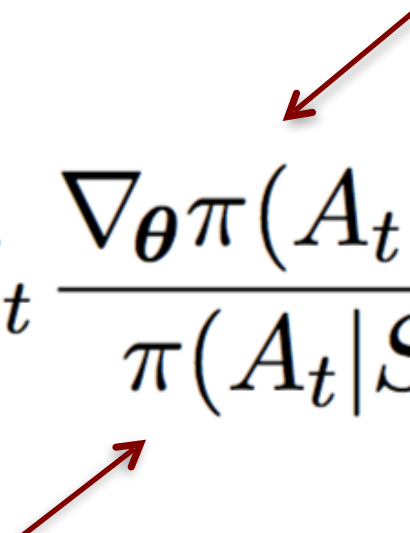
# Likelihood ratio gradient estimator

- ▶ Let's analyze the update:

$$\Delta\theta_t = \alpha G_t \nabla_{\theta} \log \pi_{\theta}(s_t, a_t)$$

- ▶ Let's us rewrite is as follows:

move most in the directions that favor actions that yield **the highest return**

$$\theta_{t+1} \doteq \theta_t + \alpha \gamma^t G_t \frac{\nabla_{\theta} \pi(A_t | S_t, \theta)}{\pi(A_t | S_t, \theta)}$$


Update is **inversely proportional to the action probability** to fight the fact that actions that are selected frequently are at an advantage (the updates will be more often in their direction)

# Variance

Variance of a random variable:

$$\text{Var}(X) = \mathbb{E} [(X - \mathbb{E}[X])^2]$$

Here we have a random vector:

$$\hat{g} = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) G_t^{(i)}$$

Variance is the trace of the covariance matrix:

$$\text{Var}(\hat{g}) = \text{tr} \left( \mathbb{E} [(\hat{g} - \mathbb{E}[\hat{g}])(\hat{g} - \mathbb{E}[\hat{g}])^T] \right) = \sum_{k=1}^n \mathbb{E} \left[ (\hat{g}_k - \mathbb{E}[\hat{g}_k])^2 \right]$$

Our goal is to minimize the variance

# Baseline choices

$$\hat{g} = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) \left( \sum_{k=t}^H R(s_k^{(i)}, a_k^{(i)}) - b \right)$$

- Constant baseline:  $b = \mathbb{E}[R(\tau)] \approx \sum_{i=1}^N R(\tau^{(i)})$
- Time-dependent baseline:  $b_t = \sum_{i=1}^N \sum_{k=t}^H R(s_k^{(i)}, a_k^{(i)})$
- State-dependent expected return:

$$b(s_t) = \mathbb{E} [r_t + r_{t+1} + r_{t+2} + \dots + r_{H-1}] = V_{\pi}(s_t)$$

$$\hat{g} = \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) (G_t^i - b(s_t^{(i)}))$$

$$\hat{g} = \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) (G_t^i - V^{\pi}(s_t^{(i)}))$$

# Variance

$$\hat{g} = \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) (G_t^i - b(s_t^{(i)}))$$

$$\text{Var}(\hat{g}) = \text{tr} \left( \mathbb{E} \left[ (\hat{g} - \mathbb{E}[\hat{g}])(\hat{g} - \mathbb{E}[\hat{g}])^T \right] \right) = \sum_{k=1}^n \mathbb{E} \left[ (\hat{g}_k - \mathbb{E}[\hat{g}_k])^2 \right]$$

- Imagine in some state  $S_1$  the rewards of all actions are  $\sim 3000$  and in  $S_2 \sim -4000$ .
- Now imagine you have  $S_1 = 3000$  and  $S_2 = -4000$ . I have much reduced the variance.
- We want to encourage an action, not when it has high return, but when it has higher return than the state expected return  $b(s)$ , i.e., when making this action MORE probable would allow me to improve over what I get now from the state.

# Estimate $V_{\pi}(s_t)$

$$\hat{g} = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) \left( \sum_{k=t}^H R(s_k^{(i)}, a_k^{(i)}) - V^{\pi}(s_k^{(i)}) \right)$$

## MC estimation

Initialize  $\phi$

- Collect trajectories  $\tau_1, \dots, \tau_N$
- Regress against empirical return:

$$\phi_{i+1} \leftarrow \arg \min_{\phi} \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{H-1} \left( V_{\phi}^{\pi}(s_t^{(i)}) - \left( \sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)}) \right) \right)^2$$

# Estimate $V_{\pi}(s_t)$

$$\hat{g} = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) \left( \sum_{k=t}^H R(s_k^{(i)}, a_k^{(i)}) - V^{\pi}(s_k^{(i)}) \right)$$

## TD estimation

Initialize  $\phi$

- Collect data  $\{s, u, s', r\}$
- Fitted V iteration:

$$\phi_{i+1} \leftarrow \min_{\phi} \sum_{(s, u, s', r)} \|r + V_{\phi_i}^{\pi}(s') - V_{\phi}(s)\|_2^2 + \lambda \|\phi - \phi_i\|_2^2$$

Bootstrapping!

# Better estimates for cumulative future reward

$$\hat{g} = \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(\alpha_t^{(i)} | s_t^{(i)}) \left( \sum_{k=t}^H R(s_k^{(i)}, a_k^{(i)}) - V^{\pi}(s_k^{(i)}) \right)$$

We are essentially attempting to estimate Q from a single rollout:

$$Q^{\pi}(s, a) = \mathbb{E}[R_0 + R_1 + \dots | S_0 = s, A_0 = a]$$

Minimize variance by:

- discounting
- introducing a learnt approximation for the expected return (critic), as opposed to use MC samples

# Actor-Critic

Reducing variance using a critic

$$Q^{\pi, \gamma}(s, a) = \mathbb{E}[R_0 + \gamma R_1 + \gamma^2 R_2 \dots | S_0 = s, A_0 = a]$$



# Actor-Critic

Reducing variance using a critic

$$\begin{aligned} Q^{\pi, \gamma}(s, a) &= \mathbb{E}[R_0 + \gamma R_1 + \gamma^2 R_2 \cdots | S_0 = s, A_0 = a] \\ &= \mathbb{E}[R_0 + \gamma V^{\pi}(S_1) | S_0 = s, A_0 = a] \end{aligned}$$

# Actor-Critic

Reducing variance using a critic

$$\begin{aligned} Q^{\pi, \gamma}(s, a) &= \mathbb{E}[R_0 + \gamma R_1 + \gamma^2 R_2 \cdots | S_0 = s, A_0 = a] \\ &= \mathbb{E}[R_0 + \gamma V^\pi(S_1) | S_0 = s, A_0 = a] \\ &= \mathbb{E}[R_0 + \gamma R_1 + \gamma^2 V^\pi(S_2) | S_0 = s, A_0 = a] \end{aligned}$$

# Actor-Critic

Reducing variance using a critic

$$\begin{aligned} Q^{\pi, \gamma}(s, a) &= \mathbb{E}[R_0 + \gamma R_1 + \gamma^2 R_2 \dots | S_0 = s, A_0 = a] \\ &= \mathbb{E}[R_0 + \gamma V^\pi(S_1) | S_0 = s, A_0 = a] \\ &= \mathbb{E}[R_0 + \gamma R_1 + \gamma^2 V^\pi(S_2) | S_0 = s, A_0 = a] \\ &= \mathbb{E}[R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 V^\pi(S_3) | S_0 = s, A_0 = a] \\ &= \dots \end{aligned}$$

# REINFORCE/Actor-critic training

- Stability of training neural networks requires the **gradient updates to be de-correlated**
- This is not the case if data arrives **sequentially**
- Gradient updates computed from some part of the space can cause the value (Q) function approximator to **oscillate**
- Our solution so far has been: **Experience buffers** where experience tuples are mixed and sampled from. Resulting sampled batches are more stationary than the ones encountered online (without buffer)
- This limits deep RL to **off-policy** methods, since data from an older policy are used to update the weights of the value approximator (critic) (except if we take care and weight such data under our current stochastic policy)

# Alternative to experience buffers

## Asynchronous Methods for Deep Reinforcement Learning

---

**Volodymyr Mnih**<sup>1</sup>

**Adrià Puigdomènech Badia**<sup>1</sup>

**Mehdi Mirza**<sup>1,2</sup>

**Alex Graves**<sup>1</sup>

**Tim Harley**<sup>1</sup>

**Timothy P. Lillicrap**<sup>1</sup>

**David Silver**<sup>1</sup>

**Koray Kavukcuoglu**<sup>1</sup>

VMNIH@GOOGLE.COM

ADRIAP@GOOGLE.COM

MIRZAMOM@IRO.UMONTREAL.CA

GRAVESA@GOOGLE.COM

THARLEY@GOOGLE.COM

COUNTZERO@GOOGLE.COM

DAVIDSILVER@GOOGLE.COM

KORAYK@GOOGLE.COM

<sup>1</sup> Google DeepMind

<sup>2</sup> Montreal Institute for Learning Algorithms (MILA), University of Montreal

# Asynchronous Deep RL for on policy learning

---

## Asynchronous Methods for Deep Reinforcement Learning

---

**Volodymyr Mnih**<sup>1</sup>

**Adrià Puigdomènech Badia**<sup>1</sup>

**Mehdi Mirza**<sup>1,2</sup>

**Alex Graves**<sup>1</sup>

**Tim Harley**<sup>1</sup>

**Timothy P. Lillicrap**<sup>1</sup>

**David Silver**<sup>1</sup>

**Koray Kavukcuoglu**<sup>1</sup>

<sup>1</sup> Google DeepMind

<sup>2</sup> Montreal Institute for Learning Algorithms (MILA), University of Montreal

VMNIH@GOOGLE.COM

ADRIAP@GOOGLE.COM

MIRZAMOM@IRO.UMONTREAL.CA

GRAVESA@GOOGLE.COM

THARLEY@GOOGLE.COM

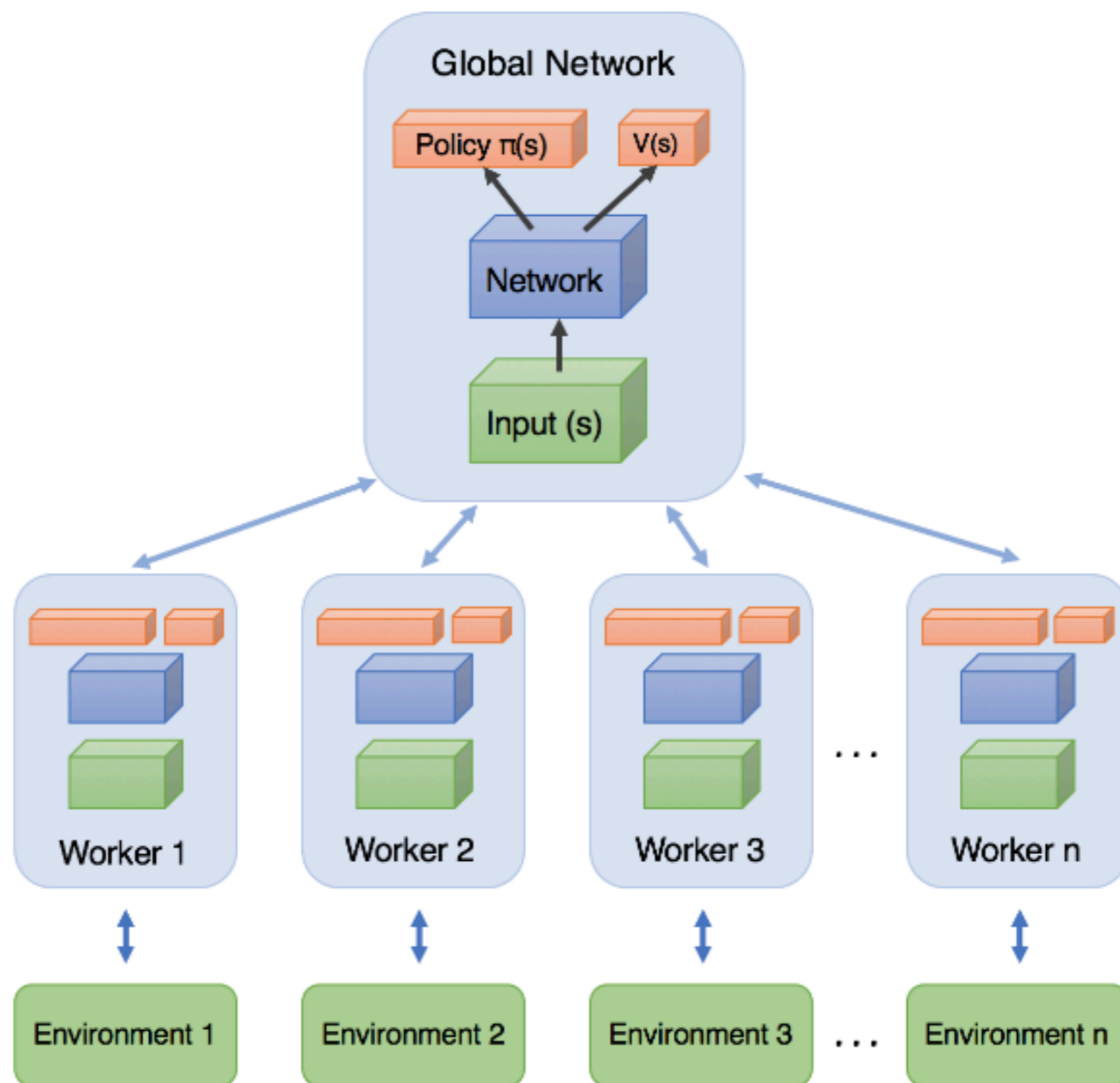
COUNTZERO@GOOGLE.COM

DAVIDSILVER@GOOGLE.COM

KORAYK@GOOGLE.COM

- Alternative: **parallelize the collection of experience and stabilize training without experience buffers**
- **Multiple threads of experience**, one per agent, each exploring in different part of the environment contributing experience tuples
- **Different exploration strategies** (e.g., various  $\epsilon$  values) in different threads increase diversity
- Now you can train on-policy, e.g., using policy gradients

# Distributed RL



---

**Algorithm S3** Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.
 

---

// Assume global shared parameter vectors  $\theta$  and  $\theta_v$  and global shared counter  $T = 0$

// Assume thread-specific parameter vectors  $\theta'$  and  $\theta'_v$

Initialize thread step counter  $t \leftarrow 1$

**repeat**

Reset gradients:  $d\theta \leftarrow 0$  and  $d\theta_v \leftarrow 0$ .

Synchronize thread-specific parameters  $\theta' = \theta$  and  $\theta'_v = \theta_v$  Copying the weights

$t_{start} = t$

Get state  $s_t$

**repeat**

Perform  $a_t$  according to policy  $\pi(a_t|s_t; \theta')$  Rollout

Receive reward  $r_t$  and new state  $s_{t+1}$

$t \leftarrow t + 1$

$T \leftarrow T + 1$

**until** terminal  $s_t$  **or**  $t - t_{start} == t_{max}$

$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t // \text{ Bootstrap from last state} \end{cases}$

**for**  $i \in \{t - 1, \dots, t_{start}\}$  **do**

$R \leftarrow r_i + \gamma R$

Accumulate gradients wrt  $\theta'$ :  $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta') (R - V(s_i; \theta'_v))$  Advantage

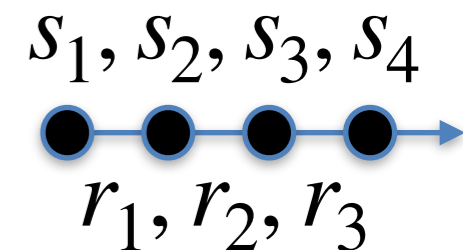
Accumulate gradients wrt  $\theta'_v$ :  $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$

**end for**

Perform asynchronous update of  $\theta$  using  $d\theta$  and of  $\theta_v$  using  $d\theta_v$ . Learning the critic

**until**  $T > T_{max}$

---



What is the approximation used for the advantage?

$$R_3 = r_3 + \gamma V(s_4, \theta'_v)$$

$$A_3 = R_3 - V(s_3; \theta'_v)$$

$$R_2 = r_2 + \gamma r_3 + \gamma^2 V(s_4, \theta'_v)$$

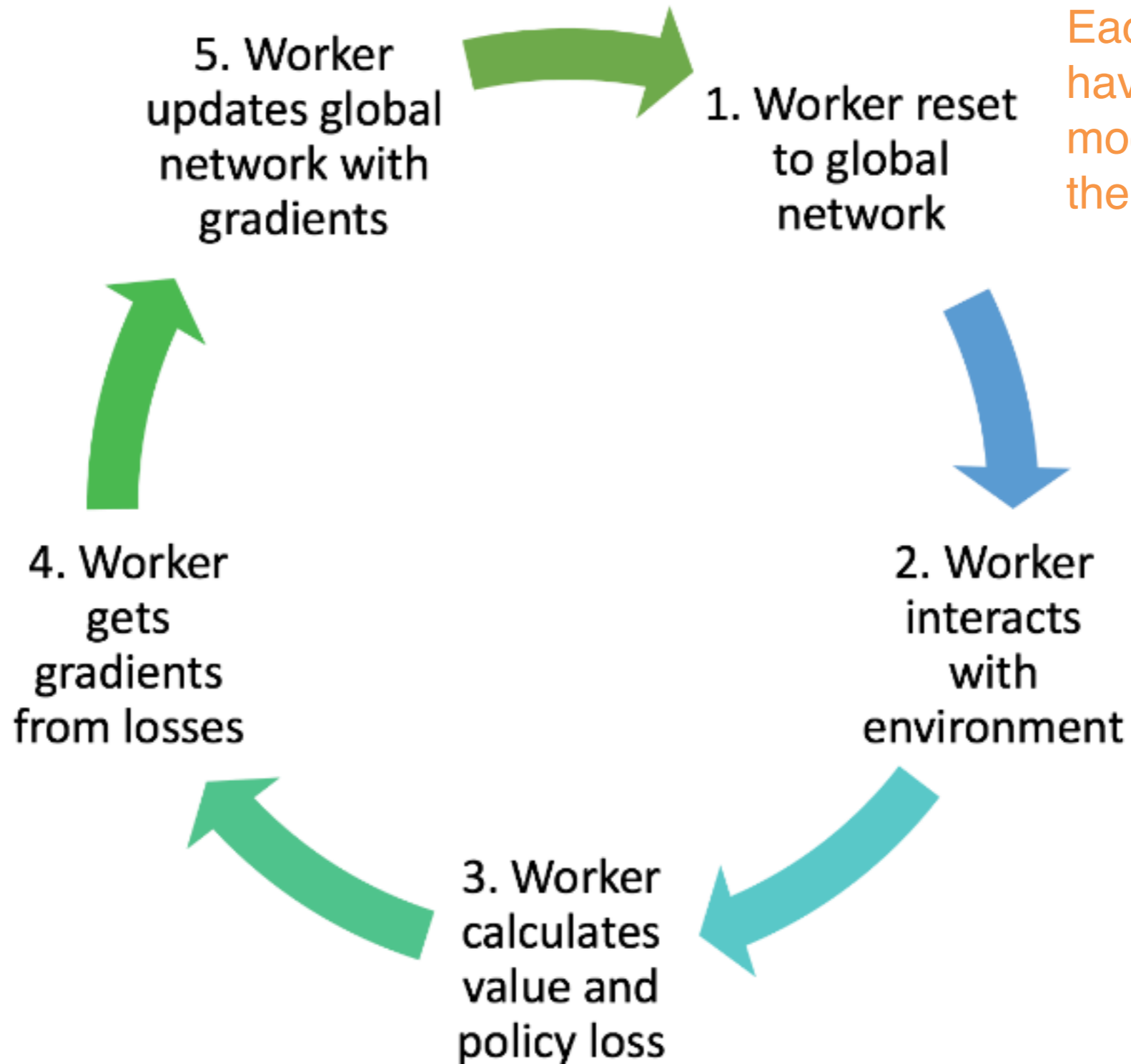
$$A_2 = R_2 - V(s_2; \theta'_v)$$

$$R_3 - V(s_3)$$



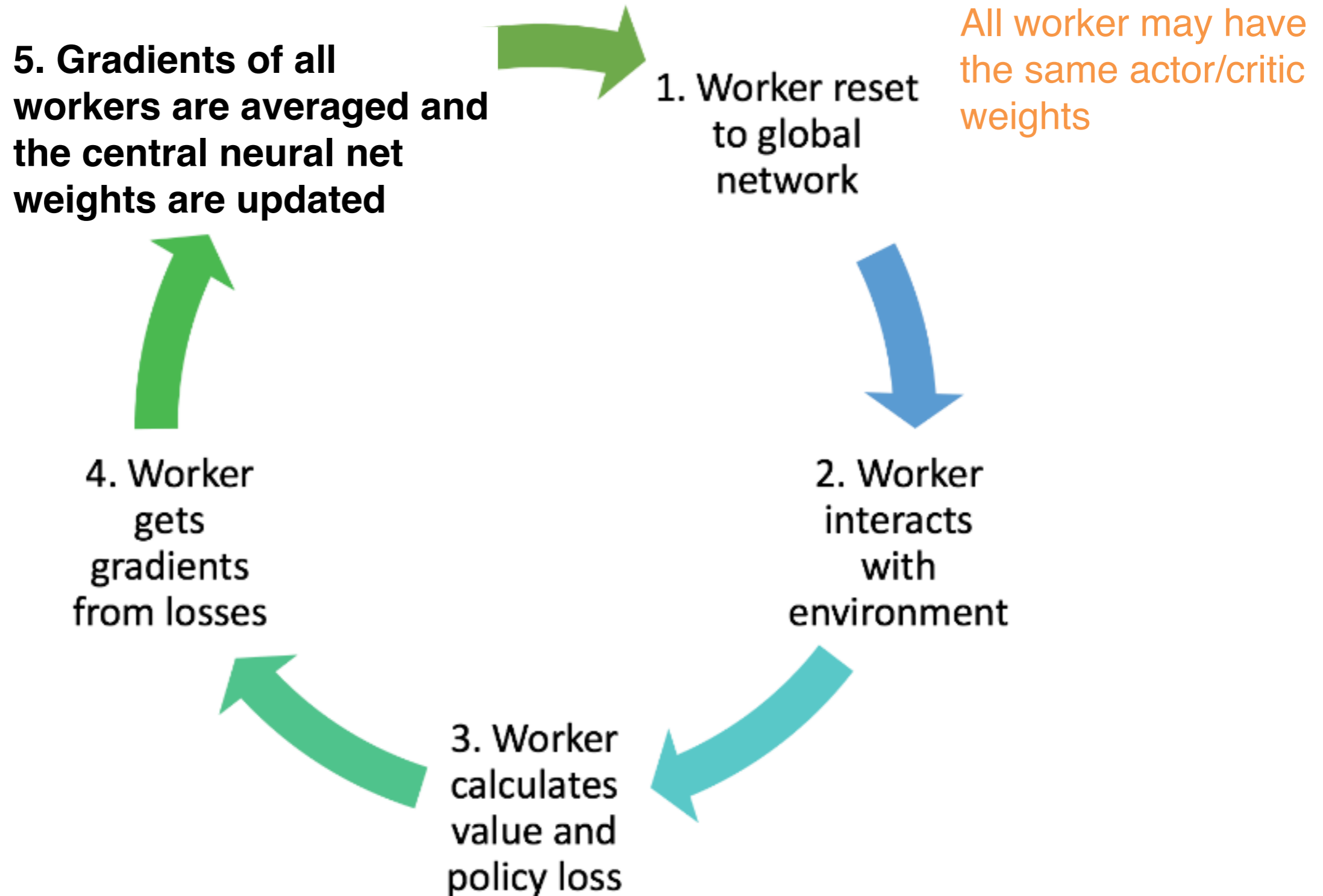
# Distributed Asynchronous RL-A3C

No locking



Each worker may have slightly modified version of the policy/critic

# Distributed Synchronous RL-A2C



---

**Algorithm S3** Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

---

*// Assume global shared parameter vectors  $\theta$  and  $\theta_v$  and global shared counter  $T = 0$*

*// Assume thread-specific parameter vectors  $\theta'$  and  $\theta'_v$*

Initialize thread step counter  $t \leftarrow 1$

**repeat**

*Reset gradients:  $d\theta \leftarrow 0$  and  $d\theta_v \leftarrow 0$ .*

*Synchronize thread-specific parameters  $\theta' \leftarrow \theta$  and  $\theta'_v \leftarrow \theta_v$*

*“We also found that adding the **entropy** of the policy  $\pi$  to the objective function improved exploration by discouraging premature convergence to suboptimal deterministic policies.” So you need to add to the policy gradient:  $+\beta \nabla_{\theta} \mathbf{H}(\pi_{\theta}(a_t | s_t; \theta))$*

*We will look into the entropy as part of the reward in later lecture*

*Perform asynchronous update of  $\theta$  using  $d\theta$  and of  $\theta_v$  using  $d\theta_v$ .*

**until**  $T > T_{max}$

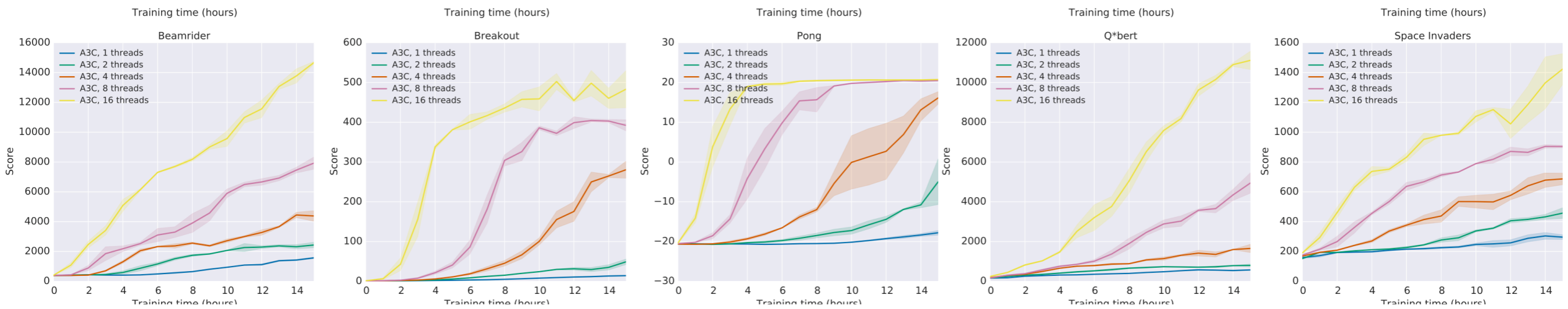
---

What is the approximation used for the advantage?

$$R_3 = r_3 + \gamma V(s_4, \theta'_v) \qquad A_3 = R_3 - V(s_3; \theta'_v)$$

$$R_2 = r_2 + \gamma r_3 + \gamma^2 V(s_4, \theta'_v) \qquad A_2 = R_2 - V(s_2; \theta'_v)$$

# Advantages of Asynchronous (multi-threaded) RL



# Summary of policy gradients so far

- ▶ The policy gradient has many equivalent forms

$$\begin{aligned}\nabla_{\theta} U(\theta) &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) \mathbf{v}_t] && \text{REINFORCE} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) Q^w(s, a)] && \text{Q Actor-Critic} \\ &= \mathbb{E}_{\pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(s, a) A^w(s, a)] && \text{Advantage Actor-Critic}\end{aligned}$$

- ▶ Each leads a stochastic gradient ascent algorithm
- ▶ Critic uses [policy evaluation](#) (e.g. MC or TD learning) to estimate

$$Q^{\pi}(s, a), A^{\pi}(s, a) \text{ or } V^{\pi}(s)$$

# Computing Gradients of Expectations

Likelihood ratio gradient estimator:

$$\max_{\theta} \cdot \mathbb{E}_{x \sim P_{\theta}(x)} f(x)$$

$$\max_{\theta} \cdot \mathbb{E}_{\tau \sim P_{\theta}(\tau)} [R(\tau)]$$

$$\mathbb{E}_{x \sim P_{\theta}(x)} \nabla_{\theta} \log P_{\theta}(x) f(x)$$

$$\mathbb{E}_{\tau \sim P_{\theta}(\tau)} [\nabla_{\theta} \log P_{\theta}(\tau) R(\tau)]$$

$$\mathbb{E}_{s \sim d^0(s), a \sim \pi_{\theta}(a|s)} \nabla_{\theta} \log \pi_{\theta}(a|s) [Q(s, a)]$$

$$\mathbb{E}_{s \sim d^0(s), a \sim \pi_{\theta}(a|s)} \nabla_{\theta} \log \pi_{\theta}(a|s) [Q(s, a) - V(s)]$$

Qs:

- Do the gradients of  $Q(s, a)$  or  $A(s, a) = Q(s, a) - V(s)$  w.r.t.  $\theta$  exist?
- Do we use them?

# What if we have a deterministic policy?

$$y = P_{\theta}(x)$$

$$\max_{\theta} . f(P_{\theta}(x))$$

$$a = \pi_{\theta}(s)$$

$$\max_{\theta} . \mathbb{E} \sum_t R(S_t, \pi_{\theta}(S_t))$$

$$\max_{\theta} . \mathbb{E} \sum_t Q(S_t, \pi_{\theta}(S_t))$$

Qs:

- Can we backpropagate through R?
- Can we backpropagate through Q?

$$\frac{df(P_{\theta}(x))}{d\theta} = \frac{df(y)}{dy} \frac{dy}{d\theta}$$

$$\frac{d\mathbb{E} \sum_t Q(S_t, \pi_{\theta}(S_t))}{d\theta} = \mathbb{E} \sum_t \frac{dQ(S_t, \pi_{\theta}(S_t))}{da} \frac{da}{d\theta}$$

# What if we have a deterministic policy?

## Pathwise derivatives

$$y = P_{\theta}(x)$$

$$\max_{\theta} . f(P_{\theta}(x))$$

$$\frac{df(P_{\theta}(x))}{d\theta} = \frac{df(y)}{dy} \frac{dy}{d\theta}$$

Q: does this expectation depend on theta?

$$\max_{\theta} . \mathbb{E} \sum_t R(S_t, \pi_{\theta}(S_t))$$

$$\max_{\theta} . \mathbb{E} \sum_t Q(S_t, \pi_{\theta}(S_t))$$

$$\frac{d\mathbb{E} \sum_t Q(S_t, \pi_{\theta}(S_t))}{d\theta} = \mathbb{E} \sum_t \frac{dQ(S_t, \pi_{\theta}(S_t))}{da} \frac{da}{d\theta}$$

## Likelihood ratio gradient estimator

$$\max_{\theta} . \mathbb{E}_{x \sim P_{\theta}(x)} f(x)$$

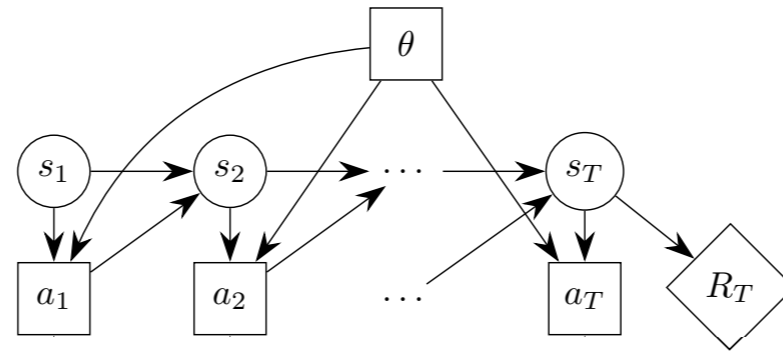
$$\mathbb{E}_{x \sim P_{\theta}(x)} \nabla_{\theta} \log P_{\theta}(x) f(x)$$

$$\max_{\theta} . \mathbb{E}_{\tau \sim P_{\theta}(\tau)} [R(\tau)]$$

$$\mathbb{E}_{\tau \sim P_{\theta}(\tau)} [\nabla_{\theta} \log P_{\theta}(\tau) R(\tau)]$$

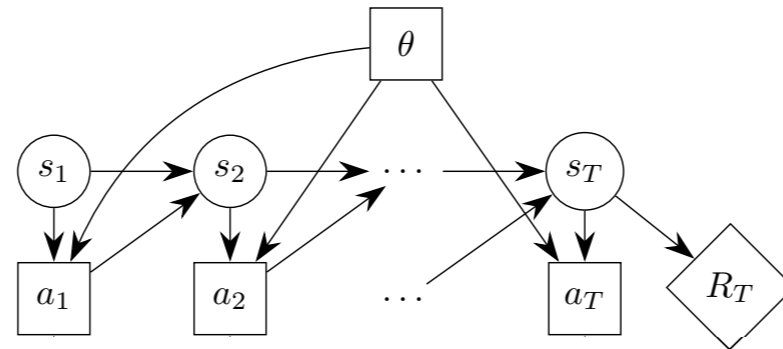


# Deep Deterministic Policy Gradients



$$\frac{d}{d\theta} \mathbb{E} [R_T] = \mathbb{E} \left[ \sum_{t=1}^T \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right]$$

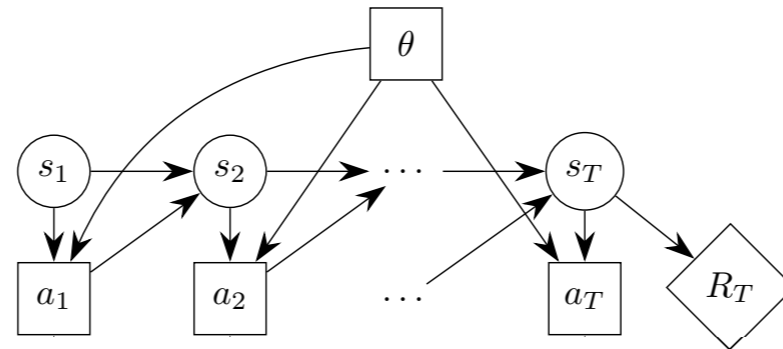
# Deep Deterministic Policy Gradients



This expectation refers to the dynamics after time  $t$

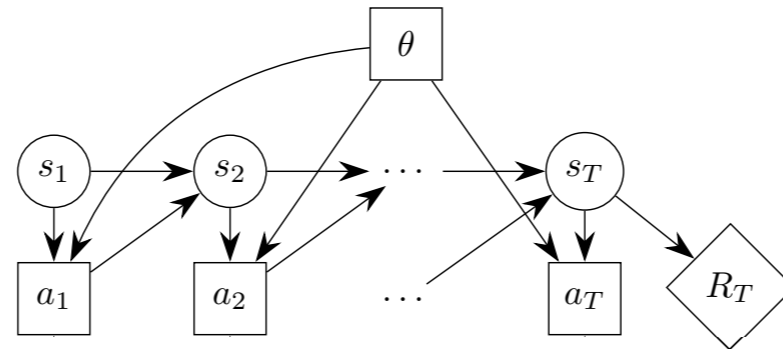
$$\frac{d}{d\theta} \mathbb{E} [R_T] = \mathbb{E} \left[ \sum_{t=1}^T \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^T \frac{d}{da_t} \mathbb{E} [R_T | a_t] \frac{da_t}{d\theta} \right]$$

# Deep Deterministic Policy Gradients



$$\begin{aligned} \frac{d}{d\theta} \mathbb{E} [R_T] &= \mathbb{E} \left[ \sum_{t=1}^T \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^T \frac{d}{da_t} \mathbb{E} [R_T | a_t] \frac{da_t}{d\theta} \right] \\ &= \mathbb{E} \left[ \sum_{t=1}^T \frac{dQ(s_t, a_t)}{da_t} \frac{da_t}{d\theta} \right] \end{aligned}$$

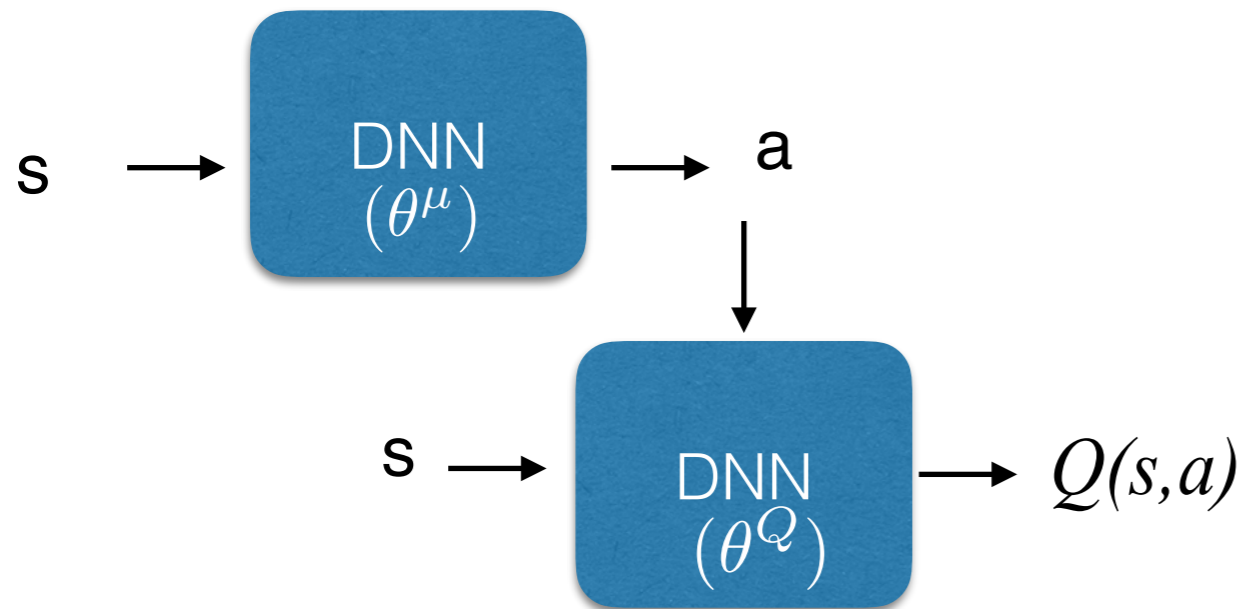
# Deep Deterministic Policy Gradients



$$\begin{aligned} \frac{d}{d\theta} \mathbb{E} [R_T] &= \mathbb{E} \left[ \sum_{t=1}^T \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^T \frac{d}{da_t} \mathbb{E} [R_T | a_t] \frac{da_t}{d\theta} \right] \\ &= \mathbb{E} \left[ \sum_{t=1}^T \frac{dQ(s_t, a_t)}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^T \frac{d}{d\theta} Q(s_t, \pi(s_t; \theta)) \right] \end{aligned}$$

# Deep Deterministic Policy Gradients

$$a = \mu(\theta)$$



We are following a stochastic behavior policy to collect data.  
Deep Q learning for continuous actions  $\rightarrow$  DDPG

# Deep Deterministic Policy Gradients

---

## Algorithm 1 DDPG algorithm

---

Randomly initialize critic network  $Q(s, a|\theta^Q)$  and actor  $\mu(s|\theta^\mu)$  with weights  $\theta^Q$  and  $\theta^\mu$ .

Initialize target network  $Q'$  and  $\mu'$  with weights  $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer  $R$

**for** episode = 1,  $M$  **do**

    Initialize a random process  $\mathcal{N}$  for action exploration

    Receive initial observation state  $s_1$

**for**  $t = 1, T$  **do**

        Select action  $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$  according to the current policy and exploration noise

        Execute action  $a_t$  and observe reward  $r_t$  and observe new state  $s_{t+1}$

        Store transition  $(s_t, a_t, r_t, s_{t+1})$  in  $R$

        Sample a random minibatch of  $N$  transitions  $(s_i, a_i, r_i, s_{i+1})$  from  $R$

        Set  $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

        Update critic by minimizing the loss:  $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

        Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

        Update the target networks:

$$\begin{aligned} \theta^{Q'} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q'} \\ \theta^{\mu'} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'} \end{aligned}$$

**end for**  
**end for**

---

# Deep Deterministic Policy Gradients

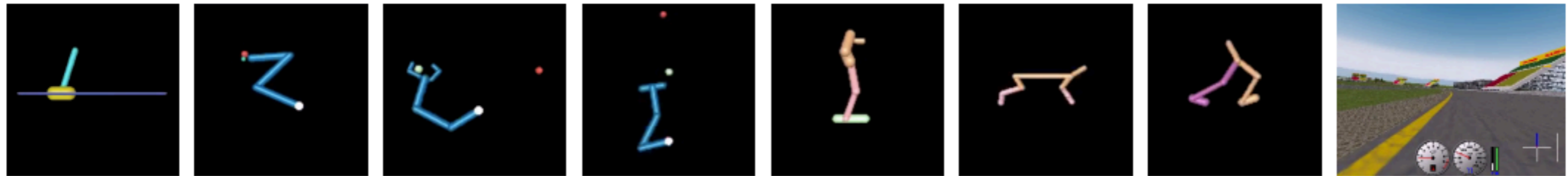


Figure 1: Example screenshots of a sample of environments we attempt to solve with DDPG. In order from the left: the cartpole swing-up task, a reaching task, a gasp and move task, a puck-hitting task, a monopod balancing task, two locomotion tasks and Torcs (driving simulator). We tackle all tasks using both low-dimensional feature vector and high-dimensional pixel inputs. Detailed descriptions of the environments are provided in the supplementary. Movies of some of the learned policies are available at <https://goo.gl/J4PIAz>.

<https://www.youtube.com/watch?v=tJBIqkC1wWM&feature=youtu.be>

# Deep Deterministic Policy Gradients

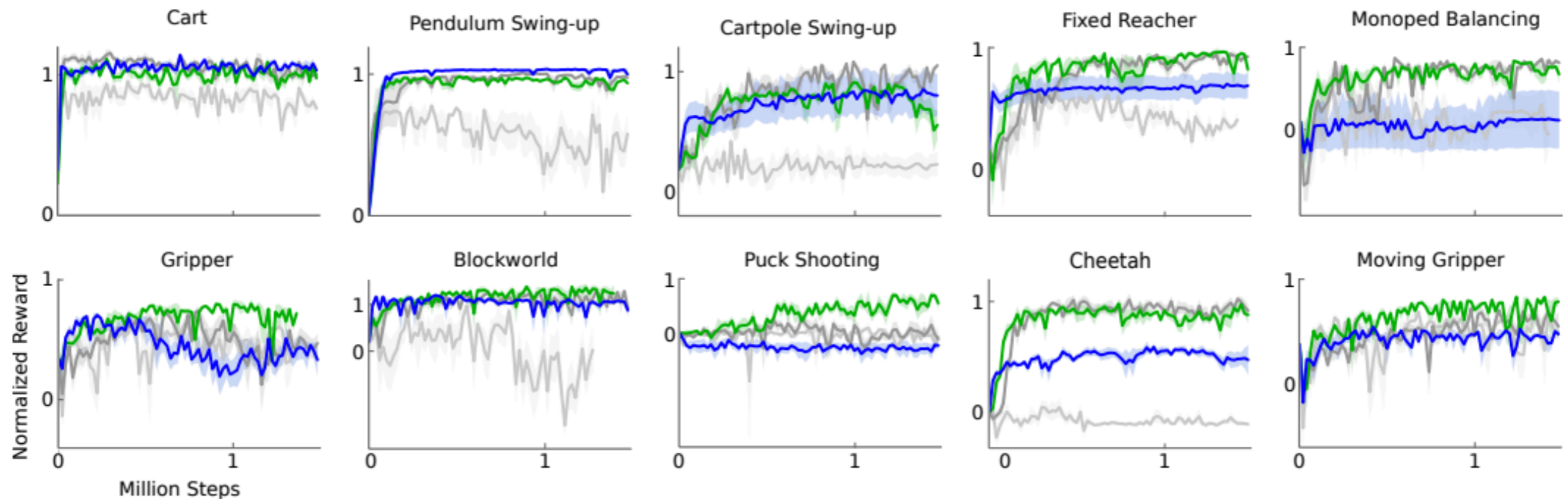


Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target networks are crucial.

State representation input can be pixels or robotic configuration and target locations

<https://www.youtube.com/watch?v=tJBIqkC1wWM&feature=youtu.be>



# Model Free Methods - Comparison

Task	Random	REINFORCE	TNPG	RWR	REPS	TRPO	CEM	CMA-ES	DDPG
Cart-Pole Balancing	77.1 ± 0.0	4693.7 ± 14.0	<b>3986.4 ± 748.9</b>	<b>4861.5 ± 12.3</b>	565.6 ± 137.6	<b>4869.8 ± 37.6</b>	4815.4 ± 4.8	2440.4 ± 568.3	4634.4 ± 87.8
Inverted Pendulum*	-153.4 ± 0.2	13.4 ± 18.0	<b>209.7 ± 55.5</b>	84.7 ± 13.8	-113.3 ± 4.6	<b>247.2 ± 76.1</b>	38.2 ± 25.7	-40.1 ± 5.7	40.0 ± 244.6
Mountain Car	-415.4 ± 0.0	-67.1 ± 1.0	<b>-66.5 ± 4.5</b>	-79.4 ± 1.1	-275.6 ± 166.3	<b>-61.7 ± 0.9</b>	-66.0 ± 2.4	-85.0 ± 7.7	-288.4 ± 170.3
Acrobot	-1904.5 ± 1.0	-508.1 ± 91.0	-395.8 ± 121.2	-352.7 ± 35.9	-1001.5 ± 10.8	-326.0 ± 24.4	-436.8 ± 14.7	-785.6 ± 13.1	<b>-223.6 ± 5.8</b>
Double Inverted Pendulum*	149.7 ± 0.1	4116.5 ± 65.2	<b>4455.4 ± 37.6</b>	3614.8 ± 368.1	446.7 ± 114.8	<b>4412.4 ± 50.4</b>	2566.2 ± 178.9	1576.1 ± 51.3	2863.4 ± 154.0
Swimmer*	-1.7 ± 0.1	92.3 ± 0.1	<b>96.0 ± 0.2</b>	60.7 ± 5.5	3.8 ± 3.3	<b>96.0 ± 0.2</b>	68.8 ± 2.4	64.9 ± 1.4	85.8 ± 1.8
Hopper	8.4 ± 0.0	714.0 ± 29.3	<b>1155.1 ± 57.9</b>	553.2 ± 71.0	86.7 ± 17.6	<b>1183.3 ± 150.0</b>	63.1 ± 7.8	20.3 ± 14.3	267.1 ± 43.5
2D Walker	-1.7 ± 0.0	506.5 ± 78.8	<b>1382.6 ± 108.2</b>	136.0 ± 15.9	-37.0 ± 38.1	<b>1353.8 ± 85.0</b>	84.5 ± 19.2	77.1 ± 24.3	318.4 ± 181.6
Half-Cheetah	-90.8 ± 0.3	1183.1 ± 69.2	<b>1729.5 ± 184.6</b>	376.1 ± 28.2	34.5 ± 38.0	<b>1914.0 ± 120.1</b>	330.4 ± 274.8	441.3 ± 107.6	<b>2148.6 ± 702.7</b>
Ant*	13.4 ± 0.7	548.3 ± 55.5	<b>706.0 ± 127.7</b>	37.6 ± 3.1	39.0 ± 9.8	<b>730.2 ± 61.3</b>	49.2 ± 5.9	17.8 ± 15.5	326.2 ± 20.8
Simple Humanoid	41.5 ± 0.2	128.1 ± 34.0	<b>255.0 ± 24.5</b>	93.3 ± 17.4	28.3 ± 4.7	<b>269.7 ± 40.3</b>	60.6 ± 12.9	28.7 ± 3.9	99.4 ± 28.1
Full Humanoid	13.2 ± 0.1	262.2 ± 10.5	<b>288.4 ± 25.2</b>	46.7 ± 5.6	41.7 ± 6.1	<b>287.0 ± 23.4</b>	36.9 ± 2.9	N/A ± N/A	119.0 ± 31.2
Cart-Pole Balancing (LS)*	77.1 ± 0.0	420.9 ± 265.5	<b>945.1 ± 27.8</b>	68.9 ± 1.5	898.1 ± 22.1	<b>960.2 ± 46.0</b>	227.0 ± 223.0	68.0 ± 1.6	
Inverted Pendulum (LS)	-122.1 ± 0.1	-13.4 ± 3.2	<b>0.7 ± 6.1</b>	-107.4 ± 0.2	-87.2 ± 8.0	<b>4.5 ± 4.1</b>	-81.2 ± 33.2	-62.4 ± 3.4	
Mountain Car (LS)	-83.0 ± 0.0	-81.2 ± 0.6	<b>-65.7 ± 9.0</b>	-81.7 ± 0.1	-82.6 ± 0.4	<b>-64.2 ± 9.5</b>	<b>-68.9 ± 1.3</b>	<b>-73.2 ± 0.6</b>	
Acrobot (LS)*	-393.2 ± 0.0	-128.9 ± 11.6	<b>-84.6 ± 2.9</b>	-235.9 ± 5.3	-379.5 ± 1.4	<b>-83.3 ± 9.9</b>	-149.5 ± 15.3	-159.9 ± 7.5	
Cart-Pole Balancing (NO)*	101.4 ± 0.1	616.0 ± 210.8	<b>916.3 ± 23.0</b>	93.8 ± 1.2	99.6 ± 7.2	606.2 ± 122.2	181.4 ± 32.1	104.4 ± 16.0	
Inverted Pendulum (NO)	-122.2 ± 0.1	6.5 ± 1.1	<b>11.5 ± 0.5</b>	-110.0 ± 1.4	-119.3 ± 4.2	<b>10.4 ± 2.2</b>	-55.6 ± 16.7	-80.3 ± 2.8	
Mountain Car (NO)	-83.0 ± 0.0	-74.7 ± 7.8	<b>-64.5 ± 8.6</b>	-81.7 ± 0.1	-82.9 ± 0.1	<b>-60.2 ± 2.0</b>	-67.4 ± 1.4	-73.5 ± 0.5	
Acrobot (NO)*	-393.5 ± 0.0	<b>-186.7 ± 31.3</b>	<b>-164.5 ± 13.4</b>	-233.1 ± 0.4	-258.5 ± 14.0	<b>-149.6 ± 8.6</b>	-213.4 ± 6.3	-236.6 ± 6.2	
Cart-Pole Balancing (SI)*	76.3 ± 0.1	431.7 ± 274.1	<b>980.5 ± 7.3</b>	69.0 ± 2.8	702.4 ± 196.4	<b>980.3 ± 5.1</b>	746.6 ± 93.2	71.6 ± 2.9	
Inverted Pendulum (SI)	-121.8 ± 0.2	-5.3 ± 5.6	<b>14.8 ± 1.7</b>	-108.7 ± 4.7	-92.8 ± 23.9	<b>14.1 ± 0.9</b>	-51.8 ± 10.6	-63.1 ± 4.8	
Mountain Car (SI)	-82.7 ± 0.0	-63.9 ± 0.2	<b>-61.8 ± 0.4</b>	-81.4 ± 0.1	-80.7 ± 2.3	<b>-61.6 ± 0.4</b>	-63.9 ± 1.0	-66.9 ± 0.6	
Acrobot (SI)*	-387.8 ± 1.0	<b>-169.1 ± 32.3</b>	<b>-156.6 ± 38.9</b>	-233.2 ± 2.6	-216.1 ± 7.7	<b>-170.9 ± 40.3</b>	-250.2 ± 13.7	-245.0 ± 5.5	
Swimmer + Gathering	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Ant + Gathering	-5.8 ± 5.0	-0.1 ± 0.1	-0.4 ± 0.1	-5.5 ± 0.5	-6.7 ± 0.7	-0.4 ± 0.0	-4.7 ± 0.7	N/A ± N/A	-0.3 ± 0.3
Swimmer + Maze	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Ant + Maze	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	N/A ± N/A	0.0 ± 0.0

# Computing Gradients of Expectations

When the variable w.r.t. which we are differentiating appears **in the distribution**:

$$\nabla_{\theta} \mathbb{E}_{x \sim P_{\theta}(x)} f(x) = \mathbb{E}_{x \sim P_{\theta}(x)} \nabla_{\theta} \log P_{\theta}(x) f(x)$$

likelihood ratio gradient estimator

When the variable w.r.t. which we are differentiating appears **inside the expectation**:

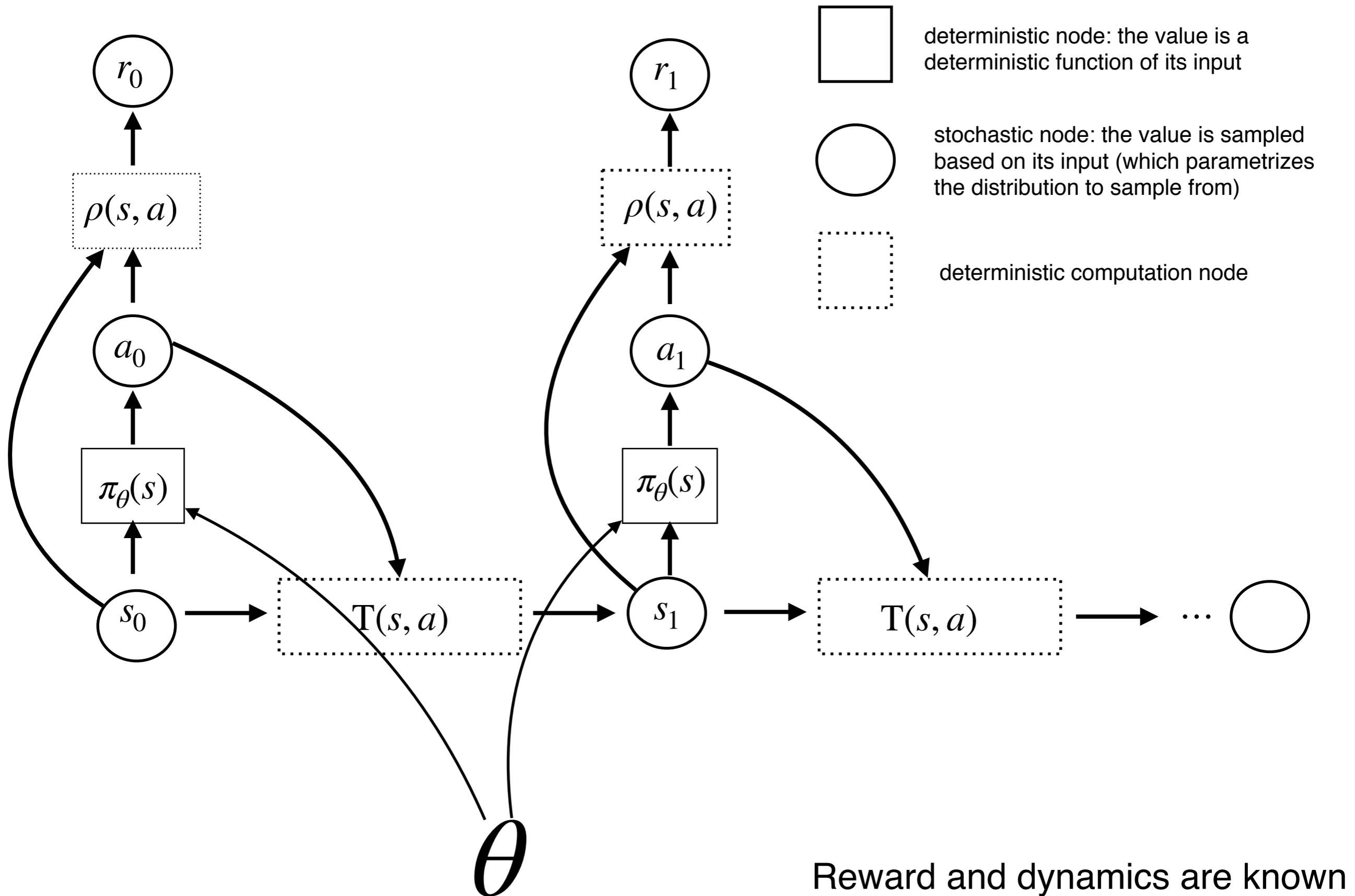
$$\nabla_{\theta} \mathbb{E}_{z \sim \mathcal{N}(0,1)} f(x(\theta), z) = \mathbb{E}_{z \sim \mathcal{N}(0,1)} \nabla_{\theta} f(x(\theta), z) = \mathbb{E}_{z \sim \mathcal{N}(0,1)} \frac{df(x(\theta), z)}{dx} \frac{dx}{d\theta}$$

pathwise derivative

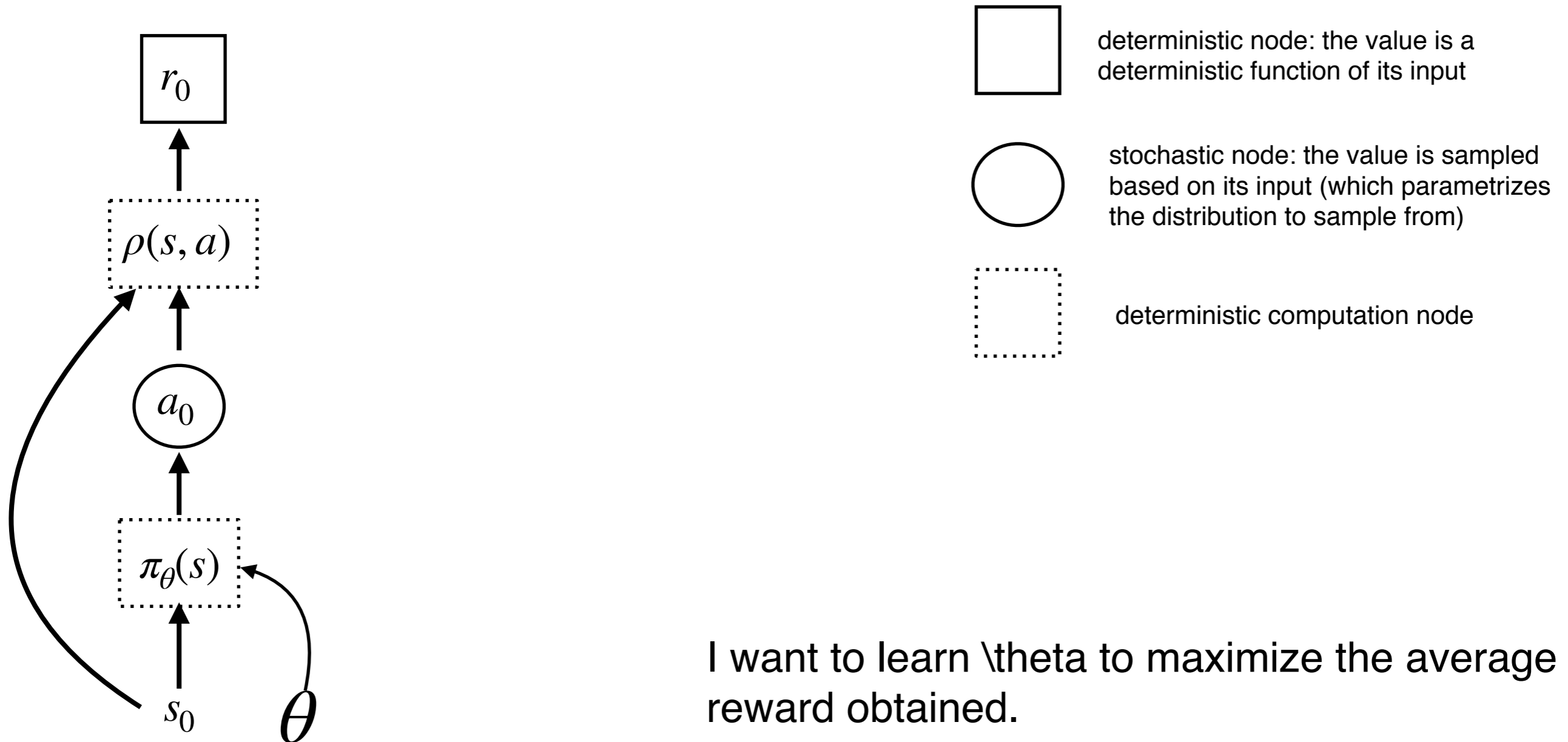
**Re-parametrization trick:** For some distributions  $P_{\theta}(x)$  we can switch from one gradient estimator to the other.

**Why would we want to do so?**

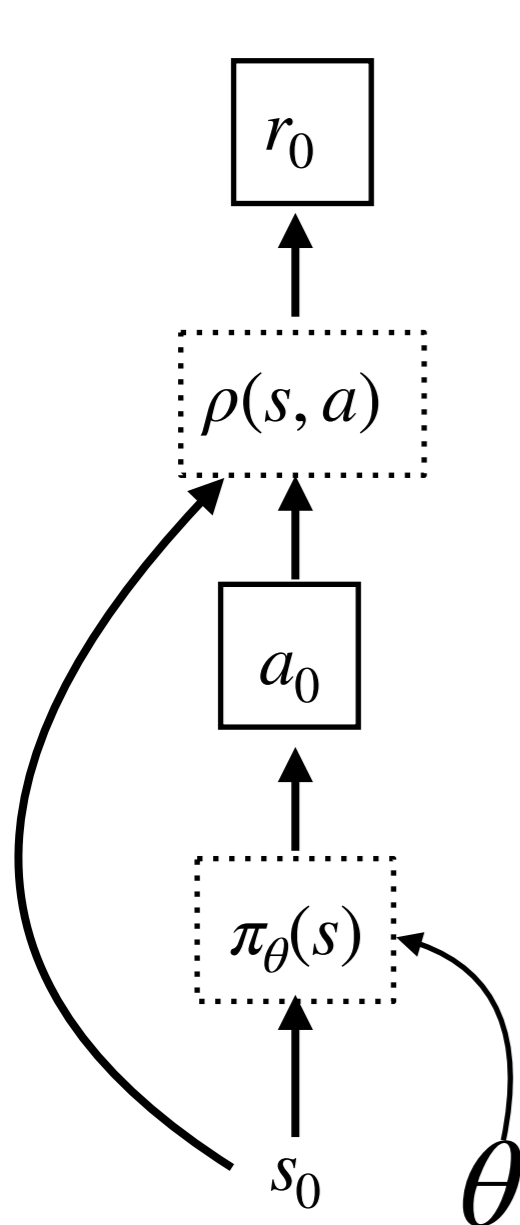
# Known MDP



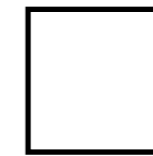
# Known MDP-let's make it simpler



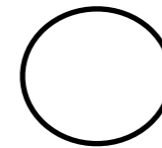
# What if the policy is deterministic?



$$a = \pi_\theta(s)$$



deterministic node: the value is a deterministic function of its input



stochastic node: the value is sampled based on its input (which parametrizes the distribution to sample from)



deterministic computation node

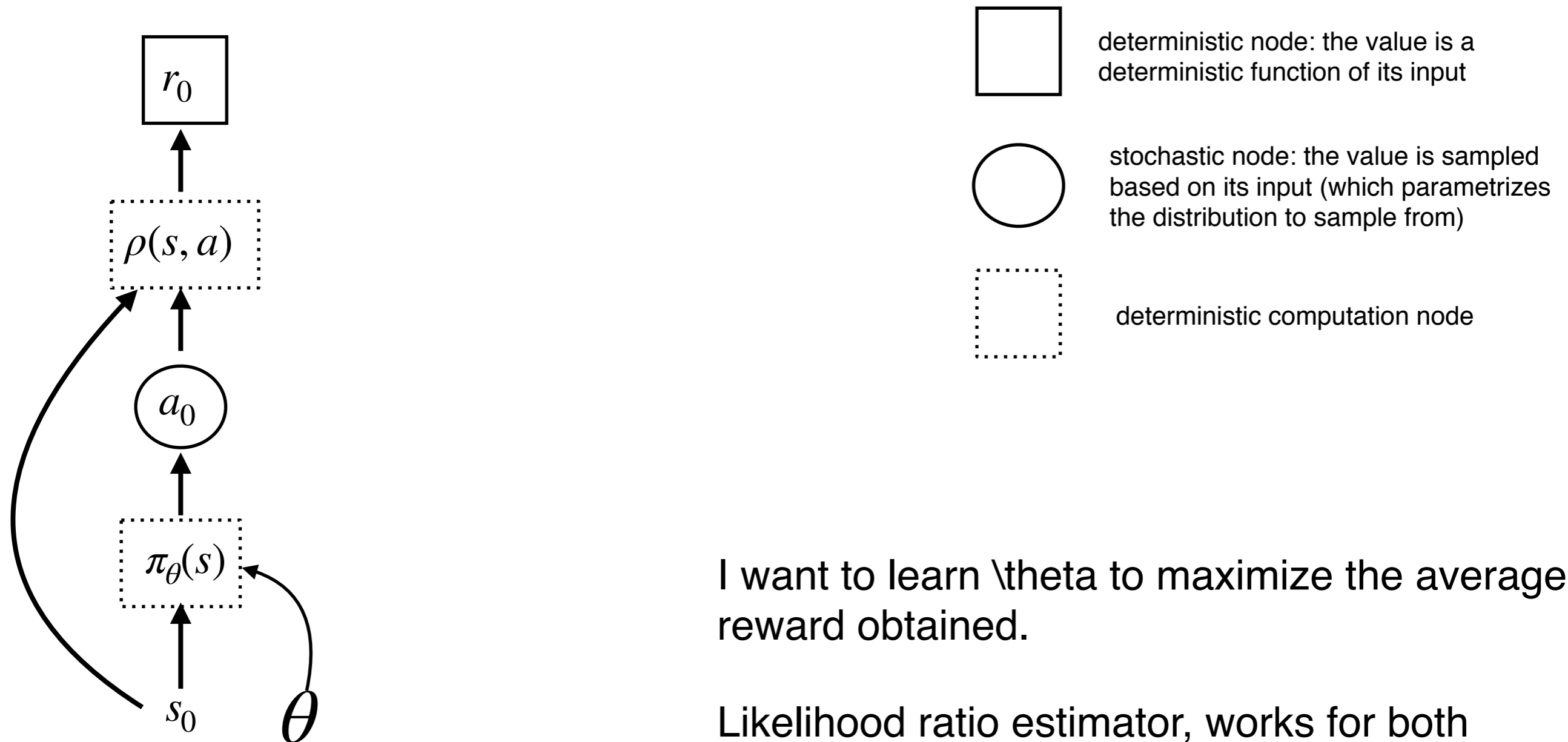
I want to learn  $\theta$  to maximize the reward obtained.

I can compute the gradient with backpropagation.

$$\nabla_\theta \rho(s, a) = \rho_a \pi_{\theta\theta}$$

Derivative of the known reward w.r.t. the action

# What if the policy is stochastic?



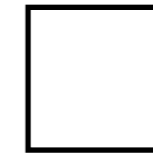
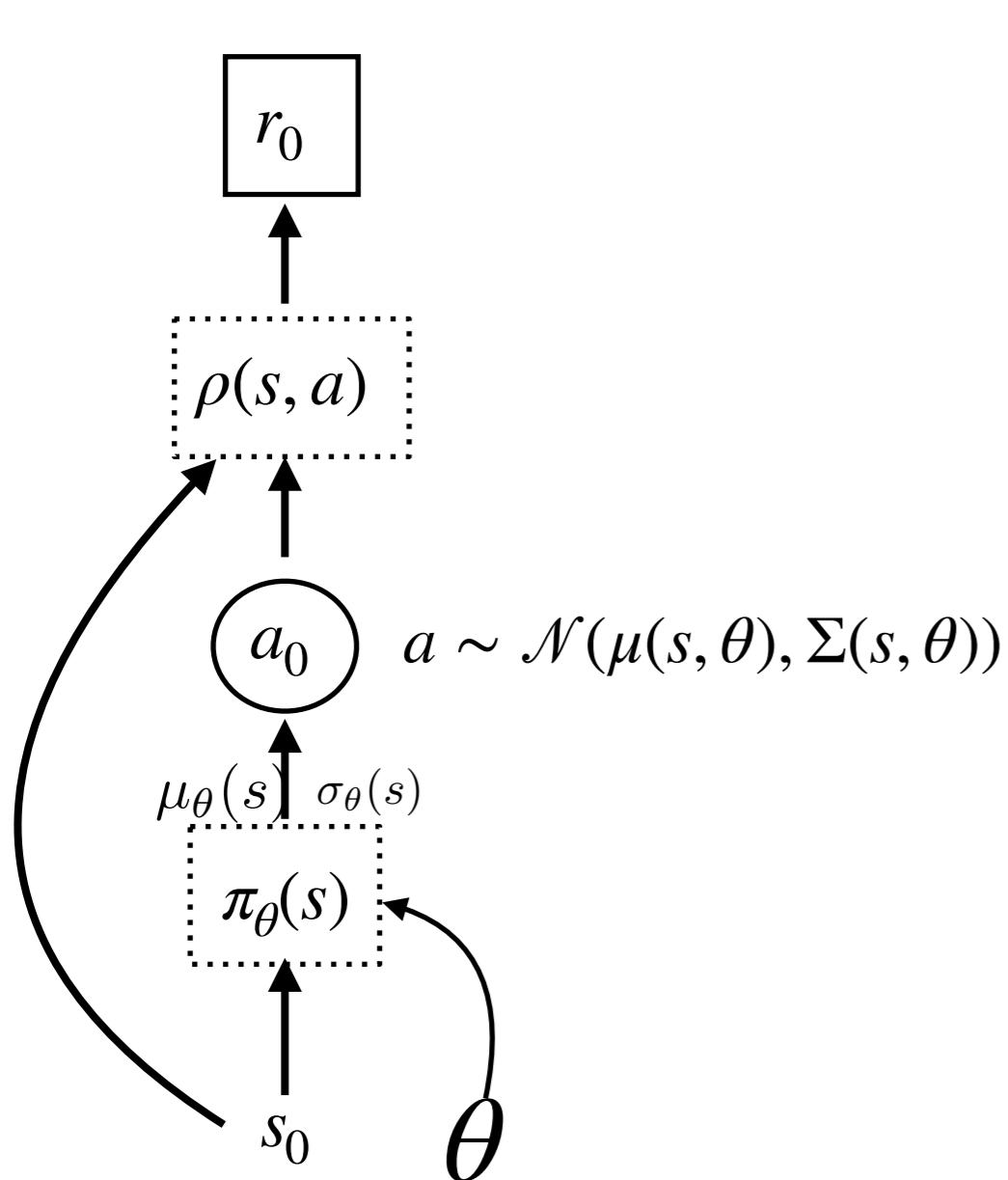
I want to learn  $\theta$  to maximize the average reward obtained.

Likelihood ratio estimator, works for both continuous and discrete actions

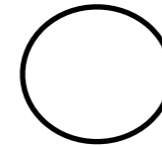
$$\mathbb{E}_a \nabla_\theta \log \pi_\theta(s) \rho(s, a)$$

It does not use the derivative of the reward w.r.t. the action.

# Policies are parametrized Gaussians



deterministic node: the value is a deterministic function of its input



stochastic node: the value is sampled based on its input (which parametrizes the distribution to sample from)



deterministic computation node

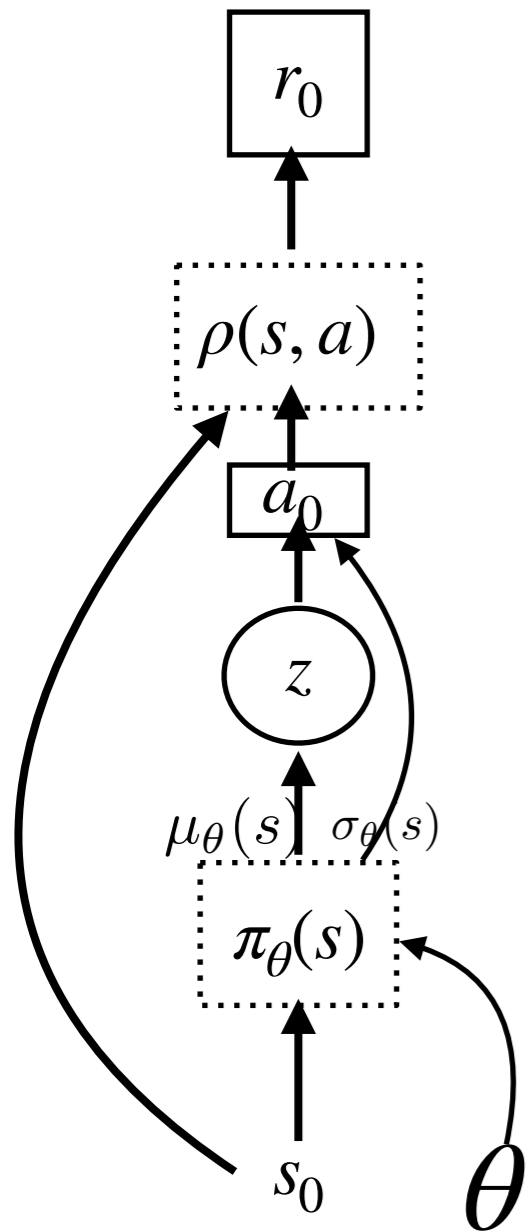
I want to learn  $\theta$  to maximize the average reward obtained.

$$\mathbb{E}_a \nabla_\theta \log \pi_\theta(s) \rho(s, a)$$

If  $\sigma^2$  is constant:

$$\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s; \theta)) \frac{\partial \mu(s; \theta)}{\partial \theta}}{\sigma^2}$$

# Re-parametrization for Gaussian



Instead of:  $a \sim \mathcal{N}(\mu(s, \theta), \Sigma(s, \theta))$

We can write:  $a = \mu(s, \theta) + z \odot \sigma(s, \theta) \quad z \sim \mathcal{N}(0, I)$

Why?

$$\mathbb{E}_z(\mu(s, \theta) + z\sigma(s, \theta)) = \mu(s, \theta)$$

Because:

$$\text{Var}_z(\mu(s, \theta) + z\sigma(s, \theta)) = \sigma(s, \theta)^2$$

What do we gain?

$$\nabla_{\theta} \mathbb{E}_z \left[ \rho(a(\theta, z), s) \right] = \mathbb{E}_z \frac{d\rho(a(\theta, z), s)}{da} \frac{da(\theta, z)}{d\theta}$$

$$\frac{da(\theta, z)}{d\theta} = \frac{d\mu(s, \theta)}{d\theta} + z \odot \frac{d\sigma(s, \theta)}{d\theta}$$

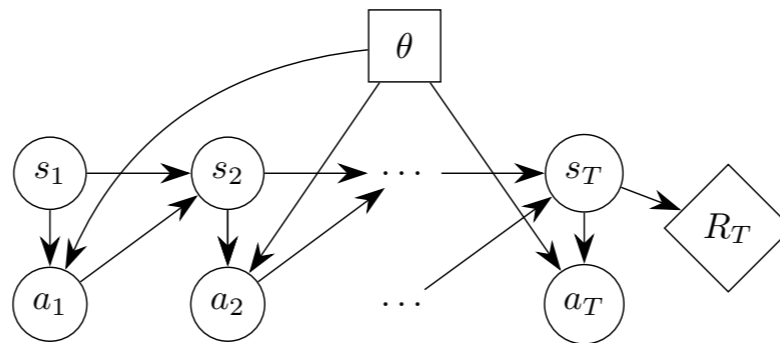
Sample estimate:

$$\nabla_{\theta} \frac{1}{N} \sum_{i=1}^N \left[ \rho(a(\theta, z_i), s) \right] = \frac{1}{N} \sum_{i=1}^N \frac{d\rho(a(\theta, z), s)}{da} \frac{da(\theta, z)}{d\theta} \Big|_{z=z_i}$$



# Re-parametrized Policy Gradients

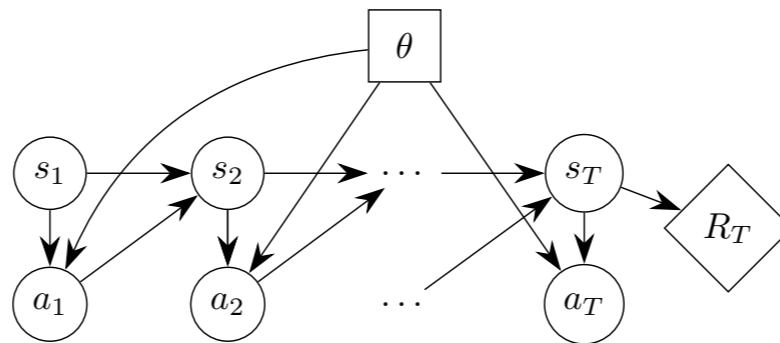
- ▶ Episodic MDP:



We want to compute:  $\nabla_{\theta} \mathbb{E}[R_T]$

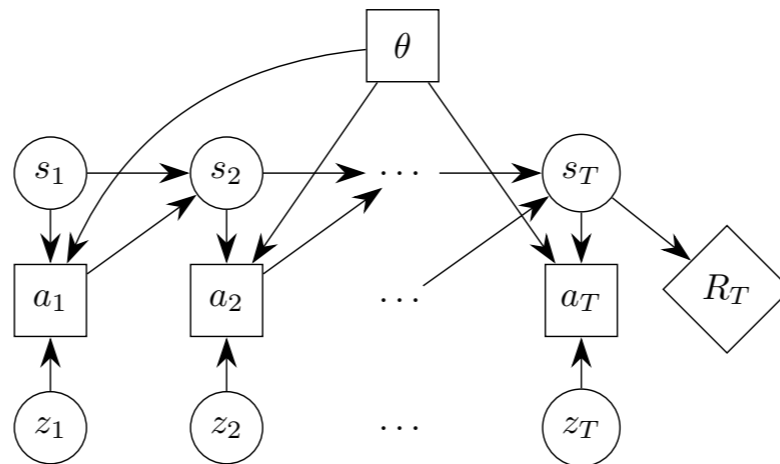
# Re-parametrized Policy Gradients

- ▶ Episodic MDP:



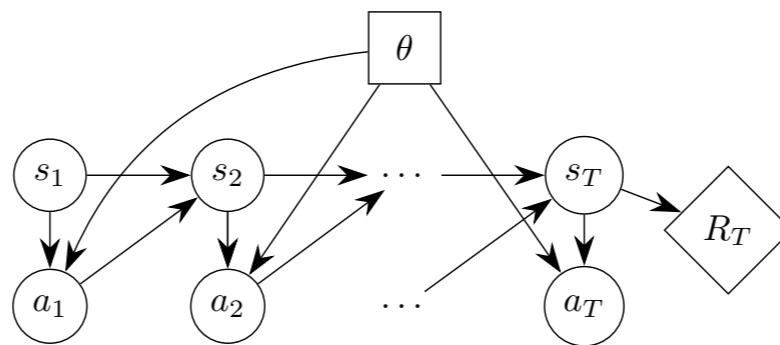
We want to compute:  $\nabla_{\theta} \mathbb{E}[R_T]$

- ▶ Reparameterize:  $a_t = \pi(s_t, z_t; \theta)$ .  $z_t$  is noise from fixed distribution.



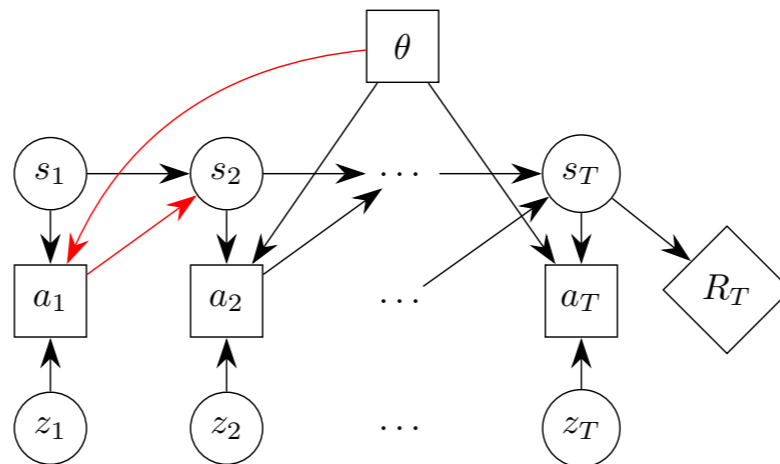
# Re-parametrized Policy Gradients

- ▶ Episodic MDP:



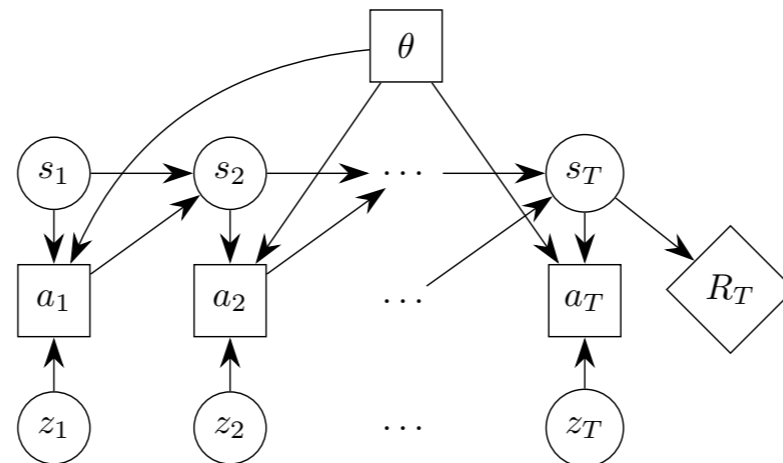
We want to compute:  $\nabla_{\theta} \mathbb{E}[R_T]$

- ▶ Reparameterize:  $a_t = \pi(s_t, z_t; \theta)$ .  $z_t$  is noise from fixed distribution.



For pathwise derivative to work, we need transition dynamics and reward function to be known.

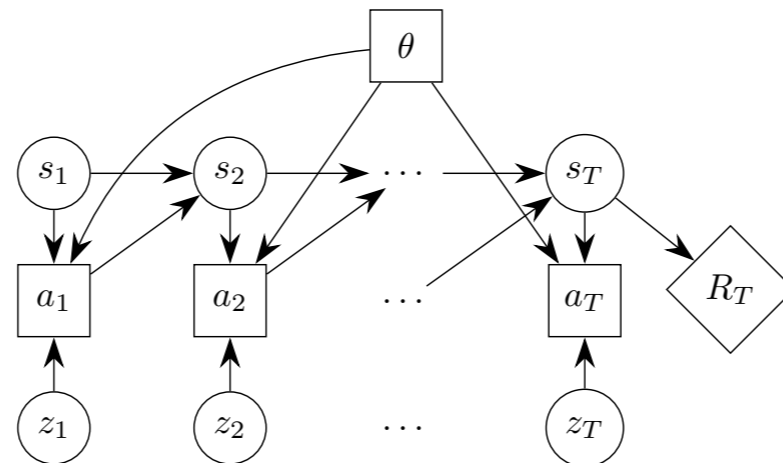
# Re-parametrized Policy Gradients



$$\frac{d}{d\theta} \mathbb{E} [R_T] = \mathbb{E} \left[ \sum_{t=1}^T \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^T \frac{d}{da_t} \mathbb{E} [R_T | a_t] \frac{da_t}{d\theta} \right]$$

For path wise derivative to work, we need transition dynamics and reward function to be known, or...

# Re-parametrized Policy Gradients



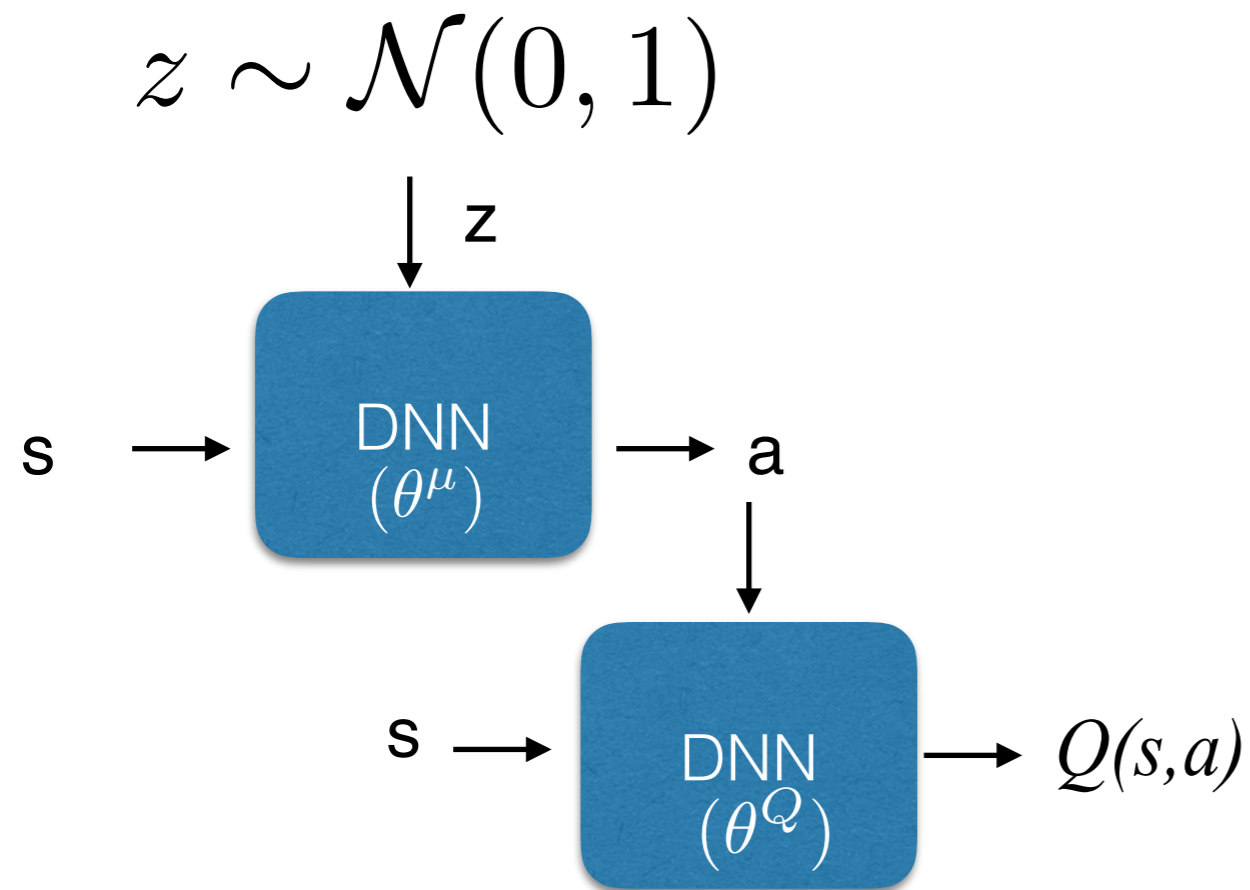
$$\begin{aligned} \frac{d}{d\theta} \mathbb{E} [R_T] &= \mathbb{E} \left[ \sum_{t=1}^T \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^T \frac{d}{da_t} \mathbb{E} [R_T | a_t] \frac{da_t}{d\theta} \right] \\ &= \mathbb{E} \left[ \sum_{t=1}^T \frac{dQ(s_t, a_t)}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^T \frac{d}{d\theta} Q(s_t, \pi(s_t, z_t; \theta)) \right] \end{aligned}$$

- ▶ Learn  $Q_\phi$  to approximate  $Q^{\pi, \gamma}$ , and use it to compute gradient estimates.

# Stochastic Value Gradients V0

- ▶ Learn  $Q_\phi$  to approximate  $Q^{\pi,\gamma}$ , and use it to compute gradient estimates.
- ▶ Pseudocode:
  - for** iteration=1, 2, ... **do**
  - Execute policy  $\pi_\theta$  to collect  $T$  timesteps of data
  - Update  $\pi_\theta$  using  $g \propto \nabla_\theta \sum_{t=1}^T Q(s_t, \pi(s_t, z_t; \theta))$
  - Update  $Q_\phi$  using  $g \propto \nabla_\phi \sum_{t=1}^T (Q_\phi(s_t, a_t) - \hat{Q}_t)^2$ , e.g. with TD( $\lambda$ )
  - end for**

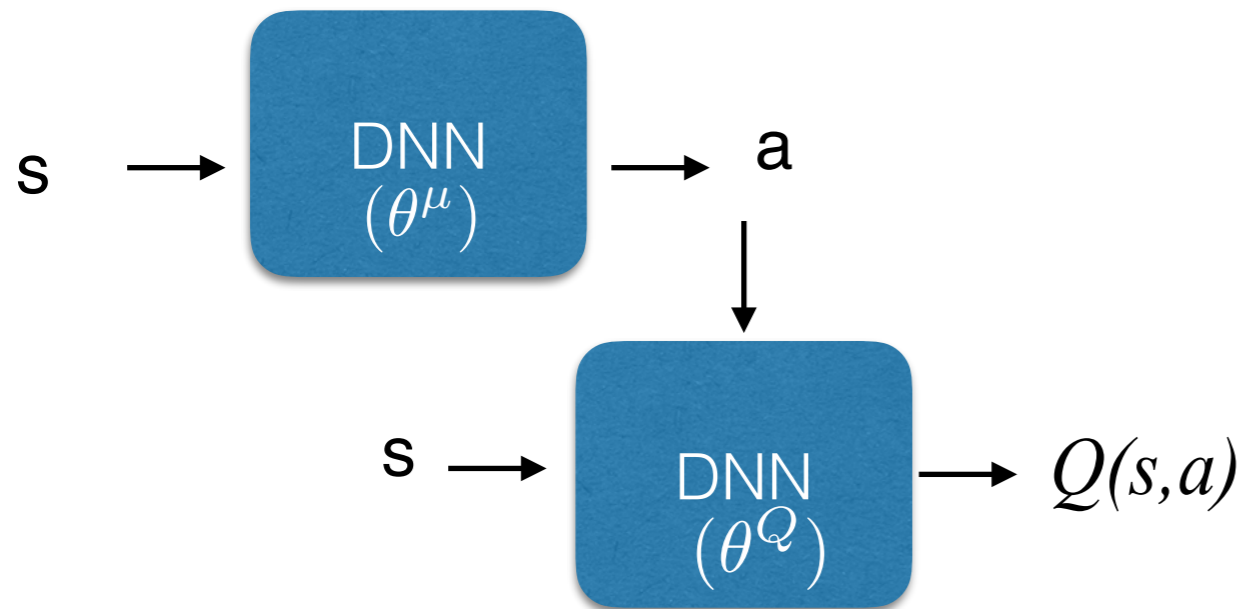
# Stochastic Value Gradients V0



$$a = \mu(s; \theta) + z\sigma(s; \theta)$$

# Compare with: Deep Deterministic Policy Gradients

$$a = \mu(\theta)$$



No z!