Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Pathwise derivatives, DDPG, Multigoal RL

Katerina Fragkiadaki

Part of the slides on path wise derivatives adapted from John Schulman

Computing Gradients of Expectations

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

$$\nabla_\theta \mathbb{E}_{x\sim p(\cdot|\theta)} \mathrm{F}(x) = \boxed{\mathbb{E}_{x\sim p(\cdot|\theta)} \nabla_\theta \log p(\,\cdot\,|\,\theta) \mathrm{F}(x)} \quad \text{e.g.} \quad \nabla_\theta \mathbb{E}_{a\sim\pi_\theta} R(a, s)$$

likelihood ratio gradient estimator

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

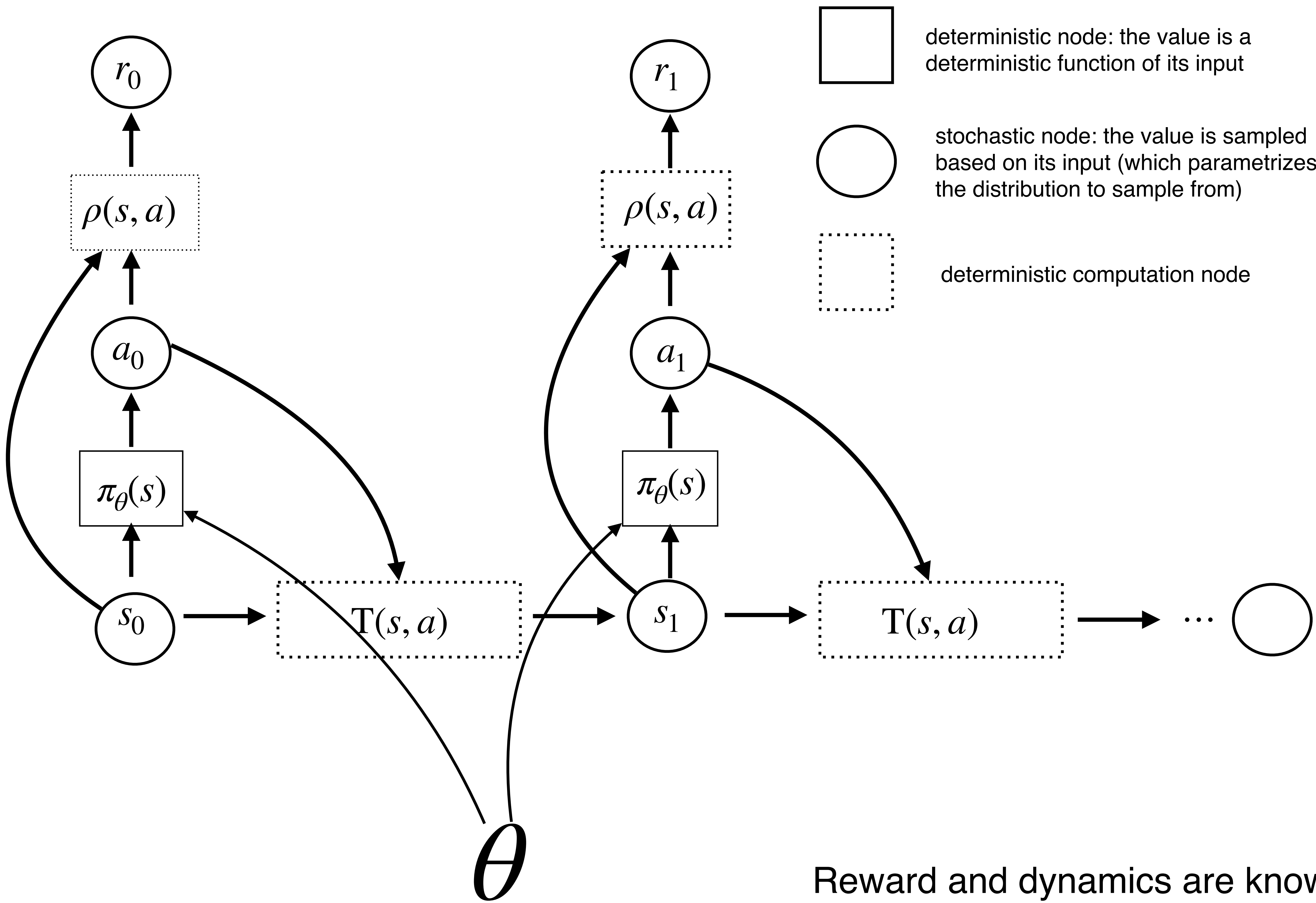
$$\nabla_\theta \mathbb{E}_{z\sim\mathcal{N}(0,1)} \mathrm{F}(x(\theta), z) = \mathbb{E}_{z\sim\mathcal{N}(0,1)} \nabla_\theta \mathrm{F}(x(\theta), z) = \boxed{\mathbb{E}_{z\sim\mathcal{N}(0,1)} \frac{d\mathrm{F}(x(\theta), z)}{dx} \frac{dx}{d\theta}}$$

pathwise derivative

Re-parametrization trick: For some distributions p(xl\theta) we can switch from one gradient estimator to the other.

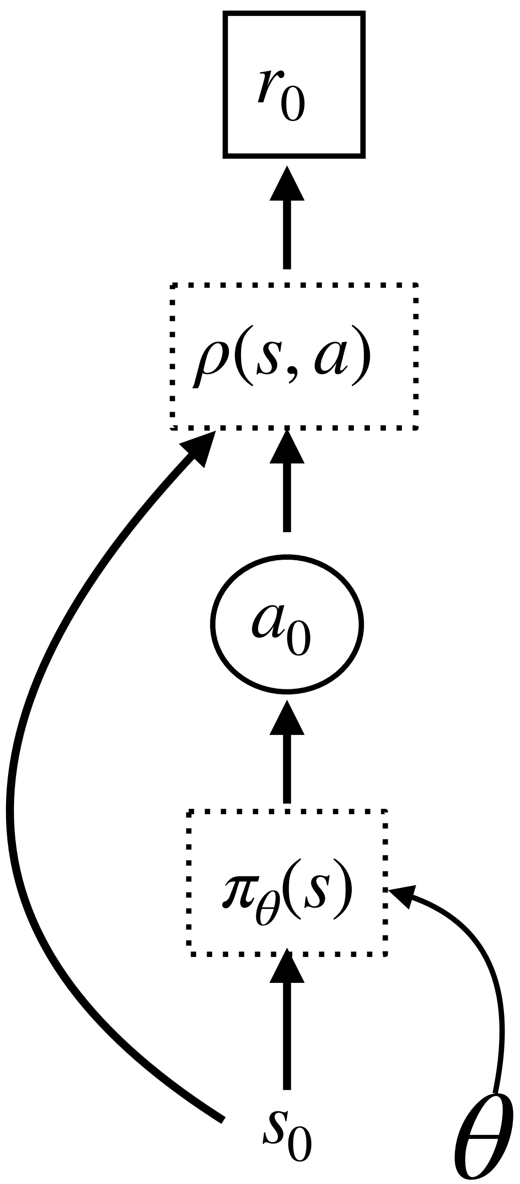
Why would we want to do so?

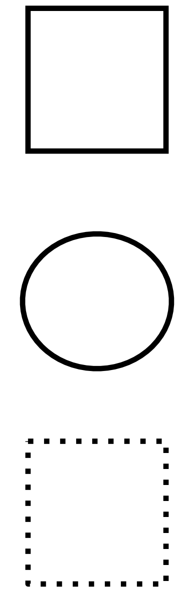
Known MDP



Reward and dynamics are known

Known MDP-let's make it simpler





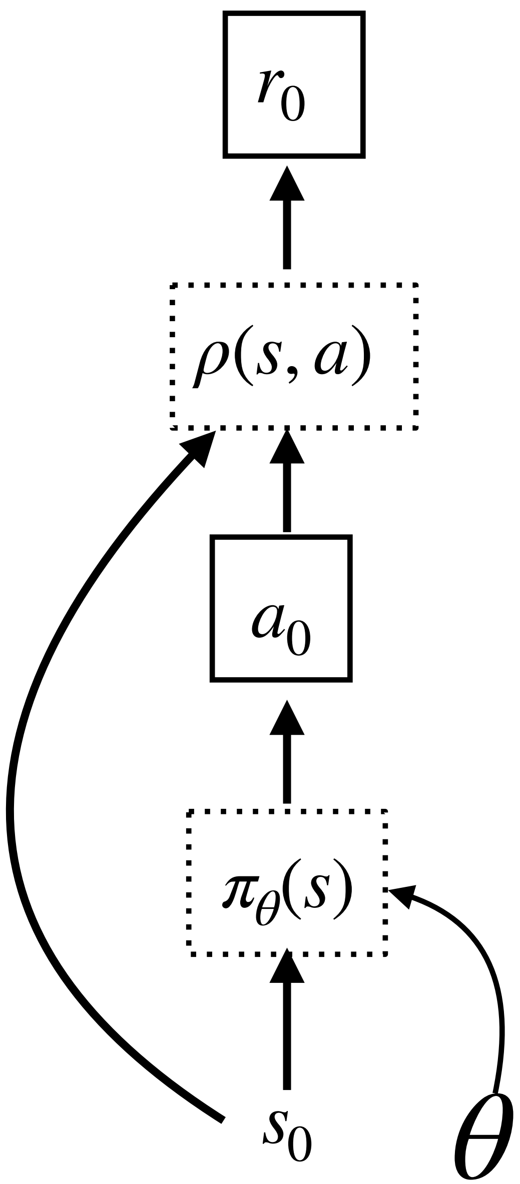
deterministic node: the value is a deterministic function of its input

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

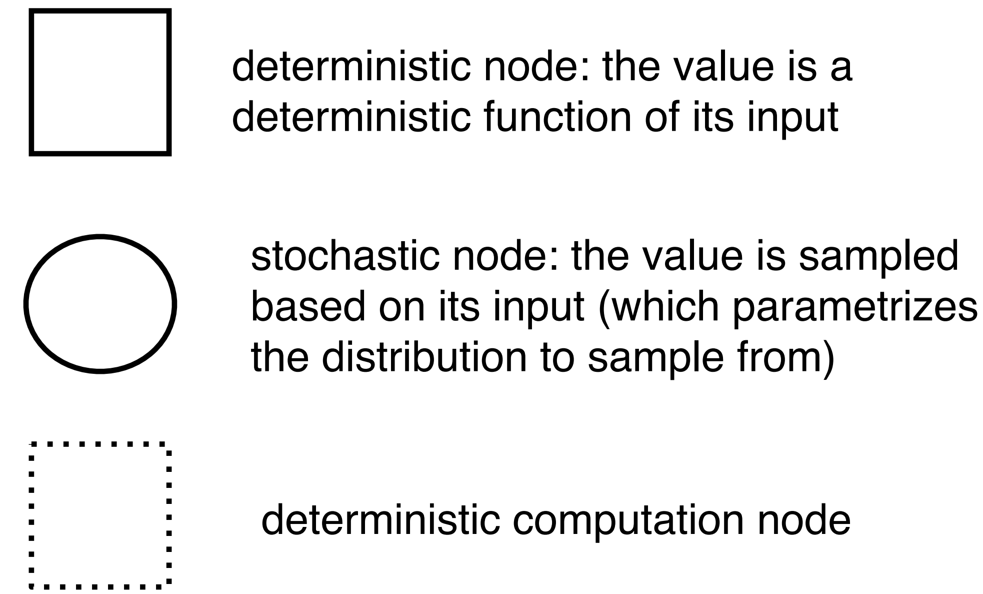
deterministic computation node

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

What if the policy is deterministic?



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

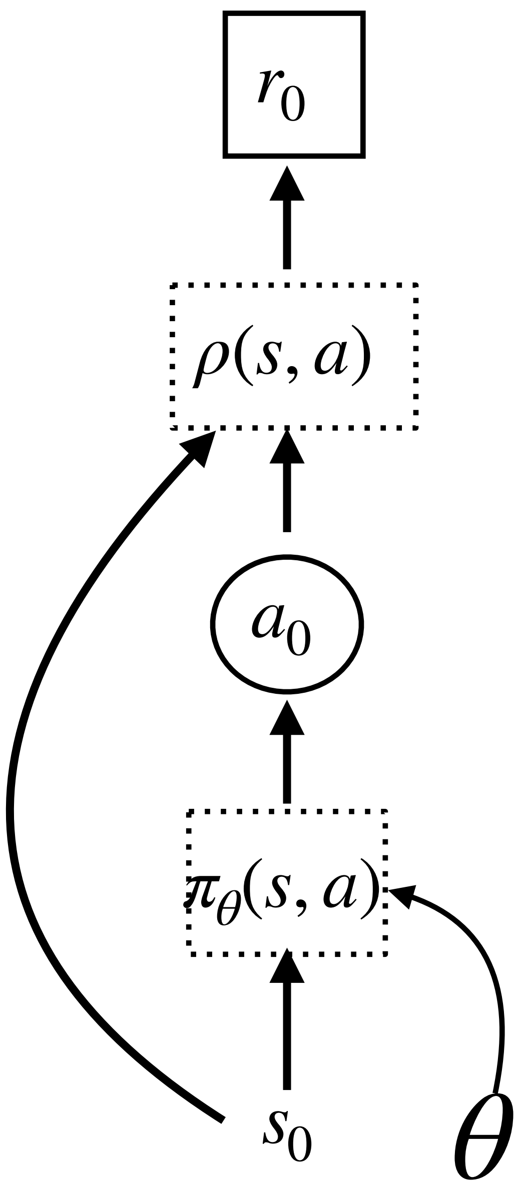


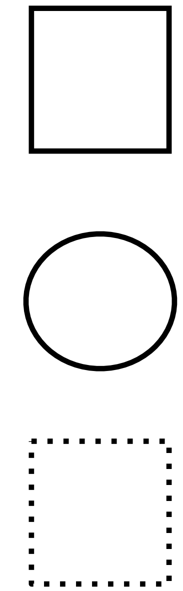
I want to learn \theta to maximize the reward obtained.

I can compute the gradient with backpropagation.

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

What if the policy is stochastic?





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)

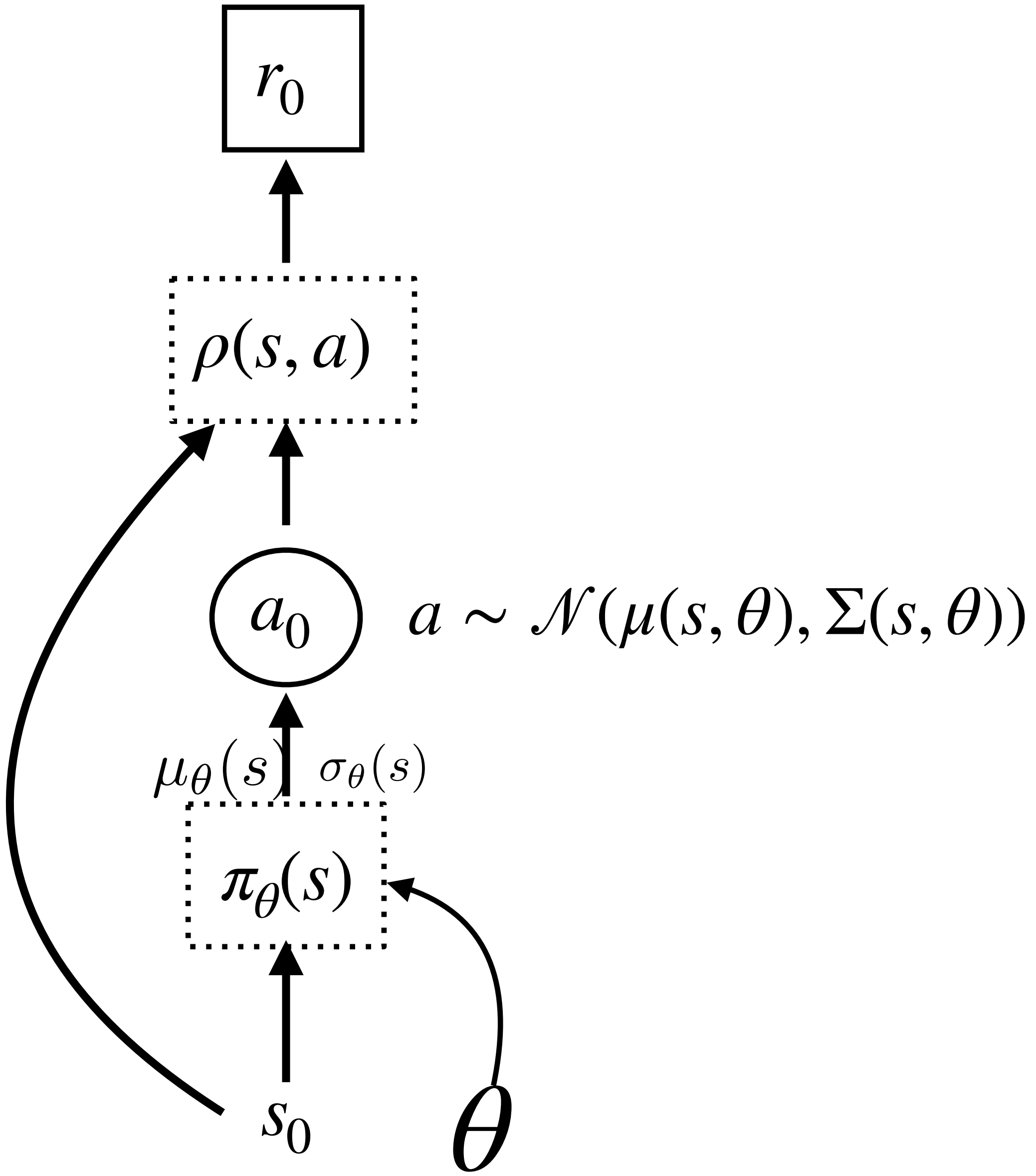
deterministic computation node

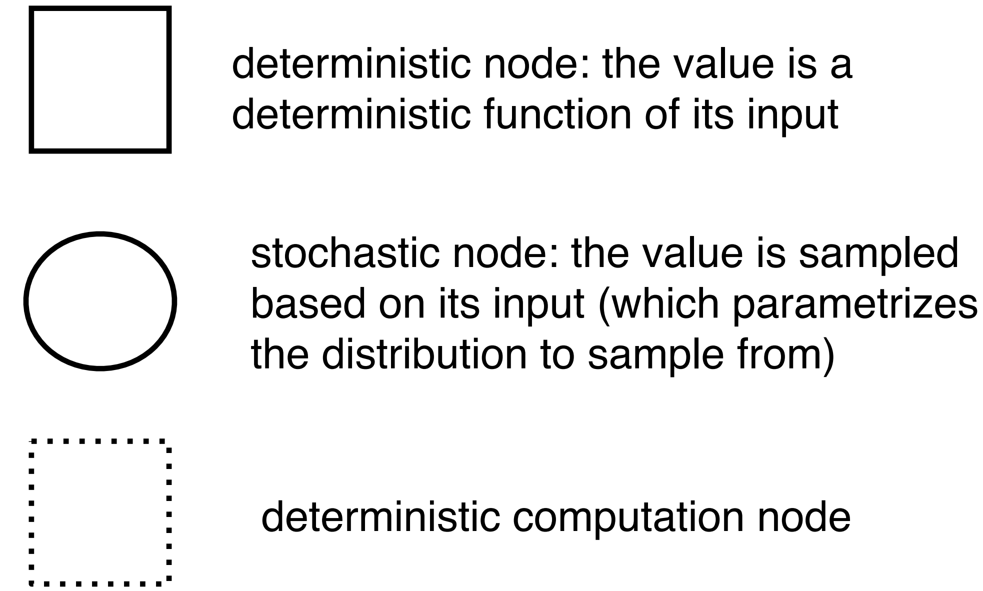
I want to learn \theta to maximize the reward obtained.

Likelihood ratio estimator, works for both continuous and discrete actions

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

Policies are parametrized Gaussians





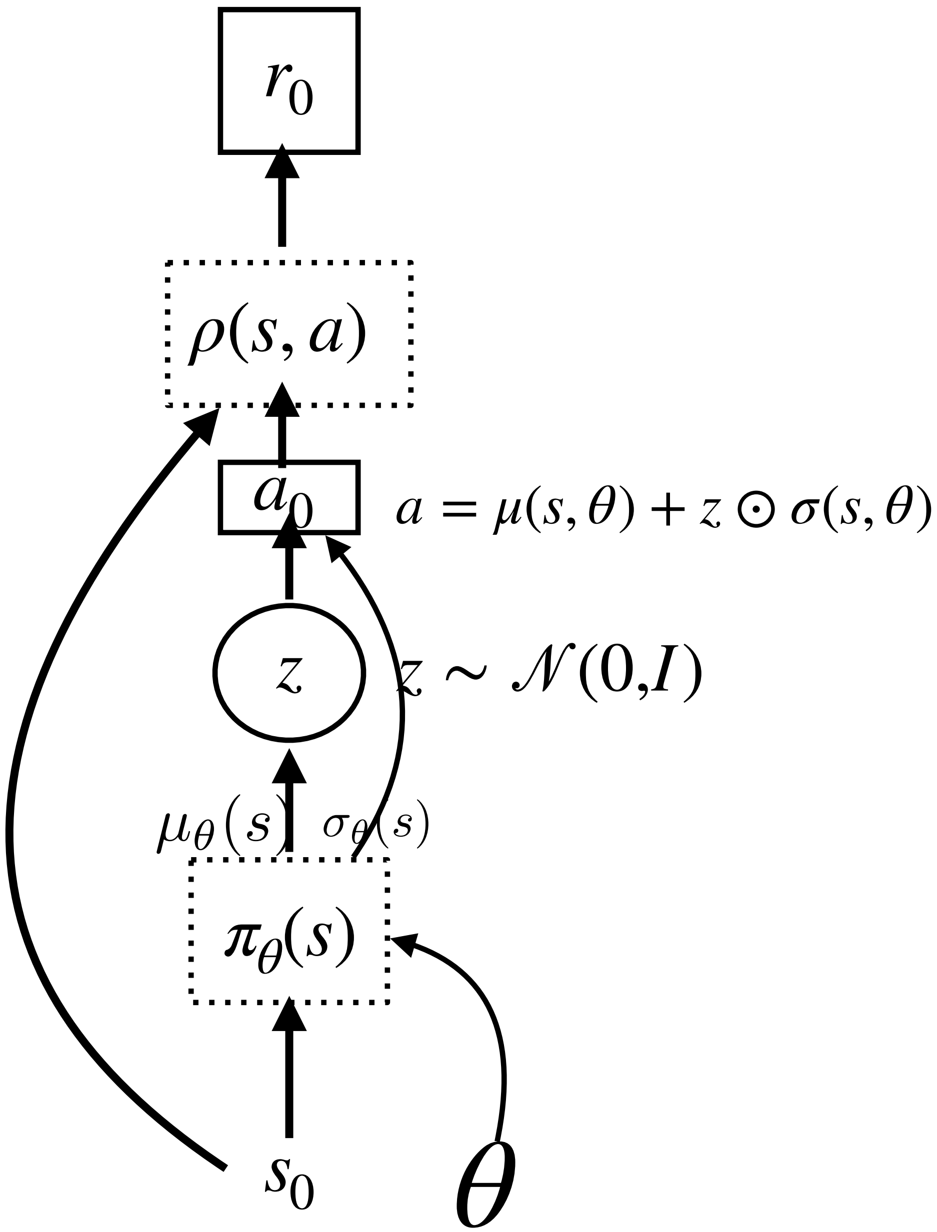
I want to learn \theta to maximize the reward obtained.

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

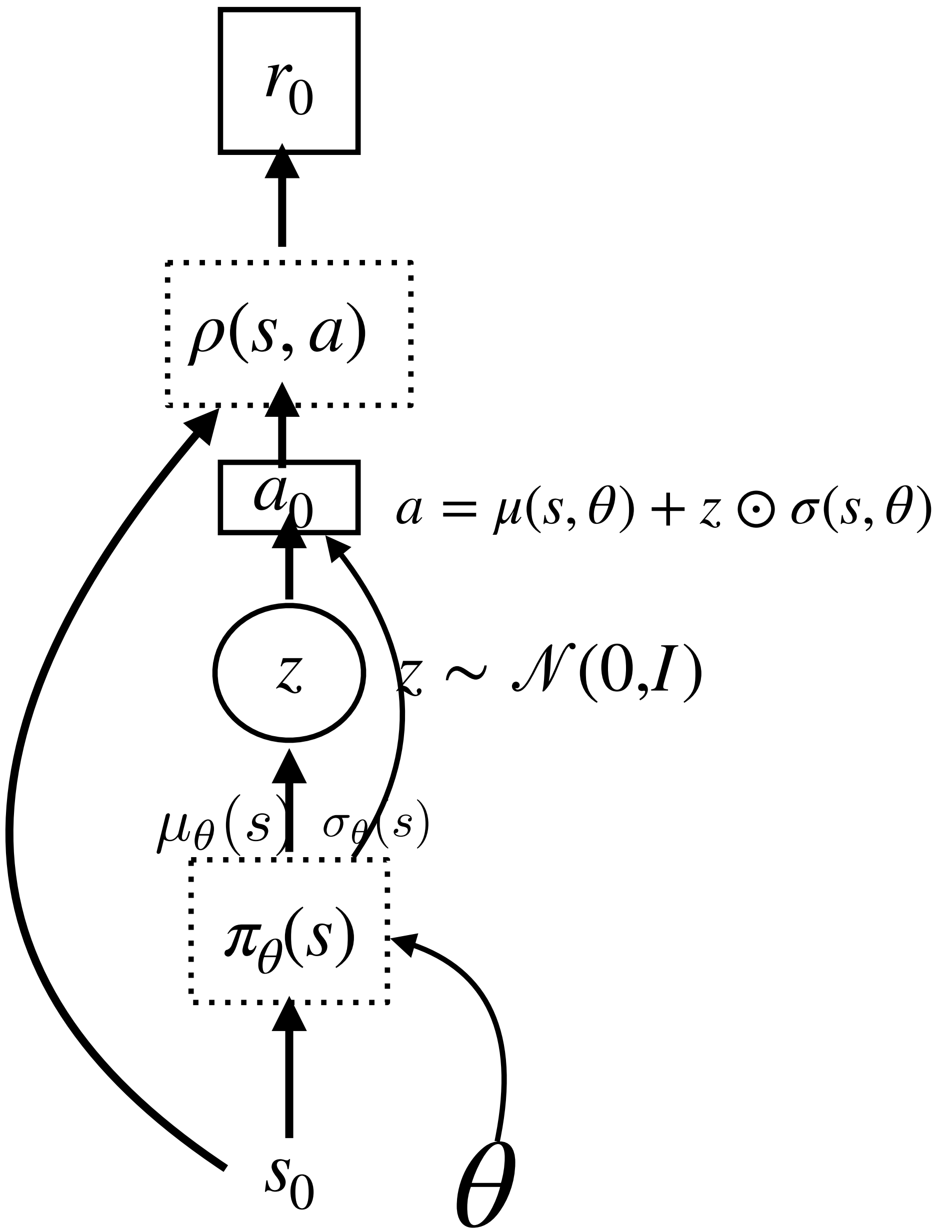
If σ^2 is constant:

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

Re-parametrization for Gaussian



Re-parametrization for Gaussian



$$\mathbb{E}(\mu + z\sigma) = \mu$$

$$\mathrm{Var}(\mu + z\sigma) = \sigma^2$$

isotropic

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

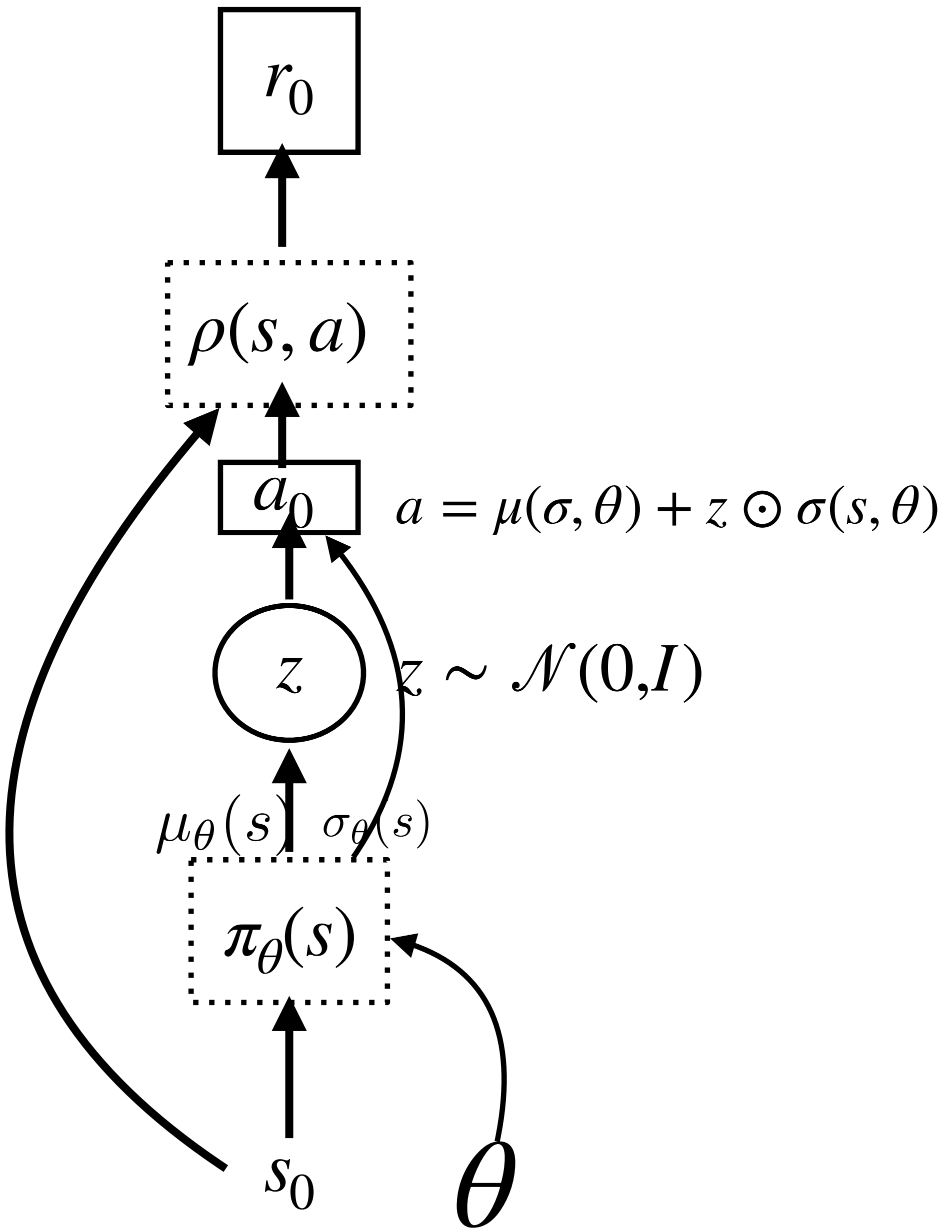
$$\frac{da}{d\theta} = \frac{d\mu(s,\theta)}{d\theta} + z \odot \frac{d\sigma(s,\theta)}{d\theta}$$

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

Sample estimate:

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

Re-parametrization for Gaussian



$$\mathbb{E}(\mu + z\sigma) = \mu$$

$$\mathrm{Var}(\mu + z\sigma) = \sigma^2$$

isotropic

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

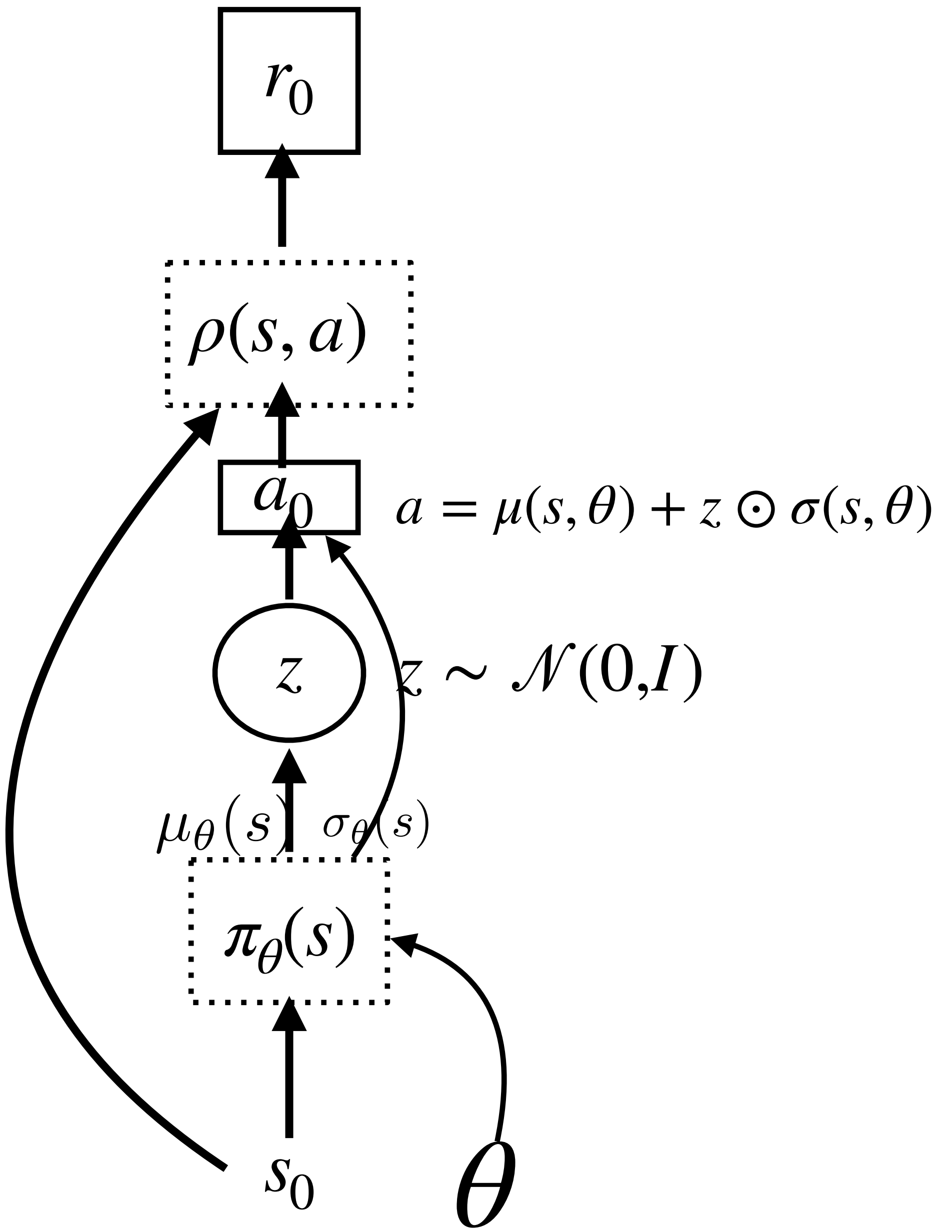
$$\frac{da}{d\theta} = \frac{d\mu(s,\theta)}{d\theta} + z \odot \frac{d\sigma(s,\theta)}{d\theta}$$

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

Sample estimate:

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

Re-parametrization for Gaussian



$$\mathbb{E}(\mu + z\sigma) = \mu$$
$$\mathrm{Var}(\mu + z\sigma) = \sigma^2$$

isotropic

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

Sample estimate:

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

general

$$a = \mu(\sigma,\theta) + Lz, \quad \Sigma = LL^\top$$

Likelihood ratio grad estimator:

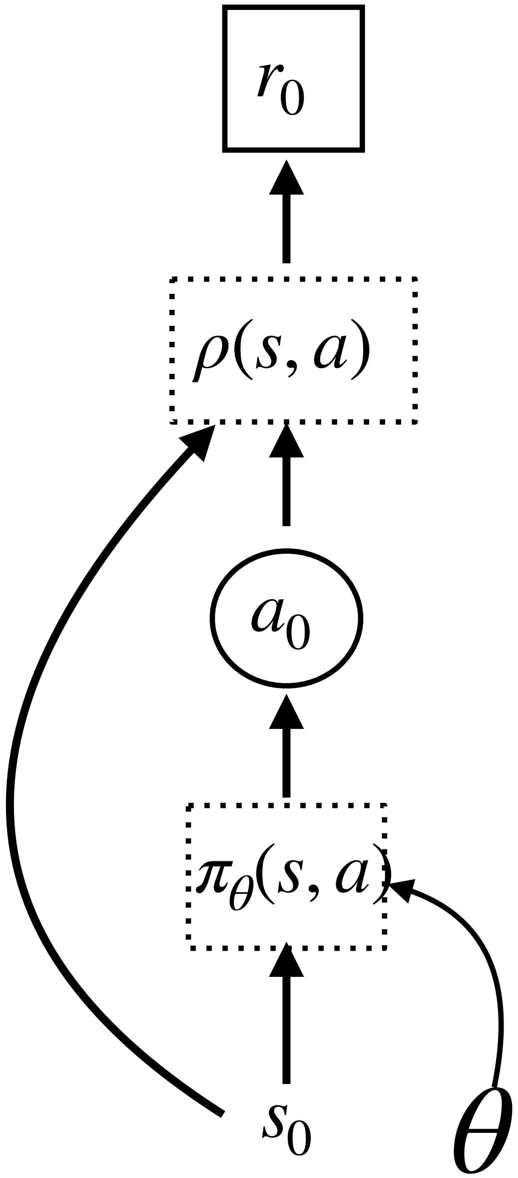
$$\boxed{\mathbb{E}_a \nabla_\theta \log \pi_\theta(s,a)\rho(s,a)}$$

Pathwise derivative:

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

The pathwise derivative uses the derivative of the reward w.r.t. the action!

Policies are parametrized Categorical distr



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.

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

Re-parametrization for categorical distributions

Consider variable y following the K categorical distribution:

$$y_k \sim \frac{\exp((\log p_k)/\tau)}{\sum_{j=0}^{K} \exp((\log p_j)/\tau)}$$

Categorical reparametrization with Gumbel-Softmax, Sang et al. 2017

Re-parametrization trick for categorical distributions

Consider variable y following the K categorical distribution:

$$y_k \sim \frac{\exp((\log p_k)/\tau)}{\sum_{j=0}^{K} \exp((\log p_j)/\tau)}$$

Re-parametrization:

$$a_k = \arg\max_k(\log p_k + \epsilon_k), \quad \epsilon_k = -\log(-\log(U)), \quad u \sim \mathcal{U}[0,1]$$

Categorical reparametrization with Gumbel-Softmax, Sang et al. 2017

Re-parametrization trick for categorical distributions

Consider variable y following the K categorical distribution:

$$a_k \sim \frac{\exp((\log p_k)/\tau)}{\sum_{j=0}^{K} \exp((\log p_j)/\tau)}$$

Reparametrization:

$$a_k = \arg\max_k(\log p_k + \epsilon_k), \quad \epsilon_k = -\log(-\log(U)), \quad u \sim \mathcal{U}[0,1]$$

In the forward pass you sample from the parametrized distribution

$$a_k \sim G(\log p)$$

In the backward pass you use the soft distribution:

$$\frac{da}{d\theta} = \frac{dG}{dp}\frac{dp}{d\theta}$$

Categorical reparametrization with Gumbel-Softmax, Sang et al. 2017

Re-parametrization trick for categorical distributions

Consider variable y following the K categorical distribution:

$$a_k \sim \frac{\exp((\log p_k)/\tau)}{\sum_{j=0}^{K} \exp((\log p_j)/\tau)}$$

Reparametrization:

$$a_k = \arg\max_k(\log p_k + \epsilon_k), \quad \epsilon_k = -\log(-\log(U)), \quad u \sim \mathcal{U}[0,1]$$

In the forward pass you sample from the parametrized distribution

$$a_k \sim G(\log p)$$

In the backward pass you use the soft distribution:

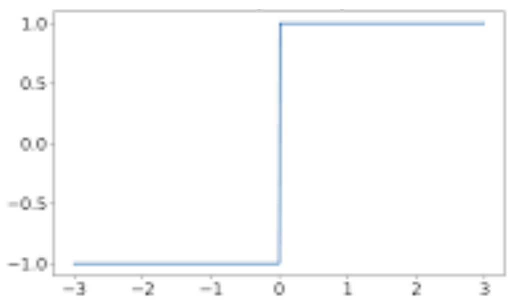
$$\frac{da}{d\theta} = \frac{dG}{dp}\frac{dp}{d\theta}$$

Categorical reparametrization with Gumbel-Softmax, Sang et al. 2017

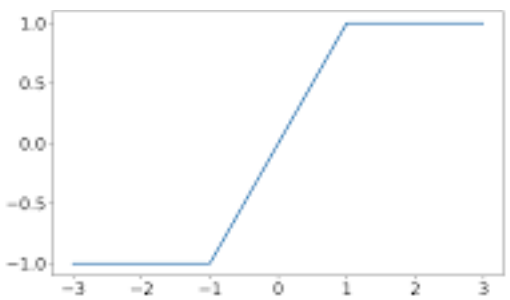
Back-propagating through discrete variables

For binary neurons:

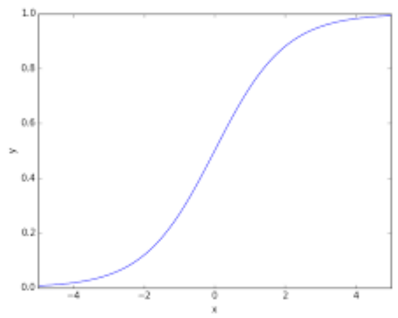
forward pass



backward pass



Straight-through

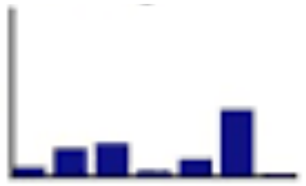


sigmoidal

For general categorically distributed neurons:

forward pass

backward pass

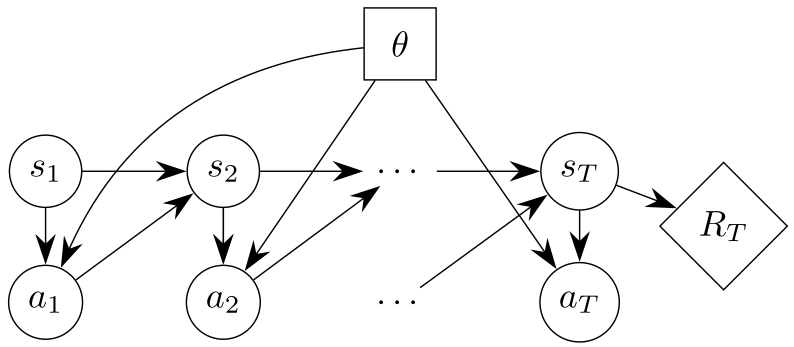


http://r2rt.com/binary-stochastic-neurons-in-tensorflow.html

Categorical reparametrization with Gumbel-Softmax, Sang et al. 2017

Re-parametrized Policy Gradients

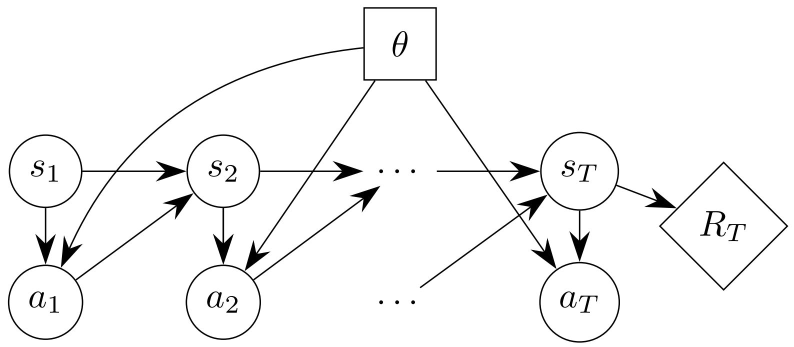
- Episodic MDP:



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

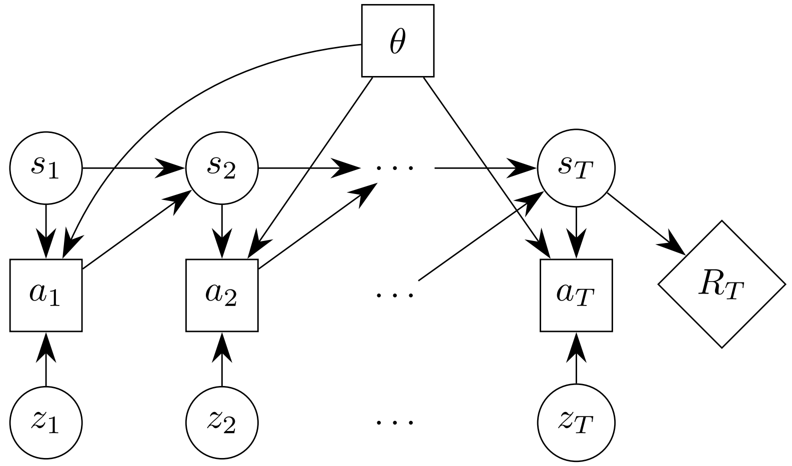
Re-parametrized Policy Gradients

- Episodic MDP:



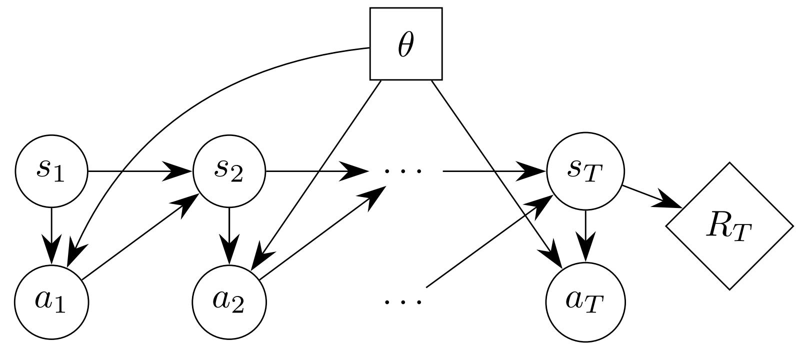
We want to compute: $\nabla_\theta \mathbb{E}[R_T]$

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



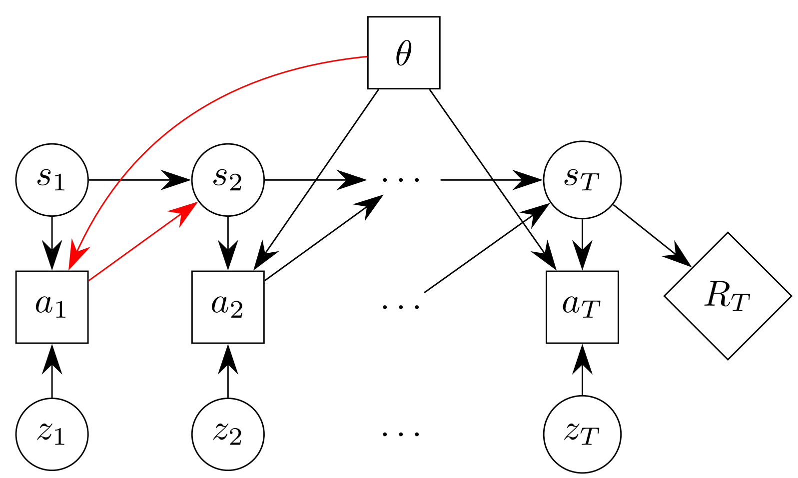
Re-parametrized Policy Gradients

- Episodic MDP:



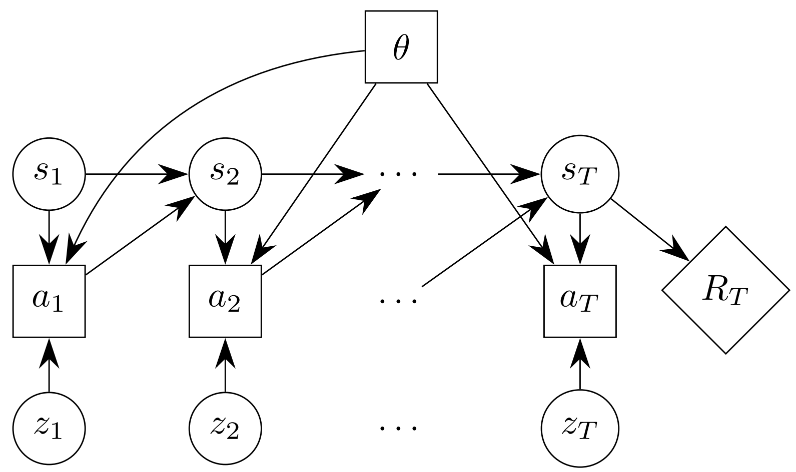
We want to compute: $\nabla_\theta \mathbb{E}[R_T]$

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



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

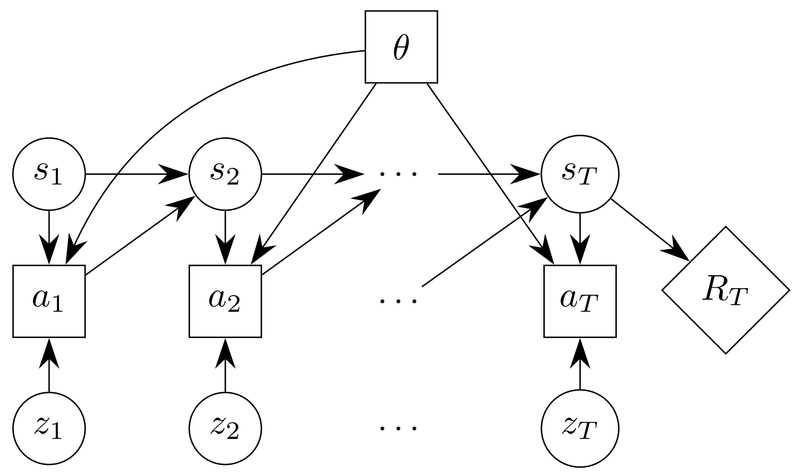
Re-parametrized Policy Gradients



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

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

Re-parametrized Policy Gradients



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

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

N. Heess, G. Wayne, D. Silver, et al. "Learning continuous control policies by stochastic value gradients". In: *NIPS*. 2015

Stochastic Value Gradients V0

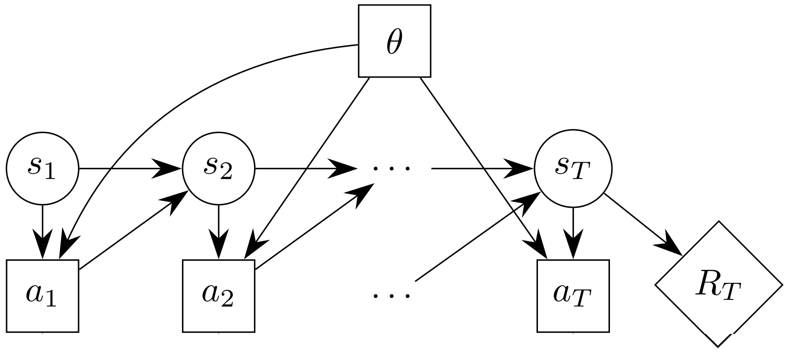
- Learn Q_ϕ to approximate $Q^{\pi,\gamma}$, and use it to compute gradient estimates.
- Pseudocode:

for iteration$=1, 2, \ldots$ **do**
  Execute policy π_θ to collect T timesteps of data
  Update π_θ using $g \propto \nabla_\theta \sum_{t=1}^{T} Q(s_t, \pi(s_t, z_t; \theta))$
  Update Q_ϕ 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

What if we give up on stochastic actions?

N. Heess, G. Wayne, D. Silver, et al. "Learning continuous control policies by stochastic value gradients". In *NIPS*. 2015

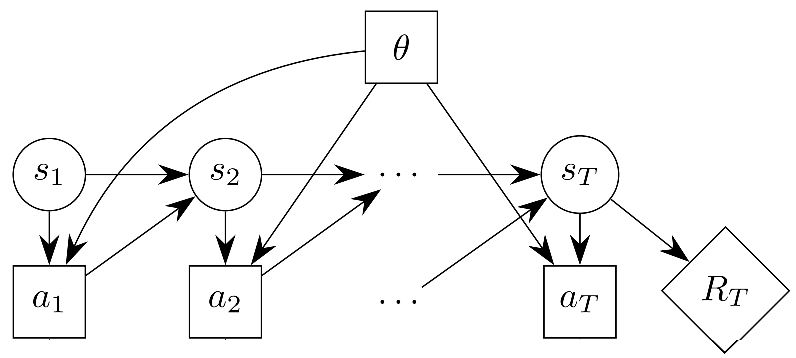
Deep Deterministic Policy Gradients



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

Continuous control with deep reinforcement learning, Lilicrap et al. 2016

Deep Deterministic Policy Gradients

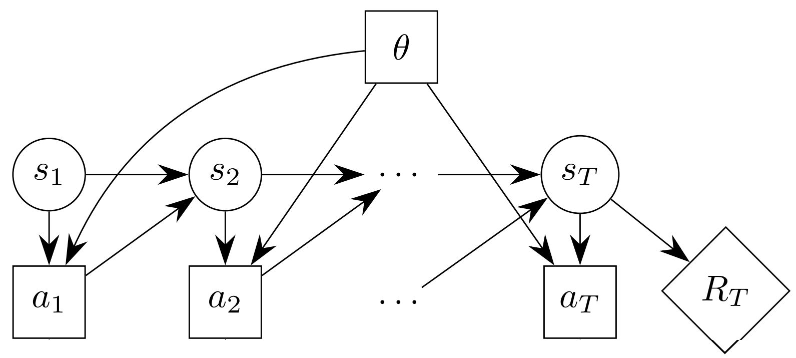


This expectation refers to the dynamics after time t

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

Continuous control with deep reinforcement learning, Lilicarp et al. 2016

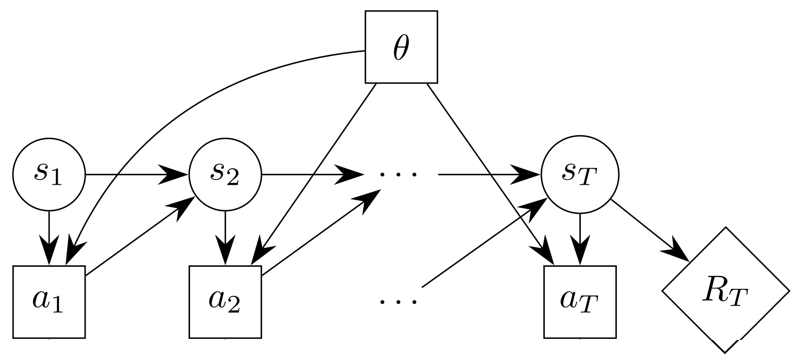
Deep Deterministic Policy Gradients



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

Continuous control with deep reinforcement learning, Lilicrap et al. 2016

Deep Deterministic Policy Gradients



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

$$= \mathbb{E}\left[\sum_{t=1}^{T}\frac{\mathrm{d}Q(s_t, a_t)}{\mathrm{d}a_t}\frac{\mathrm{d}a_t}{\mathrm{d}\theta}\right] = \mathbb{E}\left[\sum_{t=1}^{T}\frac{\mathrm{d}}{\mathrm{d}\theta}Q(s_t, \pi(s_t;\theta))\right]$$

Continuous control with deep reinforcement learning, Lilicrap et al. 2016

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 θ^Q and θ^μ.
Initialize target network Q' and μ' 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:

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

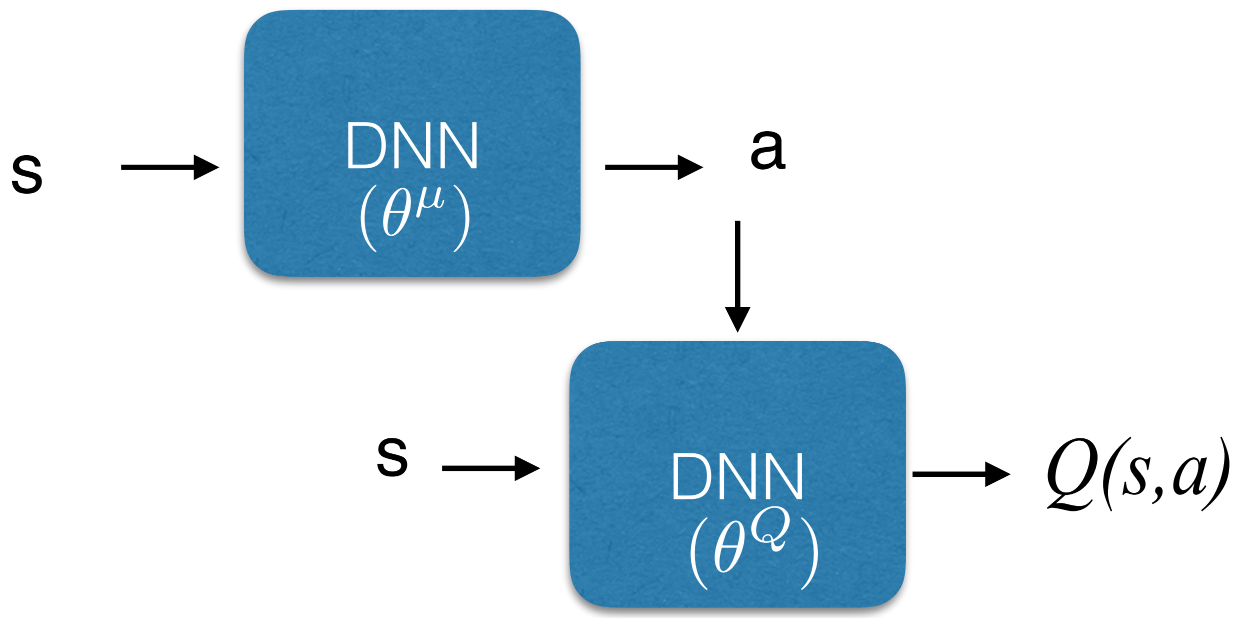
    Update the target networks:

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

  end for
end for

Deep Deterministic Policy Gradients

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

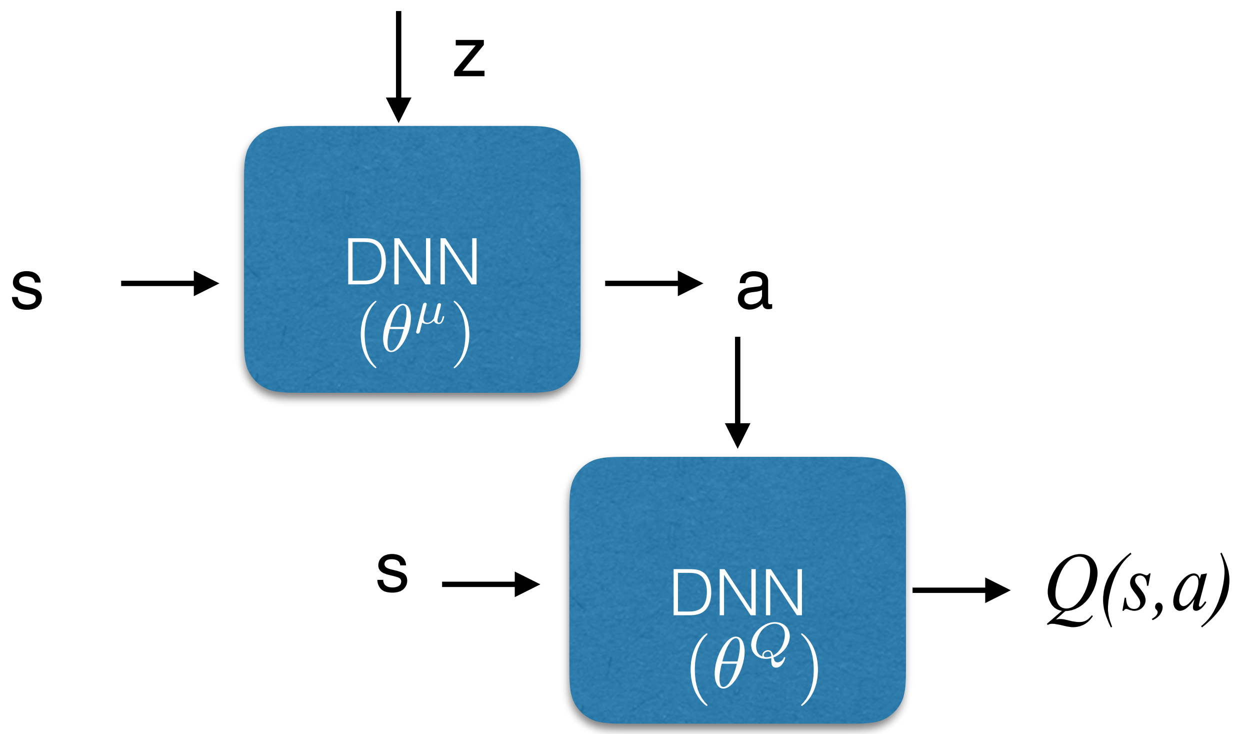


$$\nabla_{\theta^\mu} J \approx \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)} \right]$$
$$= \mathbb{E}_{s_t \sim \rho^\beta} \left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t} \right]$$

We are following a stochastic behavior policy to collect data.
Deep Q learning for contours actions-> DDPG

Stochastic Value Gradients V0

$$z \sim \mathcal{N}(0, 1)$$

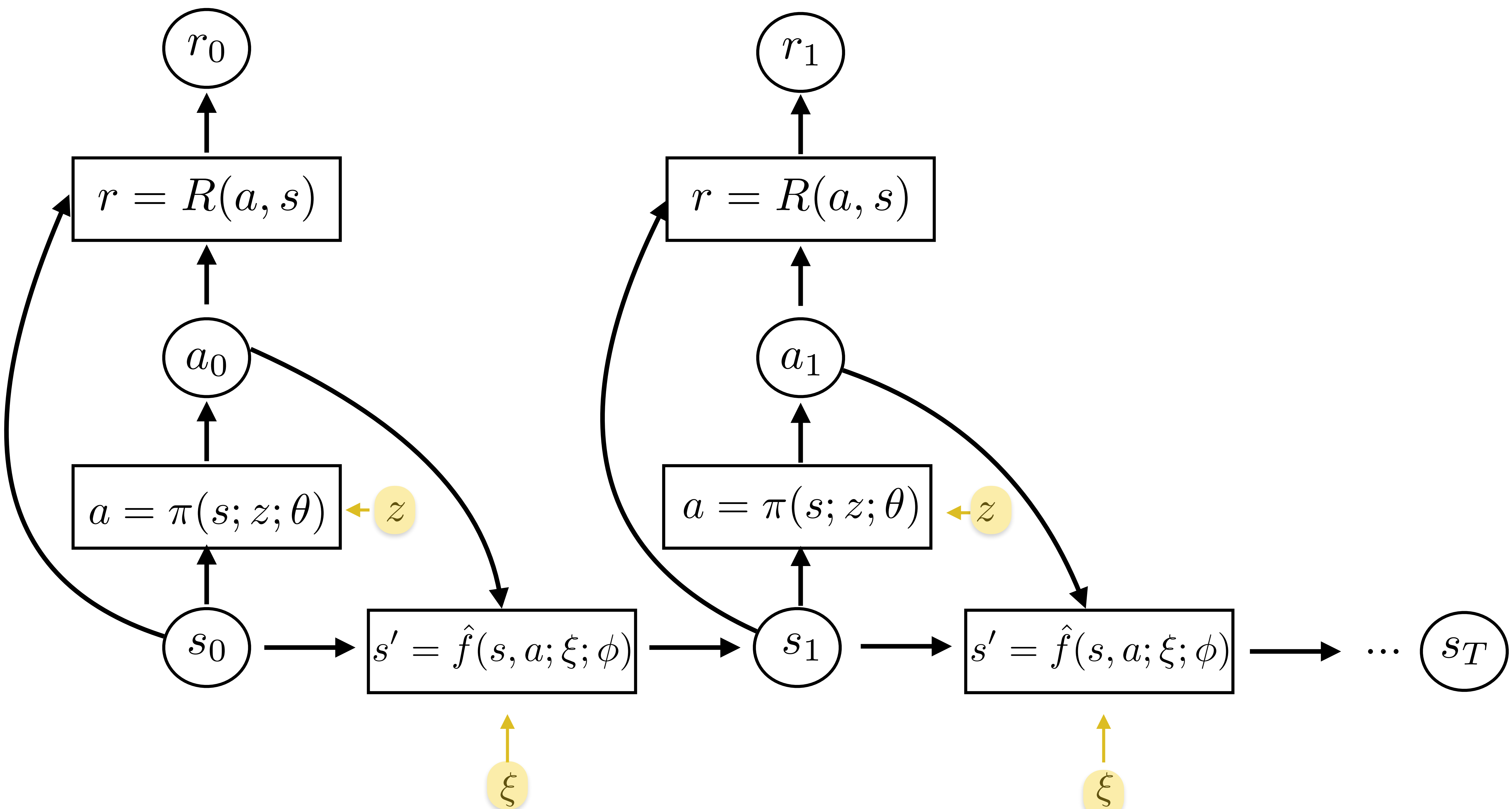


$$a = \mu(s;\theta) + z\sigma(s;\theta)$$

(where are the other versions? We will see them in the model based RL lecture)

End-to-end model based RL

Re-parametrization trick for both policies and dynamics



N. Heess, G. Wayne, D. Silver, et al. "Learning continuous control policies by stochastic value gradients". In: *NIPS*. 2015

Deep Deterministic Policy Gradients

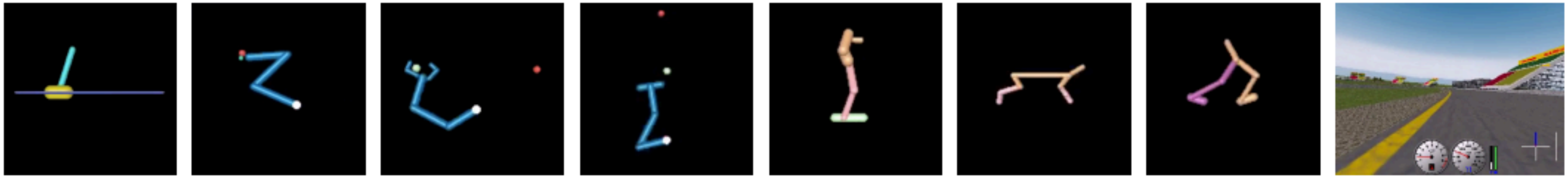


Figure 1: Example screenshots of a sample of environments we attempt to solve with DDPG. In order from the left: the cartpole swing-up task, a reaching task, a gasp and move task, a puck-hitting task, a monoped balancing task, two locomotion tasks and Torcs (driving simulator). We tackle all tasks using 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

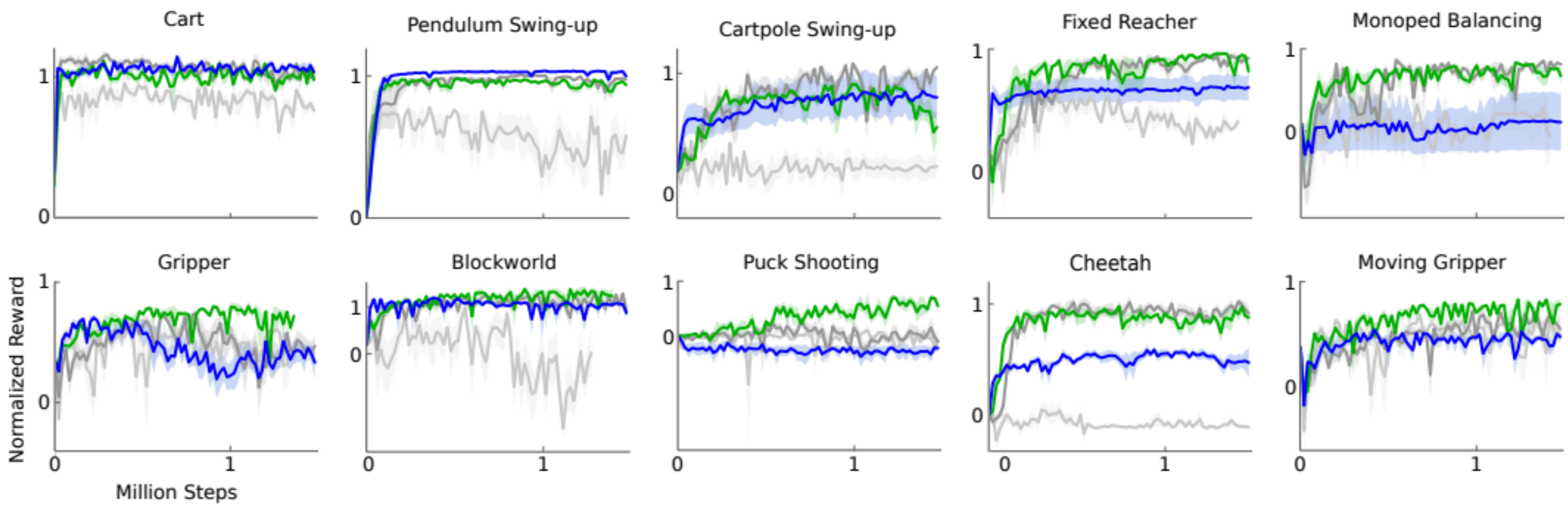


Figure 2: Performance curves for a selection of domains using variants of DPG: original DPG algorithm (minibatch NFQCA) with batch normalization (light grey), with target network (dark grey), with target networks and batch normalization (green), with target networks from pixel-only inputs (blue). Target 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	**3986.4 ± 748.9**	**4861.5 ± 12.3**	565.6 ± 137.6	**4869.8 ± 37.6**	4815.4 ± 4.8	2440.4 ± 568.3	4634.4 ± 87.8
Inverted Pendulum*	−153.4 ± 0.2	13.4 ± 18.0	**209.7 ± 55.5**	84.7 ± 13.8	−113.3 ± 4.6	**247.2 ± 76.1**	38.2 ± 25.7	−40.1 ± 5.7	40.0 ± 244.6
Mountain Car	−415.4 ± 0.0	−67.1 ± 1.0	**-66.5 ± 4.5**	−79.4 ± 1.1	−275.6 ± 166.3	**-61.7 ± 0.9**	−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	**-223.6 ± 5.8**
Double Inverted Pendulum*	149.7 ± 0.1	4116.5 ± 65.2	**4455.4 ± 37.6**	3614.8 ± 368.1	446.7 ± 114.8	**4412.4 ± 50.4**	2566.2 ± 178.9	1576.1 ± 51.3	2863.4 ± 154.0
Swimmer*	−1.7 ± 0.1	92.3 ± 0.1	**96.0 ± 0.2**	60.7 ± 5.5	3.8 ± 3.3	**96.0 ± 0.2**	68.8 ± 2.4	64.9 ± 1.4	85.8 ± 1.8
Hopper	8.4 ± 0.0	714.0 ± 29.3	**1155.1 ± 57.9**	553.2 ± 71.0	86.7 ± 17.6	**1183.3 ± 150.0**	63.1 ± 7.8	20.3 ± 14.3	267.1 ± 43.5
2D Walker	−1.7 ± 0.0	506.5 ± 78.8	**1382.6 ± 108.2**	136.0 ± 15.9	−37.0 ± 38.1	**1353.8 ± 85.0**	84.5 ± 19.2	77.1 ± 24.3	318.4 ± 181.6
Half-Cheetah	−90.8 ± 0.3	1183.1 ± 69.2	**1729.5 ± 184.6**	376.1 ± 28.2	34.5 ± 38.0	**1914.0 ± 120.1**	330.4 ± 274.8	441.3 ± 107.6	**2148.6 ± 702.7**
Ant*	13.4 ± 0.7	548.3 ± 55.5	**706.0 ± 127.7**	37.6 ± 3.1	39.0 ± 9.8	**730.2 ± 61.3**	49.2 ± 5.9	17.8 ± 15.5	326.2 ± 20.8
Simple Humanoid	41.5 ± 0.2	128.1 ± 34.0	**255.0 ± 24.5**	93.3 ± 17.4	28.3 ± 4.7	**269.7 ± 40.3**	60.6 ± 12.9	28.7 ± 3.9	99.4 ± 28.1
Full Humanoid	13.2 ± 0.1	262.2 ± 10.5	**288.4 ± 25.2**	46.7 ± 5.6	41.7 ± 6.1	**287.0 ± 23.4**	36.9 ± 2.9	N/A ± N/A	119.0 ± 31.2
Cart-Pole Balancing (LS)*	77.1 ± 0.0	420.9 ± 265.5	**945.1 ± 27.8**	68.9 ± 1.5	898.1 ± 22.1	**960.2 ± 46.0**	227.0 ± 223.0	68.0 ± 1.6	
Inverted Pendulum (LS)	−122.1 ± 0.1	−13.4 ± 3.2	**0.7 ± 6.1**	−107.4 ± 0.2	−87.2 ± 8.0	**4.5 ± 4.1**	−81.2 ± 33.2	−62.4 ± 3.4	
Mountain Car (LS)	−83.0 ± 0.0	−81.2 ± 0.6	**-65.7** ± 9.0	−81.7 ± 0.1	−82.6 ± 0.4	**-64.2 ± 9.5**	**-68.9 ± 1.3**	**-73.2 ± 0.6**	
Acrobot (LS)*	−393.2 ± 0.0	−128.9 ± 11.6	**-84.6 ± 2.9**	−235.9 ± 5.3	−379.5 ± 1.4	**-83.3 ± 9.9**	−149.5 ± 15.3	−159.9 ± 7.5	
Cart-Pole Balancing (NO)*	101.4 ± 0.1	616.0 ± 210.8	**916.3 ± 23.0**	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	**11.5 ± 0.5**	−110.0 ± 1.4	−119.3 ± 4.2	**10.4 ± 2.2**	−55.6 ± 16.7	−80.3 ± 2.8	
Mountain Car (NO)	−83.0 ± 0.0	−74.7 ± 7.8	**-64.5 ± 8.6**	−81.7 ± 0.1	−82.9 ± 0.1	**-60.2 ± 2.0**	−67.4 ± 1.4	−73.5 ± 0.5	
Acrobot (NO)*	−393.5 ± 0.0	**-186.7 ± 31.3**	**-164.5 ± 13.4**	−233.1 ± 0.4	−258.5 ± 14.0	**-149.6 ± 8.6**	−213.4 ± 6.3	−236.6 ± 6.2	
Cart-Pole Balancing (SI)*	76.3 ± 0.1	431.7 ± 274.1	**980.5 ± 7.3**	69.0 ± 2.8	702.4 ± 196.4	**980.3 ± 5.1**	746.6 ± 93.2	71.6 ± 2.9	
Inverted Pendulum (SI)	−121.8 ± 0.2	−5.3 ± 5.6	**14.8 ± 1.7**	−108.7 ± 4.7	−92.8 ± 23.9	**14.1 ± 0.9**	−51.8 ± 10.6	−63.1 ± 4.8	
Mountain Car (SI)	−82.7 ± 0.0	−63.9 ± 0.2	**-61.8 ± 0.4**	−81.4 ± 0.1	−80.7 ± 2.3	**-61.6 ± 0.4**	−63.9 ± 1.0	−66.9 ± 0.6	
Acrobot (SI)*	−387.8 ± 1.0	**-169.1 ± 32.3**	**-156.6 ± 38.9**	−233.2 ± 2.6	−216.1 ± 7.7	**-170.9 ± 40.3**	−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

Y. Duan, X. Chen, R. Houthooft, et al. "Benchmarking Deep Reinforcement Learning for Continuous Control"

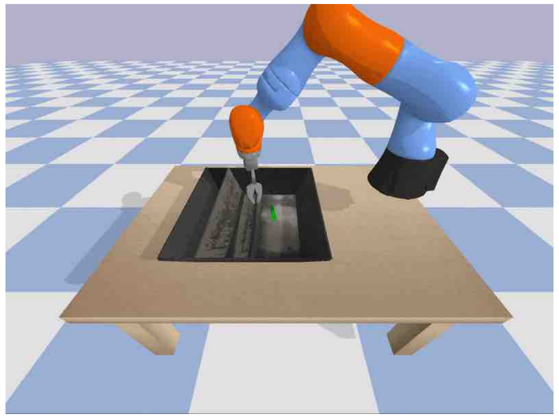
Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Multigoal RL

Katerina Fragkiadaki

So far we train one policy/value function per task, e.g., win the game of Tetris, win the game of Go, reach to a *particular* location, put the green cube inside the gray bucket, etc.



Universal value function Approximators

$$V(s;\theta) \rightarrow V(s,g;\theta)$$

$$\pi(s;\theta) \rightarrow \pi(s,g;\theta)$$

All the methods we have learnt so far can be used.

At the beginning of an episode, **we sample not only a start state but also a goal g**, which stays constant throughout the episode

The experience tuples should contain the goal.

$$(s,a,r,s') \rightarrow \left(s,g,a,r,s'\right)$$

Universal Value Function Approximators, Schaul et al.

Universal value function Approximators

$$V(s,\theta) \quad \rightarrow \quad V(s,\theta,g)$$

$$\pi(s;\theta) \quad \rightarrow \quad \pi(s,g;\theta)$$

What should be my goal representation?

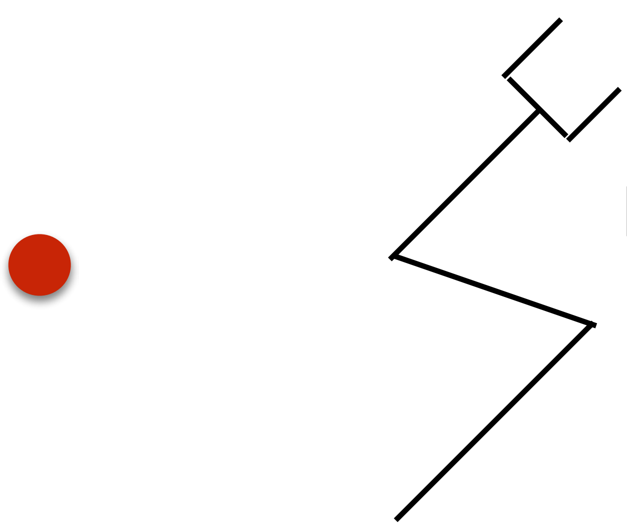
(not an easy question)

- **Manual**: 3d centroids of objects, robot joint angles and velocities, 3d location of the gripper, etc.
- **Learnt**: We supply **a target image as the goal**, and the method learns to map it to an embedding vector, e.g., asymmetric actor-critic, Lerrel et al.

Hindsight Experience Replay

Marcin Andrychowicz*, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel†, Wojciech Zaremba†
OpenAI

Main idea: use failed executions under one goal g, as successful executions under an alternative goal g' (which is where we ended spat the end of the episode)



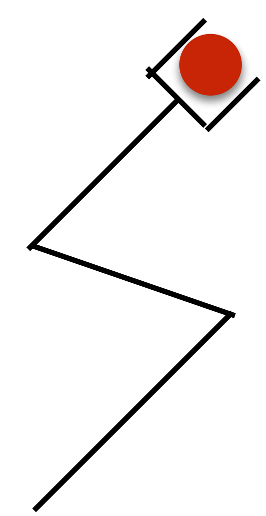
No reward :-(

Goal g

Our reacher at the end of the episode

$(s, g, a, 0, s')$

Goal g'



reward :-)

Our reacher at the end of the episode
$(s, g', a, 1, s')$

Hindsight Experience Replay

Marcin Andrychowicz*, **Filip Wolski**, **Alex Ray**, **Jonas Schneider**, **Rachel Fong**,
Peter Welinder, **Bob McGrew**, **Josh Tobin**, **Pieter Abbeel**†, **Wojciech Zaremba**†
OpenAI

Main idea: use failed executions under one goal g, as successful executions under an alternative goal g’ (which is where we ended spat the end of the episode)

Hindsight Experience Replay

Algorithm 1 Hindsight Experience Replay (HER)

Given:

- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, \ldots, s_T) = m(s_T)$
- a reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

Initialize $\mathbb{A}$ ▷ e.g. initialize neural networks
Initialize replay buffer R
for episode $= 1, M$ **do**
  Sample a goal g and an initial state s_0.
  for $t = 0, T-1$ **do**
    Sample an action a_t using the behavioral policy from $\mathbb{A}$:
      $a_t \leftarrow \pi_b(s_t||g)$ ▷ || denotes concatenation
    Execute the action a_t and observe a new state s_{t+1}
  end for
  for $t = 0, T-1$ **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in R ▷ standard experience replay
    Sample a set of additional goals for replay $G := \mathbb{S}($**current episode**$)$
    for $g' \in G$ **do**
      $r' := r(s_t, a_t, g')$
      Store the transition $(s_t||g', a_t, r', s_{t+1}||g')$ in R ▷ HER
    end for
  end for
  for $t = 1, N$ **do**
    Sample a minibatch B from the replay buffer R
    Perform one step of optimization using $\mathbb{A}$ and minibatch B
  end for
end for

Usually as additional goal we pick the goal that this episode achieved, and the reward becomes non zero

Hindsight Experience Replay

Reward shaping: instead of using binary rewards, use continuous rewards, e.g., by considering Euclidean distances from goal configuration

HER does not require reward shaping! :-)

The burden goes from designing the reward to designing the goal encoding.. :-(

Hindsight Experience Replay