10703 Deep Reinforcement Learning and Control

Russ Salakhutdinov

Machine Learning Department
rsalakhu@cs.cmu.edu

Slides developed and borrowed from
Katerina Fragkiadaki

End-to-end Model Based Reinforcement Learning

Today's Lecture

End-to-end policy optimization through back-propagation

Last week: Trajectory optimization

diagram: Igor Mordatch

Last week: Trajectory optimization

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

diagram: Igor Mordatch

Last week: Trajectory optimization

||x_t-x*||

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

diagram: Igor Mordatch

Last week: Trajectory optimization

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

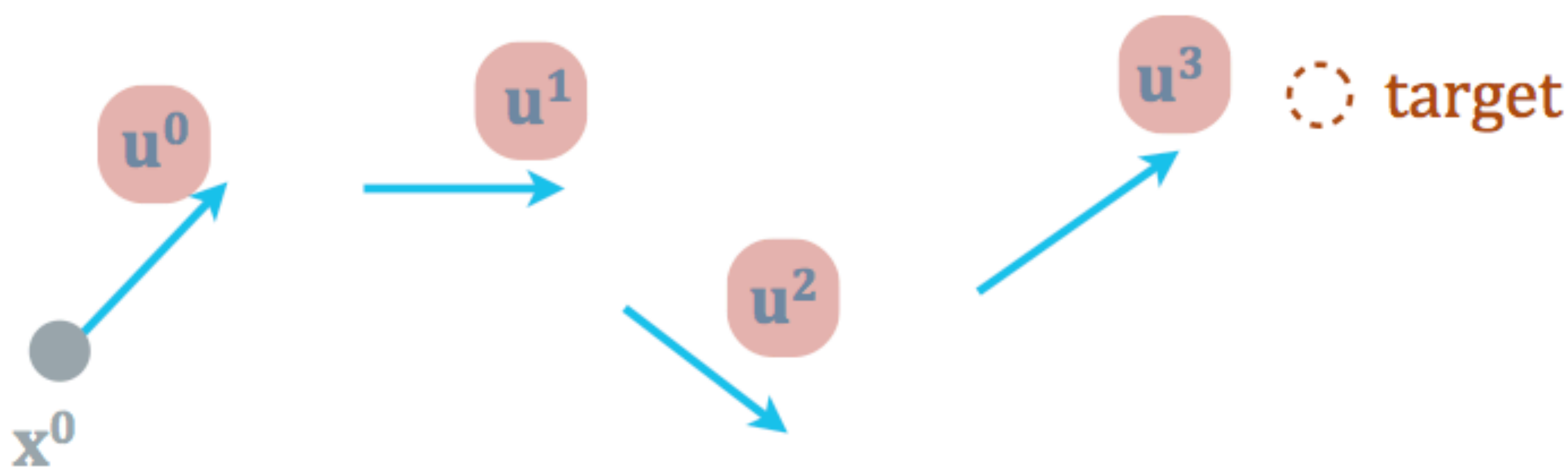


diagram: Igor Mordatch

Poor Conditioning

$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots$$
$$\cdots + c(f(f(\ldots)\ldots), \mathbf{u}_T)$$

Poor Conditioning

$$\min_{\mathbf{u}_1,\dots,\mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots$$
$$\cdots + c(f(f(\dots)\dots), \mathbf{u}_T)$$

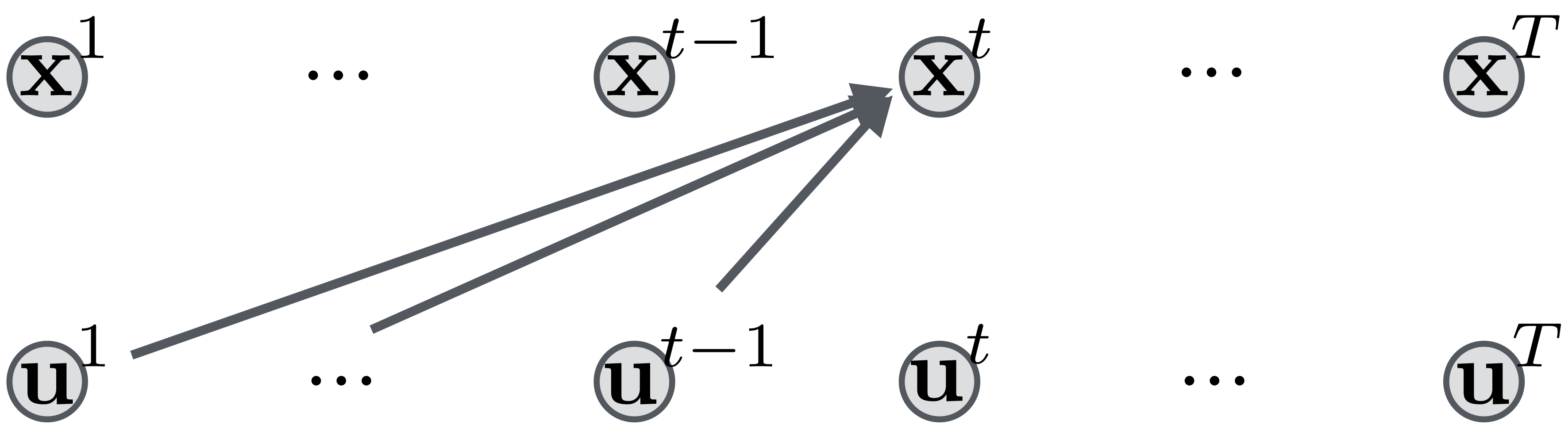


diagram: Igor Mordatch

Poor Conditioning

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

Poor Conditioning

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

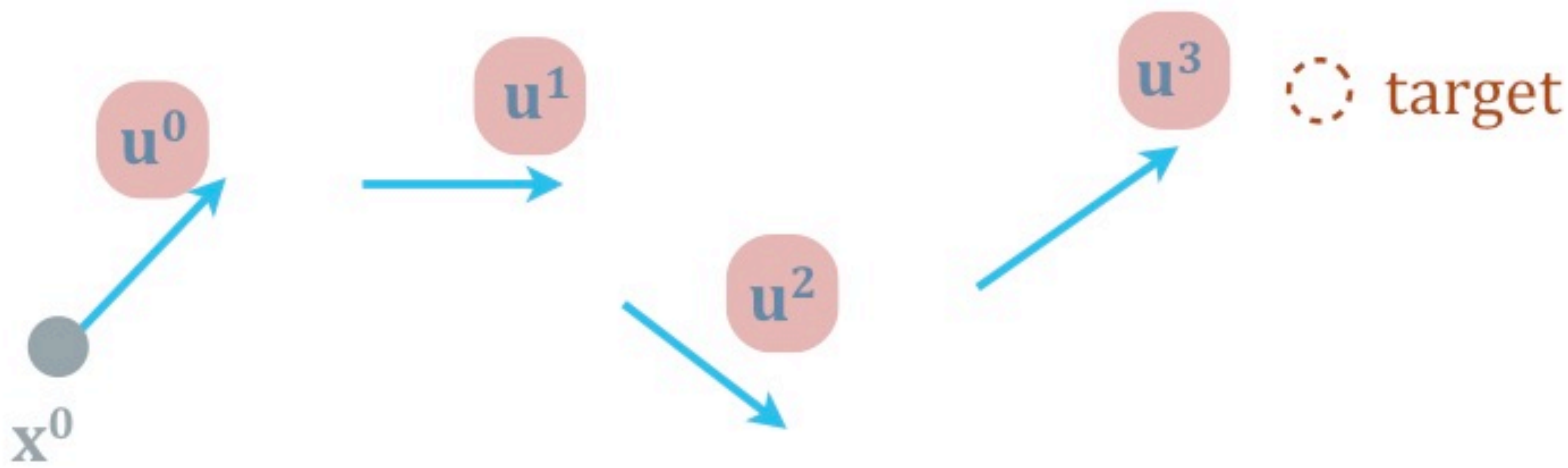
Poor Conditioning

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \quad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$

Trajectory Optimization

- Consider the special case of quadratic c and linear f

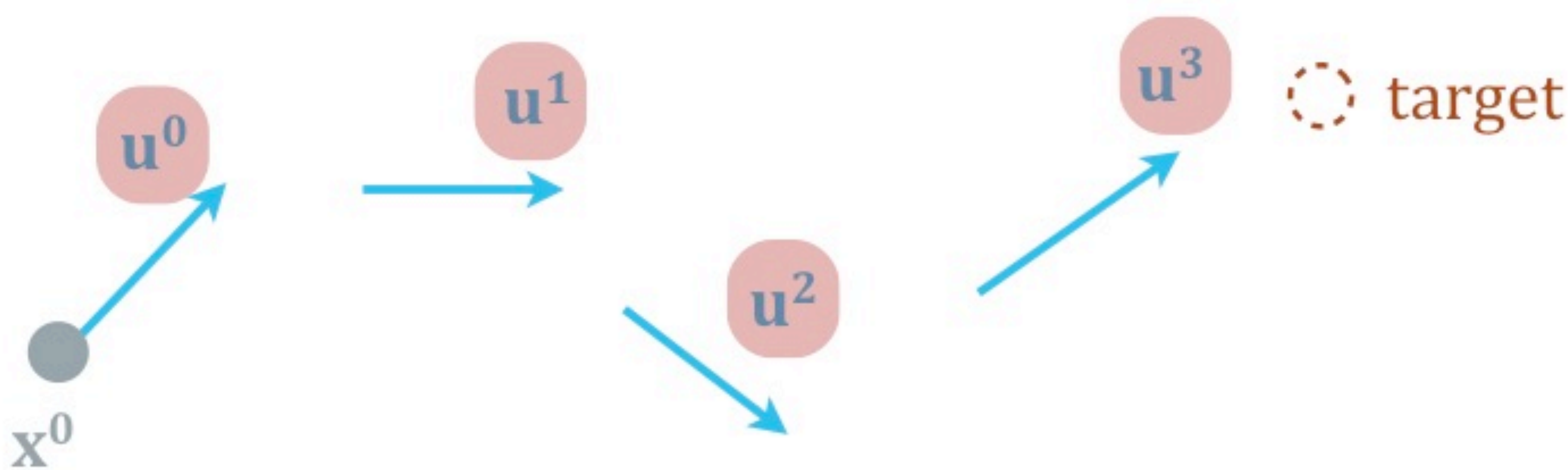
$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_{t=1} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$



Trajectory Optimization

- Consider the special case of quadratic c and linear f

$$\min_{\mathbf{u}^0 \ldots \mathbf{u}^T} \sum_{t=1} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \mathbf{u}^t)$$



Solve it using dynamic programming :

- write u_t^* as function of the state x_t at each $t = T, \ldots, 1$
- substitute x_0 (known)
- for $t = 1, \ldots, T$ substitute x_t into u_t^* and fire the dynamics forward to obtain next state state $(x_{t+1} = f(x_t, u_t^*))$

Learning Control Policies

$$\boldsymbol{\pi}_\theta \ : \ \mathbf{x} \mapsto \mathbf{u}$$

Learning Control Policies

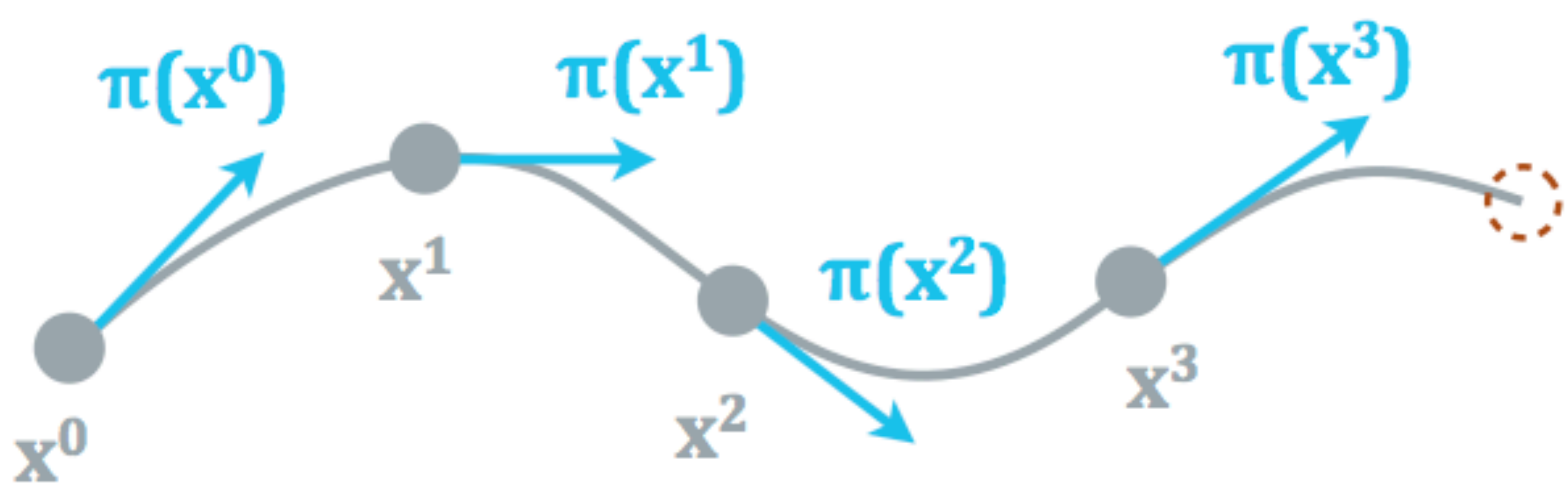
$$\boldsymbol{\pi}_\theta \ : \ \mathbf{x} \mapsto \mathbf{u}$$

$$\min_{\theta} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \boldsymbol{\pi}_\theta(\mathbf{x}^t))$$

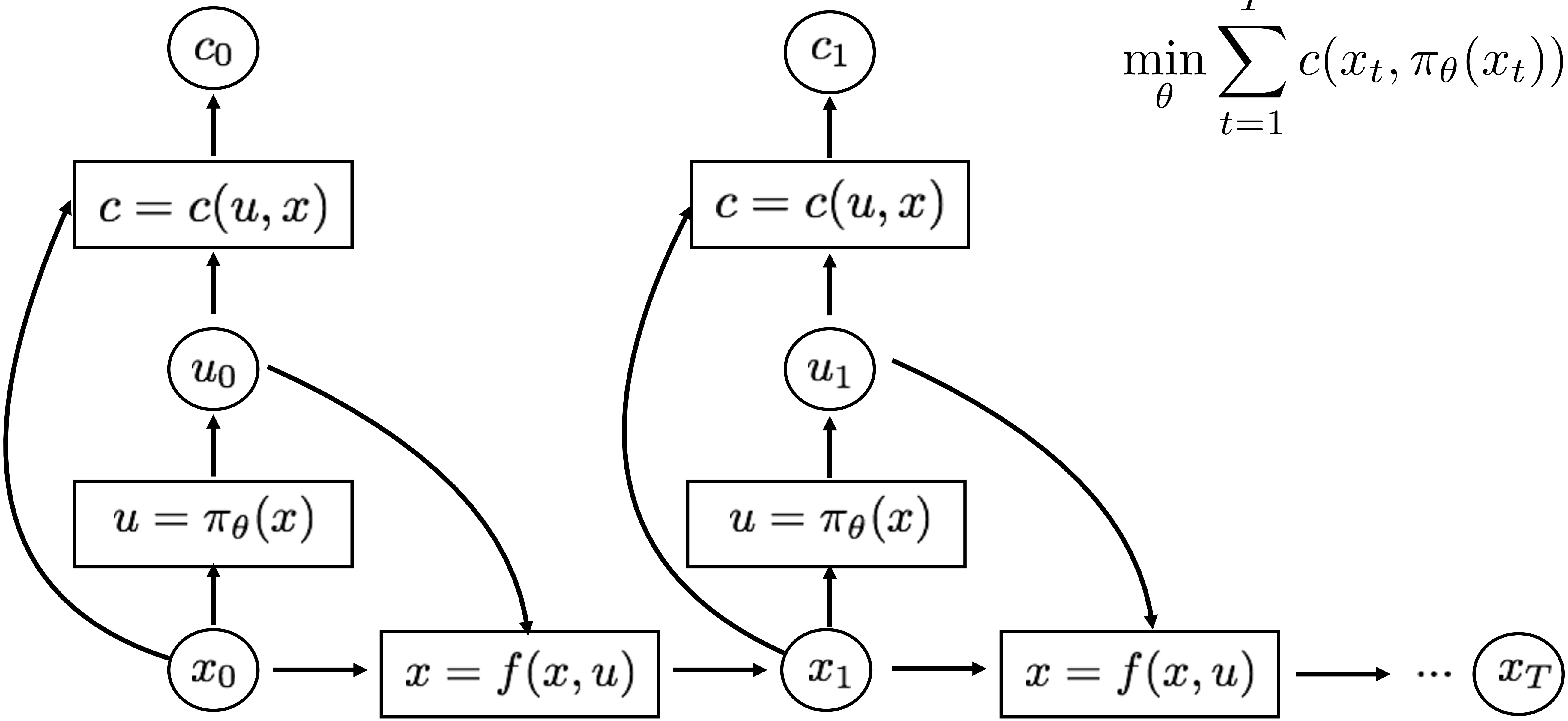
Learning Control Policies

$$\boldsymbol{\pi}_\theta \;:\; \mathbf{x} \;\mapsto\; \mathbf{u}$$

$$\min_\theta \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \qquad \mathbf{x}^{t+1} = f(\mathbf{x}^t, \boldsymbol{\pi}_\theta(\mathbf{x}^t))$$



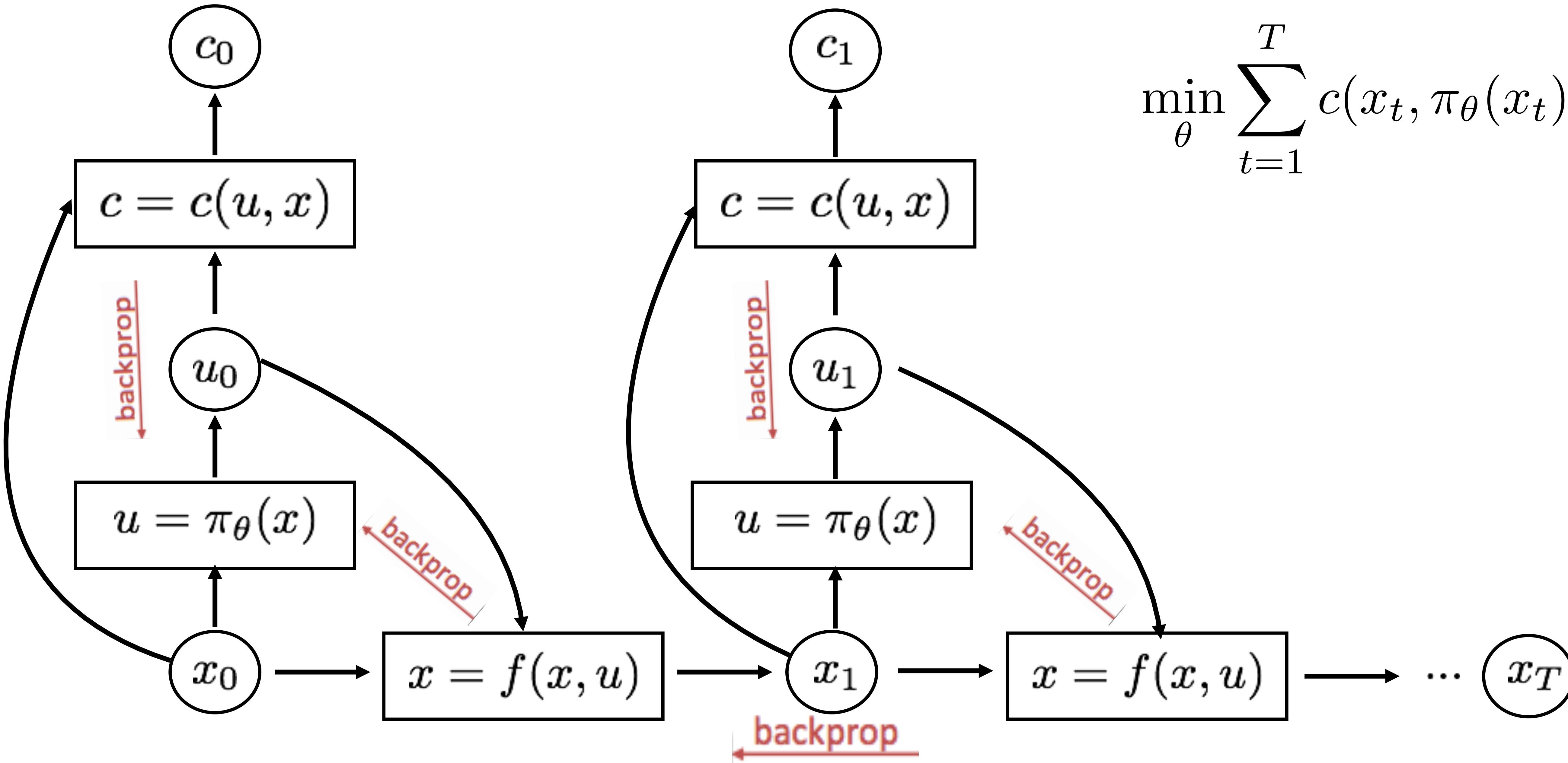
Learning Control Policies through Backpropagation



$$\min_\theta \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))$$

- So far, dynamics are assumed known and deterministic.
- Policy is assumed deterministic.
- We solve for policy parameters θ using back-propagating (through time).

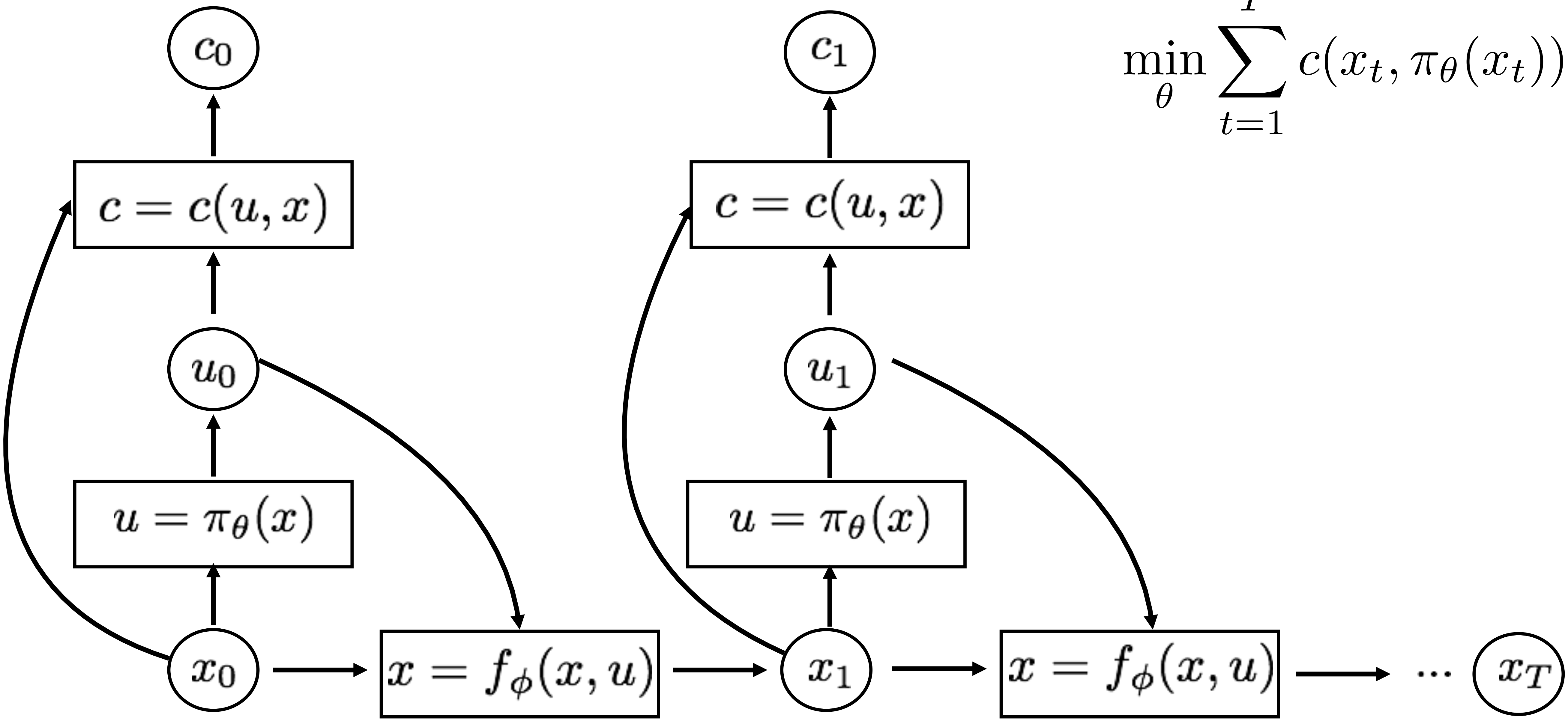
Learning Control Policies through Backpropagation



$$\min_\theta \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))$$

- So far, dynamics are assumed known and deterministic.
- Policy is assumed deterministic.
- We solve for policy parameters θ using back-propagating (through time).

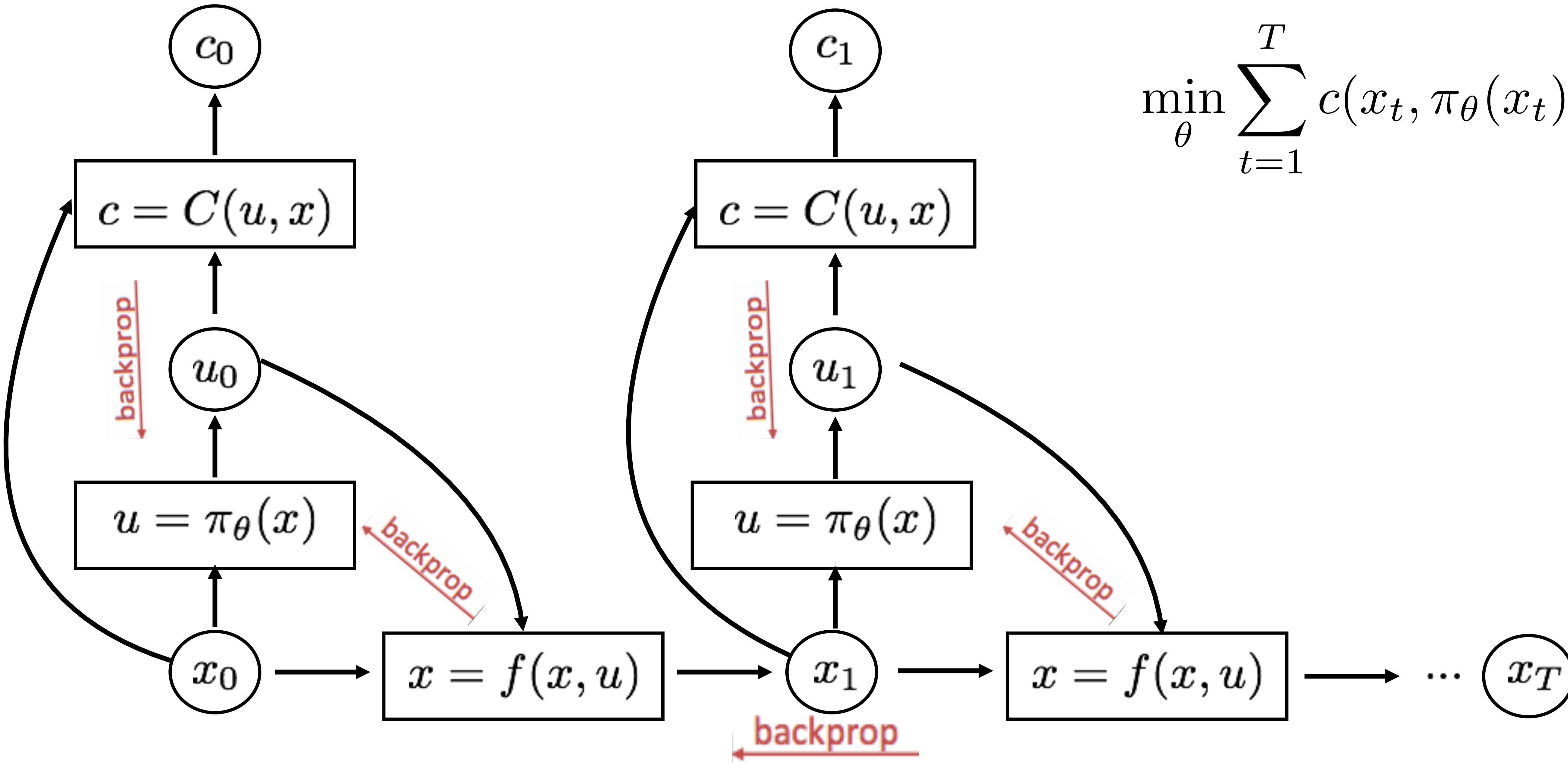
Learning Control Policies through Backpropagation



$$\min_\theta \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))$$

- Dynamics are unknown.
- Policy is assumed deterministic.
- We alternate solving for dynamics parameters ϕ (standard regression) and solving for policy parameters θ using back-propagating (through time).

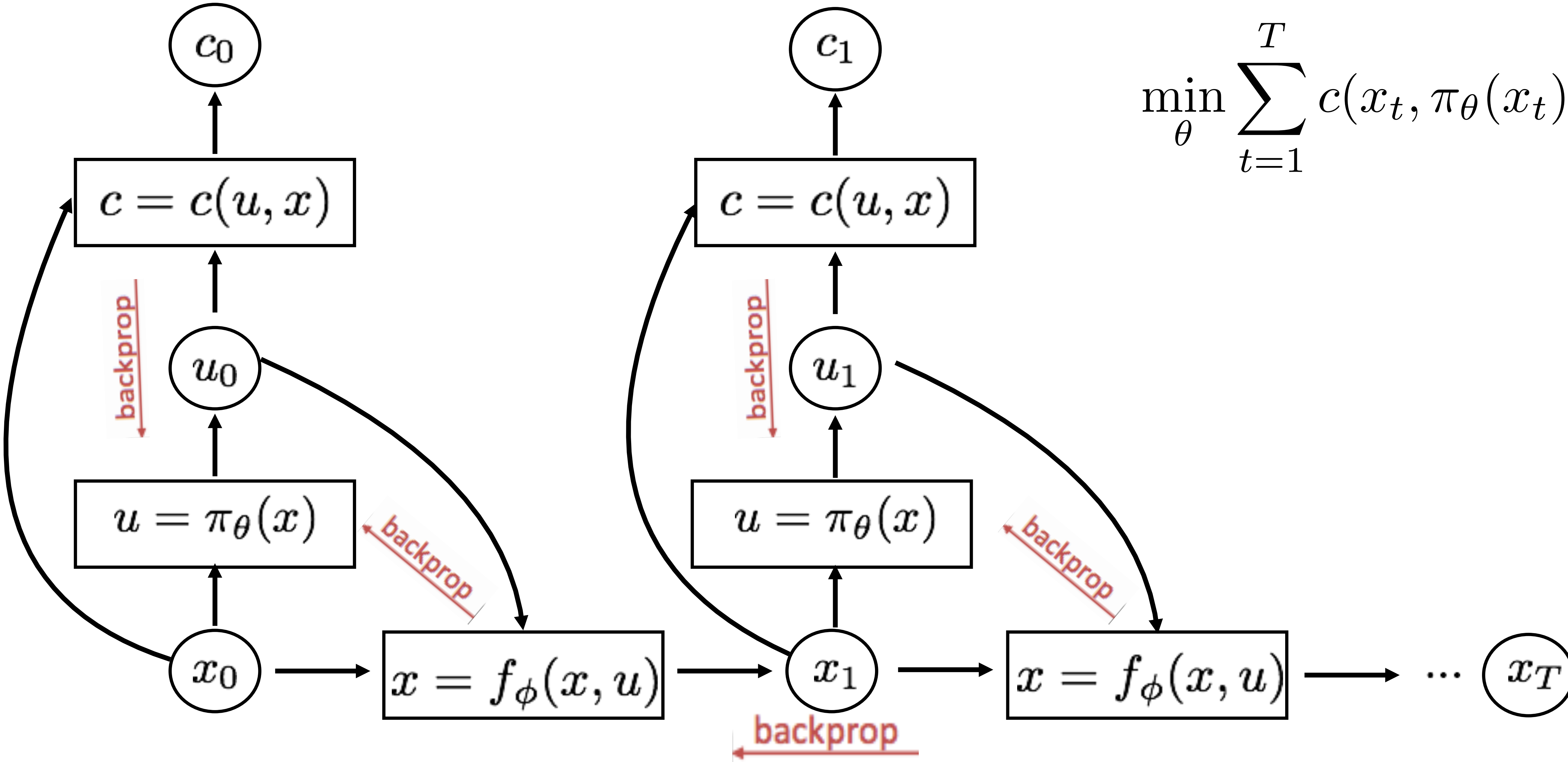
Learning Control Policies through Backpropagation



$$\min_\theta \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))$$

1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$

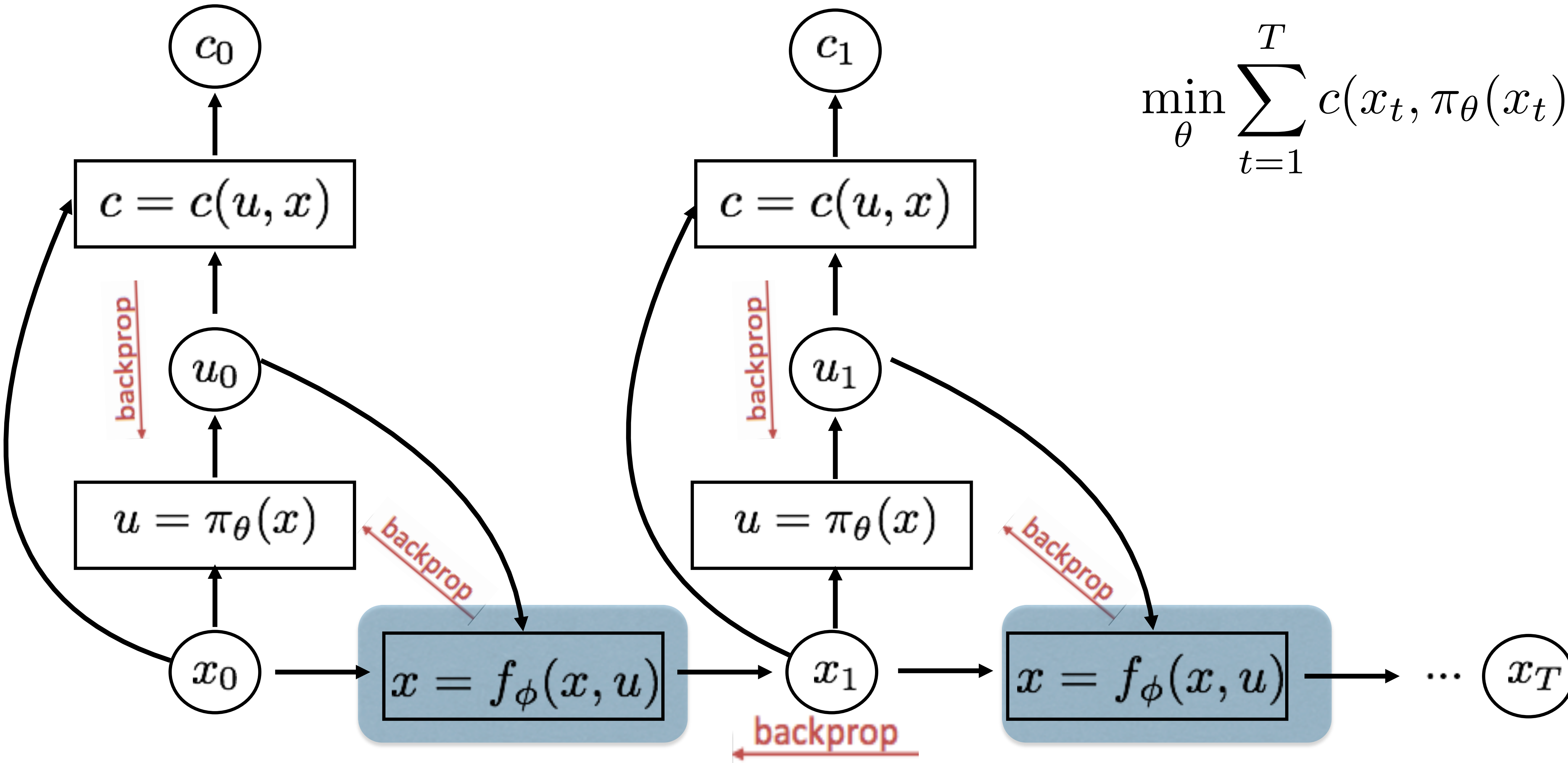
Learning Control Policies through Backpropagation



$$\min_\theta \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))$$

1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $f_\phi(x, u)$ to minimize $\sum_i ||f_\phi(x_i, u_i) - x'_i||^2$

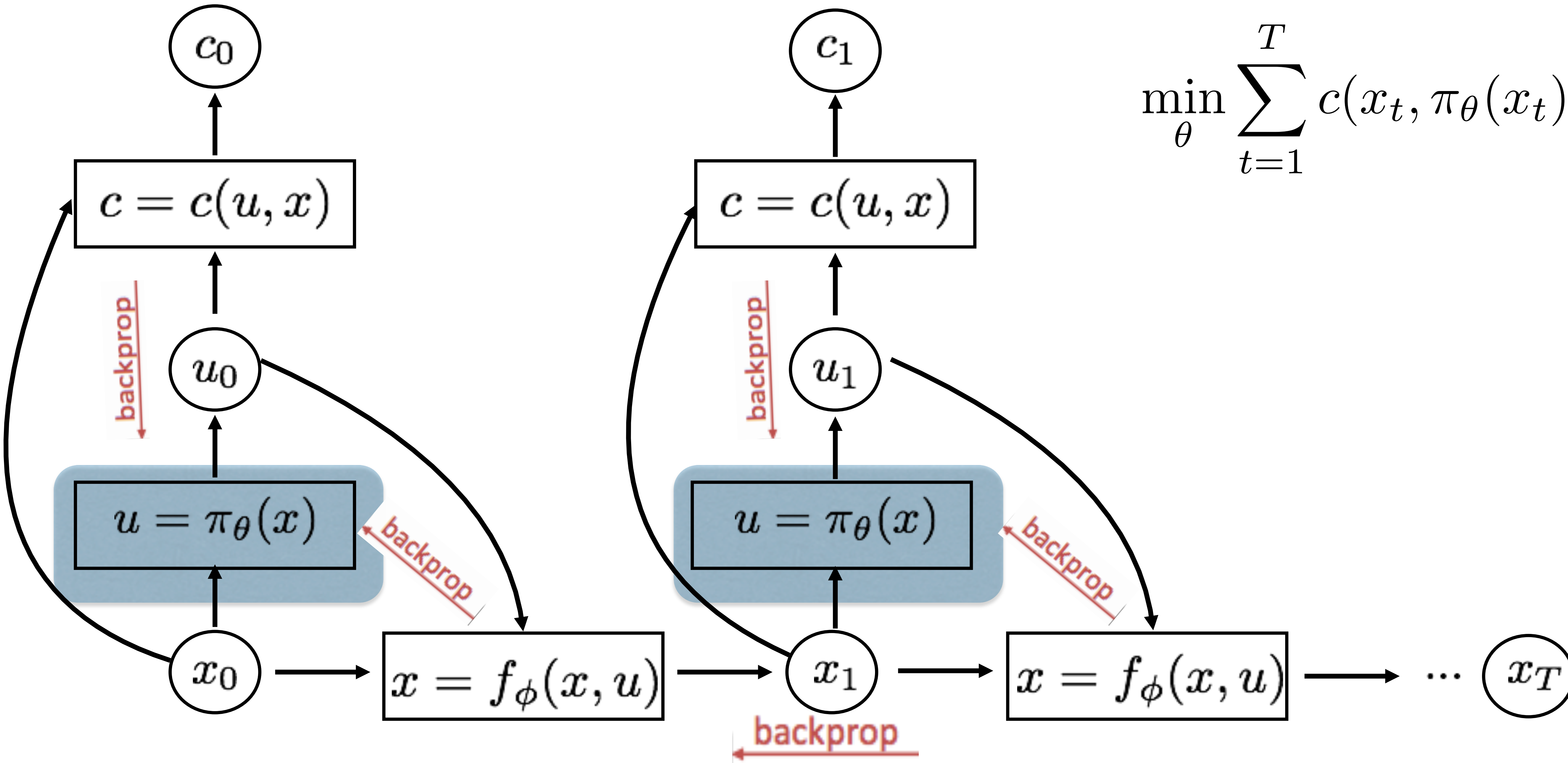
Learning Control Policies through Backpropagation



$$\min_\theta \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))$$

1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $f_\phi(x, u)$ to minimize $\sum_i ||f_\phi(x_i, u_i) - x_i'||^2$

Learning Control Policies through Backpropagation

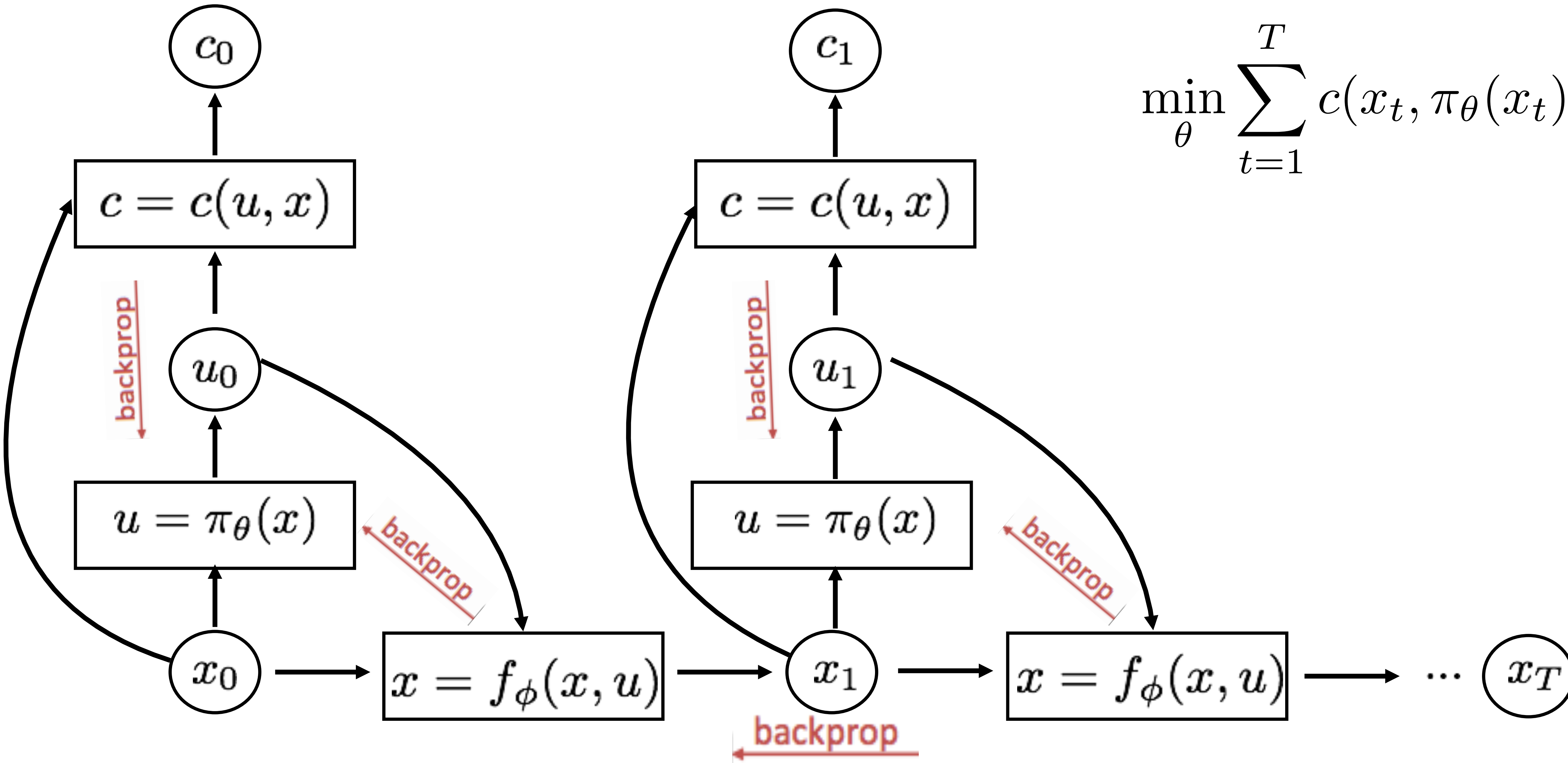


$$\min_\theta \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))$$

1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $\boxed{f_\phi(x, u)}$ to minimize $\boxed{\sum_i ||f_\phi(x_i, u_i) - x_i'||^2}$
3. backpropagate through $\boxed{f_\phi(x, u)}$ into the policy to optimize $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$

while dynamics are frozen

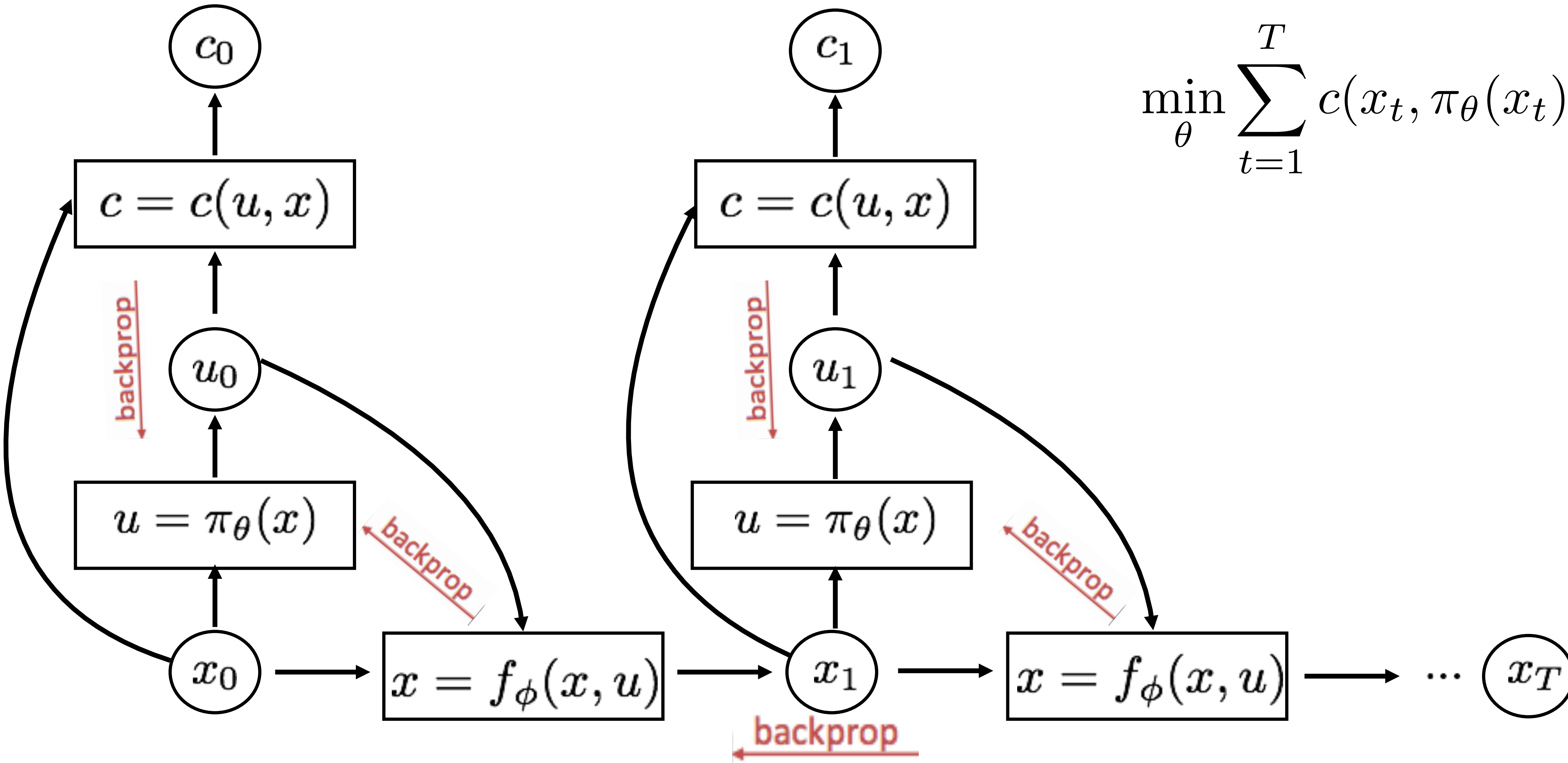
Learning Control Policies through Backpropagation



$$\min_\theta \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))$$

1. run base policy $\pi_0(\mathbf{u}_t|\mathbf{x}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')_i\}$
2. learn dynamics model $f_\phi(x, u)$ to minimize $\sum_i ||f_\phi(x_i, u_i) - x'_i||^2$
3. backpropagate through $f_\phi(x, u)$ into the policy to optimize $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$
4. run $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, appending the visited tuples $(\mathbf{x}, \mathbf{u}, \mathbf{x}')$ to $\mathcal{D}$

Learning Control Policies through Backpropagation



$$\min_\theta \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))$$

Challenges:

- Poor conditioning
- $\boldsymbol{\theta}$ couples actions across all steps -> no DP
- Chaining inaccurate dynamics naturally leads to errors

Learning Control Policies through Imitation

$$\boldsymbol{\pi}_\theta \;:\; \mathbf{x} \;\mapsto\; \mathbf{u}$$

Learning from Demonstrations:

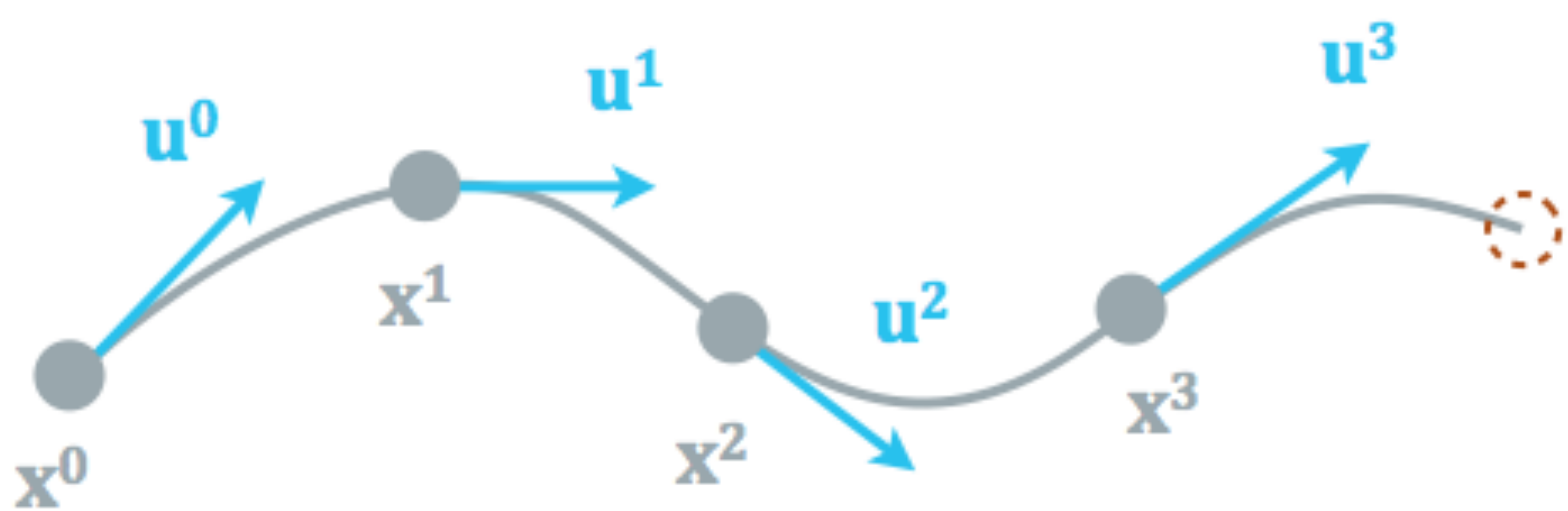
supervised learning

$$\min_\theta \sum_i ||\boldsymbol{\pi}_\theta(\mathbf{x}^i) - \mathbf{u}^i||^2$$

Training Data

input: $\mathbf{x}^i$

output: $\mathbf{u}^i$



Learning Control Policies through Imitation

$$\boldsymbol{\pi}_\theta \; : \; \mathbf{x} \mapsto \mathbf{u}$$

Learning from Demonstrations:

supervised learning

$$\min_\theta \sum_i ||\boldsymbol{\pi}_\theta(\mathbf{x}^i) - \mathbf{u}^i||^2$$

Training Data

input: $\mathbf{x}^i$

output: $\mathbf{u}^i$

Where does training data come from?

Optimal controllers trained with trajectory optimization

Learning Control Policies through Imitation

- Joint trajectory and policy optimization (last week):

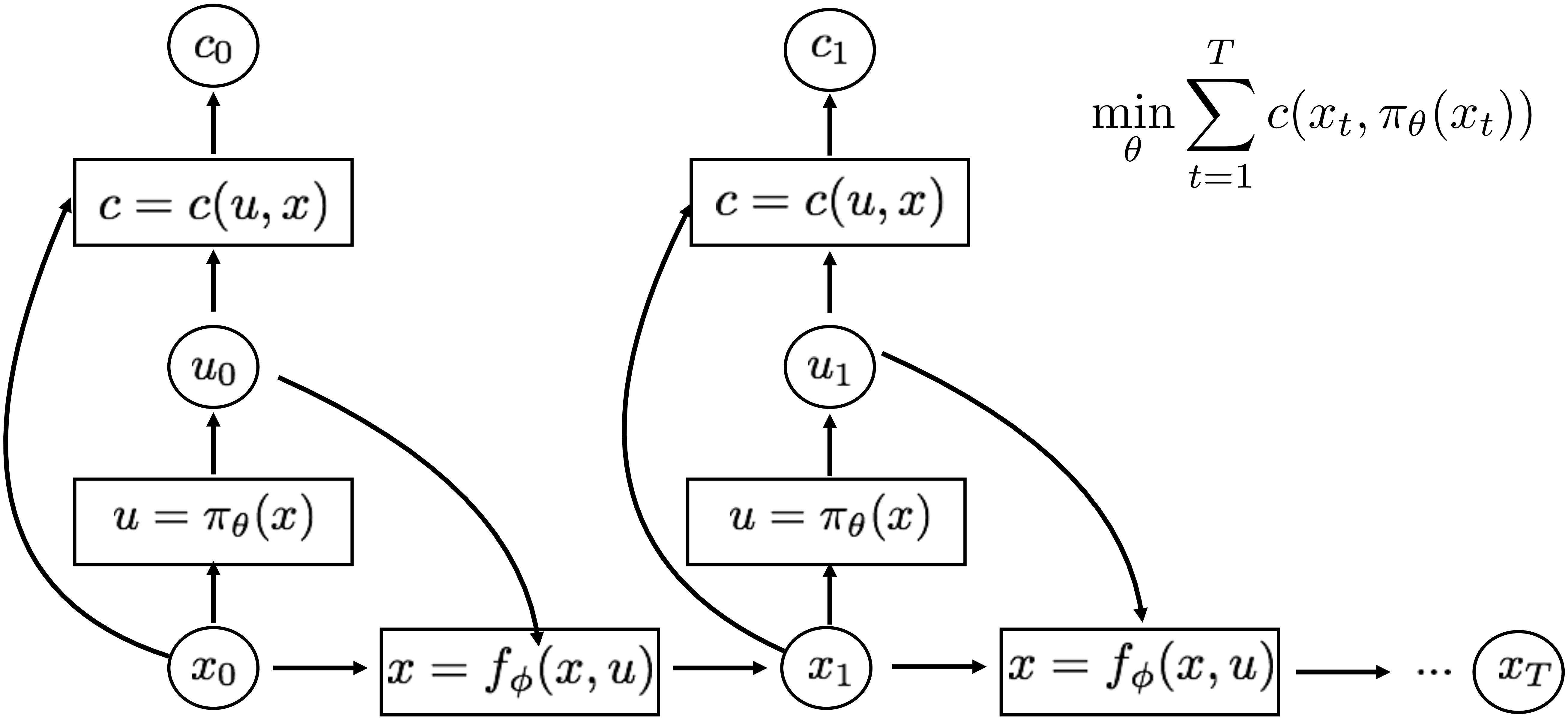
$$\min_{\mathbf{u}_1,\ldots,\mathbf{u}_T,\mathbf{x}_1,\ldots,\mathbf{x}_T,\theta} \sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_t) \text{ s.t. } \mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})$$

$$\text{s.t. } \mathbf{u}_t = \pi_\theta(\mathbf{x}_t)$$

Guided policy search, Levine and Koltun 2013

Combining the benefits of function approximation and trajectory optimization, Mordatch and Todorov 2014

Learning Control Policies through Backpropagation



$$\min_\theta \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))$$

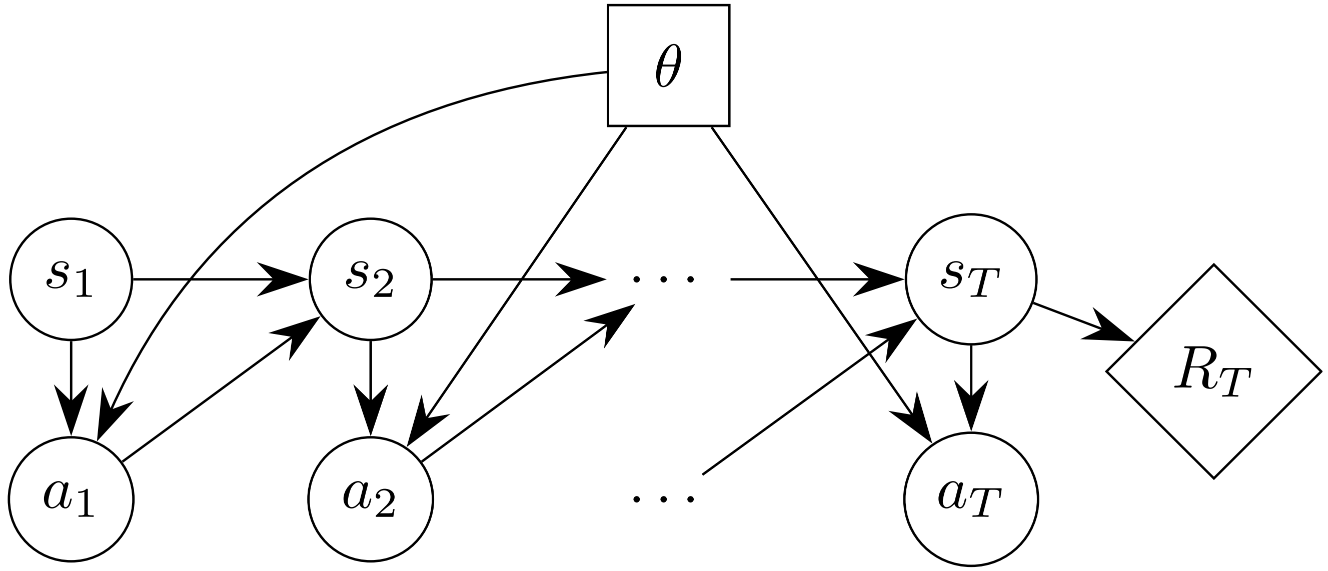
Challenges:

- Backproping through stochastic policies and stochastic environments
- Avoiding error accumulation through chaining of one step dynamics

This Lecture

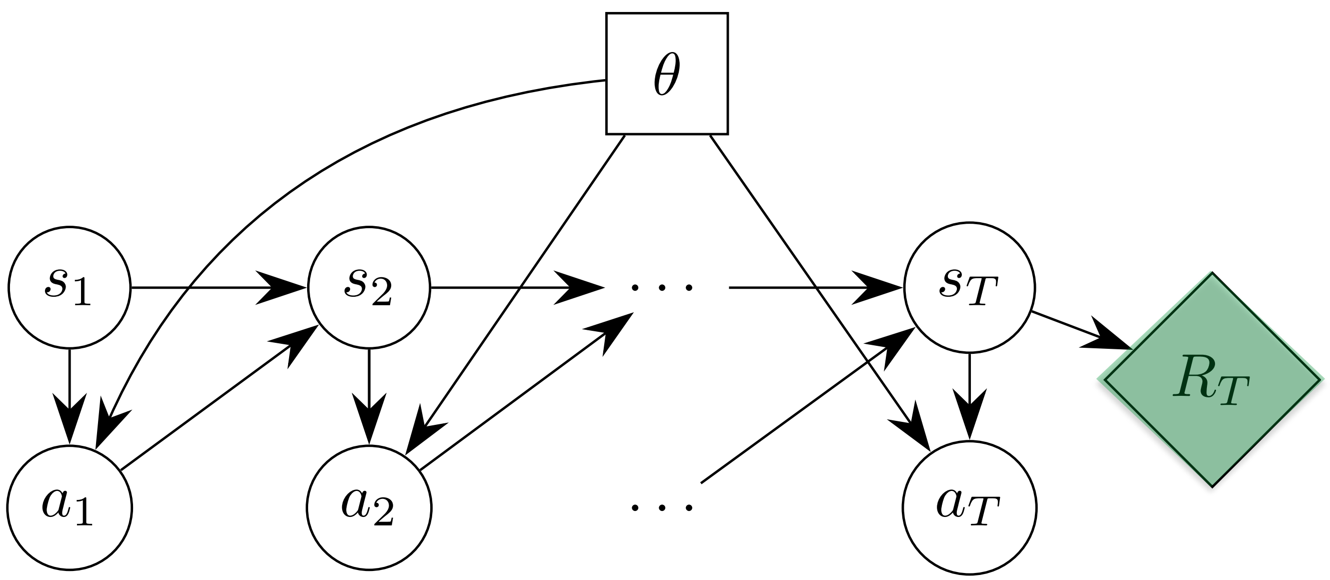
- Backproping through stochastic policies and stochastic environments
    - Re-parametrization trick
- Avoiding error accumulation through chaining of one step dynamics
    - Use function approximation for action and state value functions to predict future returns so that you do not rely on your model for long chaining

Policy optimization

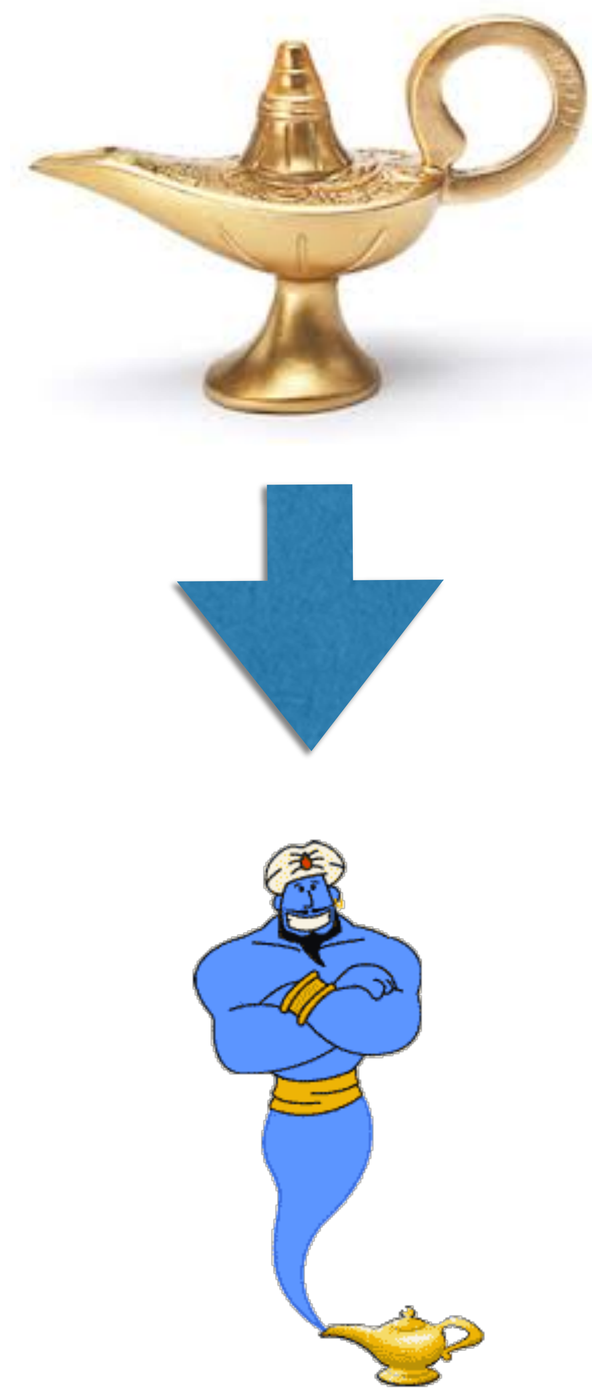


- You get a reward when the jeannie appears

Policy optimization



- You get a reward when the jeannie appears



Policy Optimization

$$\underset{\pi}{\text{maximize}}\, \mathbb{E}_{\pi}\left[\textit{expression}\right]$$

Fixed-horizon episodic: $\sum_{t=0}^{T-1} r_t$

Average-cost: $\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} r_t$

Infinite-horizon discounted: $\sum_{t=0}^{\infty} \gamma^t r_t$

Variable-length undiscounted: $\sum_{t=0}^{T_{\text{terminal}}-1} r_t$

Infinite-horizon undiscounted: $\sum_{t=0}^{\infty} r_t$

Episodic Settings

$$
\begin{aligned}
s_0 &\sim \mu(s_0) \\
a_0 &\sim \pi(a_0 \mid s_0) \\
s_1, r_0 &\sim P(s_1, r_0 \mid s_0, a_0) \\
a_1 &\sim \pi(a_1 \mid s_1) \\
s_2, r_1 &\sim P(s_2, r_1 \mid s_1, a_1) \\
&\ldots \\
a_{T-1} &\sim \pi(a_{T-1} \mid s_{T-1}) \\
s_T, r_{T-1} &\sim P(s_T \mid s_{T-1}, a_{T-1})
\end{aligned}
$$

Objective:

$$
\begin{aligned}
&\text{maximize}\, \eta(\pi), \ \text{where} \\
&\quad \eta(\pi) = E[r_0 + r_1 + \cdots + r_{T-1} \mid \pi]
\end{aligned}
$$

τ: trajectory, a sequence of action states

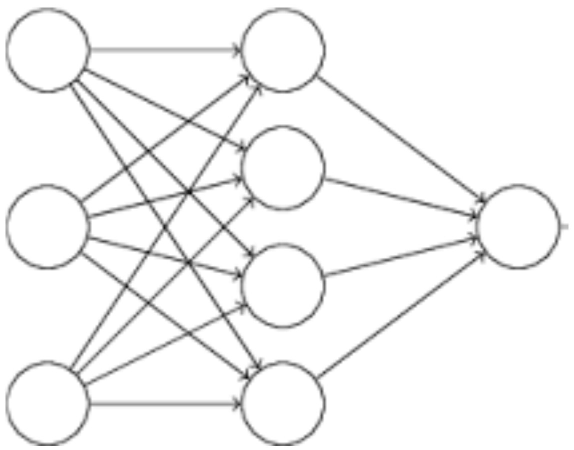
Parameterized Policies

- A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$
    - Deterministic: $a = \pi(s, \theta)$
    - Stochastic: $\pi(a|s, \theta)$
- Analogous to classification or regression with input s, output a
    - Discrete action space: network outputs vector of probabilities
    - Continuous actions space: network outputs mean and diagonal covariance of Gaussian

Parametrized policies

deterministic policy

s



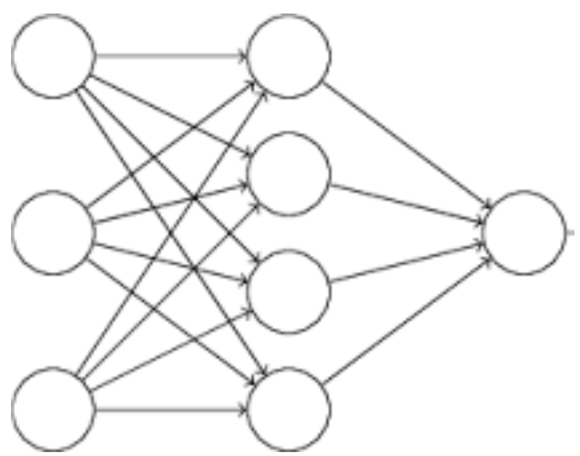
a

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

Parametrized policies

deterministic policy

s

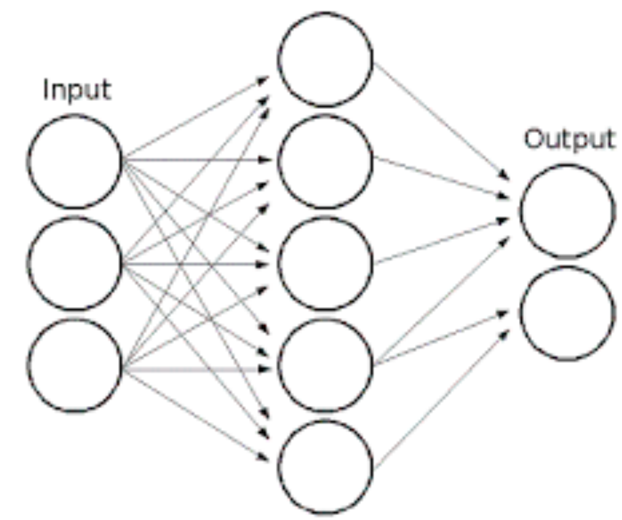


a

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

stochastic continuous policy:
usually unimodal gaussian

s



$\mu_\theta(s)$

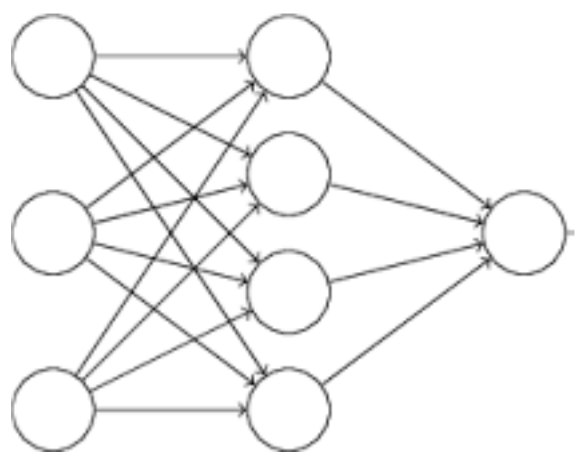
$\sigma_\theta(s)$

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

Parametrized policies

deterministic policy

s

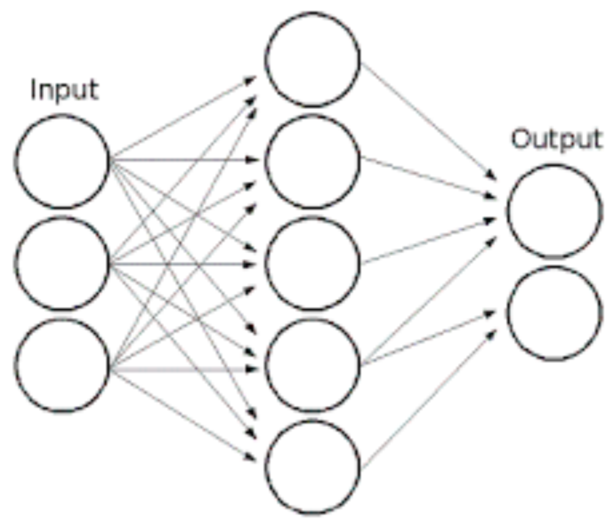


a

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

stochastic continuous policy:
usually unimodal gaussian

s



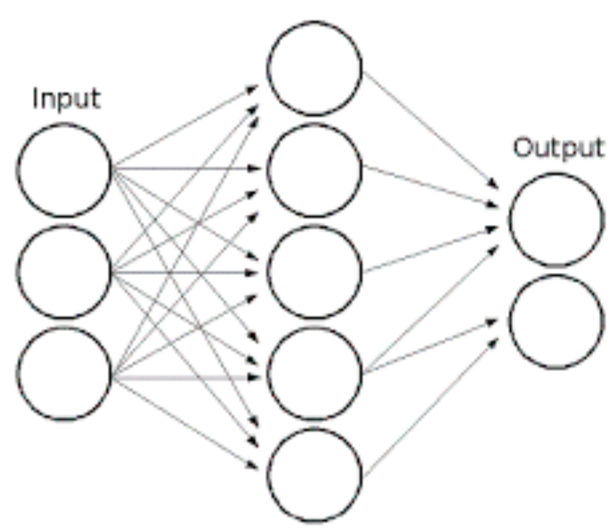
$\mu_\theta(s)$

$\sigma_\theta(s)$

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

discrete action space

s



go left

go right

How do we compute gradients?

$$\nabla_\theta \mathbb{E}\left[R_T\right]$$

- Numerically: finite differencing

How do we compute gradients?

$$\nabla_\theta \mathbb{E}\left[R_T\right]$$

- Numerically: finite differencing
- Score function gradient estimator (a.k.a. likelihood ratio gradient estimator)

Likelihood Ratio Policy Gradient

$$J(\theta) = \sum_{\tau} P[t;\theta]R(\tau)$$

Taking the gradient w.r.t. $\boldsymbol{\theta}$ gives

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} \sum_{\tau} P[\tau;\theta]R(\tau)$$

Likelihood Ratio Policy Gradient

$$J(\theta) = \sum_{\tau} P[t;\theta]R(\tau)$$

Taking the gradient w.r.t. $\boldsymbol{\theta}$ gives

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_{\tau} P[\tau;\theta]R(\tau) \\
&= \sum_{\tau} \nabla_\theta P[\tau;\theta]R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P[\tau;\theta]R(\tau)
\end{aligned}$$

Likelihood Ratio Policy Gradient

$$J(\theta) = \sum_{\tau} P[t;\theta]R(\tau)$$

Taking the gradient w.r.t. $\boldsymbol{\theta}$ gives

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_{\tau} P[\tau;\theta]R(\tau) \\
&= \sum_{\tau} \nabla_\theta P[\tau;\theta]R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P[\tau;\theta]R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P[\tau;\theta]} R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}$$

Likelihood Ratio Policy Gradient

$$J(\theta) = \sum_{\tau} P[t;\theta]R(\tau)$$

Taking the gradient w.r.t. $\boldsymbol{\theta}$ gives

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \sum_{\tau} P[\tau;\theta]R(\tau) \\
&= \sum_{\tau} \nabla_\theta P[\tau;\theta]R(\tau) \\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P[\tau;\theta]R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P[\tau;\theta]} R(\tau) \\
&= \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau)
\end{aligned}$$

Approximate with the empirical estimate for m sample paths under policy

$$\nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

Decompose Path into States and Actions

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{policy}} \right]$$

Decompose Path into States and Actions

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{policy}} \right]$$

$$= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right]$$

Decompose Path into States and Actions

$$\begin{aligned}
\nabla_\theta \log P(\tau^{(i)};\theta) &= \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right] \\
&= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right] \\
&= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)})
\end{aligned}$$

Decompose Path into States and Actions

$$
\begin{aligned}
\nabla_\theta \log P(\tau^{(i)}; \theta) &= \nabla_\theta \log \left[\prod_{t=0}^{H} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right] \\
&= \nabla_\theta \left[\sum_{t=0}^{H} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right] \\
&= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \\
&= \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{no dynamics model required!!}}
\end{aligned}
$$

Gaussian Policy

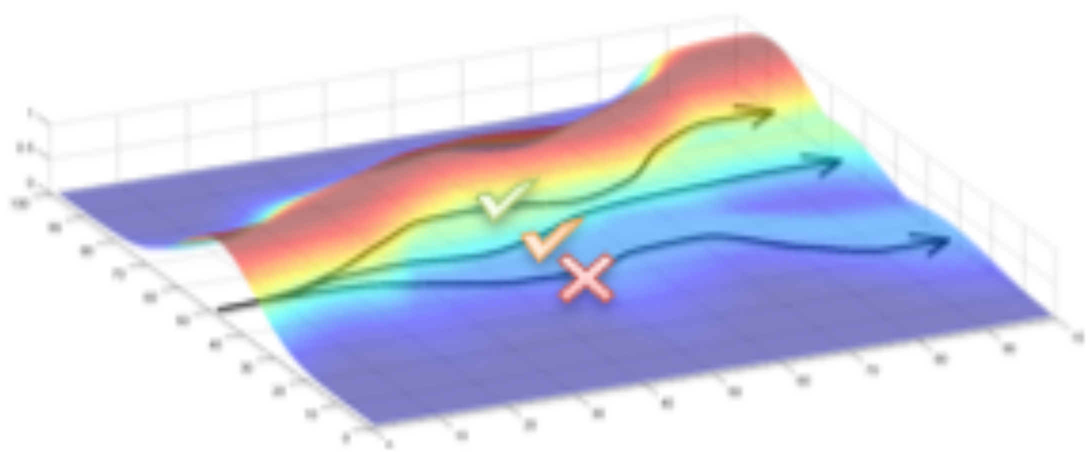
- Variance may be fixed $\boldsymbol{\sigma^2}$, or can also be parametrized
- Policy is Gaussian, $a \sim \mathcal{N}(\mu(s;\theta), \sigma^2)$
- The score function is

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

Likelihood Ratio Gradient: Intuition

$$\nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

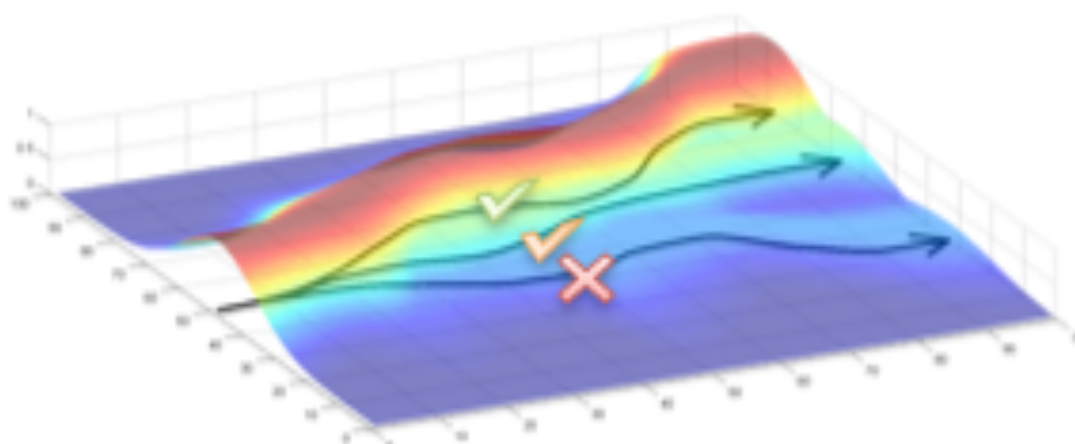
- Gradient tries to:
  - Increase probability of paths with positive R
  - Decrease probability of paths with negative R



Likelihood Ratio Gradient: Intuition

$$\nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)})$$

- Gradient tries to:
  - Increase probability of paths with positive R
  - Decrease probability of paths with negative R



- The reward function is a black box and dynamics are not used anywhere
- The world is a black box.

Likelihood Ratio Gradient Estimate

$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)})$$

Here:

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \sum_{t=0}^{H} \underbrace{\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})}_{\text{no dynamics model required!!}}$$

Unbiased means:

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

Reduce Variance using a Critic

- A critic provides an estimate of the expectation of the future reward as opposed to a single return sample.

- Use a function approximator for Q function or the advantage function.

$$
\begin{aligned}
\nabla_\theta \mathbb{E}_\tau [R] &= \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) Q^\pi(s_t, a_t) \right] \\
&= \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) A^\pi(s_t, a_t) \right]
\end{aligned}
$$

Reduce Variance using a Critic

Q-function is state-action-value function:

$$Q^{\pi,\gamma}(s,a) = \mathbb{E}_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s, a_0 = a\right]$$

State-value function:

$$\begin{aligned} V^{\pi,\gamma}(s) &= \mathbb{E}_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s\right] \\ &= \mathbb{E}_{a\sim\pi}\left[Q^{\pi,\gamma}(s,a)\right] \end{aligned}$$

Advantage function:

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

Q Actor-Critic

function QAC

Initialise s, θ

Sample $a \sim \pi_\theta$

for each step **do**

Sample reward $r = \mathcal{R}_s^a$; sample transition $s' \sim \mathcal{P}_{s,\cdot}^a$

Sample action $a' \sim \pi_\theta(s', a')$

$\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$

$\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$

$w \leftarrow w + \beta \delta \phi(s, a)$

$a \leftarrow a', s \leftarrow s'$

end for

end function

How do we compute gradients?

$$\nabla_\theta \mathbb{E}\left[R_T\right]$$

- Numerically: finite differencing
- Score function estimator
    - Problems: high variance! In particular, as the policy becomes more and more deterministic, the variance explodes.

Variance in the Gaussian case

$$x \sim \mathcal{N}(\mu(\theta), \sigma(\theta))$$

$$g = \nabla_\theta \log p_\theta(x) f(x)$$

$$= \frac{(x - \mu(\theta))\mu'(\theta)}{\sigma^2} f(x)$$

Variance in the Gaussian case

$$x \sim \mathcal{N}(\mu(\theta), \sigma(\theta))$$

$$g = \nabla_\theta \log p_\theta(x) f(x)$$

$$= \frac{(x - \mu(\theta))\mu'(\theta)}{\sigma^2} f(x)$$

Sample $x : x = \mu(\theta) + z\sigma, z \sim \mathcal{N}(0,1)$

$$\hat{g} = \frac{z\sigma\mu'(\theta)}{\sigma^2} f(\mu(\theta) + z\sigma)$$

$$= \frac{z\mu'(\theta)}{\sigma} f(\mu(\theta) + z\sigma)$$

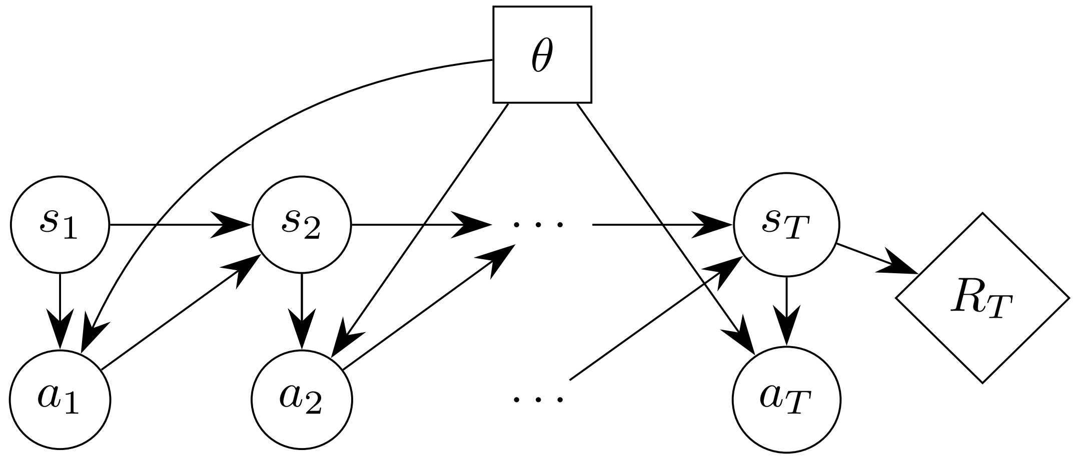
$$\mathbb{V}(\hat{g}) = \mathbb{E}[\hat{g} - g]^2 = \mathbb{E}\Big[\frac{z\mu'(\theta)}{\sigma} f(\mu(\theta) + z\sigma) - g\Big]^2$$

How do we compute gradients?

$$\nabla_\theta \mathbb{E}\left[R_T\right]$$

- Numerically: finite differencing
- Score function estimator
- Deep deterministic police gradients: giving up stochastic policies

Deep Deterministic Police Gradients

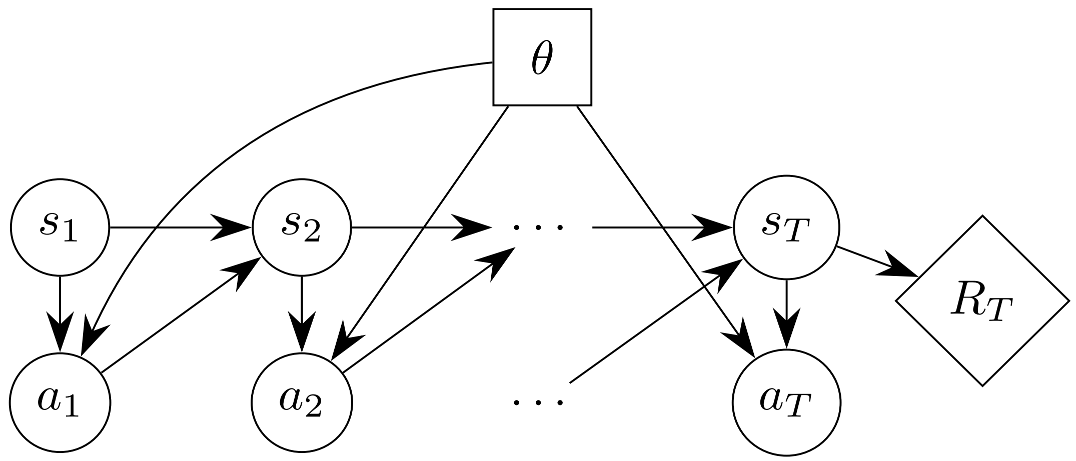


R_τ: the return of a trajectory

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

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

Deep Deterministic Police Gradients



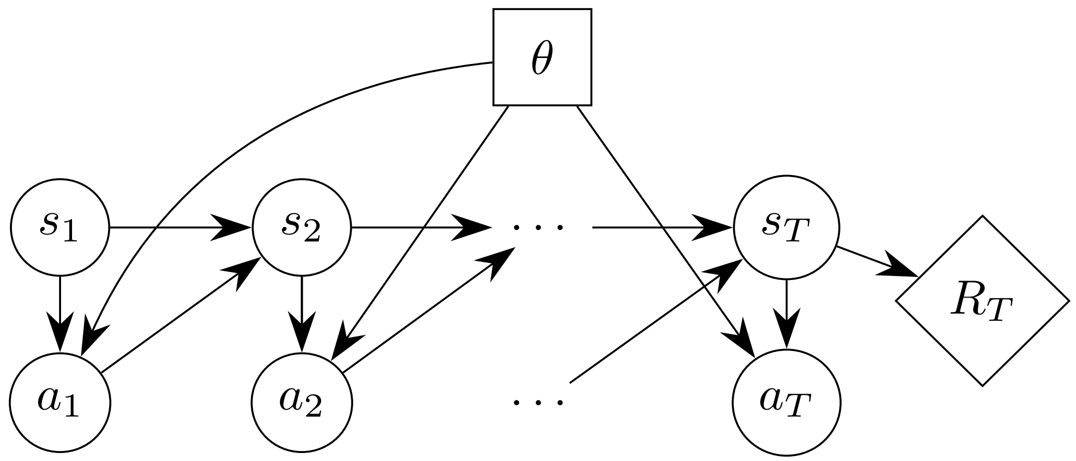
R_τ: the return of a trajectory

This expectation refers to the actions 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 Police Gradients



R_τ: the return of a trajectory

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

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

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

Remember: Q learning

- Definition

$$Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i>t} \sim E, a_{i>t} \sim \pi} \left[R_t | s_t, a_t \right]$$

Remember: Q learning

- Definition

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i > t} \sim E, a_{i > t} \sim \pi} \left[R_t | s_t, a_t \right]$$

- Bellman equation

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} \left[r(s_t, a_t) + \gamma \, \mathbb{E}_{a_{t+1} \sim \pi} \left[Q^{\pi}(s_{t+1}, a_{t+1}) \right] \right]$$

- Using a deterministic policy

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} \left[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1})) \right]$$

Remember: Q learning

- Definition

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i > t} \sim E, a_{i > t} \sim \pi} \left[R_t | s_t, a_t \right]$$

- Bellman equation

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} \left[r(s_t, a_t) + \gamma \, \mathbb{E}_{a_{t+1} \sim \pi} \left[Q^{\pi}(s_{t+1}, a_{t+1}) \right] \right]$$

- Using a deterministic policy

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} \left[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1})) \right]$$

- Deep Q learning:

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^{\beta}, a_t \sim \beta, r_t \sim E} \left[\left(Q(s_t, a_t | \theta^Q) - y_t \right)^2 \right]$$

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}) | \theta^Q)$$

$$\mu(s) \;=\; \arg\max{}_a \, Q(s, a)$$

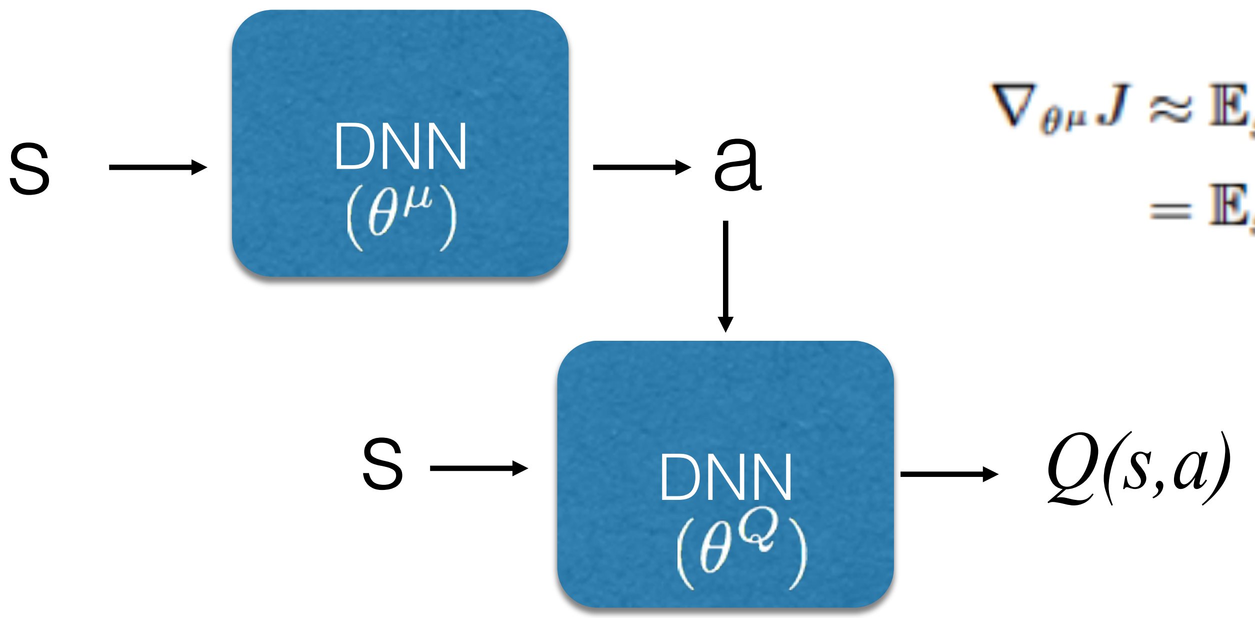
Q learning in continuous action space

- This optimization takes too long for continuous action spaces and has to be performed at every iteration:

$$\mu(s) = \arg\max_a Q(s, a)$$

Q learning in continuous action space

- Instead of parametrizing Q let's also parametrize the policy $a = \mu(\theta)$



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

How do we compute gradients?

$$\nabla_\theta \mathbb{E}\left[R_T\right]$$

- Numerically: finite differencing
- Score function estimator
- Deep deterministic police gradients: giving up stochastic policies
- Pathwise derivatives

Pathwise derivatives for Gaussian samples

- Consider normally distributed variable y

$$p(y|x) = \mathcal{N}(y|\mu(x), \sigma^2(x))$$

Pathwise derivatives for Gaussian samples

- Consider normally distributed variable y

$$p(y|x) = \mathcal{N}(y|\mu(x), \sigma^2(x))$$

- Reparametrization:

$$y = \mu(x) + \sigma(x)\xi, \text{ where } \xi \sim \mathcal{N}(0, 1)$$

Pathwise derivatives for Gaussian samples

- Consider normally distributed variable y

$$p(y|x) = \mathcal{N}(y|\mu(x), \sigma^2(x))$$

- Reparametrization:

$$y = \mu(x) + \sigma(x)\xi, \text{ where } \xi \sim \mathcal{N}(0,1)$$

- Sampling: Sample ξ and then deterministically generate $\mathbf{y} = \mathbf{f}(\mathbf{x}, \xi)$

$$\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}\mathbf{g}(\mathbf{y}) = \int \mathbf{g}(\mathbf{f}(\mathbf{x},\xi))\rho(\xi)d\xi$$

$$\nabla_{\mathbf{x}}\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}\mathbf{g}(\mathbf{y}) = \mathbb{E}_{\rho(\xi)}\mathbf{g}_{\mathbf{y}}\mathbf{f}_{\mathbf{x}} \approx \boxed{\frac{1}{M}\sum_{i=1}^{M}\mathbf{g}_{\mathbf{y}}\mathbf{f}_{\mathbf{x}}\big|_{\xi=\xi_i}}$$

Pathwise derivatives for Gaussian samples

- Consider normally distributed variable y

$$p(y|x) = \mathcal{N}(y|\mu(x), \sigma^2(x))$$

- Reparametrization:

$$y = \mu(x) + \sigma(x)\xi, \text{ where } \xi \sim \mathcal{N}(0,1)$$

- Sampling: Sample ξ and then deterministically generate $\mathbf{y} = \mathbf{f}(\mathbf{x}, \xi)$

$$\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}\mathbf{g}(\mathbf{y}) = \int \mathbf{g}(\mathbf{f}(\mathbf{x}, \xi))\rho(\xi)d\xi$$

$$\nabla_{\mathbf{x}}\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}\mathbf{g}(\mathbf{y}) = \mathbb{E}_{\rho(\xi)}\mathbf{g}_{\mathbf{y}}\mathbf{f}_{\mathbf{x}} \approx \boxed{\frac{1}{M}\sum_{i=1}^{M} \mathbf{g}_{\mathbf{y}}\mathbf{f}_{\mathbf{x}}\big|_{\xi=\xi_i}}$$

- Compare to the score function gradient estimator: $\displaystyle \frac{1}{M}\sum_{i=1}^{M} \nabla_{\mathbf{x}} \log(p(\mathbf{y}|\mathbf{x}))\mathbf{g}(\mathbf{y})$

Pathwise derivatives for Gaussian samples

Sampling: sample ξ and then deterministically generate y: $\mathbf{y} = \mathbf{f}(\mathbf{x}, \xi)$

$$\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}\mathbf{g}(\mathbf{y}) = \int \mathbf{g}(\mathbf{f}(\mathbf{x}, \xi))\rho(\xi)d\xi$$

$$\nabla_{\mathbf{x}}\mathbb{E}_{p(\mathbf{y}|\mathbf{x})}\mathbf{g}(\mathbf{y}) = \mathbb{E}_{\rho(\xi)}\mathbf{g}_{\mathbf{y}}\mathbf{f}_{\mathbf{x}} \approx \boxed{\frac{1}{M}\sum_{i=1}^{M}\mathbf{g}_{\mathbf{y}}\mathbf{f}_{\mathbf{x}}\big|_{\xi=\xi_i}}$$

Compare to the score function gradient estimator: $\frac{1}{M}\sum_{i=1}^{M}\nabla_{\mathbf{x}}\log(p(\mathbf{y}|\mathbf{x}))\mathbf{g}(\mathbf{y})$

- The pathwise derivative makes use of the gradient of g.
- Of course, that assumes we know the function g (our reward function) and how it is related to our actions.

Pathwise derivative for Gaussian Policies

- Gaussian Policies:

$$a = \mu(s,\theta) + z^{*}\sigma(s,\theta)$$

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

$$\nabla_\theta \mathbb{E}_z(R(a(\theta,z)))$$

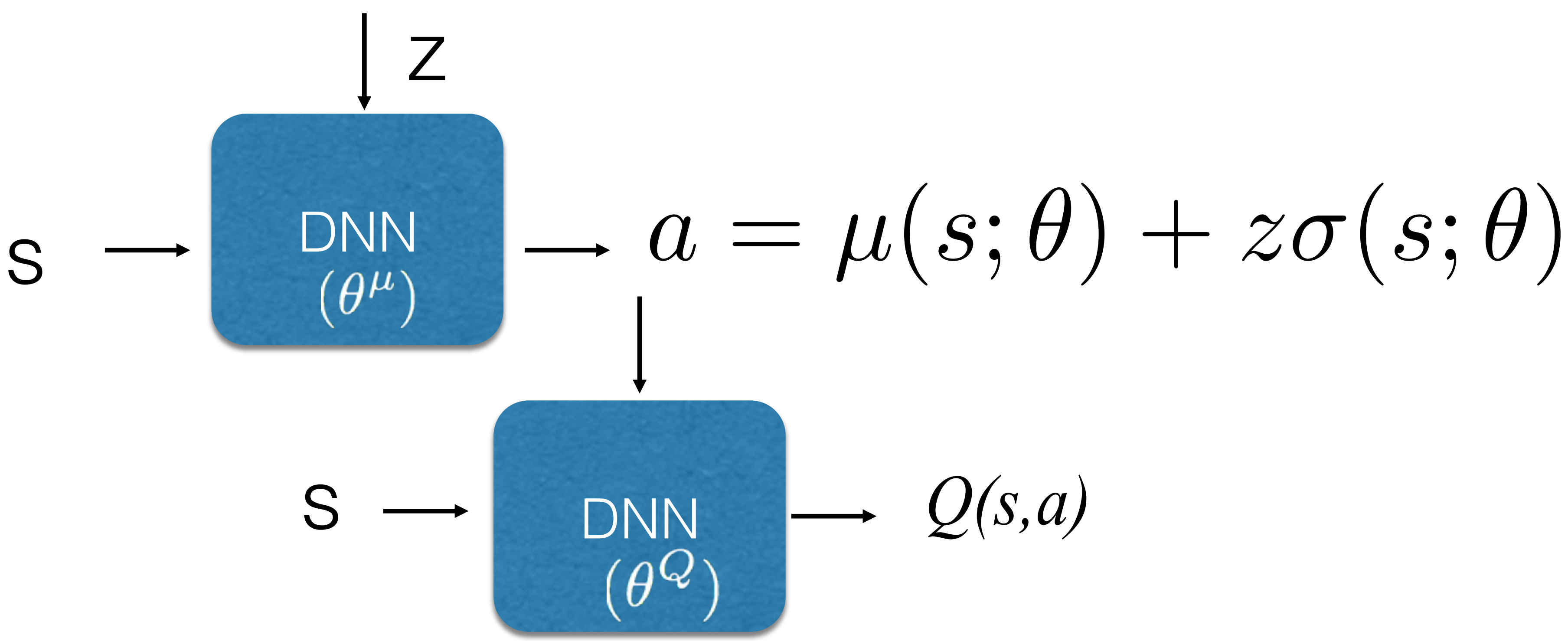
$$\mathbb{E}_z(R'(a(\theta,z))\frac{da(\theta,z)}{d\theta})$$

- R should be **known and differentiable**
- To propagate for more than 1 step, dynamics should be known and differentiable

Q learning in continuous stochastic action space

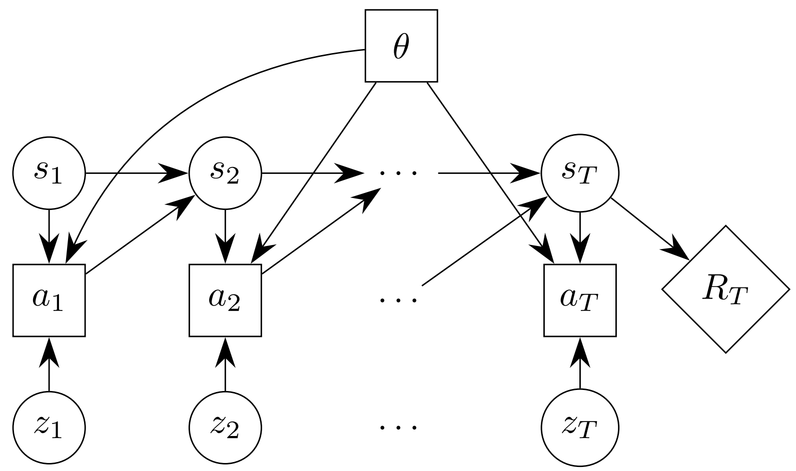
- Now use stochastic policies using the reparametrization trick!

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



Learning Continuous Control Policies by Stochastic Value Gradients, Heess et al. 2016

Q learning in continuous stochastic action space



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

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

Learning Continuous Control Policies by Stochastic Value Gradients, Heess et al. 2016

Q learning in continuous stochastic action space

SVG(0)

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 $g \propto \nabla_\theta \sum_{t=1}^{T} Q(s_t, \pi(s_t, z_t; \theta))$

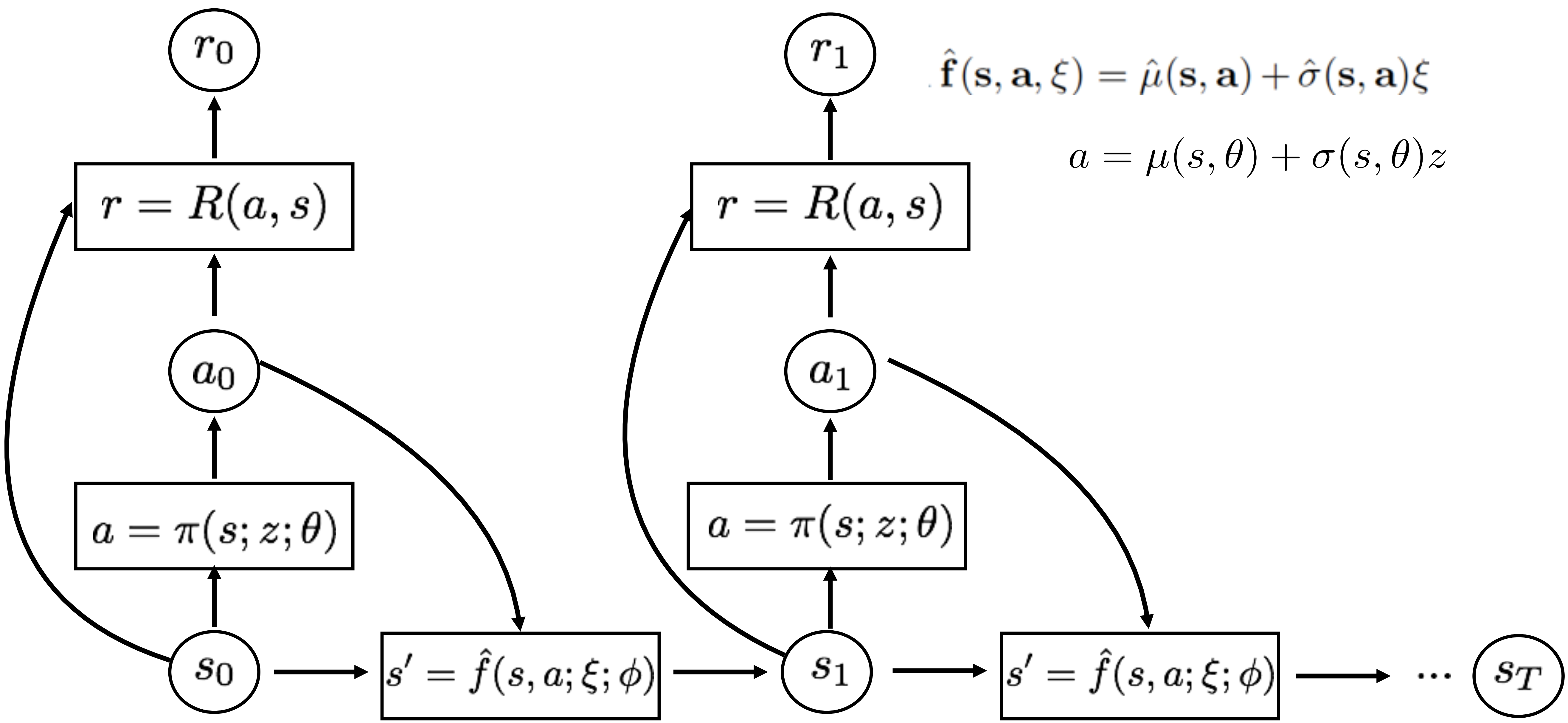
Update Q_ϕ using $g \propto \nabla_\phi \sum_{t=1}^{T} (Q_\phi(s_t, a_t) - \hat{Q}_t)^2$, e.g. with TD($\lambda$)

end for

Learning Continuous Control Policies by Stochastic Value Gradients, Heess et al. 2016

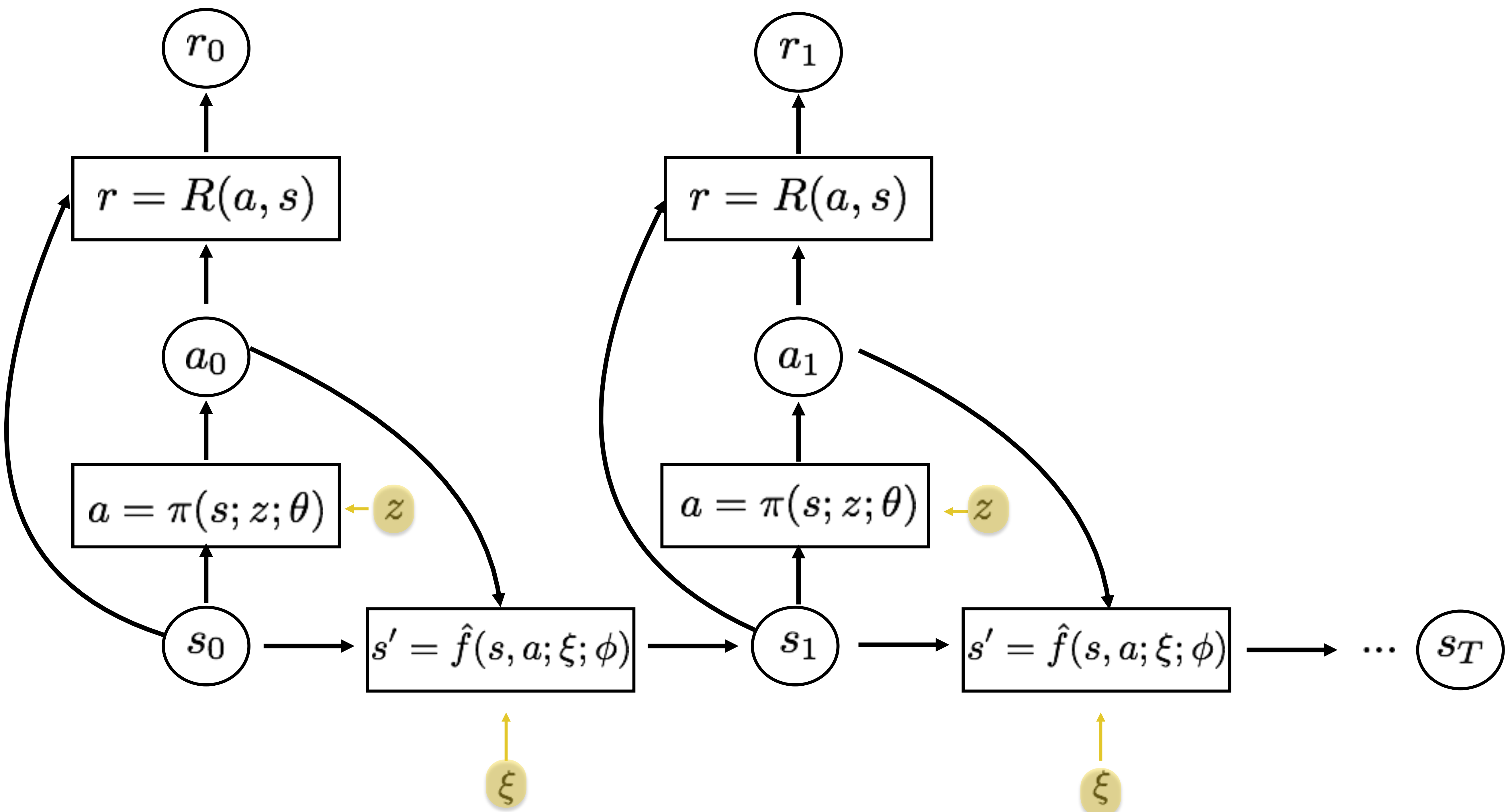
End-to-end model based RL

- Reparametrization trick for both policies and dynamics



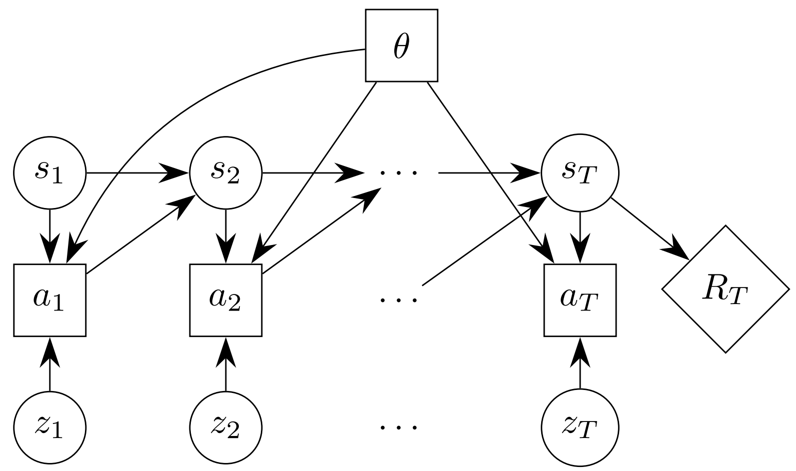
End-to-end model based RL

- Reparametrization trick for both policies and dynamics



End-to-end model based RL

$\mathrm{SVG}(\infty)$



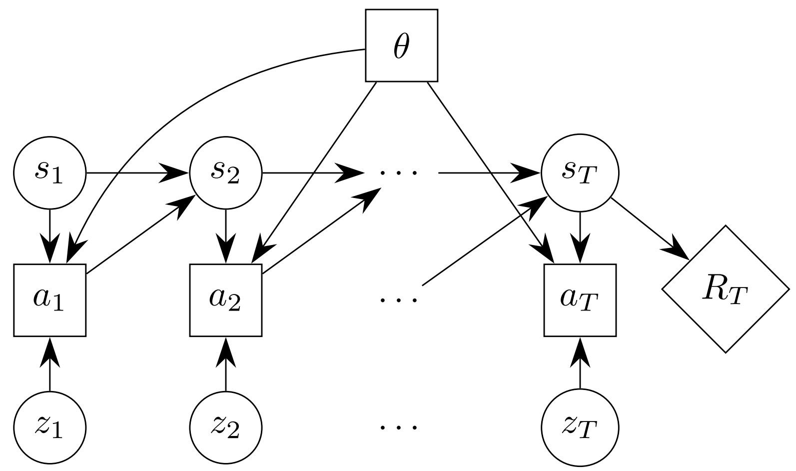
- Just learn dynamics model f
- Given whole trajectory, infer all noise variables
- Given transition (s_t, a_t, s_{t+1}), infer $\zeta_t = s_{t+1} - f(s_t, a_t)$
- Freeze all policy and dynamics noise, differentiate through entire deterministic computation graph

R should be **known and differentiable**

Learning Continuous Control Policies by Stochastic Value Gradients, Heess et al. 2016

End-to-end model based RL

SVG(1)



- Instead of learning Q, we learn
  - State-value function $V \approx V^{\pi,\gamma}$
  - Dynamics model f, approximating $s_{t+1} = f(s_t, a_t) + \zeta_t$
- Given transition (s_t, a_t, s_{t+1}) infer $\zeta_t = s_{t+1} - f(s_t, a_t)$
- $Q(s_t, a_t) = \mathbb{E}[r_t + \gamma V(s_{t+1})] = \mathbb{E}[r_t + \gamma V(f(s_t, a_t) + \zeta_t)]$

  $a_t = \pi(s_t, \theta, \zeta_t)$

Learning Continuous Control Policies by Stochastic Value Gradients, Heess et al. 2016

Re-parametrization trick for categorical distributions

- Consider variable y following the K categorical distribution:

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

Categorical reparametrization with Gumbel-Softmax Sang et al. 2017

Re-parametrization trick for categorical distributions

- Consider variable y following the K categorical distribution:

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

- Reparametrization:

$$y_k \sim G(\log p) = \frac{\exp((\log p_k + \varepsilon)/\tau)}{\sum_{j=0}^{K} \exp((\log p_j + \varepsilon)/\tau)} \ , \ \ \varepsilon = -\log(-\log(u)), u \sim \mathcal{U}[0,1]$$

- Sampling: sample u and then sample from $G(\log p)$ to generate y_k

Categorical reparametrization with Gumbel-Softmax Sang et al. 2017

Re-parametrization trick for categorical distributions

- Consider variable y following the K categorical distribution:

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

- Reparametrization:

$$y_k \sim G(\log p) = \frac{\exp((\log p_k + \varepsilon)/\tau)}{\sum_{j=0}^{K} \exp((\log p_j + \varepsilon)/\tau)} \;, \quad \varepsilon = -\log(-\log(u)), u \sim \mathcal{U}[0,1]$$

- Sampling: sample u and then sample from $G(\log p)$ to generate y_k

- In the forward pass you sample from the parametrized distribution

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

- In the backward pass you use the soft distribution:

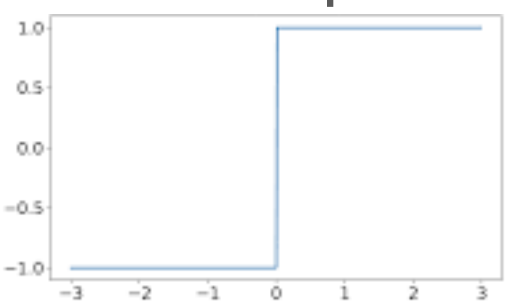
$$\frac{dc}{d\theta} = \frac{dG}{dp}\frac{dp}{d\theta}$$

Categorical reparametrization with Gumbel-Softmax Sang et al. 2017

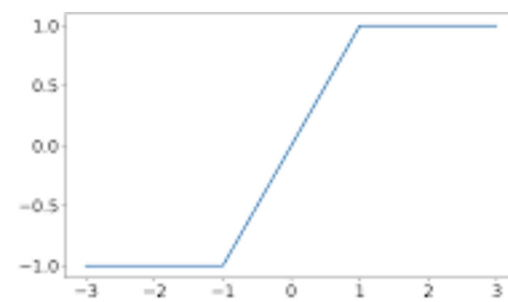
Bacproping through discrete variables

For binary neurons:

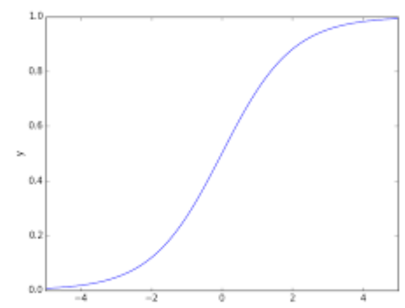
forward pass



backward pass



Straight-through



sigmoidal

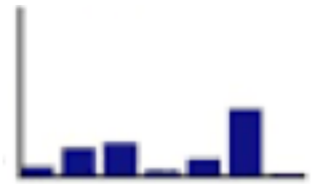
http://r2rt.com/binary-stochastic-neurons-in-tensorflow.html

Bacproping through discrete variables

For categorically distributed neurons:

forward pass

backward pass



Categorical reparametrization with Gumbel-Softmax Sang et al. 2017

Summary

- Recap of estimating gradients
- Backpropagating through sampling using the reparametrization trick.