Policy Gradient II

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Fall 2019

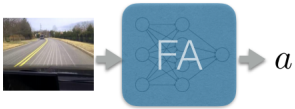
avereshc@buffalo.edu

October 22, 2019

*Slides are adopted from Deep Reinforcement Learning and Control by Katerina Fragkiadaki (Carnegie Mellon) & Policy Gradients by Pieter Abbeel – OpenAI / UC Berkeley / Gradescope

Policy function approximators

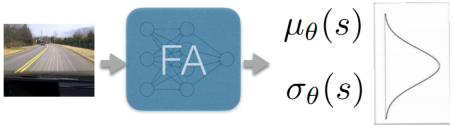
deterministic continuous policy



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

e.g. outputs a steering angle directly

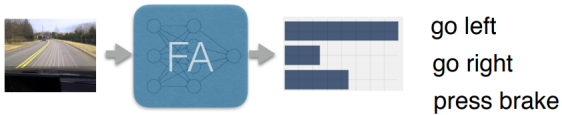
stochastic continuous policy



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

FA for stochastic multimodal continuous policies is an active area of generative model research

(stochastic) policy over discrete actions



Outputs a distribution over a discrete set of actions

Policy Optimization

- Policy based reinforcement learning is an optimization problem.

Policy Optimization

- Policy based reinforcement learning is an optimization problem.
- Find θ that maximises $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Genetic algorithms
- We will focus on stochastic gradient ascent, which is often quite efficient (and easy to use with deep nets)

Policy Gradient Methods

We consider methods for learning the policy parameter based on the gradient of some scalar performance measure $J(\theta)$ with respect to the policy parameter. We seek to *maximize performance*, so their updates approximate gradient ascent in J:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$$

Policy Objective Functions

Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ

- In episodic environments we can use the start value

$$J_1(\theta) = V_{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

Policy Objective Functions

Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ

- In episodic environments we can use the start value

$$J_1(\theta) = V_{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s d_{\pi_\theta}(s) V_{\pi_\theta}(s)$$

Policy Objective Functions

Goal: given policy $\pi_\theta(s, a)$ with parameters θ, find best θ

- In episodic environments we can use the start value

$$J_1(\theta) = V_{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

- In continuing environments we can use the average value

$$J_{avV}(\theta) = \sum_s d_{\pi_\theta}(s) V_{\pi_\theta}(s)$$

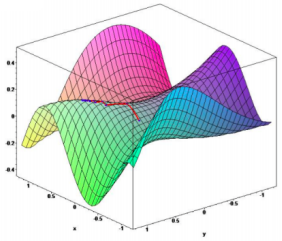
- Or the average reward per time-step

$$J_{avR}(\theta) = \sum_s d_{\pi_\theta}(s) \sum_a \pi_\theta(s, a) R_s^a$$

where $d^{\pi_\theta}(s)$ is a stationary distribution of Markov chain for π_θ

Policy Gradient

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



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

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

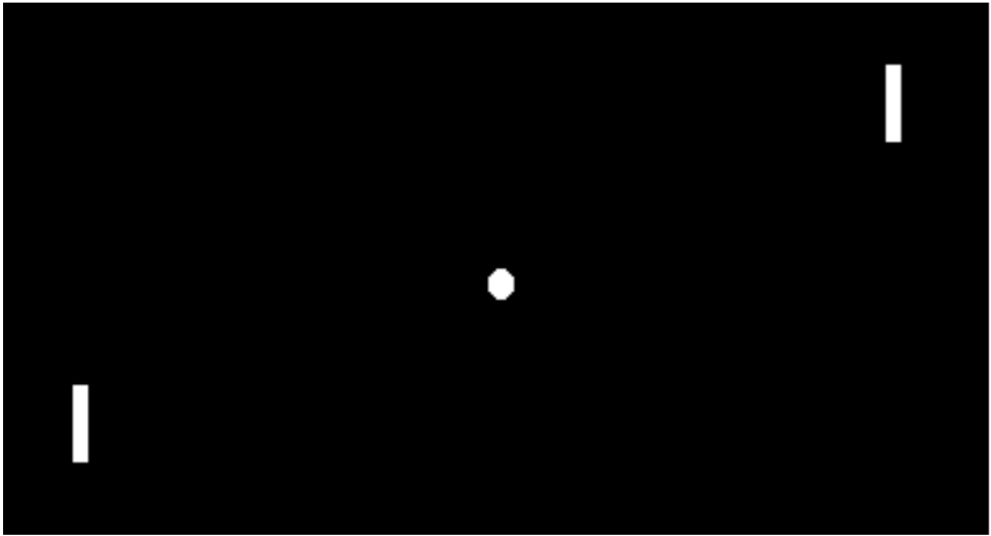
α is a step-size parameter (learning rate)

is the policy gradient

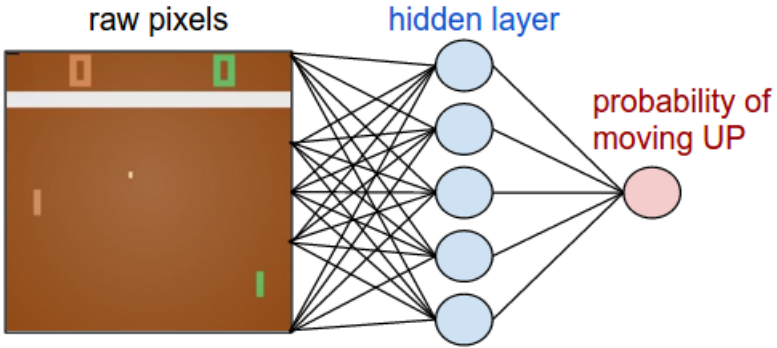
$$\nabla_\theta U(\theta) = \begin{pmatrix} \frac{\partial U(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial U(\theta)}{\partial \theta_n} \end{pmatrix}$$

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

Example: Pong from Pixels



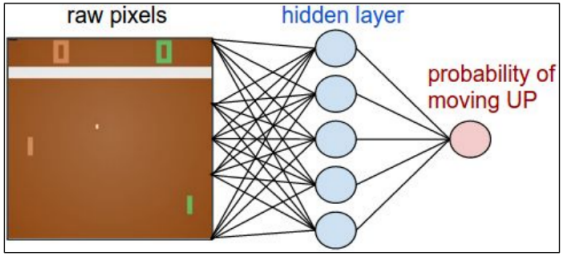
Example: Pong Policy Network



Example: Pong Policy Network

height width

[80 x 80] array

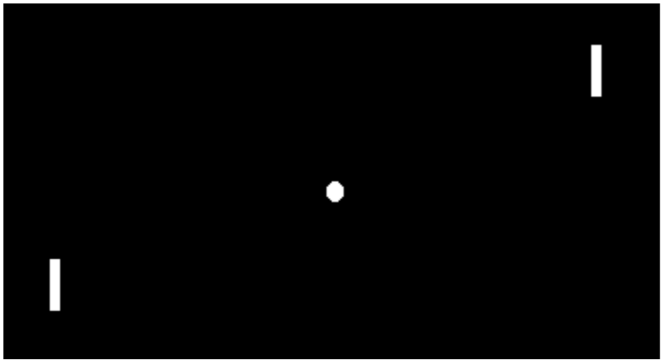


E.g. 200 nodes in the hidden network, so:

[(80*80)*200 + 200] + [200*1 + 1] = **~1.3M parameters**

Layer 1 Layer 2

Example: Pong from Pixels



Network does not see this. Network sees 80*80 = 6,400 numbers.
It gets a reward of +1 or -1, some of the time.
Q: How do we efficiently find a good setting of the 1.3M parameters?

Example: Pong from Pixels

- Random search
- Evolutionary methods
- Approximation to the gradient via finite differences
- Policy gradients

Example: Pong from Pixels

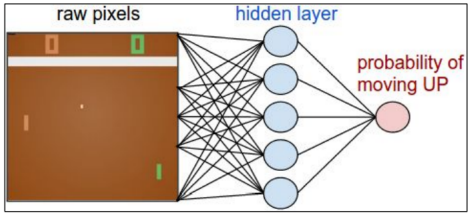
Suppose we had the training labels…
(we know what to do in any state)

(x1,UP)
(x2,DOWN)
(x3,UP)
...

Example: Pong from Pixels

Suppose we had the training labels…
(we know what to do in any state)

(x1,UP)
(x2,DOWN)
(x3,UP)
...



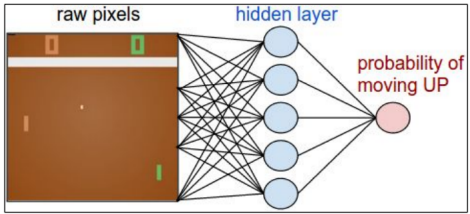
Example: Pong from Pixels

Suppose we had the training labels…
(we know what to do in any state)

(x1,UP)
(x2,DOWN)
(x3,UP)
...

maximize:

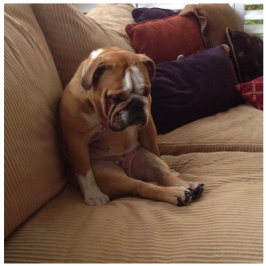
$$\sum_i \log p(y_i|x_i)$$

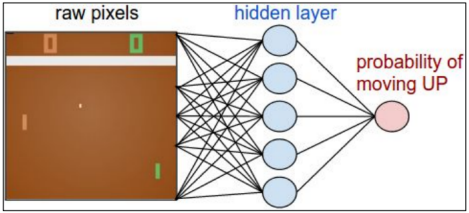


supervised learning

Example: Pong from Pixels

Except, we don't have labels...





Should we go UP or DOWN?

Example: Pong from Pixels

Except, we don't have labels...

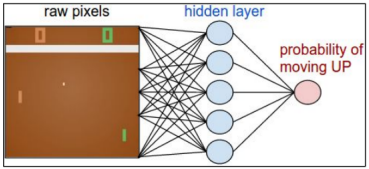
"Try a bunch of stuff and see what happens. Do more of the stuff that worked in the future."

-RL

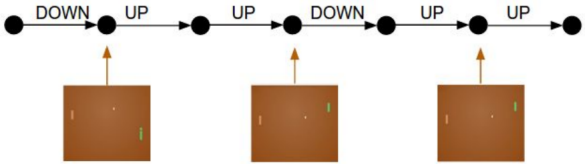
trial-and-error learning

Example: Pong from Pixels

Let's just act according to our current policy...



Rollout the policy and collect an episode

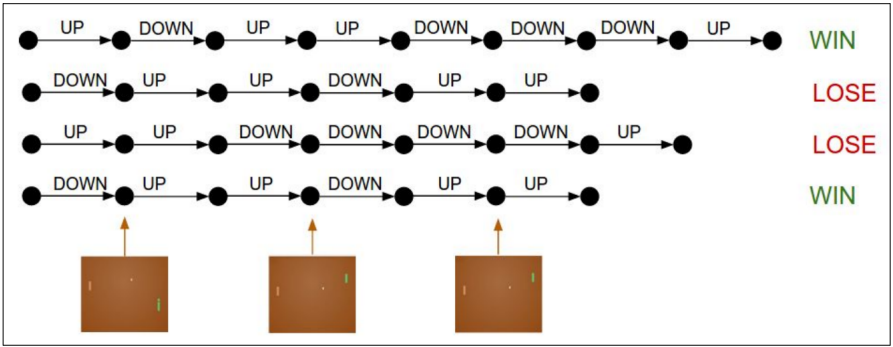


WIN

Example: Pong from Pixels

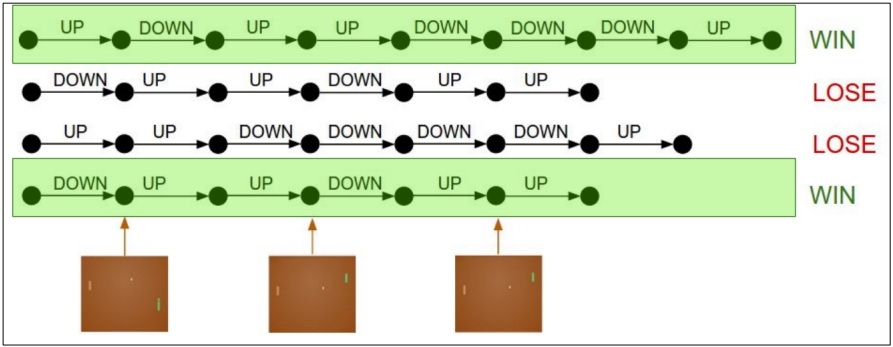
Collect many rollouts...

4 rollouts:



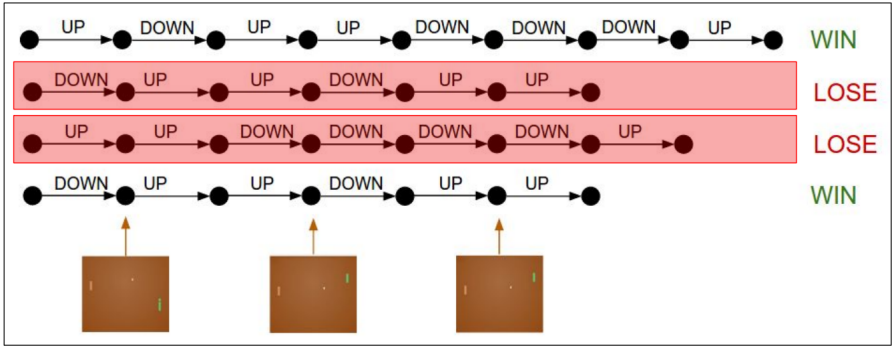
Example: Pong from Pixels

Not sure whatever we did here, but apparently it was good.



Example: Pong from Pixels

Not sure whatever we did here, but it was bad.



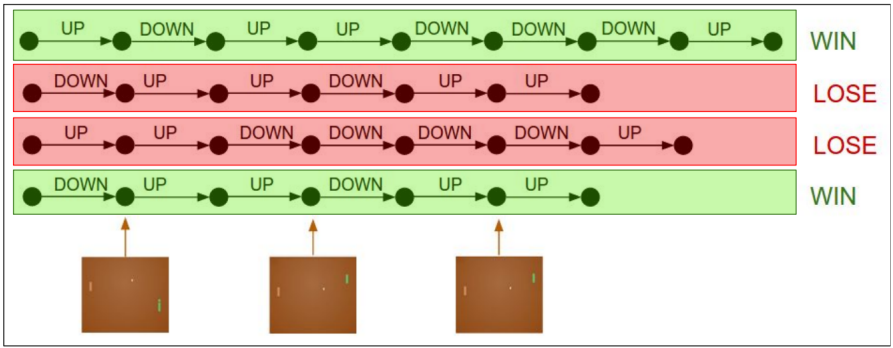
Example: Pong from Pixels

Pretend every action we took here was the correct label.

maximize: $\log p(y_i \mid x_i)$

Pretend every action we took here was the wrong label.

maximize: $(-1) * \log p(y_i \mid x_i)$



$$\nabla_\theta U(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)}) R(\tau^{(i)})$$

Supervised Learning

maximize:

$$\sum_i \log p(y_i|x_i)$$

For images x_i and their labels y_i.

Supervised Learning vs. Reinforcement Learning

Supervised Learning

maximize:

$$\sum_i \log p(y_i|x_i)$$

For images x_i and their labels y_i.

Reinforcement Learning

1) we have no labels so we sample:

$$y_i \sim p(\cdot|x_i)$$

Supervised Learning vs. Reinforcement Learning

Supervised Learning

maximize:

$$\sum_i \log p(y_i|x_i)$$

For images x_i and their labels y_i.

Reinforcement Learning

1) we have no labels so we sample:

$$y_i \sim p(\cdot|x_i)$$

2) once we collect a batch of rollouts:
maximize:

$$\sum_i A_i * \log p(y_i|x_i)$$

Supervised Learning vs. Reinforcement Learning

Supervised Learning

maximize:

$$\sum_i \log p(y_i|x_i)$$

For images x_i and their labels y_i.

Reinforcement Learning

1) we have no labels so we sample:

$$y_i \sim p(\cdot|x_i)$$

2) once we collect a batch of rollouts:
maximize:

$$\sum_i A_i * \log p(y_i|x_i)$$

We call this the **advantage**, it's a number, like +1.0 or -1.0 based on how this action eventually turned out.

Supervised Learning vs. Reinforcement Learning

Supervised Learning

maximize:

$$\sum_i \log p(y_i|x_i)$$

For images x_i and their labels y_i.

Reinforcement Learning

1) we have no labels so we sample:

$$y_i \sim p(\cdot|x_i)$$

2) once we collect a batch of rollouts:
maximize:

$$\sum_i A_i * \log p(y_i|x_i)$$

+ve advantage will make that action more likely in the future, for that state.
-ve advantage will make that action less likely in the future, for that state.

Policy Gradient

Gradient can be written as

$$\nabla_\theta J(\theta) \propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s)$$

Policy Gradient

Gradient can be written as

$$\nabla_\theta J(\theta) \propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} Q^\pi(s,a) \nabla_\theta \pi_\theta(a|s)$$

$$= \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)}$$

Policy Gradient

Gradient can be written as

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} Q^\pi(s,a)\nabla_\theta \pi_\theta(a|s) \\ &= \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) Q^\pi(s,a) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\ &= \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)] \qquad \text{; Because } (\ln x)'=1/x\end{aligned}$$

Where $\mathbb{E}_\pi$ refers to $\mathbb{E}_{s\sim d_\pi, a\sim\pi_\theta}$ when both state and action distributions follow the policy π_θ (on policy).

Policy Gradient

This vanilla policy gradient update has no bias but high variance. Many following algorithms were proposed to reduce the variance while keeping the bias unchanged.

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)]$$

REINFORCE (Monte-Carlo Policy Gradient)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return G_t as an unbiased sample of $Q^{\pi_\theta}(s_t, a_t)$

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

REINFORCE, A Monte-Carlo Policy-Gradient Method (episodic)

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta}), \forall a \in \mathcal{A}, s \in \mathcal{S}, \boldsymbol{\theta} \in \mathbb{R}^n$
Initialize policy weights $\boldsymbol{\theta}$
Repeat forever:
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  For each step of the episode $t = 0, \ldots, T-1$:
    $G_t \leftarrow$ return from step t
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t, \boldsymbol{\theta})$

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:

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

- 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

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

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)

Policy Gradient: Score Function

- We now compute the policy gradient analytically
- Assume
  - policy π_θ is differentiable whenever it is non-zero
  - we know the gradient $\nabla_\theta \pi_\theta(s, a)$
- Likelihood ratios exploit the following identity

$$\begin{aligned}\nabla_\theta \pi_\theta(s, a) &= \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)} \\ &= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)\end{aligned}$$

- The score function is $\nabla_\theta \log \pi_\theta(s, a)$

Softmax Policy: Discrete Actions

- We will use a softmax policy as a running example
- Weight actions using linear combination of features $\phi(s,a)^\top \theta$
- Probability of action is proportional to exponentiated weight

$$\pi_\theta(s,a) \propto e^{\phi(s,a)^\top \theta}$$

Nonlinear extension: replace $\phi(s,a)$ with a deep neural network with trainable weights w

Think a neural network with a softmax output probabilities

- The score function is

$$\nabla_\theta \log \pi_\theta(s,a) = \phi(s,a) - \mathbb{E}_{\pi_\theta}[\phi(s,\cdot)]$$

Gaussian Policy: Continuous Actions

- In continuous action spaces, a Gaussian policy is natural
- Mean is a linear combination of state features

$$\mu(s) = \phi(s)^\top \theta$$

Nonlinear extensions: replace $\phi(s)$ with a deep neural network with trainable weights w

- Variance may be fixed σ_2, or can also parameterized
- Policy is Gaussian $a \sim \mathcal{N}(\mu(s), \sigma^2)$
- The score function is

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

One-step MDP

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
  - Terminating after one time-step with reward $r = \mathcal{R}_{s,a}$
- First, let's look at the objective:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\ &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s, a) \mathcal{R}_{s,a} \end{aligned}$$

Intuition: Under MDP:

$$\mathbb{E}_{\pi_\theta}[r] = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} P_\theta(s, a) \mathcal{R}_{s,a} = \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} P(s) \pi_\theta(a|s) \mathcal{R}_{s,a}$$

$$= \sum_{s \in \mathcal{S}} P(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) \mathcal{R}_{s,a}$$

One-step MDP

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
  - Terminating after one time-step with reward $r = \mathcal{R}_{s,a}$
- Use likelihood ratios to compute the policy gradient

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi_\theta}[r] \\ &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \mathcal{R}_{s,a} \end{aligned}$$

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_{s \in \mathcal{S}} d(s) \sum_{a \in \mathcal{A}} \pi_\theta(s,a) \nabla_\theta \log \pi_\theta(s,a) \mathcal{R}_{s,a} \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r] \end{aligned}$$

Policy Gradient Theorem

- The policy gradient theorem generalizes the likelihood ratio approach to multi-step MDPs
- Replaces instantaneous reward r with long-term value $Q^{\pi}(s,a)$
- Policy gradient theorem applies to start state objective, average reward and average value objective

---

- For any differentiable policy $\pi_{\theta}(s, a)$, the policy gradient is

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi_{\theta}} \left[\nabla_{\theta} \log \pi_{\theta}(s, a) \; Q^{\pi_{\theta}}(s, a) \right]$$