Lecture 5: Policy Gradients

Zhi Wang & Chunlin Chen

Department of Control and Systems Engineering
Nanjing University

Oct. 29th, 2020

What we'll cover

— Contents

- The policy gradient algorithm
- What does the policy gradient do?
- Basic variance reduction: causality
- Basic variance reduction: baselines
- Policy gradient examples

— Goals

- Understand policy gradient RL
- Understand practical considerations for policy gradients

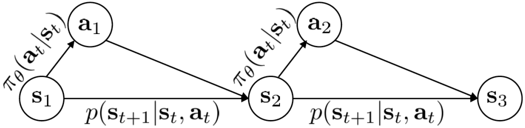
Table of Contents

Recall: the Goal of RL

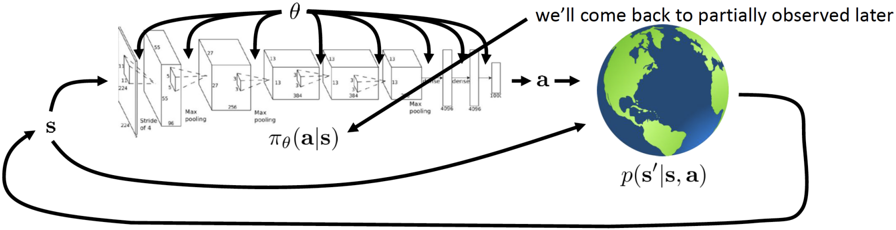
- Find **optimal policies** to maximize cumulative reward

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} r(s_t, a_t)\right]$$

  - In a **trial-and-error** manner
  - A general **optimization** framework for sequential decision-making



The goal of RL



$$p_{\boldsymbol{\theta}}(\tau) = p_{\boldsymbol{\theta}}(s_0, a_0, \dots, s_T, a_T) = p(s_0)\Pi_{t=0}^{T}\pi_{\boldsymbol{\theta}}(a_t|s_t)p(s_{t+1}|s_t, a_t)$$

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}(\tau)}\left[\sum_t r(s_t, a_t)\right]$$

The goal of RL

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}(\tau)}\left[\sum_t r(s_t, a_t)\right]$$

- Infinite horizon case

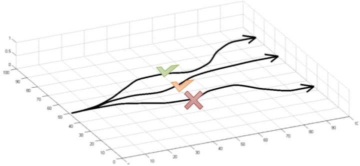
$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{(s,a)\sim p_{\boldsymbol{\theta}}(s,a)}[r(s,a)]$$

- Finite horizon case

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_{\boldsymbol{\theta}}(s_t,a_t)}[r(s_t,a_t)]$$

Evaluating the objective

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}\in\mathbb{R}^d} \underbrace{\mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}(\tau)}\left[\sum_t r(s_t, a_t)\right]}_{J(\boldsymbol{\theta})}$$



$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim p_{\boldsymbol{\theta}}(\tau)}\left[\sum_t r(s_t, a_t)\right] \approx \frac{1}{N} \underbrace{\sum_i}_{\uparrow} \sum_t r(s_{i,t}, a_{i,t})$$

sum over samples from $\pi_{\boldsymbol{\theta}}$

Direct policy differentiation

- Objective function / cost function

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\underbrace{r(\tau)}_{\sum_{t=0}^{T} r(s_t,a_t)}] = \int \pi_\theta(\tau) r(\tau)\,\mathrm{d}\tau$$

- The gradient – differentiate the objective function

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau)\,\mathrm{d}\tau = \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) r(\tau)\,\mathrm{d}\tau$$
$$= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

- A convenient identity

$$\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) = \pi_\theta(\tau)\frac{\nabla_\theta \log \pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta \pi_\theta(\tau)$$

Direct policy differentiation

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \underline{\log \pi_\theta(\tau)} r(\tau)]$$

$$\underbrace{\pi_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

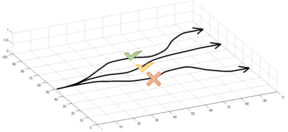
log of both sides

$$\log \pi_\theta(\tau) = \underline{\log p(\mathbf{s}_1) + \sum_{t=1}^{T}\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}$$

$$\nabla_\theta \left[\cancel{\log p(\mathbf{s}_1)} + \sum_{t=1}^{T}\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) + \cancel{\log p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)}\right]$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

Evaluating the policy gradient

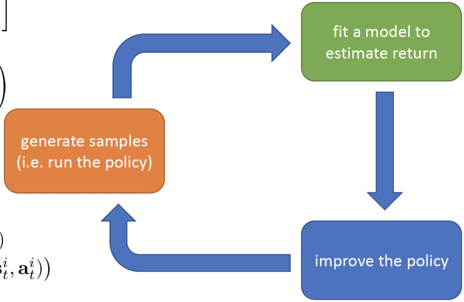


recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

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

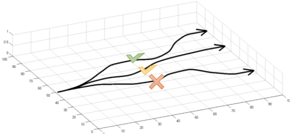


REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) \right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i) \right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Table of Contents

Evaluating the policy gradient

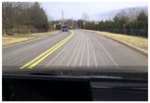


recall: $J(\theta) = E_{\tau \sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$

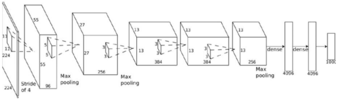
$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)\right)\right]$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})\right)$$

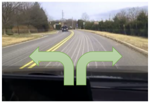
what is this?



$\mathbf{s}_t$



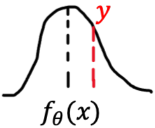
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$



$\mathbf{a}_t$

Maximum likelihood estimation (MLE)

- A method of estimating the parameters of a probability distribution by maximizing a likelihood function
  - So that under the assumed statistical model, **the observed data is most probable**



- For regression problem, input-label (x, y), prediction $\hat{y} = f_\theta(x)$

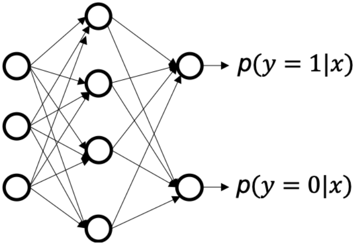
$$\text{likelihood:} \quad \theta^* = \arg\max_{\theta \in \mathbb{R}^d} \Pi_i \mathcal{N}(y_i; f_\theta(x_i), \sigma^2)$$

$$\text{negative log-likelihood:} \quad \theta^* = \arg\min_{\theta \in \mathbb{R}^d} \sum_i (y_i - f_\theta(x_i))^2$$

Maximum likelihood estimation (MLE)

- For classification problem, input-label (x, y), prediction $p_\theta(y|x)$
- Cross-entropy loss, minimize the negative log-likelihood:

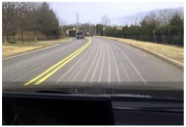
$$\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \sum_i -y_i \log p_\theta(y_i = 1|x_i) - (1 - y_i) \log(1 - p_\theta(y_i = 1|x_i))$$



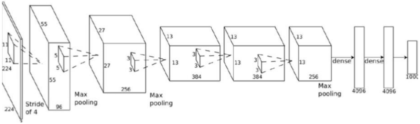
Comparison to maximum likelihood

policy gradient: $$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

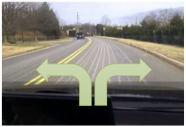
maximum likelihood: $$\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)$$



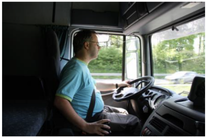
$\mathbf{s}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

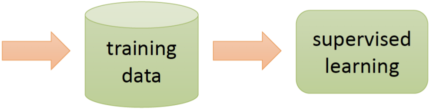


$\mathbf{a}_t$



$\mathbf{s}_t$

$\mathbf{a}_t$



$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$

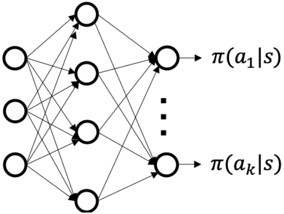
Discrete action space: Categorical distribution

- Categorical distribution for the finite number of actions
- What is Categorical distribution?

Discrete action space: Categorical distribution

- A discrete probability distribution that describes the possible results of a random variable that can take on one of k possible categories, with the probability of each category separately specified

- $k > 0$: number of categories
- $p_1, ..., p_k$: event probabilities
- $p_i > 0, \sum_i p_i = 1$



Discrete action space: Categorical distribution

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.distributions import Categorical


class CategoricalMLPPolicy(Policy):
    def __init__(self, state_dim, num_actions):
        super(CategoricalMLPPolicy, self).__init__()

        self.input_layer = nn.Linear(state_dim, 512)
        self.hidden_layer = nn.Linear(512, 512)
        self.output_layer = nn.Linear(512, num_actions)

    def forward(self, input, params=None):
        # states: (time_horizon * batch_size * state_dim) tensor of states
        x = self.input_layer(states)
        x = F.relu(x)
        x = self.hidden_layer(x)
        x = F.relu(x)

        # probabilities for discrete actions
        logits = self.output_layer(x)

        return Categorical(logits=logits)
```

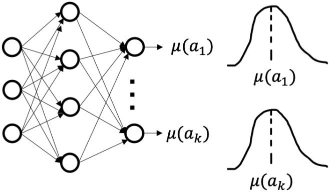
Continuous action space: Gaussian distribution

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \right) \left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \right)$$

example: $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(f_{\text{neural network}}(\mathbf{s}_t); \Sigma)$

$$\log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\|f(\mathbf{s}_t) - \mathbf{a}_t\|_\Sigma^2 + \text{const}$$

$$\nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) = -\frac{1}{2}\Sigma^{-1}(f(\mathbf{s}_t) - \mathbf{a}_t)\frac{df}{d\theta}$$



REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right) \left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Continuous action space: Gaussian distribution

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.distributions import Normal


class NormalMLPPolicy(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(NormalMLPPolicy, self).__init__()

        self.input_layer = nn.Linear(state_dim, 512)
        self.hidden_layer = nn.Linear(512, 512)
        self.output_layer = nn.Linear(512, action_dim)

        # Gaussian layer, add stochasticity to actions
        # the scale of the Gaussian distribution for the action
        self.sigma = nn.Parameter(torch.Tensor(action_dim))
        self.sigma.data.fill_(math.log(1.0))


    def forward(self, states):
        # states: (time_horizon * batch_size * state_dim) tensor of states
        x = self.input_layer(states)
        x = F.relu(x)
        x = self.hidden_layer(x)
        x = F.relu(x)

        # the mean of the Gaussian distribution for the action
        mu = self.output_layer(x)

        return Normal(loc=mu, scale=self.sigma)
```

What did we just do?

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_{i,t},\mathbf{a}_{i,t})\right)$$

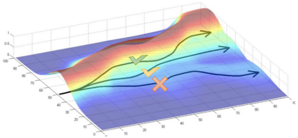
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\underbrace{\nabla_\theta \log \pi_\theta(\tau_i)}_{\sum_{t=1}^{T}\nabla_\theta \log_\theta \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} r(\tau_i)$$

maximum likelihood: $\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau_i)$

good stuff is made more likely

bad stuff is made less likely

simply formalizes the notion of “trial and error”!



REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i,\mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

Policy gradient with automatic differentiation

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \underline{\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \hat{Q}_{i,t}$$

pretty inefficient to compute these explicitly!

How can we compute policy gradients with automatic differentiation?

We need a graph such that its gradient is the policy gradient!

maximum likelihood: $\nabla_\theta J_{\mathrm{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})$ $\quad J_{\mathrm{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})$

Just implement "pseudo-loss" as a weighted maximum likelihood:

$$\tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

cross entropy (discrete) or squared error (Gaussian)

Policy gradient with automatic differentiation

- Pseudocode example (with discrete actions):
- Maximum likelihood:

```
# Given:
# states: (time_horizon * batch_size * state_dim) tensor of states
# actions: (time_horizon * batch_size * num_actions) tensor of actions

# Build the graph
logits = policy(states)
negative_likelihoods = torch.nn.CrossEntropyLoss(lables=actions, logits=logits)
loss = torch.mean(negative_likelihoods)   # The loss is a scalar
gradients = torch.autograd.grad(loss, policy.parameters())
```

Policy gradient with automatic differentiation

- Pseudocode example (with discrete actions):

- Policy gradient:

```
# Given:
# states: (time_horizon * batch_size * state_dim) tensor of states
# actions: (time_horizon * batch_size * num_actions) tensor of actions
# q_values: (time_horizon * batch_size * 1) tensor of estimated state-action values

# Build the graph
logits = policy(states)
negative_likelihoods = torch.nn.CrossEntropyLoss(lables=actions, logits=logits)
weighted_negative_likelihoods = negative_likelihoods * q_values  # element-wise multiply
loss = torch.mean(weighted_negative_likelihoods)   # The loss is a scalar
gradients = torch.autograd.grad(loss, policy.parameters())
```

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \underbrace{\left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right)}_{Q(s_{i,t}, a_{i,t}), \quad q\_values}$$

Review of the policy gradient

- Evaluating the RL objective
  - Generate samples
- Evaluating the policy gradient
  - Log gradient trick
  - Generate samples
- Understand policy gradient
  - Formalization of trial-and-error
- Can implement with automatic differentiation – need to know what to backpropagate

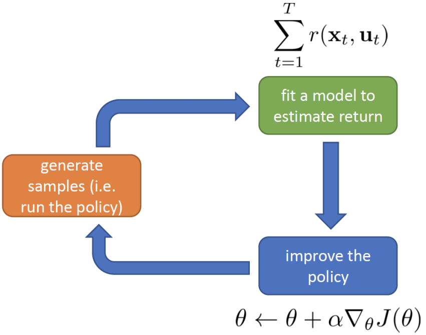
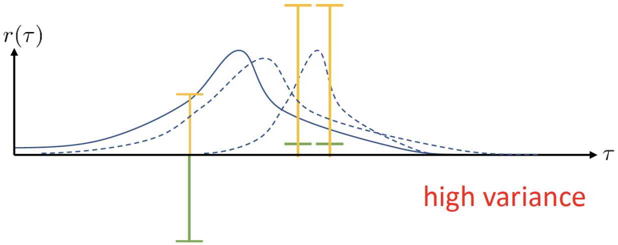


Table of Contents

What is wrong with the policy gradient?

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau) r(\tau)$$



high variance

- Even worse: what if the two "good" samples have $r(\tau) = 0$?

Reducing variance - Causality

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

$$\Downarrow$$

- **Causality**: policy at time t' cannot affect reward at time t when $t < t'$

$$\Downarrow$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \underbrace{\left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right)}_{Q(s_{i,t}, a_{i,t})}$$

"reward to go"

Reducing variance - Baselines

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau)[r(\tau) - b]$$

$$b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$$

- But... are we allowed to do that?

- The key problem is

$$\mathbb{E}_{\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] = \mathbb{E}_{\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)[r(\tau) - b]] \quad \textbf{???}$$

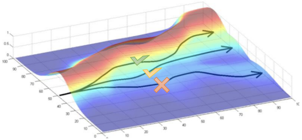
Reducing variance - Baselines

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau)[r(\tau) - b]$$

a convenient identity

$$\pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau)$$

$$b = \frac{1}{N} \sum_{i=1}^{N} r(\tau)$$



- But... are we allowed to do that?

$$\mathbb{E}_{\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) b] = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) b \, \mathrm{d}\tau = \int \nabla_\theta \pi_\theta(\tau) b \, \mathrm{d}\tau$$

$$= b \nabla_\theta \int \pi_\theta(\tau) \, \mathrm{d}\tau = b \nabla_\theta 1 = 0$$

- Subtracting a baseline is unbiased in expectation!
- Average reward is not the best baseline, but it's pretty good!

What is the best baseline?

- The best baseline makes the **variance** of $\nabla_\theta J(\theta)$ minimal

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)(r(\tau)-b)] = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}[g(\tau)(r(\tau)-b)]$$

- Compute the variance: $var[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2$

$$var = \mathbb{E}[g(\tau)^2(r(\tau)-b)^2] - \mathbb{E}[g(\tau)(r(\tau)-b)]^2$$

- **How to derive the best baseline b ???**

Analyzing the variance

can we write down the variance?

$$\text{Var}[x] = E[x^2] - E[x]^2$$

$$\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b)]$$

$$\text{Var} = E_{\tau\sim\pi_\theta(\tau)}[(\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b))^2] - \underbrace{E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b)]^2}$$

this bit is just $E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$
(baselines are unbiased in expectation)

$$\frac{d\text{Var}}{db} = \frac{d}{db} E[g(\tau)^2(r(\tau) - b)^2] = \frac{d}{db}\left(\cancel{E[g(\tau)^2 r(\tau)^2]} - 2E[g(\tau)^2 r(\tau) b] + b^2 E[g(\tau)^2]\right)$$

$$= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0$$

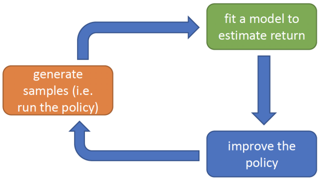
$$b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]} \longleftarrow$$

This is just expected reward, but weighted by gradient magnitudes!

Review

- The high variance of policy gradient
- Exploiting causality
  - Future doesn't affect the past
- Baselines
  - Unbiased!
- Analyzing variance
  - Can derive optimal baselines

$$\hat{Q}^{\pi}(\mathbf{x}_t, \mathbf{u}_t) = \sum_{t'=t}^{T} r(\mathbf{x}_{t'}, \mathbf{u}_{t'})$$



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

Table of Contents

Recall: On-policy vs. off-policy

Target policy $\pi(a|s)$	Behavior policy $b(a|s)$
To be evaluated or improved	To explore to generate data
Make decisions finally	Make decisions in training phase

- **On-policy** methods: $\pi(a|s) = b(a|s)$
  - Evaluate or improve the policy that is used to make decisions during training
  - e.g., SARSA
- **Off-policy** methods: $\pi(a|s) \neq b(a|s)$
  - Evaluate or improve a policy different from that used to generate the data
  - Separate exploration from control
  - e.g., Q-learning

Policy gradient is on-policy

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = \underline{E_{\tau\sim\pi_\theta(\tau)}}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

this is trouble...

- Neural networks change only a little bit with each gradient step
- On-policy learning can be extremely inefficient!

can't just skip this!

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

The off-policy case

- How to reuse samples that are generated by another behavior policy $\bar{\pi}, \bar{\pi} \neq \pi_\theta$?

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \\ &\neq \mathbb{E}_{\tau \sim \bar{\pi}(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]\end{aligned}$$

- How to estimate $\nabla_\theta J(\theta)$ using samples $\tau \sim \bar{\pi}(\tau)$?

Importance sampling

- A general technique for estimating properties of a particular distribution $p(x)$, while only having samples generated from a different distribution $q(x)$ than the distribution of interest $p(x)$

$$
\begin{aligned}
\mathbb{E}_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\
&= \int \frac{q(x)}{q(x)} p(x)f(x)\,\mathrm{d}x \\
&= \int q(x)\frac{p(x)}{q(x)} f(x)\,\mathrm{d}x \\
&= \mathbb{E}_{x\sim q(x)}\left[\frac{p(x)}{q(x)} f(x)\right]
\end{aligned}
$$

Importance sampling for estimating policy gradient

- We need to estimate the gradient $\nabla_\theta \log \pi_\theta(\tau) r(\tau)$ of a distribution $\tau \sim \pi_\theta(\tau)$, while only having samples generated from a different distribution $\tau \sim \bar{\pi}(\tau)$

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)] \\ &= \mathbb{E}_{\tau \sim \bar{\pi}(\tau)}[???]\end{aligned}$$

Off-policy learning & importance sampling

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

what if we don't have samples from $\pi_\theta(\tau)$?

(we have samples from some $\bar{\pi}(\tau)$ instead)

$$J(\theta) = E_{\tau\sim\bar{\pi}(\tau)}\left[\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} r(\tau)\right]$$

$$\pi_\theta(\tau) = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

$$\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} = \frac{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}}{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}} = \frac{\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$$

importance sampling

$$\begin{aligned}
E_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\
&= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\
&= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\
&= E_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]
\end{aligned}$$

Policy gradient with importance sampling

$\theta^\star = \arg\max_\theta J(\theta)$

$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$

can we estimate the value of some *new* parameters θ'?

$$J(\theta') = E_{\tau\sim\pi_\theta(\tau)}\left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)}r(\tau)\right]$$

the only bit that depends on θ'

$$\nabla_{\theta'}J(\theta') = E_{\tau\sim\pi_\theta(\tau)}\left[\frac{\nabla_{\theta'}\pi_{\theta'}(\tau)}{\pi_\theta(\tau)}r(\tau)\right] = E_{\tau\sim\pi_\theta(\tau)}\left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)}\nabla_{\theta'}\log\pi_{\theta'}(\tau)r(\tau)\right]$$

now estimate locally, at $\theta = \theta'$: $\nabla_\theta J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log\pi_\theta(\tau)r(\tau)]$

a convenient identity

$$\pi_\theta(\tau)\nabla_\theta\log\pi_\theta(\tau) = \nabla_\theta\pi_\theta(\tau)$$

The off-policy policy gradient

$$\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} = \frac{\prod_{t=1}^{T} \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}$$

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_{\theta'} J(\theta') = E_{\tau\sim\pi_\theta(\tau)}\left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)}\nabla_{\theta'}\log\pi_{\theta'}(\tau)r(\tau)\right] \quad \text{when } \theta \neq \theta'$$

$$= E_{\tau\sim\pi_\theta(\tau)}\left[\left(\prod_{t=1}^{T}\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}\right)\left(\sum_{t=1}^{T}\nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^{T} r(\mathbf{s}_t,\mathbf{a}_t)\right)\right] \text{ what about causality?}$$

$$= E_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^{T}\nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\underline{\left(\prod_{t'=1}^{t}\frac{\pi_{\theta'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}\right)}\left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'},\mathbf{a}_{t'})\right)\right]$$

future actions don't affect current weight

A first-order approximation for importance sampling

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \underbrace{\left(\prod_{t'=1}^{t} \frac{\pi_{\theta'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})} \right)}_{\text{exponential in } T\ldots} \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

let's write the objective a bit differently...

on-policy policy gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}$

$(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) \sim \pi_\theta(\mathbf{s}_t, \mathbf{a}_t)$

off-policy policy gradient: $\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})}{\pi_\theta(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \underbrace{\cancel{\frac{\pi_{\theta'}(\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{s}_{i,t})}}}_{\text{ignore this part}} \frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}$$

Example of off-policy policy gradient

Incremental Reinforcement Learning in Continuous Spaces via Policy Relaxation and Importance Weighting

Zhi Wang, *Student Member, IEEE*, Han-Xiong Li, *Fellow, IEEE,* and Chunlin Chen, *Member, IEEE*

Algorithm 1 Policy Relaxation With Importance Sampling

Input: Number of burn-in episodes k; learning rate α; batch size m

Output: Optimal policy parameters $\boldsymbol{\theta}^*$

```
Initialize the number of learning episodes: η ← 0
while not converged do
    if η ≤ k then
        π_r(a|s) = Uniform(A(s)), ∀s
        Sample m episodes from π_r: τ^i ~ π_r
        ∇_θ J(θ) = Σ_{i=1}^m [π_θ(τ^i)/π_r(τ^i)] ∇_θ log π_θ(τ^i) r(τ^i)
    else
        Sample m episodes from π_θ: τ^i ~ π_θ
        ∇_θ J(θ) = Σ_{i=1}^m ∇_θ log π_θ(τ^i) r(τ^i)
    end
    θ ← θ + α ∇_θ J(θ)
    η ← η + m
end
```

B. Policy Relaxation

In the new environment M_t, the agent tends to visit a small part of the whole state-action space when executing the previously learned policy, thus probably leading to a local optimum due to insufficient exploration. Hence, we propose a policy relaxation mechanism to encourage a proper exploration. Specifically, in the k burn-in learning episodes, the agent is forced to execute a relaxed policy where actions are randomly selected from the available set. For better readability, let $\boldsymbol{\theta}$ denote the current parameters in M_t, and $\pi_{\boldsymbol{\theta}}$ be the policy derived from $\boldsymbol{\theta}$. Regarding the number of learning episodes η, the agent's behavior policy π_r is relaxed as

$$\pi_r(a|s) = \begin{cases} \text{Uniform}(A(s)), & \eta \le k \\ \pi_{\boldsymbol{\theta}}(a|s), & \eta > k \end{cases} \tag{9}$$

Policy gradient in practice

- Remember that the gradient has high variance
  - This isn't the same as supervised learning!
  - Gradients will be really noisy!
- Consider using much larger batches
- Tweaking learning rates is very hard
  - Using adaptive step size rules like ADAM
  - More policy gradient specific learning rate adjustment methods...

Review

- Policy gradient is on-policy
- Can derive off-policy variant
  - Use importance sampling
  - Exponential scaling in T
  - Can ignore state portion (first-order approximation)
- Practical considerations: batch size, learning rates, optimizers

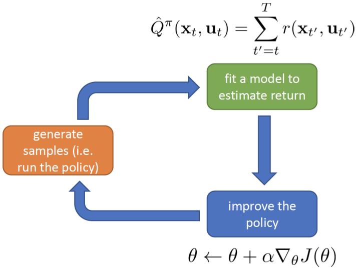
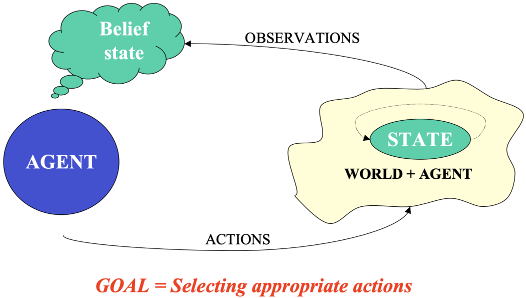


Table of Contents

Recall: Partially Observable MDP (POMDP)

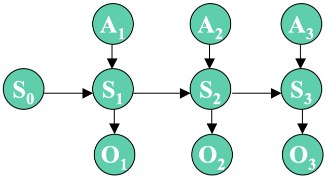
POMDP: Uncertainty

- Case 1: Uncertainty about the action outcome
- Case 2: Uncertainty about the world state due to imperfect (partial) information

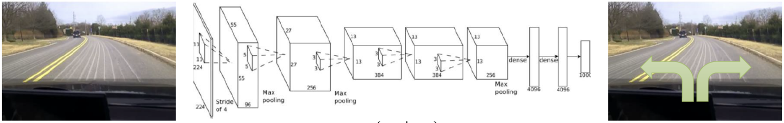


Recall: Partially Observable MDP (POMDP)

- A generalization of an MDP
  - the agent cannot directly observe the underlying state
  - it must maintain a probability distribution over the set of possible states, based on a set of observations and observation probabilities
- $M = \langle S, A, T, R, \Omega, O \rangle$
  - $\Omega = \{o_1, ..., o_k\}$ is a set of observations
  - $O(o|s', a)$ is a set of conditional observation probabilities



Partial Observability

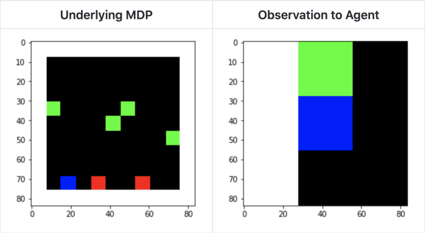


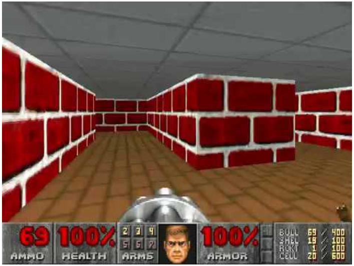
$\mathbf{o}_t$ $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ $\mathbf{a}_t$

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{N} \left(\sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_{i,t}|o_{i,t}) \right) \left(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right)$$

- Markov property is not actually used!
- Can use policy gradient in partially observable MDPs without modification, empirically

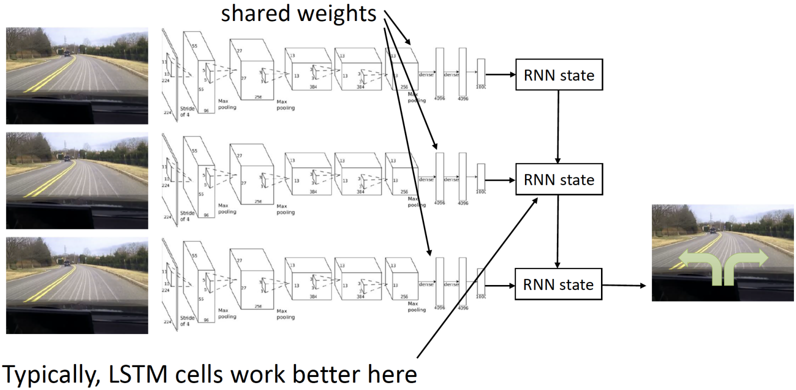
Examples: Partial Observability





Address partial observability: use the whole history

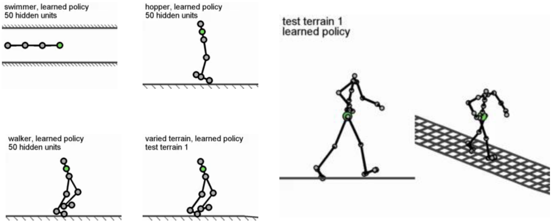
- Use recurrent neural network (RNN) as the encoder
- Embed the summary of past states into the internal **memory**



Example: policy gradient with importance sampling

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim \pi_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t) \left(\prod_{t'=1}^{t} \frac{\pi_{\theta'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})} \right) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

- Incorporate example demonstrations using importance sampling
- Neural network policies



Learning objectives of this lecture

- You should be able to...
  - Understand and be able to use the vanilla policy gradient method
  - Be able to use the baseline to reduce the variance of policy gradient
  - Know the importance sampling technique for off-policy policy gradient
  - Know the implementation tricks in practice

References

- Lecture 5 of CS285 at UC Berkeley, *Deep Reinforcement Learning, Decision Making, and Control*
  - `http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-5.pdf`
- Classic papers
  - Williams (1992). **Simple statistical gradient following algorithms for connectionist reinforcement learning**: introduces REINFORCE algorithm.
- DRL policy gradient papers
  - Y. Duan, et al., **Benchmarking Deep Reinforcement Learning for Continuous Control**, *ICML*, 2016.
  - Z. Wang, et al., **Incremental Reinforcement Learning in Continuous Spaces via Policy Relaxation and Importance Weighting**, *TNNLS*, 2019.

THE END