

# 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

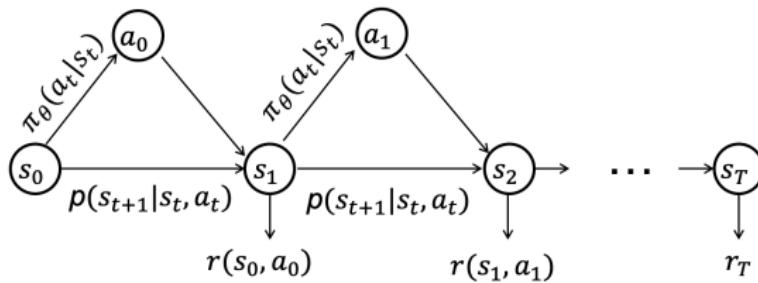
- 1 Formulation of policy gradient
- 2 Understanding policy gradient
- 3 Reducing the variance
- 4 Off-policy policy gradient
- 5 Partial Observability

# Review: the goal of RL

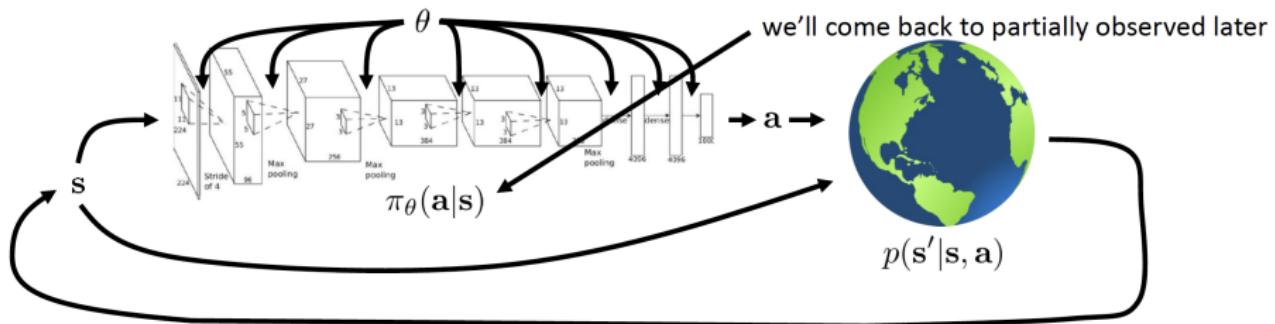
- Find **optimal policies** to maximize cumulative reward

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

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



# The goal of RL



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

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

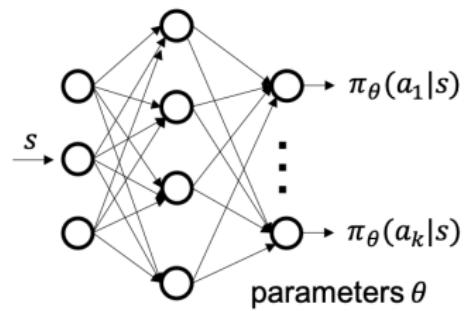
## Discrete action space: Categorical distribution

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

## Discrete action space: Categorical distribution

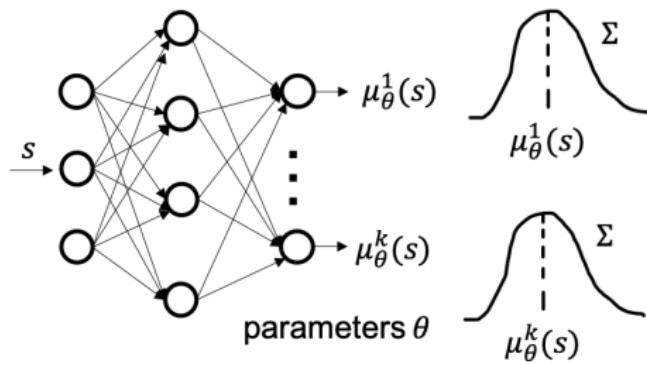
- A discrete probability distribution that describes the possible results of a random variable that can take on one of  $k$  possible categories, with the probability of each category separately specified

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



# Continuous action space: Gaussian distribution

- $k > 0$ : the dimension of action space,  $a = [a_1, \dots, a_k] \in \mathbb{R}^k$
- The Gaussian policy:  $\pi_\theta(a|s) = \mathcal{N}(\mu_\theta(s); \Sigma)$



# The goal of RL

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

- Infinite horizon case

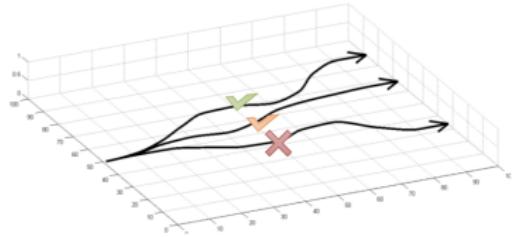
$$\theta^* = \arg \max_{\theta} \mathbb{E}_{(s,a) \sim \pi_\theta(s,a)} [r(s, a)]$$

- Finite horizon case

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

# Evaluating the objective

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



$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \approx \underbrace{\frac{1}{N} \sum_i \sum_t r(s_t^i, a_t^i)}_{\text{sum over samples from } \pi_\theta}$$

# Direct policy differentiation

- Objective function / cost function

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

- The gradient – differentiate the objective function

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

- 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

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

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

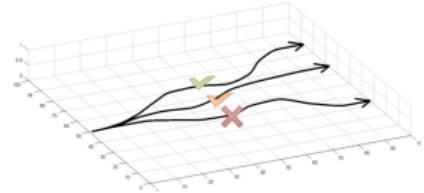
$$\begin{aligned}\nabla_{\theta} \log \pi_{\theta}(\tau) &= \nabla_{\theta} \left[ \log p(s_0) + \sum_{t=0}^T [\log \pi_{\theta}(a_t | s_t) + \log p(s_{t+1} | s_t, a_t)] \right] \\ &= \nabla_{\theta} \sum_{t=0}^T \log \pi_{\theta}(a_t | s_t)\end{aligned}$$

►  $\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}(\tau)} \left[ \left( \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \left( \sum_{t=0}^T r(s_t, a_t) \right) \right]$

# Evaluating the policy gradient

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [\sum_t r(s_t, a_t)] \approx \frac{1}{N} \sum_i \sum_t r(s_t^i, a_t^i)$$

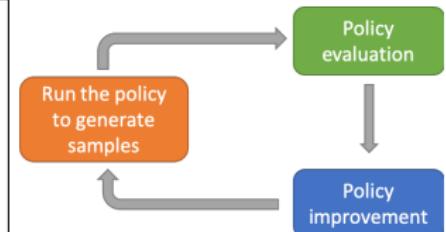
$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[ \left( \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \right) \left( \sum_{t=0}^T r(s_t, a_t) \right) \right] \\ &\approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \right) \left( \sum_{t=0}^T r(s_t^i, a_t^i) \right)\end{aligned}$$



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

REINFORCE algorithm: Loop:

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



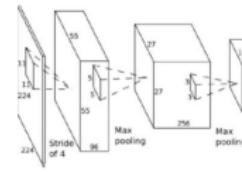
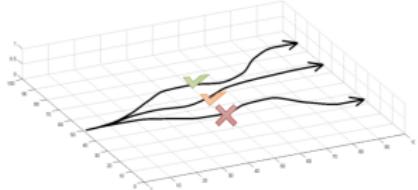
# Table of Contents

- 1 Formulation of policy gradient
- 2 Understanding policy gradient
- 3 Reducing the variance
- 4 Off-policy policy gradient
- 5 Partial Observability

# Evaluating the policy gradient

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} [\sum_t r(s_t, a_t)] \approx \frac{1}{N} \sum_i \sum_t r(s_t^i, a_t^i)$$

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[ \left( \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \right) \left( \sum_{t=0}^T r(s_t, a_t) \right) \right] \\ &\approx \frac{1}{N} \sum_{i=1}^N \underbrace{\left( \sum_{t=0}^T \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \right)}_{\text{What is this?}} \left( \sum_{t=0}^T r(s_t^i, a_t^i) \right)\end{aligned}$$



$s_t$

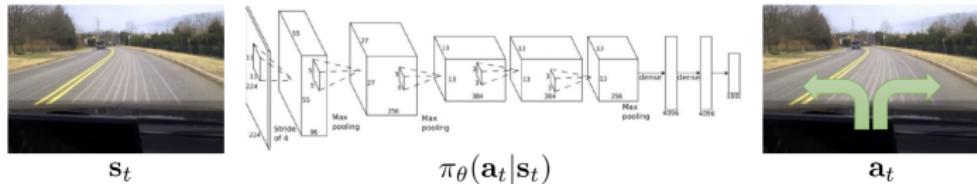
$$\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$$



$\mathbf{a}_t$

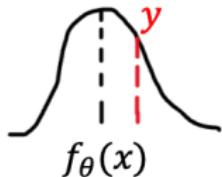
# Comparison to maximum likelihood

- Policy gradient:  $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \right) \left( \sum_{t=0}^T r(s_t^i, a_t^i) \right)$
- Maximum likelihood estimation:  $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \right)$



# 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  $\hat{y} = f_\theta(x)$



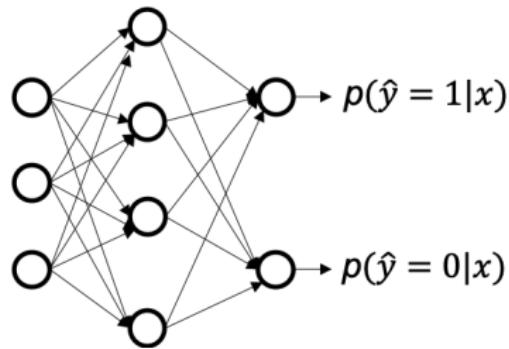
$$\text{likelihood: } \theta^* = \arg \max_{\theta \in \mathbb{R}^d} \prod_i \mathcal{N}(y_i; f_\theta(x_i), \sigma^2)$$

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

# Maximum likelihood estimation (MLE)

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

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



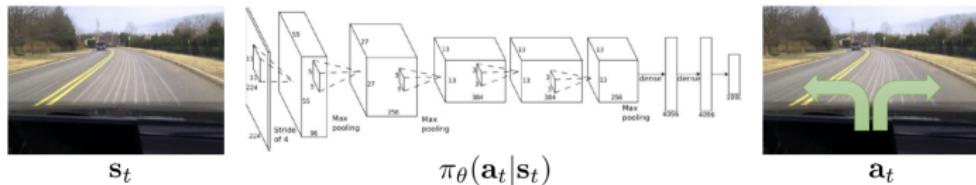
# Maximum likelihood estimation – Imitation learning

- Maximum likelihood estimation:  $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \right)$
- Review: Imitation learning – Imitating expert demonstrations  $(s_t^i, a_t^i)$ 
  - Maximizing the likelihood function  $\pi_{\theta}(a_t^i | s_t^i)$
  - Under the assumed statistical model  $\theta$ ,  
**the expert demonstrations are most probable**



# Comparison to maximum likelihood

- Policy gradient:  $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \right) \left( \sum_{t=0}^T r(s_t^i, a_t^i) \right)$ 
  - **Make what most probable?**
- Maximum likelihood estimation:  $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \right)$ 
  - Make every demonstration most probable



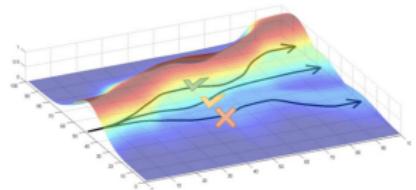
# What did we just do?

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \right) \left( \sum_{t=0}^T r(s_t^i, a_t^i) \right)$$

- Policy gradient:  $\nabla_{\theta} J(\theta) \approx \sum_{i=1}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(\tau^i) r(\tau^i)$ 
  - large  $r(\tau^i)$ : push  $\pi_{\theta}(\tau^i)$  close to 1, good stuff is made more likely
  - small  $r(\tau^i)$ : push  $\pi_{\theta}(\tau^i)$  close to 0, bad stuff is made less likely
  - simply formalizes the note of “trial and error”

REINFORCE algorithm: Loop:

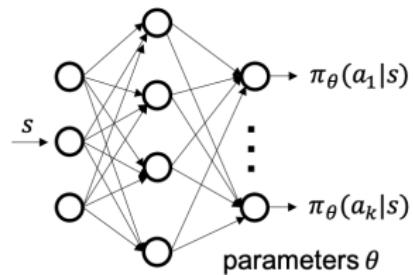
1. sample  $\{\tau^i\}$  from  $\pi_{\theta}(a_t | s_t)$  (run the policy)
2.  $\nabla_{\theta} J(\theta) \approx \sum_i (\sum_t \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i)) (\sum_t r(s_t^i, a_t^i))$
3.  $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$



## 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
- $p_1, \dots, p_k$ : event probabilities
- $p_i > 0, \sum_i p_i = 1$



# Discrete action space: Categorical distribution

```
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)

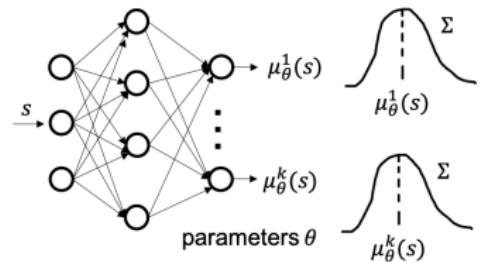
    def forward(self, input, params=None):
        # states: (time_horizon * batch_size * state_dim) tensor of states
        x = self.input_layer(states)
        x = F.relu(x)
        x = self.hidden_layer(x)
        x = F.relu(x)

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

        return Categorical(logits=logits)
```

# Continuous action space: Gaussian distribution

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) \right) \left( \sum_{t=0}^T r(s_t^i, a_t^i) \right)$$



- example:  $\pi_{\theta}(a_t | s_t) = \mathcal{N}(\mu_{\theta}(s_t); \Sigma)$
- $\log \pi_{\theta}(a_t | s_t) = -\frac{1}{2} \| \mu_{\theta}(s_t) - a_t \|_{\Sigma}^2 + \text{const}$
- $\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) = -\Sigma^{-1} (\mu_{\theta}(s_t) - a_t) \frac{d\mu_{\theta}(s_t)}{d\theta}$

REINFORCE algorithm: Loop:

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

# Continuous action space: Gaussian distribution

```
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)
```

# Policy gradient with automatic differentiation

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \underbrace{\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i)}_{\text{red}} \right) \left( \sum_{t=0}^T r(s_t^i, a_t^i) \right)$$

- 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!

- Just implement “pseudo-loss” as a *weighted* maximum likelihood:

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \underbrace{\log \pi_{\theta}(a_t^i | s_t^i)}_{\text{cross-entropy loss (discrete)}} Q^{\pi}(s_t^i, a_t^i)$$

mean squared error (continuous)

- maximum likelihood estimation

$$\nabla_{\theta} J_{ML}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i)$$

$$J_{ML}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^T \log \pi_{\theta}(a_t^i | s_t^i)$$

# 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(labels=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

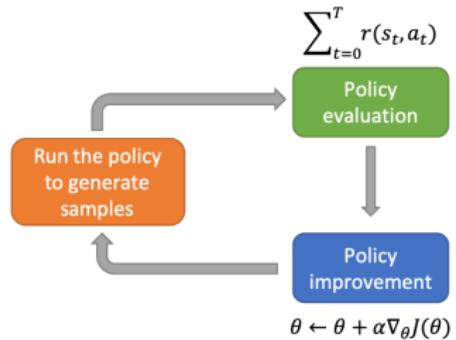
- Pseudocode example (with discrete actions):
- Policy gradient:

```
# 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(labels=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())
```

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

# 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

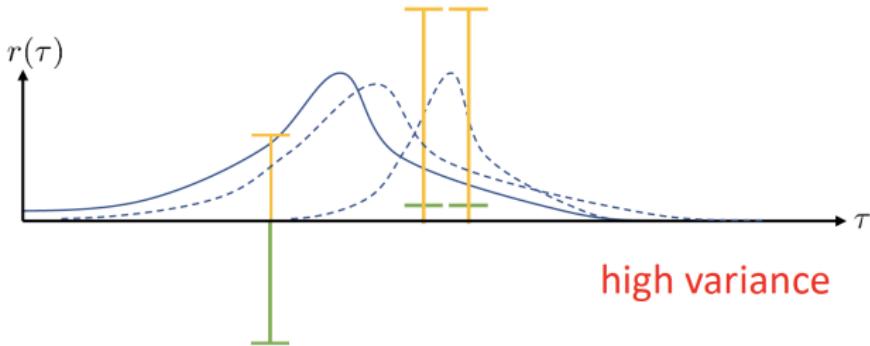


# Table of Contents

- 1 Formulation of policy gradient
- 2 Understanding policy gradient
- 3 Reducing the variance
- 4 Off-policy policy gradient
- 5 Partial Observability

# What is wrong with the policy gradient?

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



- 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  $t < t'$



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

“reward to go”

## Review: Value functions

- $v^\pi$ , the **state-value function** for policy  $\pi$

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi \left[ \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right], \forall s \in \mathcal{S}$$

- $Q^\pi$ , the **action-value function** for policy  $\pi$

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi \left[ \sum_{k=1}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right], \forall s \in \mathcal{S}, a \in \mathcal{A}(s) \end{aligned}$$

## Reducing variance - Baselines

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

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

- But... are we allowed to do that?
- The key problem is

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

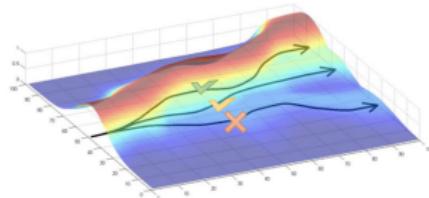
# Reducing variance - Baselines

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

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?

$$\begin{aligned}\mathbb{E}_{\pi_{\theta}(\tau)}[\nabla_{\theta} \log \pi_{\theta}(\tau) b] &= \int \pi_{\theta}(\tau) \nabla_{\theta} \log \pi_{\theta}(\tau) b d\tau = \int \nabla_{\theta} \pi_{\theta}(\tau) b d\tau \\ &= b \nabla_{\theta} \int \pi_{\theta}(\tau) d\tau = b \nabla_{\theta} 1 = 0\end{aligned}$$

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

$$var = \mathbb{E}_{\tau \sim \pi_{\theta}(\tau)} [g(\tau)^2 (r(\tau) - b)^2] - \mathbb{E}_{\tau \sim \pi_{\theta}(\tau)} [\nabla_{\theta} \log \pi_{\theta}(\tau) (r(\tau) - b)]^2$$

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

# Analyzing the variance

$$var = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} [g(\tau)^2(r(\tau) - b)^2] - \underbrace{\mathbb{E}_{\tau \sim \pi_\theta(\tau)} [\nabla_\theta \log \pi_\theta(\tau)(r(\tau) - b)]^2}_{\mathbb{E}_{\tau \sim \pi_\theta(\tau)} [\nabla_\theta \log \pi_\theta(\tau)r(\tau)]^2}$$

(baselines are unbiased in expectation)

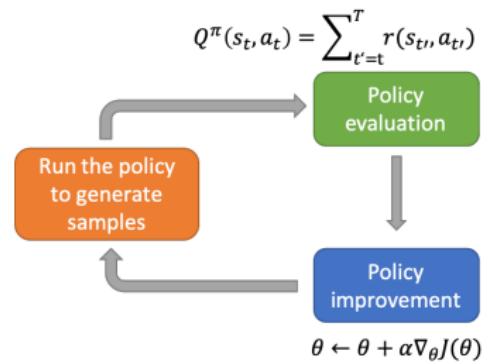
$$\begin{aligned}\frac{dvar}{db} &= \frac{d}{db} \mathbb{E}[g(\tau)^2(r(\tau) - b)^2] \\ &= \frac{d}{db} (\mathbb{E}[g(\tau)^2 r(\tau)^2] \underbrace{- 2\mathbb{E}[g(\tau)^2 r(\tau)b] + b^2 \mathbb{E}[g(\tau)^2]}_{\text{dependent of } b}) \\ &= -2\mathbb{E}[g(\tau)^2 r(\tau)] + 2b\mathbb{E}[g(\tau)^2] = 0\end{aligned}$$

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

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



# Table of Contents

- 1 Formulation of policy gradient
- 2 Understanding policy gradient
- 3 Reducing the variance
- 4 Off-policy policy gradient
- 5 Partial Observability

# Review: 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

$$\theta^* = \arg \max_{\theta \in \mathbb{R}^d} J(\theta)$$

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} [r(\tau)]$$

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

This is the trouble!

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

REINFORCE algorithm: Loop:

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

can't just skip this!

# 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  $p(x)$

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

# 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 with importance sampling

- 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  $\tau \sim \bar{\pi}(\tau)$

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

$$\frac{\pi_{\theta}(\tau)}{\bar{\pi}(\tau)} = \frac{p(s_0) \prod_{t=0}^T \pi_{\theta}(a_t | s_t) p(s_{t+1} | s_t, a_t)}{p(s_0) \prod_{t=0}^T \bar{\pi}(a_t | s_t) p(s_{t+1} | s_t, a_t)} = \frac{\prod_{t=0}^T \pi_{\theta}(a_t | s_t)}{\prod_{t=0}^T \bar{\pi}(a_t | s_t)}$$

# Policy gradient with importance sampling

- $\theta^* = \arg \max_{\theta \in \mathbb{R}^d} J(\theta), \quad J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[r(\tau)]$
- How to estimate the value of some *new* parameters  $\theta'$ ?

$$\begin{aligned}\nabla_{\theta'} J(\theta') &= \mathbb{E}_{\tau \sim \pi_{\theta'}(\tau)}[\nabla_{\theta'} \log \pi_{\theta'}(\tau) r(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[ \frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} \nabla_{\theta} \log \pi_\theta(\tau) r(\tau) \right] \\ &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[ \left( \frac{\prod_{t=0}^T \pi_{\theta'}(a_t | s_t)}{\prod_{t=0}^T \pi_\theta(a_t | s_t)} \right) \left( \sum_{t=0}^T \nabla_{\theta'} \log \pi_{\theta'}(a_t | s_t) \right) \left( \sum_{t=0}^T r(s_t, a_t) \right) \right] \\ &= \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[ \sum_{t=0}^T \nabla_{\theta'} \log \pi_{\theta'}(a_t | s_t) \left( \prod_{t'=1}^t \frac{\pi_{\theta'}(a_{t'} | s_{t'})}{\pi_\theta(a_{t'} | s_{t'})} \right) \left( \sum_{t'=t}^T r(s_{t'}, a_{t'}) \right) \right]\end{aligned}$$

Causality: future actions don't affect current weight

# A first-order approximation for importance sampling

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim \pi_{\theta}(\tau)} \left[ \sum_{t=0}^T \nabla_{\theta'} \log \pi_{\theta'}(a_t | s_t) \underbrace{\left( \prod_{t'=1}^t \frac{\pi_{\theta'}(a_{t'} | s_{t'})}{\pi_{\theta}(a_{t'} | s_{t'})} \right)}_{\text{exponential in } T} \left( \sum_{t'=t}^T r(s_{t'}, a_{t'}) \right) \right]$$

- On-policy policy gradient:  $(s_t^i, a_t^i) \sim \pi_{\theta}(s_t, a_t)$

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \nabla_{\theta} \log \pi_{\theta}(a_t^i | s_t^i) Q^{\pi}(s_t^i, a_t^i)$$

write the objective  
a bit differently

- Off-policy policy gradient:

$$\nabla_{\theta'} J(\theta') \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\pi_{\theta'}(s_t^i, a_t^i)}{\pi_{\theta}(s_t^i, a_t^i)} \nabla_{\theta'} \log \pi_{\theta'}(a_t^i | s_t^i) Q^{\pi}(s_t^i, a_t^i)$$

ignore this part

$$= \frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \frac{\pi_{\theta'}(s_t^i)}{\pi_{\theta}(s_t^i)} \frac{\pi_{\theta'}(a_t^i | s_t^i)}{\pi_{\theta}(a_t^i | s_t^i)} \nabla_{\theta'} \log \pi_{\theta'}(a_t^i | s_t^i) Q^{\pi}(s_t^i, a_t^i)$$

$$\frac{\pi_{\theta'}(s_t^i)}{\pi_{\theta}(s_t^i)} \approx 1$$

# Example of off-policy policy gradient

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

Zhi Wang<sup>✉</sup>, Student Member, IEEE, Han-Xiong Li<sup>✉</sup>, Fellow, IEEE, and Chunlin Chen<sup>✉</sup>, Member, IEEE

---

### Algorithm 1 Policy Relaxation With Importance Sampling

---

**Input:** Number of burn-in episodes  $k$ ;  
learning rate  $\alpha$ ; batch size  $m$   
**Output:** Optimal policy parameters  $\theta^*$

1 Initialize the number of learning episodes:  $\eta \leftarrow 0$   
2 **while** not converged **do**  
3   **if**  $\eta \leq k$  **then**  
4      $\pi_r(a|s) = \text{Uniform}(A(s)), \forall s$   
5     Sample  $m$  episodes from  $\pi_r$ :  $\tau^i \sim \pi_r$   
6      $\nabla_{\theta} J(\theta) = \sum_{i=1}^m \frac{\pi_{\theta}(\tau^i)}{\pi_r(\tau^i)} \nabla_{\theta} \log \pi_{\theta}(\tau^i) r(\tau^i)$   
7   **else**  
8     Sample  $m$  episodes from  $\pi_{\theta}$ :  $\tau^i \sim \pi_{\theta}$   
9      $\nabla_{\theta} J(\theta) = \sum_{i=1}^m \nabla_{\theta} \log \pi_{\theta}(\tau^i) r(\tau^i)$   
10   **end**  
11    $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$   
12    $\eta \leftarrow \eta + m$   
13 **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  $\theta$  denote the current parameters in  $M_t$ , and  $\pi_{\theta}$  be the policy derived from  $\theta$ . Regarding the number of learning episodes  $\eta$ , the agent's behavior policy  $\pi_r$  is relaxed as

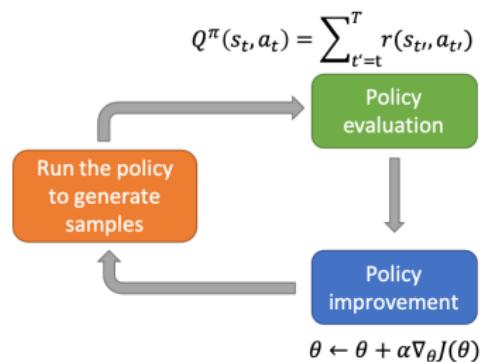
$$\pi_r(a|s) = \begin{cases} \text{Uniform}(A(s)), & \eta \leq k \\ \pi_{\theta}(a|s), & \eta > k \end{cases} \quad (9)$$

# 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



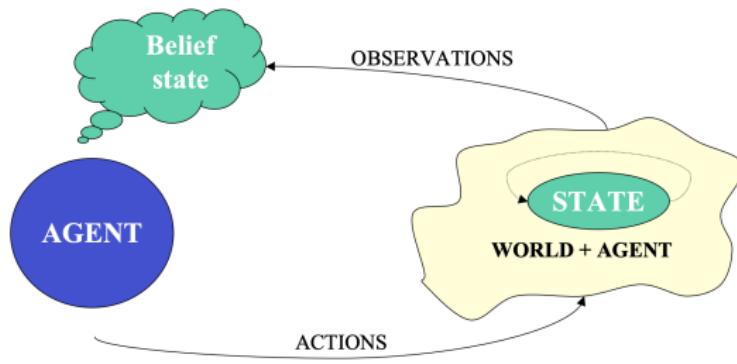
# Table of Contents

- 1 Formulation of policy gradient
- 2 Understanding policy gradient
- 3 Reducing the variance
- 4 Off-policy policy gradient
- 5 Partial Observability

# Recall: Partially Observable MDP (POMDP)

## POMDP: Uncertainty

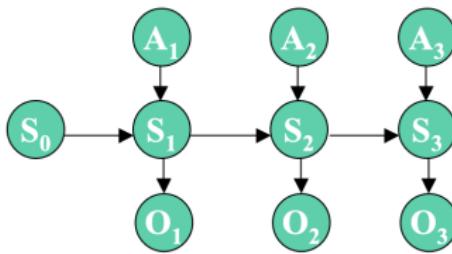
- Case 1: Uncertainty about the action outcome
- Case 2: Uncertainty about the world state due to imperfect (partial) information



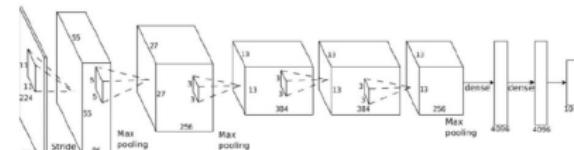
*GOAL = Selecting appropriate actions*

# 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
- $M = \langle S, A, T, R, \Omega, O \rangle$ 
  - $\Omega = \{o_1, \dots, o_k\}$  is a set of observations
  - $O(o|s', a)$  is a set of conditional observation probabilities



# Partial Observability

 $\mathbf{o}_t$ 

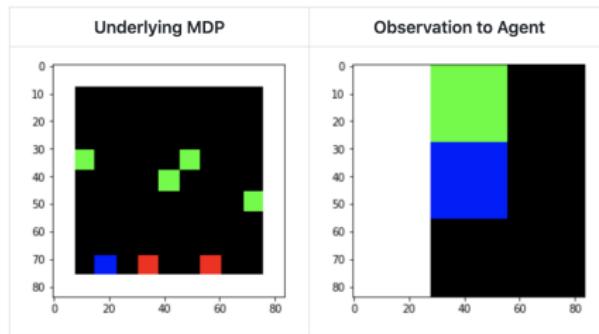
$$\pi_{\theta}(\mathbf{a}_t | \mathbf{o}_t)$$

 $\mathbf{a}_t$ 

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

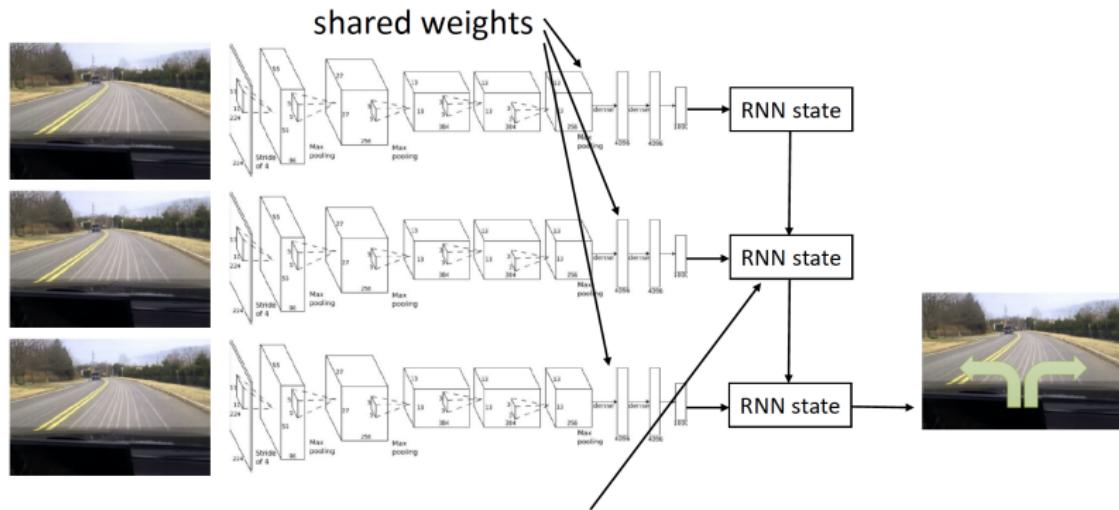
- Markov property is not actually used!
- Can use policy gradient in partially observable MDPs without modification, empirically

# 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**



Typically, LSTM cells work better here

# 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