

# Principles of Robot Autonomy II

Model-based and Model-free RL for Robot Control



# Learning from Experience

How to use trajectory data?

- Model based approach: estimate  $T(x'|x, u)$ , then use model to plan
- Model free:
  - Value based approach: estimate optimal value (or  $Q$ ) function from data
  - Policy based approach: use data to determine how to improve policy
  - Actor Critic approach: learn both a policy and a value/ $Q$  function

# Learning from Experience

How to use trajectory data?

- Model based approach: estimate  $T(x'|x, u)$ , then use model to plan
- Model free:
  - Value based approach: estimate optimal value (or  $Q$ ) function from data
  - Policy based approach: use data to determine how to improve policy
  - Actor Critic approach: learn both a policy and a value/ $Q$  function

# Model-free, policy based: Policy Gradient

Alternative: instead of learning the  $Q$  function, learn the policy directly!

Define a class of policies  $\pi_\theta$  where  $\theta$  are the parameters of the policy

Can we learn the optimal  $\theta$  from interaction?

**Goal:** use trajectories to estimate a gradient of policy performance w.r.t. parameters  $\theta$

# Policy Gradient

A particular value of  $\theta$  induces a distribution  $p(\tau; \theta)$  over possible trajectories

- Distribution comes from stochastic dynamics  $T(x' | x, u)$  as well as stochastic policy  $u \sim \pi(\cdot | x; \theta)$ .

Objective function:

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

i.e.,

$$J(\theta) = \int_{\tau} r(\tau) p(\tau; \theta) d\tau$$

where  $r(\tau)$  is the total discounted cumulative reward of a trajectory  $\tau$

# Policy Gradient

Gradient of objective w.r.t. parameters:

$$\nabla_{\theta} J(\theta) = \int_{\tau} r(\tau) \nabla_{\theta} p(\tau; \theta) d\tau$$

Trick:  $\nabla_{\theta} p(\tau; \theta) = p(\tau; \theta) \frac{\nabla_{\theta} p(\tau; \theta)}{p(\tau; \theta)} = p(\tau; \theta) \nabla_{\theta} \log p(\tau; \theta)$

$$\nabla_{\theta} J(\theta) = \int_{\tau} (r(\tau) \nabla_{\theta} \log p(\tau; \theta)) p(\tau; \theta) d\tau$$

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

# Policy Gradient

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

$$\begin{aligned} \log p(\tau; \theta) &= \log \left( \prod_{t \geq 0} T(x_{t+1} | x_t, u_t) \pi_{\theta}(u_t | x_t) \right) \\ &= \sum_{t \geq 0} \log T(x_{t+1} | x_t, u_t) + \log \pi_{\theta}(u_t | x_t) \\ \Rightarrow \nabla_{\theta} \log p(\tau; \theta) &= \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(u_t | x_t) \end{aligned}$$

# Policy Gradient

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

$$\begin{aligned} \log p(\tau; \theta) &= \log \left( \prod_{t \geq 0} T(x_{t+1} | x_t, u_t) \pi_{\theta}(u_t | x_t) \right) \\ &= \sum_{t \geq 0} \log T(x_{t+1} | x_t, u_t) + \log \pi_{\theta}(u_t | x_t) \end{aligned}$$

$$\Rightarrow \nabla_{\theta} \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(u_t | x_t)$$

We don't need to know the transition model to compute this gradient!

# Policy Gradient

If we use  $\pi_\theta$  to sample a trajectory, we can approximate the gradient via  $N$  Monte Carlo samples:

$$\begin{aligned}\nabla_\theta J(\theta) &= E_{\tau \sim p(\tau; \theta)} [r(\tau) \nabla_\theta \log p(\tau; \theta)] \\ &\approx \frac{1}{N} \sum_{i=1}^N \left( r(\tau^{(i)}) \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(u_t^{(i)} | x_t^{(i)}) \right)\end{aligned}$$

Intuition: adjust  $\theta$  to:

- Boost probability of actions taken if reward is high
- Lower probability of actions taken if reward is low

Learning by trial and error

# Time dependency of policy gradient theorem

- Previous estimator for policy gradient was

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( r(\tau^{(i)}) \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(u_t^{(i)} | x_t^{(i)}) \right)$$

Action  $u_{t'}$  can not change reward  $r_t$  for  $t < t'$  (i.e., previous timesteps):

$$\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t \geq 0} \nabla_{\theta} \log \pi_{\theta}(u_t^{(i)} | x_t^{(i)}) \sum_{k \geq t} r(x_k^{(i)}, u_k^{(i)}) \right)$$

(caveat: this is not a rigorous argument we're presenting here)

# REINFORCE


Loop forever:

Generate episode  $x_0, u_0, r_0, x_1, u_1, r_1 \dots$  with  $\pi_\theta$

Loop for all  $t = 0, \dots, N - 1$ :

$$G_t \leftarrow \sum_{k=t}^N r_k$$

Cumulative tail reward,  
the tail “return”



$$\theta \leftarrow \theta + \alpha G_t \nabla_\theta \log \pi_\theta(u_t | x_t)$$

# Policy Gradient Recap

## Pros:

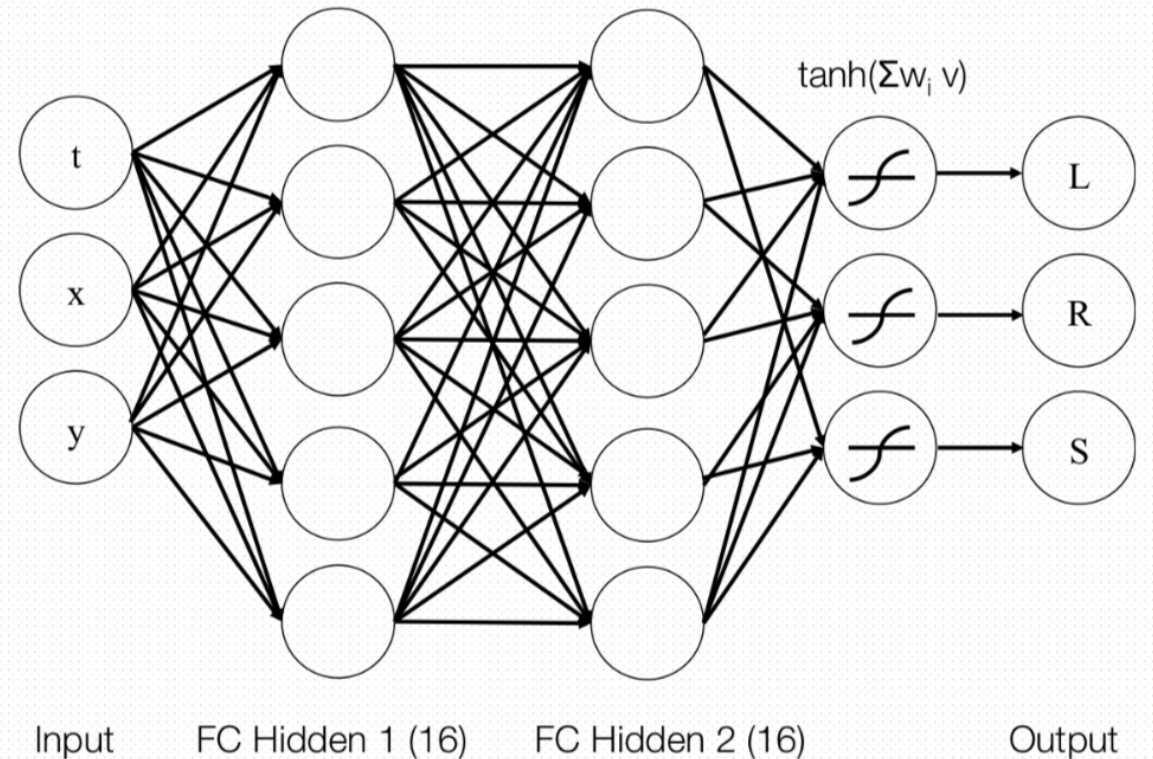
- Learns policy directly – can be more stable (less moving parts than  $Q$ -learning)
- Works for continuous action spaces (no need to “argmax”  $Q$ )
- Converges to local maximum of  $J(\theta)$

## Cons:

- Needs data from current policy to compute gradient – data inefficient
- Gradient estimates can be **very noisy**
  - Need to reduce variance of gradient estimator: baselines and actor-critic

# Deep Reinforcement Learning

- Deep  $Q$  learning:
  - Use neural network as  $Q$  function
  - Works in continuous state space domains
- Deep Policy Gradient:
  - Parameterize policy as deep neural network
  - Policy can act on high dimensional input, e.g., directly from visual feedback



# Learning from Experience

How to use trajectory data?

- **Model based approach: estimate  $T(x'|x, u)$ , then use model to plan**
- **Model free:**
  - Value based approach: estimate optimal value (or  $Q$ ) function from data
  - Policy based approach: use data to determine how to improve policy
  - Actor Critic approach: learn both a policy and a value/ $Q$  function

# Tabular model-based RL

- Discrete state/action space with stochastic transitions
- If model is known, can use value iteration/policy iteration/etc.
- Model unknown: want to build approximate model from observed transitions

# Tabular MBRL outline

- Assume initial policy
- Loop forever:
  - Take some number of actions, resulting in transition/reward data
  - Improve dynamics model
  - Choose actions/policy
- Approaches for action selection:
  - Dynamic programming/VI/etc. on approximate model
    - Expensive, gives optimal policy for model
  - Plan suboptimal sequence of actions via online control optimization

# Learning a tabular model from data

- States ( $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ )
- Actions ( $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m$ )
- Want to learn  $p(\mathbf{x}_i | \mathbf{x}_j, \mathbf{u}_k)$  for all  $i, j, k$
  
- Main strategies:
  - max likelihood point estimation
  - Bayesian approaches

# Learning a tabular model from data

- States ( $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$ )
- Actions ( $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m$ )
- Want to learn  $p(\mathbf{x}_i | \mathbf{x}_j, \mathbf{u}_k)$  for all  $i, j, k$
  
- Main strategies:
  - max likelihood point estimation
  - Bayesian approaches

# Max likelihood for tabular MBRL

- Categorical likelihood:  $p(\mathbf{x}_i | \mathbf{x}_j, \mathbf{u}_k, \boldsymbol{\theta}) = \boldsymbol{\theta}_{ijk}; \sum_i \boldsymbol{\theta}_{ijk} = 1$
- Assume data  $D = \{(\mathbf{x}, \mathbf{u}, \mathbf{x}')\}_{i=1}^d$
- Max likelihood:

$$\max_{\boldsymbol{\theta} \in \Theta} \sum_D \log p(\mathbf{x}' | \mathbf{x}, \mathbf{u}, \boldsymbol{\theta})$$

- Optimizing this gives the maximum likelihood estimate

$$\hat{\boldsymbol{\theta}}_{ijk} = \frac{N(\mathbf{x}_j, \mathbf{u}_k, \mathbf{x}_i)}{N(\mathbf{x}_j, \mathbf{u}_k)}$$

where  $N(\cdot, \cdot)$  is the empirical count

# Max likelihood for tabular MBRL

- $\theta_{ijk} = N(\mathbf{x}_j, \mathbf{u}_k, \mathbf{x}_i) / N(\mathbf{x}_j, \mathbf{u}_k)$
- Problem: what if  $N(\mathbf{x}_j, \mathbf{u}_k) = 0$ ?
  - For example, if we are starting with zero information, this model estimation scheme breaks
- Simple solution: start all of our counts at 1, i.e.,
  - Store  $N(\mathbf{x}_j, \mathbf{u}_k, \mathbf{x}_i)$ ; note that  $N(\mathbf{x}_j, \mathbf{u}_k) = \sum_{\mathbf{x}_i} N(\mathbf{x}_j, \mathbf{u}_k, \mathbf{x}_i)$
  - Replace  $N(\mathbf{x}_j, \mathbf{u}_k, \mathbf{x}_i)$  with  $N(\mathbf{x}_j, \mathbf{u}_k, \mathbf{x}_i) + 1$
  - Gives  $\theta_{ijk} = (N(\mathbf{x}_j, \mathbf{u}_k, \mathbf{x}_i) + 1) / (N(\mathbf{x}_j, \mathbf{u}_k) + n)$

# Why model-based?

- Advantages
  - Transitions give strong signal
  - Data efficiency, improved multi-task performance, generalization
- Weaknesses
  - Optimizing the wrong objective (i.e., not your ultimate task of optimizing reward)
  - May be very difficult/intractable for systems with high dimensional observations/states

# Challenges in RL for Robotics

Data-efficiency

Sim-to-real

Exploration

Reward design

# Next time

