Introduction to Reinforcement Learning

CS 285

Instructor: Sergey Levine
UC Berkeley
Definitions
Terminology & notation

- $o_t$ – observation
- $a_t$ – action
- $s_t$ – state
- $\pi_\theta(a_t|o_t)$ – policy
- $\pi_\theta(a_t|s_t)$ – policy (fully observed)

Markov property independent of $s_{t-1}$
Imitation Learning

\[ \mathbf{o}_t \quad \pi_\theta(\mathbf{a}_t | \mathbf{o}_t) \quad \mathbf{a}_t \]

Images: Bojarski et al. ‘16, NVIDIA
Reward functions

which action is better or worse?

\( r(s, a) \): reward function
tells us which states and actions are better

\( s, a, \pi(\pi \mid s) \), and \( p(s' \mid s, a) \) define

Markov decision process

high reward

low reward
Definitions

Markov chain
\[ \mathcal{M} = \{S, T\} \]

- **S** – state space
- **T** – transition operator

why “operator”?

- let \( \mu_t, i = p(s_t = i) \)
- let \( T_{i,j} = p(s_{t+1} = i|s_t = j) \)

\( \bar{\mu}_t \) is a vector of probabilities

then \( \bar{\mu}_{t+1} = T \bar{\mu}_t \)

---

Markov property
independent of \( s_{t-1} \)
Definitions

Markov decision process $\mathcal{M} = \{S, A, T, r\}$

$S$ – state space states $s \in S$ (discrete or continuous)

$A$ – action space actions $a \in A$ (discrete or continuous)

$T$ – transition operator (now a tensor!)

let $\mu_{t,j} = p(s_t = j)$

let $\xi_{t,k} = p(a_t = k)$

let $\mathcal{T}_{i,j,k} = p(s_{t+1} = i|s_t = j, a_t = k)$

$$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}$$
Definitions

Markov decision process \( \mathcal{M} = \{ \mathcal{S}, \mathcal{A}, \mathcal{T}, r \} \)

\( \mathcal{S} \) – state space
  states \( s \in \mathcal{S} \) (discrete or continuous)

\( \mathcal{A} \) – action space
  actions \( a \in \mathcal{A} \) (discrete or continuous)

\( \mathcal{T} \) – transition operator (now a tensor!)

\( r \) – reward function
  \( r : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \)
  \( r(s_t, a_t) \) – reward
Definitions

partially observed Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, \mathcal{r}\}$

$\mathcal{S} –$ state space $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A} –$ action space $a \in \mathcal{A}$ (discrete or continuous)

$\mathcal{O} –$ observation space $o \in \mathcal{O}$ (discrete or continuous)

$\mathcal{T} –$ transition operator (like before)

$\mathcal{E} –$ emission probability $p(o_t | s_t)$

$r –$ reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
The goal of reinforcement learning

\[ p_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1) \prod_{t=1}^T \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t) \]

\[ p_\theta(\tau) \]

\[ \theta^* = \arg \max_\theta E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \]
The goal of reinforcement learning

\[ p_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t) \]

Markov chain on \((s, a)\)
The goal of reinforcement learning

\[ p_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t) \]

Markov chain on \((s, a)\)

\[ p((s_{t+1}, a_{t+1})|(s_t, a_t)) = p(s_{t+1}|s_t, a_t)\pi_\theta(a_{t+1}|s_{t+1}) \]
Finite horizon case: state-action marginal

\[ \theta^* = \arg \max_\theta E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \]

\[ = \arg \max_\theta \sum_{t=1}^T E_{(s_t, a_t) \sim p_\theta(s_t, a_t)}[r(s_t, a_t)] \]

\[ p_\theta(s_t, a_t) \text{ state-action marginal} \]

\[ p((s_{t+1}, a_{t+1})|(s_t, a_t)) = p(s_{t+1}|s_t, a_t)\pi_\theta(a_{t+1}|s_{t+1}) \]
Infinite horizon case: stationary distribution

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

what if $T = \infty$?

does $p(s_t,a_t)$ converge to a stationary distribution?

$$\mu = \mathcal{T} \mu \quad \quad (\mathcal{T} - I)\mu = 0 \quad \quad \mu = p_{\theta}(s,a) \quad \text{stationary distribution}$$

$\mu$ is eigenvector of $\mathcal{T}$ with eigenvalue 1!
(always exists under some regularity conditions)

stationary = the same before and after transition

state-action transition operator

$$\begin{pmatrix} s_{t+1} \\ a_{t+1} \end{pmatrix} = \mathcal{T} \begin{pmatrix} s_t \\ a_t \end{pmatrix} \quad \begin{pmatrix} s_{t+k} \\ a_{t+k} \end{pmatrix} = \mathcal{T}^k \begin{pmatrix} s_t \\ a_t \end{pmatrix}$$
Infinite horizon case: stationary distribution

\[ \theta^* = \arg \max_{\theta} \frac{1}{T} \sum_{t=1}^{T} E_{(s_t, a_t) \sim p_\theta(s_t, a_t)}[r(s_t, a_t)] \rightarrow E_{(s, a) \sim p_\theta(s, a)}[r(s, a)] \]

(in the limit as \( T \rightarrow \infty \))

what if \( T = \infty \)?

does \( p(s_t, a_t) \) converge to a stationary distribution?

\[ \mu = \mathcal{T} \mu \]
\[ (\mathcal{T} - I) \mu = 0 \]
\[ \mu = p_\theta(s, a) \] stationary distribution

\( \mu \) is eigenvector of \( \mathcal{T} \) with eigenvalue 1!
(always exists under some regularity conditions)

stationary = the same before and after transition

**state-action** transition operator

\[
\begin{pmatrix}
  s_{t+1} \\
  a_{t+1}
\end{pmatrix}
= \mathcal{T}
\begin{pmatrix}
  s_t \\
  a_t
\end{pmatrix}
= \mathcal{T}^k
\begin{pmatrix}
  s_t \\
  a_t
\end{pmatrix}
\]
Expectations and stochastic systems

\[ \theta^* = \arg \max_\theta E_{(s,a) \sim p_\theta(s,a)}[r(s,a)] \]

infinite horizon case

\[ \theta^* = \arg \max_\theta \sum_{t=1}^{T} E_{(s_t,a_t) \sim p_\theta(s_t,a_t)}[r(s_t,a_t)] \]

finite horizon case

In RL, we almost always care about expectations

\[ r(x) - \text{not smooth} \]
\[ \pi_\theta(a = \text{fall}) = \theta \]
\[ E_{\pi_\theta}[r(x)] - \text{smooth in } \theta! \]
Algorithms
The anatomy of a reinforcement learning algorithm

- generate samples (i.e. run the policy)
- fit a model/estimate the return
- improve the policy
A simple example

generate samples (i.e. run the policy)

fit a model/ estimate the return

improve the policy

\[ J(\theta) = E_\pi \left[ \sum_t r_t \right] \approx \frac{1}{N} \sum_{i=1}^{N} \sum_t r_t^i \]

\[ \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \]
Another example: RL by backprop

- generate samples (i.e. run the policy)
- fit a model/estimate the return
- improve the policy

Learn $f_\phi$ such that $s_{t+1} \approx f_\phi(s_t, a_t)$

Backprop through $f_\phi$ and $r$ to train $\pi_\theta(s_t) = a_t$
Which parts are expensive?

\[ J(\theta) = E_\pi \left[ \sum_t r_t \right] \approx \frac{1}{N} \sum_{i=1}^{N} \sum_t r_t^i \]

real robot/car/power grid/whatever: 1x real time, until we invent time travel

MuJoCo simulator: up to 10000x real time

generate samples (i.e. run the policy)

fit a model/estimate the return

improve the policy

\[ \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \]

backprop through \( f_\phi \) and \( r \) to train \( \pi_\theta(s_t) = a_t \)

learn \( s_{t+1} \approx f_\phi(s_t, a_t) \)

expensive
trivial, fast
Value Functions
How do we deal with all these expectations?

\[
E_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=1}^{T} r(s_t, a_t) \right]
\]

\[
E_{s_1 \sim p(s_1)} \left[ E_{a_1 \sim \pi(a_1|s_1)} \left[ r(s_1, a_1) + E_{s_2 \sim p(s_2|s_1,a_1)} \left[ E_{a_2 \sim \pi(a_2|s_2)} \left[ r(s_2, a_2) + \ldots |s_2|s_1,a_1 \right] \right] \right] \right]
\]

what if we knew this part?

\[
Q(s_1, a_1) = r(s_1, a_1) + E_{s_2 \sim p(s_2|s_1,a_1)} \left[ E_{a_2 \sim \pi(a_2|s_2)} \left[ r(s_2, a_2) + \ldots |s_2|s_1,a_1 \right] \right]
\]

\[
E_{\tau \sim p_\theta(\tau)} \left[ \sum_{t=1}^{T} r(s_t, a_t) \right] = E_{s_1 \sim p(s_1)} \left[ E_{a_1 \sim \pi(a_1|s_1)} \left[ Q(s_1, a_1) |s_1 \right] \right]
\]

easy to modify \(\pi_\theta(a_1|s_1)\) if \(Q(s_1, a_1)\) is known!

example: \(\pi_\theta(a_1|s_1) = 1\) if \(a_1 = \arg \max_{a_1} Q(s_1, a_1)\)
Definition: Q-function

\[ Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta} [r(s_{t'}, a_{t'})|s_t, a_t] \]: total reward from taking \( a_t \) in \( s_t \)

Definition: value function

\[ V^\pi(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta} [r(s_{t'}, a_{t'})|s_t] \]: total reward from \( s_t \)

\[ V^\pi(s_t) = E_{a_t \sim \pi(a_t|s_t)} [Q^\pi(s_t, a_t)] \]

\[ E_{s_1 \sim p(s_1)} [V^\pi(s_1)] \] is the RL objective!
Using Q-functions and value functions

Idea 1: if we have policy $\pi$, and we know $Q^\pi(s, a)$, then we can improve $\pi$:

set $\pi'(a|s) = 1$ if $a = \text{arg max}_a Q^\pi(s, a)$

this policy is at least as good as $\pi$ (and probably better)!

and it doesn’t matter what $\pi$ is

Idea 2: compute gradient to increase probability of good actions $a$:

if $Q^\pi(s, a) > V^\pi(s)$, then $a$ is better than average  
(recall that $V^\pi(s) = E[Q^\pi(s, a)]$ under $\pi(a|s)$)

modify $\pi(a|s)$ to increase probability of $a$ if $Q^\pi(s, a) > V^\pi(s)$

These ideas are very important in RL; we’ll revisit them again and again!
The anatomy of a reinforcement learning algorithm

- generate samples (i.e. run the policy)
- fit a model/estimate the return
- improve the policy

This often uses Q-functions or value functions.
Types of Algorithms
Types of RL algorithms

\[ \theta^* = \arg \max_{\theta} E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \]

- Policy gradients: directly differentiate the above objective
- Value-based: estimate value function or Q-function of the optimal policy (no explicit policy)
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy
- Model-based RL: estimate the transition model, and then...
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
  - Something else
Model-based RL algorithms

generate samples (i.e. run the policy)

fit a model/ estimate the return

learn $p(s_{t+1}|s_t, a_t)$

improve the policy

a few options
Model-based RL algorithms

1. Just use the model to plan (no policy)
   • Trajectory optimization/optimal control (primarily in continuous spaces) – essentially backpropagation to optimize over actions
   • Discrete planning in discrete action spaces – e.g., Monte Carlo tree search
2. Backpropagate gradients into the policy
   • Requires some tricks to make it work
3. Use the model to learn a value function
   • Dynamic programming
   • Generate simulated experience for model-free learner
Value function based algorithms

- **generate samples**
  (i.e. run the policy)

- **fit a model**
  estimate the return

- **fit** $V(s)$ or $Q(s,a)$

- **improve the policy**

  $\pi(s) = \arg \max_a Q(s,a)$
Direct policy gradients

- generate samples (i.e. run the policy)
- fit a model/estimate the return
  \[ R_\tau = \sum_t r(s_t, a_t) \]
- evaluate returns
- improve the policy
  \[ \theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(s_t, a_t)] \]
Actor-critic: value functions + policy gradients

generate samples (i.e. run the policy)

fit a model/estimate the return

fit $V(s)$ or $Q(s, a)$

improve the policy

$\theta \leftarrow \theta + \alpha \nabla_{\theta} E[Q(s_t, a_t)]$
Tradeoffs Between Algorithms
Why so many RL algorithms?

- Different tradeoffs
  - Sample efficiency
  - Stability & ease of use
- Different assumptions
  - Stochastic or deterministic?
  - Continuous or discrete?
  - Episodic or infinite horizon?
- Different things are easy or hard in different settings
  - Easier to represent the policy?
  - Easier to represent the model?
Comparison: sample efficiency

• Sample efficiency = how many samples do we need to get a good policy?

• Most important question: is the algorithm off policy?
  • Off policy: able to improve the policy without generating new samples from that policy
  • On policy: each time the policy is changed, even a little bit, we need to generate new samples

\[
\theta \leftarrow \theta + \alpha \nabla \theta E\left[ \sum_t r(s_t, a_t) \right]
\]

just one gradient step
Comparison: sample efficiency

Why would we use a less efficient algorithm?
Wall clock time is not the same as efficiency!
Comparison: stability and ease of use

- Does it converge?
- And if it converges, to what?
- And does it converge every time?

Why is any of this even a question???

- Supervised learning: almost \textit{always} gradient descent
- Reinforcement learning: often \textit{not} gradient descent
  - Q-learning: fixed point iteration
  - Model-based RL: model is not optimized for expected reward
  - Policy gradient: \textit{is} gradient descent, but also often the least efficient!
Comparison: stability and ease of use

• Value function fitting
  • At best, minimizes error of fit (“Bellman error”)
    • Not the same as expected reward
  • At worst, doesn’t optimize anything
    • Many popular deep RL value fitting algorithms are not guaranteed to converge to anything in the nonlinear case

• Model-based RL
  • Model minimizes error of fit
    • This will converge
  • No guarantee that better model = better policy

• Policy gradient
  • The only one that actually performs gradient descent (ascent) on the true objective
Comparison: assumptions

• Common assumption #1: full observability
  • Generally assumed by value function fitting methods
  • Can be mitigated by adding recurrence

• Common assumption #2: episodic learning
  • Often assumed by pure policy gradient methods
  • Assumed by some model-based RL methods

• Common assumption #3: continuity or smoothness
  • Assumed by some continuous value function learning methods
  • Often assumed by some model-based RL methods
Examples of Algorithms
Examples of specific algorithms

• Value function fitting methods
  • Q-learning, DQN
  • Temporal difference learning
  • Fitted value iteration

• Policy gradient methods
  • REINFORCE
  • Natural policy gradient
  • Trust region policy optimization

• Actor-critic algorithms
  • Asynchronous advantage actor-critic (A3C)
  • Soft actor-critic (SAC)

• Model-based RL algorithms
  • Dyna
  • Guided policy search

We’ll learn about most of these in the next few weeks!
Example 1: Atari games with Q-functions

• Playing Atari with deep reinforcement learning, Mnih et al. ‘13
• Q-learning with convolutional neural networks
Example 2: robots and model-based RL

• End-to-end training of deep visuomotor policies, L.* , Finn* ’16

• Guided policy search (model-based RL) for image-based robotic manipulation
Example 3: walking with policy gradients

- High-dimensional continuous control with generalized advantage estimation, Schulman et al. ‘16
- Trust region policy optimization with value function approximation
Example 4: robotic grasping with Q-functions

• QT-Opt, Kalashnikov et al. ‘18
• Q-learning from images for real-world robotic grasping