RL Tutorial
CSC2515 Fall 2019

Adamo Young

University of Toronto

November 17, 2019
Markov Decision Process (MDP)

- $S_t \in S, A_t \in A, R_t \in \mathbb{R}$
- Timesteps: $t \in \mathbb{N}$ (discrete)
- Finite: $S, A$ both have finite number of elements (discrete)
- Fully observable: $S_t$ is known exactly
- Trajectory: sequence of state-action-reward $S_0, A_0, R_1, S_1, A_1, ...$
Markov Property

- Dynamics of the MDP are completely captured in the current state and the proposed action.
- States include all information about the past that make a difference for the future.
- \[ p(s', r|s, a) \triangleq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\} \]
MDP Generalizations

- Continuous timesteps
- Continuous states/actions
- Partially observable (POMDP): probability distribution over $S_t$
- Don’t worry about them for now
Episodic vs. Continuing Tasks

Episodic Task:
- $t = 0, 1, \ldots, T$
- $T$ (length of episode) can vary from episode to episode
- Example: Pacman

Continuing Task:
- No upper limit on $t$
- Example: learning to walk
Expected Return and Unified Task Notation

- $G_t$ is the expected return (expected sum of rewards)

\[
G_t \triangleq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}
\]

- $\gamma$ is the discount factor, $0 < \gamma \leq 1$

**Episodic Tasks:**
- $R_t = 0$ for all $t > T$, finite sum
- Often choose $\gamma = 1$
- Always converges since sum is finite

**Continuing Tasks:**
- Infinite sum
- $\gamma < 1$ (future rewards are valued less)
- If $\{R_t\}$ is bounded, the infinite sum converges

Either way, we can insure that $G_t$ is well-defined
Policies

- Policy $\pi$: a mapping from state to actions
  $$\pi(a|s) \triangleq Pr\{A_t = a|S_t = s\}$$
- Can be deterministic or probabilistic
- Example: deterministic wall-following maze navigation policy
  - Turn right if possible
  - If you can’t turn right, go straight if possible
  - If you can’t turn right or go straight, turn left if possible
  - Otherwise, turn around
Value Functions

- State-value function $v_\pi(s) \triangleq \mathbb{E}_\pi [G_t | S_t = s]$
  - Intuitively: how "good" a state $s$ is under policy $\pi$
  - $v_\pi(s) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right]$

- Action-value function $q_\pi(s, a) \triangleq \mathbb{E}_\pi [G_t | S_t = s, A_t = a]$
  - Intuitively: how "good" an action $a$ is in state $s$ under policy $\pi$
  - $q_\pi(s, a) = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right]$
Note: in this case, we are using $p(s'|s, a)$ to model the environment instead of $p(s', r|s, a)$.

This is because there is no ambiguity in reward $r$ given $s, a, s'$ (but in general there could be).

Even only two states and three possible actions can be pretty complicated to model.
Some Potentially Useful Value Identities

- Recall: $\pi(a|s) \triangleq Pr\{A_t = a|S_t = s\}$
- Recall: $v_{\pi}(s) \triangleq \mathbb{E}_\pi [G_t|S_t = s]$
- Recall: $q_{\pi}(s, a) \triangleq \mathbb{E}_\pi [G_t|S_t = s, A_t = a]$
- $v_{\pi}(s) = \mathbb{E}_\pi [q_{\pi}(s, a)|S_t = s] = \sum_{a \in A(s)} \pi(a|s)q_{\pi}(s, a)$
- $q_{\pi}(s, a) = \mathbb{E}_\pi [R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a]$
- $q_{\pi}(s, a) = \mathbb{E}_{p(s', r|s, a)} [R_{t+1} + \gamma v_{\pi}(S_{t+1})|S_t = s, A_t = a]$
- Common theme: value functions are recursive, expand one step to see a useful pattern
Solving RL Problems

Two main classes of solutions:

- **Tabular Solution Methods**
  - Provide exact solutions
  - Do not scale very well with search space

- **Approximate Solution Methods**
  - Use function approximators (i.e. neural networks) for value and/or policy functions
  - Policy Gradient Methods fall under this category
Introducing Policy Gradient

- Parameterize policy $\pi(a|s; \theta) \triangleq Pr\{A_t = a|S_t = s; \theta_t = \theta\}$
  - For example, $\pi$ could be a neural network with weights $\theta$
- Measure policy performance with scalar performance measure $J(\theta)$
- Maximize $J(\theta)$ with gradient ascent on $\theta$
- General update rule: $\theta_{t+1} = \theta_t + \alpha \hat{\nabla}_\theta J(\theta_t)$
  - $\alpha$ is step size
  - $\hat{\nabla}_\theta J(\theta_t)$ is an approximation to $\nabla_\theta J(\theta_t)$
- We are increasing the probability of trajectories with large rewards and decreasing the probability of trajectories with small rewards
- For simplicity, we’ll only consider episodic tasks for the remainder of the presentation
The Policy Gradient Theorem

- $J(\theta) \triangleq v_\pi(s_0)$, where $s_0$ is initial state of a trajectory and $\pi$ is a policy parameterized by $\theta$

- The Policy Gradient Theorem states that
  \[ \nabla J(\theta) = \nabla_\theta v_\pi(s_0) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla (\pi(a|s; \theta)) \]

- $\mu(s) \triangleq \frac{\eta(s)}{\sum_{s'} \eta(s')}$, the on-policy distribution under $\pi$
  - $\eta(s)$ is the expected number of visits to state $s$
  - $\mu(s)$ is the fraction of time spent in state $s$

- Constant of proportionality is the average length of an episode

- See p. 325 of Sutton RL Book (Second Edition) for concise proof
REINFORCE Update

\[ \nabla J(\theta) \propto \sum_s \mu(s) \sum_a q\pi(s, a) \nabla (\pi(a|s; \theta)) \]

\[ = \mathbb{E}_\pi \left[ \sum_a q\pi(S_t, a) \nabla \pi(a|S_t, \theta) \right] \]

\[ = \mathbb{E}_\pi \left[ \sum_a \pi(a|S_t, \theta) q\pi(S_t, a) \frac{\nabla \pi(a|S_t, \theta)}{\pi(a|S_t, \theta)} \right] \]

\[ = \mathbb{E}_\pi \left[ \mathbb{E}_\pi \left[ q\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \right] \]

\[ = \mathbb{E}_\pi \left[ q\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \]
REINFORCE Update (Continued)

\[ \nabla J(\theta) \propto \mathbb{E}_\pi \left[ q_\pi(S_t, A_t) \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \\
= \mathbb{E}_\pi \left[ \mathbb{E}_\pi \left[ G_t | S_t, A_t \right] \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \\
= \mathbb{E}_\pi \left[ \mathbb{E}_\pi \left[ G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right]|S_t, A_t \right] \\
= \mathbb{E}_\pi \left[ G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \right] \\
\]

Reinforce update: \( \theta_{t+1} = \theta_t + \alpha G_t \frac{\nabla \pi(A_t|S_t, \theta)}{\pi(A_t|S_t, \theta)} \)
Quick Aside: Monte Carlo

- Way of approximating an integral/expectation

\[ \mathbb{E}_{p(x)} [f(x)] = \int_{x \in \mathcal{X}} p(x) f(x) dx \]

- If \( p(x) \) and/or \( f(x) \) is complicated, the integral is difficult to solve analytically

- Simple idea: sample \( N \) values from the domain of the distribution \( \mathcal{X} \) and take the average

\[ \mathbb{E}_{p(x)} [f(x)] \approx \frac{1}{N} \sum_{i=1}^{N} p(x^{(i)}) f(x^{(i)}) \]

- The larger \( N \) is, the better the estimate
**REINFORCE Algorithm**

**REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for \( \pi_\ast \)**

- **Input:** a differentiable policy parameterization \( \pi(a|s, \theta) \)
- **Algorithm parameter:** step size \( \alpha > 0 \)
- **Initialize policy parameter** \( \theta \in \mathbb{R}^{d'} \) (e.g., to 0)

Loop forever (for each episode):
- Generate an episode \( S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T \), following \( \pi(\cdot|\cdot, \theta) \)
- Loop for each step of the episode \( t = 0, 1, \ldots, T - 1 \):
  - \( G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \)  \( (G_t) \)
  - \( \theta \leftarrow \theta + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t, \theta) \)

- Generate entire trajectories and compute \( G_t \) from it
- Gradient estimates only need to be proportional to the true gradient since constant of proportionality can be absorbed in the step size \( \alpha \)
REINFORCE as a Score Function Gradient Estimator

- We can rewrite \( J(\theta) \) in terms of an expectation over trajectories \( \tau \)
  \[ J(\theta) \triangleq \nu_\pi(s_0) = \mathbb{E}_\tau [G_0(\tau)] = \sum_\tau p(\tau|\pi_\theta) G_0(\tau) \]
  - \( p(\tau|\pi_\theta) \) is the probability of the trajectory under policy \( \pi_\theta \)
  - \( G_0(\tau) \) is the sum of the rewards given trajectory \( \tau \)

- Recall: \( \nabla_x \log f(x) = \frac{\nabla_x f(x)}{f(x)} \) by chain rule

- We can use the log-derivative trick to derive an expression for \( \hat{\nabla} J(\theta) \)

- In statistics, this estimator is called the Score Function Estimator
REINFORCE as a Score Function Gradient Estimator

\[
\nabla J(\theta) = \nabla \sum_{\tau} p(\tau | \pi_\theta) G_0(\tau)
\]

\[
= \sum_{\tau} \nabla p(\tau | \pi_\theta) G_0(\tau)
\]

\[
= \sum_{\tau} \frac{p(\tau | \pi_\theta)}{p(\tau | \pi_\theta)} \nabla p(\tau | \pi_\theta) G_0(\tau)
\]

\[
= \sum_{\tau} p(\tau | \pi_\theta) \nabla \log p(\tau | \pi_\theta) G_0(\tau)
\]

\[
= \mathbb{E}_{\tau} [\nabla \log p(\tau | \pi_\theta) G_0(\tau)]
\]

\[
\hat{\nabla J(\theta)} = \frac{1}{N} \sum_{i=1}^{N} \log p(\tau^{(i)} | \pi_\theta) G_0(\tau^{(i)})
\]

(Monte Carlo approximation)
Properties of REINFORCE/Score Function Gradient Estimator

- **Pros:**
  - Unbiased: \( E \left[ \nabla J(\theta) \right] = J(\theta) \)
  - Very general (reward does not need to be continuous/differentiable)

- **Cons:**
  - High variance (noisy)
  - Requires many sample trajectories to estimate gradient
Reducing REINFORCE Estimator Variance

- Recall: \( \nabla J(\theta) \propto \sum_s \mu(s) \sum_a q_\pi(s, a) \nabla (\pi(a|s; \theta)) \)

- Let \( b(s) \) be a baseline (some function of \( s \) that does not depend on \( \theta \))

\[
\sum_a b(s) \nabla \pi(a|s, \theta) = b(s) \nabla \sum_a \pi(a|s, \theta) = b(s) \nabla 1 = 0
\]

- Thus \( \nabla J(\theta) \propto \sum_s \mu(s) \sum_a (q_\pi(s, a) - b(s)) \nabla (\pi(a|s; \theta)) \)

- New update rule: \( \theta_{t+1} = \theta_t + \alpha (G_t - b(S_t)) \frac{\nabla \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \)

- Natural choice for \( b(s) \): an estimate of the state value \( \hat{v}(S_t, w) \) parameterized by weights \( w \)
  - Want to adjust \( \theta \) based on how much better (or worse) the path is than the average score of paths starting in the current state (\( \hat{v} \))

Adamo Young (University of Toronto)  RL Tutorial  November 17, 2019 21 / 28
REINFORCE with Baseline Algorithm

REINFORCE with Baseline (episodic), for estimating \( \pi_\theta \approx \pi_* \)

Input: a differentiable policy parameterization \( \pi(a|s, \theta) \)
Input: a differentiable state-value function parameterization \( \hat{v}(s,w) \)
Algorithm parameters: step sizes \( \alpha^\theta > 0, \alpha^w > 0 \)
Initialize policy parameter \( \theta \in \mathbb{R}^{d'} \) and state-value weights \( w \in \mathbb{R}^d \) (e.g., to 0)

Loop forever (for each episode):
  Generate an episode \( S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T \), following \( \pi(\cdot|\cdot, \theta) \)
  Loop for each step of the episode \( t = 0, 1, \ldots, T-1 \):
  \[
  G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1}R_k \\
  \delta \leftarrow G - \hat{v}(S_t, w) \\
  w \leftarrow w + \alpha^w \delta \nabla \hat{v}(S_t, w) \\
  \theta \leftarrow \theta + \alpha^\theta \gamma^t \delta \nabla \ln \pi(A_t|S_t, \theta)
  \]

- Two different step sizes: \( \alpha^\theta \) and \( \alpha^w \)
- \( \hat{v} \) is also estimated with Monte Carlo
REINFORCE with Baseline vs. without

Total reward on episode averaged over 100 runs

$G_0$

$\alpha = 2^{-13}$

$\alpha^b = 2^{-9}$, $\alpha^w = 2^{-6}$

$\nu^*(s_0)$

Episode
Extensions: Actor-Critic

- Use bootstrapping to update the state-value estimate $\hat{v}(s, w)$ for a state $s$ from the estimated values of subsequent states.
- Introduces bias but reduces variance.
- Can use one-step or $n$-step return (as opposed to full returns in REINFORCE).
One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$
Input: a differentiable state-value function parameterization $\hat{v}(s,w)$
Parameters: step sizes $\alpha^\theta > 0$, $\alpha^w > 0$

Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$ (e.g., to $0$)

Loop forever (for each episode):

1. Initialize $S$ (first state of episode)
2. $I \leftarrow 1$
3. Loop while $S$ is not terminal (for each time step):
   - $A \sim \pi(\cdot | S, \theta)$
   - Take action $A$, observe $S', R$
   - $\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$ \hspace{1cm} (if $S'$ is terminal, then $\hat{v}(S', w) = 0$)
   - $w \leftarrow w + \alpha^w \delta \nabla \hat{v}(S, w)$
   - $\theta \leftarrow \theta + \alpha^\theta I \delta \nabla \ln \pi(A|S, \theta)$
   - $I \leftarrow \gamma I$
   - $S \leftarrow S'$
OpenAI Gym

- Free python library of environments for reinforcement learning
- Easy to use, good for benchmarking
- Environments include:
  - Classic control problems
  - Atari games
  - 3D simulations (Mujoco)
  - Robotics
- Check out the docs and environments
REINFORCE bipedal walker demo

- OpenAI Gym Environment
- Train a 2D walker to navigate a landscape
- Different levels of difficulty
- Code taken from Michael Guerzhoy’s CSC411 course webpage
- Uses policy gradient with REINFORCE
- Note: requires tensorflow 1.x (not 2.x)
References

- Sutton and Barto, "Reinforcement Learning: An Introduction (Second Edition)", 2018
- Xuchan (Jenny) Bao, Intro to RL and Policy Gradient CSC421 Tutorial, 2019 (link)
- Michael Guerzhoy, CSC411 Project 4: Reinforcement Learning with Policy Gradients, 2017 (link)