Quiz 1 Recitation

Topics (Lectures 1-8):

1. Introduction to Reinforcement and Representation Learning
2. Multi-Arm Bandits
3. MDPs, Value and Policy Iteration
4. Monte Carlo Learning, Temporal Difference Learning, Monte Carlo Tree Search
5. Function Approximation, Deep Q learning
6. Policy gradients, REINFORCE, Actor-Critic methods

***Note this is not an exhaustive list. Anything covered in lectures in fair game.**

Bandits

- You have one state with k actions
- Each action gets you a reward
- Want to maximize reward in least amount of time
  - How to sample actions to do this efficiently?

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}},$$

Greedy action: $A_t \doteq \arg\max_a Q_t(a),$

Epsilon-greedy bandits

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Upper confidence bound

- the square-root term is a measure of the uncertainty or variance in the estimate of a's value
- As Nt(a) increases the uncertainty term decreases.
- On the other hand, each time an action other than a is selected, t increases but Nt(a) does not, causing the uncertainty to increase
- The use of the natural logarithm means that the increases get smaller over time, but are unbounded

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right],$$

Optimistic Initial Values

- Set initial Q values much higher than the reward
- Encourage some exploration initially
- Whichever actions are initially selected, the reward is less than the starting estimates; the learner switches to other actions, being “disappointed” with the rewards it is receiving. The result is that all actions are tried several times before the value estimates converge. The system does a fair amount of exploration even if greedy actions are selected all the time.

Gradient Bandit Algorithms

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a),$$

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t,$$

$\bar{R}_t \in \mathbb{R}$ is the average of all the rewards

MDPs

A **Finite** Markov Decision Process is a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- p is one step dynamics function
- r is a reward function
- γ is a discount factor $\gamma \in [0,1]$

policy: $\pi(a|s)$

* denotes optimal $q_*(s,a) \doteq \max_\pi q_\pi(s,a)$

State-value function: $\mathrm{v}_\pi(s) = \mathbb{E}[G_t \,|\, S_t = s]$

Action-value function: $\mathrm{q}_\pi(s,a) = \mathbb{E}[G_t \,|\, S_t = s, A_t = a]$

Policy evaluation

- Find value function for a given policy
- Converges to unique true value function in limit
- In practice, use iterative policy evaluation (below) - stop when max delta below a threshold
- Can update value function “in place” or use two copies

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Bellman Backup & Contraction Mapping Theorem

- value function v_pi is the unique solution to its Bellman equation.
- Bellman backup operator is gamma-contraction

$$
\begin{aligned}
v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_k(s') \Big],
\end{aligned}
$$

- Define the Bellman expectation backup operator

$$F^\pi(\mathrm{v}) = r^\pi + \gamma T^\pi \mathrm{v}$$

Contraction Mapping Theorem

An operator F on a normed vector space $\mathcal{X}$ is a **γ-contraction**, for $0 < \gamma < 1$ provided for all $x, y \in \mathcal{X}$:

$$\|F(x) - F(y)\| \leq \gamma \|x - y\|$$

Theorem (Contraction mapping)
For a γ-contraction F in a complete normed vector space $\mathcal{X}$:

- F converges to a unique fixed point in $\mathcal{X}$,
- at a linear convergence rate γ.

Policy improvement

- Given value function for current policy, do one-step look-ahead and check if it is better to change policy to new action in each state
- Strictly improving except when policy is already optimal

3. Policy Improvement
   $policy\text{-}stable \leftarrow true$
   For each $s \in \mathcal{S}$:
      $old\text{-}action \leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
      If $old\text{-}action \neq \pi(s)$, then $policy\text{-}stable \leftarrow false$
   If $policy\text{-}stable$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Value iteration

- Combine policy evaluation and policy iteration in each state sweep
- Effectively combines one sweep of policy evaluation and one sweep of policy improvement.
- Often much faster convergence than policy iteration

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Monte Carlo

- We do not assume complete knowledge of the environment.
- Monte Carlo methods require only experience—sample sequences of states, actions, and rewards from actual or simulated interaction with an environment.
- Average returns observed after visits to a state
- Does not depend on estimates of other states (no bootstrapping)
- Get rid of exploring starts:
  - **on-policy** (e.g. epsilon greedy),
  - **off policy** (i.e. importance sampling)

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow$ average$(Returns(S_t))$

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
  $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
  $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
  Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow$ average$(Returns(S_t, A_t))$
      $\pi(S_t) \leftarrow \text{argmax}_a\, Q(S_t, a)$

Temporal Difference

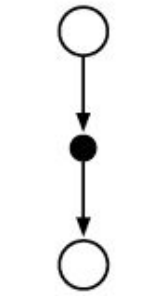
- We do not assume complete knowledge of the environment.
- TD methods require only experience—sample sequences of states, actions, and rewards from actual or simulated interaction with an environment.
- Bootstrap from estimates of other states
- Monte Carlo need to wait until end of episode, while TD(0) methods only need to wait one step
- **On-policy** (i.e. SARSA), **off-policy** (i.e. Q-learning)

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

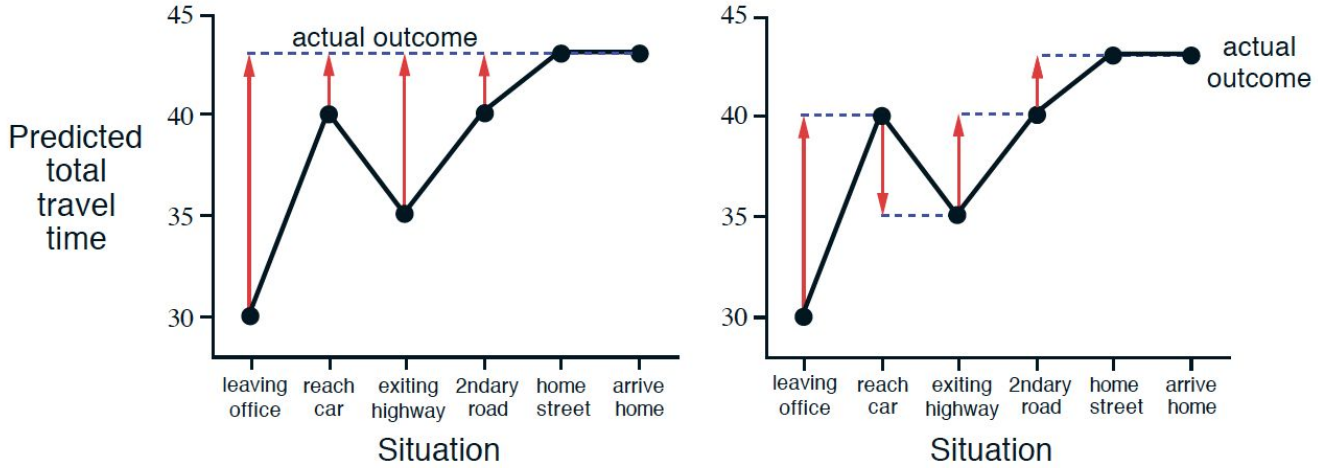
Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

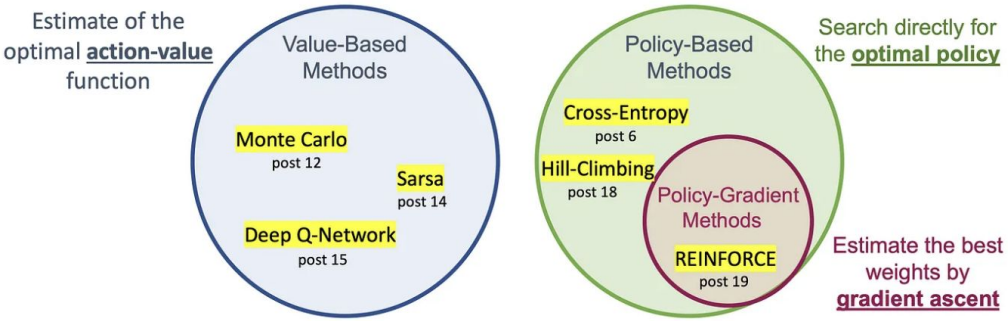
Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal



Monte Carlo vs Temporal Difference



Policy-Gradient Methods



- Value-based methods: learn a value function (an optimal value function leads to an optimal policy)
  - Goal: minimize the loss between the predicted and target value
  - Policy is implicit as it is generated directly from the value function (e.g. eps-greedy from Q-function)
  - Examples: Monte Carlo, DQN, SARSA
- Policy-based methods: learn to approximate optimal policy directly (without learning a value function)
  - Parameterize the policy, e.g. using a neural network
  - Policy outputs a probability distribution over actions (stochastic policy)
  - Goal: maximize the performance of the parameterized policy using gradient ascent

REINFORCE: Algorithm

REINFORCE, or Monte Carlo policy-gradient, uses an estimated return from an entire episode to update the policy parameter θ.

In a loop,

1. Use the policy π_θ to collect episode τ
2. Use the episode to estimate the gradient g = $\nabla\theta J(\theta)$

$$\nabla_\theta J(\theta) \approx \hat{g} = \sum_{t=0} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$$

Estimation of the gradient (given we use only one trajectory to estimate the gradient)

Probability of the agent to select action at from state st given our policy

Direction of the steepest increase of the (log) probability of selecting action at from state st

Cumulative return

3. Update the weights of the policy: θ ← θ + αg

REINFORCE - Baseline: Algorithm

Initialize policy parameter θ, baseline b

for iteration$=1, 2, \cdots$ **do**

Collect a set of trajectories by executing the current policy

At each timestep t in each trajectory τ^i, compute

Return $G_t^i = \sum_{t'=t}^{T-1} r_{t'}^i$, and

Advantage estimate $\hat{A}_t^i = G_t^i - b(s_t)$.

Re-fit the baseline, by minimizing $\sum_i \sum_t ||b(s_t) - G_t^i||^2$,

Update the policy, using a policy gradient estimate $\hat{g}$,

Which is a sum of terms $\nabla_\theta \log \pi(a_t|s_t, \theta)\hat{A}_t$.

Advantage Actor Critic: Algorithm

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\, \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\, \delta\, \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}}\, I\, \delta\, \nabla \ln \pi(A|S, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Policy-based methods, pros and cons

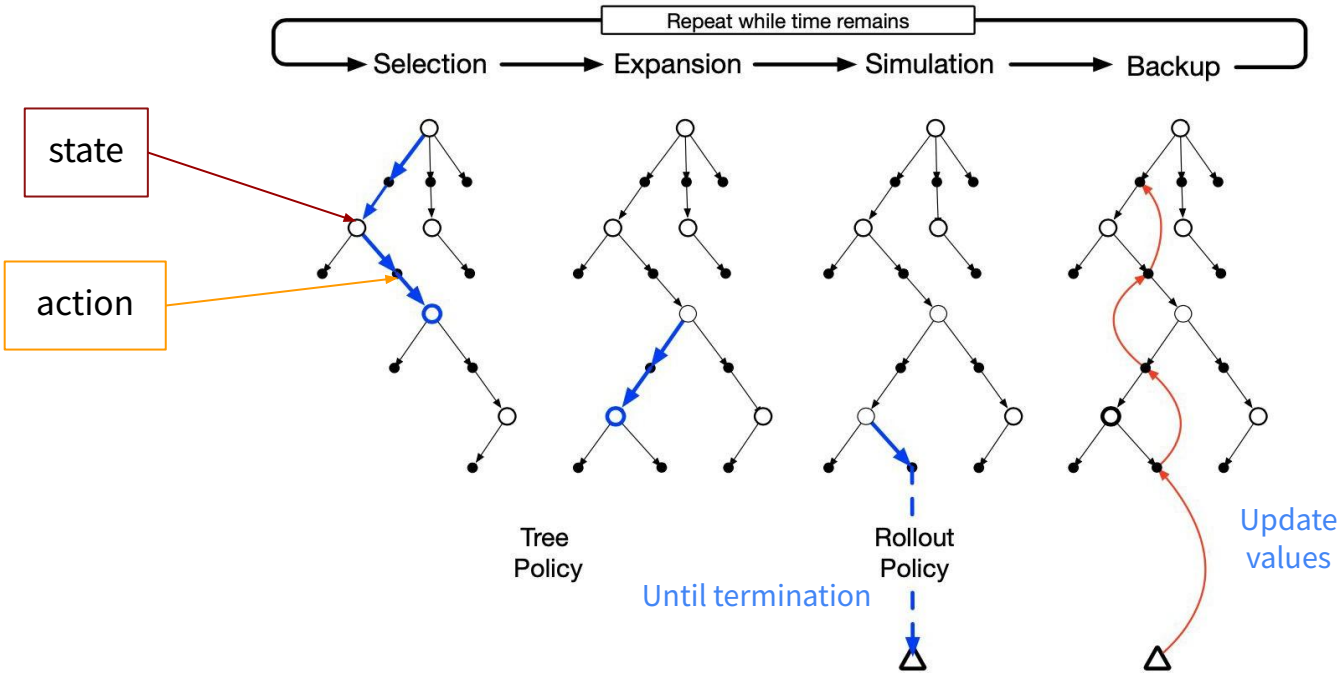
Pros

- We can estimate the policy directly without storing additional data
- Policy-gradient methods can learn a stochastic policy
  - We don’t need to implement an exploration/exploitation trade-off by hand
- More effective in high-dimensional action spaces and continuous action spaces
- Better convergence properties

Cons

- Converges to a local maximum sometimes
- Slower, step-by-step: it can take longer to train (inefficient)
- Gradient estimate is very noisy: there is a possibility that the collected trajectory may not be representative of the policy

Monte Carlo Tree Search



Deep Q-Network

Algorithm 4 DQN

procedure DQN
  Initialize network Q_ω *and* Q_{target} *as a clone of* Q_ω
  Initialize replay buffer R *and burn in with trajectories followed by random policy*
  Initialize $c = 0$
  repeat for E **training episodes:**
    Initialize S_0
    for $t = 0, 1, \ldots, T-1$:
      $$a_t = \begin{cases} \arg\max_a Q_\omega(s_t, a) & \text{with probability } 1-\epsilon \\ \text{Random action} & \text{otherwise} \end{cases}$$
      Take a_t *and observe* r_t, s_{t+1}
      Store (s_t, a_t, r_t, s_{t+1}) *in* R
      Sample minibatch of (s_i, a_i, r_i, s_{i+1}) *with size* N *from* R
      $$y_i = \begin{cases} r_i & s_{i+1} \text{ is terminal} \\ r_i + \gamma \max_a Q_{\text{target}}(s_{i+1}, a) & \text{otherwise} \end{cases}$$
      $L(\omega) = \frac{1}{N} \sum_{i=0}^{N-1} (y_i - Q_\omega(s_i, a_i))^2$
      Update Q_ω *using* `Adam` $(\nabla_\omega L(\omega))$
      $c = c + 1$
      Replace Q_{target} *with current* Q_ω *if* $c \ \% \ 50 = 0$
end procedure

Big Picture Table

Method	On/Off Policy?	Bootstraps?
Monte Carlo Methods	On*	N
SARSA	On*	Y
Expected SARSA	Either	Y
Q-Learning	Off	Y
REINFORCE	On*	N
Actor Critic, A2C	On*	Y

* can be made off policy with importance sampling