Topics:
• Reinforcement Learning Part 1
  • Markov Decision Processes
  • Value Iteration
Admin

- HW4 due **April 10th**
- See OH on Attention/Seq2seq and HW4
Reinforcement Learning
Introduction
Reinforcement Learning

- Evaluative feedback in the form of reward
- No supervision on the right action

Unsupervised Learning

- Input: \( \{X\} \)
- Learning output: \( P(x) \)
- Example: Clustering, density estimation, etc.

Supervised Learning

- Train Input: \( \{X, Y\} \)
- Learning output: \( f : X \to Y, P(y|x) \)
- e.g. classification

Types of Machine Learning
**RL**: Sequential decision making in an environment with evaluative feedback.

- **Environment** may be unknown, non-linear, stochastic and complex.
- **Agent** learns a **policy** to map states of the environments to actions.
  - Seeking to maximize cumulative reward in the long run.
Signature Challenges in Reinforcement Learning

- Evaluative feedback: Need trial and error to find the right action
- Delayed feedback: Actions may not lead to immediate reward
- Non-stationarity: Data distribution of visited states changes when the policy changes
- Fleeting nature of time and online data

Slide adapted from: Richard Sutton
**Robot Locomotion**

- **Objective**: Make the robot move forward
- **State**: Angle and position of the joints
- **Action**: Torques applied on joints
- **Reward**: +1 at each time step upright and moving forward

Figures copyright John Schulman et al., 2016. Reproduced with permission.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
**Atari Games**

- **Objective**: Complete the game with the highest score
- **State**: Raw pixel inputs of the game state
- **Action**: Game controls e.g. Left, Right, Up, Down
- **Reward**: Score increase/decrease at each time step

---

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Examples of RL tasks

---

**Go**

- **Objective**: Defeat opponent
- **State**: Board pieces
- **Action**: Where to put next piece down
- **Reward**: +1 if win at the end of game, 0 otherwise

---

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Markov Decision Processes
MDPs: Theoretical framework underlying RL
Markov Decision Processes (MDPs): Theoretical framework underlying RL

An MDP is defined as a tuple \((S, A, R, T, \gamma)\)

\(S\) : Set of possible states

\(A\) : Set of possible actions

\(R(s, a, s')\) : Distribution of reward

\(T(s, a, s')\) : Transition probability distribution, also written as \(p(s'|s, a)\)

\(\gamma\) : Discount factor
**MDPs**: Theoretical framework underlying RL

An MDP is defined as a tuple \((S, A, R, T, \gamma)\)

- \(S\) : Set of possible states
- \(A\) : Set of possible actions
- \(R(s, a, s')\) : Distribution of reward
- \(T(s, a, s')\) : Transition probability distribution, also written as \(p(s'|s,a)\)
- \(\gamma\) : Discount factor

**Interaction trajectory**: \(\ldots, s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, \ldots\)
**MDPs**: Theoretical framework underlying RL

An MDP is defined as a tuple \( (S, A, R, T, \gamma) \)
- \( S \): Set of possible states
- \( A \): Set of possible actions
- \( R(s, a, s') \): Distribution of reward
- \( T(s, a, s') \): Transition probability distribution, also written as \( p(s'|s, a) \)
- \( \gamma \): Discount factor

**Interaction trajectory**: \( \ldots S_t, A_t, r_{t+1}, S_{t+1}, A_{t+1}, r_{t+2}, S_{t+2}, \ldots \)

**Markov property**: Current state completely characterizes state of the environment

**Assumption**: Most recent observation is a sufficient statistic of history

\[
p(S_{t+1} = s'|S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, \ldots S_0 = s_0) = p(S_{t+1} = s'|S_t = s_t, A_t = a_t)
\]
MDP Variations

**Fully observed MDP**
- Agent receives the true state $s_t$ at time $t$
- Example: Chess, Go

**Partially observed MDP**
- Agent perceives its own partial observation $o_t$ of the state $s_t$ at time $t$, using past states e.g. with an RNN
- Example: Poker, First-person games (e.g. Doom)

Source: https://github.com/mwydmuch/ViZDoom
Fully observed MDP

- Agent receives the true state $s_t$ at time $t$
- Example: Chess, Go

Partially observed MDP

- Agent perceives its own partial observation $o_t$ of the state $s_t$ at time $t$, using past information

We will assume **fully observed MDPs** for this lecture

Source: https://github.com/mwydmuch/ViZDoom
In **Reinforcement Learning**, we assume an underlying **MDP** with unknown:

- Transition probability distribution $T$
- Reward distribution $R$

**MDP**

$(S, A, R, T, \gamma)$

**MDPs in the context of RL**
In Reinforcement Learning, we assume an underlying MDP with unknown:
- Transition probability distribution $T$
- Reward distribution $R$

- Evaluative feedback comes into play, trial and error necessary
In Reinforcement Learning, we assume an underlying MDP with unknown:

- Transition probability distribution $T$
- Reward distribution $R$

Evaluative feedback comes into play, trial and error necessary

For this lecture, assume that we know the true reward and transition distribution and look at algorithms for solving MDPs i.e. finding the best policy

- Rewards known everywhere, no evaluative feedback
- Know how the world works i.e. all transitions
A Grid World MDP

- Agent lives in a 2D grid environment
- State: Agent's 2D coordinates
- Actions: N, E, S, W
- Rewards: +1/-1 at absorbing states
- Walls block agent's path
- Actions to not always go as planned
  - 20% chance that agent drifts one cell left or right of direction of motion (except when blocked by wall).

Figure credits: Pieter Abbeel
Agent lives in a 2D grid environment

Figure credits: Pieter Abbeel
Agent lives in a 2D grid environment

- State: Agent’s 2D coordinates
- Actions: N, E, S, W
- Rewards: +1/-1 at absorbing states
Agent lives in a 2D grid environment

State: Agent’s 2D coordinates
Actions: N, E, S, W
Rewards: +1/-1 at absorbing states

Walls block agent’s path
Actions to not always go as planned
20% chance that agent drifts one cell left or right of direction of motion (except when blocked by wall).
Solving MDPs by finding the **best/optimal policy**

- **Solving MDPs by finding the best/optimal policy**

- Formally, a policy is a mapping from states to actions.
  - Deterministic
  - Stochastic

- What is a good policy?
  - Maximize current reward?
  - Sum of all future rewards?
  - Discounted sum of future rewards!
Solving MDPs by finding the best/optimal policy

Formally, a policy is a mapping from states to actions.

What is a good policy?

- Maximize current reward?
- Sum of all future rewards?
- Discounted sum of future rewards!

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
</tbody>
</table>
- Solving MDPs by finding the **best/optimal policy**

- Formally, a **policy** is a mapping from states to actions
  - Deterministic $\pi(s) = a$

$$n = |S|$$
$$m = |A|$$

? \[ \pi \]
Solving MDPs by finding the **best/optimal policy**

- Formally, a **policy** is a mapping from states to actions
  - Deterministic $\pi(s) = a$

$$n = |S|$$
$$m = |A|$$

$$\pi$$
Solving MDPs by finding the **best/optimal policy**

Formally, a **policy** is a mapping from states to actions

- **Deterministic** $\pi(s) = a$
- **Stochastic** $\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$

$n = |S|$

$m = |A|$
Solving MDPs by finding the **best/optimal policy**

Formally, a **policy** is a mapping from states to actions

- Deterministic $\pi(s) = a$
- Stochastic $\pi(a|s) = P(A_t = a|S_t = s)$

$n = |S|$
$m = |A|$
Solving MDPs by finding the **best/optimal policy**

Formally, a **policy** is a mapping from states to actions

- Deterministic \( \pi(s) = a \)
- Stochastic \( \pi(a | s) = \mathbb{P}(A_t = a | S_t = s) \)

What is a good policy?

- Maximize current reward? Sum of all future rewards?
- **Discounted sum of future rewards!**
  - Discount factor: \( \gamma \)

\[
\begin{align*}
\text{Worth Now} & = 1 \\
\text{Worth Next Step} & = \gamma \\
\text{Worth In Two Steps} & = \gamma^2
\end{align*}
\]
Formally, the optimal policy is defined as:

$$
\pi^* = \arg \max_\pi \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi \right]
$$
Formally, the **optimal policy** is defined as:

\[
\pi^* = \arg \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi \right]
\]
Formally, the **optimal policy** is defined as:

\[
\pi^* = \arg \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | \pi \right]
\]
Formally, the **optimal policy** is defined as:

\[
\pi^* = \arg \max_{\pi} \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \middle| \pi \right]
\]

where

- \( s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t) \)

Expectation over initial state, actions from policy, next states from transition distribution.
Some optimal policies for three different grid world MDPs (gamma=0.99)
- Varying reward for non-absorbing states (states other than +1/-1)

![Grid World MDPs](image)

- R(s) = -0.03
- R(s) = -0.4
- R(s) = -2.0

Image Credit: Byron Boots, CS 7641
A value function is a prediction of discounted sum of future reward.
A **value function** is a prediction of discounted sum of future reward

**State** value function / \( V \)-function / \( V : S \rightarrow \mathbb{R} \)
- **Value function** is a prediction of discounted sum of future reward.

- **State value function / V-function /** \( V : S \rightarrow \mathbb{R} \)
  - How good is this state?
  - Am I likely to win/lose the game from this state?
A value function is a prediction of discounted sum of future reward

State value function / V-function / $V : S \rightarrow \mathbb{R}$
- How good is this state?
- Am I likely to win/lose the game from this state?

State-Action value function / Q-function / $Q : S \times A \rightarrow \mathbb{R}$
A **value function** is a prediction of discounted sum of future reward.

**State value function / V-function** / $V : S \to \mathbb{R}$
- How good is this state?
- Am I likely to win/lose the game from this state?

**State-Action value function / Q-function** / $Q : S \times A \to \mathbb{R}$
- How good is this state-action pair?
- In this state, what is the impact of this action on my future?
For a policy that produces a trajectory sample \((s_0, a_0, s_1, a_1, s_2 \cdots)\)
For a policy that produces a trajectory sample \( (s_0, a_0, s_1, a_1, s_2 \cdots) \)

The **V-function** of the policy at state \( s \), is the expected cumulative reward from state \( s \):

\[
V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi \right]
\]
For a policy that produces a trajectory sample \((s_0, a_0, s_1, a_1, s_2 \cdots)\)

The **V-function** of the policy at state \(s\), is the expected cumulative reward from state \(s\):

\[
V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi \right]
\]
For a policy that produces a trajectory sample \((s_0, a_0, s_1, a_1, s_2 \cdots)\)

The **V-function** of the policy at state \(s\), is the expected cumulative reward from state \(s\):

\[
V^\pi(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi \right]
\]

\(s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)\)
For a policy that produces a trajectory sample \((s_0, a_0, s_1, a_1, s_2 \cdots)\)

The **Q-function** of the policy at state \(s\) and action \(a\), is the expected cumulative reward upon taking action \(a\) in state \(s\) (and following policy thereafter):
For a policy that produces a trajectory sample \((s_0, a_0, s_1, a_1, s_2 \cdots)\)

The **Q-function** of the policy at state \(s\) and action \(a\), is the expected cumulative reward upon taking action \(a\) in state \(s\) (and following policy thereafter):

\[
Q^\pi(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]
\]

\[
s_0 \sim p(s_0), a_t \sim \pi(\cdot| s_t), s_{t+1} \sim p(\cdot| s_t, a_t)
\]
The V and Q functions corresponding to the optimal policy $\pi^*$

$$V^*(s) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi^* \right]$$

$$Q^*(s, a) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi^* \right]$$
Recursive Bellman expansion (from definition of $Q$)

$Q^*(s, a) = \mathbb{E}_{a_t \sim \pi^*(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)} \left[ \sum_{t \geq 0} \gamma^t r(s_t, a_t) \mid s_0 = s, a_0 = a \right]$ 

(Expected) return from $t = 0$

$= \gamma^0 r(s, a) + \mathbb{E}_{s' \sim p(\cdot | s, a)} \left[ \gamma \mathbb{E}_{a_t \sim \pi^*(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)} \left[ \sum_{t \geq 1} \gamma^{t-1} r(s_t, a_t) \mid s_1 = s' \right] \right]$ 

(Reward at $t = 0$) + $\gamma$ * (Return from expected state at $t=1$)

$= r(s, a) + \gamma \mathbb{E}_{s' \sim p(s' | s, a)} [V^*(s')]$ 

$= \mathbb{E}_{s' \sim p(s' | s, a)} [r(s, a) + \gamma V^*(s')]$
Equations relating optimal quantities

\[
V^*(s) = \max_a Q^*(s, a) \quad \pi^*(s) = \arg\max_a Q^*(s, a)
\]

Recursive Bellman optimality equation

\[
Q^*(s, a) = \mathbb{E}_{s' \sim p(s' | s, a)} [r(s, a) + \gamma V^*(s)]
= \sum_{s'} p(s' | s, a) [r(s, a) + \gamma V^*(s)]
= \sum_{s'} p(s' | s, a) [r(s, a) + \gamma \max_{a'} Q^*(s', a')]
\]
Equations relating optimal quantities

\[
V^*(s) = \max_a Q^*(s, a) \quad \quad \quad \pi^*(s) = \arg \max_a Q^*(s, a)
\]

Recursive Bellman optimality equation

\[
Q^*(s, a) = \mathbb{E}_{s' \sim p(s'|s,a)} \left[ r(s, a) + \gamma V^*(s') \right] \\
= \sum_{s'} p(s'|s,a) \left[ r(s, a) + \gamma V^*(s') \right] \\
= \sum_{s'} p(s'|s,a) \left[ r(s, a) + \gamma \max_{a'} Q^*(s', a') \right]
\]

\[
V^*(s) = \max_a \sum_{s'} p(s'|s,a) \left[ r(s, a) + \gamma V^*(s') \right]
\]
Based on the **bellman optimality equation**

\[ V^*(s) = \max_a \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma V^*(s') \right] \]

**Algorithm**

- Initialize values of all states
- While not converged:
  - For each state: 
    \[ V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s, a) \left[ r(s, a) + \gamma V^i(s') \right] \]
  - Repeat until convergence (no change in values)

\[ V^0 \rightarrow V^1 \rightarrow V^2 \rightarrow \ldots \rightarrow V^i \rightarrow \ldots \rightarrow V^* \]

**Time complexity per iteration** \( O(|S|^2|A|) \)
• A robot car wants to travel far, quickly
• Three states: **Cool**, **Warm**, **Overheated**
• Two actions: **Slow**, **Fast**
• Going faster gets double reward

---

Example: Racing

Slide Credit: http://ai.berkeley.edu
Racing Search Tree
Racing Search Tree

Assume no discount!

\[ V_{k+1}(s) \leftarrow \max_{a} \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right] \]
Racing Search Tree

\[ V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right] \]

\[ V_1(s_0) = \max \left( 1.0 \cdot \left[ +1 + V(s) \right] \right) = 1 \]

\[ F_{\text{Fast}} \]

\[ 0.5 \left[ +2 + V(s) \right] + 0.5 \left( +2 \cdot V(s_0) \right) \]

\[ = 2 \]
$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right]$

$V_2(s_0) = \max \left[ \left. \begin{array}{c} \text{Slow} \\ \text{Fast} \end{array} \right| 1 \cdot (+1 + V_1(s_0)) \right]$

$= 3.5$

Assume no discount!

Slide Credit: http://ai.berkeley.edu
Value Iteration Update:

\[ V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a) \left[ r(s,a) + \gamma V^i(s') \right] \]

Q-Iteration Update:

\[ Q^{i+1}(s, a) \leftarrow \sum_{s'} p(s'|s,a) \left[ r(s,a) + \gamma \max_{a'} Q^i(s', a') \right] \]

The algorithm is same as value iteration, but it loops over actions as well as states
For **Value Iteration**:  

Theorem: will converge to unique optimal values  
Basic idea: approximations get refined towards optimal values  
Policy may converge long before values do  

\[
\text{Time complexity per iteration } O(|S|^2 |A|)
\]

**Feasible for:**  
- 3x4 Grid world?  
- Chess/Go?  
- Atari Games with integer image pixel values [0, 255] of size 16x16 as state?
Summary: MDP Algorithms

Value Iteration
- Bellman update to state value estimates

Q-Value Iteration
- Bellman update to (state, action) value estimates
Reinforcement Learning, Deep RL
Recall RL assumptions:

- $T(s, a, s')$ unknown, how actions affect the environment.
- $R(s, a, s')$ unknown, what/when are the good actions?

But, we can learn by trial and error.
- Gather experience (data) by performing actions.
- Approximate unknown quantities from data.

**Reinforcement Learning**
- Old Dynamic Programming Demo
  - https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

- RL Demo
  - https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

Learning Based Methods: RL

Slide credit: Dhruv Batra
Q-Learning

• We’d like to do Q-value updates to each Q-state:

\[ Q_{k+1}(s, a) \leftarrow \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right] \]

  - But can’t compute this update without knowing T, R

• Instead, compute average as we go
  - Receive a sample transition \((s, a, r, s')\)
  - This sample suggests

\[ Q(s, a) \approx r + \gamma \max_{a'} Q(s', a') \]

  - But we want to average over results from \((s, a)\)
  - So keep a running average

\[ Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha) \left[ r + \gamma \max_{a'} Q(s', a') \right] \]
Q-Learning Properties

• Amazing result: Q-learning converges to optimal policy -- even if you’re acting suboptimally!

• This is called off-policy learning

• Caveats:
  – You have to explore enough
  – You have to eventually make the learning rate small enough
  – ... but not decrease it too quickly
  – Basically, in the limit, it doesn’t matter how you select action

Slide Credit: http://ai.berkeley.edu