A Brief Introduction to Reinforcement Learning

Heejin Jeong and Steven Chen
heavily modified by Lyle Ungar
What is Reinforcement Learning?
Markov Decision Process (MDP)
Dynamic Programming
Exploration-Exploitation Trade-off
Monte Carlo Methods
Q-Learning
Recent Achievements in RL
  - AlphaGo
WHAT IS REINFORCEMENT LEARNING?
Reinforcement Learning Idea

Learn a function (policy) that maximizes an agent’s long-term reward in an environment

From Sutton Reinforcement Learning: An Introduction (2016 draft)
Tic-Tac-Toe Example

- **State**
  - Current board position

- **Action**
  - move
  - Possible actions depend on state

- **Policy**
  - Given state, what action to take

- **Reward**
  - -1/0/1 for lose/tie/win
  - 0 for all intermediate states

- Exploration policy
  - Search to find out what happens and how good each state is.

- Exploitation policy
  - Use what was learned to do well.

Sutton & Barto, Reinforcement Learning
RL Challenges

- Often a long sequence of actions before you discover consequences of the actions
  - E.g., win or lose game only after moves are complete
- Never see the result of actions not taken
- Never told what the best action was
A mouse (or robot) is placed in a maze

- On each trial, starts on a lettered square
- Can move to any adjacent square, except for the maroon one.
- If land in a lettered square, nothing happens.
- If land in **Food** get fruit loops (+1) and leave maze.
- If land in **Shock** get a mild shock (-1) and leave maze.
- Initially no knowledge.
Mouse in Maze Example

**Goal**: learn optimal policy e.g. by learning value of every square
- Initial values all set to 0
- On each trial, move through maze until exit.
- Update values of squares as you leave them.
- Do many trials to learn values of every square.
Mouse in Maze Example

- Update rule for value $V(s)$ of square $s$ you just left, when entering square $s'$
  - $\Delta V = c \left( V(s') - V(s) + R(s) \right)$
  - $V(s)$ and $V(s')$ are the values of the two squares before the update.
  - After update $V(s) = V(s) + \Delta V$.
  - $R(s)$ is the reward you get when leaving square $s$.
  - The constant $c$ is the learning rate.
Mouse in Maze Example

Trial 1: A ➔ B ➔ C ➔ Food ➔ Get reward 1 and exit

- Define $V(\text{exit}) = 0$, always.
- $\Delta V = 0.5 \left( V(s') - V(s) + R(s) \right)$
- New value of A: $V(A) + 0.5 \left( V(B) - V(A) + R(A) \right) = 0$
- New value of B: $V(B) + 0.5 \left( V(C) - V(B) + R(B) \right) = 0$
- New value of C: $V(C) + 0.5 \left( V(\text{Food}) - V(C) + R(C) \right) = 0$
- New value of Food: $V(\text{Food}) + 0.5 \left( V(\text{Exit}) - V(\text{Food}) + R(\text{Food}) \right) = 0.5$
Mouse in Maze Example

Trial 2: A → B → C → Food → Get reward 1 and exit

- New value of A: \( V(A) + 0.5 \left( V(B) - V(A) + R(A) \right) = 0 \)
- New value of B: \( V(B) + 0.5 \left( V(C) - V(B) + R(B) \right) = 0 \)
- New value of C: \( V(C) + 0.5 \left( V(Food) - V(C) + R(C) \right) = 0.25 \)
- New value of Food: \( V(Food) + 0.5 \left( V(Exit)-V(Food) + R(Food) \right) = 0.75 \)
Mouse in Maze Example

- After many trials learn values

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Food</th>
<th>Values after convergence</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.812</td>
<td>0.868</td>
<td>0.918</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>Shock</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>0.762</td>
<td>0.660</td>
<td>-1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>0.705</td>
<td>0.655</td>
<td>0.611</td>
<td>0.388</td>
<td></td>
</tr>
</tbody>
</table>

What is implicit in these values?
What would the value of A be under an optimal policy with no discounting and deterministic motion?
Examples of RL

- Flying Helicopter Stunts
- Playing Video/Board/Strategy games
- Managing Investment Portfolios
- Medicine

Stanford Autonomous Helicopter

https://www.youtube.com/watch?v=VCdxqn0fcnE
MARKOV DECISION PROCESS (MDP)
MDPs generalize HMMs

**HMM**

- Markov transition matrix: $M_k$,
- Emission matrix: $B_v$

**MDP**

- Different transition matrix for each action, $a$
- Emission, $x_t$, includes reward, $R_t$

$M = \text{Markov transition matrix}$
$B = \text{emission matrix}$

$M^{(a)} = \text{Different transition matrix for each action, } a$

Emission, $x_t$, includes reward, $R_t$
**MDP Example**

- **State:** agent position
- **Action:** up, down, left, right
  - excluding actions that cause collisions
- **Transition:** where you actually move (depends on state and action)
- **Reward:**
  - 0 - have not reached exit
  - 1 - reached good exit
  - -1 - reached bad exit
MDP Specification

Joint distribution $p(s', r | s, a) = Pr \{ S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a \}$ can be used to specify MDP

Traditional specification of MDP is 5-tuple $(S, A(\cdot), p(\cdot | \cdot, \cdot), r(\cdot, \cdot, \cdot), \gamma)$ where

- $S$ is a finite set of states
- $A(s)$ is a finite set of actions
- $p(s' | s, a) = Pr(s_{t+1} = s' | S_t = s, A_t = a) = \sum_{r \in \mathcal{R}} p(s', r | s, a)$
- $r(s, a, s') = \mathbb{E} [R_{t+1} | S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in \mathcal{R}} r p(s', r | s, a)}{p(s' | s, a)}$
- $\gamma \in [0, 1]$ is the discount factor

Goal: Find policy $a_t = \pi(s_t)$ that maximizes long term return

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{k+t+1}$$
Notation summary

- $s_t$  state
- $V(s_t)$  value
- $a_t$  action
- $\gamma$  discount factor
- $r(s_t,a_t,s_{t+1})$  reward (usually just $r(s_{t+1})=R_{t+1}$)
- $G_t$  expected discounted reward (‘return’)
- $p(s_{t+1}|s_t,a_t)$  model
MDP generalize with N Nets

- $s_t$  
  state – a vector

- $a_t$  
  action – a vector

- $V(s_t)$  
  value – a nonlinear function of $s_t$

- $p(s_{t+1}|s_t, a_t)$  
  model – a nonlinear function of $s_t$ and $a_t$
  - Often deterministic: $s_{t+1} = f(s_t, a_t)$
Policy, Value, and Q Values

Policy Specific

- **Policy (could be stochastic):** $\pi(a|s)$

- **Value:**

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

$$= \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s \right]$$

- **Q value:**

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

$$= \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s A_t = a \right]$$

Optimal

- **Policy (deterministic):**

$$\pi^*(s) = \arg\max_a q_*(s, a)$$

- **Value:**

$$v_*(s) = \max_{\pi} v_{\pi}(s)$$

$$= \max_a q_*(s, a)$$

- **Q value:**

$$q_*(s, a) = \max_{\pi} q_{\pi}(s, a)$$
Questions

◆ What is $V(A)$?
◆ What is $R(A)$?
◆ What is $q(A, \text{move to D})$?
◆ What is $\pi^*(A)$?
◆ What are possible reasons that $V(A) < 1$?
Bellman’s Equation

\[ v_\pi(s) = \mathbb{E}_\pi [G_t | S_t = s] \]

\[ = \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \]

\[ = \mathbb{E}_\pi \left[ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s \right] \]

\[ = \sum_a \pi(a | s) \sum_{s', r} p(s', r | s, a) \left[ r + \gamma \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_{t+1} = s' \right] \right] \]

\[ = \sum_a \pi(a | s) \sum_{s', r} p(s', r | s, a) \left[ r + \gamma v_\pi(s') \right] , \forall s \in S \]
**Bellman’s Equation**: Holds for all policies $\pi(a|s)$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[ r + \gamma v_\pi(s') \right], \forall s \in S$$

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a) \left[ r + \gamma v_\pi(s') \right], \forall s \in S, \forall a \in A(s)$$

**Bellman’s Optimality Equation**: Holds for optimal policies $\pi^*(s)$

$$v_*(s) = \max_{a \in A(s)} \sum_{s',r} p(s',r|s,a) \left[ r + \gamma v_*(s') \right], \forall s \in S$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a) \left[ r + \gamma \max_{a'} q_*(s') \right], \forall s \in S, \forall a \in A(s)$$
How to solve MDPs?

Solving MDP

Known Simple Model

Unknown or Complicated Model

Dynamic Programming

Generalized Policy Iteration

Reinforcement Learning
Policy Iteration, Value Iteration

DYNAMIC PROGRAMMING
Compute $v_\pi$ for an arbitrary policy $\pi$

Turn *Bellman’s Equation* into an update rule to find a fixed point

Randomly initialize initial approximation $v_0$

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left[ r + \gamma v_k(s') \right]$$

Bellman’s Equation shows that $v_k = v_\pi$ is a fixed point for this update rule

Sequence $\{v_k\} \rightarrow v_\pi$ as $k \rightarrow \infty$. 
Policy Improvement

Greedily update policy $\pi(s) \rightarrow \pi'(s)$

Initialize a random policy $\pi_0$

$$\pi'(s) = \arg\max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')]$$

Policy gives a strictly better policy except when original policy is already optimal
Policy Iteration

Policy Iteration (using iterative policy evaluation)

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

2. Policy Evaluation
   Repeat
   \( \Delta \leftarrow 0 \)
   For each \( s \in \mathcal{S} \):
   \( v \leftarrow V(s) \)
   \( V(s) \leftarrow \sum_{s', r} p(s', r|s, \pi(s))[r + \gamma V(s')] \)
   \( \Delta \leftarrow \max(\Delta, |v - V(s)|) \)
   until \( \Delta < \theta \) (a small positive number)

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

Assuming deterministic policy \( \pi(s) \)
Generalized Policy Iteration

- **Policy iteration** alternates between *Policy Evaluation* and *Policy Improvement*
- **Value iteration** performs a single iteration of *Policy Evaluation* in between each *Policy Improvement*
- **Generalized policy iteration** interleaves *Policy Evaluation* and *Policy Improvement* arbitrarily

From Sutton Reinforcement Learning: An Introduction (2016 draft)
EXPLORATION-EXPLOITATION TRADE-OFF
When should I stop exploring?

State: $|S| = 1$
Action: $a_k :$ pulling kth arm ($k = 1, \cdots N$)
Gambling Machines: Return 1 with unknown probability $p_k$ and 0 otherwise
Reward = 1 or 0
Cost: waste in making a suboptimal pull

Should I select the best arm based on my current knowledge? Or should I explore other arms?
On or off policy control

- **Action (behavior) policy, $\mu$:** policy for choosing an action
- **Target policy, $\pi$:** policy that we want to update

**On-policy Control**

Learn a policy $\pi$ using experience sampled from $\pi$. ($\mu = \pi$)

**Off-policy Control**

Learn a policy $\pi$ using experience sampled from $\mu$. ($\mu \neq \pi$)
- Safe exploration
- Learn by observing others
$\epsilon$-greedy Exploration

- **Continual exploration**
  - With probability $\epsilon$, perform a randomly selected action
  - With probability $1 - \epsilon$, perform a greedy action

- **For any $\epsilon$-greedy policy, the $\epsilon$-greedy policy $\mu$ with respect to $Q^\pi$ is an improvement**

- **Time-varying** $\epsilon = \epsilon_t$

  $$\epsilon_t = \frac{n_0}{n_0 + \text{visits}(s_t)}$$

An annealing schedule

$n_0 = 100$
Learning Methods

\[ \mathcal{M} = \langle S, A, P, R, \gamma \rangle \]
Monte Carlo (MC) Methods in RL

- Estimate expected reward by sampling
  - avoids full search
  - defined for episodic tasks

\[ \mathbb{E}_\pi \left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \]
Monte Carlo (MC) Prediction

- **Return**, $G_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-1} r_{t+T}$
  
  In MC, use empirical mean return starting from $s_t$ or $(s_t, a_t)$ instead of expected return for $V^\pi(s_t)$ or $Q^\pi(s, a)$

- $V^\pi(s) = \text{average of the returns following all the visits to } s \text{ in a set of episodes}$

- $Q^\pi(s, a) = \text{average of the returns following all the visits to } (s, a) \text{ in a set of episodes}$
Monte Carlo Updates

Sample an episode following the current action policy (obtain a return, $G_t$, of the episode)

Update the action value with the average of $[G_1, G_2, \ldots, G_N]$

*Image from Sutton Reinforcement Learning: An Introduction (2016 draft)*
Temporal Difference (TD) Learning

Q-LEARNING
Temporal Difference (TD) Prediction

◆ Combination of DP & MC

◆ Unlike MC, TD learns from current predictions rather than waiting until termination

◆ TD(0): One-step look ahead
  - TD target: $r_t + \gamma V(s_{t+1})$
  - TD error: $\delta_t = r_t + \gamma V(s_{t+1}) - Q(s_t, a_t)$
Q-learning: Off-policy TD(0)

- **On experience** < $s_t, a_t, r_t, s_{t+1}$ > with greedy target policy $\pi$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left( r_t + \gamma \max_{a' \in A} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$$

$\alpha$: Learning rate

- **Convergence** is guaranteed for discrete $S, A$ if:
  - $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$ ($\alpha \in (0,1)$)
  - All $(s,a)$ pairs are visited infinitely often

*Proof in [Watkins & Dayan 1992]*
Q-learning: Off-policy TD(0)

Q-learning

Initialize $Q(s, a), \forall s \in S, a \in A(s)$, arbitrarily, and $Q(terminal-state, \cdot) = 0$

Repeat (for each episode):
  Initialize $S$
  Repeat (for each step of episode):
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Take action $A$, observe $R$, $S'$
    $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma \max_a Q(S', a) - Q(S, A)]$
  $S \leftarrow S'$
until $S$ is terminal

* When your subsequent state $s_{t+1} = S'$ is a terminal state, your "expected future total reward" is just the immediate reward, $r_t + \gamma r_{t+1} + \ldots$
Monte Carlo vs. Q-learning

- **MC: High Variance, Low Bias**
  - Less sensitive to initial Q values

- **Q-learning (TD): Low Variance, High Bias**
  - Online learning is possible. We wait only one time step!
  - For applications with long episodes: delaying all learning until an episode’s end is too slow
  - Needed for non-episodic (continuing) tasks
  - In practice, TD methods converge faster than constant $\alpha$ MC methods on stochastic tasks
Summary

From David Silver UCL Course on RL: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html
Recent Achievements in RL

**AlphaGo**


**Robot Manipulator**

Levine et al., Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. Arxiv 2016

**Drifting Car**

Power Utilization

High PUE
ML Control On
ML Control Off
Low PUE

21/12246258/google-deepmind-ai-data-center-cooling
Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

Starting from random play, and given no domain knowledge except the game rules, AlphaZero achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case.

What you should know

- S,V,A,R,G, $\gamma$
- Exploration/exploitation
- Model based vs. model free RL
  - POMDP
- On policy / off policy
- Q-learning (TD)