Intro to Reinforcement Learning

Paul Liang

pliang@cs.cmu.edu

@pliang279
 Acknowledgement: Much of the material and slides for this lecture were borrowed from the Deep RL Bootcamp at UC Berkeley organized by Pieter Abbeel, Yan Duan, Xi Chen, and Andrej Karpathy, as well as Katerina Fragkiadaki and Ruslan Salakhutdinov’s 10-703 course at CMU, who in turn borrowed much from Rich Sutton’s class and David Silver’s class on Reinforcement Learning, and with inspiration from Chun Kai Ling and Hai Pham’s slides.
Contents

- Introduction to RL
- Markov Decision Processes (MDPs)
- Solving known MDPs using value and policy iteration
- Solving unknown MDPs using function approximation and Q-learning
Reinforcement Learning

ALVINN, 1989
Reinforcement Learning

ALVINN, 1989

AlphaGo, 2016

DQN, 2015
Reinforcement Learning

ALVINN, 1989  
AlphaGo, 2016  
DQN, 2015
Reinforcement Learning

Trajectory

\[ s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots \]
Markov Decision Process (MDPs)

An MDP is defined by:
- Set of states $S$
- Set of actions $A$
- Transition function $P(s' | s, a)$
- Reward function $R(s, a, s')$
- Start state $s_0$
- Discount factor $\gamma$
- Horizon $H$
Markov Decision Process (MDPs)

An MDP is defined by:
- Set of states $S$
- Set of actions $A$
- Transition function $P(s’ | s, a)$
- Reward function $R(s, a, s’)$
- Start state $s_0$
- Discount factor $\gamma$
- Horizon $H$

Trajectory

$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$
Markov assumption + Fully observable

A state should summarize all past information and have the \textbf{Markov property}.

\[
P[R_{t+1} = r, S_{t+1} = s'| S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t] = P[R_{t+1} = r, S_{t+1} = s'| S_t, A_t]
\]

for all \( s' \in S \), \( r \in \mathcal{R} \), and all histories.

- We should be able to throw away the history once state is known.
Markov assumption + Fully observable

A state should summarize all past information and have the **Markov property**.

\[
P[R_{t+1} = r, S_{t+1} = s' | S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t] = P[R_{t+1} = r, S_{t+1} = s' | S_t, A_t]
\]

for all \( s' \in S, r \in R \), and all histories

- We should be able to throw away the history once state is known

If some information is only partially observable: Partially Observable MDP (POMDP)
Return

In continuing tasks, we often use simple total discounted reward:

\[ G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \]

\( \gamma \) close to 0 leads to "myopic" evaluation
\( \gamma \) close to 1 leads to "far-sighted" evaluation
**Policy**

**Definition:** A policy is a distribution over actions given states,

$$\pi(a \mid s) = \Pr(A_t = a \mid S_t = s), \forall t$$

- A policy fully defines the behavior of an agent
- The policy is stationary (time-independent)
- During learning, the agent changes his policy as a result of experience

Special case: deterministic policies

$$\pi(s) = \text{the action taken with prob = 1 when } S_t = s$$
**Policy**

**Definition:** A policy is a distribution over actions given states,

\[ \pi(a \mid s) = \Pr(A_t = a \mid S_t = s), \forall t \]

- A policy fully defines the behavior of an agent
- The policy is stationary (time-independent)
- During learning, the agent changes his policy as a result of experience

Special case: deterministic policies

\[ \pi(s) = \text{the action taken with prob = 1 when } S_t = s \]
Learn the optimal policy to maximize return

An MDP is defined by:
- Set of states $S$
- Set of actions $A$
- Transition function $P(s' \mid s, a)$
- Reward function $R(s, a, s')$
- Start state $s_0$
- Discount factor $\gamma$
- Horizon $H$

**Goal:**

**Return:**

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

**Goal:**

$$\arg \max_\pi \mathbb{E} \left[ \sum_{t=0}^{H} \gamma^t R_t \mid \pi \right]$$
Learn the optimal policy to maximize return

An MDP is defined by:

- Set of states $S$
- Set of actions $A$
- Transition function $P(s' | s, a)$
- Reward function $R(s, a, s')$
- Start state $s_0$
- Discount factor $\gamma$
- Horizon $H$

Goal:

Return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Arg max $\pi$

$$\sum_{t=0}^{H} \gamma^t R_t | \pi$$
Simple Example

M4-1 Quiz 2

Legend:
Grey: walls, Red: cliff (terminal state), Orange: gold (terminal state), Blue: slippery slope

Actions:
• up, down, left, right
• Taking any action on blue tiles causes you to fall down with probability $p$.

Q: Which of the settings for gamma and $p$ result in an optimal agent's first action to be "Left"?

A: $\gamma = 0.1$, $p = 0$
B: $\gamma = 0.99999$, $p = 0$
C: $\gamma = 0.1$, $p = 0.2$
D: $\gamma = 0.99999$, $p = 0.2$
Reinforcement Learning vs Supervised Learning

Reinforcement Learning
- Sequential decision making

Supervised Learning
- One-step decision making
Reinforcement Learning vs Supervised Learning

Reinforcement Learning
- Sequential decision making
- Maximize cumulative reward

Supervised Learning
- One-step decision making
- Maximize immediate reward
Reinforcement Learning vs Supervised Learning

Reinforcement Learning
- Sequential decision making
- Maximize cumulative reward
- Sparse rewards

Supervised Learning
- One-step decision making
- Maximize immediate reward
- Dense supervision
Reinforcement Learning vs Supervised Learning

Reinforcement Learning
● Sequential decision making
● Maximize cumulative reward
● Sparse rewards
● Environment maybe unknown

Supervised Learning
● One-step decision making
● Maximize immediate reward
● Dense supervision
● Environment always known
Intersection between RL and supervised learning

Imitation learning!
Intersection between RL and supervised learning

Imitation learning!

Obtain expert trajectories (e.g. human driver/video demonstrations):

\[ S_0, a_0, r_0, S_1, a_1, r_1, S_2, a_2, r_2, \ldots \]
Intersection between RL and supervised learning

Imitation learning!

Obtain expert trajectories (e.g. human driver/video demonstrations):

\[ S_0, a_0, r_0, S_1, a_1, r_1, S_2, a_2, r_2, \ldots \]

Perform supervised learning by predicting expert action

\[ D = \{(s_0, a^*_0), (s_1, a^*_1), (s_2, a^*_2), \ldots\} \]
Intersection between RL and supervised learning

Imitation learning!

Perform supervised learning by predicting expert action

\[ D = \{(s_0, a^{*0}), (s_1, a^{*1}), (s_2, a^{*2}), \ldots\} \]

Obtain expert trajectories (e.g. human driver/video demonstrations):

\[ S_0, a_0, r_0, S_1, a_1, r_1, S_2, a_2, r_2, \ldots \]

But: distribution mismatch between training and testing

Hard to recover from sub-optimal states

Sometimes not safe/possible to collect expert trajectories
Learn the optimal policy to maximize return

An MDP is defined by:
- Set of states \( S \)
- Set of actions \( A \)
- Transition function \( P(s' | s, a) \)
- Reward function \( R(s, a, s') \)
- Start state \( s_0 \)
- Discount factor \( \gamma \)
- Horizon \( H \)

Goal:

\[
\arg\max_{\pi} \mathbb{E} \left[ \sum_{t=0}^{H} \gamma^t R_t | \pi \right]
\]

Return:

\[
G_t = R_{t+1} + \gamma R_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}
\]
State and action value functions

**Definition:** The state-value function $V^\pi(s)$ of an MDP is the expected return starting from state $s$, and then following policy $\pi$

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

Captures long term reward
State and action value functions

**Definition:** The *state-value function* $V^\pi(s)$ of an MDP is the expected return starting from state $s$, and then following policy $\pi$

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

Captures long term reward

The *action-value function* $Q^\pi(s, a)$ is the expected return starting from state $s$, taking action $a$, and then following policy $\pi$

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

Captures long term reward
Optimal state and action value functions

- **Definition:** The *optimal state-value function* $V^*(s)$ is the maximum value function over all policies

\[ V^*(s) = \max_{\pi} V^\pi(s) \]
Optimal state and action value functions

- **Definition**: The *optimal state-value function* $V^*(s)$ is the maximum value function over all policies

$$V^*(s) = \max_\pi V^\pi(s)$$

- The *optimal action-value function* $Q^*(s, a)$ is the maximum action-value function over all policies

$$Q^*(s, a) = \max_\pi Q^\pi(s, a)$$
Solving MDPs

- **Prediction**: Given an MDP \((S, A, T, r, \gamma)\) and a policy

\[
\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]
\]

find the state and action value functions.

\[
\begin{bmatrix}
V^\pi(s) \\
Q^\pi(s, a)
\end{bmatrix}
\]
Solving MDPs

- **Prediction**: Given an MDP \((S, A, T, r, \gamma)\) and a policy

\[
\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]
\]

\[
V^\pi(s) \quad Q^\pi(s, a)
\]

find the state and action value functions.

- **Optimal control**: given an MDP \((S, A, T, r, \gamma)\), find the optimal policy (aka the planning problem). Compare with the learning problem with missing information about rewards/dynamics.

\[
V^*(s) \quad Q^*(s, a)
\]
Value functions

- **Value functions** measure the goodness of a particular state or state/action pair: how good is for the agent to be in a particular state or execute a particular action at a particular state, for a given policy.
- **Optimal value functions** measure the best possible goodness of states or state/action pairs under all possible policies.

<table>
<thead>
<tr>
<th></th>
<th>state values</th>
<th>action values</th>
</tr>
</thead>
<tbody>
<tr>
<td>prediction</td>
<td>$V_\pi$</td>
<td>$q_\pi$</td>
</tr>
<tr>
<td>control</td>
<td>$V_*$</td>
<td>$q_*$</td>
</tr>
</tbody>
</table>
Relationships between state and action values

State value functions

\[ V^\pi(s) \]

\[ V^*(s) = \max_\pi V^\pi(s) \]

Action value functions

\[ Q^\pi(s, a) \]

\[ Q^*(s, a) \]
Relationships between state and action values

State value functions

\[ V^\pi(s) \]

\[ V^*(s) = \max_\pi V^\pi(s) \]

Action value functions

\[ Q^\pi(s, a) \]

\[ Q^*(s, a) = \max_\pi Q^\pi(s, a) \]
Relationships between state and action values

State value functions

\[ V^\pi(s) \]

\[ V^*(s) = \max_\pi V^\pi(s) \]

Action value functions

\[ Q^\pi(s, a) \]

\[ Q^*(s, a) = \max_\pi Q^\pi(s, a) \]

\[ V^*(s) = \max_a Q^*(s, a) \]
Relationships between state and action values

State value functions

\[ V^\pi(s) = \sum_a \pi(a|s)Q^\pi(s, a) \]

\[ V^*(s) = \max_\pi V^\pi(s) \]

Action value functions

\[ Q^\pi(s, a) \]

\[ Q^*(s, a) = \max_\pi Q^\pi(s, a) \]

\[ V^*(s) = \max_a Q^*(s, a) \]
Obtaining the optimal policy

Optimal policy can be found by maximizing over $Q^*(s,a)$

$$\pi^*(a|s) = \begin{cases} 
1, & \text{if } a = \arg \max_a Q^*(s,a) \\
0, & \text{else}
\end{cases}$$
Obtaining the optimal policy

Optimal policy can be found by maximizing over $Q^*(s,a)$

$$
\pi^*(a|s) = \begin{cases} 
1, & \text{if } a = \arg \max_a \ Q^*(s,a) \\
0, & \text{else}
\end{cases}
$$

Optimal policy can also be found by maximizing over $V^*(s')$ with one-step look ahead

$$
\pi^*(a|s) = \begin{cases} 
1, & \text{if } a = \arg \max_a \ E_{s'} [r(s, a, s') + \gamma V^*(s')] \\
0, & \text{else}
\end{cases}
$$
Obtaining the optimal policy

Optimal policy can be found by maximizing over $Q^*(s,a)$

$$
\pi^*(a|s) = \begin{cases} 
1, & \text{if } a = \arg\max_a Q^*(s, a) \\
0, & \text{else}
\end{cases}
$$

Optimal policy can also be found by maximizing over $V^*(s')$ with one-step look ahead

$$
\pi^*(a|s) = \begin{cases} 
1, & \text{if } a = \arg\max_a \mathbb{E}_{s'} \left[ r(s, a, s') + \gamma V^*(s') \right] \\
0, & \text{else}
\end{cases}
$$

$$
\pi^*(a|s) = \begin{cases} 
1, & \text{if } a = \arg\max_a \left[ \sum_{s'} p(s'|s, a)(r(s, a, s') + \gamma V^*(s')) \right] \\
0, & \text{else}
\end{cases}
$$
So, how do we find $Q^*(s,a)$ and $V^*(s)$?

Recursively:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \ldots$$

$$= r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \ldots)$$

$$= r_{t+1} + \gamma G_{t+1}$$
Bellman expectation

Recursively:

\[ G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \ldots \]

\[ = r_{t+1} + \gamma \left( r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \ldots \right) \]

\[ = r_{t+1} + \gamma G_{t+1} \]
Bellman expectation

Recursively:

\[ G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \ldots \]
\[ = r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \ldots) \]
\[ = r_{t+1} + \gamma G_{t+1} \]

By taking expectations:

\[ V^\pi (s) = \mathbb{E}_\pi [G_t | S_t = s] \]
\[ = \mathbb{E}_\pi [r_{t+1} + \gamma G_{t+1} | S_t = s] \]
\[ = \mathbb{E}_\pi [r_{t+1} + \gamma V^\pi (S_{t+1}) | S_t = s] \]
Bellman expectation

Recursively:

\[ G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \ldots \]
\[ = r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \ldots) \]
\[ = r_{t+1} + \gamma G_{t+1} \]

By taking expectations:

\[ V^\pi (s) = \mathbb{E}_\pi [G_t | S_t = s] \]
\[ = \mathbb{E}_\pi [r_{t+1} + \gamma G_{t+1} | S_t = s] \]
\[ = \mathbb{E}_\pi [r_{t+1} + \gamma V^\pi (S_{t+1}) | S_t = s] \]
\[ = \sum_a \pi (a | s) \]
Bellman expectation

Recursively:

\[ G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \ldots \]
\[ = r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \ldots) \]
\[ = r_{t+1} + \gamma G_{t+1} \]

By taking expectations:

\[ V^\pi (s) = \mathbb{E}_\pi [G_t | S_t = s] \]
\[ = \mathbb{E}_\pi [r_{t+1} + \gamma G_{t+1} | S_t = s] \]
\[ = \mathbb{E}_\pi [r_{t+1} + \gamma V^\pi (S_{t+1}) | S_t = s] \]
\[ = \sum_a \pi(a | s) \mathbb{E}_{s'} [r(s, a, s') + \gamma V^\pi (s')] \]
Bellman expectation

Recursively:

\[ G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \ldots \]
\[ = r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \ldots) \]
\[ = r_{t+1} + \gamma G_{t+1} \]

By taking expectations:

\[ V^\pi(s) = \mathbb{E}_\pi [G_t | S_t = s] \]
\[ = \mathbb{E}_\pi [r_{t+1} + \gamma G_{t+1} | S_t = s] \]
\[ = \mathbb{E}_\pi [r_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s] \]
\[ = \sum_a \pi(a | s) \mathbb{E}_{s'} [r(s, a, s') + \gamma V^\pi(s')] \]
\[ = \sum_a \pi(a | s) \sum_{s'} p(s' | s, a) [r(s, a, s') + \gamma V^\pi(s')] \]
Bellman expectation for state value functions

$$V^\pi(s) = \sum_a \pi(a|s)$$
Bellman expectation for state value functions

\[
V^\pi (s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a)
\]
Bellman expectation for state value functions

\[ V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) \left[ r(s, a, s') + \gamma V^\pi(s') \right] \]
Bellman expectation for action value functions

\[ Q^\pi(s, a) = \sum_{s'} p(s' | s, a) \]
Bellman expectation for action value functions

\[
Q^{\pi}(s, a) = \sum_{s'} p(s'|s, a) \left( r(s, a, s') \right)
\]
Bellman expectation for action value functions

\[ Q^\pi(s, a) = \sum_{s'} p(s'|s, a) \left( r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') \right) \]
Bellman expectation for action value functions

\[
Q^\pi(s, a) = \sum_{s'} p(s'|s, a) \left( r(s, a, s') + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s', a') \right)
\]
Solving the Bellman expectation equations

\[ V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) [r(s,a,s') + \gamma V^\pi(s')] \]
Solving the Bellman expectation equations

\[ V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) [r(s,a,s') + \gamma V^\pi(s')] \]

Solve the linear system

variables: \( V^\pi(s) \) for all s

constants: \( p(s'|s,a), r(s,a,s') \)
Solving the Bellman expectation equations

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

Solve the linear system

variables: \( V^\pi(s) \) for all \( s \)
constants: \( p(s'|s,a), r(s,a,s') \)

Solve by iterative methods

\[ V_{[k+1]}^\pi(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s,a) \left[ r(s,a,s') + \gamma V_{[k]}^\pi(s') \right] \]
Policy Evaluation

Policy evaluation
Iterate until convergence:

\[
V_{[k+1]}^{\pi}(s) = \sum_a \pi(a|s) \sum_{s'} p(s'|s, a) \left[ r(s, a, s') + \gamma V_{[k]}^{\pi}(s') \right]
\]
Policy Iteration

1. Policy evaluation
Iterate until convergence:

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

2. Policy Improvement
Find the best action according to one-step look ahead

\[ \pi_{[k+1]}(a|s) = \arg \max_a \sum_{s'} p(s'|s, a) \left[ r(s, a, s') + \gamma V_{[k]}^{\pi}(s') \right] \]
Policy Iteration

1. Policy evaluation
   Iterate until convergence:
   \[
   V_{[k+1]}^\pi(s) = \sum_a \pi_{[k]}(a|s) \sum_{s'} p(s'|s,a) \left[ r(s,a,s') + \gamma V_{[k]}^\pi(s') \right]
   \]

2. Policy Improvement
   Find the best action according to one-step look ahead
   \[
   \pi_{[k+1]}(a|s) = \arg \max_a \sum_{s'} p(s'|s,a) \left[ r(s,a,s') + \gamma V_{[k]}^\pi(s') \right]
   \]
Policy Iteration

1. **Policy evaluation**
   Iterate until convergence:
   
   \[
   V_{[k+1]}^\pi(s) = \sum_a \pi_{[k]}(a|s) \sum_{s'} p(s'|s,a) \left[ r(s,a,s') + \gamma V_{[k]}^\pi(s') \right]
   \]

2. **Policy Improvement**
   Find the best action according to one-step look ahead
   
   \[
   \pi_{[k+1]}(a|s) = \arg \max_a \sum_{s'} p(s'|s,a) \left[ r(s,a,s') + \gamma V_{[k]}^\pi(s') \right]
   \]

Repeat until policy converges. Guaranteed to converge to optimal policy.
Bellman optimality for state value functions

The value of a state under an optimal policy must equal the expected return for the best action from that state:

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

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

For the Bellman expectation equations we summed over all leaves, here we choose the best branch.
Bellman optimality for state value functions

The value of a state under an optimal policy must equal the expected return for the best action from that state.

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

For the Bellman expectation equations we summed over all leaves, here we choose the best branch.
Bellman optimality for action value functions

For the Bellman expectation equations we summed over all leaves, here we choose the best branch.

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

\[ = \mathbb{E}_{s'} \left[ r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right] \]
Bellman optimality for action value functions

For the Bellman expectation equations we summed over all leaves, here we choose the best branch.

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

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

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

Solve by iterative methods

\[ V_{[k+1]}^*(s) = \max_a \left[ \sum_{s'} p(s' | s, a) (r(s, a, s') + \gamma V_{[k]}^*(s')) \right] \]
Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all $s$.

For $k = 1, \ldots, H$:

For all states $s$ in $S$:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left( R(s,a,s') + \gamma V_{k-1}^*(s') \right)$$
Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all $s$.

For $k = 1, \ldots, H$:

For all states $s$ in $S$:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left( R(s,a,s') + \gamma V_{k-1}^*(s') \right)$$

$$\pi_k^*(s) \leftarrow \arg \max_a \sum_{s'} P(s'|s,a) \left( R(s,a,s') + \gamma V_{k-1}^*(s') \right)$$

Find the best action according to one-step look ahead

This is called a value update or Bellman update/back-up
**Value Iteration** Repeat until policy converges. Guaranteed to converge to optimal policy.

**Algorithm:**

Start with $V_0^*(s) = 0$ for all $s$.

For $k = 1, \ldots, H$:

For all states $s$ in $S$:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left( R(s,a,s') + \gamma V_{k-1}^*(s') \right)$$

$$\pi_k^*(s) \leftarrow \arg \max_a \sum_{s'} P(s'|s,a) \left( R(s,a,s') + \gamma V_{k-1}^*(s') \right)$$

Find the best action according to one-step look ahead

This is called a **value update** or **Bellman update/back-up**
Q-Value Iteration

\[ Q^*(s, a) = \text{expected utility starting in } s, \text{ taking action } a, \text{ and (thereafter) acting optimally} \]

**Bellman Equation:**

\[ Q^*(s, a) = \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q^*(s', a')) \]

**Q-Value Iteration:**

\[ Q_{k+1}^*(s, a) \leftarrow \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q_k^*(s', a')) \]
Summary: Exact methods

Fully known
MDP
states
transitions
rewards
Summary: Exact methods

- Fully known MDP
- states
- transitions
- rewards

Bellman optimality equations

\[ Q^*(s, a) \]
\[ V^*(s) \]

Q-value iteration

Value iteration

Repeat until policy converges. Guaranteed to converge to optimal policy.
Summary: Exact methods

Fully known MDP states transitions rewards

- Bellman optimality equations
  - $Q^*(s, a)$: Q-value iteration
  - $V^*(s)$: Value iteration

- Bellman expectation equations
  - $Q^\pi(s, a)$: Q-policy iteration
  - $V^\pi(s)$: Policy iteration

Repeat until policy converges. Guaranteed to converge to optimal policy.
Summary: Exact methods

Fully known MDP states transitions rewards

Bellman optimality equations

Bellman expectation equations

$Q^*(s,a)$

$V^*(s)$

$Q^\pi(s,a)$

$V^\pi(s)$

Q-value iteration

Value iteration

Q-policy iteration

Policy iteration

Repeat until policy converges. Guaranteed to converge to optimal policy.

Limitations:
Iterate over and storage for all states and actions: requires small, discrete state and action space
Update equations require fully observable MDP and known transitions
Solving unknown MDPs using function approximation
Recap: Q-value iteration

\[ Q^*(s, a) = \text{expected utility starting in } s, \text{ taking action } a, \text{ and (thereafter) acting optimally} \]

Bellman Equation:

\[ Q^*(s, a) = \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q^*(s', a')) \]
Recap: Q-value iteration

$Q^*(s, a) =$ expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

$$Q^*(s, a) = \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q^*(s', a'))$$

Q-Value Iteration:

$$Q_{k+1}^*(s, a) \leftarrow \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q_k^*(s', a'))$$
Recap: Q-value iteration

Q*(s, a) = expected utility starting in s, taking action a, and (thereafter) acting optimally

Bellman Equation:

\[ Q^*(s, a) = \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q^*(s', a')) \]

Q-Value Iteration:

\[ Q_{k+1}^*(s, a) \leftarrow \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q_k^*(s', a')) \]

This is problematic when do not know the transitions
Tabular Q-learning

- **Q-value iteration:** \( Q_{k+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q_k(s', a')) \)

- **Rewrite as expectation:** \( Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s, a)} \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right] \)
Tabular Q-learning

- Q-value iteration: \( Q_{k+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a)(R(s, a, s') + \gamma \max_{a'} Q_k(s', a')) \)
- Rewrite as expectation: \( Q_{k+1} \leftarrow \mathbb{E}_{s'|s, a} \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right] \)
- (Tabular) Q-Learning: replace expectation by samples
Tabular Q-learning

- **Q-value iteration:** \( Q_{k+1}(s, a) \leftarrow \sum_{s'} P(s'|s, a) (R(s, a, s') + \gamma \max_{a'} Q_k(s', a')) \)

- **Rewrite as expectation:** \( Q_{k+1} \leftarrow \mathbb{E}_{s' \sim P(s'|s, a)} \left[ R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right] \)

- **(Tabular) Q-Learning:** replace expectation by samples
  - For an state-action pair \((s, a)\), receive: \( s' \sim P(s'|s, a) \) **simulation and exploration**
  - Consider your old estimate: \( Q_k(s, a) \)
  - Consider your new sample estimate:
    \[
    \text{target}(s') = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')
    \]
    \[
    \text{error}(s') = \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)
    \]
Tabular Q-learning update

Learning rate

\[ Q_{k+1}(s, a) = Q_k(s, a) + \alpha \text{ error}(s') \]

\[ = Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right) \]

Key idea: implicitly estimate the transitions via simulation
Tabular Q-learning

Algorithm:

- Start with $Q_0(s, a)$ for all $s, a$.
- Get initial state $s$.
- For $k = 1, 2, \ldots$ till convergence:
  - Sample action $a$, get next state $s'$.
  - If $s'$ is terminal:
    - target $= r(s, a, s')$
  - Sample new initial state $s'$.
  - else:
    - target $= r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
- $Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)$
- $s \leftarrow s'$

Bellman optimality

$$Q^*(s, a) = \mathbb{E}_{s'} \left[ r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right]$$
Tabular Q-learning

Algorithm:
Start with $Q_0(s, a)$ for all $s, a$.
Get initial state $s$
For $k = 1, 2, \ldots$ till convergence
  
  **Sample action $a$, get next state $s'$**
  
  If $s'$ is terminal:
  
  $\text{target} = r(s, a, s')$
  
  Sample new initial state $s'$
  
  else:
  
  $\text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$

$$Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)$$

$s \leftarrow s'$
Tabular Q-learning

Algorithm:
Start with $Q_0(s, a)$ for all $s$, $a$.
Get initial state $s$
For $k = 1, 2, \ldots$ till convergence
  Sample action $a$, get next state $s'$
  If $s'$ is terminal:
    \[ \text{target} = r(s, a, s') \]
    Sample new initial state $s'$
  else:
    \[ \text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a') \]
\[ Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right) \]
$s \leftarrow s'$

- Choose random actions?
- Choose action that maximizes $Q_k(s, a)$ (i.e. greedily)?
- $\varepsilon$-Greedy: choose random action with prob. $\varepsilon$, otherwise choose action greedily
Epsilon-greedy

Poor estimates of $Q(s,a)$ at the start:

Bad initial estimates in the first few cases can drive policy into sub-optimal region, and never explore further.

$$\pi(s) = \begin{cases} 
\max_a \hat{Q}(s, a) & \text{with probability } 1 - \epsilon \\
\text{random action} & \text{otherwise}
\end{cases}$$

Gradually decrease epsilon as policy is learned.
Tabular Q-learning

Algorithm:

Start with $Q_0(s, a)$ for all $s, a$.
Get initial state $s$
For $k = 1, 2, \ldots$ till convergence

- **Sample action $a$, get next state $s'$**
  
  If $s'$ is terminal:
  
  $\text{target} = r(s, a, s')$
  
  Sample new initial state $s'$
  
  else:
  
  $\text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$
  
  $Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)$

$s \leftarrow s'$

- $\epsilon$-Greedy: choose random action with prob. $\epsilon$, otherwise choose action greedily
Convergence

- 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
Tabular Q-learning

Algorithm:
Start with $Q_0(s, a)$ for all $s, a$.
Get initial state $s$
For $k = 1, 2, \ldots$ till convergence

\begin{itemize}
  \item \textbf{Sample action $a$, get next state $s'$}
  \item If $s'$ is terminal:
    \begin{align*}
      \text{target} &= r(s, a, s') \\
      \text{Sample new initial state $s'$}
    \end{align*}
  \item else:
    \begin{align*}
      \text{target} &= r(s, a, s') + \gamma \max_{a'} Q_k(s', a') \\
      Q_{k+1}(s, a) &= Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)
    \end{align*}
\end{itemize}

$s \leftarrow s'$
Summary: Exact methods

Bellman optimality equations

\[ Q^*(s, a) \]  Q-value iteration

\[ V^*(s) \]  Value iteration

Bellman expectation equations

\[ Q^\pi(s, a) \]  Q-policy iteration

\[ V^\pi(s) \]  Policy iteration

Repeat until policy converges. Guaranteed to converge to optimal policy.

Limitations:
Iterate over and storage for all states and actions: requires small, discrete state and action space
Update equations require fully observable MDP and known transitions
Summary: Tabular Q-learning

MDP with unknown transitions → Bellman optimality equations → Replace true expectation over transitions with estimates → Tabular Q-learning

\[ s' \sim P(s'|s, a) \]

old estimate \[ Q^*(s, a) = \mathbb{E}_{s'} \left[ r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right] \]

target

simulation and exploration, epsilon greedy is important!
**Summary: Tabular Q-learning**

MDP with unknown transitions $\xrightarrow{\text{Bellman optimality equations}}$ Replace true expectation over transitions with estimates $\xrightarrow{\text{Tabular Q-learning}}$

\[ s' \sim P(s'|s,a) \quad \text{simulation and exploration, epsilon greedy is important!} \]

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

old estimate \quad target

\[ Q_{k+1}(s,a) \leftarrow Q_k(s,a) + \alpha \left( r(s,a,s') + \gamma \max_{a'} Q_k(s',a') - Q_k(s,a) \right) \]
Summary: Tabular Q-learning

MDP with unknown transitions → Bellman optimality equations → Replace true expectation over transitions with estimates

\[ s' \sim P(s' | s, a) \]

simulation and exploration, epsilon greedy is important!

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

old estimate \hspace{5cm} target

\[ Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha \left[ r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right] \]

Tabular: keep a $|S| \times |A|$ table of $Q(s,a)$
Still requires small and discrete state and action space
How can we generalize to unseen states?
Deep Q-learning

Q-learning with function approximation to extract informative features from high-dimensional input states.
Deep Q-learning

Represent value function by Q-network with weights $w$

\[ Q(s, a, w) \approx Q^*(s, a) \]
Deep Q-learning

- Optimal Q-values should obey Bellman equation

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

- Treat right-hand \( r + \gamma \max_{a'} Q(s', a', \mathbf{w}) \) as a target

- Minimize MSE loss by stochastic gradient descent

\[ l = \left( r + \gamma \max_{a} Q(s', a', \mathbf{w}) - Q(s, a, \mathbf{w}) \right)^2 \]

- Remember VFA lecture: Minimize mean-squared error between the true action-value function \( q_\pi(S, A) \) and the approximate Q function:

\[ J(\mathbf{w}) = \mathbb{E}_\pi \left[ (q_\pi(S, A) - \hat{q}(S, A, \mathbf{w}))^2 \right] \]
Deep Q-learning

- Minimize MSE loss by stochastic gradient descent

\[ I = \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \]

- Converges to \( Q^* \) using **table lookup representation**

- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets
Experience replay

- To remove correlations, build data-set from agent’s own experience:

<table>
<thead>
<tr>
<th>$s_1, a_1, r_2, s_2$</th>
<th>$s, a, r, s'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_2, a_2, r_3, s_3$</td>
<td>exploration, epsilon greedy is important!</td>
</tr>
<tr>
<td>$s_3, a_3, r_4, s_4$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$s_t, a_t, r_{t+1}, s_{t+1}$</td>
<td></td>
</tr>
</tbody>
</table>

- Sample random mini-batch of transitions $(s,a,r,s')$ from $D$
Fixed Q-targets

- Sample random mini-batch of transitions \((s,a,r,s')\) from \(D\)
- Compute Q-learning targets w.r.t. old, fixed parameters w-
Fixed Q-targets

- Sample **random mini-batch** of transitions \((s, a, r, s')\) from \(D\)
- Compute Q-learning targets w.r.t. old, fixed parameters w–
- Optimize MSE between Q-network and Q-learning targets

\[
\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i} \left[ \left( r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i) \right)^2 \right]
\]

Q-learning target

Q-network
Fixed Q-targets

- Sample random mini-batch of transitions \((s,a,r,s')\) from \(D\)
- Compute Q-learning targets w.r.t. old, fixed parameters \(w\)-
- Optimize MSE between Q-network and Q-learning targets

\[
\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i} \left[ \left( r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i) \right)^2 \right]
\]

- Use stochastic gradient descent

Update \(w\)- with updated \(w\) every \(~1000\) iterations
Deep Q-learning for Atari
Deep Q-learning for Atari

- End-to-end learning of values $Q(s, a)$ from pixels $s$
- Input state $s$ is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

- Network architecture and hyperparameters fixed across all games

Mnih et.al., Nature, 2014
Deep Q-learning for Atari

- End-to-end learning of values $Q(s,a)$ from pixels $s$
- Input state $s$ is stack of raw pixels from last 4 frames
- Output is $Q(s,a)$ for 18 joystick/button positions
- Reward is change in score for that step

- Network architecture and hyperparameters fixed across all games

Mnih et.al., Nature, 2014
Deep Q-learning for Atari

- End-to-end learning of values $Q(s,a)$ from pixels $s$
- Input state $s$ is stack of raw pixels from last 4 frames
  
  Encourage Markov property
- Output is $Q(s,a)$ for 18 joystick/button positions
- Reward is change in score for that step

- Network architecture and hyperparameters fixed across all games

Mnih et.al., Nature, 2014
Superhuman results
Superhuman results

- Needs reaction speed
- Short term reward
- Less exploration required
- Deep RL >>> humans
Superhuman results

Needs reaction speed
Short term reward
Less exploration required
Deep RL >>> humans

Super long term reward
More exploration required
Requires knowledge of complex dynamics e.g. key, ladder.
Challenge for deep RL
Superhuman results on Montezuma’s Revenge

Encourages agent to explore its environment by maximizing **curiosity**. I.e. how well can I predict my environment?

1. Less training data
2. Stochastic
3. Unknown dynamics

So I should explore more.

Burda et. al., ICLR 2019
Summary: Exact methods

- **Bellman optimality equations**
  - $Q^*(s, a)$
  - $V^*(s)$

- **Bellman expectation equations**
  - $Q^\pi(s, a)$
  - $V^\pi(s)$

Repeat until policy converges. Guaranteed to converge to optimal policy.

**Limitations:**
- Iterate over and storage for all states and actions: requires small, discrete state and action space
- Update equations require fully observable MDP and known transitions
Summary: Tabular Q-learning

MDP with unknown transitions → Bellman optimality equations → Replace true expectation over transitions with estimates

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

old estimate \hspace{2cm} target

$Q_{k+1}(s, a) \leftarrow Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)$

simulation and exploration, epsilon greedy is important!

Tabular: keep a |S| x |A| table of Q(s,a)
Still requires small and discrete state and action space
How can we generalize to unseen states?
Summary: Deep Q-learning

\[ Q^*(s, a) = \mathbb{E}_{s'} \left[ r(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right] \]
Summary: Deep Q-learning

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

old estimate

\[ \mathcal{L}_i(w_i) = \mathbb{E}_{s, a, r, s' \sim D_i} \left[ \left( r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i) \right)^2 \right] \]

Q-learning target

Q-network
Summary: Deep Q-learning

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

old estimate

\[ \mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s' \sim D_i} \left[ \left( r + \gamma \max_{a'} Q(s', a'; w_i^-) - Q(s, a; w_i) \right)^2 \right] \]

Q-learning target

Q-network

Stochastic gradient descent + Experience replay + Fixed Q-targets

Works for high-dimensional state and action spaces

Generalizes to unseen states