Admin

Matching for midterm presentation

→ Give feedback to relevant teams

Due by Wednesday 10/28 8pm ET

https://forms.gle/fEQgmk4g6Yfop6Ct5
Instructions for the **midterm presentations** are on piazza resources:

- [https://piazza.com/class_profile/get_resource/kcnr11wq24q6z7/kg742kik5kv6tg](https://piazza.com/class_profile/get_resource/kcnr11wq24q6z7/kg742kik5kv6tg)
- Deadline for pre-recorded presentation: Friday, November 13th, 2020 at 8pm ET
- 7 minutes, mostly about error analysis and updated ideas, don't try to present everything...

Instructions for **midterm report** are also online:

- [https://piazza.com/class_profile/get_resource/kcnr11wq24q6z7/kgraer741fw3n4](https://piazza.com/class_profile/get_resource/kcnr11wq24q6z7/kgraer741fw3n4)
- Deadline: Sunday, November 15th, 2020
- 8 pages for teams of 3 and 9 pages for the other teams
- Multimodal baselines, error analysis, proposed ideas
Used Materials

Acknowledgement: Some 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.
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

AlphaGo, 2016

DQN, 2015
Reinforcement Learning

The diagram illustrates a basic reinforcement learning setup. The agent interacts with the environment through actions $A_t$ and receives rewards $R_t$. The states of the environment are denoted as $S_t$. A trajectory is formed by the sequence of states, actions, and rewards:

$$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$$
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 $H$

Trajectory

$s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \ldots$
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

We aim to maximize 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

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

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

Special case: deterministic policies
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:**

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

**Goal:**

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

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

\[ D = \{ (s0, a^0), (s1, a^1), (s2, a^2), \ldots \} \]

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:

$\pi$:

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

Return:

Goal:

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

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

Captures long term reward

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

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

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

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

- Definition: 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.

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

- **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.
Value functions

- **Value functions** measure the goodness of a particular state or state/action pair: how good is it for an 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 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

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

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

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**

So, how do we find $Q^*(s,a)$ and $V^*(s)$?

**Recursively:**

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

**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

So, how do we find $Q^*(s,a)$ and $V^*(s)$?

Recursively:

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

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

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

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

So, how do we find $Q^*(s,a)$ and $V^*(s)$?

Recurrsively:

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

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

So, how do we find $Q^*(s,a)$ and $V^*(s)$?

Recursively:

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

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

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

By taking expectations:

$$V^\pi(s) = E_\pi [G_t | S_t = s]$$

$$= E_\pi [r_{t+1} + \gamma G_{t+1} | S_t = s]$$

$$= E_\pi [r_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s]$$

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

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

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

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

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

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

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

1. Policy evaluation
   Iterate until convergence:

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

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

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

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.

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

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

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

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

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

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

Solve by iterative methods

\[ V^*_{[k+1]}(s) = \max_a \left[ \sum_{s'} p(s'|s,a) \left( r(s,a,s') + \gamma V^*_k(s') \right) \right] \]
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 \textit{Repeat until policy converges. Guaranteed to converge to optimal policy.}

**Algorithm:**

Start with $V^*_0(s) = 0$ \textit{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 \textit{value update} or \textit{Bellman update/back-up}
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:

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

Bellman optimality equations

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

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

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

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:

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

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

- **(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 $k = 1, 2, ...$ 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:

$\text{target} = r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$

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

$s \leftarrow s'$

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

- 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:
    - Target $= r(s, a, s')$.
    - Sample new initial state $s'$.
  - Else:
    - Target $= r(s, a, s') + \gamma \max_{a'} Q_k(s', a')$.
    - $Q_{k+1}(s, a) = Q_k(s, a) + \alpha \left( r(s, a, s') + \gamma \max_{a'} Q_k(s', a') - Q_k(s, a) \right)$.
    - $s \leftarrow s'$.

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

Still requires small and discrete state and action space
How can we generalize to unseen states?

- $\epsilon$-Greedy: choose random action with prob. $\epsilon$, otherwise choose action greedily
Summary: Tabular Q-learning

MDP with unknown transitions \(\rightarrow\) Bellman optimality equations \(\rightarrow\) Replace true expectation over transitions with estimates

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

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

old estimate \(\rightarrow\) target
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{1cm} 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!
Summary: Tabular Q-learning

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

Tabular Q-learning

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

simulation and exploration, epsilon greedy is important!

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

old estimate

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

Tabular: keep a |S| x |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.

DQN, 2015
Deep Q-learning

Represent value function by Q network with weights $w$

$$Q(s, a, w) \approx Q^*(s, a)$$

+ high-dimensional, continuous states
+ generalization to new states
Deep Q-learning

- Optimal Q-values should obey Bellman equation

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

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

- Minimize MSE loss by stochastic gradient descent

\[
l = \left( r + \gamma \max_a Q(s', a', w) - Q(s, a, w) \right)^2
\]
Deep Q-learning

- Minimize MSE loss by stochastic gradient descent
  \[ l = \left( r + \gamma \max_a Q(s', a', w) - Q(s, a, w) \right)^2 \]

- 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>$s_2, a_2, r_3, s_3$</th>
<th>$s_3, a_3, r_4, s_4$</th>
<th>$s_t, a_t, r_{t+1}, s_{t+1}$</th>
</tr>
</thead>
</table>

$\rightarrow$ $s, a, r, s'$

exploration, epsilon greedy is important!

- 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 \(w\)
- Optimize MSE between Q-network and Q-learning targets

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

| \(s_1, a_1, r_2, s_2\) |
| \(s_2, a_2, r_3, s_3\) |
| \(s_3, a_3, r_4, s_4\) |
| \(\ldots\) |
| \(s_t, a_t, r_{t+1}, s_{t+1}\) |
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 \(\sim 1000\) 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
Superhuman results

Slides from Fragkiadaki
Superhuman results

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

Slides from Fragkiadaki
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 → Tabular Q-learning

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

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

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?
Summary: Deep Q-learning

\[ Q^*(s, a) = \frac{Q(s, a) \cdot r(s, a, s') + \gamma \max_{a'} Q^*(s', a')}{\text{old estimate}} \]

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

Stochastic gradient descent + Experience replay + Fixed Q-targets

Works for high-dimensional state and action spaces
Generalizes to unseen states
Applications: RL and Language
RL and Language

Task-independent

[...] having the correct
[...] known lock and
[...] unless the correct

*key* can open the lock [...]  
*key* device was discovered [...]  
*key* is inserted [...]  

Pre-training

\[ V_{key} \, V_{skull} \, V_{ladder} \, V_{rope} \]

Pre-trained

Task-dependent

Language-assisted

**Key** Opens a door of the same color as the key.

**Skull** They come in two varieties, rolling skulls and bouncing skulls ... you must jump over rolling skulls and walk under bouncing skulls.

Language-conditional

Go down the ladder and walk right immediately to avoid falling off the conveyor belt, jump to the yellow rope and again to the platform on the right.
Language-conditional RL

- Instruction following
- Rewards from instructions
- Language in the observation and action space
Language-conditional RL: Instruction following

- Navigation via instruction following

Train
- Go to the short red torch
- Go to the blue keycard
- Go to the largest yellow object
- Go to the green object

Test
- Go to the tall green torch
- Go to the red keycard
- Go to the smallest blue object

Chaplot et. al., AAAI 2018
Misra et. al., EMNLP 2017
Language-conditional RL: Instruction following

- Navigation via instruction following

![Diagram of a video game environment with objects and instructions]

**Train**
- Go to the short red torch
- Go to the blue keycard
- Go to the largest yellow object
- Go to the green object

**Test**
- Go to the tall green torch
- Go to the red keycard
- Go to the smallest blue object

**Alignment**
- Ground language
- Recognize objects
- Navigate to objects
- Generalize to unseen objects

Chaplot et. al., AAAI 2018
Misra et. al., EMNLP 2017
Language-conditional RL: Instruction following

- Interaction with the environment

Chaplot et al., AAAI 2018
Language-conditional RL: Instruction following

- Gated attention via element-wise product
Language-conditional RL: Instruction following

- **Policy learning**
  - **Asynchronous Advantage Actor-Critic (A3C) (Mnih et al.)**
    - uses a deep neural network to parametrize the policy and value functions and runs multiple parallel threads to update the network parameters.
    - use **entropy regularization** for improved exploration
    - use **Generalized Advantage Estimator** to reduce the variance of the policy gradient updates (Schulman et al.)
Language-conditional RL: Instruction following

Chaplot et. al., AAAI 2018
Language-conditional RL: Instruction following

Grounding is important for generalization:
blue armor, red pillar -> blue pillar

Chaplot et al., AAAI 2018
Language-conditional RL: Rewards from instructions

Montezuma’s revenge

Sparse, long-term reward problem
General solution: reward shaping via auxiliary rewards
Language-conditional RL: Rewards from instructions

Montezuma’s revenge

Sparse, long-term reward problem
General solution: reward shaping via auxiliary rewards

Encourages agent to explore its environment by maximizing **curiosity**.
How well can I **predict** my environment?
1. Less training data
2. Stochastic
3. Unknown dynamics
So I should **explore more**.

Pathak et. al., ICML 2017
Burda et. al., ICLR 2019
Language-conditional RL: Rewards from instructions

Sparse, long-term reward problem
General solution: reward shaping via auxiliary rewards

Natural language for reward shaping

“Jump over the skull while going to the left”
from Amazon Mturk :-( asked annotators to play the game and describe entities

Intermediate rewards to speed up learning

Montezuma’s revenge

Goyal et. al., IJCAI 2019
Language-conditional RL: Rewards from instructions

Montezuma’s revenge

Natural language for reward shaping

Encourages agent to take actions related to the instructions

Goyal et. al., IJCAI 2019
Language-conditional RL: Rewards from instructions

Natural language for reward shaping

Encourages agent to take actions related to the instructions

Montezuma’s revenge

Goyal et al., IJCAI 2019
Language-conditional RL: Language in S and A

- Embodied QA: Navigation + QA

Most methods similar to instruction following

Das et. al., CVPR 2018
Language-assisted RL

- Language for communicating domain knowledge
- Language for structuring policies
Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language

from Amazon Mturk:-( asked annotators to play the game and describe entities

Narasimhan et. al., JAIR 2018
Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language.
Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language
Language-assisted RL: Domain knowledge

- Properties of entities in the environment are annotated by language

Grounded language learning
Helps to ground the meaning of text to the dynamics, transitions, and rewards
Language helps in multi-task learning and transfer learning

Narasimhan et. al., JAIR 2018
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

*The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.*

Figure 1: An excerpt from the user manual of the game Civilization II.
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

1. Choose relevant sentences
2. Label words into action-description, state-description, or background

Branavan et. al., JAIR 2012
Language-assisted RL: Domain knowledge

● Learning to read instruction manuals

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

1. Choose **relevant** sentences
2. Label words into **action-description**, **state-description**, or **background**

Map tile attributes:
- Terrain type (e.g. grassland, mountain, etc)
- Tile resources (e.g. wheat, coal, wildlife, etc)

City attributes:
- City population
- Amount of food produced

Unit attributes:
- Unit type (e.g., worker, explorer, archer, etc)
- Is unit in a city?
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.

1. Choose relevant sentences
2. Label words into action-description, state-description, or background

Branavan et. al., JAIR 2012
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

1. Choose **relevant** sentences
2. Label words into **action-description**, **state-description**, or **background**

*The natural resources available where a population settles affects its ability to produce food and goods. Build your city on a plains or grassland square with a river running through it if possible.*

**Map tile attributes:**
- Terrain type (e.g., grassland, mountain, etc)
- Tile resources (e.g., wheat, coal, wildlife, etc)

**City attributes:**
- City population
- Amount of food produced

**Unit attributes:**
- Unit type (e.g., worker, explorer, archer, etc)
- Is unit in a city?
Language-assisted RL: Domain knowledge

● Learning to read instruction manuals

Relevant sentences

- Phalanxes are twice as effective at defending cities as warriors.
- Build the city on plains or grassland with a river running through it.
- You can rename the city if you like, but we'll refer to it as washington.
- There are many different strategies dictating the order in which advances are researched.

A: action-description
S: state-description

Branavan et. al., JAIR 2012
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

<table>
<thead>
<tr>
<th>Method</th>
<th>% Win</th>
<th>% Loss</th>
<th>Std. Err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0</td>
<td>100</td>
<td>—</td>
</tr>
<tr>
<td>Built-in AI</td>
<td>0</td>
<td>0</td>
<td>—</td>
</tr>
<tr>
<td>Game only</td>
<td>17.3</td>
<td>5.3</td>
<td>± 2.7</td>
</tr>
<tr>
<td>Sentence relevance</td>
<td>46.7</td>
<td>2.8</td>
<td>± 3.5</td>
</tr>
<tr>
<td><strong>Full model</strong></td>
<td><strong>53.7</strong></td>
<td>5.9</td>
<td>± 3.5</td>
</tr>
<tr>
<td>Random text</td>
<td>40.3</td>
<td>4.3</td>
<td>± 3.4</td>
</tr>
<tr>
<td>Latent variable</td>
<td>26.1</td>
<td>3.7</td>
<td>± 3.1</td>
</tr>
</tbody>
</table>

Grounded language learning
Ground the meaning of text to the dynamics, transitions, and rewards
Language helps in learning

Branavan et. al., JAIR 2012
Language-assisted RL: Domain knowledge

- Learning to read instruction manuals

Language is most important at the start when you don’t have a good policy. Afterwards, the model relies on game features.

Branavan et. al., JAIR 2012
Language for structuring policies

- Composing modules for Embodied QA

Q: What color is the sofa in the living room?
Language for structuring policies

- Composing modules for Embodied QA

Das et. al., CoRL 2018