MACHINE LEARNING TECHNIQUES
AND APPLICATIONS

Reinforcement Learning
Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Outline: Reinforcement learning

Topics:

1. The Big Picture
2. Basic Concepts
3. Dynamic programming
4. Monte Carlo
5. Temporal-difference learning

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Learning from Interaction

The Big Picture

Environment

agent

feedback

interaction

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
I know what I have to do, but not how to do it!
Reinforcement learning is in-between supervised and unsupervised learning.

A reward is provided at each time step, or after a number of time steps. The reward is only indicative of how well the agent performs. It does not tell the agent what other step it should have taken to obtain a better reward.

The term RL is borrowed from the literature on similar learning mechanisms in animals.
The Big Picture Reinforcement Learning (RL)

Environment

feedback

interaction

agent

reward

task

Task is encoded in the reward signal

The task is encoded in the reward signal.
Reinforcement Learning (RL)

RL tries to infer the optimal path to the goal, through a process of trial-and-error, so as to maximize the reward.

Reinforcement learning is a tedious learning method.

It is slow and is functional only in well-defined problems with small search space.
An agent in an environment. Described by:

- A set of possible states of the world (environment + agent): $S$
- A set of possible actions the agent can take: $A$
- A state transition function $P: S \times A \Rightarrow S$
- A reward function $\rho: S \times A \Rightarrow R$
- A policy $\pi: S \Rightarrow A$

At each time step, the agent is in a state $s$, takes an action $a=\pi(s)$, and thus gets a reward $r=\rho(s,a)$ and is brought to a new state $s'=P(s,a)$. 

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Find the policy $\pi^*$ that will maximize the total reward.
RL: The Means

- The state-value function: $V^\pi: S \rightarrow R$
  
  This function describes how much total reward we can expect if we are in a state $s$. It says which are the good states to be in, when following a given policy $\pi$.

- The action value function: $Q^\pi: S \times A \rightarrow R$
  
  This function describes how much total reward we can expect if we are in a state $s$ and take action $a$. It says what are the good actions to take when being in a particular state (assuming we then follow a given policy $\pi$).
Reinforcement learning works like this:
1. Choose a policy $\pi$.
2. Estimate the corresponding state-value function $V^\pi$ and action-value function $Q^\pi$.
3. Update the policy according to the action-value function. If there is a better policy go back to 2.

We will see three methods for doing this:
- Dynamic Programming
- Monte Carlo
- Temporal-difference learning
Outline: Reinforcement learning

Topics:

1. The Big Picture
2. Basic Concepts
3. Dynamic programming
4. Monte Carlo
5. Temporal-difference learning

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
What is Reinforcement Learning?

Basic Concept
What is Reinforcement Learning?

- Trial-and-Error type of learning, observed in many animals.

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Basic Concept

What is Reinforcement Learning?

- Trial-and-Error type of learning, observed in many animals.

- Well-suited for online learning on a robot.

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
What is Reinforcement Learning?

- Trial-and-Error type of learning, observed in many animals.
- Well-suited for online learning on a robot.
- Learner is not told which actions to take (i.e. no supervision), but receives a *reward* every so often.
What is Reinforcement Learning?

- Trial-and-Error type of learning, observed in many animals.
- Well-suited for online learning on a robot.
- Learner is not told which actions to take (i.e. no supervision), but receives a *reward* every so often.
- Possibility of delayed reward
  - Sacrifice short-term gains for greater long-term gains.

*Adapted from* R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Agent and environment interact at discrete time steps: \( t = 0, 1, 2, \ldots \)

Agent observes state at step \( t \):
\[
s_t \in S
\]

produces action at step \( t \):
\[
a_t \in A(s_t)
\]

gets resulting reward:
\[
r_t \in \mathcal{R}
\]

and resulting next state:
\[
s_{t+1}
\]
The Agent Learns a Policy

Basic Concept

Policy at step $t$, $\pi_t$:

- a mapping from states to action probabilities

$$\pi_t(s, a) = \text{probability to take action } a_t = a \text{ when } s_t = s$$
The Agent Learns a Policy

Basic Concept

Policy at step $t$, $\pi_t$:

- a mapping from states to action probabilities

$$\pi_t(s, a) = \text{probability to take action } a_t = a \text{ when } s_t = s$$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
The Agent Learns a Policy

Policy at step $t$, $\pi_t$:

- a mapping from states to action probabilities

\[
\pi_t(s, a) = \text{probability to take action } a_t = a \text{ when } s_t = s
\]

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.

- Roughly, the agent’s goal is to get as much reward as it can over the long run.

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
The policy determines which action $a$ to take from a state $s$.

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
The policy determines which action \( a \) to take from a state \( s \).
The policy determines which action $a$ to take from a state $s$.

Note: both the environment and the policy can be stochastic/probabilistic.
Policy

Stochastic environment: stochastic transition, an action \( a \) can lead to two different states

The policy determines which action \( a \) to take from a state \( s \)

Note: both the environment and the policy can be stochastic/probabilistic
Policy

Stochastic environment: stochastic transition, an action $a$ can lead to two different states

Stochastic policy: Actions are chosen with some probability

The policy determines which action $a$ to take from a state $s$

Note: both the environment and the policy can be stochastic/probabilistic

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Goals and Rewards

Basic Concept
Goals and Rewards

- The reward $r$ is a scalar signal that measures the success of the controller to reach its goal.
Goals and Rewards

- The reward $r$ is a scalar signal that measures the success of the controller to reach its goal.
- A goal should specify what we want to achieve, not how we want to achieve it.

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Goals and Rewards

- The reward $r$ is a scalar signal that measures the success of the controller to reach its goal.
- A goal should specify **what** we want to achieve, not **how** we want to achieve it.
- The agent must be able to measure success:
  - explicitly;
  - frequently during its lifespan.

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Returns

Suppose the sequence of rewards after step $t$ is:

$$r_{t+1}, r_{t+2}, r_{t+3}, \cdots$$

What do we want to maximize?
Suppose the sequence of rewards after step $t$ is:

$$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step $t$. 

---

*B. Concept*
Basic Concept

Returns

Suppose the sequence of rewards after step $t$ is:

$$r_{t+1}, r_{t+2}, r_{t+3}, \ldots$$

What do we want to maximize?

In general,

we want to maximize the **expected return**, $E\{R_t\}$, for each step $t$.

**Episodic tasks**: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze. The return is defined as:

$$R_t = r_{t+1} + r_{t+2} + \cdots + r_T,$$

where $T$ is a final time step at which a **terminal state** is reached, ending an episode.

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Continuing tasks: interaction does not have natural episodes, Therefore one needs to discount future rewards.

Discounted return:

\[ R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \]

where \( \gamma, 0 \leq \gamma \leq 1 \), is the discount rate.

shortsighted 0 \leftarrow \gamma \rightarrow 1 farsighted
An Example

Goal: to avoid failure, e.g. the pole falling beyond a critical angle or the cart hitting the end of the track.
An Example

Goal: to avoid failure, e.g. the pole falling beyond a critical angle or the cart hitting the end of the track.

As an episodic task where episode ends upon failure:

reward = +1 for each step before failure

⇒ return = number of steps before failure
An Example

Goal: to avoid failure, e.g. the pole falling beyond a critical angle or the cart hitting the end of the track.

As an episodic task where episode ends upon failure:

reward $= +1$ for each step before failure

$\Rightarrow$ return $= \text{number of steps before failure}$

As a continuing task with discounted return:

reward $= -1$ upon failure; 0 otherwise

$\Rightarrow$ return $= 0 + \gamma \cdot 0 + ... + \gamma^k \cdot (-1) = -\gamma^k$, for $k$ steps before failure

In either case, return is maximized by avoiding failure for as long as possible.
Another Example

Goal: Get to the top of the hill as quickly as possible.

Return is maximized by minimizing the number of steps to reach the top of the hill.

\[ \text{reward} = -1 \text{ for each step when not at the top of the hill} \]
\[ \Rightarrow \text{return} = - \text{number of steps before reaching top of the hill} \]
The value of a state is the expected return starting from that state; depends on the agent’s policy:

State-value function for policy \( \pi \):

\[
V^\pi(s) = E_\pi \left\{ R_t \mid s_t = s \right\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\}
\]

where \( E_\pi \) is the expectation operator.

The value of taking an action in a state under policy \( \pi \) is the expected return starting from that state, taking that action, and thereafter following \( \pi \):

Action-value function for policy

\[
Q^\pi(s, a) = E_\pi \{ R_t \mid s_t = s, a_t = a \}
\]
The state value function and the action value function provide the same information. They are directly related:

\[ V^\pi(s) = \sum_{a \in A} \pi(s, a) Q^\pi(s, a) \]
The state value function and the action value function provide the same information. They are directly related:

\[ V^\pi(s) = \sum_{a \in A} \pi(s, a) Q^\pi(s, a) \]

Example:

\[ V^\pi(s) = \sum_{a_i} \frac{1}{3} Q^\pi(s, a_i) \]

for an equiprobable policy
The state value function and the action value function provide the same information. They are directly related:

$$V^\pi(s) = \sum_{a \in A} \pi(s, a) Q^\pi(s, a)$$

Example:

$$V^\pi(s) = \sum_{a_i} \frac{1}{3} Q^\pi(s, a_i)$$

for an equiprobable policy

V(s) tells you how good a state is, Q(s,a) tells you how good an action from a given state is

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
These functions are very important, they can be used to improve the policy (cf the optimal value function, and the general policy iteration, later on in this lecture).
These functions are very important, they can be used to improve the policy (cf the optimal value function, and the general policy iteration, later on in this lecture).
These functions are very important, they can be used to improve the policy (cf the optimal value function, and the general policy iteration, later on in this lecture).

If $V(S_2) > V(S_1)$

→ Change policy to go to $S_2$
These functions are very important, they can be used to improve the policy (cf the optimal value function, and the general policy iteration, later on in this lecture).

If $V(S2) > V(S1)$

→ Change policy to go to S2
Reinforcement learning algorithms

Basic Concept
Reinforcement learning algorithms

- The goals of reinforcement learning algorithms are:
The goals of reinforcement learning algorithms are:

1. To correctly estimate the state (or action) value function for a given policy. Note that this is usually a very difficult task (especially if there are many states and action, and if the environment is stochastic).
The goals of reinforcement learning algorithms are:

1. To correctly estimate the state (or action) value function for a given policy. Note that this is usually a very difficult task (especially if there are many states and action, and if the environment is stochastic).

2. To use that knowledge to improve the policy, and return to 1.
The goals of reinforcement learning algorithms are:

1. To correctly estimate the state (or action) value function for a given policy. Note that this is usually a very difficult task (especially if there are many states and action, and if the environment is stochastic).

2. To use that knowledge to improve the policy, and return to 1.

And to do this until the optimal value function is found (cf next slides)
Most RL algorithms are made for problems that present the **Markov Property**:

\[
\Pr\left\{ s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0 \right\} = \\
\Pr\left\{ s_{t+1} = s', r_{t+1} = r | s_t, a_t \right\}
\]

for all \( s', r, \) and histories \( s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0 \).

In words: the probability of a transition to a new state (and new reward) depends only on the current state and action, not on the history of previous states and actions.
Markov Decision Processes

- If a reinforcement learning task has the Markov Property, it is basically a **Markov Decision Process (MDP).**
- If state and action sets are finite, it is a **finite MDP.**
- To define a finite MDP, you need to give:
  - **state and action sets**
  - one-step “dynamics” defined by **transition probabilities:**
    \[ P_{ss'}^a = \Pr\{ s_{t+1} = s'| s_t = s, a_t = a \} \text{ for all } s, s' \in S, a \in A(s). \]
  - **reward probabilities:**
    \[ R_{ss'}^a = E\{ r_{t+1} | s_t = s, a_t = a, s_{t+1} = s' \} \text{ for all } s, s' \in S, a \in A(s). \]
- MDP \( \Rightarrow \) these probabilities depend only on the current state and action. This makes things much easier!

---

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Basic Concept

MDP

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Basic Concept

MDP

$P_{ss'}^a$: Transition probability of going to state $s'$ when taking action $a$ from state $s$
Basic Concept

**MDP**

- $P_{ss'}^a$: Transition probability of going to state $s'$ when taking action $a$ from state $s$
- $R_{ss'}^a$: Corresponding expected reward

*Adapted from* R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
These two quantities define the environment in which the agent acts. For some problems (e.g. single player games), they might be completely known, in which case, you have a perfect model of the environment.
The Bellman equation is a recursive equation describing MDPs. The basic idea is:

\[
R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots
\]

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

\[
= r_{t+1} + \gamma R_{t+1}
\]
Bellman Equation for a Policy $\pi$

The Bellman equation is a recursive equation describing MDPs. The basic idea is:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots$$

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

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

So:

$$V^\pi(s) = E_\pi \left\{ R_t \mid s_t = s \right\}$$

$$= E_\pi \left\{ r_{t+1} + \gamma V(s_{t+1}) \mid s_t = s \right\}$$
Bellman Equation for a Policy $\pi$

The Bellman equation is a recursive equation describing MDPs. The basic idea is:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots$$

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

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

So:

$$V^\pi(s) = E_\pi \{ R_t | s_t = s \}$$

$$= E_\pi \{ r_{t+1} + \gamma V(s_{t+1}) | s_t = s \}$$

Or, without the expectation operator (assuming a MDP):

$$V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'} + \gamma V^\pi(s')]$$
The Bellman equation is a recursive equation describing MDPs. The basic idea is:

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

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

\[ = r_{t+1} + \gamma R_{t+1} \]

So:

\[ V^\pi(s) = E_\pi \{ R_t | s_t = s \} \]

\[ = E_\pi \{ r_{t+1} + \gamma V(s_{t+1}) | s_t = s \} \]

Or, without the expectation operator (assuming a MDP):

\[ V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')] \]
More on the Bellman Equation

Basic Concept

\[ V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right] \]

This is a set of equations (in fact, linear), one for each state. The value function for \( \pi \) is its unique solution.

Backup diagram:

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Example of a value function: Gridworld

- Actions: *north, south, east, west*; deterministic.
- If would take agent off the grid: no move but reward = −1
- Other actions produce reward = 0, except actions that move agent out of special states A and B as shown.

\[
V^\pi(s) = \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')] 
\]

\[
V^\pi(s) = \sum_{a=1}^{4} 0.25 [r_s + 0.9 \cdot V^\pi(s')] 
\]

Obtained by solving the 25 equations with 25 unknowns.

State-value function for equiprobable random policy; \(\gamma = 0.9\)

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Optimal Value Functions

Knowing the value function for a given policy is good, but what we are really looking for is the **optimal value function** and the corresponding **optimal policy**.
For finite MDPs, policies can be partially ordered:

\[ \pi \succeq \pi' \quad \text{if and only if} \quad V^\pi(s) \geq V^\pi'(s) \quad \text{for all} \quad s \in S \]
For finite MDPs, policies can be partially ordered:

$$\pi \succeq \pi' \quad \text{if and only if} \quad V^\pi (s) \geq V^{\pi'} (s) \quad \text{for all} \quad s \in S$$

There is always at least one (and possibly many) policies that is better than or equal to all the others. This is an optimal policy. We denote them all $\pi^*$. 

Basic Concept

Optimal Value Functions
For finite MDPs, policies can be partially ordered:

\[ \pi \geq \pi' \text{ if and only if } V^\pi(s) \geq V^{\pi'}(s) \text{ for all } s \in S \]

There is always at least one (and possibly many) policies that is better than or equal to all the others. This is an optimal policy. We denote them all \( \pi^* \).

Optimal policies share the same optimal state-value function:

\[ V^*(s) = \max_{\pi} V^\pi(s) \text{ for all } s \in S \]
Optimal Value Functions

- For finite MDPs, policies can be partially ordered:
  \[ \pi \geq \pi' \text{ if and only if } V^\pi(s) \geq V^{\pi'}(s) \text{ for all } s \in S \]

- There is always at least one (and possibly many) policies that is better than or equal to all the others. This is an optimal policy. We denote them all \( \pi^* \).

- Optimal policies share the same optimal state-value function:
  \[ V^*(s) = \max_\pi V^\pi(s) \text{ for all } s \in S \]

- Optimal policies also share the same optimal action-value function:
  \[ Q^*(s,a) = \max_\pi Q^\pi(s,a) \text{ for all } s \in S \text{ and } a \in A(s) \]
  This is the expected return for taking action \( a \) in state \( s \) and thereafter following an optimal policy.

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Why Optimal State-Value Functions are Useful

Any policy that always chooses the next highest $V^*$ value (i.e. is greedy) is an optimal policy.

Therefore, given $V^*$, one-step-ahead search produces the long-term optimal actions.

E.g., back to the gridworld:

![Gridworld diagram](image)
Solving the Bellman Optimality Equation

- Finding an optimal policy by solving the Bellman Optimality Equation requires the following:
  - Accurate knowledge of environment dynamics;
  - Enough space and time to do the computation;
  - The Markov Property.

- How much space and time do we need?
  - Polynomial in number of states (via dynamic programming methods; cf next slides),
  - BUT, number of states is often huge (e.g., backgammon has about $10^{20}$ states).

- We usually have to settle for approximations.

- Many RL methods can be understood as approximately solving the Bellman Optimality Equation.
Solving the Bellman Optimality Equation

Different methods:

- Dynamic programming
- Monte Carlo method
- TD (Time Delayed) learning
Reinforcement learning

Topics:

1. The Big Picture

2. Basic Concepts

3. Dynamic programming

4. Monte Carlo

5. Temporal-difference learning

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Dynamic Programming

**Dynamic Programming:**

a collection of algorithms than can be used to compute optimal policies given a **perfect model of the environment** as a Markov Decision Process (MDP)

Basic idea:

DP algorithms are iterative algorithms that

1. use the model of the environment and the Bellman equation to compute the state value function for a given policy, and
2. improve the policy based on the new state value function, and then return to 1

*Adapted from* R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Sweep through all states, and do

\[ V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')] \]
DP Policy Evaluation

Sweep through all states, and do

\[ V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')] \]
Sweep through all states, and do

\[
V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V(s')] 
\]
Sweep through all states, and do

$$V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$$
Sweep through all states, and do

\[ V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V(s') \right] \]
Sweep through all states, and do

\[ V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V(s') \right] \]
DP Policy Evaluation

Sweep through all states, and do

\[ V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^{a} [R_{ss'}^{a} + \gamma V(s')] \]
DP Policy Evaluation

Sweep through all states, and do

\[ V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')] \]
Sweep through all states, and do

\[ V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')] \]
Iterative Policy Evaluation

Input $\pi$, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in S^+$
Repeat
\[ \Delta \leftarrow 0 \]
For each $s \in S$:
\[ v \leftarrow V(s) \]
\[ V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')] \]
\[ \Delta \leftarrow \max(\Delta, |v - V(s)|) \]
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$
Iterative Policy Evaluation

Input $\pi$, the policy to be evaluated
Initialize $V(s) = 0$, for all $s \in S^+$

Repeat
\[ \Delta \leftarrow 0 \]
For each $s \in S$:
\[ v \leftarrow V(s) \]
\[ V(s) \leftarrow \sum_a \pi(s, a) \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V(s') \right] \]
\[ \Delta \leftarrow \max(\Delta, |v - V(s)|) \]
until $\Delta < \theta$ (a small positive number)
Output $V \approx V^\pi$

Perfect knowledge of the environment means perfectly knowing $P_{ss'}^a$ and $R_{ss'}^a$. 

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
A Small Gridworld

Goal: getting quickly to a terminal state (shaded squares)

An undiscounted (i.e. $\gamma=1$) episodic task

Nonterminal states: 1, 2, . . . , 14;

Actions that would take agent off the grid leave state unchanged

Reward is $-1$ until the terminal state is reached

→ Try to get out of the grid as fast as possible

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Iterative Policy Eval for the Small Gridworld

\[ V_k \text{ for the} \]

Random Policy

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Greedy Policy w.r.t. \( V_k \)

\[ \pi = \text{random (uniform)} \]

action choices

\( \pi \) = random (uniform) action choices
Iterative Policy Eval for the Small Gridworld

\[ k = 3 \]

\[
\begin{array}{cccc}
0.0 & -2.4 & -2.9 & -3.0 \\
-2.4 & -2.9 & -3.0 & -2.9 \\
-2.9 & -3.0 & -2.9 & -2.4 \\
-3.0 & -2.9 & -2.4 & 0.0 \\
\end{array}
\]

\[ k = 10 \]

\[
\begin{array}{cccc}
0.0 & -6.1 & -8.4 & -9.0 \\
-6.1 & -7.7 & -8.4 & -8.4 \\
-8.4 & -8.4 & -7.7 & -6.1 \\
-9.0 & -8.4 & -6.1 & 0.0 \\
\end{array}
\]

\[ k = \infty \]

\[
\begin{array}{cccc}
0.0 & -14.0 & -22.0 & -22.0 \\
-14.0 & -18.0 & -22.0 & -22.0 \\
-22.0 & -22.0 & -18.0 & -14.0 \\
-22.0 & -22.0 & -14.0 & 0.0 \\
\end{array}
\]

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Policy Improvement

Suppose we have computed $V^\pi$ for a deterministic policy $\pi$.

For a given state $s$,

would it be better to do an action $a \neq \pi(s)$?

The value of doing $a$ in state $s$ is:

$$Q^\pi(s, a) = E_\pi \left\{ r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a \right\}$$

$$= \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right]$$

It is better to switch to action $a$ for state $s$ if and only if

$$Q^\pi(s, a) > V^\pi(s)$$

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Policy Improvement Cont.

Do this for all states to get a new policy $\pi'$ that is **greedy** with respect to $V^\pi$:

$$
\pi'(s) = \arg\max_a Q^\pi(s,a)
= \arg\max_a \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right]
$$

Then $V^{\pi'} \geq V^\pi$
What if $V_{\pi'} = V_{\pi}$?

i.e., for all $s \in S$, $V_{\pi'}(s) = \max_a \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V_{\pi}(s') \right]$?

But this is the Bellman Optimality Equation.

So $V_{\pi'} = V^*$ and both $\pi$ and $\pi'$ are optimal policies.
Policy Iteration

\[ \pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \cdots \rightarrow \pi^* \rightarrow V^* \rightarrow \pi^* \]

policy evaluation  policy improvement

“greedification”
Policy Iteration

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

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

3. Policy Improvement
   \( \text{policy-stable} \leftarrow \text{true} \)
   For each \( s \in \mathcal{S} \):
   \[ b \leftarrow \pi(s) \]
   \[ \pi(s) \leftarrow \arg \max_a \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V(s') \right] \]
   If \( b \neq \pi(s) \), then \( \text{policy-stable} \leftarrow \text{false} \)
   If \( \text{policy-stable} \), then stop; else go to 2

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Generalized Policy Iteration (GPI):
any interaction of policy evaluation and policy improvement, independent of their granularity.

A geometric metaphor for convergence of GPI:
DP: summary

- DP algorithms applied to RL consist in iteratively updating the policy by systematically computing $V(s)$ for all states using Bellman’s equation
DP: summary

- DP algorithms applied to RL consist in iteratively updating the policy by systematically computing $V(s)$ for all states using Bellman’s equation.
- They use *bootstrapping*: updating estimates based on other estimates.
DP: summary

- DP algorithms applied to RL consist in iteratively updating the policy by systematically computing $V(s)$ for all states using Bellman’s equation.
- They use *bootstrapping*: updating estimates based on other estimates.
- Very expensive computationally: sweep over all states at each iteration.
DP: summary

- DP algorithms applied to RL consist in iteratively updating the policy by systematically computing $V(s)$ for all states using Bellman’s equation.
- They use *bootstrapping*: updating estimates based on other estimates.
- Very expensive computationally: sweep over all states at each iteration.
- DP is only applicable when:
  - The environment is perfectly known
  - The number of states is not too large
Reinforcement learning

Topics:

1. Basic concepts
2. Dynamic programming
3. Monte Carlo
4. Temporal-difference learning

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Monte Carlo Methods

- Monte Carlo methods learn from complete sample returns
  - Only defined for episodic tasks (i.e. tasks that have a clear end)

- Monte Carlo methods learn directly from experience. It can be performed *On-line*: No model necessary and still attains optimality

- No model so we have to learn $Q^\pi(s, a)$ instead of $V^\pi(s)$
Monte Carlo Policy Evaluation

Simply follow the policy during many episodes and compute the average returns

\[ R(s) = \sum_{k=0}^{T} \gamma^k r_{k+1} \]

obtained for each visited state

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Monte Carlo Policy Evaluation

Simply follow the policy during many episodes and compute the average returns

\[ R(s) = \frac{1}{T} \sum_{k=0}^{T} \gamma^k r_{k+1} \]

obtained for each visited state.
Simply follow the policy during many episodes and compute the average returns

\[ R(s) = \sum_{k=0}^{T} \gamma^k r_{k+1} \]

obtained for each visited state.
Monte Carlo Policy Evaluation

Simply follow the policy during many episodes and compute the average returns

\[ R(s) = \sum_{k=0}^{T} \gamma^k r_{k+1} \]

obtained for each visited state

\[ V(s) = \frac{R_{e1}(s) + R_{e2}(s) + R_{e3}(s)}{3} \]

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
First-visit Monte Carlo policy evaluation

Initialize:
\( \pi \leftarrow \) policy to be evaluated
\( V \leftarrow \) an arbitrary state-value function
\( Returns(s) \leftarrow \) an empty list, for all \( s \)

Repeat forever:
(a) Generate an episode using \( \pi \)
(b) For each state \( s \) appearing in the episode:
   \( R \leftarrow \) following the first occurrence of \( s \)
   Append \( R \) to \( Returns(s) \)
   \( V(s) \leftarrow \) average(\( Returns(s) \))
First-visit Monte Carlo policy evaluation

Initialize:
\( \pi \leftarrow \text{policy to be evaluated} \)
\( V \leftarrow \text{an arbitrary state-value function} \)
\( \text{Returns}(s) \leftarrow \text{an empty list, for all } s \)

Repeat forever:
(a) Generate an episode using \( \pi \)
(b) For each state \( s \) appearing in the episode:
   \( R \leftarrow \text{following the first occurrence of } s \)
   Append \( R \) to \( \text{Returns}(s) \)
   \( V(s) \leftarrow \text{average}(\text{Returns}(s)) \)

This algorithm converges to the real \( V(s) \): the more episodes, the better the approximation
Monte Carlo Estimation of Action Values (Q)

- Converges asymptotically if every state-action pair is visited.
- Importance of *Exploring starts*: Every state-action pair should have a non-zero probability of being the starting pair.
- Guarantee every (s,a) pair are selected an infinite amount of time.

- **On-Policy** or **Off-Policy** MC instead of random start state exploration.
On-Policy

- Slowly evaluate and improve the policy over time through an $\epsilon$-soft policy $\pi$

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
On-Policy

- Slowly evaluate and improve the policy over time through an $\epsilon$-soft policy $\pi$

$$Q(s, a) \leftarrow \text{arbitrary}$$

$$\pi \leftarrow \text{arbitrary}$$
On-Policy

- Slowly evaluate and improve the policy over time through an $\epsilon$-soft policy $\pi$

\[
Q(s, a) \leftarrow \text{arbitrary} \\
\pi \leftarrow \text{arbitrary}
\]

a) generate episode using $\pi$

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
On-Policy

- Slowly evaluate and improve the policy over time through an $\varepsilon$-soft policy $\pi$

  $Q(s, a) \leftarrow$ arbitrary
  $\pi \leftarrow$ arbitrary

  a) generate episode using $\pi$

  b) for each $(s, a)$ pair in episode:

  $Q(s, a) \leftarrow$ update with $R$

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
On-Policy

- Slowly evaluate and improve the policy over time through an \(\epsilon\)-soft policy \(\pi\)

\[
Q(s, a) \leftarrow \text{arbitrary} \\
\pi \leftarrow \text{arbitrary}
\]

a) generate episode using \(\pi\)

b) for each \((s, a)\) pair in episode:

\[
Q(s, a) \leftarrow \text{update with } R
\]

c) for each \(s\) in an episode:

\[
a^* \leftarrow \arg \max_a Q(s, a)
\]

for all \(a \in A(s)\):

\[
\pi(s, a) = \begin{cases} 
1 - \epsilon + \epsilon/|A(s)| & \text{if } a = a^* \\
\epsilon/|A(s)| & \text{if } a \neq a^*
\end{cases}
\]
Backup diagram for Monte Carlo

- Entire episode included
- Only one choice at each state (unlike DP)
- MC does not bootstrap
- Time required to estimate one state does not depend on the total number of states

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Monte Carlo Control

- **MC policy iteration**: Policy evaluation using MC methods followed by policy improvement
- **Policy improvement step**: greedify with respect to value (or action-value) function

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Summary

- MC has several advantages over DP:
  - Can learn directly from interacting with the environment
  - No need for full models
  - No need to learn about ALL states
  - Less harm by Markovian violations
Summary

- MC has several advantages over DP:
  - Can learn directly from interacting with the environment
  - No need for full models
  - No need to learn about ALL states
  - Less harm by Markovian violations
- MC methods provide an alternate policy evaluation process
Summary

MC has several advantages over DP:
- Can learn directly from interacting with the environment
- No need for full models
- No need to learn about ALL states
- Less harm by Markovian violations

MC methods provide an alternate policy evaluation process

One issue to watch for: maintaining sufficient exploration
- exploring starts, soft policies
Summary

- MC has several advantages over DP:
  - Can learn directly from interacting with the environment
  - No need for full models
  - No need to learn about ALL states
  - Less harm by Markovian violations
- MC methods provide an alternate policy evaluation process
- One issue to watch for: maintaining sufficient exploration
  - exploring starts, soft policies
- No bootstrapping (as opposed to DP)
Reinforcement learning

Topics:

1. Basic concepts
2. Dynamic programming
3. Monte Carlo
4. Temporal-difference learning

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Temporal Difference (TD) learning

- TD learning combines ideas from DP and from MC
- Like MC, TD methods learn directly from raw experience
- Like DP, TD methods do bootstrapping, i.e. they update estimates without waiting for a final outcome
Simple Monte Carlo

\[ V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)] \]

where \( R_t \) is the actual return following state \( s_t \).
cf. Dynamic Programming

\[ V(s_t) \leftarrow E_\pi \left\{ r_{t+1} + \gamma V(s_t) \right\} \]
Simplest TD Method

\[ V(s_t) \leftarrow V(s_t) + \alpha \left[ r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right] \]
TD Prediction

Policy Evaluation (the prediction problem):
for a given policy $\pi$, compute the state-value function $V^\pi$

Recall: Simple every-visit Monte Carlo method:

$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$

↑

**target**: the actual return after time $t$

The simplest TD method, TD(0):

$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$

↓

**target**: an estimate of the return

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
TD Bootstraps and Samples

**Bootstrapping**: update involves an estimate
- MC does not bootstrap
- DP bootstraps
- TD bootstraps

**Sampling**: update does not involve an expected value (i.e. an average over probabilities)
- MC samples
- DP does not sample
- TD samples

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Advantages of TD Learning
Advantages of TD Learning

- TD methods do not require a model of the environment, only experience.
Advantages of TD Learning

- TD methods do not require a model of the environment, only experience
- TD, but not MC, methods can be fully incremental
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences
Advantages of TD Learning

- TD methods do not require a model of the environment, only experience
- TD, but not MC, methods can be fully incremental
  - You can learn before knowing the final outcome
    - Less memory
    - Less peak computation
  - You can learn without the final outcome
    - From incomplete sequences
- Both MC and TD converge

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Sarsa On-Policy TD Control

- Perform GPI with TD to get $Q^*(s, a)$
Sarsa On-Policy TD Control

- Perform **GPI** with TD to get $Q^*(s, a)$
  1. Generate episode from $Q^\pi(s, a)$

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Sarsa On-Policy TD Control

- Perform **GPI** with TD to get $Q^*(s, a)$

  1. Generate episode from $Q^\pi(s, a)$

  $S \leftarrow$ initialise the starting state (random?)

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Sarsa On-Policy TD Control

- Perform **GPI** with TD to get $Q^*(s, a)$
  1) Generate episode from $Q^\pi(s, a)$

  - $S$ ← initialise the starting state (random ?)
  - $a$ ← pic action $\epsilon$-greedy from $Q^\pi(s, a)$

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Sarsa On-Policy TD Control

 Perform GPI with TD to get $Q^*(s, a)$

1) Generate episode from $Q^\pi(s, a)$

- Initialise the starting state (random ?)

- Pick action $\epsilon$-greedy from $Q^\pi(s, a)$

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Sarsa On-Policy TD Control

- Perform **GPI** with TD to get $Q^*(s, a)$
  1) Generate episode from $Q^\pi(s, a)$

\[ S \quad \leftarrow \text{initialise the starting state (random ?)} \]

\[ \begin{align*} 
  S & \quad \leftarrow \text{pick action } \epsilon\text{-greedy from } Q^\pi(s, a) \\
  \alpha & \quad \leftarrow \text{pick action } \epsilon\text{-greedy from } Q^\pi(s, a) \\
  r & \\
  S' & \quad \leftarrow \text{pick action } \epsilon\text{-greedy from } Q^\pi(s, a) \\
  \alpha' \end{align*} \]

*Adapted from* R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
Sarsa On-Policy TD Control

- Perform **GPI** with TD to get $Q^*(s, a)$
  1) Generate episode from $Q^\pi(s, a)$

$S$ ← initialise the starting state (random ?)

$\alpha$ ← pic action $\epsilon$-greedy from $Q^\pi(s, a)$

$S' \quad r$

$\alpha' \quad$ ← pic action $\epsilon$-greedy from $Q^\pi(s, a)$

$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$

*Adapted from* R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction
TD-Gammon

Tesauro, 1992–1995

Start with a random network
Play very many games against self
Learn a value function from this simulated experience

This produces arguably the best player in the world

Adapted from R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction