

Topics:

- Reinforcement Learning Part 1
 - **Markov Decision Processes**
 - **Value Iteration**

CS 4803-DL / 7643-A
ZSOLT KIRA

Reinforcement Learning Introduction

Supervised Learning

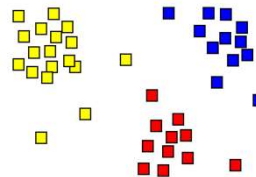
- Train Input: $\{X, Y\}$
- Learning output:
 $f : X \rightarrow Y, P(y|x)$
- e.g. classification



→
Sheep
Dog
Cat
Lion
Giraffe

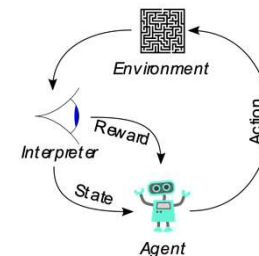
Unsupervised Learning

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



Reinforcement Learning

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



Types of Machine Learning

RL: Sequential decision making in an environment with evaluative feedback.

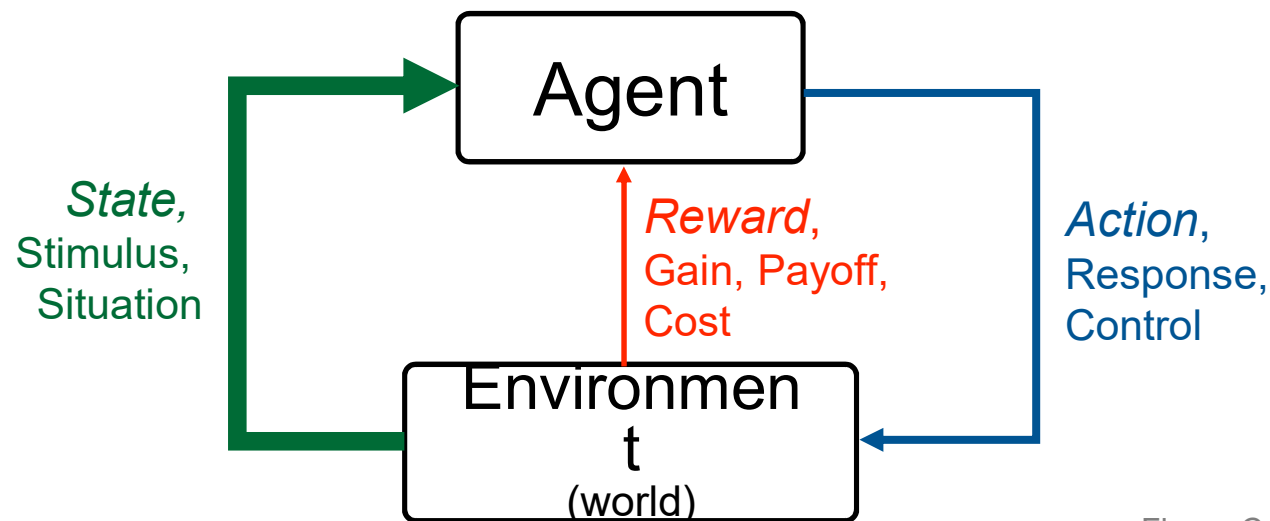


Figure Credit: Rich Sutton

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

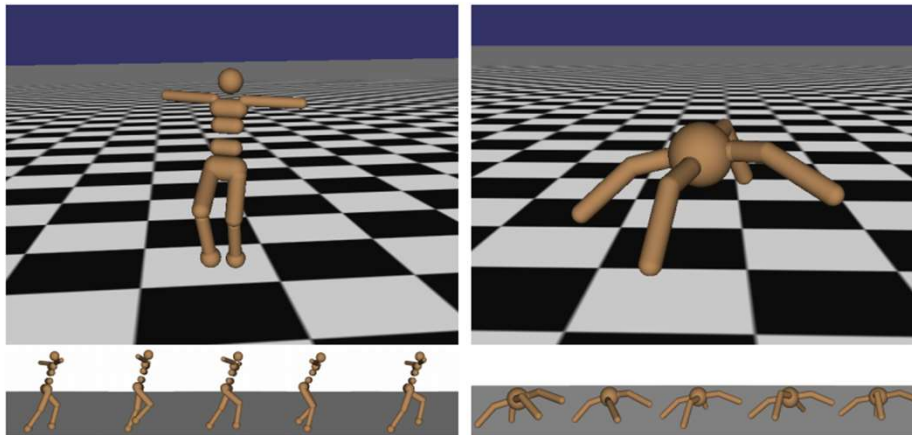
What is Reinforcement Learning?

Signature Challenges in Reinforcement Learning

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

Slide adapted from: Richard Sutton

Robot Locomotion



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

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

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

Examples of RL tasks

Atari Games



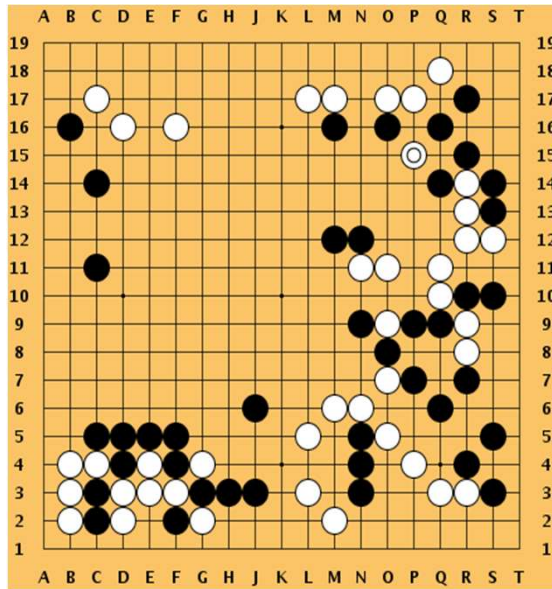
- **Objective:** Complete the game with the highest score
- **State:** Raw pixel inputs of the game state
- **Action:** Game controls e.g. Left, Right, Up, Down
- **Reward:** Score increase/decrease at each time step

Figures copyright Volodymyr Mnih et al., 2013. Reproduced with permission.

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

Examples of RL tasks

Go



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

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

Examples of RL tasks

Markov Decision Processes

- **MDPs:** Theoretical framework underlying RL

- **MDPs:** Theoretical framework underlying RL
- An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$
 - \mathcal{S} : Set of possible states
 - \mathcal{A} : Set of possible actions
 - $\mathcal{R}(s, a, s')$: Distribution of reward
 - $\mathbb{T}(s, a, s')$: **Transition** probability distribution, also written as $p(s'|s,a)$
 - γ : Discount factor

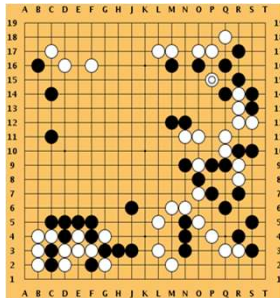
- **MDPs:** Theoretical framework underlying RL
- An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$
 - \mathcal{S} : Set of possible states
 - \mathcal{A} : Set of possible actions
 - $\mathcal{R}(s, a, s')$: Distribution of reward
 - $\mathbb{T}(s, a, s')$: Transition probability distribution, also written as $p(s'|s,a)$
 - γ : Discount factor
- **Interaction trajectory:** $\dots s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, \dots$

- **MDPs:** Theoretical framework underlying RL
- An MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$
 - \mathcal{S} : Set of possible states
 - \mathcal{A} : Set of possible actions
 - $\mathcal{R}(s, a, s')$: Distribution of reward
 - $\mathbb{T}(s, a, s')$: Transition probability distribution, also written as $p(s'|s,a)$
 - γ : Discount factor
- **Interaction trajectory:** $\dots s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, \dots$
- **Markov property:** Current state **completely** characterizes state of the environment
- **Assumption:** **Most recent observation** is a sufficient statistic of history
$$p(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, \dots, S_0 = s_0) = p(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

Markov Decision Processes (MDPs)

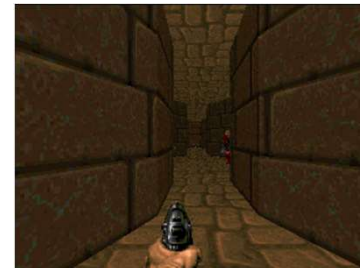
Fully observed MDP

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



Partially observed MDP

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



Source: <https://github.com/mwydmuch/ViZDoom>

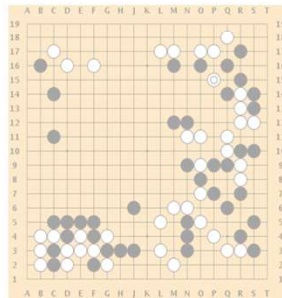
Fully observed MDP

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

Partially observed MDP

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

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



Source: <https://github.com/mwydmuch/ViZDoom>

- In **Reinforcement Learning**, we assume an underlying **MDP** with unknown:
 - Transition probability distribution \mathbb{T}
 - Reward distribution \mathcal{R}

$$\text{MDP} \\ (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$$

- In **Reinforcement Learning**, we assume an underlying **MDP** with unknown:
 - Transition probability distribution \mathbb{T}
 - Reward distribution \mathcal{R}
- Evaluative feedback comes into play, trial and error necessary

$$\text{MDP} \\ (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$$

- In **Reinforcement Learning**, we assume an underlying **MDP** with unknown:
 - Transition probability distribution \mathbb{T}
 - Reward distribution \mathcal{R}
- MDP
 $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{T}, \gamma)$
- Evaluative feedback comes into play, trial and error necessary
 - For this lecture, **assume that we know the true reward and transition distribution** and look at algorithms for **solving MDPs** i.e. finding the best policy
 - Rewards known everywhere, no evaluative feedback
 - Know how the world works i.e. all transitions

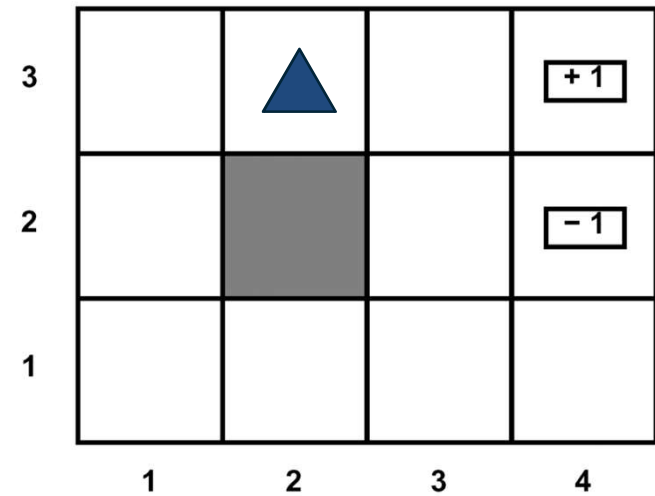


Figure credits: Pieter Abbeel

A Grid World MDP

- Agent lives in a 2D grid environment

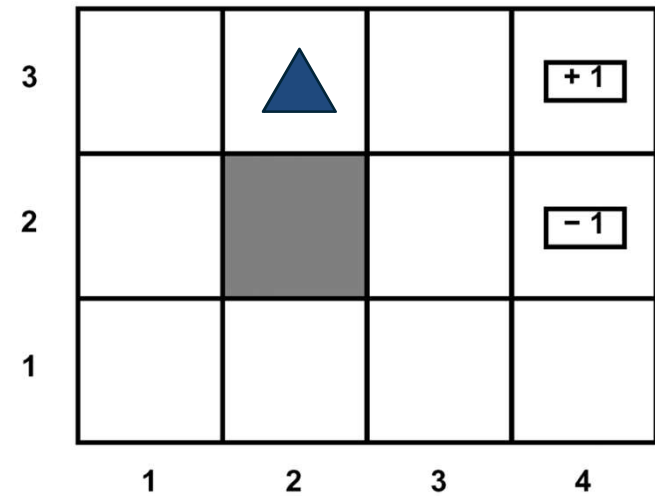


Figure credits: Pieter Abbeel

A Grid World MDP

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

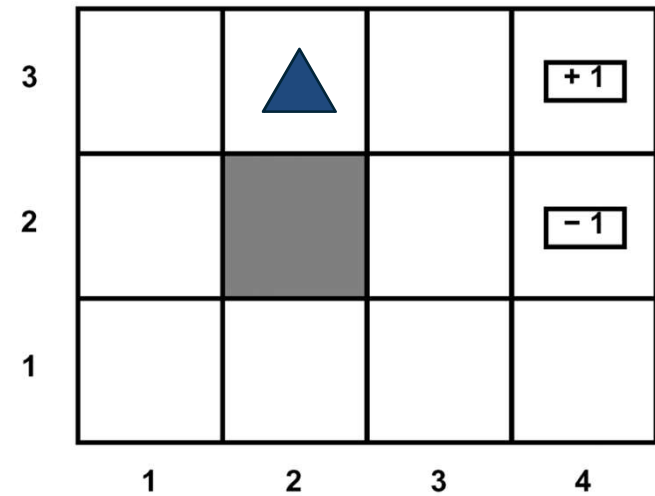


Figure credits: Pieter Abbeel

A Grid World MDP

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

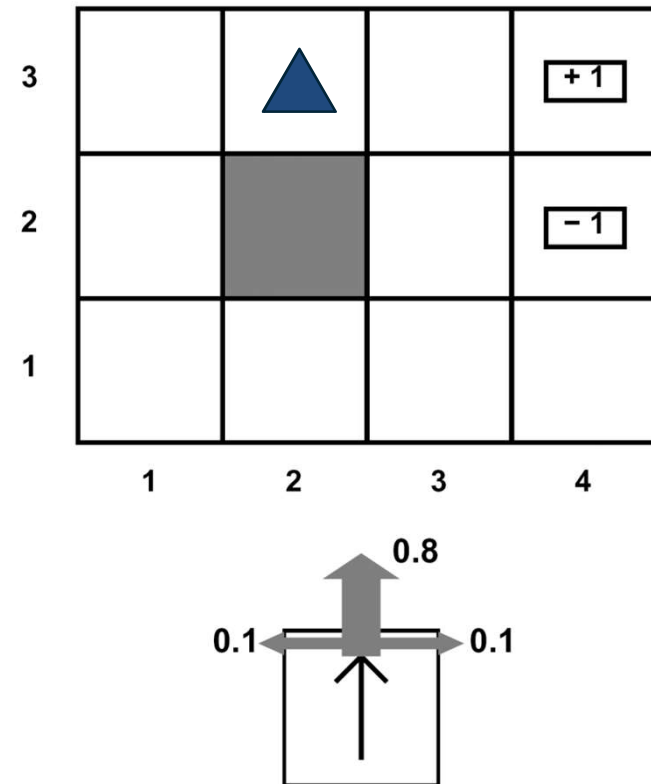


Figure credits: Pieter Abbeel

A Grid World MDP

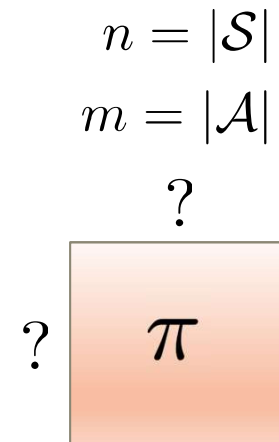
- Solving MDPs by finding the **best/optimal policy**

- Solving MDPs by finding the **best/optimal policy**
- Formally, a **policy** is a mapping from states to actions

e.g.

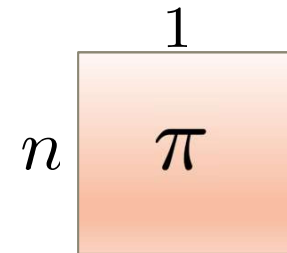
State	Action
A	→ 2
B	→ 1

- Solving MDPs by finding the **best/optimal policy**
- Formally, a **policy** is a mapping from states to actions
 - Deterministic $\pi(s) = a$



- Solving MDPs by finding the **best/optimal policy**
- Formally, a **policy** is a mapping from states to actions
 - Deterministic $\pi(s) = a$

$$n = |\mathcal{S}|$$
$$m = |\mathcal{A}|$$

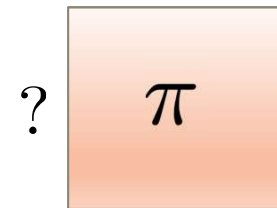


- Solving MDPs by finding the **best/optimal policy**
- Formally, a **policy** is a mapping from states to actions
 - Deterministic $\pi(s) = a$
 - Stochastic $\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$

$$n = |\mathcal{S}|$$

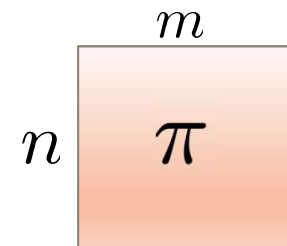
$$m = |\mathcal{A}|$$

?



- Solving MDPs by finding the **best/optimal policy**
- Formally, a **policy** is a mapping from states to actions
 - Deterministic $\pi(s) = a$
 - Stochastic $\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$

$$n = |\mathcal{S}|$$
$$m = |\mathcal{A}|$$



- Solving MDPs by finding the **best/optimal policy**
- Formally, a **policy** is a mapping from states to actions
 - Deterministic $\pi(s) = a$
 - Stochastic $\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$
- What is a good policy?
 - Maximize **current reward**? Sum of all **future rewards**?
 - **Discounted sum of future rewards!**
 - Discount factor: γ



1

Worth Now



γ

Worth Next Step



γ^2

Worth In Two Steps

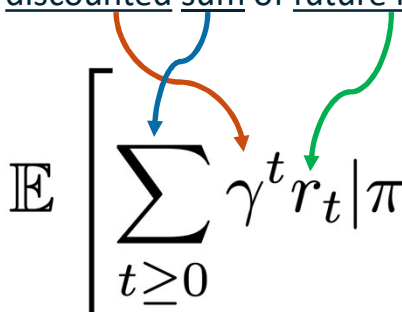
Solving MDPs: Optimal policy

- Formally, the **optimal policy** is defined as:

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

- Formally, the **optimal policy** is defined as:

discounted sum of future rewards

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


- Formally, the **optimal policy** is defined as:

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

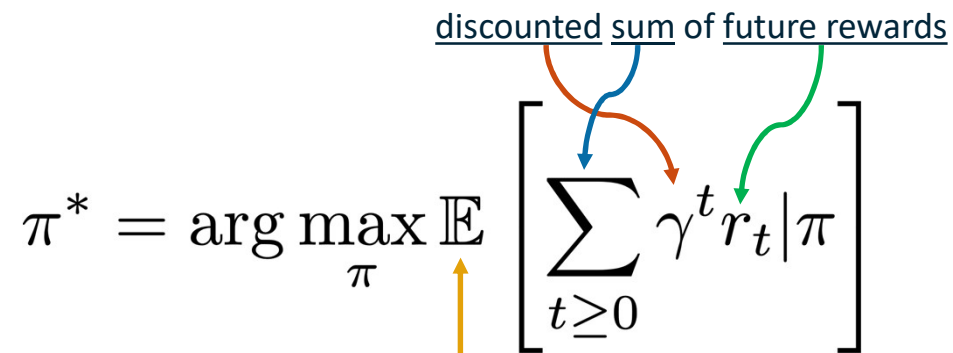
discounted sum of future rewards

?

- Formally, the **optimal policy** is defined as:

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

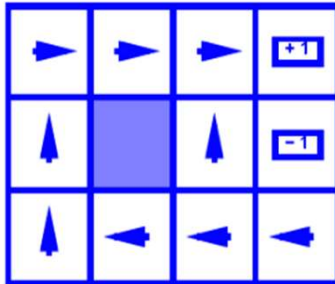
discounted sum of future rewards



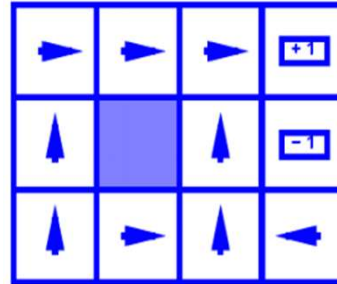
$$s_0 \sim p(s_0), a_t \sim \pi(\cdot | s_t), s_{t+1} \sim p(\cdot | s_t, a_t)$$

Expectation over initial state, actions from policy,
next states from transition distribution

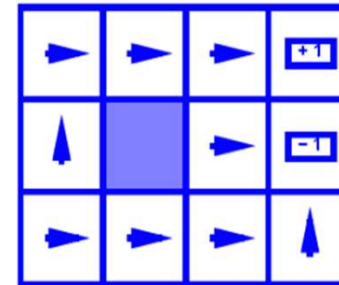
- Some optimal policies for three different grid world MDPs ($\gamma=0.99$)
 - Varying reward for non-absorbing states (states other than +1/-1)



$$R(s) = -0.03$$



$$R(s) = -0.4$$



$$R(s) = -2.0$$

Image Credit: Byron Boots, CS 7641

Optimal policy examples

- A **value function** is a prediction of discounted sum of future reward

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

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

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

- A **value function** is a prediction of discounted sum of future reward
- **State** value function / **V**-function / $V : \mathcal{S} \rightarrow \mathbb{R}$
 - How good is this state?
 - Am I likely to win/lose the game from this state?
- **State-Action** value function / **Q**-function / $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
 - How good is this state-action pair?
 - In this state, what is the impact of this action on my future?

- For a policy that produces a trajectory sample $(s_0, a_0, s_1, a_1, s_2 \dots)$

- For a policy that produces a trajectory sample $(s_0, a_0, s_1, a_1, s_2 \dots)$
- The **V-function** of the policy at state s , is the expected cumulative reward from state s :

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

- For a policy that produces a trajectory sample $(s_0, a_0, s_1, a_1, s_2 \dots)$
- The **V-function** of the policy at state s , is the expected cumulative reward from state s :

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

↑
?

- For a policy that produces a trajectory sample $(s_0, a_0, s_1, a_1, s_2 \dots)$
- The **V-function** of the policy at state s , is the expected cumulative reward from state s :

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

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

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

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

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

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

- The V and Q functions corresponding to the optimal policy π^*

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

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

Recursive Bellman expansion (from definition of Q)

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

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

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

- Equations relating optimal quantities

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

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

- Recursive Bellman optimality equation

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

- Equations relating optimal quantities

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

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

- Recursive Bellman optimality equation

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

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

Based on the **bellman optimality equation**

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

Algorithm

- Initialize values of all states

- While not converged:

- For each state: $V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s, a) [r(s, a) + \gamma V^i(s')]$

- Repeat until convergence (no change in values)

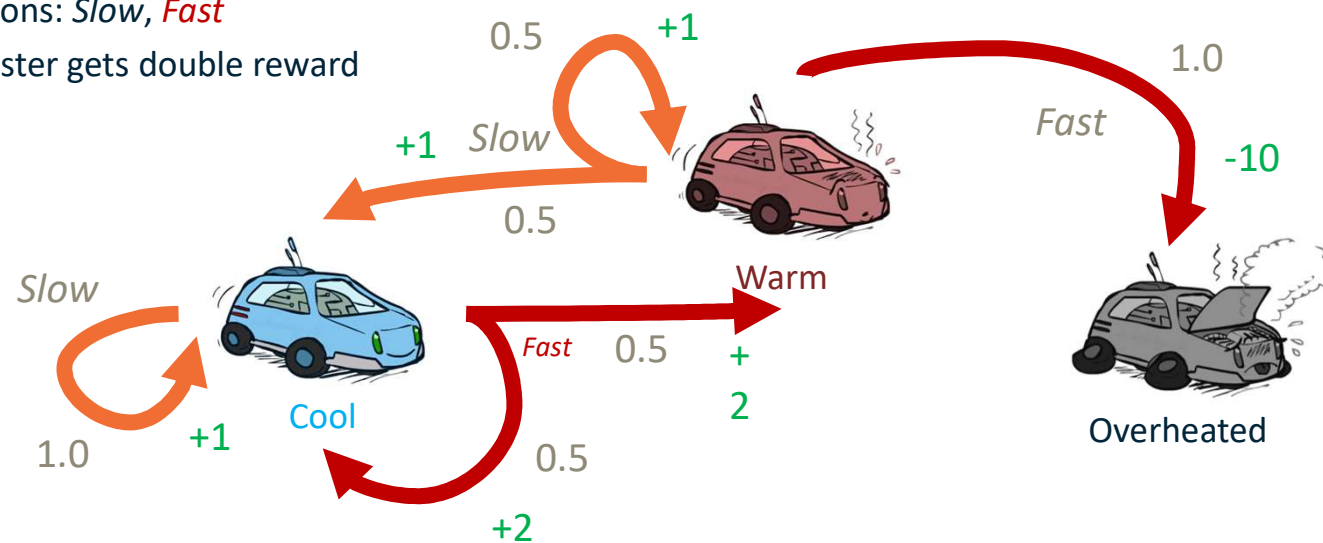
$$V^0 \rightarrow V^1 \rightarrow V^2 \rightarrow \dots \rightarrow V^i \rightarrow \dots \rightarrow V^*$$

Time complexity per iteration $O(|\mathcal{S}|^2 |\mathcal{A}|)$

Value Iteration

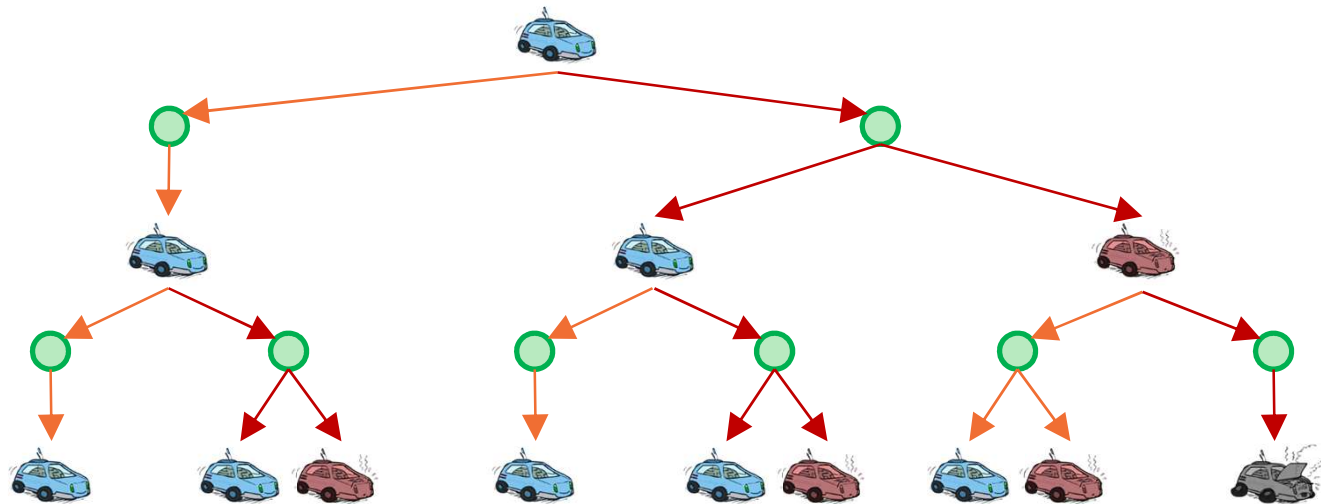


- A robot car wants to travel far, quickly
- Three states: **Cool**, **Warm**, Overheated
- Two actions: *Slow*, *Fast*
- Going faster gets double reward



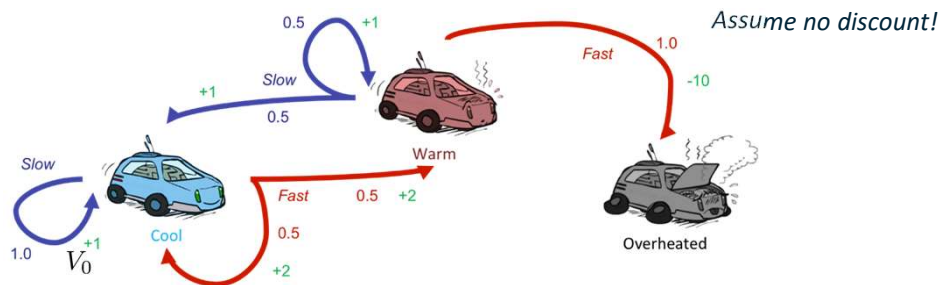
Slide Credit: <http://ai.berkeley.edu>

Example: Racing






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

Racing Search Tree

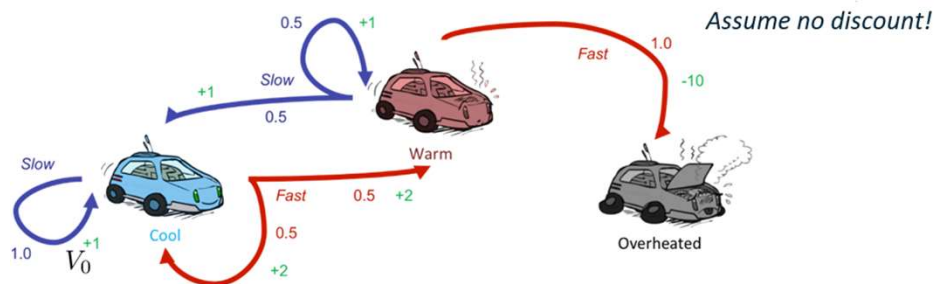





$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V_k(s')]$$

			
V_2	3.5	2.5	0
V_1	2	1	0
	0	0	0

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

Racing Search Tree



V_2

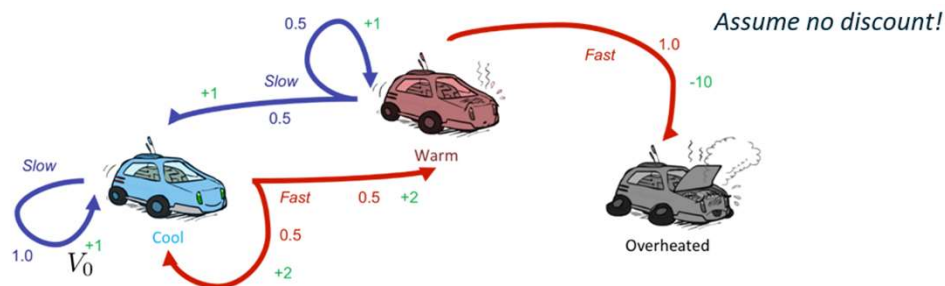
V_1

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

$$\begin{aligned}
 V_1(s_0) &= \max \left(\overset{\text{slow}}{\left[1.0 \cdot (+1 + V(s_1)) \right]}, \overset{\text{Fast}}{\left[0.5 [+2 + V(s_1)] + 0.5 [+2 \cdot V(s_0)] \right]} \right) \\
 &= 2
 \end{aligned}$$

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

Racing Search Tree



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

$$V_2(s_0) = \max \left[\begin{array}{l} \text{slow} \\ 1.0 (+1 + V_1(s_0)) \\ \text{fast} \\ 0.5 (+2 + V_1(s_0)) + 0.5 (-10 + V_1(s_1)) \end{array} \right]$$

= 3.5

V_2	3.5	2.5	0
V_1	2	1	0
	0	0	0

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

Racing Search Tree

Value Iteration Update:

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

Q-Iteration Update:

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

The algorithm is same as value iteration, but it loops over actions as well as states

For Value Iteration:

Theorem: will converge to unique optimal values

Basic idea: approximations get refined towards optimal values

Policy may converge long before values do

Time complexity per iteration $O(|\mathcal{S}|^2|\mathcal{A}|)$

Feasible for:

- ✦ 3x4 Grid world?
- ✦ Chess/Go?
- ✦ Atari Games with integer image pixel values [0, 255] of size 16x16 as state?

Summary: MDP Algorithms

Value Iteration

- ◆ Bellman update to state value estimates

Q-Value Iteration

- ◆ Bellman update to (state, action) value estimates



Reinforcement Learning, Deep RL

- Recall RL assumptions:
 - $\mathbb{T}(s, a, s')$ unknown, how actions affect the environment.
 - $\mathcal{R}(s, a, s')$ unknown, what/when are the good actions?
- But, we can learn by trial and error.
 - Gather experience (data) by performing actions.
 - Approximate unknown quantities from data.

Reinforcement Learning

- Old Dynamic Programming Demo
 - https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html
- RL Demo
 - https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

Slide credit: Dhruv Batra

Q-Learning

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

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

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

- Instead, compute average as we go

- Receive a sample transition (s,a,r,s')
- This sample suggests

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

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

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

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy -- even if you're acting suboptimally!
- This is called **off-policy learning**
- Caveats:
 - You have to explore enough
 - You have to eventually make the learning rate small enough
 - ... but not decrease it too quickly
 - Basically, in the limit, it doesn't matter how you select action

