Topics:

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

CS 4803-DL / 7643-A

ZSOLT KIRA

- **Assignment 4 out**
  - Due date **extended** to **April 8th 11:59pm EST.**

- **Projects**
  - Will try to get **feedback** back to you in next few days or so (grading will be separate)

- Outline of rest of course:
  - Reinforcement Learning
  - Guest lectures/other topics (e.g. self-supervised learning, audio)
    - **April 7th:** Wav2Vec !!
    - **April 9th:** Ishan Misra (FB) on Self-Supervised Learning
    - **April 14th:** Automatic Speech Recognition Systems
  - Generative models (VAEs / GANs)

Nirbhay Modhe

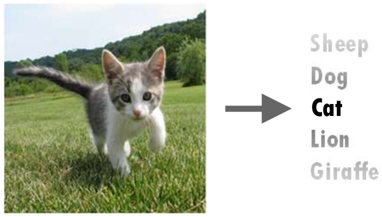
Nirbhay Modhe is a PhD Student in the School of Interactive Computing at Georgia Tech advised by Prof. Dhruv Batra. His research interests within Reinforcement Learning (RL) include model based RL, generalization guarantees in RL and unsupervised or reward-free RL for exploration. Prior to starting his PhD program in 2017, he received his Bachelor's degree in Computer Science at the Indian Institute of Technology (IIT), Kanpur where he worked with Prof. Piyush Rai on Bayesian ML applied to multi-label learning.

Slides Brought to You By…

Reinforcement Learning Introduction

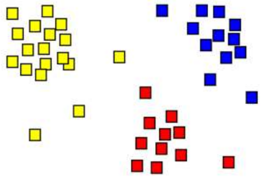
Supervised Learning

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



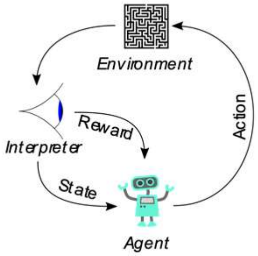
Unsupervised Learning

- 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

Georgia Tech

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

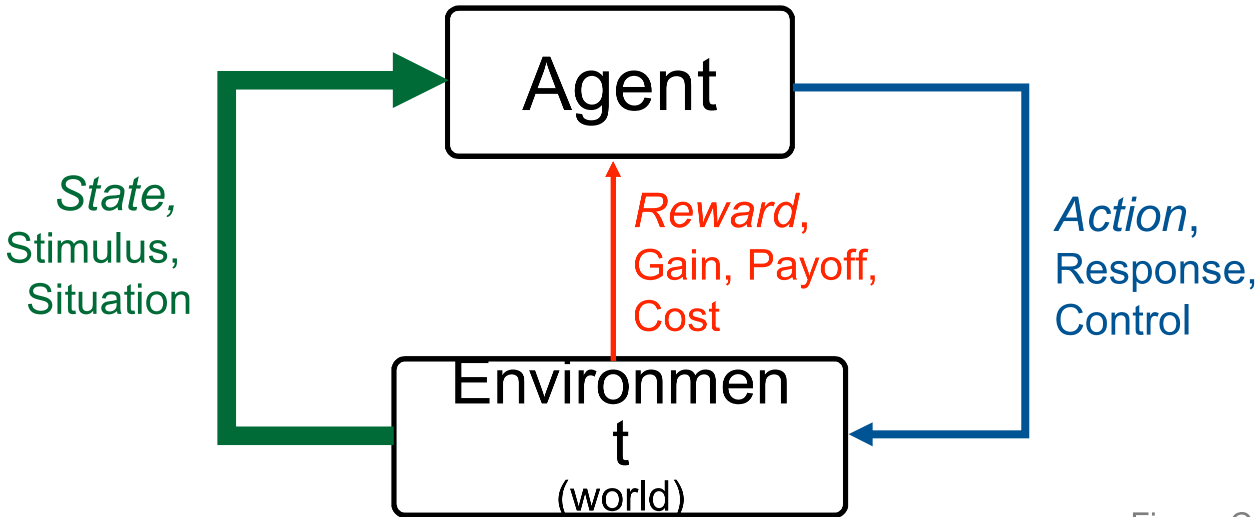


Figure Credit: Rich Sutton

- **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?

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

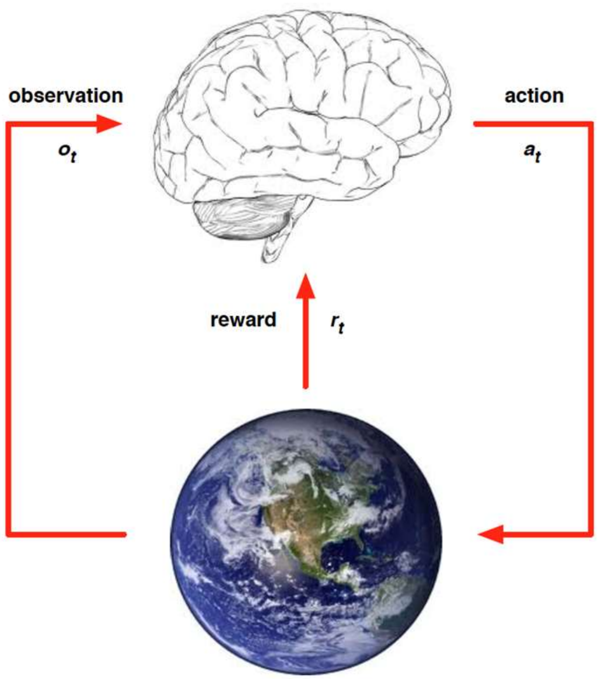
Evaluative Feedback

- Pick an action, receive a reward (positive or negative)
- No supervision for what the "correct" action is or would have been, unlike supervised learning

Sequential Decisions

- Plan and execute actions over a sequence of states
- Reward may be delayed, requiring optimization of future rewards (long-term planning).

RL: Environment Interaction API



- At each time step t, the agent:
  - Receives observation o_t
  - Executes action a_t

- At each time step t, the environment:
  - Receives action a_t
  - Emits observation o_{t+1}
  - Emits scalar reward r_{t+1}

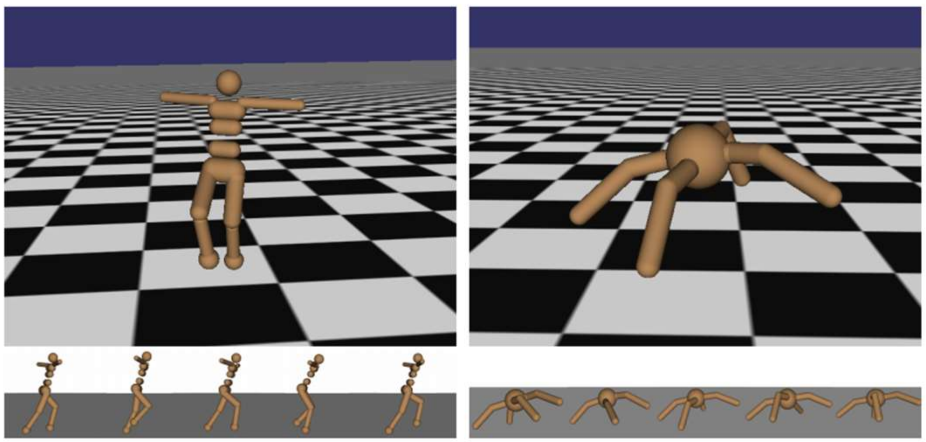
Slide credit: David Silver

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

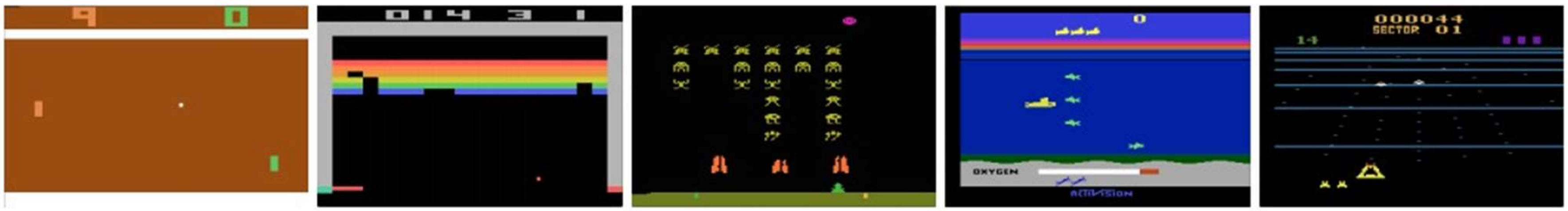


Figures copyright John Schulman et al., 2016. Reproduced with permission.

- **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

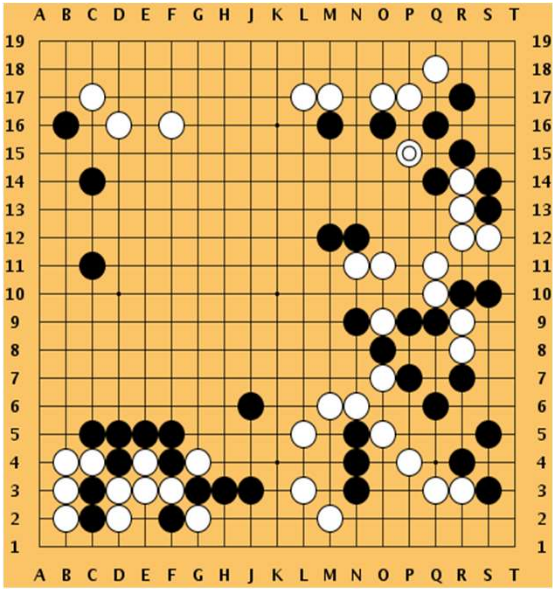
Atari Games



- **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

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

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

Markov Decision Processes

- **MDPs**: Theoretical framework underlying RL

Markov Decision Processes (MDPs)

Georgia Tech

- **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

Markov Decision Processes (MDPs)

- **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**: $\ldots s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, \ldots$

Markov Decision Processes (MDPs)

- **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**: $\ldots s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}, r_{t+2}, s_{t+2}, \ldots$
- **Markov property**: Current state completely characterizes state of the environment
- **Assumption**: Most recent observation is a sufficient statistic of history

$$p\left(S_{t+1} = s' | S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, \ldots S_0 = s_0\right) = p\left(S_{t+1} = s' | S_t = s_t, A_t = a_t\right)$$

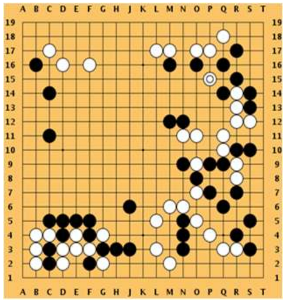
Markov Decision Processes (MDPs)

Georgia Tech

MDP Variations

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

Georgia Tech

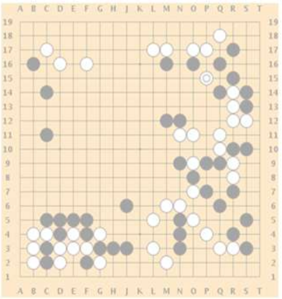
Fully observed MDP

- 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

MDP Variations

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

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

MDPs in the context of RL

Georgia Tech

- 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

MDPs in the context of RL

- 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

MDPs in the context of RL

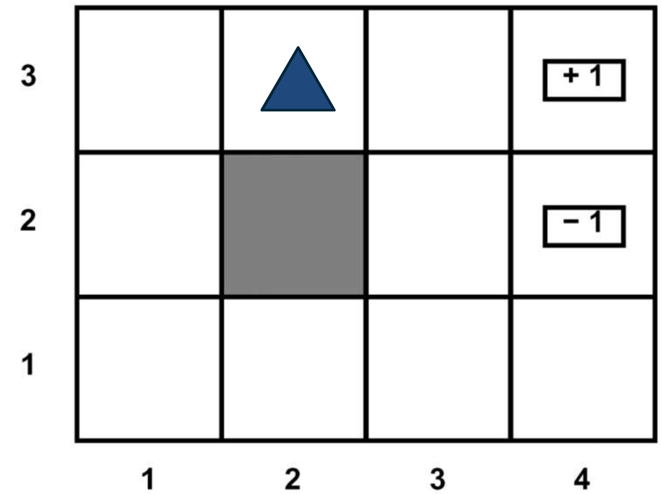


Figure credits: Pieter Abbeel

A Grid World MDP

- Agent lives in a 2D grid environment

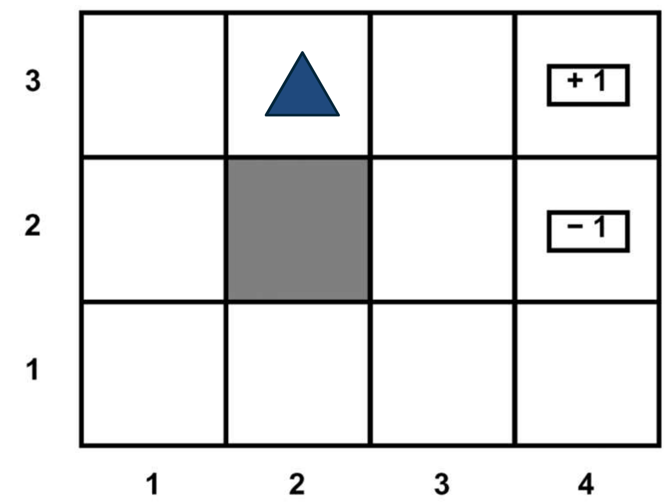


Figure credits: Pieter Abbeel

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

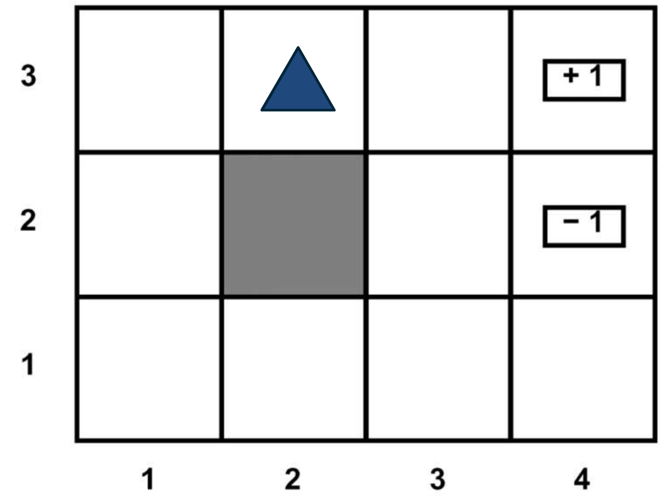


Figure credits: Pieter Abbeel

A Grid World MDP

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 to not always go as planned
  - 20% chance that agent drifts one cell left or right of direction of motion (except when blocked by wall).

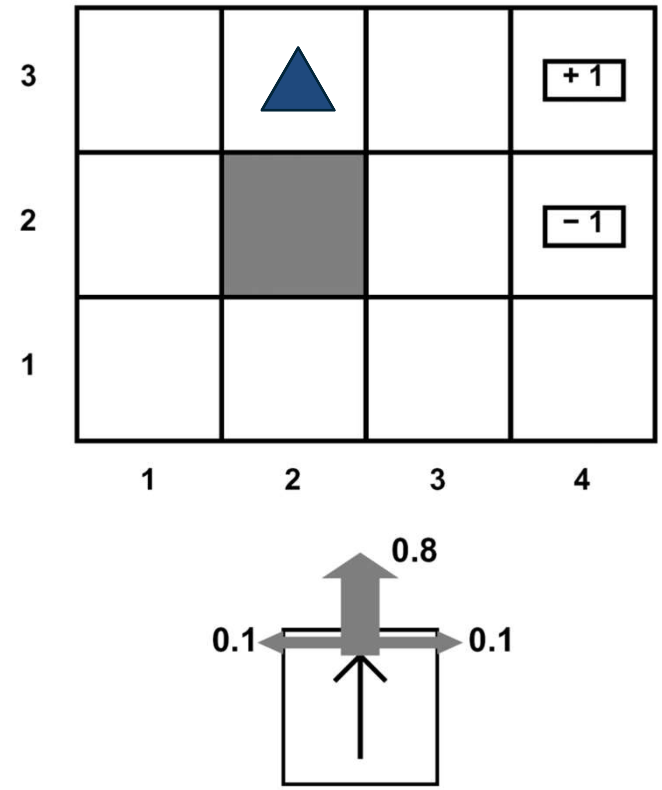


Figure credits: Pieter Abbeel

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

Solving MDPs: Optimal policy

Georgia Tech

- 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: Optimal policy

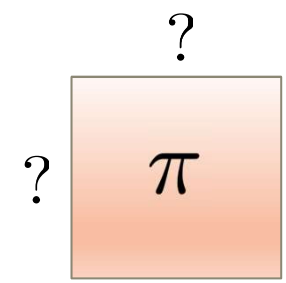
Solving MDPs: Optimal policy

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

- Formally, a **policy** is a mapping from states to actions
  - Deterministic $\pi(s) = a$

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

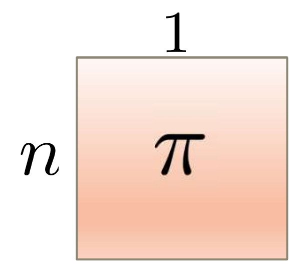
$$m = |\mathcal{A}|$$



Solving MDPs: Optimal policy

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

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

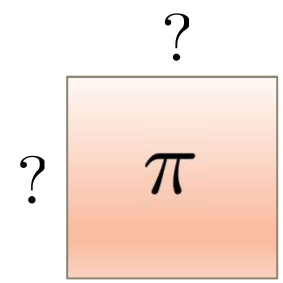


Solving MDPs: Optimal policy

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

- Formally, a **policy** is a mapping from states to actions
  - Deterministic $\pi(s) = a$
  - Stochastic $\pi(a|s) = \mathbb{P}(A_t = a | S_t = s)$

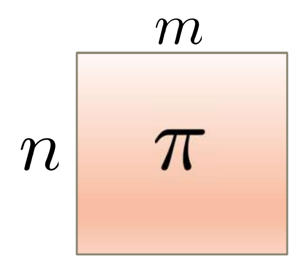
$$n = |\mathcal{S}|$$
$$m = |\mathcal{A}|$$



Solving MDPs: Optimal policy

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

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



- 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

Solving MDPs: Optimal policy

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

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

Solving MDPs: Optimal policy

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

discounted sum of future rewards

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

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

discounted sum of future rewards

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

?

Solving MDPs: Optimal policy

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

discounted sum of future rewards

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

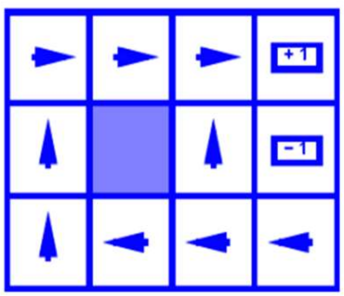
$$\boldsymbol{s}_0 \sim p\left(\boldsymbol{s}_0\right), a_t \sim \pi\left(\cdot|\boldsymbol{s}_t\right), \boldsymbol{s}_{t+1} \sim p\left(\cdot|\boldsymbol{s}_t, a_t\right)$$

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

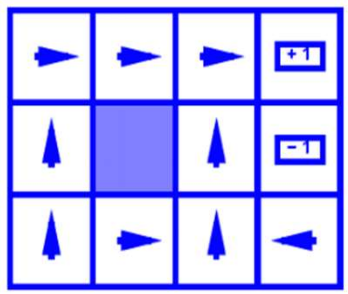
Solving MDPs: Optimal policy

Optimal policy examples

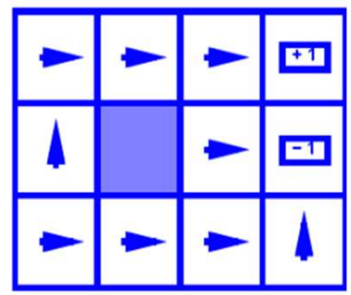
- Some optimal policies for three different grid world MDPs (gamma=0.99)
  - Varying reward for non-absorbing states (states other than +1/-1)



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

Image Credit: Byron Boots, CS 7641

Georgia Tech

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

Value Function

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

Value Function

Georgia Tech

- 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 / $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

Value Function

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

- **State** value function / **V**-function / $V : \mathcal{S} \to \mathbb{R}$
  - How good is this state?
  - Am I likely to win/lose the game from this state?

- **State-Action** value function / **Q**-function / $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$
  - How good is this state-action pair?
  - In this state, what is the impact of this action on my future?

Value Function

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

Value Function

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

- The **V-function** of the policy at state s, is the expected cumulative reward from state s:

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

Value Function

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

- The **V-function** of the policy at state s, is the expected cumulative reward from state s:

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

?

Value Function

Georgia Tech

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

- The **V-function** of the policy at state s, is the expected cumulative reward from state s:

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

$$\boldsymbol{s}_0 \sim p\left(\boldsymbol{s}_0\right), a_t \sim \pi\left(\cdot|\boldsymbol{s}_t\right), \boldsymbol{s}_{t+1} \sim p\left(\cdot|\boldsymbol{s}_t, a_t\right)$$

Value Function

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

- 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):

Action-Value Function

Georgia Tech

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

- 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):

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

$$\boldsymbol{s}_0 \sim p(\boldsymbol{s}_0), a_t \sim \pi(\cdot|\boldsymbol{s}_t), \boldsymbol{s}_{t+1} \sim p(\cdot|\boldsymbol{s}_t, a_t)$$

Action-Value Function

Georgia Tech

- The V and Q functions corresponding to the optimal policy $\pi^\star$

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

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

Optimal V & Q functions

Georgia Tech

Recursive Bellman expansion (from definition of Q)

(Expected) return from t = 0

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

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

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

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

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

- Equations relating optimal quantities

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

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

- Recursive Bellman optimality equation

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

Bellman Optimality Equations

- Equations relating optimal quantities

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

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

- Recursive Bellman optimality equation

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

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

Bellman Optimality Equations

Georgia Tech

Based on the **bellman optimality equation**

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

Algorithm

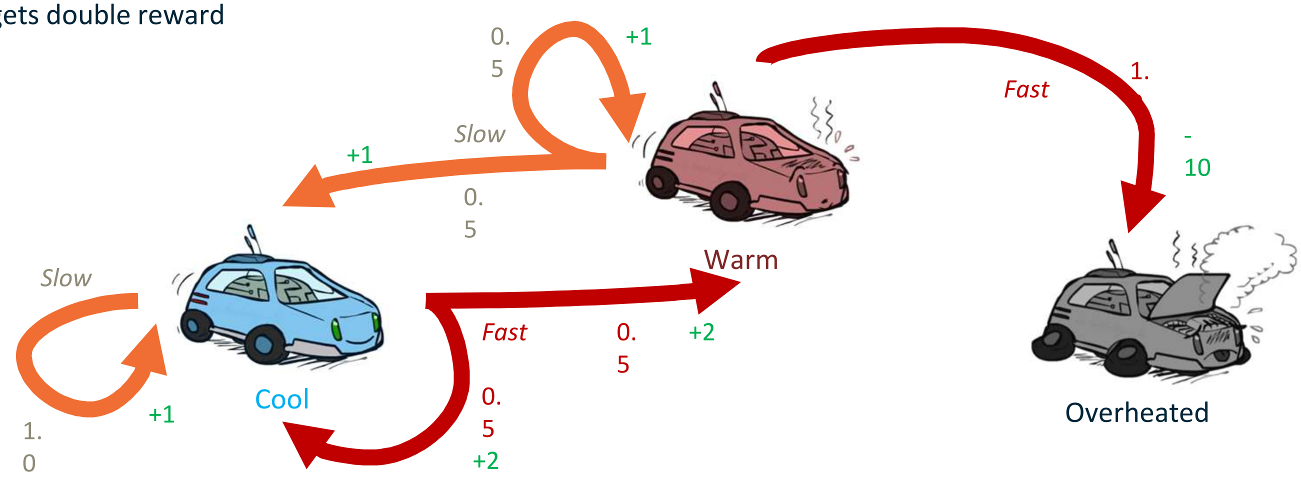
- Initialize values of all states
- While not converged:
  - For each state: $V^{i+1}(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)\left[r(s,a) + \gamma V^i(s')\right]$
- Repeat until convergence (no change in values)

$$V^0 \rightarrow V^1 \rightarrow V^2 \rightarrow \quad \cdots \rightarrow V^i \rightarrow \ldots \quad \rightarrow V^*$$

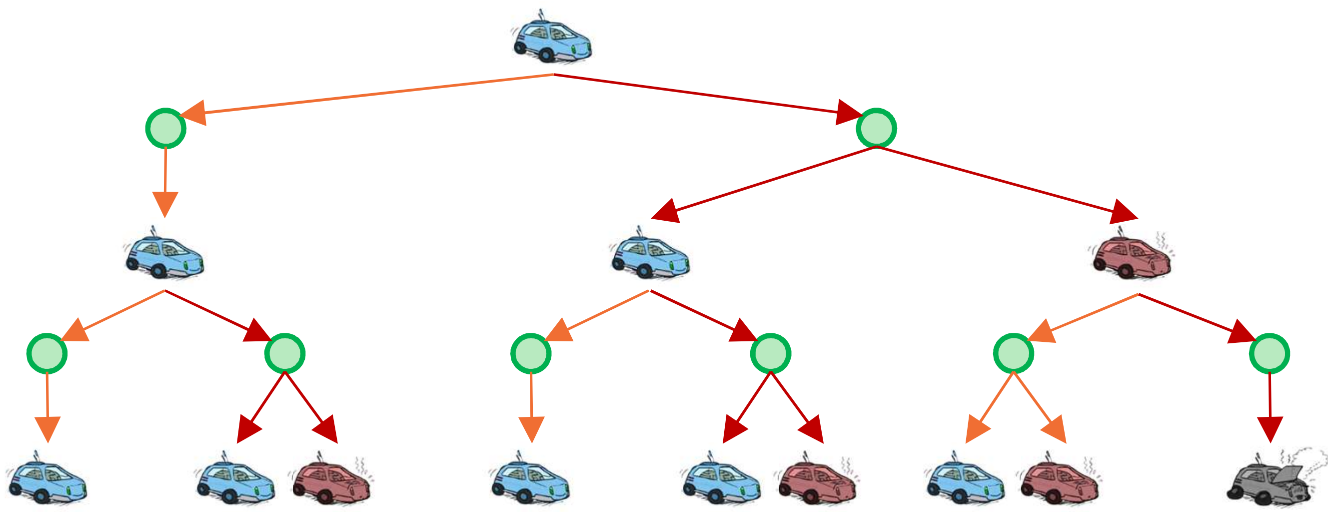
Time complexity per iteration $O(|\mathcal{S}|^2|\mathcal{A}|)$

Example: Racing

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



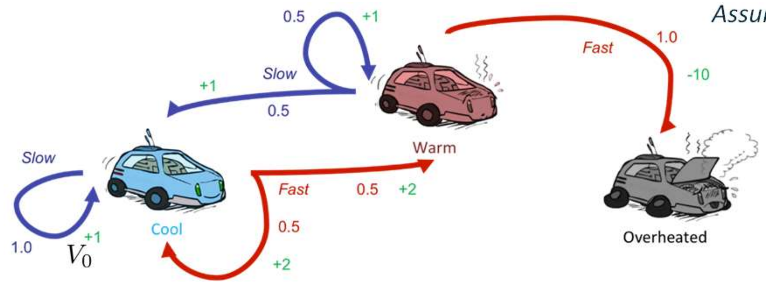
Slide Credit: http://ai.berkeley.edu



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

Racing Search Tree

Georgia Tech



Assume no discount!

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

V_2	3.5	2.5	0
V_1	2	1	0
	0	0	0

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

Racing Search Tree

Georgia Tech

Value Iteration Update:

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

Q-Iteration Update:

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

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

Q-Iteration

Policy iteration: Start with arbitrary π_0 and refine it.

$$\pi_0 \rightarrow \pi_1 \rightarrow \pi_2 \rightarrow \quad \ldots \quad \rightarrow \pi^*$$

Involves repeating two steps:

- **Policy Evaluation:** Compute V^π (similar to Value Iteration)
- **Policy Refinement:** Greedily change actions as per V^π at next states

$$\pi_i(s) \leftarrow \underset{a}{\operatorname{argmax}} \sum_{s'} p\left(s' \mid s, a\right)\left[r(s, a)+\gamma V^{\pi_i}\left(s'\right)\right]$$

$$\pi_0 \longrightarrow V^{\pi_0} \longrightarrow \pi_1 \longrightarrow V^{\pi_1} \longrightarrow \quad \ldots \quad \longrightarrow \pi^* \longrightarrow V^{\pi^*}$$

Why do policy iteration?

- π_i often converges to π^* much sooner than V^{π_i} to V^{π^*}

Policy Iteration

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

Policy Iteration

- Policy evaluation + refinement

Today, we saw

- **MDPs**: Theoretical framework underlying RL, solving MDPs
- **Policy**: How an agents acts at states
- **Value function (Utility)**: How good is a particular state or state-action pair?

Recap & Next Lecture

Today, we saw

- **MDPs**: Theoretical framework underlying RL, solving MDPs
- **Policy**: How an agents acts at states
- **Value function (Utility)**: How good is a particular state or state-action pair?

- **Solving an MDP with known rewards/transition**
  - **Value Iteration:** Bellman update to state value estimates
  - **Q-Value Iteration:** Bellman update to (state, action) value estimates

- **Policy Iteration**
  - Policy evaluation + refinement

Summary: MDP Algorithms

Next Lecture:

- Departure from known rewards and transitions: **Reinforcement Learning**