Machine Learning: Markov Decision Processes

Forest Agostinelli
University of South Carolina

Topics Covered in This Class

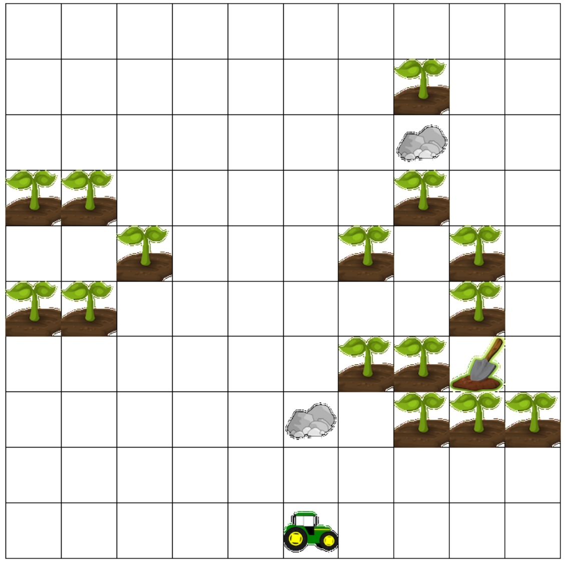
- **Part 1: Search**
  - Pathfinding
    - Uninformed search
    - Informed search
  - Adversarial search
  - Optimization
    - Local search
    - Constraint satisfaction
- **Part 2: Knowledge Representation and Reasoning**
  - Propositional logic
  - First-order logic
  - Prolog
- **Part 3: Knowledge Representation and Reasoning Under Uncertainty**
  - Probability
  - Bayesian networks
- **Part 4: Machine Learning**
  - Supervised learning
    - Inductive logic programming
    - Linear models
    - Deep neural networks
    - PyTorch
  - Reinforcement learning
    - Markov decision processes
    - Dynamic programming
    - Model-free RL
  - Unsupervised learning
    - Clustering
    - Autoencoders

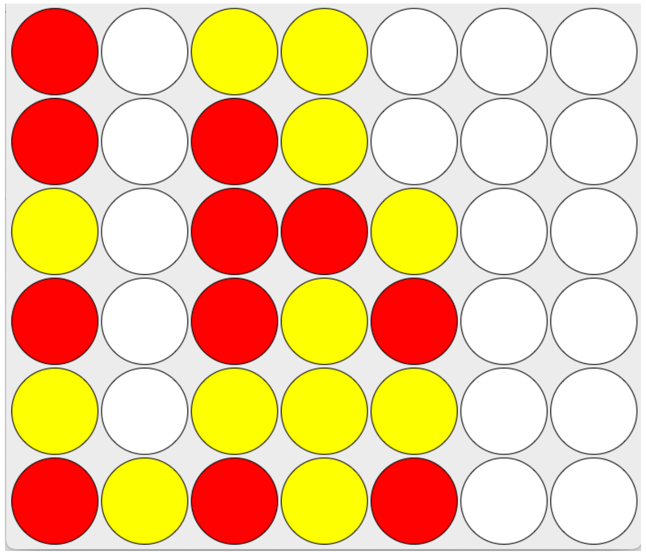
Outline

- Motivation
- Reinforcement learning
- The Markov property
- Markov decision processes
- Value functions
- Textbook: Reinforcement Learning: An Introduction (2nd Edition)
  - Freely available here: http://incompleteideas.net/book/RLbook2020.pdf

Reinforcement Learning

- Learning to maximize reward in sequential decision making problems
- Learning is done through experience
- Decisions affect experience and experience affects decisions
- There is often uncertainty involved with this process
  - What happens if I make this decision?
  - How good is this outcome?

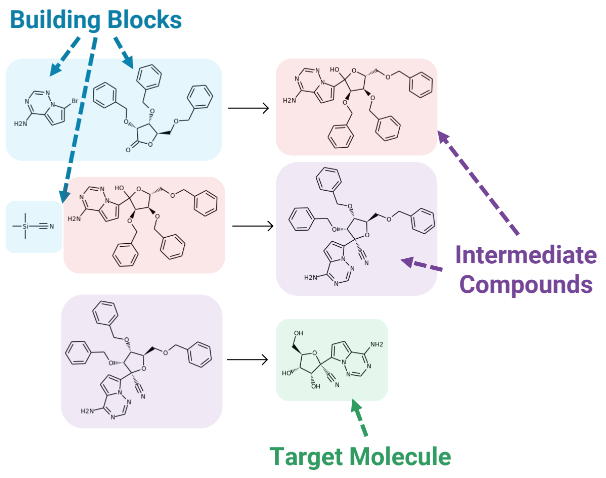


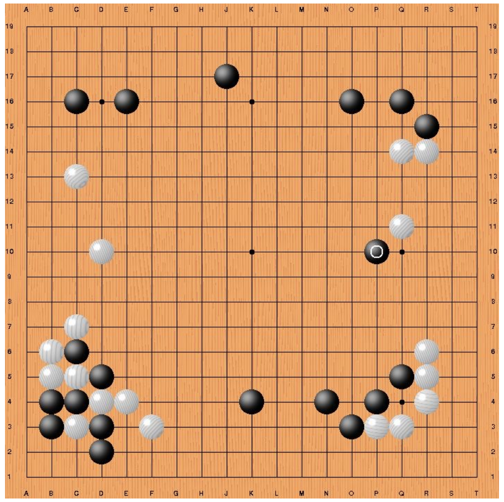


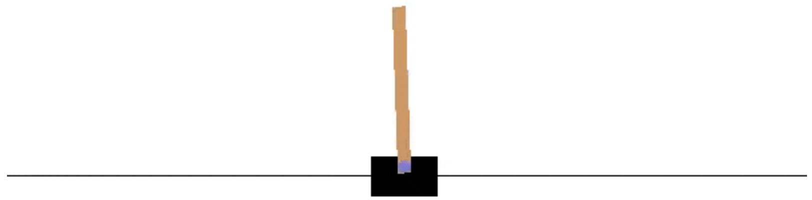
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	

Solving Problems with Search

- A* Search
- Minimax Search
- Informative heuristics may not be easy to construct
- May not be deterministic
- May not have an explicit goal state
  - Agent may continue indefinitely



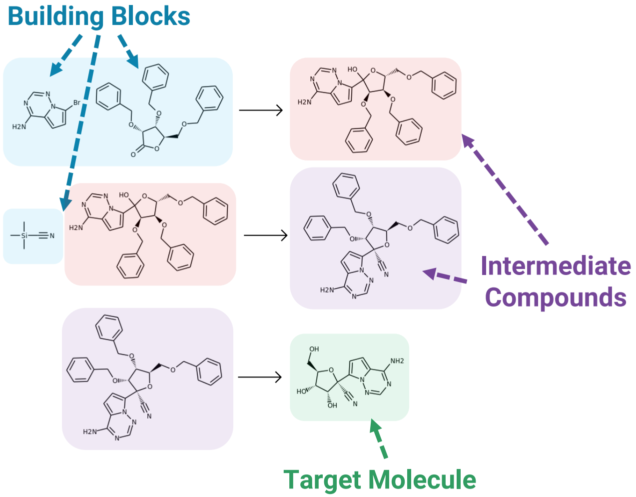


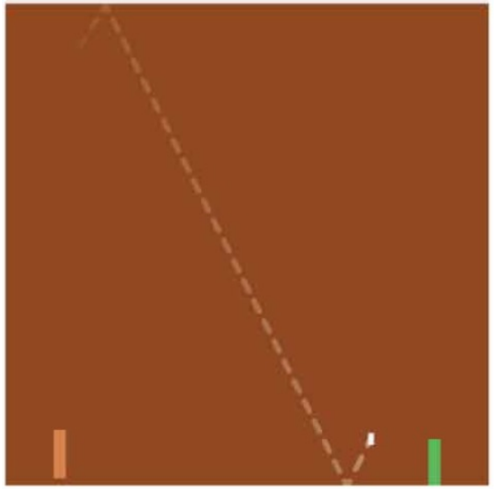


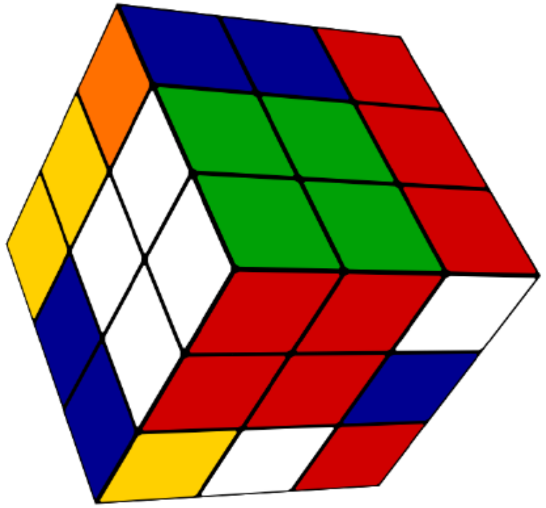
Solving Problems with Supervised Learning

- We can use imitation learning to train an agent to imitate the actions we want it to take
- Use powerful machine learning models, such as deep neural networks, to learn from human example and then generalize to similar problems
- Time consuming
- Due to imperfections in learning, there may be a shift in the distribution of what is seen during training vs in the real world, causing the agent to run into situations it was never prepared for
- Humans do not always know *how* to solve the problem!

Solving Problems with Supervised Learning

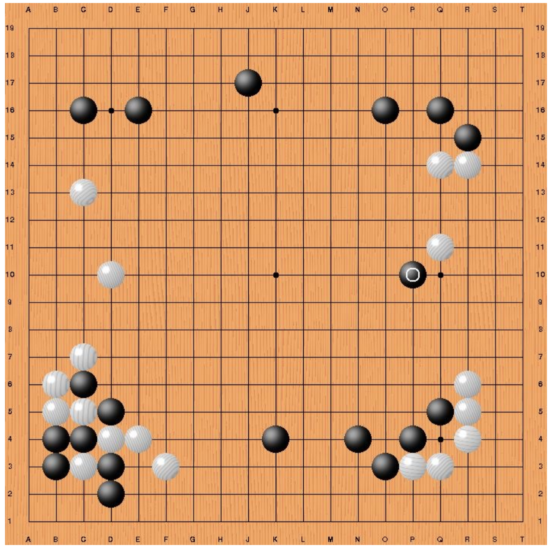






Input: $n \times n$ skew-symmetric matrix $\mathbf{A}$, vector $\mathbf{b}$.
Output: The resulting vector $\mathbf{c} = \mathbf{Ab}$ computed in $\frac{(n-1)(n+2)}{2}$ multiplications.

(1) **for** $i = 1, \ldots, n-2$ **do**
(2) **for** $j = i+1, \ldots, n$ **do**
(3) $w_{ij} = a_{ij}(b_j - b_i)$ ▷ Computing the first $(n-2)(n+1)/2$ intermediate products
(4) **for** $i = 1, \ldots, n$ **do**
(5) $q_i = b_i \sum_{j=1}^{n} a_{ji}$ ▷ Computing the final n intermediate products
(6) **for** $i = 1, \ldots, n-2$ **do**
(7) $c_i = \sum_{j=1}^{i-1} w_{ji} + \sum_{j=i+1}^{n} w_{ij} - q_i$
(8) $c_{n-1} = -\sum_{i=1}^{n-2} \sum_{j=i+1}^{n-2} w_{ij} - \sum_{j=1}^{n-2} w_{jn} + \sum_{i=1, i \neq n-1}^{n} q_i$
(9) $c_n = -\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{ij} + \sum_{i=1}^{n-1} q_i$



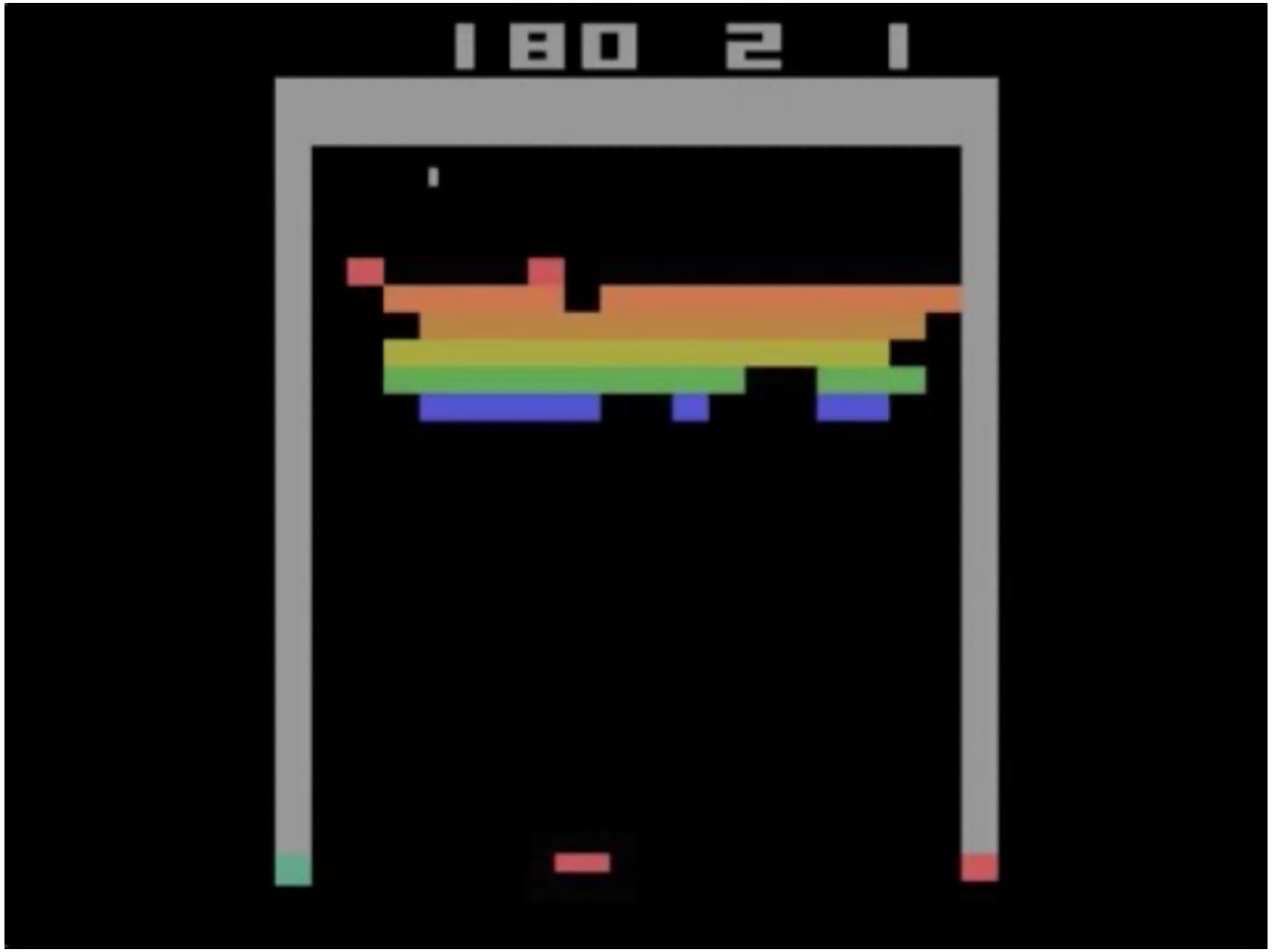
Fawzi, Alhussein, et al. "Discovering faster matrix multiplication algorithms with reinforcement learning." Nature 610.7930 (2022): 47-53.

Reinforcement Learning

- We need machine learning algorithms that learn from their own experience.
- For this, we turn to reinforcement learning
- Reinforcement learning is frequently combined with techniques we have learned in this class such as **search** and **supervised learning**

RL Successes: Atari

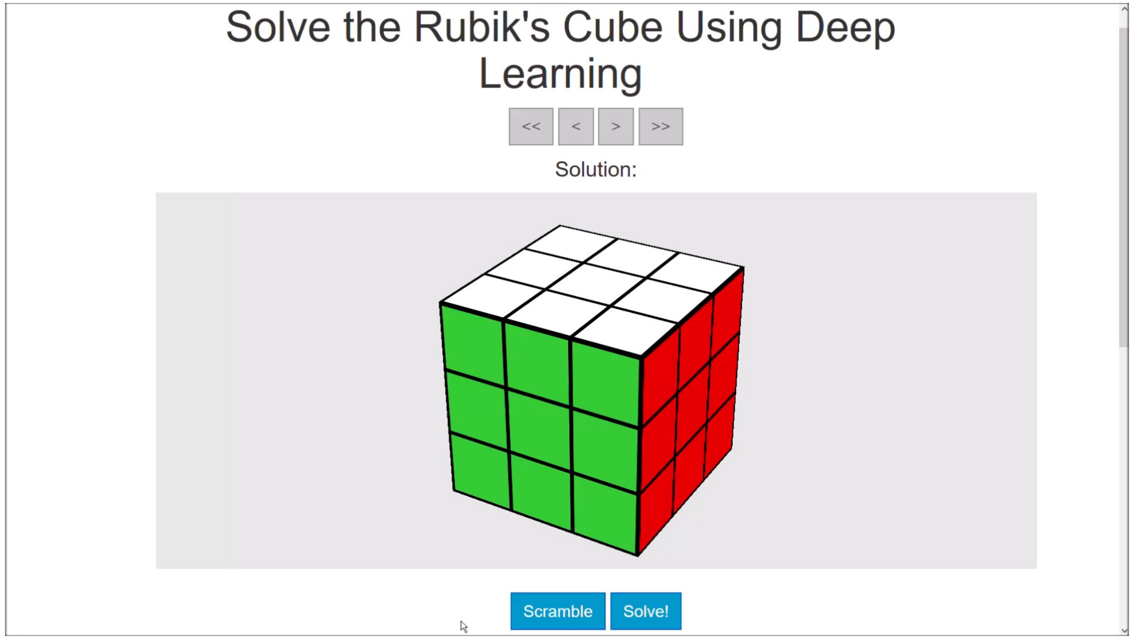
- Using RL algorithms, a deep neural network is trained to play Atari games using raw pixels.
- Current work shows we can train deep neural networks to play better than humans on 57 different Atari games.
- RL was combined with **deep neural networks**

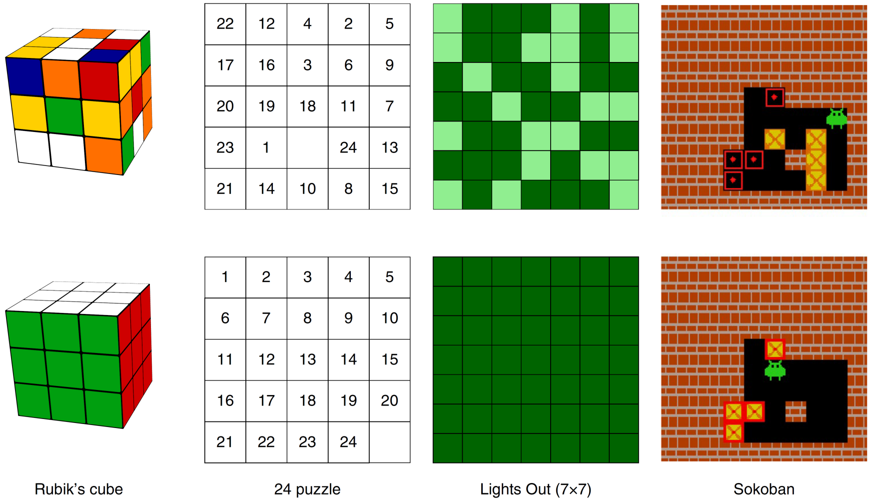


Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." Nature 518.7540 (2015): 529-533.

RL Successes : Solving the Rubik's Cube

- 4.3 x 10^{19} possible combinations
- No domain-specific knowledge
- DeepCubeA solves the Rubik's cube and other puzzles
  - Puzzles have up to 3.0 x 10^{62} possible combinations.
- Finds a shortest path in the majority of verifiable cases
- http://deepcube.igb.uci.edu/
- RL was combined with **deep neural networks** and **A* search**

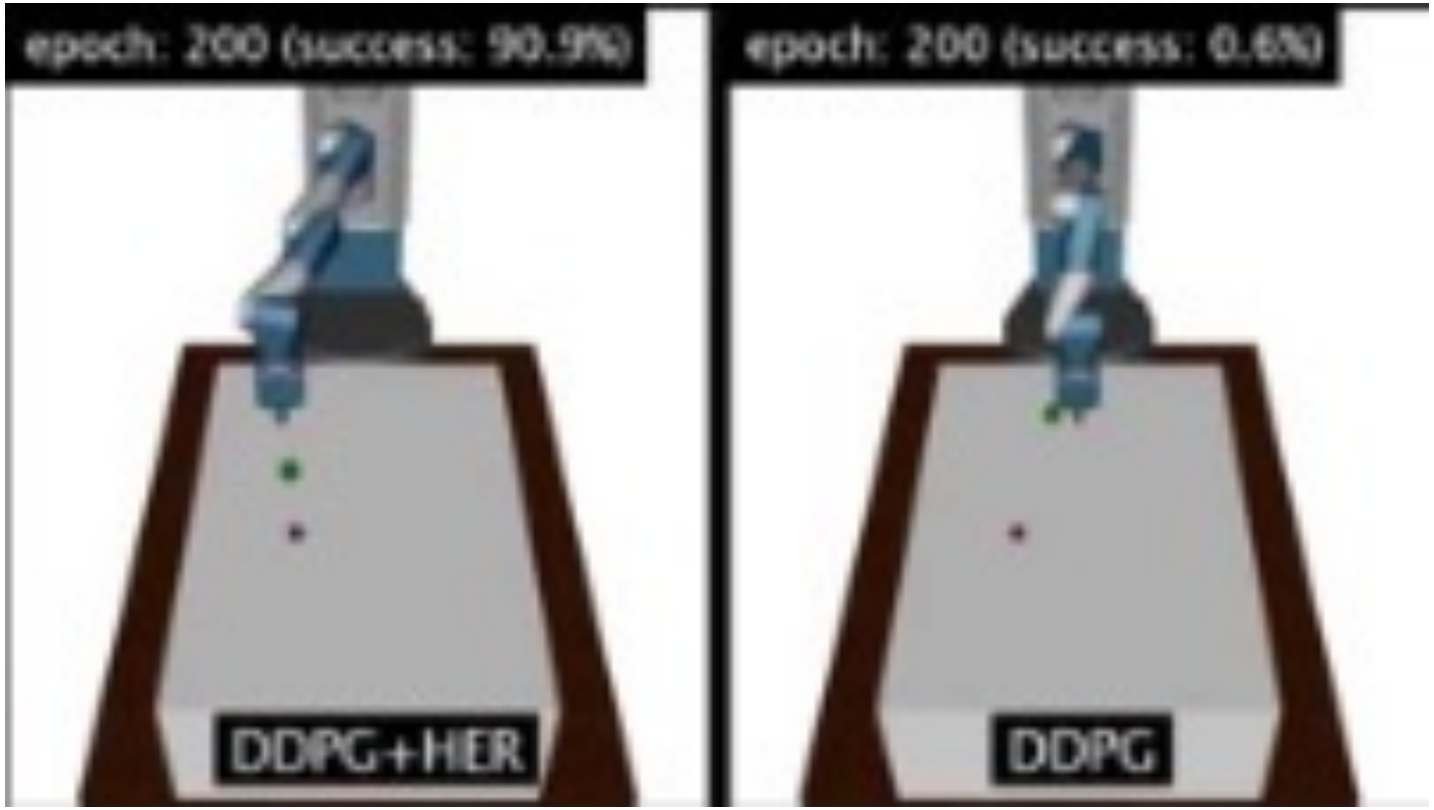




Agostinelli, Forest, et al. "Solving the Rubik's cube with deep reinforcement learning and search." Nature Machine Intelligence 1.8 (2019): 356-363.

RL Successes: Robotics

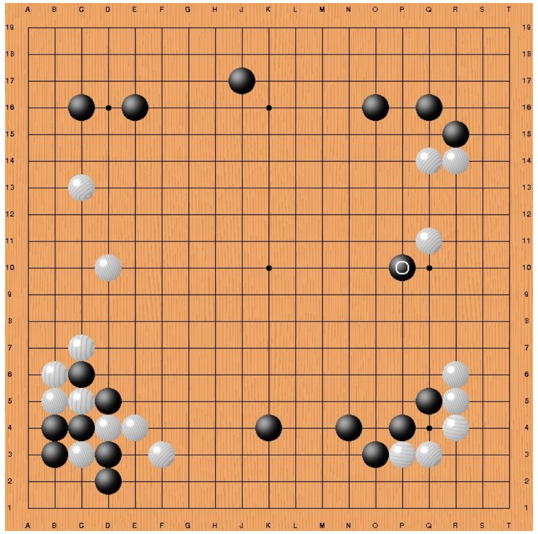
- Can solve problems in continuous environments
- Can train in simulation and transfer to the real world (Sim2Real)
- RL was combined with **deep neural networks**



Andrychowicz, Marcin, et al. "Hindsight experience replay." *Advances in neural information processing systems* 30 (2017).

RL Successes: Go

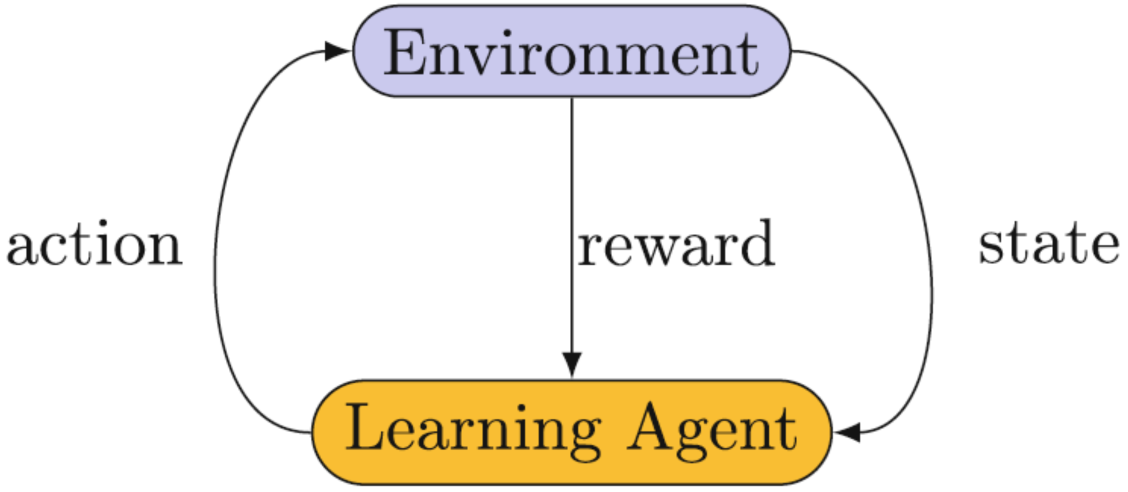
- AlphaGo learned how to play Go from expert demonstrations data and from self-play
  - Defeated one of the best Go players, Lee Sedol, 4 to 1
  - Move 37
- AlphaGoZero builds on AlphaGo and learns only from self-play
- RL was combined with **deep neural networks** and **Monte Carlo tree search**



Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359.

Reinforcement Learning

- **Reinforcement learning**: learning to map states to actions so that we maximize the expected future reward we receive from the environment.
- This mapping of states to actions is called a policy function.
  - Deterministic: $a = \pi(s)$
  - Stochastic: $\pi(a|s) = P(A = a|S = s)$
- At each time step t
  - In state S_t, agent takes action A_t
  - Based on state s_t and action a_t, the environment transitions to state S_{t+1} and outputs reward R_{t+1}



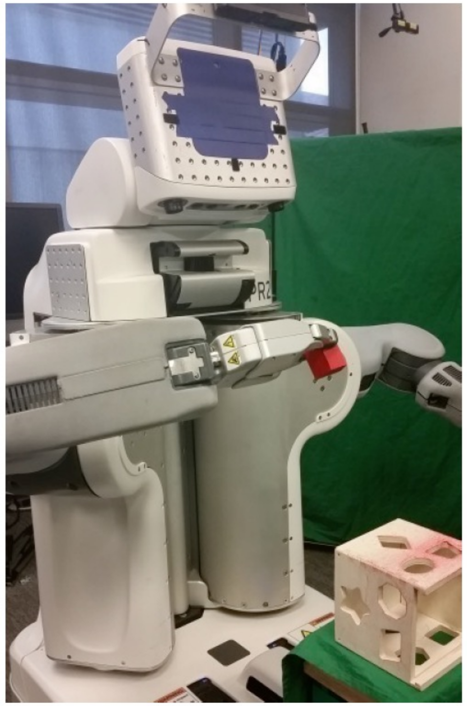
Agostinelli, Forest, et al. "From reinforcement learning to deep reinforcement learning: An overview." Braverman Readings in Machine Learning. Key Ideas From Inception to Current State. Springer, Cham, 2018. 298-328.

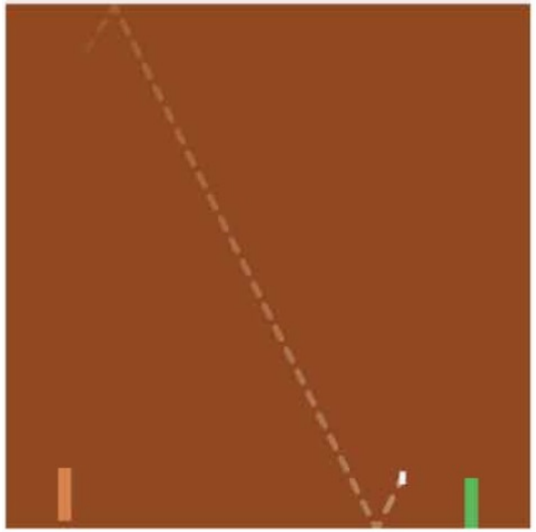
Notation

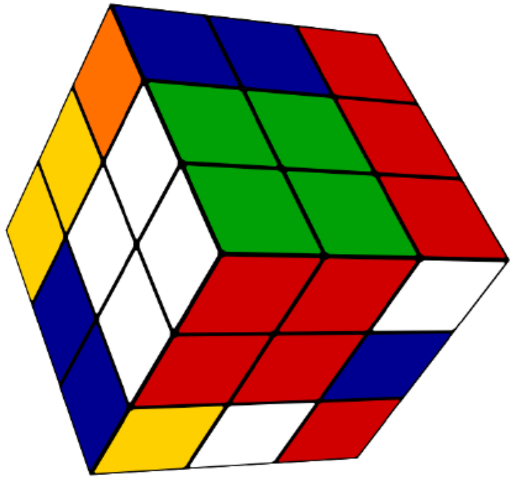
- For notation, capital letters (i.e. S_t) are used for random variables and lowercase letters (i.e. s_t) are used for a particular value of that random variable.
- For states, s typically refers to the current state and s' typically refers to the next state

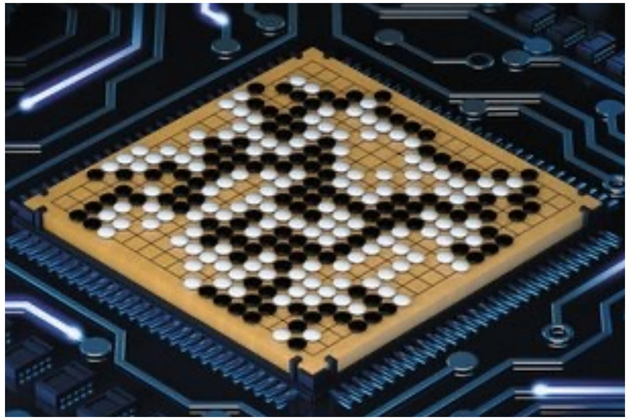
Reinforcement Learning

states, actions, rewards?



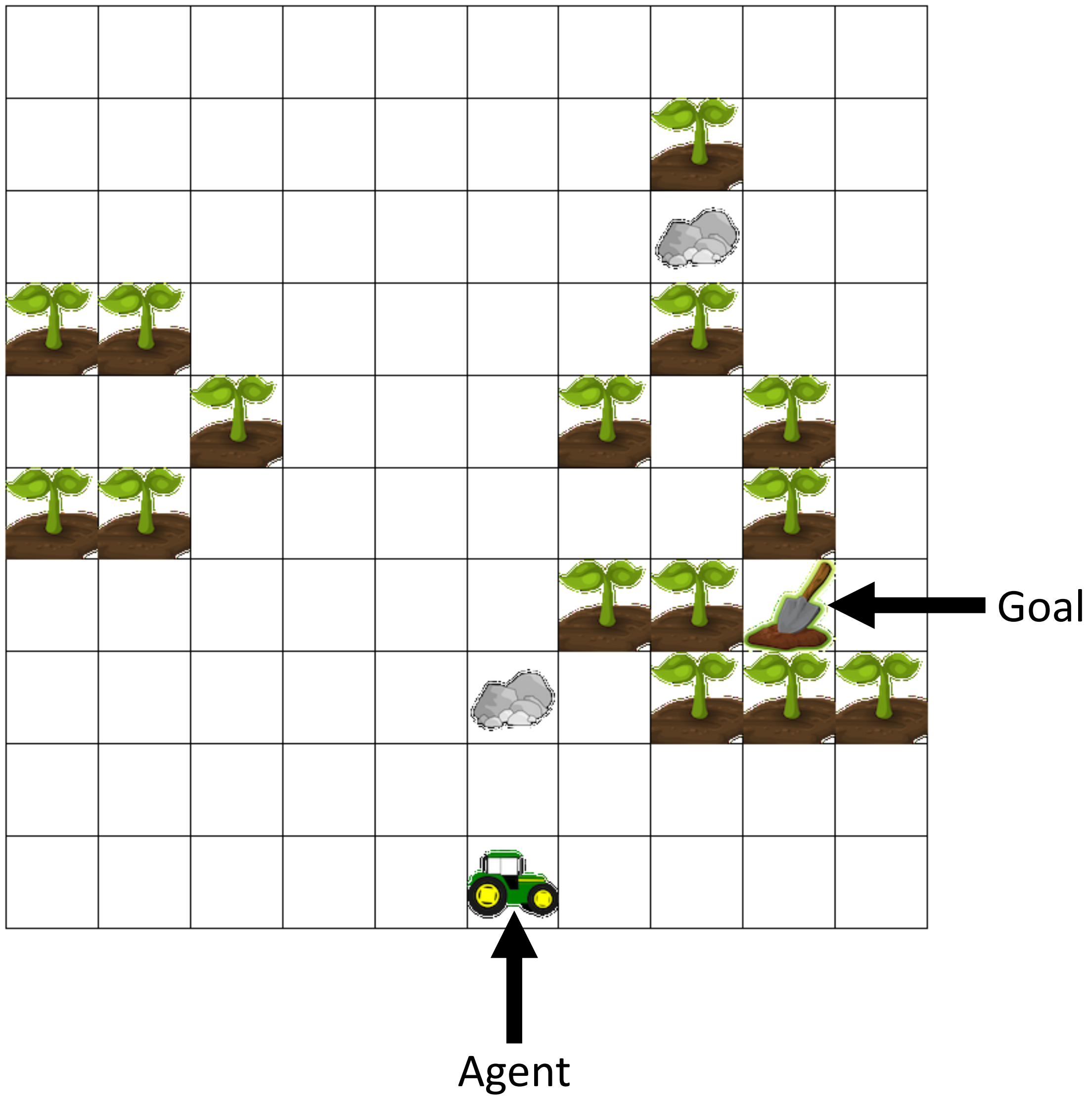






Reinforcement Learning: A Simple Example

- **State**: the configuration of the environment
  - Location of agent, plants, rocks, and goal
  - Or, visual input from a drone
- **Action**: A decision made by the agent that can affect the state of the environment
  - up, down, left, right
- **Reward**: A scalar signal sent from the environment to the agent that is a function of the current state, the action, and the next state
  - Large negative reward for driving on a plant
  - Medium negative reward for driving on a rock
  - Small negative reward for driving on any other space
- What is an optimal policy?
  - A policy that maximizes reward we receive from the environment

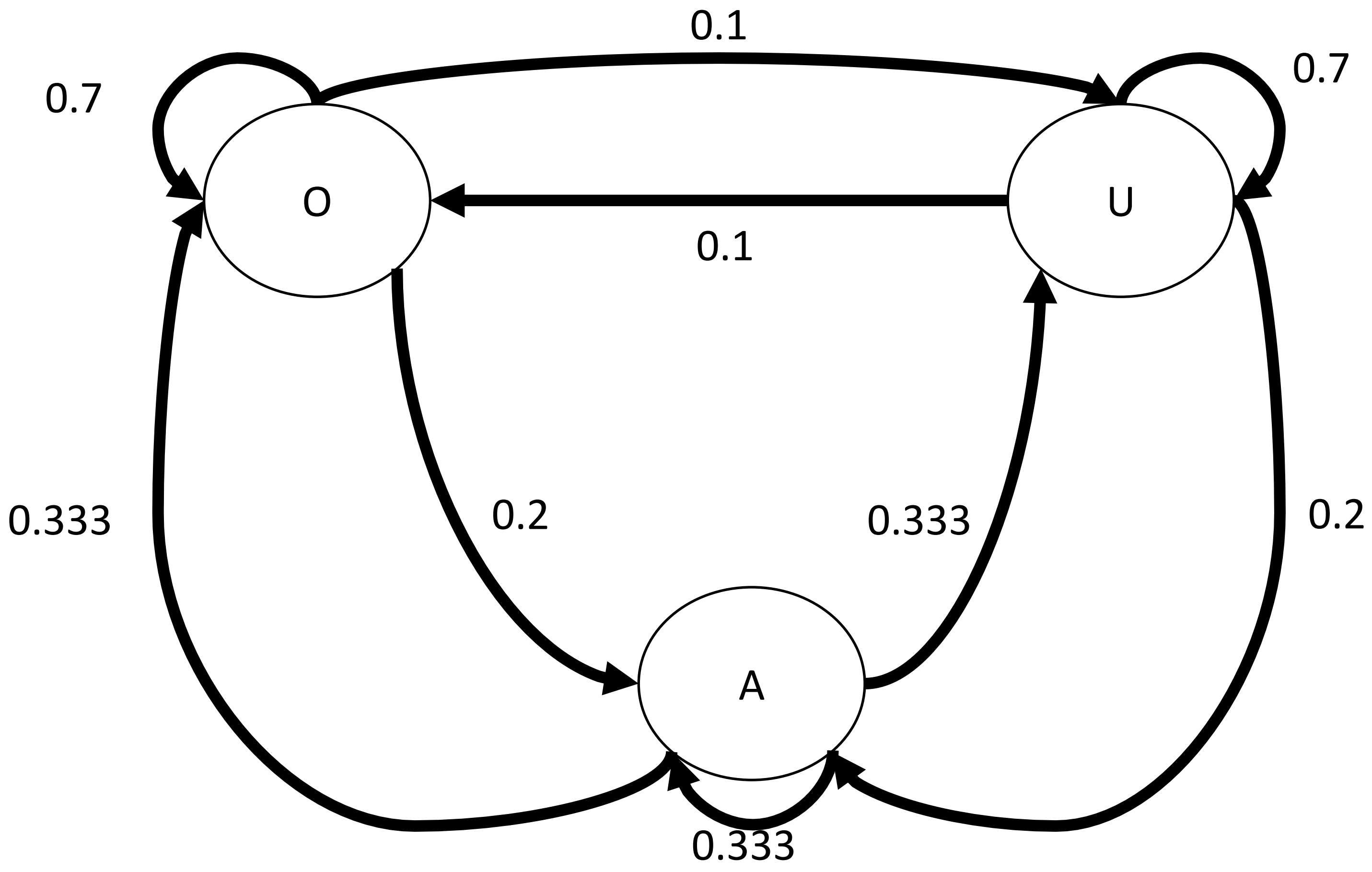


Markov Process

- Models a sequence of random variable S_t, often referred to as the state, whose future value only depends on the current value (memoryless)
- $P(S_{t+1} = s' | S_t = s) = P(S_{t+1} = s' | S_t = s, S_{t-1} = s_{t-1}, \ldots, S_0 = s_0)$
  - Referred to as the dynamics or transition probabilities
- In other words, the future state is independent of the past states given the current state
- This is also referred to as the Markov property

Markov Process: Hospital Example

- The hospital can be in three states
  - Over capacity (O): Insufficient staff and resources
  - Under capacity (U): More than enough staff and resources
  - At capacity (A): About the right number of staff and resources for patients
- With Markov processes, we are often interested in the how the state changes over time
  - I.e. Does a steady state distribution exist, if so, what is it?

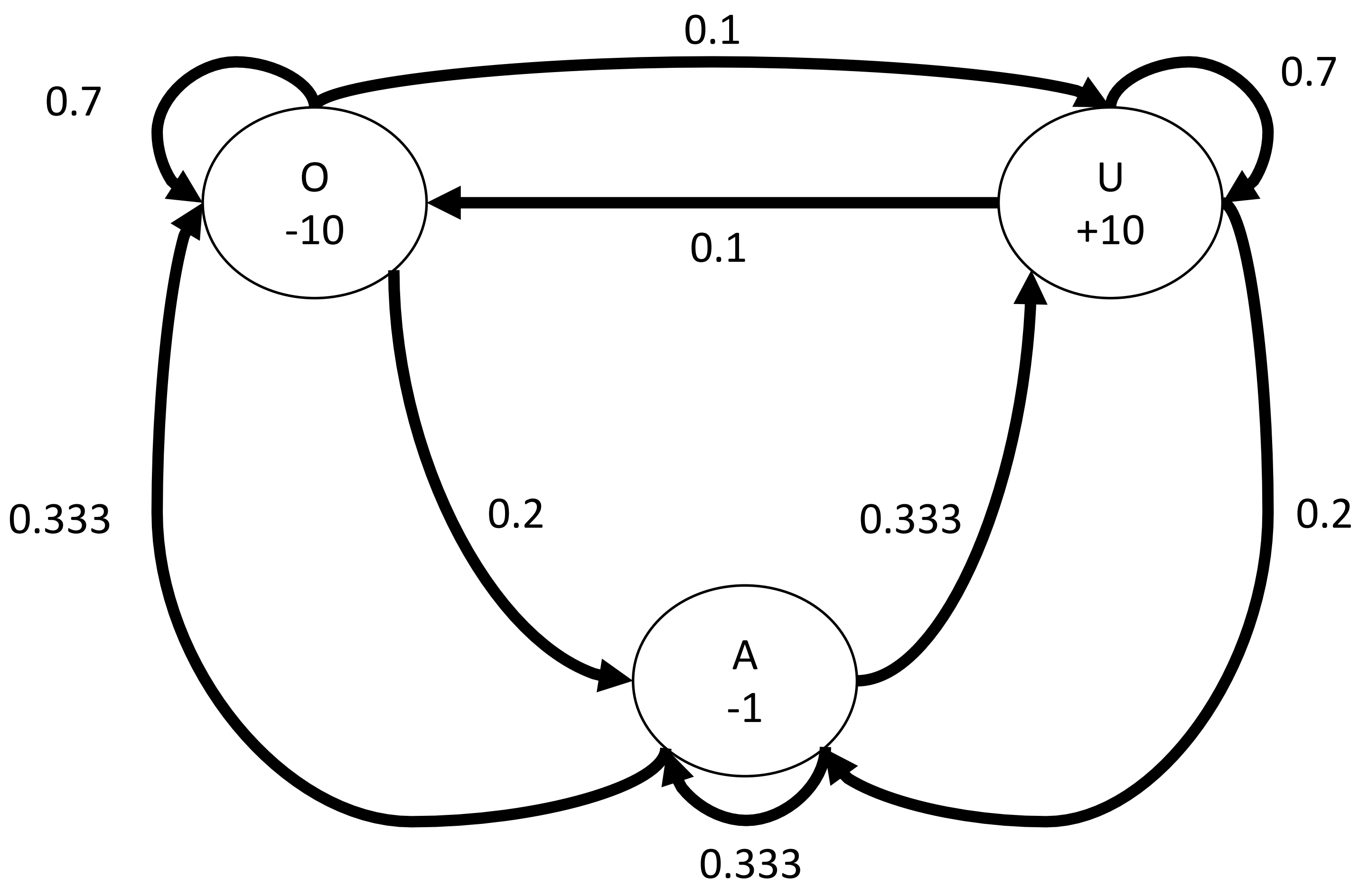


Markov Reward Process (MRP)

- There may be some scalar value we can assign to the transitions between states to indicate how good each transition is
- Dynamics: $P(S_{t+1} = s', R_{t+1} = r | S_t = s\ )$
- The state transition dynamics can be computed as
  - $p(s'|s) = \sum_{r \in \mathcal{R}} p(s', r|s)$
- The reward dynamics of a state can be computed as
  - $r(s) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s) = \mathbb{E}[R_{t+1}|S_t = s]$

MRP: Hospital Example

- Expected reward $r(s)$ is shown below the state name



Markov Decision Processes (MDPs)

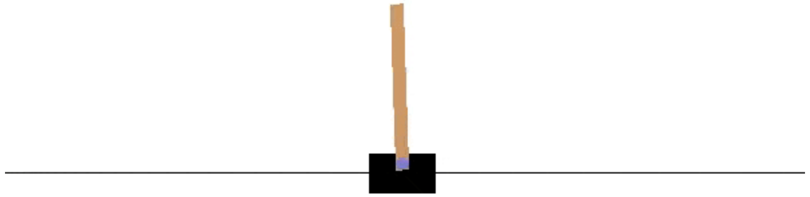
- **States**
- **Actions**
- **Transition Probabilities:** $P(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$
  - Defines the dynamics of the MDP
- The state-transition probabilities can be obtained from the transition probabilities
  - $p(s'|s, a) = \sum_{r \in \mathcal{R}} p(s', r|s, a)$
- The expected reward can be obtained from the transition probabilities
  - $r(s, a\ ) = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$
- For now, we assume the state and actions are discrete and finite, however, this restriction can be relaxed to be continuous and infinite

The Markov Property for MDPs

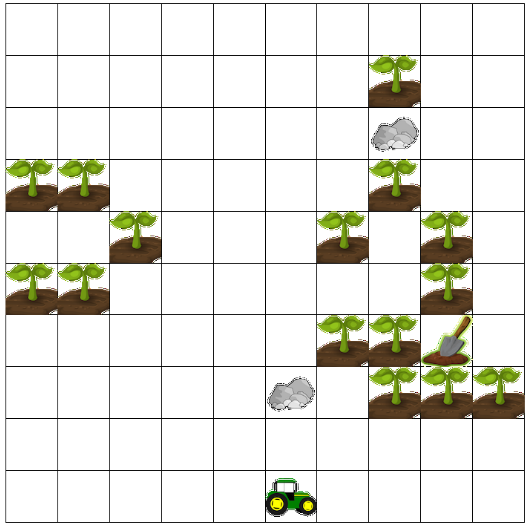
- Our policy at timepoint t is only based on the current state s
  - $\pi(a|s) = P(A_t = a|S_t = s)$
- However, the agent has a history up until S_t
  - $\mathrm{H_t} = S_0, A_0, R_1 S_1, A_1, R_2 \ldots S_{t-1}, A_{t-1}, R_t, S_t$
- We assume that all relevant information about the future is contained in the current state and action
  - $P(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a) = P(S_{t+1} = s', R_{t+1} = r|H_t = h_{t+1}, A_t = a)$
  - The joint distribution of the next state and reward is conditionally independent of the history given the current state and action

Quick Quiz: State Representations

- **Cartpole**
- Actions: apply force left, apply force right
- Rewards: +1 for every step
- Episode ends when the pole falls over



- **AI Farm - Harvesting**
- Actions: Same as before
- Rewards: Same as before and +1 for every crop harvested. Can only harvest crop once.
- Episode ends when all crops are harvested



- What should the state representation be?
- Remember, the Markov property must hold: $P(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$

MDPs: Episodes and Returns

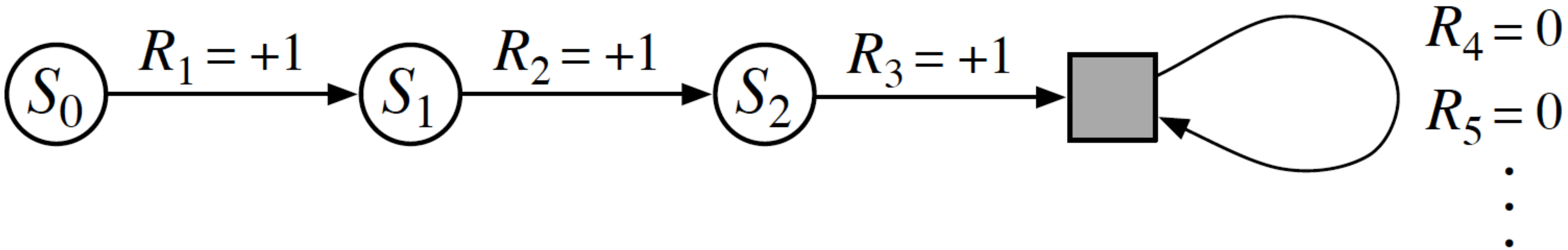
- **Episode:** Starts at some start state at timepoint 0 and ends at a special state, called the terminal state, at timepoint T
- **Return:** the sum of rewards after timestep t
  - $G_t = R_{t+1} + R_{t+2} + R_{t+3} \ldots + R_T$
  - We seek to maximize the expected return

MDPs: Continuing Tasks

- There are environments in which an agent's experience is not guaranteed to terminate
- Maximizing $G_t = R_{t+1} + R_{t+2} + R_{t+3} \ldots + R_T$ becomes problematic
  - $T = \infty$
  - Therefore, it is possible that G_t could be ∞.
- Therefore, we discount the reward
  - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \ldots$
  - $0 \leq \gamma < 1$
    - Will converge to a finite number as long as rewards are finite
    - Geometric series: $\sum_{k=0}^{\infty} ar^k = \frac{a}{1-r}$ for $|r| < 1$
  - γ can be set to 1 if T is finite
    - In finite cases, sometimes it is still better to make it less than 1 to reduce variance when doing function approximation

MDPs: Unifying Continuing and Episodic Tasks

- Episodic tasks can be posed as continuing tasks with a terminal state that:
  - Only transitions to itself
  - Only generates rewards of 0
- The return
  - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \ldots$
  - $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$
  - Works both when T is finite and infinite
  - If T is not infinite, then γ can be 1



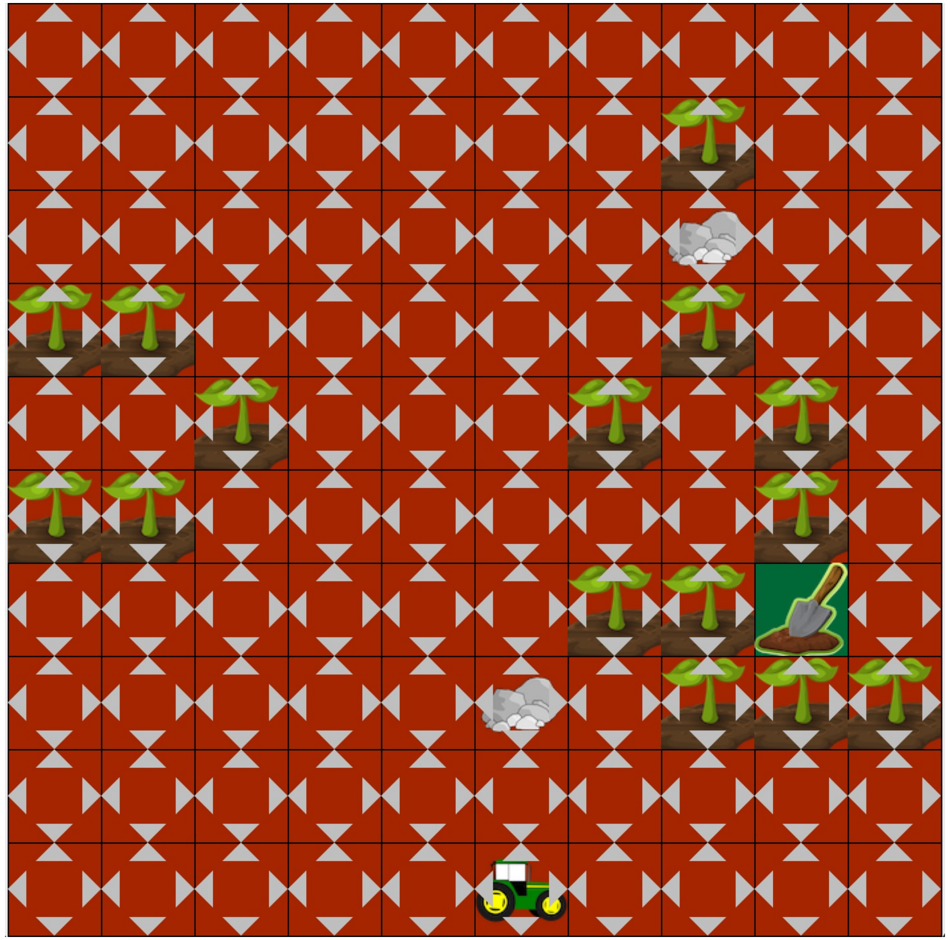
Value Functions

- State-value function
  - The expected return when in state s and following policy π
  - $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \, | S_t = s]$
- Action-value function
  - The expected return when taking action a in state s and then following policy π
  - $q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \, | S_t = s, A_t = a]$

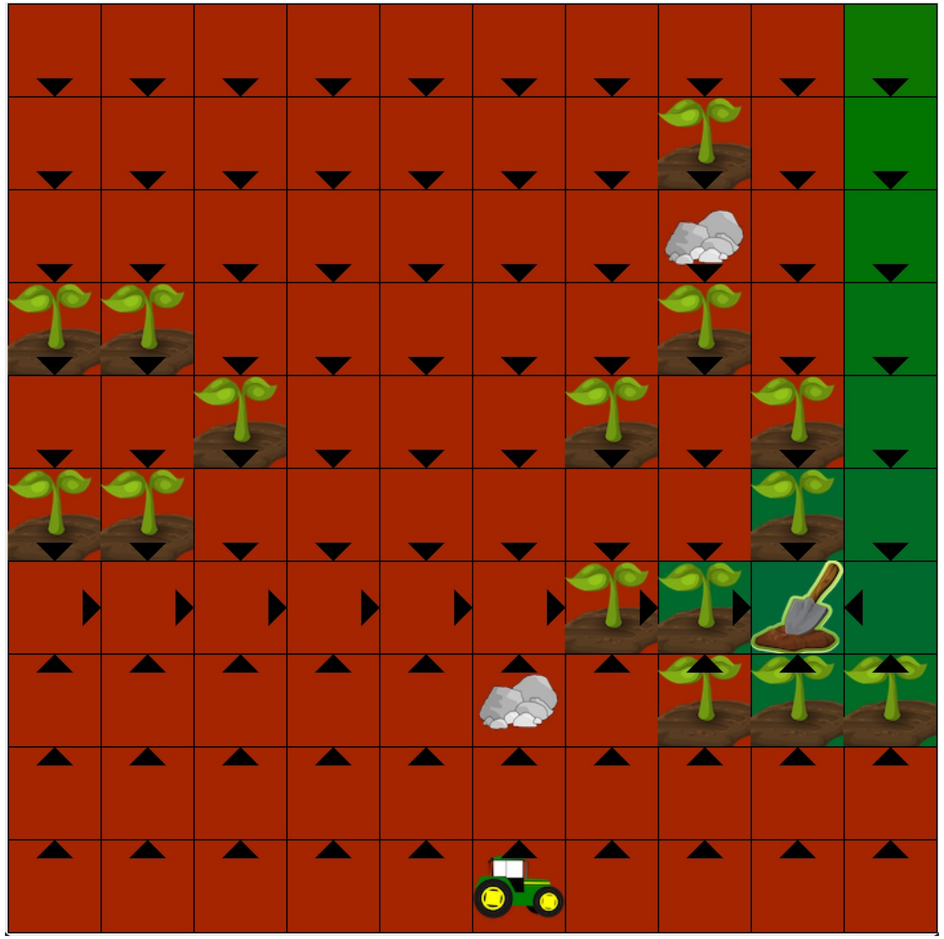
Optimal Policy and Value Function

- Which policy is better?
  - $\pi \geq \pi'$ if and only if $v_\pi(s) \geq v_{\pi'}(s)$ for all $s \in S$
- A policy that achieves the greatest possible return from any state is an optimal policy
  - $\pi_* \geq \pi'$ for all π'
- The optimal value function is the value function obtained when following the optimal policy
  - $v_*(s) = \max_\pi v_\pi(s)$
  - $q_*(s, a) = \max_\pi q_\pi(s, a)$
- Optimal policies can be obtained by behaving greedily with respect to the optimal value function
- Many RL methods first learn a value function and then induce a policy by behaving greedily with respect to the value function
- Is the optimal policy unique?
- Is the optimal value function unique?

Value Functions

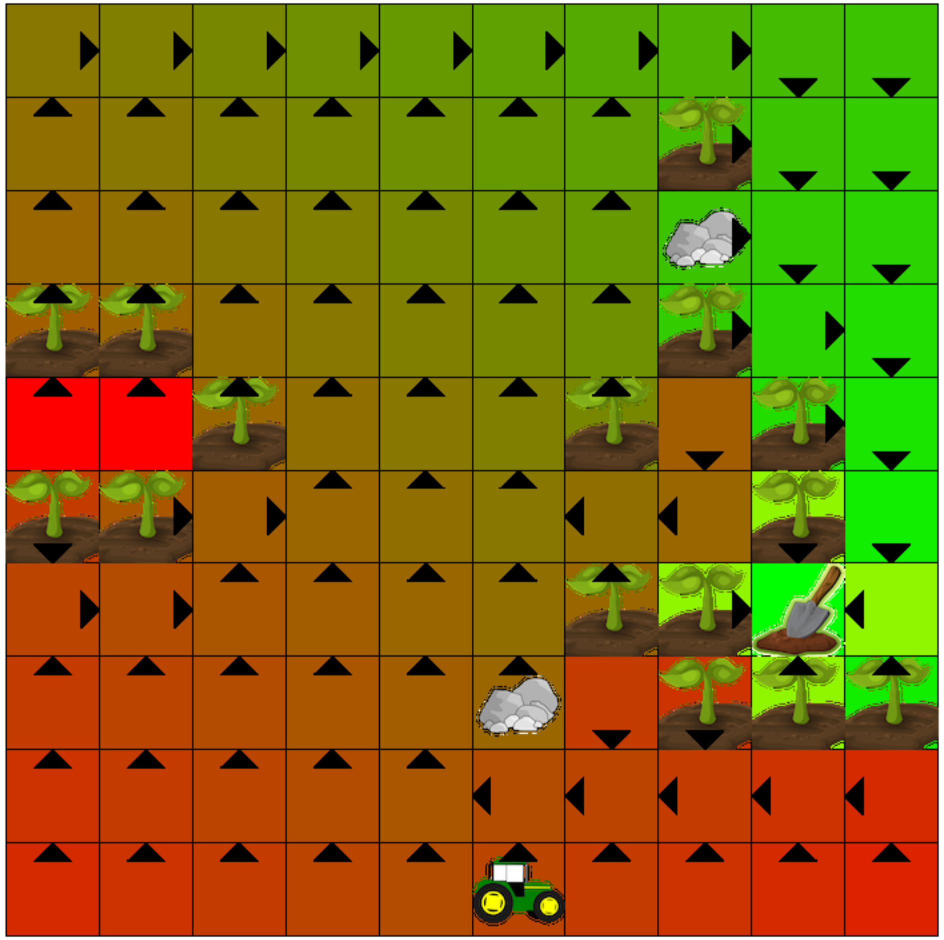


$v_\pi(s)$
π is uniform random for all states



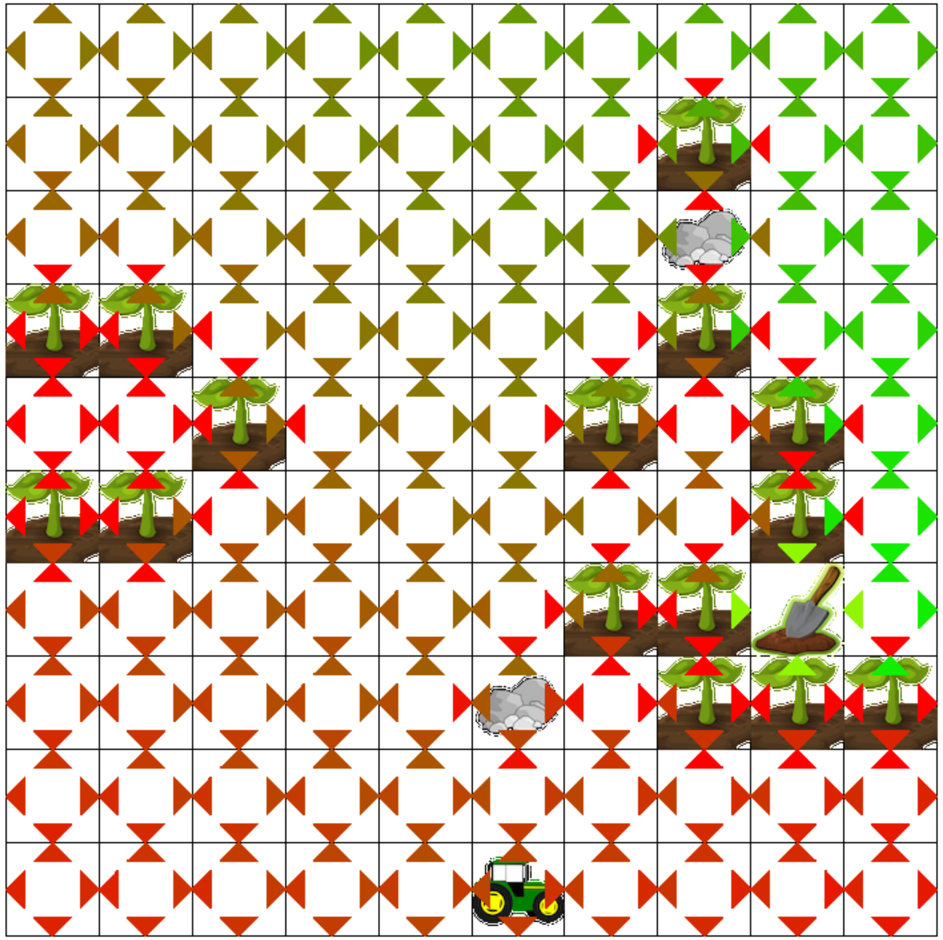
$v_\pi(s)$
π says to go up if below goal, go down if above goal, otherwise, go in the direction of the goal

Optimal Value Function



$$v_*(s)$$

$$\pi_* = \operatorname*{argmax}_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_*(s'))$$



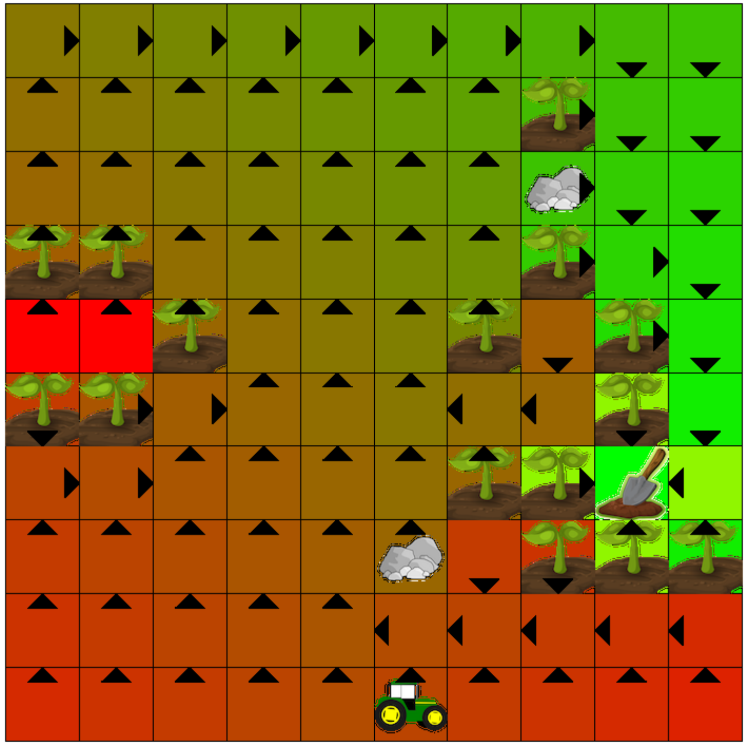
$$q_*(s,a)$$

$$\pi_* = \operatorname*{argmax}_a q_*(s,a)$$

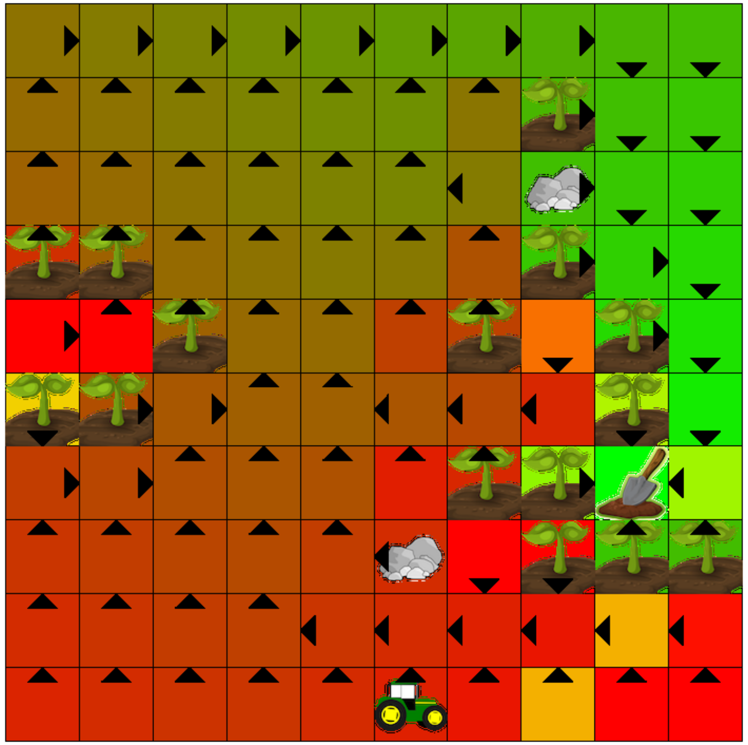
Optimal Value Function: Strong Winds

- Wind blows you right with some probability p

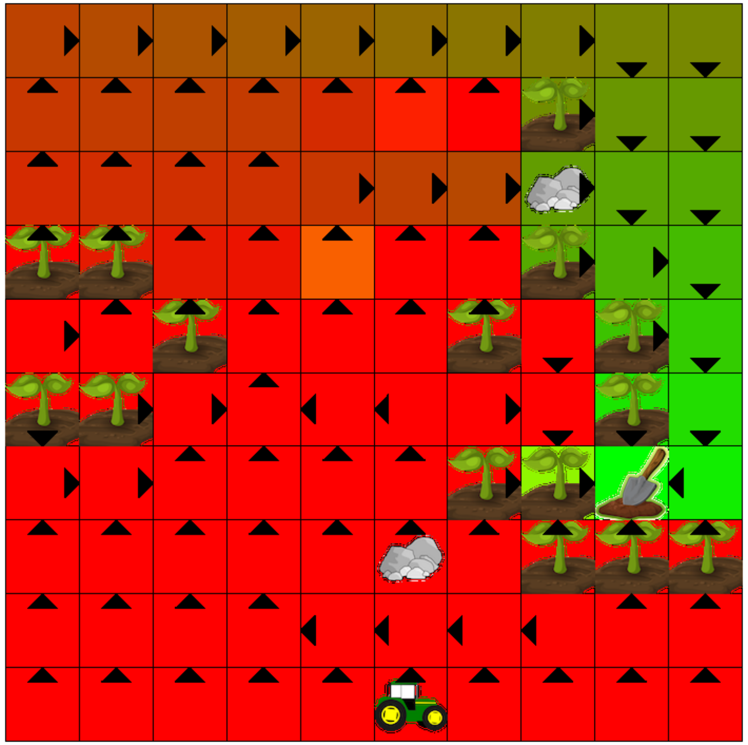
$p = 0$

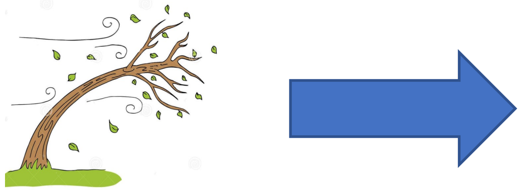


$p = 0.1$



$p = 0.5$

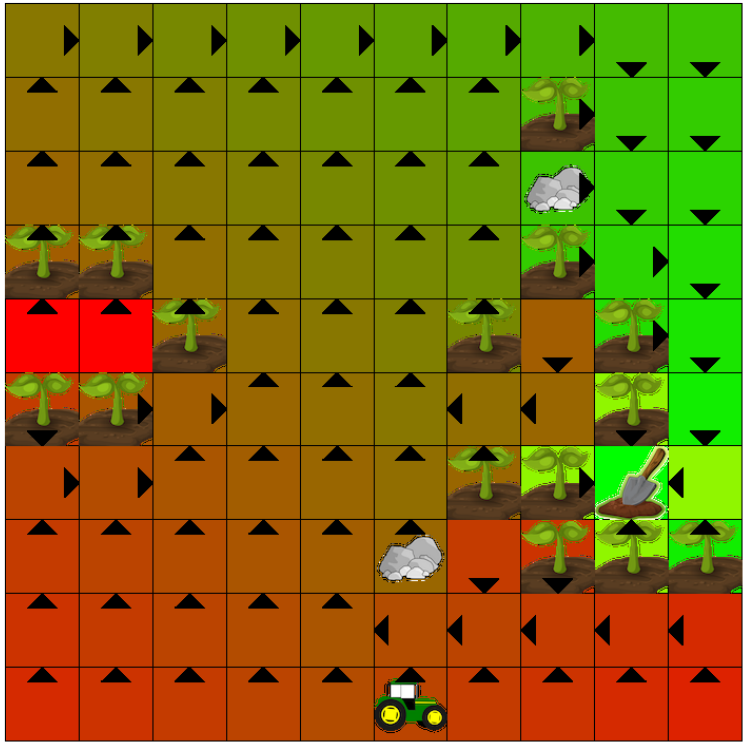




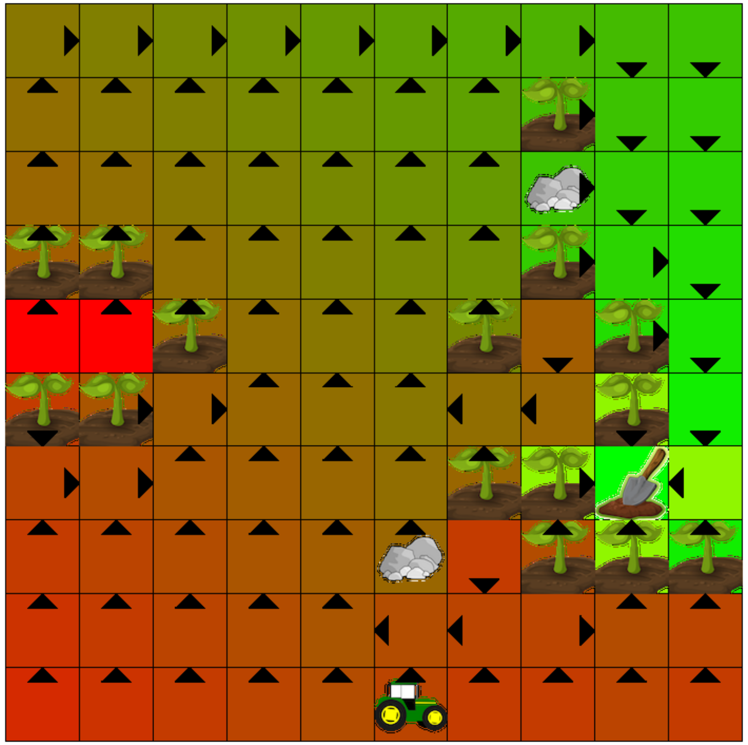
Optimal Value Function: Effect of Rewards

- Reward r of driving on a plant

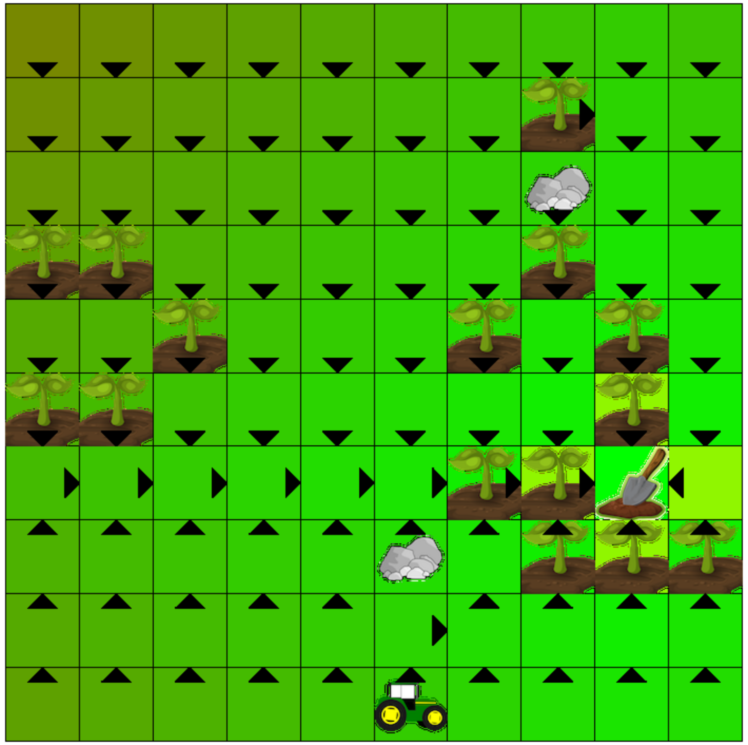
$r = -50$



$r = -20$



$r = -2$

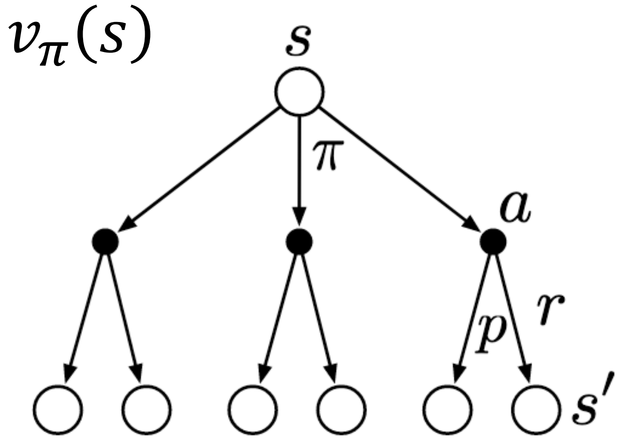


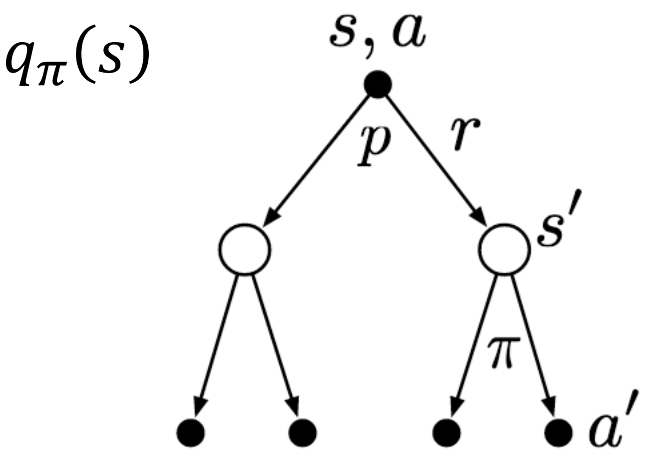
Bellman Equation

- $v_\pi(s) = \mathbb{E}_{a\sim\pi(a|s)}[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty}\gamma^k R_{t+1+k}\,|S_t = s]$
- $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] = \mathbb{E}_\pi[R_{t+1}|S_t = s] + \gamma\mathbb{E}_\pi[G_{t+1}|S_t = s]$
- $\mathbb{E}_\pi[R_{t+1}|S_t = s] = \sum_r p(r|s)r = \sum_a\sum_r p(r,a|s)r = \sum_a\sum_r p(r|s,a)\pi(a|s)r$
- $\mathbb{E}_\pi[R_{t+1}|S_t = s] = \sum_a \pi(a|s)\sum_r p(r|s,a)r = \sum_a \pi(a|s)r(s,a)$
- $\mathbb{E}_\pi[G_{t+1}|S_t = s] = \sum_a \pi(a|s)\sum_{s'} p(s'|s,a)\mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']$
- $\mathbb{E}_\pi[G_{t+1}|S_{t+1} = s'] = v_\pi(s')$
- $v_\pi(s) = \sum_a \pi(a|s)r(s,a) + \gamma\sum_a \pi(a|s)\sum_{s'} p(s'|s,a)v_\pi(s')$
- $v_\pi(s) = \sum_a \pi(a|s)(r(s,a) + \gamma\sum_{s'} p(s'|s,a)\,v_\pi(s'))$
  - $p(s'|s,a) = \sum_{r\in\mathcal{R}} p(s',r|s,a)$
  - $r(s,a\,) = \sum_{r\in\mathcal{R}} r\sum_{s'\in\mathcal{S}} p(s',r|s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$
- From the definition of value, we can write the value function in terms of itself
  - This will be the foundation of reinforcement learning

Bellman Equation

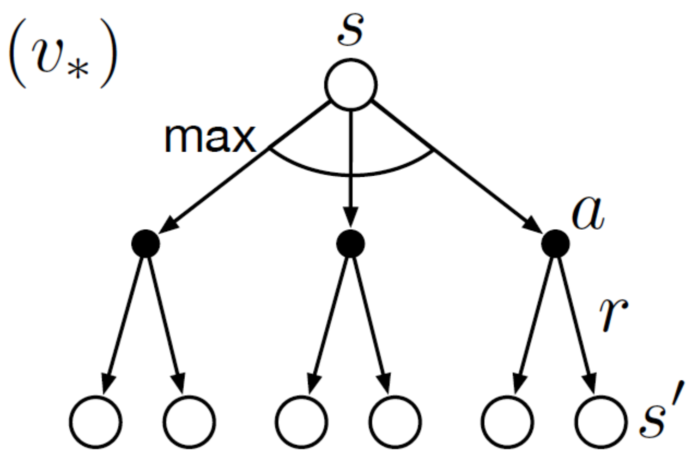
- $v_\pi(s) = \sum_a \pi(a|s)(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s'))$
- $q_\pi(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \sum_a \pi(a|s') q_\pi(s',a)$
- Optimal substructure: Can construct an optimal solution from optimal solutions of subproblems

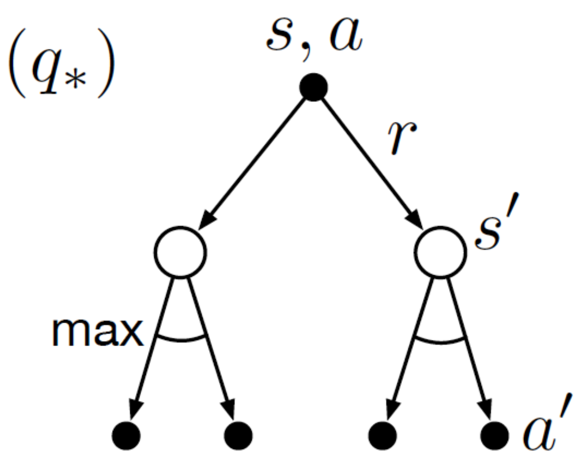




Bellman Optimality Equation

- $v_*(s) = \max_a \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_*(s') \right)$
- $q_*(s,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) \max_{a'} q_*(s',a')$





Summary

- Reinforcement learning studies how we can maximize reward in sequential decision making problems
- Given a sequential decision making problem, we first must characterize our problem as a Markov decision process (MDPs)
  - The joint probability of the next state and reward is independent of the history given the current state and action
- The problem of computing the value of a given policy has optimal substructure (Bellman equation)
  - $v_\pi(s) = \sum_a \pi(a|s)(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s'))$
- In the case of computing the value of an optimal policy (Bellman optimality equation)
  - $v_*(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_*(s'))$