Machine Learning: Dynamic Programming

Forest Agostinelli
University of South Carolina

Topics Covered in This Class

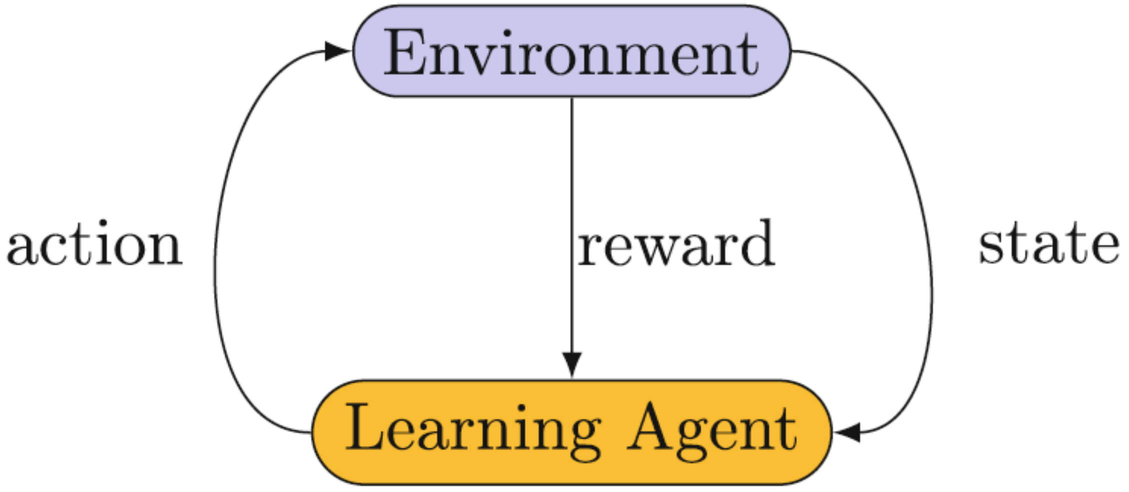
- **Part 1: Search**
  - Pathfinding
    - Uninformed search
    - Informed search
  - Adversarial search
  - Optimization
    - Local search
    - Constraint satisfaction
- **Part 2: Knowledge Representation and Reasoning**
  - Propositional logic
  - First-order logic
  - Prolog
- **Part 3: Knowledge 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

- Background
- Policy Evaluation
- Policy Improvement
- Policy Iteration
  - Modified Policy Iteration
  - Value Iteration
- Approximate value iteration

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 $\mathrm{S_t}$, agent takes action $\mathrm{A_t}$
  - Based on state $\mathrm{s_t}$ and action $\mathrm{a_t}$, the environment transitions to state $\mathrm{S_{t+1}}$ and outputs reward $\mathrm{R_{t+1}}$



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

MDPs: Returns and Value

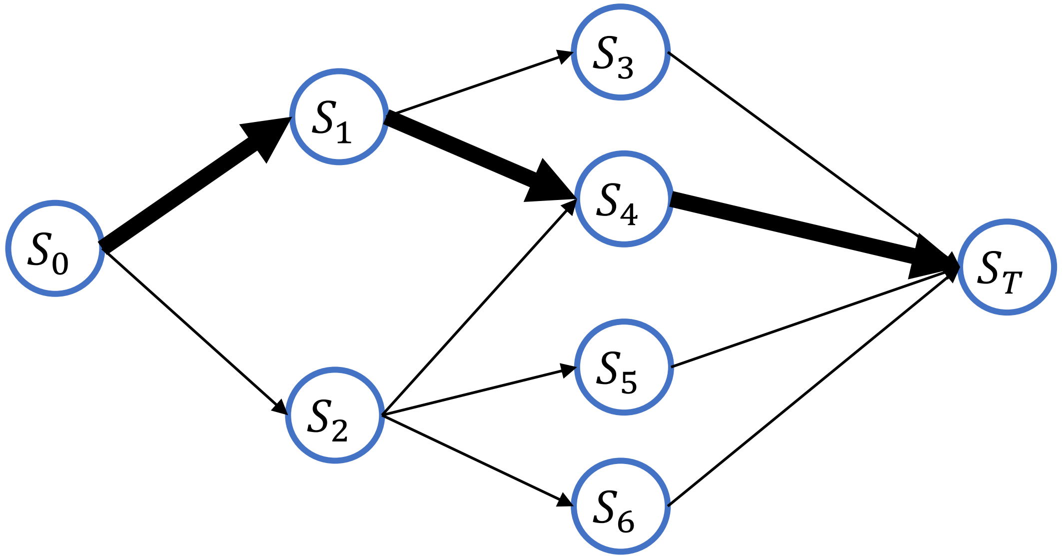
- **Return:** the sum of rewards after timestep t
  - $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \dots$
  - We seek to maximize the expected return
- **State-value function**
  - $v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \,|S_t = s]$
  - $v_*(s) = \max_\pi v_\pi(s)$
- **Action-value function**
  - $q_\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+1+k} \,|S_t = s, A_t = a]$
  - $q_*(s,a) = \max_\pi q_\pi(s,a)$
- Value functions are specific to a given policy π

Dynamic Programming

- Solves problems by recursively breaking them down into simpler subproblems
- Requires
  - Optimal substructure: Can construct an optimal solution from optimal solutions of subproblems

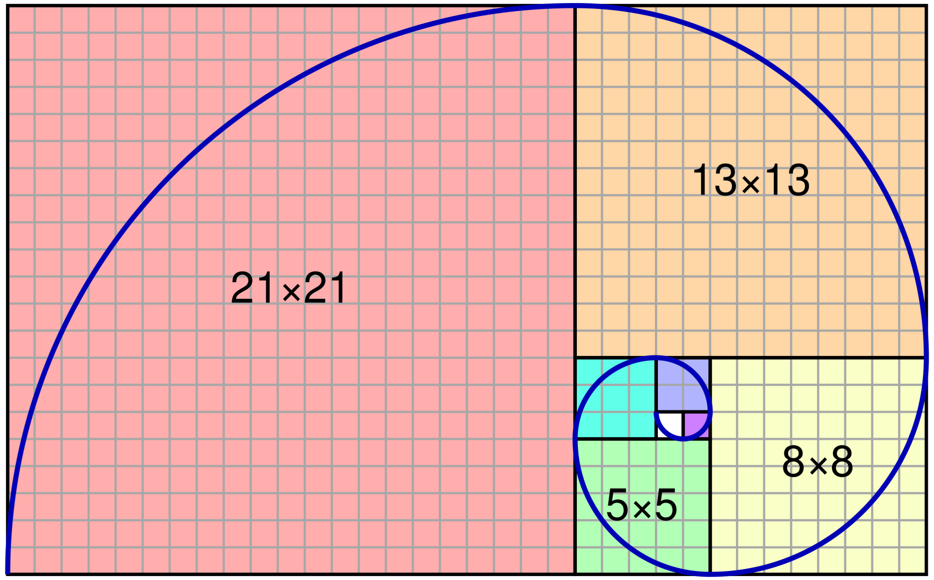
$$v_*(s) = \max_a \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_*(s') \right)$$

    - Principle of optimality
  - Overlapping subproblems: Solutions to subproblems are re-used
    - Value functions

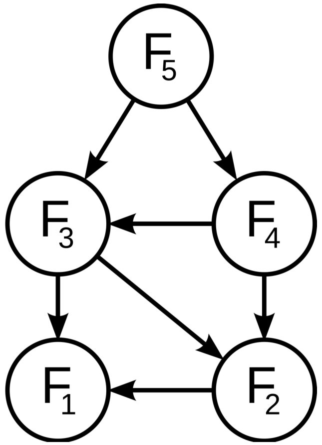


Dynamic Programming

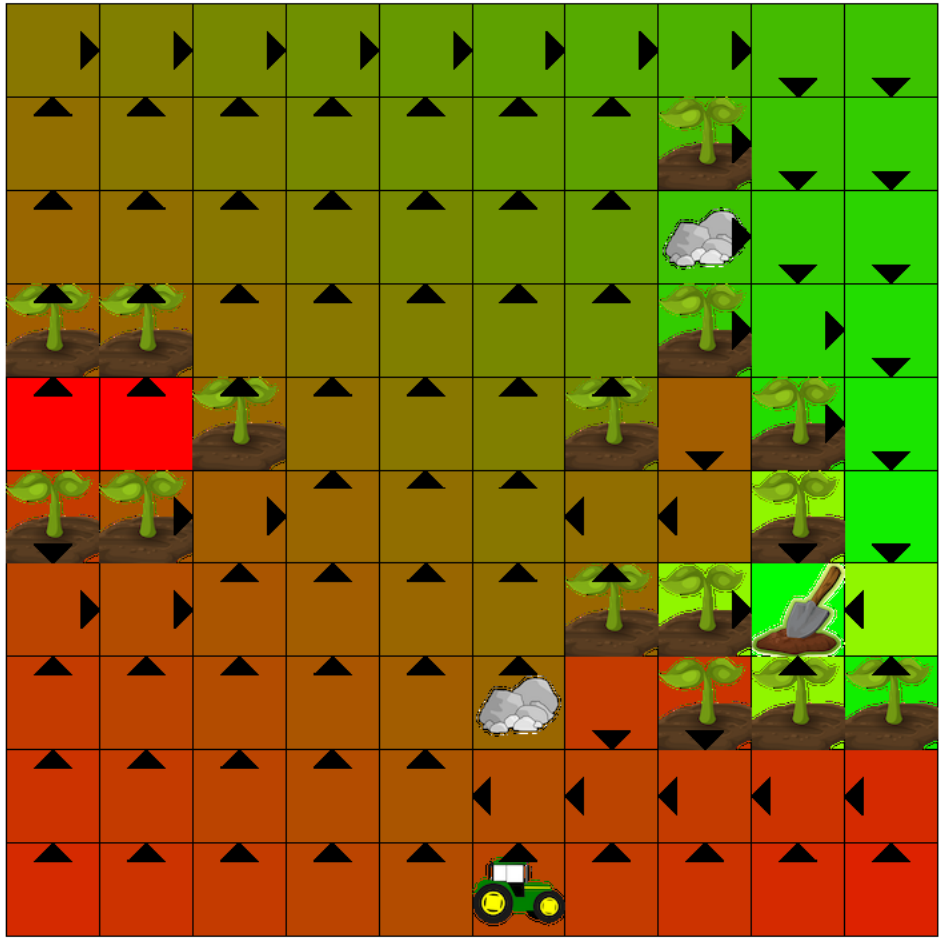
- Fibonacci sequence
- Scheduling
- Sequence alignment (DNA)
- AI Farm



By Jahobr - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=58460223



By en:User:Dcoatzee, traced by User:Stannered - en:Image:Fibonacci dynamic programming.png, Public Domain, https://commons.wikimedia.org/w/index.php?curid=3325402



Dynamic Programming: History

- The term *dynamic programming* was originally used in the 1940s by Richard Bellman to describe the process of solving problems where one needs to find the best decisions one after another.
- By 1953, he refined this to the modern meaning, referring specifically to nesting smaller decision problems inside larger decisions.
- The word *dynamic* was chosen by Bellman to capture the time-varying aspect of the problems, and because it sounded impressive.
- The word *programming* referred to the use of the method to find an optimal *program*.

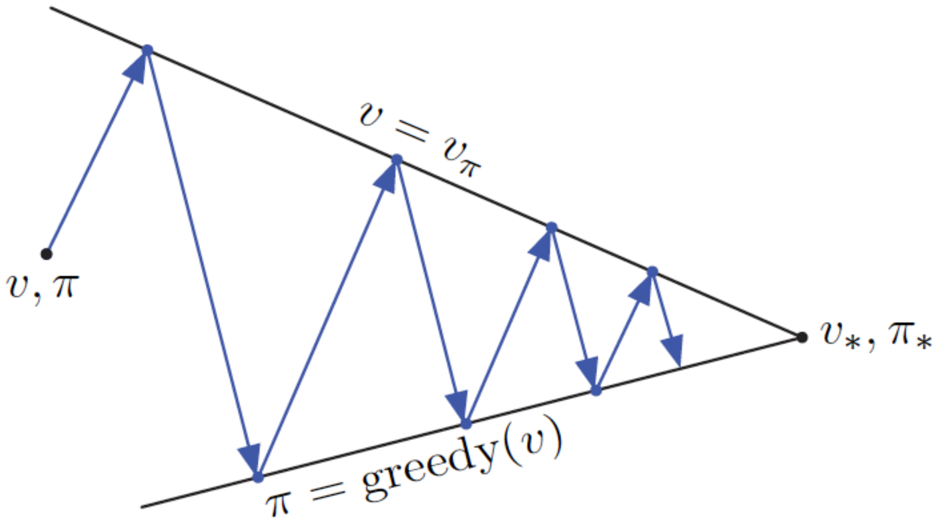
From: https://en.wikipedia.org/wiki/Dynamic_programming

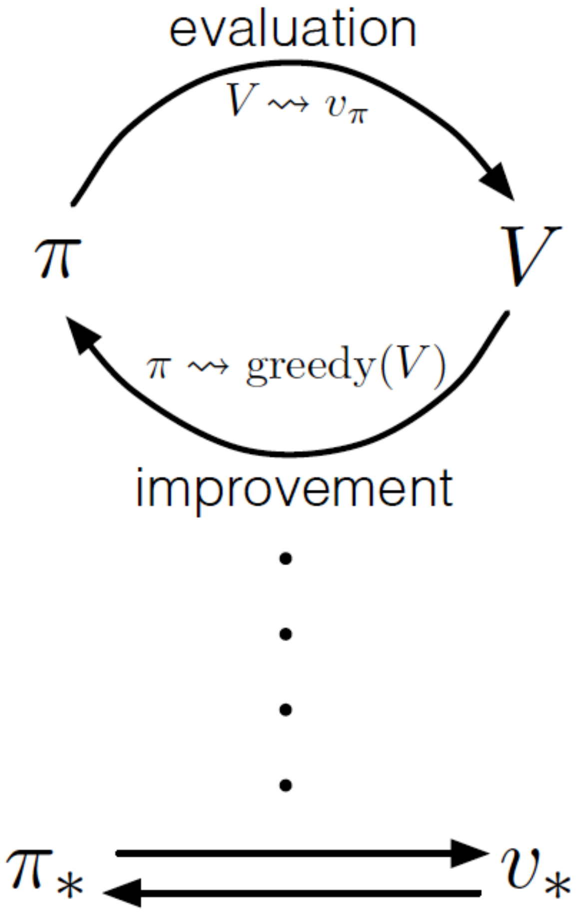
Dynamic Programming

- We will use it to evaluate a policy and compute an optimal policy given a perfect model an MDP
  - $p(s', r|s, a)$
- Foundational for reinforcement learning
- Using dynamic programming, we will do policy iteration by iterating between policy evaluation and policy improvement

Generalized Policy Iteration

- **Policy Evaluation:** Estimate the expected future reward when following policy π
- **Policy Improvement:** Improve policy π so that it obtains a greater expected future reward
- We can obtain an optimal policy by iterating between policy evaluation and policy improvement



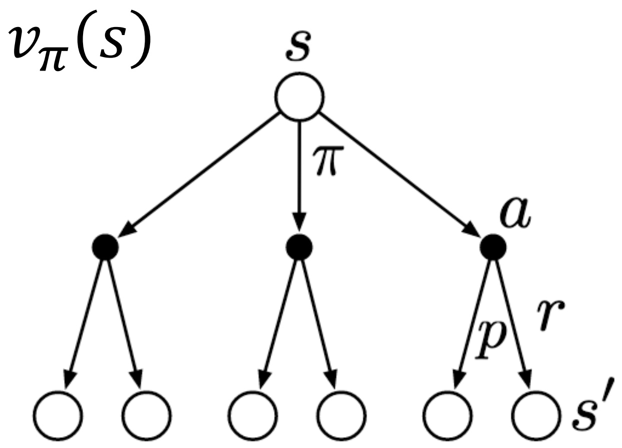


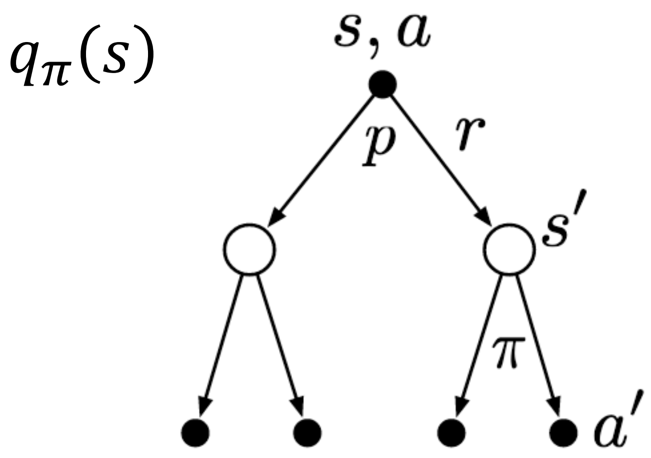
Dynamic Programming: Applications

- We assume that we are given a model of the MDP that characterizes our problem
- While this assumption does not hold in many contexts, it does in many others
  - Organic chemistry
  - Puzzles
  - Quantum computing
  - Theorem proving
- Furthermore, it builds the foundation that we will use to explore model-free algorithms

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





Outline

- Background
- Policy Evaluation
- Policy Improvement
- Policy Iteration
  - Modified Policy Iteration
  - Value Iteration
- Approximate value iteration

Policy Evaluation

- Estimate the expected future reward when following policy π
- From the Bellman equation we know that
  - $v_\pi(s) = \sum_a \pi(a|s)(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s'))$
- Given a policy π, what if we searched for a function V that satisfies the Bellman equation?
  - Will we have successfully evaluated π?
- If so, how should we search for V?
- Use the Bellman equation as an update rule
  - $V(s) = \sum_a \pi(a|s)(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s'))$

Policy Evaluation

Algorithm 1 Policy Evaluation

procedure POLICY EVALUATION($\mathcal{S}, V, \pi, \gamma$)
    $\Delta \leftarrow \inf$
    while $\Delta > 0$ **do**
        $\Delta \leftarrow 0$
        for $s \in \mathcal{S}$ **do**
            $v \leftarrow V(s)$
            $V(s) \leftarrow \sum_a \pi(a|s)(r(s,a) + \gamma \sum_{s'} p(s'|s,a)V(s'))$
            $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
        end for
    end while
    return V $\triangleright\ v_\pi$
end procedure

Is this guaranteed to converge to v_π if the threshold is 0?

Policy Evaluation: Convergence

$$v_\pi(s) = \sum_a \pi(a|s)(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s'))$$

$$v_\pi(s) = \sum_a \pi(a|s) r(s,a) + \gamma \sum_a \pi(a|s) \sum_{s'} p(s'|s,a)\, v_\pi(s')$$

$$\boldsymbol{v}^\pi = \boldsymbol{r}^\pi + \gamma \boldsymbol{P}^\pi \boldsymbol{v}^\pi$$

- Matrices:
  - $\boldsymbol{v}^{\boldsymbol{\pi}}$ is a vector of values for each state, size $|\mathcal{S}|$
  - $\boldsymbol{r}^{\boldsymbol{\pi}}$ is a vector of rewards for each state, size $|\mathcal{S}|$
  - $\boldsymbol{P}^{\boldsymbol{\pi}}$ is a matrix of transition probabilities for each pair of states, size $|\mathcal{S}|$x $|\mathcal{S}|$

Policy Evaluation: Convergence

- Define T as the Bellman backup operator:

$$T(\boldsymbol{v}) \coloneqq \boldsymbol{r}^{\pi} + \gamma \boldsymbol{P}^{\pi} \boldsymbol{v}$$

- We know that there exists a fixed point

$$T(\boldsymbol{v}^{\pi}) = \boldsymbol{v}^{\pi}$$

- The infinity norm:

$$||\boldsymbol{x}||_{\infty} \coloneqq \max_{\mathrm{i}} |\mathrm{x_i}|$$

- If:

$$||T(\boldsymbol{u}) - T(\boldsymbol{v})||_{\infty} \leq \gamma ||\boldsymbol{u} - \boldsymbol{v}||_{\infty}$$

$$\text{for } \gamma \in [0,1)$$

- Then T is a contraction mapping
- Banach-Caccioppoli fixed point theorem proves that repeated applications of T will converge to a unique fixed point (i.e. $\boldsymbol{v}^{\pi}$).

Policy Evaluation: Convergence

$$\left|\left|T(\boldsymbol{u}) - T(\boldsymbol{v})\right|\right|_{\infty} = \left|\left|\boldsymbol{r}^{\pi} + \gamma \boldsymbol{P}^{\pi}\boldsymbol{u} - \boldsymbol{r}^{\pi} + \gamma \boldsymbol{P}^{\pi}\boldsymbol{v}\right|\right|_{\infty}$$

$$= \left|\left|\gamma \boldsymbol{P}^{\pi}\boldsymbol{u} + \gamma \boldsymbol{P}^{\pi}\boldsymbol{v}\right|\right|_{\infty}$$

$$= \gamma\left|\left|\boldsymbol{P}^{\pi}(\boldsymbol{u} + \boldsymbol{v})\right|\right|_{\infty}$$

$$\leq \gamma\left|\left|\boldsymbol{P}^{\pi}\left|\left|\boldsymbol{u} + \boldsymbol{v}\right|\right|_{\infty}\right|\right|_{\infty}$$ // $\boldsymbol{P}^{\pi}$ is a matrix of transition probabilities, sums to 1

$$\leq \gamma\left|\left|\boldsymbol{u} + \boldsymbol{v}\right|\right|_{\infty}$$

Policy Evaluation AI Farm

- Policy: Uniform random
- AI Farm took 4576 iterations to converge with $\gamma = 1$
  - $\gamma = 0.99$ takes 1345 iterations
  - $\gamma = 0.9$ takes 191 iterations
- What if policy said to always go up?
  - $\gamma = 1$
  - $\gamma < 1$

Outline

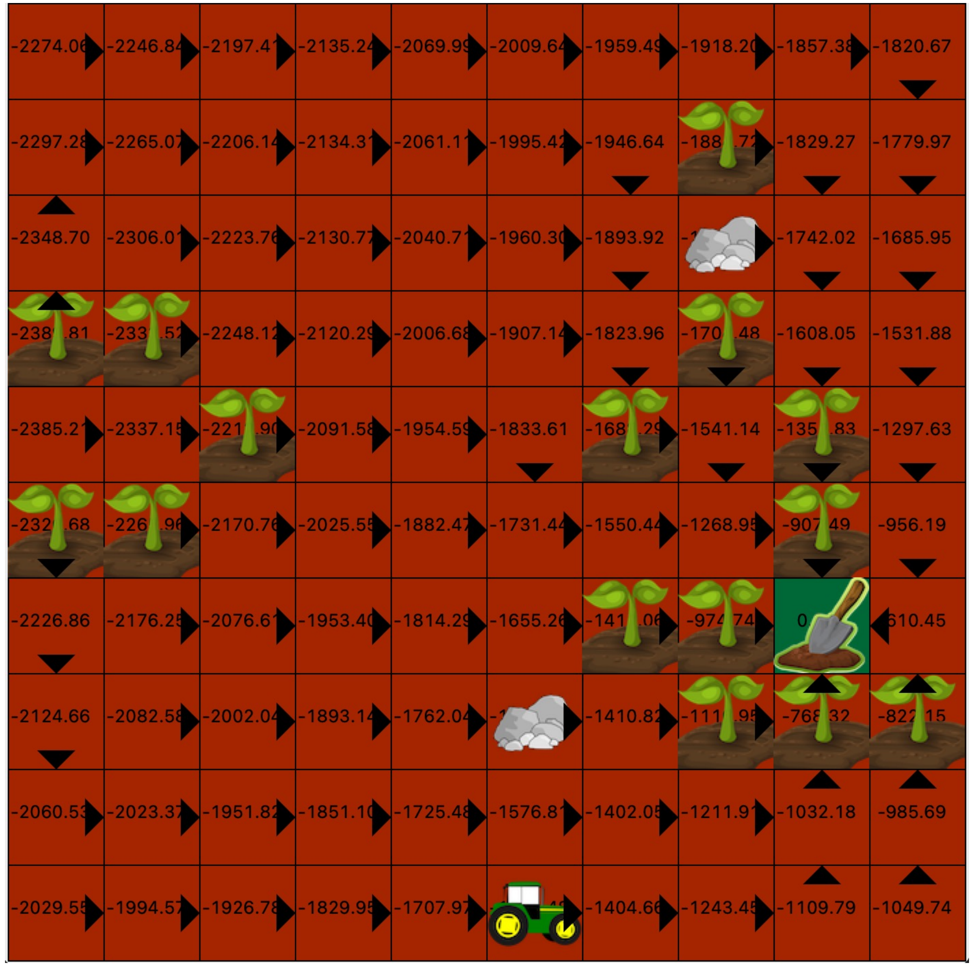
- Background
- Policy Evaluation
- Policy Improvement
- Policy Iteration
  - Modified Policy Iteration
  - Value Iteration
- Approximate value iteration

Policy Improvement

- We have evaluated policy π, how can we find a better policy?
- $\pi' \geq \pi$ if and only if $v_{\pi'}(s) \geq v_\pi(s)$ for all $s \in \mathcal{S}$
- We set the policy to be greedy with respect to v_π
- $\pi'(s) = \underset{a}{\operatorname{argmax}}(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_\pi(s'))$
- This policy will always be the same or better than the previous one
  - Policy improvement theorem.

Policy Improvement

- Better than original uniform random policy
- Still not optimal



When acting greedily with respect to v_π

Outline

- Background
- Policy Evaluation
- Policy Improvement
- Policy Iteration
  - Modified Policy Iteration
  - Value Iteration
- Approximate value iteration

Policy Iteration

- Policy improvement improves a policy π and obtains a new policy π' such that, $\pi' \geq \pi$
- If $\pi' = \pi$, then $v_{\pi'} = v_\pi$
- Therefore, $v_{\pi'}(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_{\pi'}(s'))$
- This is the same as the Bellman optimality equation!
- Does this mean $\pi' = \pi_*$ and $v_{\pi'} = v_*$?
- There is a proof showing that the Bellman optimality equation is a unique fixed point
  - Similar to that of the Bellman equation

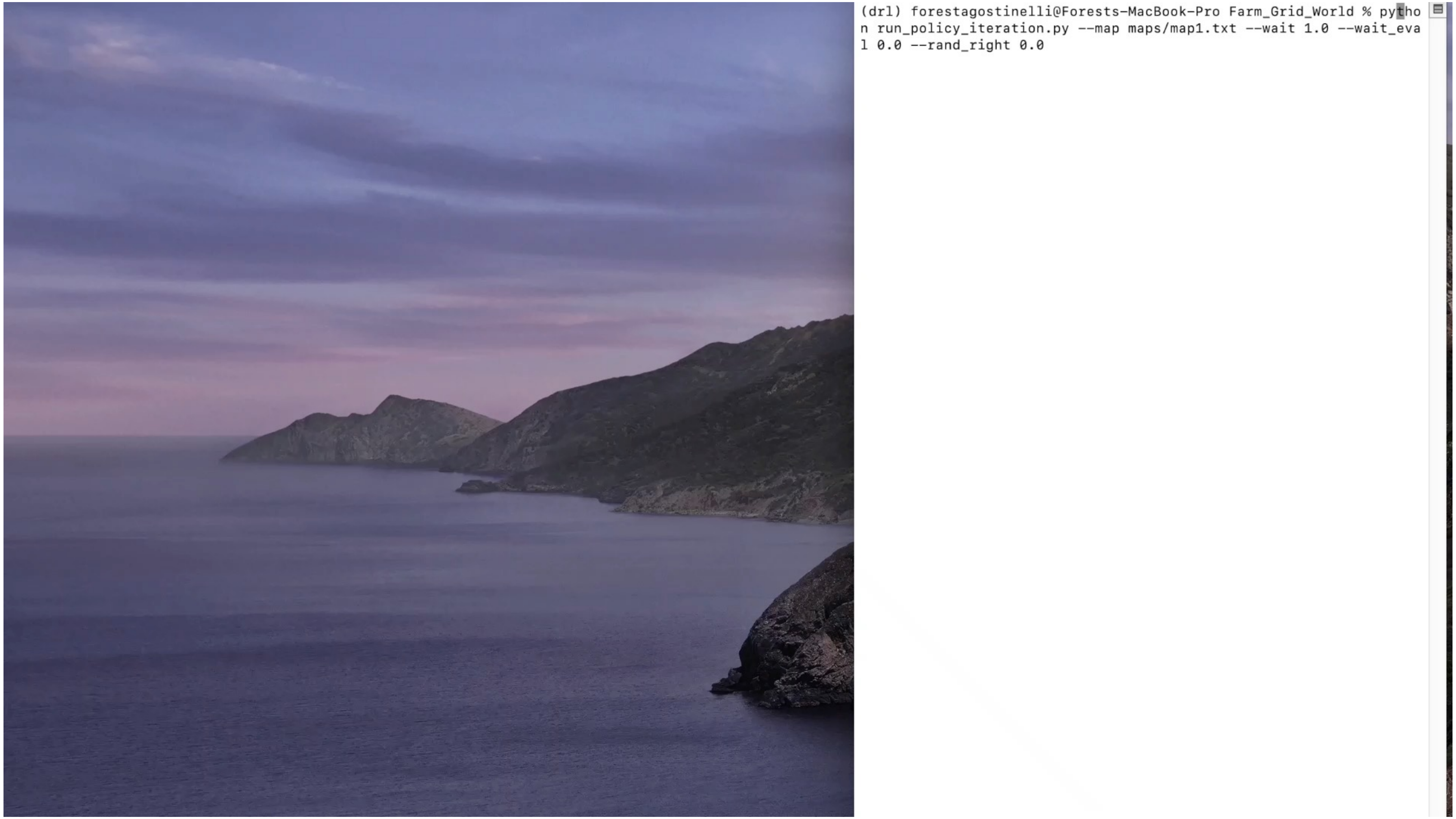
Policy Iteration

Algorithm Policy Iteration

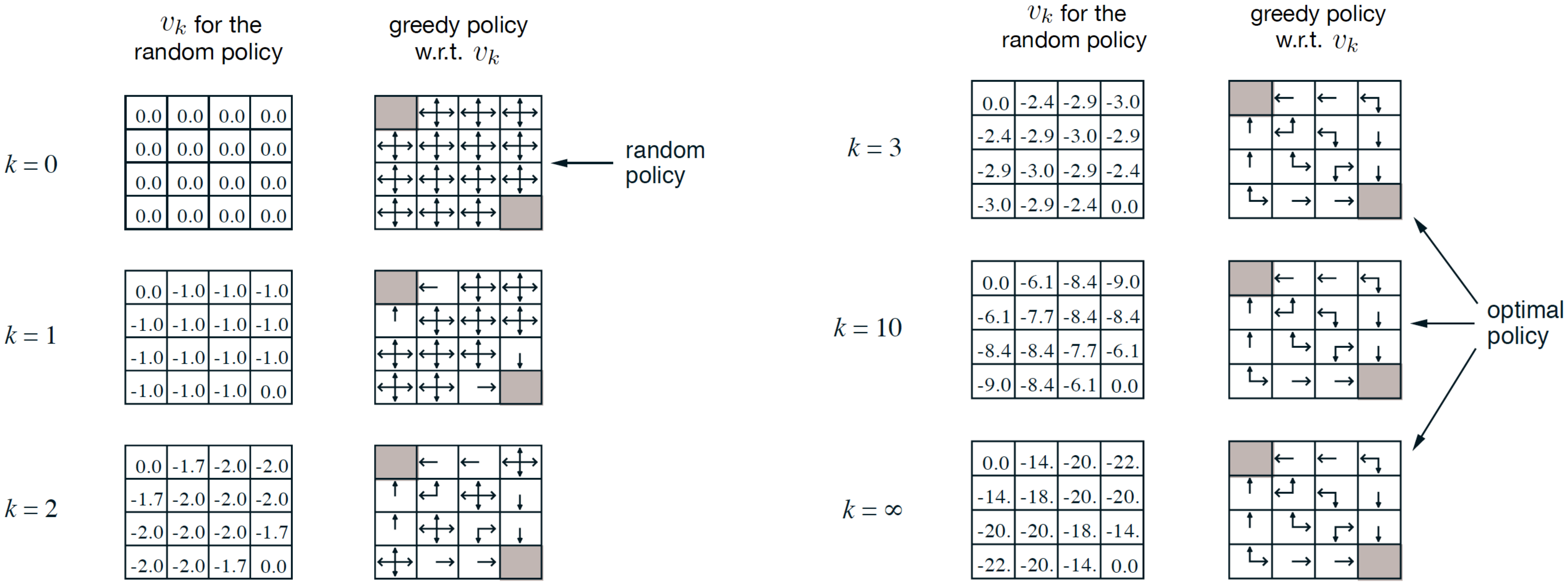
Inputs:
π: Initial Policy
V: Initial value function. Value of terminal state must be zero
θ: Termination threshold

while has not converged **do**
V = Policy_Evaluation(π, V, θ)
π = Policy_Improvement(V)
end while
return π ▷ Approximation of $\pi*$

Policy Iteration: AI Farm



Do We Have to Wait For Policy Evaluation to Converge?

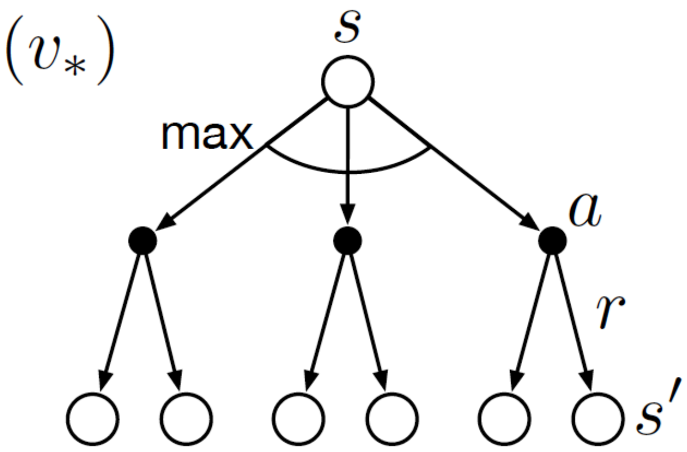


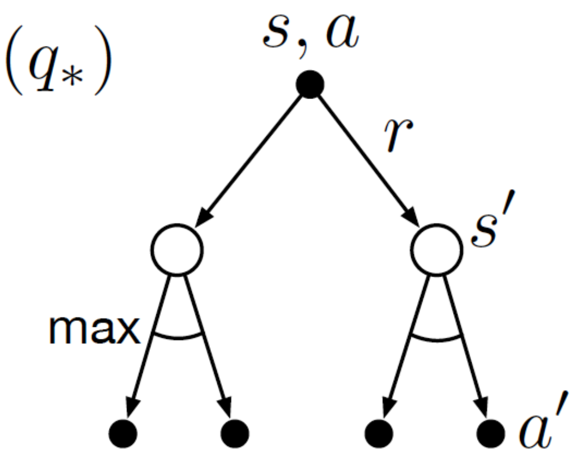
Modified Policy Iteration

- We do not have to run policy evaluation to convergence.
- We can stop after reaching some threshold
- We can stop after k iterations
  - $k = 1$ is the same as value iteration

Bellman Optimality Equation

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





Value Iteration

- Find the optimal value function
- Recall the Bellman optimality equation
  - $v_*(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, v_*(s'))$
- Use this as an update rule
  - $V(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s'))$
- Combines policy evaluation and policy improvement into one step
- There is a proof similar to that of policy evaluation to prove that repeatedly updating V will converge to a unique fixed point
  - Bellman optimality equation

Value Iteration

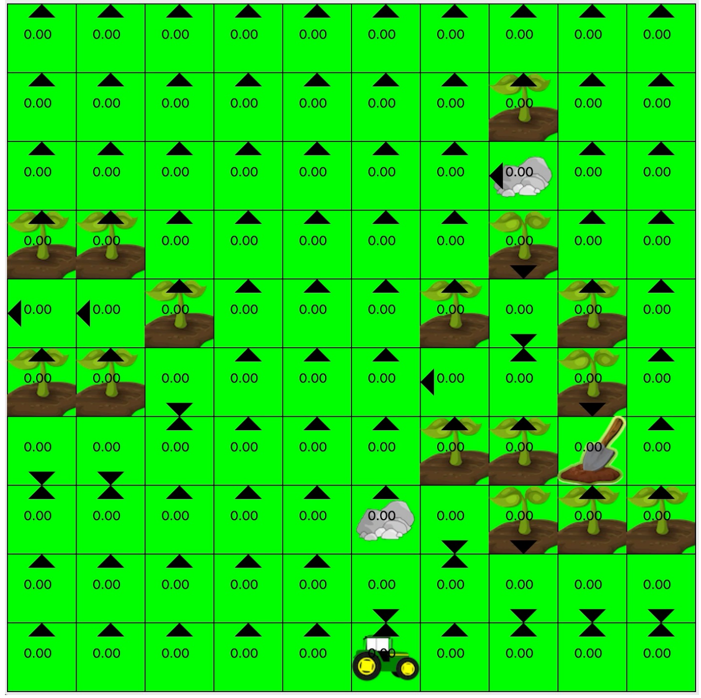
Algorithm Value Iteration

Inputs:
  V: Initial value function. Value of terminal state must be zero
  θ: Termination threshold

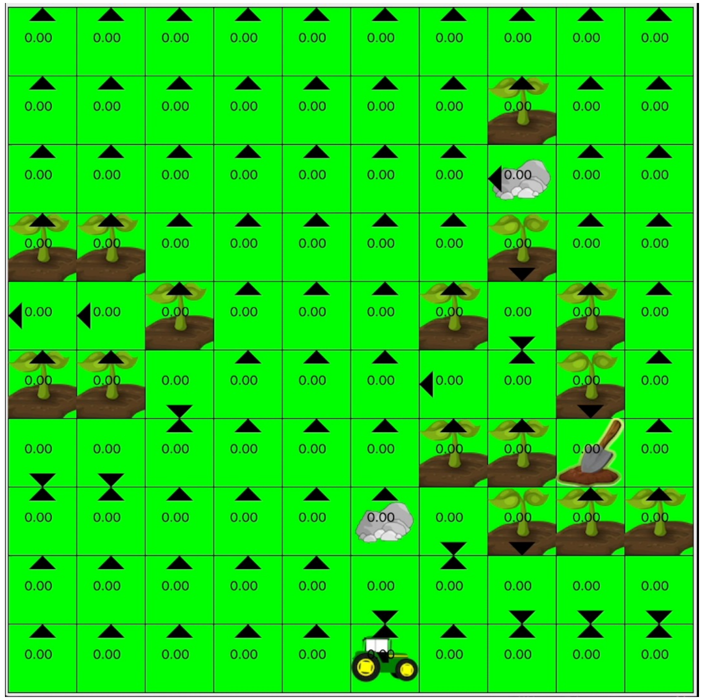
procedure VALUE_ITERATION(V, θ, γ)
  $\Delta \leftarrow \inf$
  while $\Delta > \theta$ **do**
    $\Delta \leftarrow 0$
    for $s \in \mathcal{S}$ **do**
      $v \leftarrow V(s)$
      $V(s) \leftarrow \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a) V(s'))$
      $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
  end while
  return V ▷ Approximation of v_*
end procedure

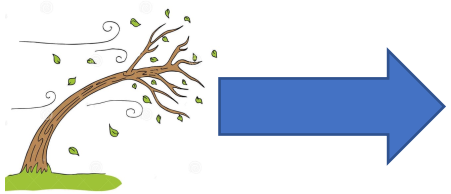
Value Iteration: AI Farm

$p = 0$

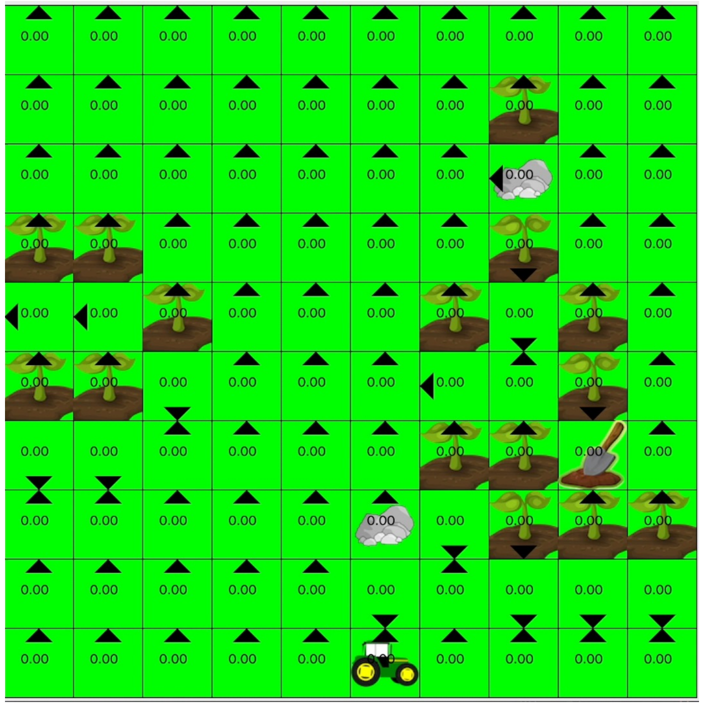


$p = 0.1$





$p = 0.5$



Asynchronous Policy Iteration

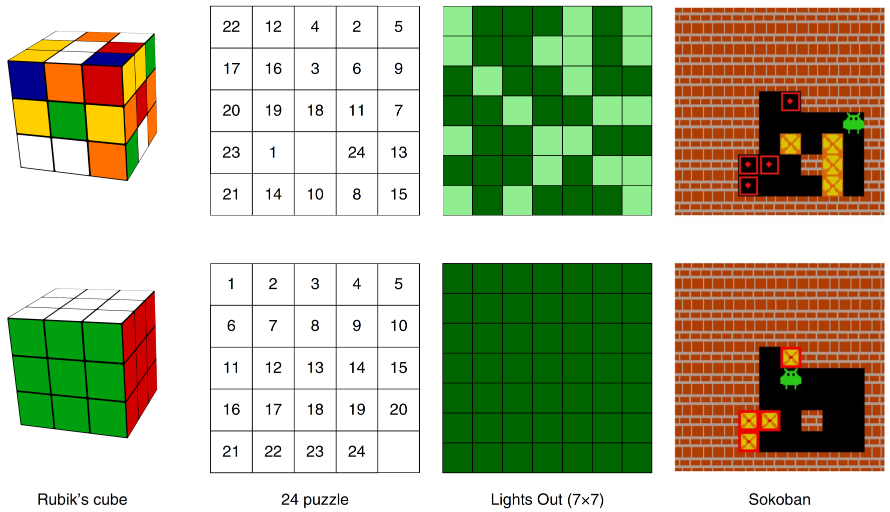
- We do not have to update all states at every iteration
- Prioritize states based on Bellman error
  - Difference between current and updated estimation of state value
- Prioritize states based on relevance to the agent's experience
- To converge, must continue to update all states, though it does not have to be at every iteration

Outline

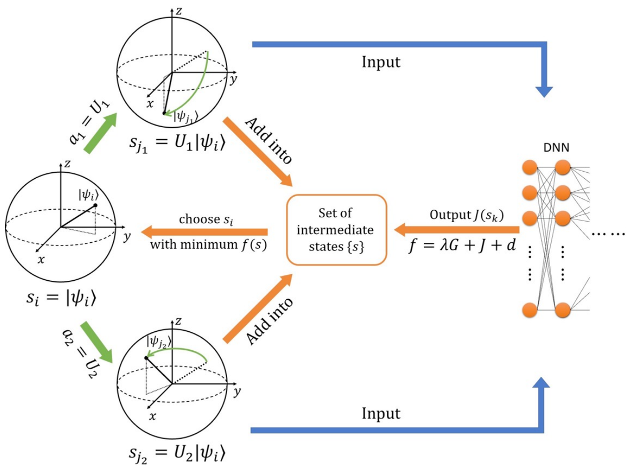
- Background
- Policy Evaluation
- Policy Improvement
- Policy Iteration
  - Modified Policy Iteration
  - Value Iteration
- Approximate value iteration

Approximate Value Iteration

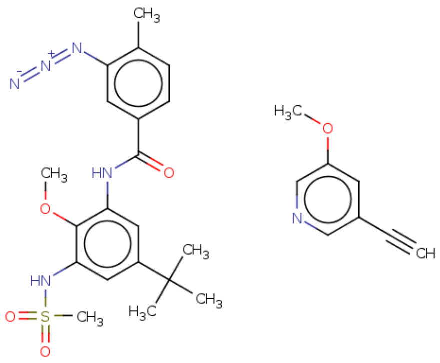
- Sometimes, we can accurately define the MDP, however, the state space is too large to store in a table
- Therefore, we will need to use an architecture to approximate the value
- The goal is to have an architecture whose number of parameters is far less than the size of the state space



Puzzles



Quantum Compiling



Chemical Synthesis

Approximate Value Iteration

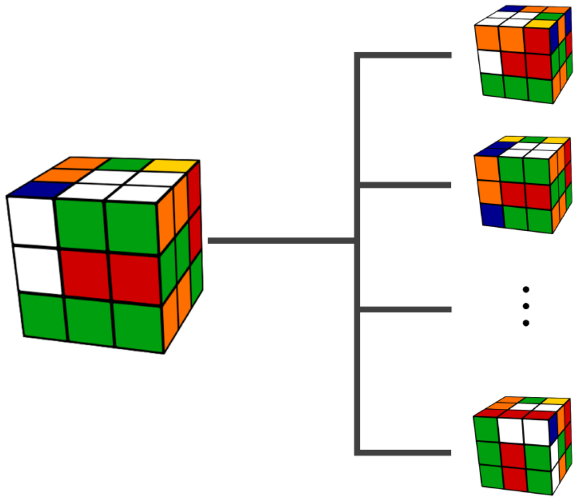
- Value Iteration
  - $V(s) = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s'))$
- Approximate Value Iteration
  - $y = \max_a (r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, \hat{v}(s', \boldsymbol{w}))$
  - $E(\boldsymbol{w}) = \frac{1}{2}\big(y - \hat{v}(s, \boldsymbol{w})\big)^2$
  - $\nabla_{\mathbf{w}} E(\mathbf{w}) = \big(y - \hat{v}(s, \mathbf{w})\big) \nabla_{\mathbf{w}} \hat{v}(s, \boldsymbol{w})$
  - Even though y depends on $\mathbf{w}$, we do not differentiate y with respect to $\mathbf{w}$.
  - Only supervision is that $y = 0$ for terminal states

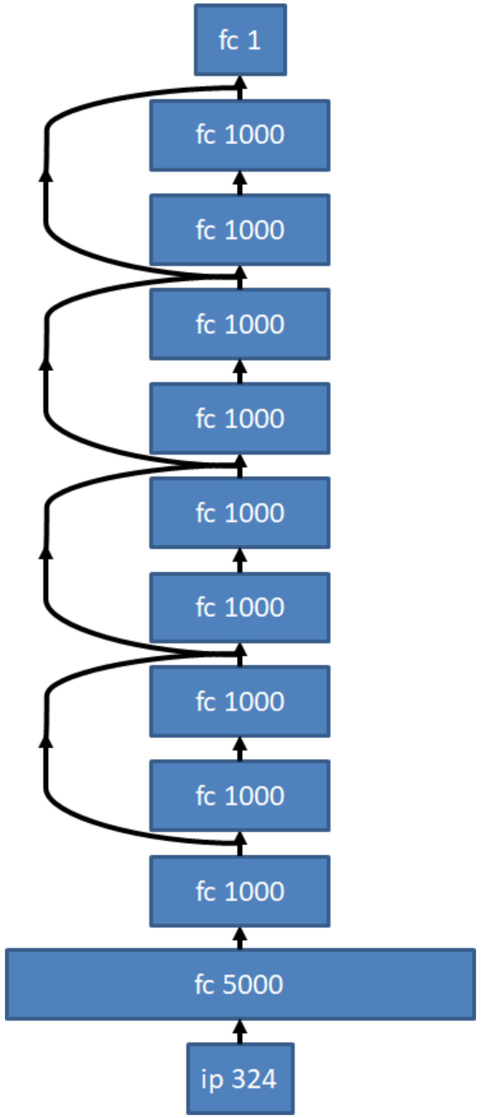
Convergence

- Unlike in the tabular case, we cannot guarantee convergence in the case of non-linear function approximators
- However, there are many different methods that we can use to make deep reinforcement learning work in practice

Deep Approximate Value Iteration

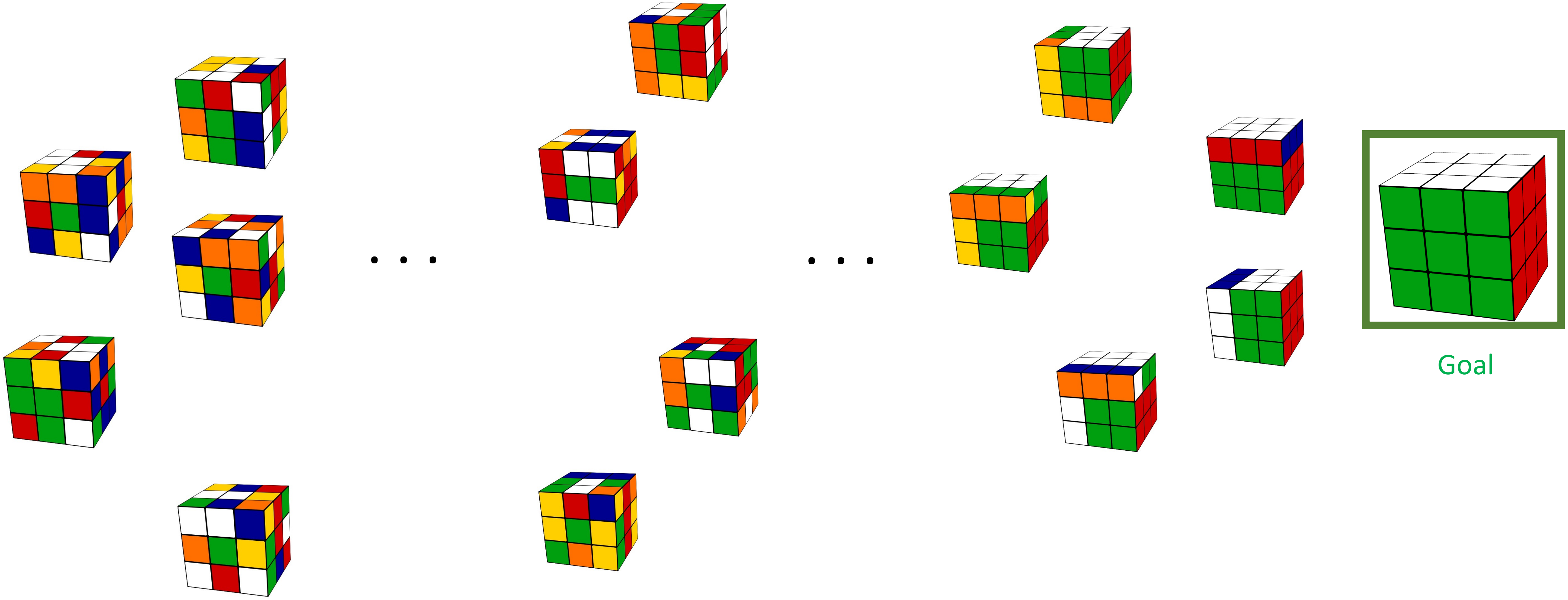
- $y = \max_a \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, \hat{v}(s', \boldsymbol{w})\right)$
- $E(\boldsymbol{w}) = \frac{1}{2}\left(y - \hat{v}(s, \boldsymbol{w})\right)^2$





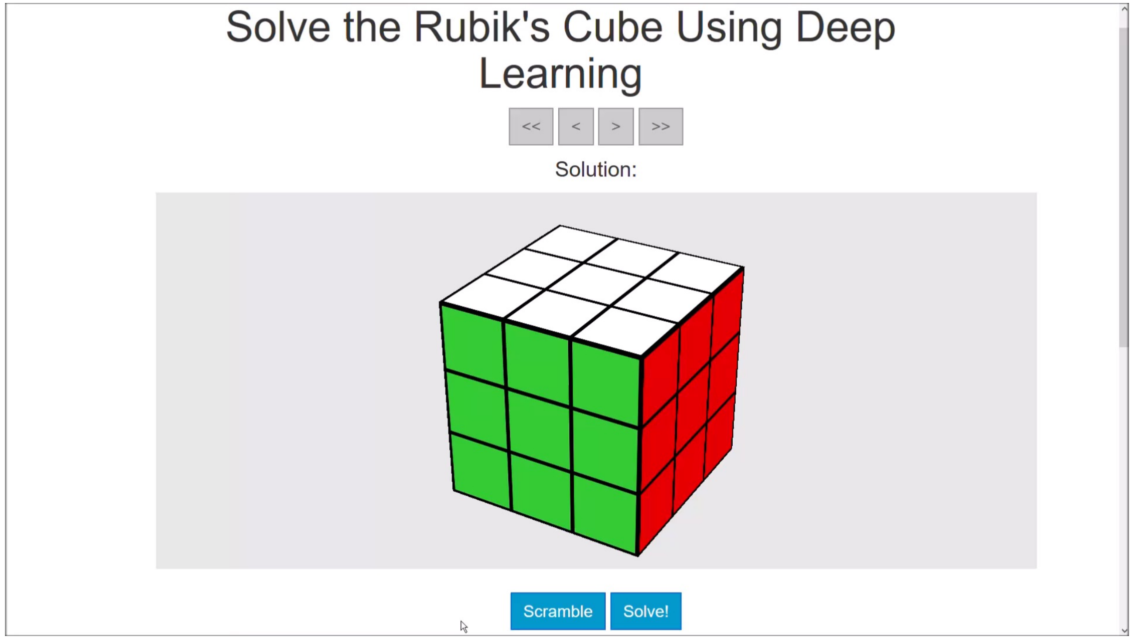
Prioritized Sweeping

- Prioritized sweeping: Generate training data by taking moves in reverse from the goal

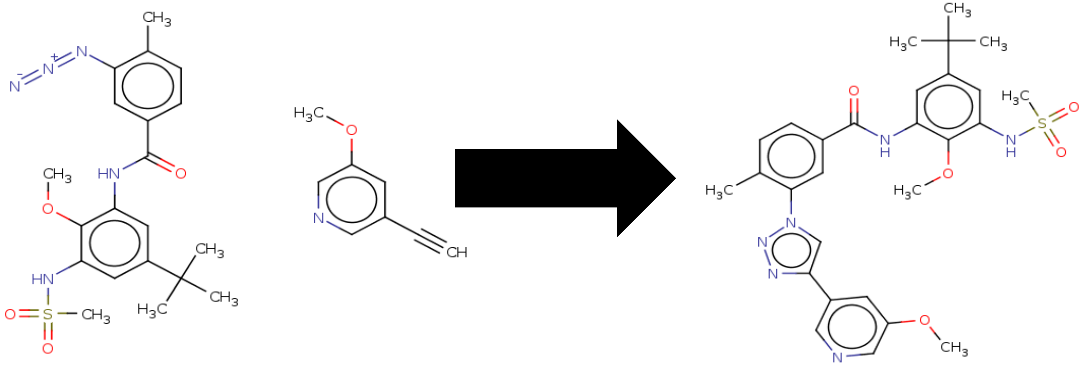


Solving the Rubik's Cube

- Deep approximate value iteration and A* search
- Solves the Rubik's cube and 6 other puzzles
- Can be applied to other areas in the natural sciences



http://deepcube.igb.uci.edu/



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

Summary

- **Policy Evaluation**: Uses Bellman equation as an update rule

$$V(s) = \sum_{a} \pi(a|s)\left(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s')\right)$$

- **Policy Improvement**: Behave greedily with respect to value function

$$\pi'(s) = \operatorname*{argmax}_{a}\left(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s')\right)$$

- **Policy Iteration**: Iterate between policy evaluation and policy improvement until convergence
- **Value Iteration**: Uses Bellman optimality equation as an update rule

$$V(s) = \max_{a}\left(r(s,a) + \gamma \sum_{s'} p(s'|s,a)\, V(s')\right)$$