Dynamic Programming

CSE 4/546 Reinforcement Learning – Deep Dive Session
Feb 23, 2024

Presented By:

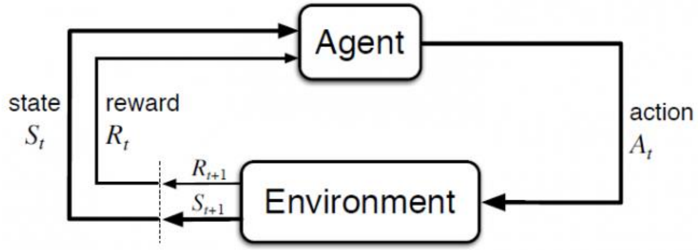
- Lasvitha Pregada (lasvitha@buffalo.edu)
- Smitha Kannur Ashok (smithaka@buffalo.edu)

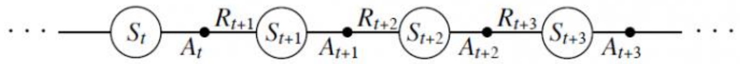
Topics Covered

- MDP and Bellman Optimality Equation
- Introduction to Dynamic Programming
- Why Dynamic Programming?
- Breakdown of Dynamic Programming
- Problem Statement
- Policy Evaluation
- Policy Improvement
- Policy Iteration
- Value Iteration
- Efficiency of Dynamic Programming

MDP and Bellman Optimality Equation

Markov Decision Process





- St: State of the agent at time t
- At: Action taken by agent at time t
- Rt: Reward obtained at time t

Bellman Optimality Equation

In the context of reinforcement learning, the Bellman Optimality Equation expresses the relationship between the value of a state (or state-action pair) and the values of its possible successor states (or successor state-action pairs).

- **For state values (V):**

  $V^*(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) \cdot V^*(s')\right)$

- **For action values (Q):**

  $Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) \cdot \max_{a'} Q^*(s',a')$

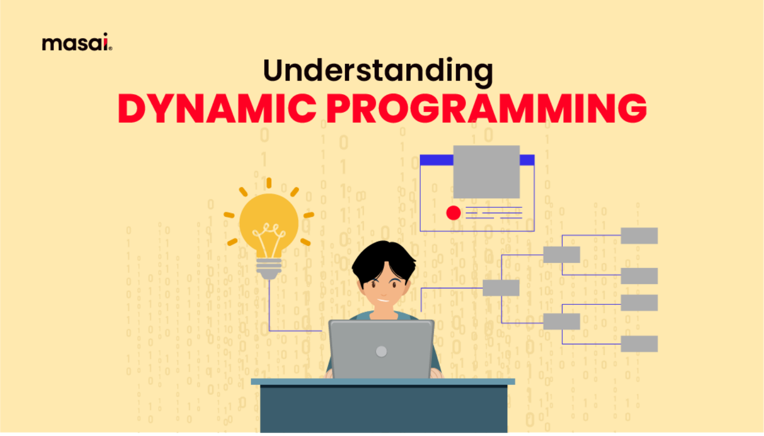
Here:

- $V^*(s)$ is the optimal value of state s.
- $Q^*(s,a)$ is the optimal value of taking action a in state s.
- $R(s,a)$ is the immediate reward for taking action a in state s.
- $P(s'|s,a)$ is the probability of transitioning to state s' given that action a is taken in state s.
- γ is the discount factor, which accounts for the present value of future rewards.

Introduction to Dynamic Programming

What is Dynamic Programming :

- An optimization technique used to solve complex problems by breaking them into smaller subproblems and use them to find overall optimal solution.
- A planning problem where in given a complete MDP, Dynamic Programming can find an optimal policy.



A connection to Reinforcement learning:

- Has foundation and algorithms related to MDP's
- RL directly employs Dynamic Programming for problem solving.

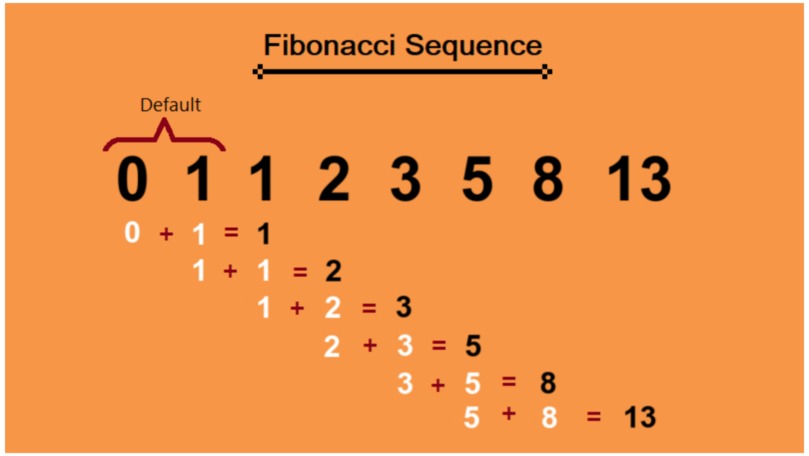
Why Dynamic Programming?

Offers efficient solutions to problems that exhibits overlapping subproblems and optimal substructure.

Fibonacci Series:

Dynamic Programming Solution:

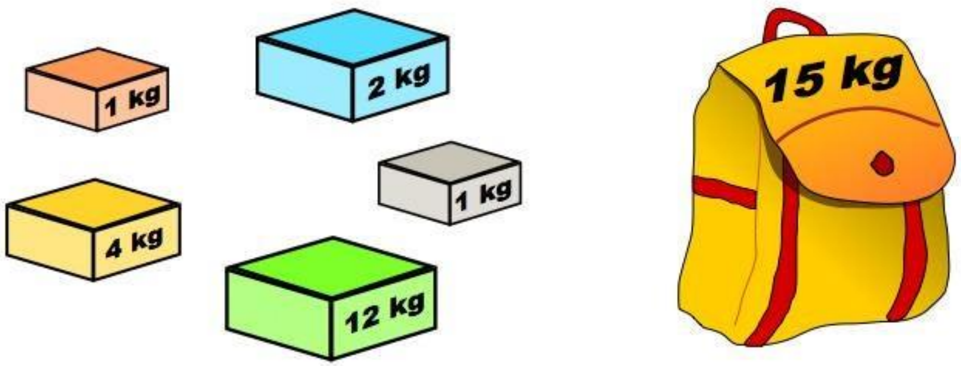
- Stores previously computed Fibonacci numbers in an array.



Knapsack problem:

Dynamic Programming Solution:

- Breaks down the problem into smaller subproblems and lowe weight limits to build the solution iteratively.



Breakdown of Dynamic Programming

- Dynamic Programming solve a category of problems called planning problems which given the model and specifications of MDP can find the optimal policy.

- To solve the MDP and get optimal policy, the solution should include these components:

Policy Evaluation
- To find out how good a policy is

Policy Improvemen t
- To improve an arbitrary policy

Policy Iteration
- Policy Evaluation + Policy Iteration.

Problem Statement

An explanation of "Frozen Lake Environment" to understand dynamic programming

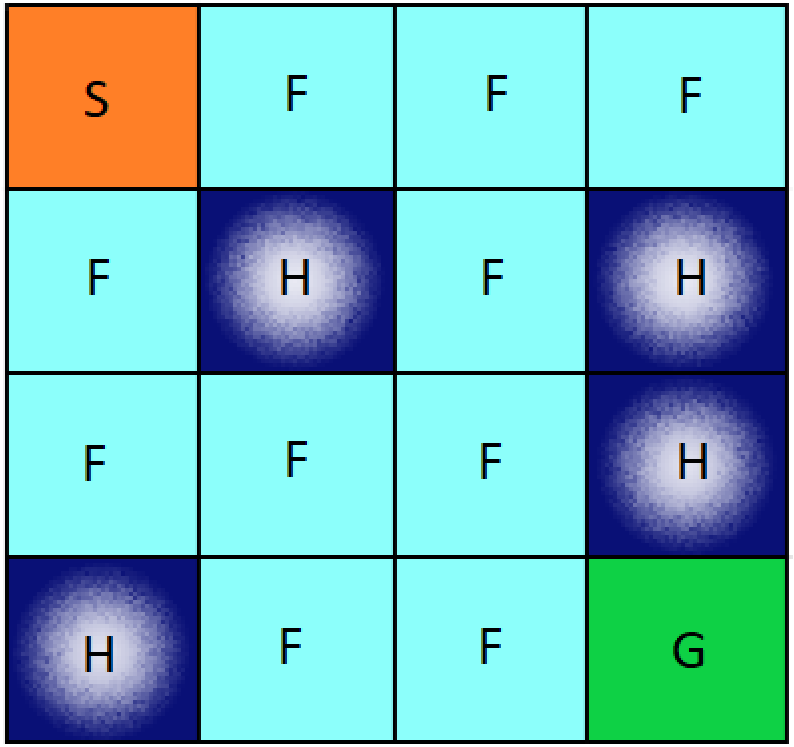
The grid has the following elements:

- **S**: Starting point
- **G**: Goal
- **F**: Frozen ice (safe to step on)
- **H**: Hole (fall into the hole, and the episode ends)

Positive reward : When reached state G

Negative reward : When reached state H

<u>OBJECTIVE:</u> To find the optimal policy that maximizes the expected cumulative reward over time.



Policy Evaluation

Policy Evaluation is a dynamic programming algorithm used to calculate an SVF to evaluate if the determined policy is suitable for a given MDP.

Inputs:

1. Markov Decision Process (MDP):
- States (S)
- Actions (A)
- Transition probabilities (P)
- Rewards (R)
- Discount factor (γ)

2 . Policy π

Outputs:

1. State Value Function ($V\pi$) :
   The estimated value of each state under given policy π.

Pros:

1. Convergence
2. Versatility
3. Applicability

Cons:

1. Computational Complexity
2. Memory requirements
3. Dependency on accurate models

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

Pseudocode[1]: Richard S. Sutton and Andrew G. Barto Book

Policy Improvement

Policy Iteration is a dynamic programming algorithm used to find if there is any policy that could improve the performance of existing policy

Inputs:

1. Markov Decision Process (MDP):
   - States (S), Actions (A),Transition probabilities (P), Rewards (R), Discount factor (γ)

2. Current Policy (π) :
   - The policy for which. Policy improvement is perfromed.

3. State Value function (Vπ) :
   - The state value function for current policy π.

Outputs:

1. Improved Policy (π'):
   - A new policy that is expected to be better than the current policy.

Pros:

1. Greedy Improvement
2. Iterative Refinement
3. Flexibility

Cons:

1. Local Optimal
2. Computational Complexity
3. Dependency on Accurate Value Estimates.

$$
\begin{aligned}
V^\pi(s) &\leq Q^\pi(s, \pi'(s)) \\
&= E_{\pi'}\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s\} \\
&\leq E_{\pi'}\{r_{t+1} + \gamma Q^\pi(s_{t+1}, \pi'(s_{t+1})) \mid s_t = s\} \\
&= E_{\pi'}\{r_{t+1} + \gamma E_{\pi'}\{r_{t+2} + \gamma V^\pi(s_{t+2})\} \mid s_t = s\} \\
&= E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 V^\pi(s_{t+2}) \mid s_t = s\right\} \\
&\leq E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^\pi(s_{t+3}) \mid s_t = s\right\} \\
&\;\vdots \\
&\leq E_{\pi'}\left\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots \mid s_t = s\right\} \\
&= V^{\pi'}(s).
\end{aligned}
$$

Pseudocode[1]: Richard S. Sutton and Andrew G. Barto Book

Policy Iteration

Policy Iteration is a dynamic programming algorithm used to find an optimal policy for a Markov Decision Process (MDP)

Inputs:

1. Markov Decision Process (MDP):
- States (S)
- Actions (A)
- Transition probabilities (P)
- Rewards (R)
- Discount factor (γ)

Outputs:

1. Optimal Policy ($\pi*$):
   - A policy that maximizes the expected cumulative reward.
2. Optimal Value Function (V*):
   - The value function associated with the optimal policy.

Pros:

1. Guaranteed Convergence
2. Model Flexibility
3. Policy Improvement

Cons:

1. Computational Complexity
2. Not Suitable for Large MDPs
3. Deterministic Policies

1. Initialization

   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

   Repeat

   $\Delta \leftarrow 0$

   For each $s \in \mathcal{S}$:

   $v \leftarrow V(s)$

   $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[\mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right]$

   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement

   policy-stable $\leftarrow$ *true*

   For each $s \in \mathcal{S}$:

   $b \leftarrow \pi(s)$

   $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V(s') \right]$

   If $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*

   If *policy-stable*, then stop; else go to 2

Pseudocode[1]: Richard S. Sutton and Andrew G. Barto Book

Value Iteration

Value Iteration is another dynamic programming algorithm used to find an optimal policy for a Markov Decision Process (MDP)

Inputs:

1. Markov Decision Process (MDP):
- States (S)
- Actions (A)
- Transition probabilities (P)
- Rewards (R)
- Discount factor (γ)

Outputs:

1. Optimal Policy ($\pi*$):
    - A policy that maximizes the expected cumulative reward.
2. Optimal Value Function (V*):
    - The value function associated with the optimal policy.

Pros:

1. Simplicity
2. Space Efficiency
3. Convergence Speed

Cons:

1. Deterministic Policies
2. Not Guaranteed to Converge to Exact Solution
3. May Require Tuning

Initialize V arbitrarily, e.g.,$V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$\quad \Delta \leftarrow 0$

$\quad$ For each $s \in \mathcal{S}$:

$\quad\quad v \leftarrow V(s)$

$\quad\quad V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

$\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that

$\quad \pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Pseudocode[1]: Richard S. Sutton and Andrew G. Barto Book

Efficiency of Dynamic Programming

Dynamic programming (DP) is a powerful approach in reinforcement learning (RL) that is particularly effective for solving problems modeled as Markov Decision Processes (MDPs).

- DP is most efficient when the problem has the Markov property
- Well-suited for problems with finite state and action spaces
- Highly efficient when the problem exhibits optimal substructure and overlapping subproblems
- For small to moderately sized problems, DP can often provide exact solutions.
- The choice between Policy Iteration and Value Iteration can impact efficiency.
- DP assumes complete knowledge of the transition probabilities and rewards.
- DP methods typically do not handle the exploration-exploitation trade-off explicitly.

References

- *Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press.*
- *Dynamic Programming Notebook*

Thank You