Week 3: Dynamic Programming Basics

Your Name

Your Institution

June 30, 2025

Introduction to Dynamic Programming

Dynamic Programming (DP) is a powerful algorithmic technique for solving complex problems by breaking them down into simpler subproblems. It is particularly relevant in Reinforcement Learning (RL) as it provides systematic methods for solving Markov Decision Processes (MDPs).

Key Concepts

- **Markov Decision Processes (MDPs)**: Frameworks for modeling decision-making.
  - **States**: Possible situations the agent can be in.
  - **Actions**: Choices available to the agent at each state.
  - **Transition Probabilities**: Probability of moving from one state to another given an action.
  - **Rewards**: Immediate returns received after state transitions.
- **Reinforcement Learning (RL)**: Agents learn to make decisions through interaction to maximize cumulative rewards.

Importance of Dynamic Programming in RL

Dynamic programming methods, such as Value Iteration and Policy Iteration, enable agents to evaluate and improve their policies iteratively. These methods leverage the principles of optimality and effectively address exploration-exploitation tradeoffs in decision-making.

Examples to Illustrate DP:

1. **Fibonacci Sequence:** Efficiently calculated with memoization:

$$F(n) = F(n-1) + F(n-2) \tag{1}$$

2. **Shortest Path in a Grid:** Using DP to build a table of minimum paths.

Summary

- Dynamic programming reduces computational time by avoiding repeated calculations (overlapping subproblems) using optimality of substructure.
- It is foundational for developing algorithms that derive optimal policies in reinforcement learning settings, particularly in MDPs.
- Understanding DP principles and methods bridges theoretical decision-making with practical RL implementations.

What is Dynamic Programming?

Definition

Dynamic Programming (DP) is a method for solving complex problems by breaking them down into simpler subproblems. It is useful in circumstances where the problem can be divided into overlapping subproblems that can be solved independently.

Key Principles of Dynamic Programming

1. **Optimal Substructure**
   - An optimal solution to the problem can be constructed from optimal solutions of its subproblems.
   - **Example:** Consider finding the shortest path in a weighted graph. If the shortest path from vertex A to vertex C passes through vertex B, then the paths from A to B and B to C must also be the shortest paths among their respective vertices.
   - **Formula:**

$$f(n) = \min(f(i) + f(j)) \quad \text{(for all valid } i, j\text{)}$$

2. **Overlapping Subproblems**
   - A problem has overlapping subproblems if the same subproblems are solved multiple times.
   - **Example:** The Fibonacci sequence.
   - **Recursive Formula:**

$$F(n) = F(n-1) + F(n-2) \quad \text{with base cases } F(0) = 0, F(1) = 1$$

Benefits of Dynamic Programming

- **Efficiency:** By storing previously computed results (memoization or tabulation), dynamic programming reduces time complexity significantly.
- **Reusability:** Once a subproblem is solved, its solution can be reused in larger problems, aiding in optimization.

Summary Key Points

- Dynamic programming is a powerful optimization approach based on solving subproblems and utilizing their solutions.
- Key principles include **optimal substructure** and **overlapping subproblems**.
- By storing and reusing solutions, DP saves computational time and enhances performance.

Next Steps

In the upcoming slide, we will delve into **Policy Evaluation** using dynamic programming techniques, particularly through the lens of the Bellman equation. This will illustrate how these foundational concepts are applied in practice.

Policy Evaluation - Introduction

- Policy evaluation is vital in dynamic programming.
- It measures the effectiveness of a policy in generating rewards.
- The process involves calculating the value function for the given policy.

Policy Evaluation - The Bellman Equation

Bellman Equation

The Bellman equation defines the value function $V^\pi(s)$ for a policy π:

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

- $V^\pi(s)$: Value of state s under policy π
- $\pi(a|s)$: Probability of action a in state s
- $p(s', r|s, a)$: Transition probability
- γ: Discount factor $(0 < \gamma < 1)$

Policy Evaluation - Procedure and Example

1. **Initialize**: Set $V^{\pi}(s)$ for all states to arbitrary values (often zero).
2. **Iterate**: Update the value function:

$$V^{\pi}_{\text{new}}(s) = \sum_{a \in A} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V^{\pi}(s')]$$

3. **Convergence Check**: Stop when changes are below a threshold.

Example

In a grid world with states $S = \{s_1, s_2\}$ and actions $A = \{\textit{left}, \textit{right}\}$, using a deterministic policy:

- Initialize $V^{\pi}(s_1) = 0, V^{\pi}(s_2) = 0$
- Update to find $V^{\pi}(s_1) = 1$ after several iterations.

Policy Improvement - Overview

Overview of Policy Improvement

Policy Improvement is a fundamental process in reinforcement learning that refines a given policy based on evaluations from the value function. By leveraging insights gained from a policy's performance, we systematically enhance the policy.

Understanding Policies and Value Functions

- **Policy (π)**: A strategy defining actions an agent should take in each state to maximize cumulative rewards.
  - Deterministic: Single action per state.
  - Stochastic: Probability distribution over actions.
- **Value Function (V)**: Represents the expected return from each state under policy π.

$$V(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \,\middle|\, s_0 = s \right] \tag{2}$$

The Process of Policy Improvement

1. **Begin with a Policy**: Start with an arbitrary policy π.
2. **Policy Evaluation**: Calculate the value function $V^{\pi}(s)$ using the Bellman Equation:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s'} P(s' \mid s, a) \Big[R(s, a, s') + \gamma V^{\pi}(s') \Big] \tag{3}$$

3. **Policy Improvement Step**:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} \sum_{s'} P(s' \mid s, a) \Big[R(s, a, s') + \gamma V(s') \Big] \tag{4}$$

4. **Repeat**: Iterate the process (Policy Evaluation followed by Policy Improvement) until convergence.

Example of Policy Improvement

Example

Consider a simple grid world where an agent moves in directions (up, down, left, right) to reach a goal cell while avoiding penalties.

1. Initial Policy: Assume the agent randomly chooses actions.
2. Evaluate: Calculate value functions based on the random policy.
3. Improve: Update the policy to maximize expected rewards.
4. Repeat: Continue until the agent consistently reaches the goal.

Key Points and Summary

- **Iteration**: Policy Improvement is cyclic (Policy Evaluation -> Policy Improvement).
- **Convergence**: Ensures that we converge to the optimal policy maximizing expected return.
- **Efficiency**: Effective in large state spaces where enumerating actions is impractical.

Summary

In reinforcement learning, Policy Improvement leverages the value function acquired through policy evaluation to refine the agent's strategy, ensuring optimal decision-making through iterative evaluation and improvement.

Value Iteration - Introduction

What is Value Iteration?

Value Iteration is a fundamental algorithm in reinforcement learning and dynamic programming designed to solve Markov Decision Processes (MDPs). It systematically combines policy evaluation and policy improvement in an iterative process.

Value Iteration - Key Concepts

- **Dynamic Programming**: A technique that breaks down complex problems into simpler subproblems, optimizing decisions under uncertainty.
- **Policy**: A strategy defining the action an agent should take in each state.
- **Value Function**: Estimates how good it is to be in a given state, reflecting the expected return when following a policy.

Value Iteration - Process Overview

1. **Initialization**: Start with an arbitrary value function, often set to zero for all states.
2. **Policy Evaluation**: For each state, compute the expected value based on possible actions, incorporating immediate and discounted future rewards.
3. **Policy Improvement**: Update the policy by selecting the actions that maximize the expected values from the value function.
4. **Convergence Check**: Repeat until the changes in value function fall below a predefined threshold.

Value Iteration - Example

Consider a simple grid world:

- **Initialization**: $V(s) = 0$ for all states.
- **Value Update**: Calculate new values:

$$V_{new}(s) = R(s) + \gamma \sum_{s'} P(s'|s, a)V(s') \tag{5}$$

- Where:
  - $R(s)$ = reward for state s.
  - γ = discount factor $(0 < \gamma < 1)$.
  - P = transition probability from state s to s' given action a.
- **Repeat**: Continue until the value function stabilizes.

Value Iteration - Key Points

- **Convergence**: Guarantees optimal value function in a finite state and action space.
- **Optimal Policy**: An optimal policy is derived by selecting actions that maximize the value in each state once convergence is achieved.
- **Applications**: Widely used in robotics, game AI, and operations research.

Value Iteration - Conclusion

Value Iteration provides an elegant solution to find optimal policies in uncertain environments through iterative refinement of the value function and policy. Understanding its structure lays the groundwork for tackling more complex problems in dynamic programming and enhances algorithmic thinking in reinforcement learning.

The Bellman Equation

What is the Bellman Equation?

The Bellman equation is a foundational concept in dynamic programming and reinforcement learning, establishing a recursive relationship between the value of a state and the values of its subsequent states. It is essential for algorithms that evaluate and optimize policies within a Markov Decision Process (MDP).

Mathematical Formulation

- The Bellman equation is applied in two contexts:
  - Policy Evaluation
  - Optimal Policy Evaluation

Policy Evaluation

For a given policy π, the Bellman equation is formulated as follows:

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')] \tag{6}$$

Optimal Policy Evaluation

The equation for optimal policy evaluation is given by:

$$V^{*}(s) = \max_{a \in A} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \gamma V^{*}(s')] \tag{7}$$

Key Points and Example

- **Recursive Nature**: Value of a state involves immediate rewards and the expected value of future states.
- **Policy Iteration**: Framework for evaluating and improving policies iteratively.
- **Optimality**: Focuses on maximizing expected value by choosing the best possible actions.

Example of Bellman Equation

Consider a simple MDP with states $S = \{A, B\}$ and actions $A = \{Go, Stay\}$:

- Transition Probabilities:
  - $P(B|A, Go) = 1$
  - $P(A|A, Stay) = 1$
- Immediate Rewards:
  - $R(A, Go, B) = 5$
  - $R(A, Stay, A) = 1$

Convergence of Dynamic Programming - Overview

Understanding Convergence

- **Definition of Convergence:** Refers to the process where an algorithm approaches a fixed point, resulting in no significant change in values or policies through iterations.

Convergence of Dynamic Programming - Conditions

Conditions for Convergence

To ensure convergence in dynamic programming, the following conditions should be met:

1. **Boundedness:**
   - State and action values remain finite.
   - Example: Rewards bounded in a specific range (e.g., [-10, 10]) lead to finite value functions.
2. **Discount Factor:**
   - γ must satisfy $0 \leq \gamma < 1$.
   - Promotes convergence by valuing immediate rewards more than future rewards.
3. **Monotonicity:**
   - Updates should bring the value function closer to the optimal solution.
   - Example: Bellman update ensures $V'(s) \geq V(s)$ for all states.
4. **Cauchy Condition:**
   - Successive approximations should diminish in difference over time, represented as $||V_{n+1} - V_n|| < \epsilon$ for large n.

Convergence of Dynamic Programming - Significance

Significance of Convergence in Reinforcement Learning

- **Stability of Policies:** Converged algorithms yield stable policies, fostering trust in systems (e.g., autonomous robots).
- **Performance Guarantees:** Convergence assures optimal or near-optimal policies essential for critical applications.
- **Efficient Learning:** Leads to faster refinement of strategies and reduces computational overhead.

Key Points to Remember

- Convergence is vital for reliable dynamic programming and reinforcement learning.
- Key conditions include boundedness, an appropriate discount factor, monotonicity, and Cauchy conditions.
- Successful convergence improves the effectiveness of RL systems in real-world scenarios.

Applications of Dynamic Programming

Overview

Dynamic programming (DP) is a powerful algorithmic technique that breaks problems into simpler subproblems. It is widely used in reinforcement learning (RL) for systematic problem-solving.

Key Applications of Dynamic Programming

1. **Robotics**
2. **Finance**

Dynamic Programming in Robotics

- **Path Planning:** DP is used to find the most efficient path for robots in a navigation environment.
- **Example:** A robot moving from point A to point B while avoiding obstacles calculates routes using DP to minimize costs.
- **Illustration:** In a 5x5 grid, DP evaluates the cost to reach each cell, recursively identifying the minimum cost path.

Dynamic Programming in Finance

- **Portfolio Optimization:** DP assists in maximizing returns through strategic investment reallocations over time.
- **Example:** An investor distributes wealth among assets, with DP evaluating optimal allocations considering changing market conditions.
- **Bellman Equation:**

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s'|s,a) V(s') \right) \tag{8}$$

Here, V is the value function, R is the reward, P is the transition probability, and γ is the discount factor.

Advantages of Dynamic Programming

- **Optimal Solutions:** Guarantees finding optimal solutions by exploring all states and actions.
- **Efficiency:** Uses memoization to avoid redundant calculations, improving performance in large state spaces.
- **Broad Applicability:** Techniques extend to fields such as operations research, economics, and bioinformatics.

Conclusion

Summary

Dynamic programming is essential for addressing complex challenges in reinforcement learning, especially in robotics and finance. It enables systematic exploration of options leading to optimal decision-making solutions.

Key Points to Remember

- DP simplifies problems into manageable subproblems.
- Applications in robotics include path planning and navigation.
- In finance, DP optimizes investment returns through strategic allocations.
- Understanding the Bellman equation is crucial for implementing DP effectively in RL contexts.

Implementing Dynamic Programming in Python

Introduction

Dynamic Programming (DP) is a powerful technique used for solving complex problems by breaking them down into simpler subproblems.
In reinforcement learning, it involves evaluating and improving policies to find the optimal solution.

Key Concepts of DP

1. **Policy Evaluation**:
   - Estimates the value of each state in a given policy.
   - Value Function:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, \pi\right] \tag{9}$$

2. **Policy Improvement**:
   - Updates the policy based on the value function.
   - Updated Policy:

$$\pi'(s) = \arg\max_a Q^{\pi}(s, a) \tag{10}$$

3. **Value Iteration**:
   - Computes the optimal policy iteratively.
   - Bellman Equation:

$$V_{new}(s) = \max_a \left(R(s, a) + \gamma \sum_{s'} P(s'|s, a) V(s') \right) \tag{11}$$

Python Implementations: Examples

Example 1: Policy Evaluation

```
import numpy as np

def policy_evaluation(policy, rewards, transitions,
    gamma=0.9, theta=1e-10):
     V = np.zeros(len(rewards))
     while True:
          delta = 0
          for s in range(len(rewards)):
               v = 0
               for a in range(len(policy[s])):
                    for s_next, prob in transitions[s][a].
                        items():
                         v += policy[s][a] * prob * (
                             rewards[s] + gamma * V[s_next])
               delta = max(delta, abs(v - V[s]))
               V[s] = v
          if delta < theta:
               break
```

Summary and Key Takeaways - Concept Overview

Key Concepts Covered:

1. **Dynamic Programming (DP) Fundamentals:**
   - A method for solving complex problems by breaking them down into simpler subproblems.
   - Key principles: *Optimal Substructure* and *Overlapping Subproblems*.
2. **Reinforcement Learning Context:**
   - DP techniques optimize decision-making processes.
   - Helps evaluate and improve policies in stochastic environments to maximize cumulative rewards.
3. **Core Algorithms:**
   - *Policy Evaluation*
   - *Policy Improvement*
   - *Value Iteration*

Summary and Key Takeaways - Examples

Examples:

- **Policy Evaluation Example:**
  - A grid world where an agent moves toward a goal.
  - Using a policy, we compute expected returns until the value function stabilizes.
- **Value Iteration Example:**
  - In a maze-like environment, we iteratively update each state's value based on available actions and their expected rewards until convergence.

Summary and Key Takeaways - Key Points and Conclusion

Key Points to Emphasize:

- Dynamic programming enhances reinforcement learning through efficient learning and decision-making strategies.
- Effective DP implementation reduces computation time for larger and complex problems in real-time.

Critical Formula:

$$V(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')] \tag{12}$$

Conclusion: Dynamic programming serves as a backbone for reinforcement learning algorithms, providing systematic approaches to evaluate and improve policies, which are vital in uncertain environments.