Reinforcement Learning

Dynamic Programming

- The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process (MDP)
- Classical DP algorithms are of limited utility in reinforcement learning because of
  - Their assumption of a perfect model
  - Their great computational expense

Cont…

- DP can be applied to problems with continuous state and action spaces
- An approximate solutions for tasks with continuous states and actions is to quantize the state and action spaces and then apply finite-state DP methods on a **finite MDP** environment
- That is, state, action and reward sets, S, A(s), and R, for $s \in S$, are finite
- That its dynamics are given by a set of probabilities

  $p(s', r|s, a)$, for all $s \in S$, $a \in A(s)$, $r \in R$, and $s' \in S+$

Policy Evaluation

- Policy evaluation is a fundamental concept in reinforcement learning and dynamic programming, where the aim is to determine the value function for a given policy
- The value function Vπ(s) for a policy ‘π’ is the expected cumulative reward that an agent can expect to receive while following a particular policy in a given environment, starting from state ‘s’ and represented as:

$$
\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s\right] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big],
\end{aligned}
$$

Cont…

$$V_\pi(s) := \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right]$$

- Where: s is the current state, s′ represents the next state, π(s) denotes the action chosen by the policy in state s,
- P(s′,r|s, a) is the probability of transitioning to state s′ given that action a is taken in state s,
- R(s,π(s),s′) is the immediate reward received upon transitioning from state ‘s’ to state ‘s’’ by taking action a for π(s),
- γ is the discount factor (0 ≤ γ < 1) which represents the importance of future rewards relative to immediate rewards

Process of Policy Evaluation

- The MDP represents the environment where the agent interacts
- Policy π selects actions based on the current state
- The value function Vπ(s) assigns a value to each state based on the expected cumulative reward under policy π
- Policy evaluation process:
  - Initialize value function arbitrarily
  - Iterate until convergence:
    - Update value function using Bellman Equation
    - Repeat until values stabilize
- Policy evaluation involves iteratively updating the value function until it converges to the true value function for the given policy
- This process allows the agent to understand the long-term consequences of following a particular policy in the environment.

Iterative Policy Evaluation Algorithm

- Initialize the value function V(s) arbitrarily for all states s
- Set a convergence threshold ϵ to determine when to stop iterating
- Repeat until convergence:
  - For each state s:
    - Initialize a variable Δ to track the maximum change in value for any state in this iteration (set $\Delta=0$)
    - Compute the new value $V'(s)$ using the Bellman expectation equation: $V'(s)=\sum a \in A \pi(a|s) \sum s',rp(s',r|s,a)[r+\gamma V(s')]$
    - Update the value function for state s: $V(s) \leftarrow V'(s)$
    - Update Δ with the maximum absolute change in value for any state: $\Delta \leftarrow \max(\Delta, |V'(s)-V(s)|)$
- Convergence Check:
- If $\Delta<\epsilon$, stop iterating as the value function has converged
- Otherwise, repeat the iterative update process.
- **Output:**
- Return the estimated value function V(s)

Policy Improvement

- Policy improvement is a critical step in reinforcement learning where the current policy is updated to select better actions
- The primary goal of policy improvement is to enhance the agent's performance by updating the current policy to select actions that lead to higher expected returns
- Policy improvement often occurs iteratively, where the current policy is evaluated, and then modifications are made to improve its performance based on the evaluation results
- This iterative process continues until a satisfactory policy is obtained
- Policy improvement balances exploration (trying out different actions to discover their outcomes) and exploitation (selecting actions that are currently believed to be the best)
- By updating the policy, the agent can gradually transition from exploration to exploitation as it learns more about the environment.

Cont…

- Policy improvement is essential in reinforcement learning because it allows the agent to adapt its behavior based on the observed rewards and states in the environment
- Without policy improvement, the agent would be unable to learn from its experiences and would likely continue to select suboptimal actions
- Policy evaluation can be achieved through various techniques such as dynamic programming, Monte Carlo methods, or Temporal Difference learning
- Policy improvement typically involves selecting actions to maximize the value function or directly improving the policy using methods like policy gradients.

Differences between Policy evaluation and Policy Improvement

- Policy evaluation focuses on assessing the quality of a given policy without altering it, while policy improvement focuses on modifying the policy to enhance its performance
- Policy evaluation aims to estimate the value function for a given policy, while policy improvement aims to identify actions that lead to higher expected returns and update the policy accordingly
- Policy evaluation can be achieved through various techniques such as dynamic programming, Monte Carlo methods, or Temporal Difference learning
- Policy improvement typically involves selecting actions to maximize the value function or directly improving the policy using methods like policy gradients
- The output of policy evaluation is an estimate of the value function, while the output of policy improvement is an updated policy

Iterative Algorithms

- There are two fundamental iterative algorithms used in dynamic programming to find optimal policies in (MDPs) aimed at finding the optimal policy
  - Policy Iteration
  - Value Iteration

Policy Iteration

- Policy iteration algorithm consists of two main steps: policy evaluation and policy improvement.
- In the policy evaluation step, the algorithm evaluates the current policy by computing the value function associated with that policy
- This is typically done using iterative methods like iterative policy evaluation
- In the policy improvement step, the algorithm updates the policy by selecting actions that maximize the expected return according to the current value function
- Policy iteration alternates between policy evaluation and policy improvement until convergence to the optimal policy is achieved
- Policy iteration is guaranteed to converge to the optimal policy, although it may take more iterations than value iteration
- However, each iteration of policy iteration can be less computationally expensive compared to value iteration, particularly when the number of states and actions is large

Cont…

- **Initialization:**
  - Start with an initial policy π, which can be arbitrary or based on some heuristic
  - Initialize the value function V(s) for all states s to arbitrary values (usually 0) or to some initial estimates.
- **Policy Evaluation:**
  - Given the current policy π, evaluate the value function V(s) associated with that policy
  - Iterate until convergence (usually determined by a small threshold or a maximum number of iterations):
    - For each state s, update the value function using the Bellman expectation equation:

$$V(s) \leftarrow \sum a \pi(a|s) \sum s', r p(s', r|s, a)[r + \gamma V(s')]$$

Where:

a is an action,

$\pi(a|s)\pi(a|s)$ is the probability of taking action a in state s according to the policy,

$p(s', r|s, a)$ is the transition probability to state s′ with reward r given action a in state s,

γ is the discount factor,

V(s′) is the value of the next state,

The inner summation is over all possible next states s′ and rewards r

Cont…

- **Policy Improvement:**
  - Once the value function has converged, improve the policy by selecting actions that maximize the expected return according to the current value function.
  - Update the policy by selecting the action with the highest value for each state: $\pi'(s)=\text{argmax}_a \sum_{s',r} p(s',r|s,a)[r+\gamma V(s')]$
  - If the policy has not changed ($\pi=\pi'\pi=\pi'$), then stop; otherwise, update the policy and return to step 2 (policy evaluation).
- **Convergence:**
  - Repeat steps 2 and 3 until the policy no longer changes, i.e., until convergence to the optimal policy is achieved.
  - Convergence is typically determined by checking if the policy has stabilized (i.e., $\pi=\pi'\pi=\pi'$).
- **Output:**
  - Once the optimal policy is found, return the optimal policy π^* and the corresponding optimal value function V^*.
- These steps are repeated iteratively until convergence to the optimal policy is achieved
- Policy iteration guarantees convergence to the optimal policy in a finite number of iterations for finite MDPs

Value Iteration

- Value iteration is an iterative algorithm that computes the optimal value function and policy simultaneously
- The algorithm iteratively updates the value function until it converges to the optimal value function
- At each iteration, it computes the value of each state by taking the maximum expected return over all possible actions
- Value iteration typically converges to the optimal value function and policy after a finite number of iterations
- It is computationally efficient and often converges faster than policy iteration

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

Repeat

$\Delta \leftarrow 0$

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

$v \leftarrow V(s)$

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

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

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

Output a deterministic policy, π, such that

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

Cont…

- **Initialization**: Initially, the algorithm initializes the value function arbitrarily for all states in the environment
- This value function represents the expected cumulative reward the agent can achieve from each state.
- **Value Iteration**: The algorithm iteratively updates the value function until it converges to the optimal values
- At each iteration, for each state in the environment, the algorithm computes the expected value of taking each possible action from that state and updates the value function accordingly
- The update rule is based on the Bellman equation, which expresses the relationship between the value of a state and the values of its successor states:
- $V(s)=\max_a(R(s,a)+\gamma\sum_{s'}P(s'|s,a)V(s'))$
- where:
  - V(s) is the value of state s,
  - R(s,a) is the immediate reward obtained by taking action a in state s,
  - γ is the discount factor that determines the importance of future rewards compared to immediate rewards,
  - P(s′|s,a) is the probability of transitioning to state s′ from state s by taking action a

Cont…

- **Convergence**: The algorithm repeats the value iteration process until the values of the states no longer change significantly between iterations, or until a predefined convergence criterion is met
- **Policy Extraction**: Once the value function has converged, the optimal policy can be extracted by selecting the action that maximizes the value function at each state
- Value iteration is guaranteed to converge to the optimal value function and policy for finite MDPs under certain conditions, such as having a finite number of states and actions and a bounded reward function
- However, it may not be computationally feasible for very large MDPs due to its iterative nature
- In such cases, approximate methods like Q-learning or deep reinforcement learning techniques are often used

Asynchronous dynamic programming (ADP)

- It is a class of reinforcement learning algorithms that update the value function or policy of a Markov decision process (MDP) in a non-synchronous, or asynchronous, manner
- Unlike synchronous methods, which update all states or state-action pairs simultaneously, asynchronous methods update states or state-action pairs individually and asynchronously

Cont…

- **Asynchronous Value Iteration (AVI)**: In AVI, instead of updating all states in each iteration, only a subset of states is updated. The selection of states to update can be random or based on some priority criteria.
- **Asynchronous Policy Iteration (API)**: In API, policy improvement and policy evaluation steps are not synchronized. Instead of updating the policy after a full policy evaluation, the policy is updated after each evaluation of a state.
- **Real-Time Dynamic Programming (RTDP)**: RTDP is a specific asynchronous algorithm designed for large state spaces. It focuses updates on states relevant to the current trajectory or goal.

Cont…

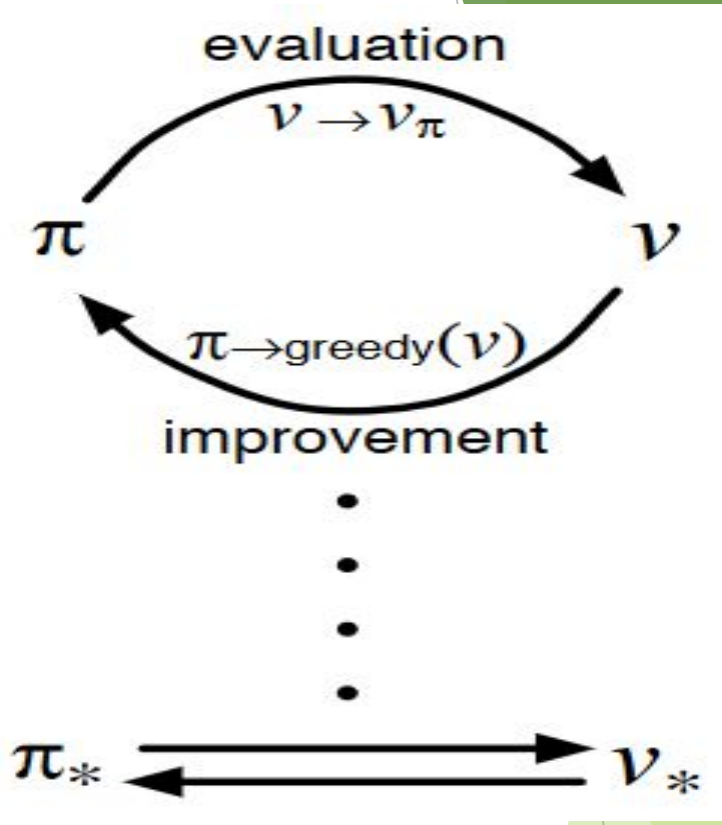
- ADP is useful in reinforcement learning for several reasons:
- **Efficiency**: ADP algorithms can be more efficient than synchronous methods, particularly in environments with large state or action spaces. By updating only a subset of states or state-action pairs, ADP reduces computational overhead
- **Online Learning**: ADP allows for online learning, where updates can be made as new data becomes available. This is especially beneficial in continuous or partially observable environments where immediate feedback is essential.
- **Exploration**: Asynchronous updates can facilitate exploration by allowing the agent to focus on states or actions that are more uncertain or have higher potential for learning.
- **Parallelization**: ADP algorithms are inherently parallelizable since updates to different states or state-action pairs can be performed independently. This makes them suitable for distributed computing environments and accelerates learning in parallel hardware architectures.
- **Adaptability**: ADP methods can adapt dynamically to changes in the environment or task requirements. By updating only relevant states or actions, they can quickly adjust to new information or goals.

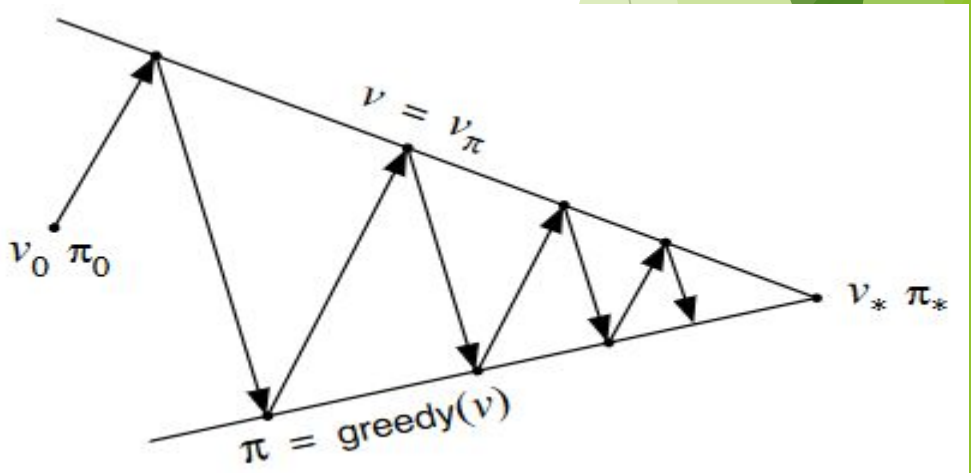
Generalized Policy Iteration

- Generalized Policy Iteration Policy iteration consists of two simultaneous, interacting processes:
  - 1. Making the value function consistent with the current policy (policy evaluation)
  - 2. Making the policy greedy with respect to the current value function (policy improvement)
- These two processes alternate, each completing before the other begins

Cont…

- Generalized Policy Iteration combines these two steps iteratively: policy evaluation and policy improvement.
- In each iteration, the agent evaluates the current policy to estimate its value function
- Then, based on the estimated value function, the agent improves its policy
- This process continues iteratively, with each iteration potentially leading to an improvement in the policy and value function
- The iteration stops when either the policy or the value function converges to an optimal or near-optimal solution





Cont…

- Generalized Policy Iteration provides a general framework for reinforcement learning algorithms to iteratively refine both the policy and the value function, ensuring that the agent learns to make better decisions in the environment
- By continuously evaluating and improving the policy, the agent can converge to an optimal or near-optimal solution over time, effectively solving the RL task at hand

Example

- Consider a 3 X 3 grid world problem where the goal is to reach either the top left corner or the bottom right corner. The agent can choose from four actions {up, down, left, right} which deterministically cause the corresponding state transitions, except that actions that would take the agent off the grid leave the state unchanged. Model this as an undiscounted, episodic task, where the reward is -1 for all transitions. The agent follows the equiprobable random policy. Calculate values after third iteration.