Exact Methods : Value Iteration

Easwar Subramanian

TCS Innovation Labs, Hyderabad

Email : easwar.subramanian@tcs.com / cs5500.2020@iith.ac.in

August 26, 2022

Overview

Review

Policy Iteration

Question : Is there a way to arrive at π_* starting from an arbitrary policy π ?

Answer : Policy Iteration

- Evaluate the policy π
  - ★ Compute $V^{\pi}(s) = \mathbb{E}_{\pi}(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s)$
- Improve the policy π

$$\pi'(s) = \text{greedy}(V^{\pi}(s))$$

$$\pi_0 \xrightarrow{\text{E}} V^{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V^{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} V^*,$$

Policy Iteration : Algorithm

Algorithm Policy Iteration

Start with an initial policy π_1
for $i = 1, 2, \cdots, N$ **do**
    Evaluate $V^{\pi_i}(s) \quad \forall s \in \mathcal{S}$. That is,
    for $k = 1, 2, \cdots, K$ **do**
        For all $s \in \mathcal{S}$ calculate

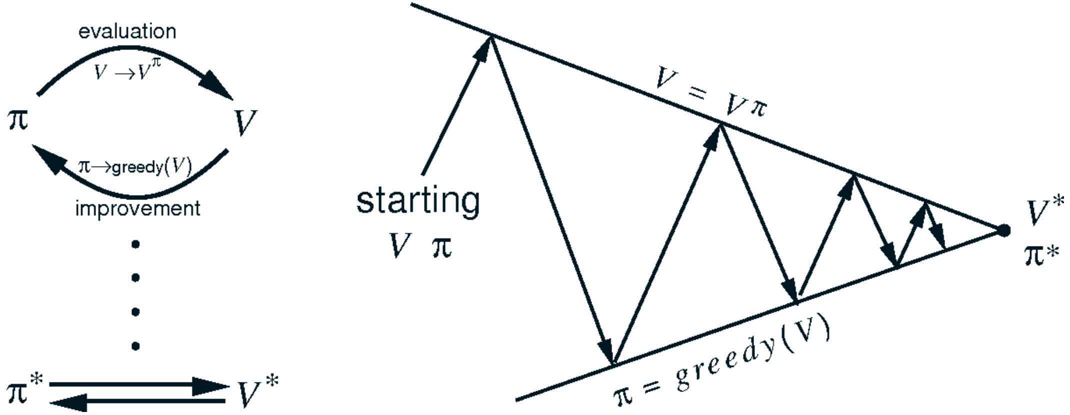
$$V^{\pi_i}_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^{\pi_i}_k(s')\right]$$

    end for
    Perform policy Improvement

$$\pi_{i+1} = \text{greedy}(V^{\pi_i})$$

end for

Policy Iteration : Schematic Representation



- The sequence $\{\pi_1, \pi_2, \cdots, \}$ is guaranteed to converge.
- At convergence, both current policy and the value function associated with the policy are optimal.

Figure Source: David Silver's UCL course

Value Iteration

Question : Is there a way to arrive at V_* starting from an arbitrary value function V_0 ?

Answer : Value Iteration

Optimality Equation for State Value Function

Recall the Bellman Evaluation Equation for an MDP with policy π

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

<u>**Question**</u> : Can we have a recursive formulation for $V_*(s)$?

$$V_*(s) = \max_{a} Q_*(s,a) = \max_{a} \left[\sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} \left(\mathcal{R}^{a}_{ss'} + \gamma V_*(s')\right) \right]$$

Optimality Equation for Action-Value Function

Similarly, there is a recursive formulation for $Q_*(\cdot, \cdot)$

$$Q_*(s, a) = \left[\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q_*(s', a')\right)\right]$$

<u>**Question**</u> : These are also a system of equations with $n = |\mathcal{S}|$ with n variables. Can we solve them ?

<u>**Answer**</u> : Optimality equations are non-linear system of equations with n unknowns and n non-linear constraints (i.e., the max operator).

Solving the Bellman Optimality Equation

- Bellman optimality equations are non-linear
- In general, there are no closed form solutions
- Iterative methods are typically used

Bellman's Optimality Principle

Principle of Optimality

The tail of an optimal policy must be optimal

- Any optimal policy can be subdivided into two components; an optimal first action, followed by an optimal policy from successor state s'.

Solution Methodology : Dynamic Programming

Bellman optimality equation :

$$V_*(s) = \max_a \left[\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V_*(s') \right) \right]$$

Optimal Substructure : Optimal solution can be constructed from optimal solutions to subproblems

Overlapping Subproblems : Problem can be broken down into subproblems and can be reused several times

- Markov Decision Processes, generally, satisfy both these characterstics
- Dynamic Programming is a popular solution method for problems having such properties

Value Iteration : Idea

- Suppose we know the value $V_*(s')$
- Then the solution $V_*(s)$ can be found by one step look ahead

$$V_*(s) \leftarrow \max_a \left[\sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma V_*(s') \right) \right]$$

- Idea of value iteration is to perform the above updates iteratively

Value Iteration : Algorithm

Algorithm Value Iteration

Start with an initial value function $V_0(\cdot)$;
for $k = 0, 1, 2, \cdots, K$ **do**
  for $s \in \mathcal{S}$ **do**
    Calculate

$$V_{k+1}(s) \leftarrow \max_a \left[\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma V_k(s') \right) \right]$$

  end for
end for

Value Iteration : Example

No noise and discount factor $\gamma = 1$

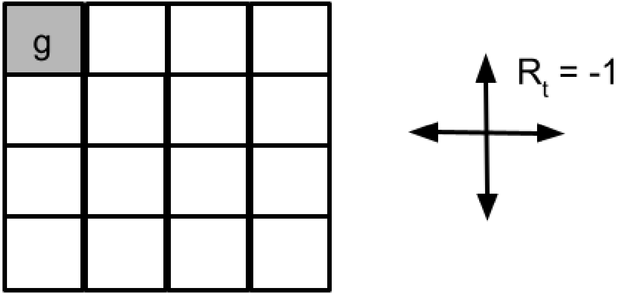


Figure Source: David Silver's UCL Course

Value Iteration : Example

$$V_{k+1}(s) \leftarrow \max_{a} \left[\sum_{s' \in \mathcal{S}} \mathcal{P}^{a}_{ss'} \left(\mathcal{R}^{a}_{ss'} + \gamma V_k(s') \right) \right]$$

g			

Problem

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

V_1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1

V_2

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-2
-2	-2	-2	-2

V_3

0	-1	-2	-3
-1	-2	-3	-3
-2	-3	-3	-3
-3	-3	-3	-3

V_4

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-4
-3	-4	-4	-4

V_5

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-5

V_6

0	-1	-2	-3
-1	-2	-3	-4
-2	-3	-4	-5
-3	-4	-5	-6

V_7

Value Iteration : Remarks

- The sequence of value functions $\{V_1, V_2, \cdots, \}$ converge
- It converges to V_*
- Convergence is independent of the choice of V_0.
- Intermediate value functions need not correspond to a policy in the sense of satisfying the Bellman Evaluation Equation
- However, for any k, one can come up with a greedy policy as follows

$$\pi_{k+1}(s) \leftarrow \text{greedy} V_k(s)$$

Optimality Equation for Action-Value Function

There is a recursive formulation for $Q_*(\cdot, \cdot)$

$$Q_*(s,a) = \left[\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} \left(\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q_*(s', a') \right) \right]$$

One could similarly conceive an iterative algorithm to compute optimal Q_* using the above recursive formulation !!

Value Iteration : Policy Evaluation

Iterative application of Bellman Evaluation Equation

Iterative update rule :

$$V_{k+1}^{\pi}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V_{k}^{\pi}(s')\right]$$

The sequence of value functions $\{V_1^{\pi}, V_2^{\pi}, \cdots, \}$ converge to V^{π}

Policy Iteration : Example Revisited

Update Rule :

$$V_{k+1}^{\pi_i}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s'} \mathcal{P}_{ss'}^{a} \left[\mathcal{R}_{ss'}^{a} + \gamma V_{k}^{\pi_i}(s') \right]$$

v_k for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

greedy policy w.r.t. v_k

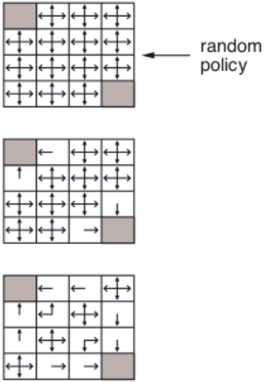


Figure Source: David Silver's UCL course

Policy Iteration : Example Revisited

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

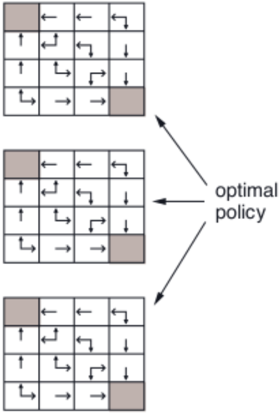


Figure Source: David Silver's UCL course

Modified Policy Iteration

Can we computationally simplify policy iteration process ?

- We need not wait for policy evaluation to converge to V^{π}
- We can have a stopping criterion like ϵ-convergence of value function evaluation or K iterations of policy evaluation
- Extreme case of $K = 1$ is **value iteration**. We update the policy every iteration

Prediction and Control using Dynamic Programming

- Dynamic Programming assumes full knowledge of MDP
- Used for both **prediction** and **control** in an MDP
- Prediction
  - Input MDP ($< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$) and policy π
  - Output : $V^{\pi}(\cdot)$
- Control
  - Input MDP ($< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$)
  - Output : Optimal value function $V_*(\cdot)$ or optimal policy π_*

Slide Credit: David Silver's UCL course

Synchronous Dynamic Programming

Problem	Bellman Equation	Algorithm
Prediction	Bellman Evaluation Equation	Policy Evaluation
Control	Bellman Evaluation Equation + Greedy Policy Improvement	Policy Iteration
Control	Bellman Optimality Equation	Value Iteration

- All the methods described above have synchronous backups
- All states are backed up in every iteration

Slide Credit: David Silver's UCL course

Possible Extensions

Asynchronous Dynamic Programming

- Updates to states are done individually, in any order
- For each selected state, apply the appropriate backup
- Can significantly reduce computation
- Convergence guarantees exist, if all states are selected sufficient number of times

Real Time Dynamic Programming

- Idea : update only states that are relevant to agent
- After each time step, we get s_t, a_t, r_{t+1}
- Perform the following update

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

Few Remarks

MDP and RL setting

- **MDP Setting** : The agent has knowledge of the state transition matrices $\mathcal{P}^a_{ss'}$ and the reward function $\mathcal{R}$
- **RL Setting** : The agent does not have knowledge of the state transition matrices $\mathcal{P}^a_{ss'}$ and the reward function $\mathcal{R}$
  - ★ The goal in both cases are same; Determine optimal sequence of actions such that the total discounted future reward is maximum.
  - ★ Although, this course would assume Markovian structure to state transitions, in many (sequential) decision making problems we may have to consider the history as well.

Concluding Remarks

- Recall that a (stochastic) policy is a distribution over actions given states
- Markov policy means that the policy depends only on the current state and not on the history
- Policies could be stationary or non-stationary
- In general, the optimal policy for an MDP need not be unique
- For finite horizon MDP, the optimal policy need not be even stationary
- For infinite horizon, an MDP admits an optimal policy that is deterministic and stationary. But there could other optimal policies that are stochastic and non-stationary.

Concluding Remarks

- The grid world problem is an example **stochastic shortest path** problem where we consider only policies that are 'proper'
  - ★ A policy that has a non-zero chance to finally reach the terminal state

  Under this assumption the theory on convergence will work out for even $\gamma = 1$.
- The total discounted return G_t could have infinite terms or $\gamma = 1$ but not both

On Value Iteration Convergence : Technical Questions

- How do we know that value iteration converges to V_* ?
- Or that iterative policy evaluation converges to V_π ?
- And therefore that policy iteration converges to π_* ?
- Is the solution unique ?
- How fast do these algorithms converge ? (Depends on discount factor γ)
- These questions were resolved by
  - ★ **Banach Fixed Point Theorem / Contraction Mapping Theorem**