Value Iteration Algorithm

and its extensions

Jan Mrkos

PUI Tutorial
Week 10

Outline

- Review of MDP concepts
- Value Iteration algorithm
- VI extensions

Value function of a policy

Look at the following definition of a value function of a policy for inifnite-horizon MDP. It contains multiple mistakes, correct them on a piece of paper:

Def: Value function of a policy for infinite-horizon MDP

Assume infinite horizon MDP with $\gamma \in [0, 100]$. Then let Value function of a policy π for every state $s \in S$ be defined as

$$V^{\pi}(s) = \sum_{s' \in S} R(s, \pi(s), s') R(s, \pi(s), s') + \gamma \pi(s')$$

Value function of a policy

Look at the following definition of a value function of a policy for inifnite-horizon MDP. It contains multiple mistakes, correct them on a piece of paper:

Def: Value function of a policy for infinite-horizon MDP

Assume infinite horizon MDP with $\gamma \in [0, 100]$. Then let Value function of a policy π for every state $s \in S$ be defined as

$$V^{\pi}(s) = \sum_{s' \in S} R(s, \pi(s), s')R(s, \pi(s), s') + \gamma\pi(s')$$

$$\gamma \in [0, 1)$$

$$V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

Value function of a policy

Look at the following definition of a value function of a policy for inifnite-horizon MDP. It contains multiple mistakes, correct them on a piece of paper:

Def: Value function of a policy for infinite-horizon MDP

Assume infinite horizon MDP with $\gamma \in [0, 100]$. Then let Value function of a policy π for every state $s \in S$ be defined as

$$V^{\pi}(s) = \sum_{s' \in S} R(s, \pi(s), s') R(s, \pi(s), s') + \gamma \pi(s')$$

$$\gamma \in [0, 1)$$

$$V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^{\pi}(s')]$$

Question: Difference to def. of an optimal value function?

Solving MDPs

From last time:
Optimal value function $V*$ in *acyclic* MDP can be found by:

Solving MDPs

From last time:
Optimal value function $V*$ in *acyclic* MDP can be found by:

- substituting values in order from terminal states to the initial state.

Solving MDPs

From last time:
Optimal value function $V*$ in *acyclic* MDP can be found by:

- substituting values in order from terminal states to the initial state.

Value function of a policy V^{π} in *ANY* MDP can be found by:

Solving MDPs

From last time:
Optimal value function $V*$ in *acyclic* MDP can be found by:

- substituting values in order from terminal states to the initial state.

Value function of a policy V^{π} in *ANY* MDP can be found by:

- solving a system of linear equations.

Solving MDPs

From last time:

Optimal value function $V*$ in *acyclic* MDP can be found by:

- substituting values in order from terminal states to the initial state.

Value function of a policy V^{π} in *ANY* MDP can be found by:

- solving a system of linear equations.

Optimal value function $V*$ in *ANY* MDP can be found by:

Solving MDPs

From last time:
Optimal value function $V*$ in *acyclic* MDP can be found by:

- substituting values in order from terminal states to the initial state.

Value function of a policy V^{π} in *ANY* MDP can be found by:

- solving a system of linear equations.

Optimal value function $V*$ in *ANY* MDP can be found by:

- solving a system of non-linear equations (max)?

Hard stuff to do analytically $\rightarrow$ iterative methods.

Value Iteration algorithm

Basic algorithm for finding solution of Bellman Equations iteratively:

Algorithm: Value Iteration

$n \leftarrow 0$;

$\forall s,\ V_0(s) \leftarrow 0$; `// arbitrarily chosen initialization`

Value Iteration algorithm

Basic algorithm for finding solution of Bellman Equations iteratively:

Algorithm: Value Iteration

$n \leftarrow 0$;
$\forall s,\ V_0(s) \leftarrow 0$; `// arbitrarily chosen initialization`
while *running* **do**

Value Iteration algorithm

Basic algorithm for finding solution of Bellman Equations iteratively:

Algorithm: Value Iteration

$n \leftarrow 0$;
$\forall s,\ V_0(s) \leftarrow 0$; // arbitrarily chosen initialization
while *running* **do**
  Set $n = n + 1$;
  foreach *state* $s \in S$ **do**

Value Iteration algorithm

Basic algorithm for finding solution of Bellman Equations iteratively:

Algorithm: Value Iteration

$n \leftarrow 0$;

$\forall s,\ V_0(s) \leftarrow 0$; `// arbitrarily chosen initialization`

while *running* **do**

  Set $n = n + 1$;

  foreach *state* $s \in S$ **do**

    $V_n(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_{n-1}(s')]$; `// Bellman backup`

return V_n and greedy policy π^{V_n}

Value Iteration algorithm

Basic algorithm for finding solution of Bellman Equations iteratively:

Algorithm: Value Iteration

$n \leftarrow 0$;
$\forall s,\ V_0(s) \leftarrow 0$; `// arbitrarily chosen initialization`
while *running* **do**
  Set $n = n + 1$;
  foreach *state* $s \in S$ **do**
    $V_n(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V_{n-1}(s')]$; `// Bellman backup`
return V_n and greedy policy π^{V_n}

Basic version alternates 2 arrays to store state values.

Value Iteration algorithm

Basic algorithm for finding solution of Bellman Equations iteratively:

Algorithm: Value Iteration

$n \leftarrow 0$;

$\forall s,\ V_0(s) \leftarrow 0$; `// arbitrarily chosen initialization`

while *running* **do**

  Set $n = n + 1$;

  foreach *state* $s \in S$ **do**

    $V_n(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_{n-1}(s')]$; `// Bellman backup`

return V_n and greedy policy π^{V_n}

Basic version alternates 2 arrays to store state values.

Q: What is *greedy policy*?

Value Iteration algorithm

Basic algorithm for finding solution of Bellman Equations iteratively:

Algorithm: Value Iteration

$n \leftarrow 0$;
$\forall s,\ V_0(s) \leftarrow 0$; `// arbitrarily chosen initialization`
while *running* **do**
  Set $n = n + 1$;
  foreach *state* $s \in S$ **do**
    $V_n(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V_{n-1}(s')]$; `// Bellman backup`
return V_n and greedy policy π^{V_n}

Basic version alternates 2 arrays to store state values.

Q: What is *greedy policy*?
A: $\pi^{V_n}(s) = \arg\max_{a \in A} Q_n(s,a)$

Value Iteration algorithm

Basic algorithm for finding solution of Bellman Equations iteratively:

Algorithm: Value Iteration

$n \leftarrow 0$;
$\forall s,\ V_0(s) \leftarrow 0$; `// arbitrarily chosen initialization`
while *running* **do**
  Set $n = n + 1$;
  foreach *state* $s \in S$ **do**
    $V_n(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_{n-1}(s')]$; `// Bellman backup`
return V_n and greedy policy π^{V_n}

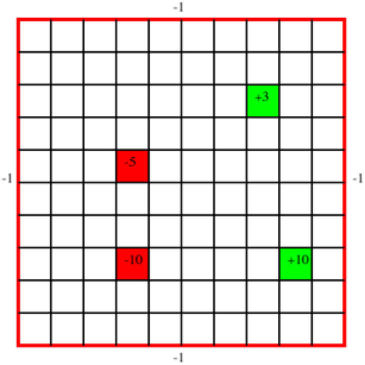
Basic version alternates 2 arrays to store state values.

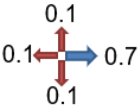
Q: What is *greedy policy*?
A: $\pi^{V_n}(s) = \arg\max_{a \in A} Q_n(s, a) = \arg\max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_{n-1}(s')]$.

Example - gridworld

Gridworld domain. Assuming $\gamma = 1$.
Using VI and initializing $\forall s \; V_0(s) = 0$, calculate V_1, V_2, V_3 for the nine states around the +10 tile.
Domain rules:

- Moving into edges gives -1 reward
- Moving onto marked tiles gives corresponding reward





Example from https://artint.info

Value Iteration algorithm

Basic algorithm for finding solution of Bellman Equations iteratively.

Algorithm: Value Iteration

$n \leftarrow 0$;
$\forall s,\ V_0(s) \leftarrow 0$; `// arbitrarily chosen initialization`
while *running* **do**
  Set $n = n + 1$;
  foreach *state* $s \in S$ **do**
    $V_n(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_{n-1}(s')]$; `// Bellman backup`
return V_n and greedy policy π^{V_n}

Question: Does it converge? How fast? When do we stop?

Residual - definition

Def: Residual

Residual of value function V_n from V_{n+1} at state $s \in S$ is defined by:

$$Res^{V_n}(s) = |V_n(s) - V_{n+1}(s)|$$

Residual - definition

Def: Residual

Residual of value function V_n *from* V_{n+1} *at state* $s \in S$ is defined by:

$$Res^{V_n}(s) = |V_n(s) - V_{n+1}(s)|$$

Residual of value function V *from* V' is given by:

$$Res^{V_n} = ||V_n - V_{n+1}||_\infty = \max_s |V_n(s) - V_{n+1}(s)|$$

VI stopping criterion[1]

Stopping criterion: When residual of consecutive value functions is below low value of ϵ:

$$||V_n - V_{n+1}|| < \epsilon$$

However, this does not imply ϵ distance of value of greedy policy from optimal value function.

[1] 👁 for proof, see: [Singh, Yee: An Upper Bound on the Loss from Approximate Optimal-Value Functions, p.229]

VI stopping criterion[1]

Stopping criterion: When residual of consecutive value functions is below low value of ϵ:

$$||V_n - V_{n+1}|| < \epsilon$$

However, this does not imply ϵ distance of value of greedy policy from optimal value function. Theorems for general MDP exist of form:

$$V_n, V^* \text{as above} \implies \forall s \; |V_n(s) - V^*(s)| < \epsilon \times (\text{Some MDP dependent term})$$

[1] 👁 for proof, see: [Singh, Yee: An Upper Bound on the Loss from Approximate Optimal-Value Functions, p.229]

VI stopping criterion[1]

Stopping criterion: When residual of consecutive value functions is below low value of ϵ:

$$||V_n - V_{n+1}|| < \epsilon$$

However, this does not imply ϵ distance of value of greedy policy from optimal value function. Theorems for general MDP exist of form:

$$V_n, V^* \text{as above} \implies \forall s \; |V_n(s) - V^*(s)| < \epsilon \times (\text{Some MDP dependent term})$$

In case of discounted $(\gamma < 1)$ infinite-horizon MDPs:

$$V_n, V^* \text{as above} \implies \forall s \; |V_n(s) - V^*(s)| < 2\frac{\epsilon\gamma}{1-\gamma}$$

[1] 👁 for proof, see: [Singh, Yee: An Upper Bound on the Loss from Approximate Optimal-Value Functions, p.229]

VI with stopping criterion

Algorithm: Value Iteration with epsilon stop

```
n ← 0;
∀s, V_0(s) ← 0;                                   // arbitrarily chosen initialization
while Res^V > ε do
    Set n = n + 1;
    foreach state s ∈ S do
        V_n(s) ← max_{a∈A} Σ_{s'∈S} T(s, a, s')[R(s, a, s') + γ V_{n−1}(s')];
        Res^V(s) ← |V_n(s) − V_{n−1}(s)|;             // Update residual
return V_n and greedy policy π^{V_n}
```

Where:

$n \leftarrow 0;$

$\forall s,\ V_0(s) \leftarrow 0;$

while $Res^V > \epsilon$ **do**

Set $n = n + 1;$

foreach *state* $s \in S$ **do**

$V_n(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V_{n-1}(s')];$

$Res^V(s) \leftarrow |V_n(s) - V_{n-1}(s)|;$

return V_n and greedy policy π^{V_n}

Value iteration properties

- Convergence: VI converges from any initialization (unlike PI)
- Termination: when residual is ”small”

Value iteration properties

- Convergence: VI converges from any initialization (unlike PI)
- Termination: when residual is "small"

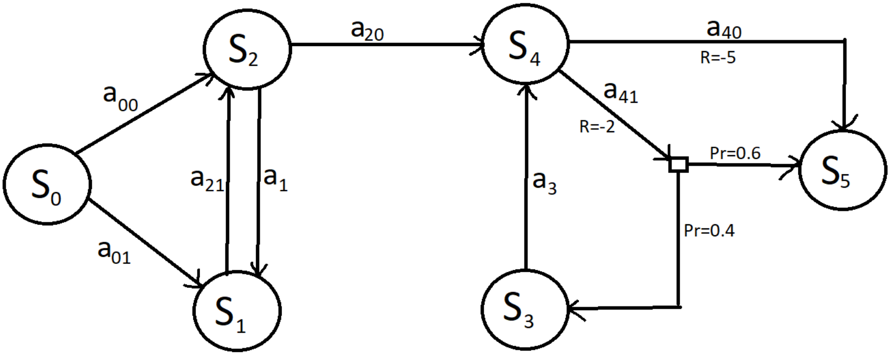
Q: What are the memory requirements of VI?

Value iteration properties

- Convergence: VI converges from any initialization (unlike PI)
- Termination: when residual is "small"

Q: What are the memory requirements of VI?
A: Value of each state needs to be stored twice, residual can be calculated on the fly.

MDP example



- All undeclared rewards are -1

Task: Initialize VI with negative distance to S_5 and calculate first 3 iterations of VI, with state ordering S_0 to S_5

Gauss-Seidel (Asynchronous) VI

Algorithm: Gauss-Seidel VI

$n \leftarrow 0$;
$\forall s,\ V_0(s) \leftarrow 0$;
while $Res^V > \epsilon$ **do**
  Set $n = n + 1$;
  foreach *state* $s \in S$ **do**
    $V_n(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_n(s')]$;
    $Res^V(s) \leftarrow |V_n(s) - V_{n-1}(s)|$;
return V_n and greedy policy π^{V_n}

Gauss-Seidel (Asynchronous) VI

Algorithm: Gauss-Seidel VI

$n \leftarrow 0$;
$\forall s,\ V_0(s) \leftarrow 0$;
while $Res^V > \epsilon$ **do**
    Set $n = n + 1$;
    foreach *state* $s \in S$ **do**
        $V_n(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_n(s')]$;
        $Res^V(s) \leftarrow |V_n(s) - V_{n-1}(s)|$;
return V_n and greedy policy π^{V_n}

Q: Memory requirements compared to VI?

Gauss-Seidel (Asynchronous) VI

Algorithm: Gauss-Seidel VI

1 $n \leftarrow 0$;
2 $\forall s,\ V_0(s) \leftarrow 0$;
3 **while** $Res^V > \epsilon$ **do**
4   Set $n = n + 1$;
5   **foreach** *state* $s \in S$ **do**
6         $V_n(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V_n(s')]$;
7         $Res^V(s) \leftarrow |V_n(s) - V_{n-1}(s)|$;
8 **return** V_n and greedy policy π^{V_n}

Q: Memory requirements compared to VI?

Q: Is order of states in line 5 important?

Asyncchronous VI

Algorithm: Asynchronous VI

$\forall s,\ V(s) \leftarrow 0$;
while $Res^V > \epsilon$ **do**
  $s \leftarrow \text{select } s \in S$;
  $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V(s')]$;
  $Res^V(s) \leftarrow |V_{\text{old}}(s) - V_{\text{new}}(s)|$;
return V and greedy policy π^V

Asyncchronous VI

Algorithm: Asynchronous VI

$\forall s,\ V(s) \leftarrow 0;$
while $Res^V > \epsilon$ **do**
  $s \leftarrow$ select $s \in S$;
  $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V(s')];$
  $Res^V(s) \leftarrow |V_{\text{old}}(s) - V_{\text{new}}(s)|;$
return V and greedy policy π^V

Q: Convergence condition?

Asyncchronous VI

Algorithm: Asynchronous VI

$\forall s,\ V(s) \leftarrow 0$;
while $Res^V > \epsilon$ **do**
    $s \leftarrow \text{select } s \in S$;
    $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V(s')]$;
    $Res^V(s) \leftarrow |V_{\text{old}}(s) - V_{\text{new}}(s)|$;
return V and greedy policy π^V

Q: Convergence condition? A: Asymptotic as VI if every state visited ∞ often.

Asyncchronous VI

Algorithm: Asynchronous VI

1 $\forall s,\ V(s) \leftarrow 0$;
2 **while** $Res^V > \epsilon$ **do**
3 $\quad s \leftarrow \text{select } s \in S$;
4 $\quad V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma V(s')]$;
5 $\quad Res^V(s) \leftarrow |V_{\text{old}}(s) - V_{\text{new}}(s)|$;
6 **return** V and greedy policy π^V

Q: Convergence condition? A: Asymptotic as VI if every state visited ∞ often.

Q: How to pick s on line 5?

Asyncchronous VI

Algorithm: Asynchronous VI

1 $\forall s,\ V(s) \leftarrow 0$;
2 **while** $Res^V > \epsilon$ **do**
3 $\quad s \leftarrow \text{select } s \in S$;
4 $\quad V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')]$;
5 $\quad Res^V(s) \leftarrow |V_{\text{old}}(s) - V_{\text{new}}(s)|$;
6 **return** V and greedy policy π^V

Q: Convergence condition? A: Asymptotic as VI if every state visited ∞ often.
Q: How to pick s on line 5? A: Simplest is *Gauss-Seidel VI*, that is run AVI over all states iteratively.

Prioritized VI

Q: How else can we pick states to update? (A: by ordering them in clever ways)

Prioritized VI

Q: How else can we pick states to update? (A: by ordering them in clever ways)

E.g.: build priority queue of states to update $\rightarrow$ *Prioritized Sweeping VI*. Update states in the order of the queue. Priority function:

$$\text{priority}_{PS}(s) \leftarrow \max\{\text{priority}_{PS}(s), \max_{a \in A}\{T(s, a, s')Res^V(s')\}\}$$

Prioritized VI

Q: How else can we pick states to update? (A: by ordering them in clever ways)

E.g.: build priority queue of states to update $\rightarrow$ *Prioritized Sweeping VI*. Update states in the order of the queue. Priority function:

$$\text{priority}_{PS}(s) \leftarrow \max\{\text{priority}_{PS}(s), \max_{a \in A}\{T(s,a,s')Res^V(s')\}\}$$

EXAMPLE ON BOARD

Prioritized VI

Q: How else can we pick states to update? (A: by ordering them in clever ways)

E.g.: build priority queue of states to update → *Prioritized Sweeping VI*. Update states in the order of the queue. Priority function:

$$\text{priority}_{PS}(s) \leftarrow \max\{\text{priority}_{PS}(s), \max_{a \in A}\{T(s,a,s')Res^V(s')\}\}$$

EXAMPLE ON BOARD

Convergence?

Prioritized VI

Q: How else can we pick states to update? (A: by ordering them in clever ways)

E.g.: build priority queue of states to update → *Prioritized Sweeping VI*. Update states in the order of the queue. Priority function:

$$\text{priority}_{PS}(s) \leftarrow \max\{\text{priority}_{PS}(s), \max_{a \in A}\{T(s, a, s')Res^V(s')\}\}$$

EXAMPLE ON BOARD

Convergence?

- If all states start with non-zero priority

Prioritized VI

Q: How else can we pick states to update? (A: by ordering them in clever ways)

E.g.: build priority queue of states to update $\rightarrow$ *Prioritized Sweeping VI*. Update states in the order of the queue. Priority function:

$$\text{priority}_{PS}(s) \leftarrow \max\{\text{priority}_{PS}(s), \max_{a \in A}\{T(s,a,s')Res^V(s')\}\}$$

EXAMPLE ON BOARD

Convergence?

- If all states start with non-zero priority
- OR If you interleave regular VI sweeps with Prioritized VI

Partitioned VI

Subdivide state space into partitions. Pre-solve parts independently.

- Example is *Topological VI*

Partitioned VI

Subdivide state space into partitions. Pre-solve parts independently.

- Example is *Topological VI*

Topological VI:

1. Find acyclic paritioning (by finding strongly connected components in the graph)
2. Run VI in each partition to convergence backward from terminal states

Partitioned VI

Subdivide state space into partitions. Pre-solve parts independently.

- Example is *Topological VI*

Topological VI:

1. Find acyclic paritioning (by finding strongly connected components in the graph)
2. Run VI in each partition to convergence backward from terminal states

---

Q: Why acyclic partitioning?

Partitioned VI

Subdivide state space into partitions. Pre-solve parts independently.

- Example is *Topological VI*

Topological VI:

1. Find acyclic paritioning (by finding strongly connected components in the graph)
2. Run VI in each partition to convergence backward from terminal states

---

Q: Why acyclic partitioning?

EXAMPLE ON BOARD

Thank you for participating in the tutorials :-)

Please fill out the feedback form →

https://forms.gle/JxJBaGeLgwxVbKiNA