Value Iteration on a 4×4 Grid World: Step-by-Step Example

Environment Setup

Let's define a 4×4 grid world with the following properties:

- **States**: 16 cells labeled (0,0) to (3,3)
- **Actions**: Up (↑), Right (→), Down (↓), Left (←)
- **Rewards**:
  - +1 at state (3,3) (goal state)
  - -1 at state (1,1) (penalty state)
  - 0 everywhere else
- **Terminal states**: (3,3)
- **Transitions**: Deterministic (actions always succeed)
- **Discount factor** γ = 0.9
- **Convergence threshold** θ = 0.01

Here's our grid world:

Copy

```
+---+---+---+---+
|(0,0)|(0,1)|(0,2)|(0,3)|
| 0  | 0  | 0  | 0  |
+---+---+---+---+
|(1,0)|(1,1)|(1,2)|(1,3)|
| 0  |-1  | 0  | 0  |
+---+---+---+---+
|(2,0)|(2,1)|(2,2)|(2,3)|
| 0  | 0  | 0  | 0  |
+---+---+---+---+
|(3,0)|(3,1)|(3,2)|(3,3)|
| 0  | 0  | 0  |+1  |
+---+---+---+---+
```

Value Iteration Algorithm

Initialization

Initialize the value function V(s) to 0 for all non-terminal states, and V(terminal) = 0 (which in our case is already 0):

Initial Value Function:

	Col 0	Col 1	Col 2	Col 3
Row 0	0.00	0.00	0.00	0.00
Row 1	0.00	0.00	0.00	0.00
Row 2	0.00	0.00	0.00	0.00
Row 3	0.00	0.00	0.00	0.00

Value Iteration Steps

Iteration 1

For each state s, compute:
V(s) ← max_a [Σ_s' p(s'|s,a)[r + γV(s')]]

In our deterministic environment, this simplifies to:
V(s) ← max_a [r(s,a) + γV(s')]

For state (0,0):

- Going Right: 0 + 0.9×V(0,1) = 0 + 0.9×0 = 0
- Going Down: 0 + 0.9×V(1,0) = 0 + 0.9×0 = 0
- Going Left/Up: Invalid (edge of grid) So V(0,0) = 0

For state (1,1):

- Going Right: -1 + 0.9×V(1,2) = -1 + 0.9×0 = -1
- Going Down: -1 + 0.9×V(2,1) = -1 + 0.9×0 = -1
- Going Left: -1 + 0.9×V(1,0) = -1 + 0.9×0 = -1
- Going Up: -1 + 0.9×V(0,1) = -1 + 0.9×0 = -1 So V(1,1) = -1

For state (3,2):

- Going Right: 0 + 0.9×V(3,3) = 0 + 0.9×0 = 0
- Going Left/Down/Up: 0 + 0.9×0 = 0 So V(3,2) = 0

Computing for all states:

Value Function (Iteration 1):

	Col 0	Col 1	Col 2	Col 3
Row 0	0.00	0.00	0.00	0.00
Row 1	0.00	-1.00	0.00	0.00
Row 2	0.00	0.00	0.00	0.00
Row 3	0.00	0.00	0.00	1.00

Maximum change in value: Δ = 1.00 > θ, so we continue.

Iteration 2

For state (0,0):

- Going Right: 0 + 0.9×V(0,1) = 0 + 0.9×0 = 0
- Going Down: 0 + 0.9×V(1,0) = 0 + 0.9×0 = 0 So V(0,0) = 0

For state (3,2):

- Going Right: 0 + 0.9×V(3,3) = 0 + 0.9×1 = 0.9
- Going Left/Down/Up: 0 + 0.9×0 = 0 So V(3,2) = 0.9

For state (2,3):

- Going Right: 0 + 0.9×V(3,3) = 0 + 0.9×1 = 0.9
- Going Down: 0 + 0.9×V(3,3) = 0 + 0.9×1 = 0.9
- Going Left/Up: 0 + 0.9×0 = 0 So V(2,3) = 0.9

Computing for all states:

Value Function (Iteration 2):

	Col 0	**Col 1**	**Col 2**	**Col 3**
Row 0	0.00	0.00	0.00	0.00
Row 1	0.00	-1.00	0.00	0.00
Row 2	0.00	0.00	0.00	0.90
Row 3	0.00	0.00	0.90	1.00

Maximum change in value: Δ = 0.90 > θ, so we continue.

Iteration 3

For state (1,2):

- Going Right: 0 + 0.9×V(1,3) = 0 + 0.9×0 = 0
- Going Down: 0 + 0.9×V(2,2) = 0 + 0.9×0 = 0
- Going Left: 0 + 0.9×V(1,1) = 0 + 0.9×(-1) = -0.9
- Going Up: 0 + 0.9×V(0,2) = 0 + 0.9×0 = 0 So V(1,2) = 0

For state (2,2):

- Going Right: 0 + 0.9×V(2,3) = 0 + 0.9×0.9 = 0.81
- Going Down: 0 + 0.9×V(3,2) = 0 + 0.9×0.9 = 0.81
- Going Left/Up: 0 + 0.9×0 = 0 So V(2,2) = 0.81

For state (3,1):

- Going Right: 0 + 0.9×V(3,2) = 0 + 0.9×0.9 = 0.81
- Going Left/Down/Up: 0 + 0.9×0 = 0 So V(3,1) = 0.81

Computing for all states:

Value Function (Iteration 3):

	Col 0	Col 1	Col 2	Col 3
Row 0	0.00	0.00	0.00	0.00
Row 1	0.00	-1.00	0.00	0.00
Row 2	0.00	0.00	0.81	0.90
Row 3	0.00	0.81	0.90	1.00

Maximum change in value: Δ = 0.81 > θ, so we continue.

Iteration 4

For state (1,3):

- Going Right: 0 + 0.9×V(1,3) = 0 + 0.9×0 = 0
- Going Down: 0 + 0.9×V(2,3) = 0 + 0.9×0.9 = 0.81
- Going Left: 0 + 0.9×V(1,2) = 0 + 0.9×0 = 0
- Going Up: 0 + 0.9×V(0,3) = 0 + 0.9×0 = 0 So V(1,3) = 0.81

For state (2,1):

- Going Right: 0 + 0.9×V(2,2) = 0 + 0.9×0.81 = 0.729
- Going Down: 0 + 0.9×V(3,1) = 0 + 0.9×0.81 = 0.729
- Going Left/Up: 0 + 0.9×0 = 0 So V(2,1) = 0.729

Computing for all states:

Value Function (Iteration 4):

	Col 0	Col 1	Col 2	Col 3
Row 0	0.00	0.00	0.00	0.73
Row 1	0.00	-1.00	0.00	0.81
Row 2	0.00	0.73	0.81	0.90
Row 3	0.00	0.81	0.90	1.00

Maximum change in value: Δ = 0.81 > θ, so we continue.

Iteration 5

Computing updated values for all states:

Value Function (Iteration 5):

	Col 0	Col 1	Col 2	Col 3
Row 0	0.00	0.00	0.66	0.73
Row 1	0.00	-1.00	0.73	0.81
Row 2	0.66	0.73	0.81	0.90
Row 3	0.73	0.81	0.90	1.00

Maximum change in value: $\Delta = 0.66 > \theta$, so we continue.

Let's continue for a few more iterations until we reach convergence.

Iteration 10 (after continuing)

Value Function (Iteration 10):

	Col 0	Col 1	Col 2	Col 3
Row 0	0.59	0.66	0.73	0.81
Row 1	0.66	-1.00	0.81	0.90
Row 2	0.73	0.81	0.90	0.90
Row 3	0.81	0.90	0.90	1.00

Maximum change in value: $\Delta = 0.003 < \theta$, so we stop.

Final Optimal Policy

After value iteration has converged, we compute the optimal deterministic policy:

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

For state (0,0):

- Going Right: $0 + 0.9 \times V(0,1) = 0 + 0.9 \times 0.66 = 0.594$
- Going Down: $0 + 0.9 \times V(1,0) = 0 + 0.9 \times 0.66 = 0.594$
- Going Left/Up: Invalid (edge of grid)

We get a tie. By convention, let's choose Right when there's a tie.

Computing for all states:

Optimal Policy:

	Col 0	Col 1	Col 2	Col 3
Row 0	→	→	→	↓
Row 1	↑	↑	→	↓
Row 2	→	→	→	↓
Row 3	→	→	→	*

The optimal policy successfully navigates around the penalty state at (1,1) and guides the agent to the goal state at (3,3).

Comparison to Policy Iteration

Value iteration combines aspects of policy evaluation and policy improvement into a single update. In each iteration, it performs a "partial" policy evaluation (just one sweep) followed immediately by policy improvement.

The main differences from policy iteration:

1. Value iteration doesn't explicitly represent a policy during iterations
2. Value iteration doesn't wait for the value function to converge before improving the policy
3. Value iteration might converge in fewer iterations for some problems

Python Implementation

Here's Python code to implement value iteration on the 4×4 grid world:

python

Copy

```
import numpy as np

# Grid world parameters
rows, cols = 4, 4
gamma = 0.9  # Discount factor
theta = 0.01  # Convergence threshold

# Define rewards
rewards = np.zeros((rows, cols))
rewards[1, 1] = -1  # Penalty state
rewards[3, 3] = 1   # Goal state

# Define terminal states
terminal = np.zeros((rows, cols), dtype=bool)
terminal[3, 3] = True  # Goal is terminal

# Actions: 0=Up, 1=Right, 2=Down, 3=Left
actions = ["↑", "→", "↓", "←"]
dr = [-1, 0, 1, 0]
dc = [0, 1, 0, -1]

def value_iteration():
    # Initialize value function
    V = np.zeros((rows, cols))

    iteration = 0
    max_iterations = 20  # Limit iterations for demonstration

    while True:
        iteration += 1
        delta = 0

        # Value update
        for r in range(rows):
            for c in range(cols):
                if terminal[r, c]:
                    V[r, c] = rewards[r, c]
                    continue

                v = V[r, c]

                # Calculate max action value
                action_values = []
                for a in range(4):  # For each action
                    nr = r + dr[a]
                    nc = c + dc[a]
```

```
                    if 0 <= nr < rows and 0 <= nc < cols:
                        action_values.append(rewards[r, c] + gamma * V[nr, nc])
                    else:
                        # Invalid action (off grid) - stay in place
                        action_values.append(rewards[r, c] + gamma * V[r, c])

                V[r, c] = max(action_values)
                delta = max(delta, abs(v - V[r, c]))

        print(f"Iteration {iteration}:")
        print_table(V)
        print(f"Delta: {delta:.4f}\n")

        # Check for convergence
        if delta < theta or iteration >= max_iterations:
            break

    # Extract the optimal policy
    policy = np.zeros((rows, cols), dtype=int)
    for r in range(rows):
        for c in range(cols):
            if terminal[r, c]:
                continue

            action_values = []
            for a in range(4):
                nr = r + dr[a]
                nc = c + dc[a]

                if 0 <= nr < rows and 0 <= nc < cols:
                    action_values.append(rewards[r, c] + gamma * V[nr, nc])
                else:
                    action_values.append(rewards[r, c] + gamma * V[r, c])

            policy[r, c] = np.argmax(action_values)

    print("Optimal Policy:")
    print_policy(policy)

    return V, policy

def print_table(V):
    print("+-------+-------+-------+-------+")
    for r in range(rows):
        for c in range(cols):
            print(f"| {V[r, c]:.2f} ", end="")
        print("|")
        print("+-------+-------+-------+-------+")
```

```python
def print_policy(policy):
    print("+---+---+---+---+")
    for r in range(rows):
        for c in range(cols):
            if terminal[r, c]:
                print("| * ", end="")
            else:
                print(f"| {actions[policy[r, c]]} ", end="")
        print("|")
        print("+---+---+---+---+")

# Run the algorithm
V, policy = value_iteration()
```

This code will trace through the value iteration algorithm on our 4×4 grid world, printing the value function at each iteration and the final optimal policy.