

Introduction to Dynamic Programming (DP)

Infinite Horizon

Outline

- Infinite Horizon Problems
 - Notation
 - Value Iteration
 - Policy Iteration
- Operator Formulation
- Introduction to Q-function

The Infinite Horizon Case

Limiting Solution: Bellman's Equation

Given that $V^*(s) = \lim_{t \rightarrow \infty} V_t^*(s)$, it follows

$$V^*(i) = \min_a \left\{ c(i, a) + \gamma \sum_j P(j \mid i, a) V^*(j) \right\}$$

Greedy Policy and Optimality

$$\mu \text{ is optimal iff } \min_a \left[c(s, a) + \gamma \mathbb{E} V^*(s') \right] = V^*(s).$$

Value Iteration (special case)

$$V_0^*(i) = 0$$

$$V_{t+1}^*(i) = \min_a \left\{ c(i, a) + \gamma \sum_j P(j \mid i, a) V_t^*(j) \right\}$$

We have shown convergence.

What is the interpretation of this result?

Q Function

$$Q(s, a) = c(s, a) + \gamma \mathbb{E}V^*(s')$$

We see from above: $\min_a Q(s, a) = V^*(s)$

Q will have advantages soon.

Operators

What do we mean by an operator?

$$V : \mathcal{S} \rightarrow \mathbb{R}$$

$$TV : \mathcal{S} \rightarrow \mathbb{R}$$

Note that: $T^k V = T(T^{k-1} V)$; (repeated operation).

Define DP operator:

$$(TV)(i) = \min_a \left\{ c(i, a) + \gamma \sum_j P(j \mid i, a) V(j) \right\}$$

The DP equation says: $TV = V$

This has a unique solution V^* .

Evaluation Operator

Restrict to a fixed policy:

$$(T_{\mu}V)(i) = c(i, \mu(i)) + \gamma \sum_j P(j \mid i, \mu(i)) V(j)$$

Then the equation $T_{\mu}V = V$ has a unique solution V^{μ} .

Matrix Representation (Discrete States)

This can be written in matrix form: $V^\mu = \underline{c} + \gamma P_\mu V^\mu$

A simple form for evaluation.

Repeated Operations

$T^k V$: k -iterations of DP algorithm.

$T^k V_0 \longrightarrow V^*$ (value iteration)

$T_\mu^k V$ = repeated evaluation of μ

Definitions

Value function: $V : \mathcal{S} \rightarrow \mathbb{R}$

We say $V_1 \leq V_2$ if $V_1(i) \leq V_2(i) \quad \forall i$

Define $\|V\|_\infty = \max_i |V(i)|$

Follows from this: $\beta = \|V_1 - V_2\|_\infty$

Then $V_1 - \beta \mathbf{1} \leq V_2 \leq V_1 + \beta \mathbf{1}$

Contraction Operator

T is a contraction if there exists $\gamma < 1$ such that

$$\|TV_1 - TV_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty$$

Contraction $\Rightarrow$ nice properties.

Contraction Theorem

Contraction Mapping Theorem.

Let T be a $\|\cdot\|_\infty$ contraction. Then:

$TV = V$ has a unique solution $\bar{V}$,

and for any initial V_0 ,

$$\|T^k V_0 - \bar{V}\|_\infty \leq \gamma^k \|V_0 - \bar{V}\|_\infty.$$

(Already shown).

Monotonicity of DP Operator

If $V_1 \leq V_2$ then $TV_1 \leq TV_2$

$$c(i, a) + \gamma \sum_j P(j \mid i, a) V_1(j) \leq c(i, a) + \gamma \sum_j P(j \mid i, a) V_2(j)$$

$$\Rightarrow \min_a \left\{ c(i, a) + \gamma \sum_j P(j \mid i, a) V_1(j) \right\} \leq \min_a \left\{ c(i, a) + \gamma \sum_j P(j \mid i, a) V_2(j) \right\}$$

More Properties

- Constant effect: $T(V + \beta \mathbf{1}) = TV + \gamma\beta \mathbf{1}$

(Show by Direct computation).

- T is a contraction with factor γ .

(Already shown).

How do we compute V^* in the finite case?

Value Iteration: $V_k = T^k V_0$

Then $\|T^k V_0 - V^*\| \leq \gamma^k \|V^* - V_0\|$

Stop when improvements are small.

- Start with initial policy μ .
- **Policy evaluation:** $V^\mu = T_\mu V^\mu$ (linear equations).
- **Policy update:** $TV^\mu = T_{\mu'} V^\mu$ (greedy policy).

Policy Iteration: Property 1

- **Improvement:** $V^{\mu'} \leq V^{\mu}$

We have $TV^{\mu} = T_{\mu'}V^{\mu} \leq T_{\mu}V^{\mu} = V^{\mu}$,

using monotonicity, we want to show $V^{\mu'} \leq V^{\mu}$.

$$T_{\mu'}^2 V^{\mu} \leq T_{\mu'} V^{\mu} \leq V^{\mu}$$

$$T_{\mu'}^3 V^{\mu} \leq T_{\mu'} V^{\mu} < V^{\mu}$$

This implies that $V^{\mu'} \leq V^{\mu}$.

If $T_{\mu'}V^{\mu} = TV^{\mu} = T_{\mu}V^{\mu} = V^{\mu}$, then V^{μ} is optimal.

Policy Iteration

- **Terminates in a finite number of steps.**

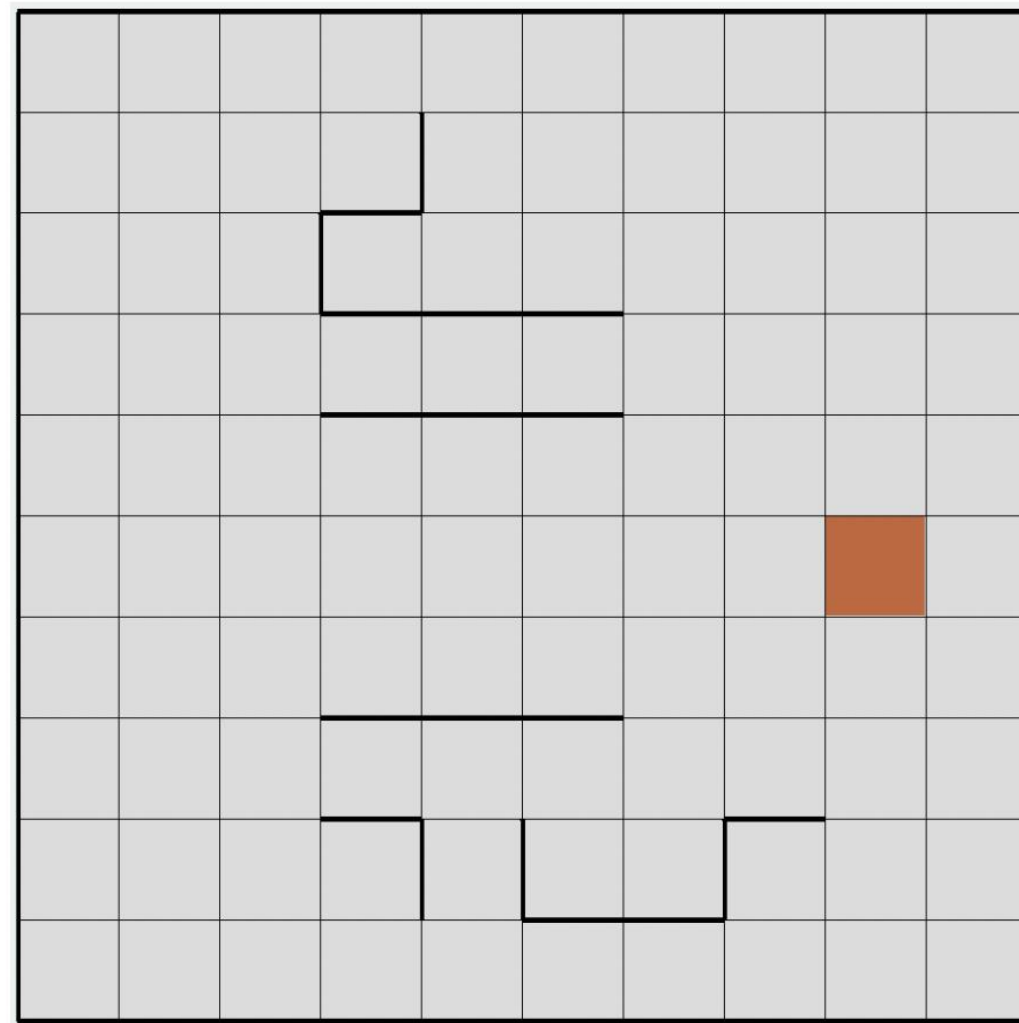
Follows from having a finite number of policies.

Error Bounds

T, T_μ, γ as before.

- If $\|TV - V\| \leq \varepsilon$, then $\|V - V^*\| \leq \frac{\varepsilon}{1-\gamma}$.
- Similarly for T_μ, V^μ .
- If $\|V - V^*\| \leq \varepsilon$ and $TV = T_\mu V$, then $\|V^\mu - V^*\| \leq \frac{2\gamma\varepsilon}{1-\gamma}$.

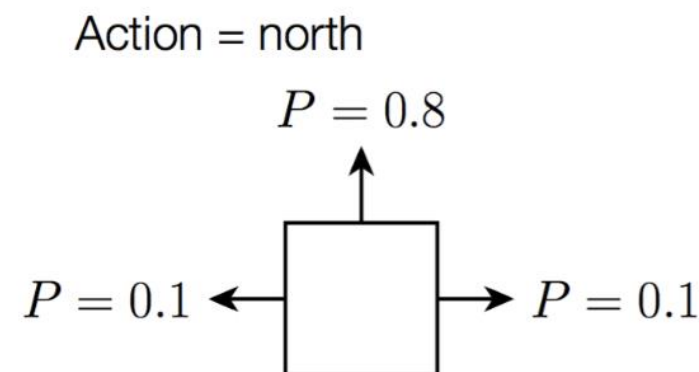
Grid-World Problem



Grid World: Winter Parking with Ice and Potholes

- Simple grid world with a goal state (green, desired parking spot) with reward (+1), a “bad state” (red, pothole) with reward (−100), and all other states neutral (+0).
- *Omnidirectional vehicle (agent)* can head in any direction. Actions move in the desired direction with probability 0.8, and in one of the perpendicular directions otherwise.
- Taking an action that would bump into a wall leaves the agent where it is.

0	0	0	1
0		0	-100
0	0	0	0



Value Iteration: Step 1

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

0	0	0.72	1.81
0		0	-99.91
0	0	0	0

$\hat{V}$ at one iteration

(b)

$$V_{i+1}(s) = \max_{a \in A} \left[r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} V_i(s') \right], \quad \text{for all } s$$

Let's arbitrarily initialize V_0 as the reward function, since it can be any function.

Need to also do this for all the “unnamed” states, too.

Grid World: Value Iteration

$$\gamma = 0.9$$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

Recall value iteration algorithm:

$$V_{i+1}(s) = \max_{a \in A} \left[r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} V_i(s') \right] \quad \text{for all } s$$

Let's arbitrarily initialize V_0 as the reward function, since it can be any function.

Example update (red state):

$$V_1(\text{red}) = -100 + \gamma \max \left\{ \begin{array}{ll} 0.8V_0(\text{green}) + 0.1V_0(\text{red}) + 0 & [\text{up}] \\ 0 + 0.1V_0(\text{red}) + 0 & [\text{down}] \\ 0 + 0.1V_0(\text{green}) + 0 & [\text{left}] \\ 0.8V_0(\text{red}) + 0.1V_0(\text{green}) + 1 & [\text{right}] \end{array} \right\}$$

Value Iteration

$$\gamma = 0.9$$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

$$V_{i+1}(s) = \max_{a \in A} \left[r(s, a) + \gamma \mathbb{E}_{s' \sim P(\cdot | s, a)} V_i(s') \right], \quad \text{for all } s$$

Let's arbitrarily initialize V_0 as the reward function, since it can be any function.

Example update (green state):

$$V_1(\text{green}) = 1 + \gamma \max \left\{ \begin{array}{ll} 0.8V_0(\text{green}) + 0.1V_0(\text{green}) & [\text{up}] \\ 0.8V_0(\text{red}) + 0.1V_0(\text{green}) & [\text{down}] \\ 0.1V_0(\text{green}) + 0.1V_0(\text{red}) & [\text{left}] \\ 0.8V_0(\text{red}) + 0.1V_0(\text{green}) + 0 & [\text{right}] \end{array} \right\}$$

$$= 1 + 0.9 (0.9 \times 1) = 1.81 \quad [\text{best: go up}]$$

Value Iteration

Running value iteration with $\gamma = 0.9$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

Running value iteration with $\gamma = 0.9$

0	0	0.72	1.81
0		0	-99.91
0	0	0	0

$\hat{V}$ at one iteration

(b)

Running value iteration with $\gamma = 0.9$

0.809	1.598	2.475	3.745
0.268		0.302	-99.59
0	0.034	0.122	0.004

$\hat{V}$ at five iterations

(c)

Running value iteration with $\gamma = 0.9$

2.686	3.527	4.402	5.812
2.021		1.095	-98.82
1.390	0.903	0.738	0.123

$\hat{V}$ at 10 iterations

(d)

Running value iteration with $\gamma = 0.9$

5.470	6.313	7.190	8.669
4.802		3.347	-96.67
4.161	3.654	3.222	1.526

$\hat{V}$ at 1000 iterations

(e)

Running value iteration with $\gamma = 0.9$

→	→	→	↑
↑		←	←
↑	←	←	↓

Resulting policy after 1000 iterations

(f)

Poicy Iteration

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) = \text{North}$

0	0	0	1
0		0	-100
0	0	0	0

Original reward function

(a)

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) = \text{North}$

0.418	0.884	2.331	6.367
0.367		-8.610	-105.7
-0.168	-4.641	-14.27	-85.05

V^π at one iteration

(b)

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) = \text{North}$

5.414	6.248	7.116	8.634
4.753		2.881	-102.7
2.251	1.977	1.849	-8.701

V^π at two iterations

(c)

Running policy iteration with $\gamma = 0.9$, initialized with policy $\pi(s) = \text{North}$

5.470	6.313	7.190	8.669
4.803		3.347	-96.67
4.161	3.654	3.222	1.526

V^π at three iterations (converged)

(d)

...

Practical Difficulties of DP

- Curse of dimensionality
 - Exponential growth of computation and storage requirements in number of states and actions
 - Quick explosion of number of states in combinatorial problems
- Modeling
 - Not always available
 - Sometimes simulation is easier / preferred
- Real-time computation constraints
 - Data changes, necessitates online replanning

Must Approximate!

Q-Functions

Why use Qs

$$Q(s, a) = \mathbb{E} \left[c(s, a, w) + \gamma \min_{a \in A(s')} Q(s', a) \right], \quad \text{with } s' = f(s, a, w)$$
$$\triangleq (\mathcal{F}Q)(s, a)$$

- Having Q-functions is convenient for online policy implementation.
- Model-free?
- More sample efficient
- Q-Learning is a sampling based method to approximate the Q-function.

Q-Functions

Why use Qs

$$Q_{t+1}(i, a) = (\mathcal{F}Q_t)(i, a)$$

$$Q_{t+1}(i, a) = \mathbb{E} \left[c(i, a, j) + \gamma \min_{a \in A(j)} Q_t(j, a) \right]$$

$\mathcal{F}$ Operator has similar properties as the T operator

Q-Learning for MDPs: Sneak Preview

Q-Iteration (VI for Q-factors):

$$Q_{t+1}(i, a) = (\mathcal{F}Q_t)(i, a)$$

$$Q_{t+1}(i, a) = \mathbb{E} \left[c(i, a) + \gamma \min_{a \in A(j)} Q_t(j, a) \right] = \sum_j p_{ij}(a) g(i, a, j)$$

Q-Learning: Move in the direction of sample of $(\mathcal{F}Q)(i, a)$

$$Q_{t+1}(i, a) = (1 - \alpha_t)Q_t(i, a) + \alpha_t \left(c(i, a) + \gamma \min_{a \in A(j)} Q_t(j, a) \right)$$

$$\begin{aligned} Q_{t+1}(i, a) &= Q_t(i, a) + \alpha_t \left(c(i, a) + \gamma \min_{a \in A(j)} Q_t(j, a) - Q_t(i, a) \right) \\ &= Q_t(i, a) + \alpha_t \{ c(i, a) + \gamma Q_t(j, a^*) - Q_t(i, a) \}, \end{aligned}$$

$$\text{with: } a^* = \arg \min_{a \in A(j)} Q_t(j, a)$$

Q-Learning for MDPs

Tabular version

$$Q_{t+1}(i, a) = Q_t(i, a) + \alpha_t \left(c(i, a) + \gamma \min_{a \in A(j)} Q_t(j, a) - Q_t(i, a) \right)$$

- Choice of state i :
 - random
 - along a trajectory
- Choice of control a :
 - greedy
 - ϵ -greedy / random
 - fixed-policy
- Converges(to the optimal Q) under conditions:
 - Step size
 - All state-action pairs must be visited i.o.