A Guided Tour of Chapter 5: Dynamic Programming

Ashwin Rao

ICME, Stanford University

Dynamic Programming for Prediction and Control

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP
- (Optimal Policy can be extracted from Optimal Value Function)

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP
- (Optimal Policy can be extracted from Optimal Value Function)
- Planning versus Learning: access to the $\mathcal{P}_R$ function ("model")

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP
- (Optimal Policy can be extracted from Optimal Value Function)
- Planning versus Learning: access to the $\mathcal{P}_R$ function ("model")
- Original use of *DP* term: MDP Theory *and* solution methods

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP
- (Optimal Policy can be extracted from Optimal Value Function)
- Planning versus Learning: access to the $\mathcal{P}_R$ function ("model")
- Original use of *DP* term: MDP Theory *and* solution methods
- Bellman refered to DP as the *Principle of Optimality*

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP
- (Optimal Policy can be extracted from Optimal Value Function)
- Planning versus Learning: access to the $\mathcal{P}_R$ function ("model")
- Original use of *DP* term: MDP Theory *and* solution methods
- Bellman refered to DP as the *Principle of Optimality*
- Later, the usage of the term DP diffused out to other algorithms

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP
- (Optimal Policy can be extracted from Optimal Value Function)
- Planning versus Learning: access to the $\mathcal{P}_R$ function ("model")
- Original use of *DP* term: MDP Theory *and* solution methods
- Bellman refered to DP as the *Principle of Optimality*
- Later, the usage of the term DP diffused out to other algorithms
- In CS, it means "recursive algorithms with overlapping subproblems"

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP
- (Optimal Policy can be extracted from Optimal Value Function)
- Planning versus Learning: access to the $\mathcal{P}_R$ function ("model")
- Original use of *DP* term: MDP Theory *and* solution methods
- Bellman refered to DP as the *Principle of Optimality*
- Later, the usage of the term DP diffused out to other algorithms
- In CS, it means "recursive algorithms with overlapping subproblems"
- We restrict the term DP to: "Algorithms for Prediction and Control"

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP
- (Optimal Policy can be extracted from Optimal Value Function)
- Planning versus Learning: access to the $\mathcal{P}_R$ function ("model")
- Original use of *DP* term: MDP Theory *and* solution methods
- Bellman refered to DP as the *Principle of Optimality*
- Later, the usage of the term DP diffused out to other algorithms
- In CS, it means "recursive algorithms with overlapping subproblems"
- We restrict the term DP to: "Algorithms for Prediction and Control"
- Specifically applied to the setting of FiniteMarkovDecisionProcess

Dynamic Programming for Prediction and Control

- Prediction: Compute the Value Function of an MRP
- Control: Compute the Optimal Value Function of an MDP
- (Optimal Policy can be extracted from Optimal Value Function)
- Planning versus Learning: access to the $\mathcal{P}_R$ function ("model")
- Original use of *DP* term: MDP Theory *and* solution methods
- Bellman refered to DP as the *Principle of Optimality*
- Later, the usage of the term DP diffused out to other algorithms
- In CS, it means "recursive algorithms with overlapping subproblems"
- We restrict the term DP to: "Algorithms for Prediction and Control"
- Specifically applied to the setting of FiniteMarkovDecisionProcess
- Later we cover extensions such as Asynchronous DP, Approximate DP

Solving the Value Function as a *Fixed-Point*

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms
- Each of the 3 algorithms is founded on the Bellman Equations

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms
- Each of the 3 algorithms is founded on the Bellman Equations
- Each is an iterative algorithm converging to the true Value Function

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms
- Each of the 3 algorithms is founded on the Bellman Equations
- Each is an iterative algorithm converging to the true Value Function
- Each algorithm is based on the concept of *Fixed-Point*

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms
- Each of the 3 algorithms is founded on the Bellman Equations
- Each is an iterative algorithm converging to the true Value Function
- Each algorithm is based on the concept of *Fixed-Point*

Definition

The Fixed-Point of a function $f : \mathcal{X} \rightarrow \mathcal{X}$ (for some arbitrary domain $\mathcal{X}$) is a value $x \in \mathcal{X}$ that satisfies the equation: $x = f(x)$.

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms
- Each of the 3 algorithms is founded on the Bellman Equations
- Each is an iterative algorithm converging to the true Value Function
- Each algorithm is based on the concept of *Fixed-Point*

Definition

The Fixed-Point of a function $f : \mathcal{X} \rightarrow \mathcal{X}$ (for some arbitrary domain $\mathcal{X}$) is a value $x \in \mathcal{X}$ that satisfies the equation: $x = f(x)$.

- Some functions have multiple fixed-points, some have none

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms
- Each of the 3 algorithms is founded on the Bellman Equations
- Each is an iterative algorithm converging to the true Value Function
- Each algorithm is based on the concept of *Fixed-Point*

Definition

The Fixed-Point of a function $f : \mathcal{X} \rightarrow \mathcal{X}$ (for some arbitrary domain $\mathcal{X}$) is a value $x \in \mathcal{X}$ that satisfies the equation: $x = f(x)$.

- Some functions have multiple fixed-points, some have none
- DP algorithms are based on functions with a unique fixed-point

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms
- Each of the 3 algorithms is founded on the Bellman Equations
- Each is an iterative algorithm converging to the true Value Function
- Each algorithm is based on the concept of *Fixed-Point*

Definition

The Fixed-Point of a function $f : \mathcal{X} \rightarrow \mathcal{X}$ (for some arbitrary domain $\mathcal{X}$) is a value $x \in \mathcal{X}$ that satisfies the equation: $x = f(x)$.

- Some functions have multiple fixed-points, some have none
- DP algorithms are based on functions with a unique fixed-point
- Simple example: $f(x) = \cos(x)$, Fixed-Point: $x^* = \cos(x^*)$

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms
- Each of the 3 algorithms is founded on the Bellman Equations
- Each is an iterative algorithm converging to the true Value Function
- Each algorithm is based on the concept of *Fixed-Point*

Definition

The Fixed-Point of a function $f : \mathcal{X} \rightarrow \mathcal{X}$ (for some arbitrary domain $\mathcal{X}$) is a value $x \in \mathcal{X}$ that satisfies the equation: $x = f(x)$.

- Some functions have multiple fixed-points, some have none
- DP algorithms are based on functions with a unique fixed-point
- Simple example: $f(x) = \cos(x)$, Fixed-Point: $x^* = \cos(x^*)$
- For any x_0, $\cos(\cos(\ldots \cos(x_0) \ldots))$ converges to fixed-point x^*

Solving the Value Function as a *Fixed-Point*

- We will be covering 3 Dynamic Programming algorithms
- Each of the 3 algorithms is founded on the Bellman Equations
- Each is an iterative algorithm converging to the true Value Function
- Each algorithm is based on the concept of *Fixed-Point*

Definition

The Fixed-Point of a function $f : \mathcal{X} \rightarrow \mathcal{X}$ (for some arbitrary domain $\mathcal{X}$) is a value $x \in \mathcal{X}$ that satisfies the equation: $x = f(x)$.

- Some functions have multiple fixed-points, some have none
- DP algorithms are based on functions with a unique fixed-point
- Simple example: $f(x) = \cos(x)$, Fixed-Point: $x^* = \cos(x^*)$
- For any x_0, $\cos(\cos(\ldots\cos(x_0)\ldots))$ converges to fixed-point x^*
- Why does this work? How fast does it converge?

Banach Fixed-Point Theorem

Banach Fixed-Point Theorem

Theorem (Banach Fixed-Point Theorem)

Let $\mathcal{X}$ be a non-empty set equipped with a complete metric $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. Let $f : \mathcal{X} \rightarrow \mathcal{X}$ be such that there exists a $L \in [0, 1)$ such that $d(f(x_1), f(x_2)) \leq L \cdot d(x_1, x_2)$ for all $x_1, x_2 \in \mathcal{X}$. Then,

Banach Fixed-Point Theorem

Theorem (Banach Fixed-Point Theorem)

Let $\mathcal{X}$ be a non-empty set equipped with a complete metric $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Let $f : \mathcal{X} \to \mathcal{X}$ be such that there exists a $L \in [0, 1)$ such that $d(f(x_1), f(x_2)) \leq L \cdot d(x_1, x_2)$ for all $x_1, x_2 \in \mathcal{X}$. Then,

- *There exists a unique Fixed-Point $x^* \in \mathcal{X}$, i.e.,*

$$x^* = f(x^*)$$

Banach Fixed-Point Theorem

Theorem (Banach Fixed-Point Theorem)

Let $\mathcal{X}$ be a non-empty set equipped with a complete metric $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. Let $f : \mathcal{X} \rightarrow \mathcal{X}$ be such that there exists a $L \in [0, 1)$ such that $d(f(x_1), f(x_2)) \leq L \cdot d(x_1, x_2)$ for all $x_1, x_2 \in \mathcal{X}$. Then,

- *There exists a unique Fixed-Point $x^* \in \mathcal{X}$, i.e.,*

$$x^* = f(x^*)$$

- *For any $x_0 \in \mathcal{X}$, and sequence $[x_i | i = 0, 1, 2, \ldots]$ defined as $x_{i+1} = f(x_i)$ for all $i = 0, 1, 2, \ldots$,*

$$\lim_{i \rightarrow \infty} x_i = x^*$$

Banach Fixed-Point Theorem

Theorem (Banach Fixed-Point Theorem)

Let $\mathcal{X}$ be a non-empty set equipped with a complete metric $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. Let $f : \mathcal{X} \rightarrow \mathcal{X}$ be such that there exists a $L \in [0, 1)$ such that $d(f(x_1), f(x_2)) \leq L \cdot d(x_1, x_2)$ for all $x_1, x_2 \in \mathcal{X}$. Then,

- *There exists a unique Fixed-Point $x^* \in \mathcal{X}$, i.e.,*

$$x^* = f(x^*)$$

- *For any $x_0 \in \mathcal{X}$, and sequence $[x_i | i = 0, 1, 2, \ldots]$ defined as $x_{i+1} = f(x_i)$ for all $i = 0, 1, 2, \ldots$,*

$$\lim_{i \rightarrow \infty} x_i = x^*$$

Banach Fixed-Point Theorem

Theorem (Banach Fixed-Point Theorem)

Let $\mathcal{X}$ be a non-empty set equipped with a complete metric $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. Let $f : \mathcal{X} \rightarrow \mathcal{X}$ be such that there exists a $L \in [0, 1)$ such that $d(f(x_1), f(x_2)) \leq L \cdot d(x_1, x_2)$ for all $x_1, x_2 \in \mathcal{X}$. Then,

- *There exists a unique Fixed-Point $x^* \in \mathcal{X}$, i.e.,*

$$x^* = f(x^*)$$

- *For any $x_0 \in \mathcal{X}$, and sequence $[x_i | i = 0, 1, 2, \ldots]$ defined as $x_{i+1} = f(x_i)$ for all $i = 0, 1, 2, \ldots$,*

$$\lim_{i \rightarrow \infty} x_i = x^*$$

If you have a complete metric space $\langle \mathcal{X}, d \rangle$ and a contraction f (with respect to d), then you have an algorithm to solve for the fixed-point of f.

Policy Evaluation (for *Prediction*)

Policy Evaluation (for *Prediction*)

- MDP with $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}, \mathcal{N} = \{s_1, s_2, \ldots, s_m\}$

Policy Evaluation (for *Prediction*)

- MDP with $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}, \mathcal{N} = \{s_1, s_2, \ldots, s_m\}$
- Given a policy π, compute the Value Function of π-implied MRP

Policy Evaluation (for *Prediction*)

- MDP with $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}, \mathcal{N} = \{s_1, s_2, \ldots, s_m\}$
- Given a policy π, compute the Value Function of π-implied MRP
- $\mathcal{P}_R^\pi : \mathcal{N} \times \mathcal{D} \times \mathcal{S} \rightarrow [0, 1]$ is given as a data structure

Policy Evaluation (for *Prediction*)

- MDP with $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}, \mathcal{N} = \{s_1, s_2, \ldots, s_m\}$
- Given a policy π, compute the Value Function of π-implied MRP
- $\mathcal{P}_R^\pi : \mathcal{N} \times \mathcal{D} \times \mathcal{S} \rightarrow [0, 1]$ is given as a data structure
- Extract (from $\mathcal{P}_R^\pi$) $\mathcal{P}^\pi : \mathcal{N} \times \mathcal{S} \rightarrow [0, 1]$ and $\mathcal{R}^\pi : \mathcal{N} \rightarrow \mathbb{R}$

Policy Evaluation (for *Prediction*)

- MDP with $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}, \mathcal{N} = \{s_1, s_2, \ldots, s_m\}$
- Given a policy π, compute the Value Function of π-implied MRP
- $\mathcal{P}_R^\pi : \mathcal{N} \times \mathcal{D} \times \mathcal{S} \to [0,1]$ is given as a data structure
- Extract (from $\mathcal{P}_R^\pi$) $\mathcal{P}^\pi : \mathcal{N} \times \mathcal{S} \to [0,1]$ and $\mathcal{R}^\pi : \mathcal{N} \to \mathbb{R}$
- For non-large spaces, we can compute (in vector notation):

$$\boldsymbol{V}^\pi = (\boldsymbol{I_m} - \gamma \boldsymbol{\mathcal{P}}^\pi)^{-1} \cdot \boldsymbol{\mathcal{R}}^\pi$$

Policy Evaluation (for *Prediction*)

- MDP with $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}, \mathcal{N} = \{s_1, s_2, \ldots, s_m\}$
- Given a policy π, compute the Value Function of π-implied MRP
- $\mathcal{P}_R^\pi : \mathcal{N} \times \mathcal{D} \times \mathcal{S} \rightarrow [0, 1]$ is given as a data structure
- Extract (from $\mathcal{P}_R^\pi$) $\mathcal{P}^\pi : \mathcal{N} \times \mathcal{S} \rightarrow [0, 1]$ and $\mathcal{R}^\pi : \mathcal{N} \rightarrow \mathbb{R}$
- For non-large spaces, we can compute (in vector notation):

$$\boldsymbol{V}^\pi = (\boldsymbol{I_m} - \gamma \boldsymbol{\mathcal{P}}^\pi)^{-1} \cdot \boldsymbol{\mathcal{R}}^\pi$$

- Note: $\boldsymbol{V}^\pi, \boldsymbol{\mathcal{R}}^\pi$ are m-column vectors ($\in \mathbb{R}^m$) and $\boldsymbol{\mathcal{P}}^\pi$ is $m \times m$ matrix

Policy Evaluation (for *Prediction*)

- MDP with $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}, \mathcal{N} = \{s_1, s_2, \ldots, s_m\}$
- Given a policy π, compute the Value Function of π-implied MRP
- $\mathcal{P}_R^\pi : \mathcal{N} \times \mathcal{D} \times \mathcal{S} \rightarrow [0,1]$ is given as a data structure
- Extract (from $\mathcal{P}_R^\pi$) $\mathcal{P}^\pi : \mathcal{N} \times \mathcal{S} \rightarrow [0,1]$ and $\mathcal{R}^\pi : \mathcal{N} \rightarrow \mathbb{R}$
- For non-large spaces, we can compute (in vector notation):

$$\boldsymbol{V}^\pi = (\boldsymbol{I_m} - \gamma \boldsymbol{\mathcal{P}}^\pi)^{-1} \cdot \boldsymbol{\mathcal{R}}^\pi$$

- Note: $\boldsymbol{V}^\pi, \boldsymbol{\mathcal{R}}^\pi$ are m-column vectors ($\in \mathbb{R}^m$) and $\boldsymbol{\mathcal{P}}^\pi$ is $m \times m$ matrix
- So we look for an iterative algorithm to solve MRP Bellman Equation:

$$\boldsymbol{V}^\pi = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{V}^\pi$$

Bellman Policy Operator and its Fixed-Point

Bellman Policy Operator and its Fixed-Point

- Define the *Bellman Policy Operator* $\boldsymbol{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ as:

$$\boldsymbol{B}^\pi(\boldsymbol{V}) = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{V} \text{ for any Value Function vector } \boldsymbol{V} \in \mathbb{R}^m$$

Bellman Policy Operator and its Fixed-Point

- Define the *Bellman Policy Operator* $\boldsymbol{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ as:

$$\boldsymbol{B}^\pi(\boldsymbol{V}) = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{V} \text{ for any Value Function vector } \boldsymbol{V} \in \mathbb{R}^m$$

- $\boldsymbol{B}^\pi$ is an affine transformation on vectors in $\mathbb{R}^m$

Bellman Policy Operator and its Fixed-Point

- Define the *Bellman Policy Operator* $\boldsymbol{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ as:

$$\boldsymbol{B}^\pi(\boldsymbol{V}) = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{V} \text{ for any Value Function vector } \boldsymbol{V} \in \mathbb{R}^m$$

- $\boldsymbol{B}^\pi$ is an affine transformation on vectors in $\mathbb{R}^m$
- So, the MRP Bellman Equation can be expressed as:

$$\boldsymbol{V}^\pi = \boldsymbol{B}^\pi(\boldsymbol{V}^\pi)$$

Bellman Policy Operator and its Fixed-Point

- Define the *Bellman Policy Operator* $\boldsymbol{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ as:

$$\boldsymbol{B}^\pi(\boldsymbol{V}) = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{V} \text{ for any Value Function vector } \boldsymbol{V} \in \mathbb{R}^m$$

- $\boldsymbol{B}^\pi$ is an affine transformation on vectors in $\mathbb{R}^m$
- So, the MRP Bellman Equation can be expressed as:

$$\boldsymbol{V}^\pi = \boldsymbol{B}^\pi(\boldsymbol{V}^\pi)$$

- This means $\boldsymbol{V}^\pi \in \mathbb{R}^m$ is a Fixed-Point of $\boldsymbol{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$

Bellman Policy Operator and its Fixed-Point

- Define the *Bellman Policy Operator* $\boldsymbol{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ as:

$$\boldsymbol{B}^\pi(\boldsymbol{V}) = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{V} \text{ for any Value Function vector } \boldsymbol{V} \in \mathbb{R}^m$$

- $\boldsymbol{B}^\pi$ is an affine transformation on vectors in $\mathbb{R}^m$
- So, the MRP Bellman Equation can be expressed as:

$$\boldsymbol{V}^\pi = \boldsymbol{B}^\pi(\boldsymbol{V}^\pi)$$

- This means $\boldsymbol{V}^\pi \in \mathbb{R}^m$ is a Fixed-Point of $\boldsymbol{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$
- Metric $d : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$ defined as L^∞ norm:

$$d(\boldsymbol{X}, \boldsymbol{Y}) = \|\boldsymbol{X} - \boldsymbol{Y}\|_\infty = \max_{s \in \mathcal{N}} |(\boldsymbol{X} - \boldsymbol{Y})(s)|$$

Bellman Policy Operator and its Fixed-Point

- Define the *Bellman Policy Operator* $\boldsymbol{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ as:

$$\boldsymbol{B}^\pi(\boldsymbol{V}) = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{V} \text{ for any Value Function vector } \boldsymbol{V} \in \mathbb{R}^m$$

- $\boldsymbol{B}^\pi$ is an affine transformation on vectors in $\mathbb{R}^m$
- So, the MRP Bellman Equation can be expressed as:

$$\boldsymbol{V}^\pi = \boldsymbol{B}^\pi(\boldsymbol{V}^\pi)$$

- This means $\boldsymbol{V}^\pi \in \mathbb{R}^m$ is a Fixed-Point of $\boldsymbol{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$
- Metric $d : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$ defined as L^∞ norm:

$$d(\boldsymbol{X}, \boldsymbol{Y}) = \|\boldsymbol{X} - \boldsymbol{Y}\|_\infty = \max_{s \in \mathcal{N}} |(\boldsymbol{X} - \boldsymbol{Y})(s)|$$

- $\boldsymbol{B}^\pi$ is a contraction function under L^∞ norm: For all $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^m$,

$$\max_{s \in \mathcal{N}} |(\boldsymbol{B}^\pi(\boldsymbol{X}) - \boldsymbol{B}^\pi(\boldsymbol{Y}))(s)| = \gamma \cdot \max_{s \in \mathcal{N}} |(\boldsymbol{\mathcal{P}}^\pi \cdot (\boldsymbol{X} - \boldsymbol{Y}))(s)|$$

$$\leq \gamma \cdot \max_{s \in \mathcal{N}} |(\boldsymbol{X} - \boldsymbol{Y})(s)|$$

Policy Evaluation Convergence Theorem

Invoking the Banach Fixed-Point Theorem for $\gamma < 1$ gives:

Policy Evaluation Convergence Theorem

Invoking the Banach Fixed-Point Theorem for $\gamma < 1$ gives:

Theorem (Policy Evaluation Convergence Theorem)

For a Finite MDP with $|\mathcal{N}| = m$ *and* $\gamma < 1$*, if* $\boldsymbol{V}^\pi \in \mathbb{R}^m$ *is the Value Function of the MDP when evaluated with a fixed policy* $\pi : \mathcal{N} \times \mathcal{A} \rightarrow [0, 1]$*, then* $\boldsymbol{V}^\pi$ *is the unique Fixed-Point of the Bellman Policy Operator* $\boldsymbol{B}^\pi : \mathbb{R}^m \rightarrow \mathbb{R}^m$*, and*

$$\lim_{i \to \infty} (\boldsymbol{B}^\pi)^i(\boldsymbol{V_0}) \rightarrow \boldsymbol{V}^\pi \text{ for all starting Value Functions } \boldsymbol{V_0} \in \mathbb{R}^m$$

Policy Evaluation algorithm

Policy Evaluation algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$

Policy Evaluation algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \ldots$, calculate in each iteration:

$$\boldsymbol{V_{i+1}} = \boldsymbol{B}^\pi(\boldsymbol{V_i}) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \boldsymbol{V_i}$$

Policy Evaluation algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \ldots$, calculate in each iteration:

$$\boldsymbol{V_{i+1}} = \boldsymbol{B}^\pi(\boldsymbol{V_i}) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \boldsymbol{V_i}$$

- Stop when $d(\boldsymbol{V_i}, \boldsymbol{V_{i+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_i} - \boldsymbol{V_{i+1}})(s)|$ is small enough

Policy Evaluation algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \ldots$, calculate in each iteration:

$$\boldsymbol{V_{i+1}} = \boldsymbol{B}^\pi(\boldsymbol{V_i}) = \mathcal{R}^\pi + \gamma \mathcal{P}^\pi \cdot \boldsymbol{V_i}$$

- Stop when $d(\boldsymbol{V_i}, \boldsymbol{V_{i+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_i} - \boldsymbol{V_{i+1}})(s)|$ is small enough

Policy Evaluation algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \ldots$, calculate in each iteration:

$$\boldsymbol{V_{i+1}} = \boldsymbol{B}^\pi(\boldsymbol{V_i}) = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{V_i}$$

- Stop when $d(\boldsymbol{V_i}, \boldsymbol{V_{i+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_i} - \boldsymbol{V_{i+1}})(s)|$ is small enough

Banach Fixed-Point Theorem also assures speed of convergence (dependent on choice of starting Value Function $\boldsymbol{V_0}$ and on choice of γ).

Policy Evaluation algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \ldots$, calculate in each iteration:

$$\boldsymbol{V_{i+1}} = \boldsymbol{B}^\pi(\boldsymbol{V_i}) = \boldsymbol{\mathcal{R}}^\pi + \gamma \boldsymbol{\mathcal{P}}^\pi \cdot \boldsymbol{V_i}$$

- Stop when $d(\boldsymbol{V_i}, \boldsymbol{V_{i+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_i} - \boldsymbol{V_{i+1}})(s)|$ is small enough

Banach Fixed-Point Theorem also assures speed of convergence (dependent on choice of starting Value Function $\boldsymbol{V_0}$ and on choice of γ).

Running time of each iteration is $O(m^2)$. Constructing the MRP from the MDP and the policy takes $O(m^2 k)$ operations, where $m = |\mathcal{N}|, k = |\mathcal{A}|$.

Greedy Policy

Greedy Policy

- Now we move on to solving the MDP *Control* problem

Greedy Policy

- Now we move on to solving the MDP *Control* problem
- We want to iterate *Policy Improvements* to drive to an *Optimal Policy*

Greedy Policy

- Now we move on to solving the MDP *Control* problem
- We want to iterate *Policy Improvements* to drive to an *Optimal Policy*
- *Policy Improvement* is based on a "greedy" technique

Greedy Policy

- Now we move on to solving the MDP *Control* problem
- We want to iterate *Policy Improvements* to drive to an *Optimal Policy*
- *Policy Improvement* is based on a "greedy" technique
- The *Greedy Policy Function* $G : \mathbb{R}^m \rightarrow (\mathcal{N} \rightarrow \mathcal{A})$ (interpreted as a function mapping a Value Function vector $\boldsymbol{V}$ to a deterministic policy $\pi'_D : \mathcal{N} \rightarrow \mathcal{A}$) is defined as:

$$G(\boldsymbol{V})(s) = \pi'_D(s) = \arg\max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}(s')\}$$

Greedy Policy

- Now we move on to solving the MDP *Control* problem
- We want to iterate *Policy Improvements* to drive to an *Optimal Policy*
- *Policy Improvement* is based on a "greedy" technique
- The *Greedy Policy Function* $G : \mathbb{R}^m \rightarrow (\mathcal{N} \rightarrow \mathcal{A})$
(interpreted as a function mapping a Value Function vector $\boldsymbol{V}$ to a deterministic policy $\pi'_D : \mathcal{N} \rightarrow \mathcal{A}$) is defined as:

$$G(\boldsymbol{V})(s) = \pi'_D(s) = \arg\max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}(s')\}$$

Greedy Policy

- Now we move on to solving the MDP *Control* problem
- We want to iterate *Policy Improvements* to drive to an *Optimal Policy*
- *Policy Improvement* is based on a "greedy" technique
- The *Greedy Policy Function* $G : \mathbb{R}^m \rightarrow (\mathcal{N} \rightarrow \mathcal{A})$ (interpreted as a function mapping a Value Function vector $\boldsymbol{V}$ to a deterministic policy $\pi'_D : \mathcal{N} \rightarrow \mathcal{A}$) is defined as:

$$G(\boldsymbol{V})(s) = \pi'_D(s) = \arg\max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}(s')\}$$

Definition (Value Function Comparison)

We say $X \geq Y$ for Value Functions $X, Y : \mathcal{N} \rightarrow \mathbb{R}$ of an MDP iff:

$$X(s) \geq Y(s) \text{ for all } s \in \mathcal{N}$$

Greedy Policy

- Now we move on to solving the MDP *Control* problem
- We want to iterate *Policy Improvements* to drive to an *Optimal Policy*
- *Policy Improvement* is based on a "greedy" technique
- The *Greedy Policy Function* $G : \mathbb{R}^m \rightarrow (\mathcal{N} \rightarrow \mathcal{A})$ (interpreted as a function mapping a Value Function vector $\boldsymbol{V}$ to a deterministic policy $\pi'_D : \mathcal{N} \rightarrow \mathcal{A}$) is defined as:

$$G(\boldsymbol{V})(s) = \pi'_D(s) = \arg\max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}(s')\}$$

Definition (Value Function Comparison)

We say $X \geq Y$ for Value Functions $X, Y : \mathcal{N} \rightarrow \mathbb{R}$ of an MDP iff:

$$X(s) \geq Y(s) \text{ for all } s \in \mathcal{N}$$

We say π_1 better ("improvement") than π_2 if $\boldsymbol{V}^{\pi_1} \geq \boldsymbol{V}^{\pi_2}$

Policy Improvement Theorem

Theorem (Policy Improvement Theorem)

For a finite MDP, for any policy π,

$$\boldsymbol{V}^{\pi'_D} = \boldsymbol{V}^{G(\boldsymbol{V}^{\pi})} \geq \boldsymbol{V}^{\pi}$$

Policy Improvement Theorem

Theorem (Policy Improvement Theorem)

For a finite MDP, for any policy π,

$$\boldsymbol{V}^{\pi'_D} = \boldsymbol{V}^{G(\boldsymbol{V}^{\pi})} \geq \boldsymbol{V}^{\pi}$$

- Note that applying $\boldsymbol{B}^{\pi'_D} = \boldsymbol{B}^{G(\boldsymbol{V}^{\pi})}$ repeatedly, starting with $\boldsymbol{V}^{\pi}$, will converge to $\boldsymbol{V}^{\pi'_D}$ (Policy Evaluation with policy $\pi'_D = G(\boldsymbol{V}^{\pi})$):

$$\lim_{i \to \infty} (\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^{\pi}) = \boldsymbol{V}^{\pi'_D}$$

Policy Improvement Theorem

Theorem (Policy Improvement Theorem)

For a finite MDP, for any policy π*,*

$$\boldsymbol{V}^{\pi'_D} = \boldsymbol{V}^{G(\boldsymbol{V}^\pi)} \geq \boldsymbol{V}^\pi$$

- Note that applying $\boldsymbol{B}^{\pi'_D} = \boldsymbol{B}^{G(\boldsymbol{V}^\pi)}$ repeatedly, starting with $\boldsymbol{V}^\pi$, will converge to $\boldsymbol{V}^{\pi'_D}$ (Policy Evaluation with policy $\pi'_D = G(\boldsymbol{V}^\pi)$):

$$\lim_{i \to \infty} (\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^\pi) = \boldsymbol{V}^{\pi'_D}$$

- So the proof is complete if we prove that:

$$(\boldsymbol{B}^{\pi'_D})^{i+1}(\boldsymbol{V}^\pi) \geq (\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^\pi) \text{ for all } i = 0, 1, 2, \ldots$$

Theorem (Policy Improvement Theorem)

For a finite MDP, for any policy π,

$$\boldsymbol{V}^{\pi'_D} = \boldsymbol{V}^{G(\boldsymbol{V}^{\pi})} \geq \boldsymbol{V}^{\pi}$$

- Note that applying $\boldsymbol{B}^{\pi'_D} = \boldsymbol{B}^{G(\boldsymbol{V}^{\pi})}$ repeatedly, starting with $\boldsymbol{V}^{\pi}$, will converge to $\boldsymbol{V}^{\pi'_D}$ (Policy Evaluation with policy $\pi'_D = G(\boldsymbol{V}^{\pi})$):

$$\lim_{i \to \infty} (\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^{\pi}) = \boldsymbol{V}^{\pi'_D}$$

- So the proof is complete if we prove that:

$$(\boldsymbol{B}^{\pi'_D})^{i+1}(\boldsymbol{V}^{\pi}) \geq (\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^{\pi}) \text{ for all } i = 0, 1, 2, \ldots$$

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^{\pi}) | i = 0, 1, 2, \ldots]$ with repeated applications of $\boldsymbol{B}^{\pi'_D}$

Proof by Induction

Proof by Induction

- To prove the base case (of proof by induction), note that:

$$\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}^{\pi}(s')\} = \max_{a \in \mathcal{A}} Q^{\pi}(s,a)$$

Proof by Induction

- To prove the base case (of proof by induction), note that:

$$\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}^{\pi}(s')\} = \max_{a \in \mathcal{A}} Q^{\pi}(s,a)$$

- $\boldsymbol{V}^{\pi}(s)$ is weighted average of $Q^{\pi}(s,\cdot)$ while $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})(s)$ is maximum

$$\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi}) \geq \boldsymbol{V}^{\pi}$$

Proof by Induction

- To prove the base case (of proof by induction), note that:

$$\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}^{\pi}(s')\} = \max_{a \in \mathcal{A}} Q^{\pi}(s,a)$$

- $\boldsymbol{V}^{\pi}(s)$ is weighted average of $Q^{\pi}(s,\cdot)$ while $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})(s)$ is maximum

$$\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi}) \geq \boldsymbol{V}^{\pi}$$

- Induction step is proved by monotonicity of $\boldsymbol{B}^{\pi}$ operator (for any π):

$$\text{Monotonicity Property of } \boldsymbol{B}^{\pi} : \boldsymbol{X} \geq \boldsymbol{Y} \Rightarrow \boldsymbol{B}^{\pi}(\boldsymbol{X}) \geq \boldsymbol{B}^{\pi}(\boldsymbol{Y})$$

$$\text{So } (\boldsymbol{B}^{\pi'_D})^{i+1}(\boldsymbol{V}^{\pi}) \geq (\boldsymbol{B}^{\pi'_D})^{i}(\boldsymbol{V}^{\pi}) \Rightarrow (\boldsymbol{B}^{\pi'_D})^{i+2}(\boldsymbol{V}^{\pi}) \geq (\boldsymbol{B}^{\pi'_D})^{i+1}(\boldsymbol{V}^{\pi})$$

Intuitive Understanding of Policy Improvement Theorem

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^\pi) | i = 0, 1, 2, \ldots]$

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^{\pi})|i = 0, 1, 2, \ldots]$
- Each stage of further application of $\boldsymbol{B}^{\pi'_D}$ improves the Value Function

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^{\pi})|i = 0, 1, 2, \ldots]$
- Each stage of further application of $\boldsymbol{B}^{\pi'_D}$ improves the Value Function
- Stage 0: Value Function $\boldsymbol{V}^{\pi}$ means execute policy π throughout

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^\pi) | i = 0, 1, 2, \ldots]$
- Each stage of further application of $\boldsymbol{B}^{\pi'_D}$ improves the Value Function
- Stage 0: Value Function $\boldsymbol{V}^\pi$ means execute policy π throughout
- Stage 1: VF $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^\pi)$ means execute improved policy π'_D for the 1st time step, then execute policy π for all further time steps

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^{\pi}) | i = 0, 1, 2, \ldots]$
- Each stage of further application of $\boldsymbol{B}^{\pi'_D}$ improves the Value Function
- Stage 0: Value Function $\boldsymbol{V}^{\pi}$ means execute policy π throughout
- Stage 1: VF $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})$ means execute improved policy π'_D for the 1st time step, then execute policy π for all further time steps
- Improves the VF from Stage 0: $\boldsymbol{V}^{\pi}$ to Stage 1: $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})$

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^\pi) | i = 0, 1, 2, \ldots]$
- Each stage of further application of $\boldsymbol{B}^{\pi'_D}$ improves the Value Function
- Stage 0: Value Function $\boldsymbol{V}^\pi$ means execute policy π throughout
- Stage 1: VF $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^\pi)$ means execute improved policy π'_D for the 1st time step, then execute policy π for all further time steps
- Improves the VF from Stage 0: $\boldsymbol{V}^\pi$ to Stage 1: $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^\pi)$
- Stage 2: VF $(\boldsymbol{B}^{\pi'_D})^2(\boldsymbol{V}^\pi)$ means execute improved policy π'_D for first 2 time steps, then execute policy π for all further time steps

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^{\pi}) | i = 0, 1, 2, \ldots]$
- Each stage of further application of $\boldsymbol{B}^{\pi'_D}$ improves the Value Function
- Stage 0: Value Function $\boldsymbol{V}^{\pi}$ means execute policy π throughout
- Stage 1: VF $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})$ means execute improved policy π'_D for the 1st time step, then execute policy π for all further time steps
- Improves the VF from Stage 0: $\boldsymbol{V}^{\pi}$ to Stage 1: $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})$
- Stage 2: VF $(\boldsymbol{B}^{\pi'_D})^2(\boldsymbol{V}^{\pi})$ means execute improved policy π'_D for first 2 time steps, then execute policy π for all further time steps
- Improves the VF from Stage 1: $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})$ to Stage 2: $(\boldsymbol{B}^{\pi'_D})^2(\boldsymbol{V}^{\pi})$

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^\pi) | i = 0, 1, 2, \ldots]$
- Each stage of further application of $\boldsymbol{B}^{\pi'_D}$ improves the Value Function
- Stage 0: Value Function $\boldsymbol{V}^\pi$ means execute policy π throughout
- Stage 1: VF $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^\pi)$ means execute improved policy π'_D for the 1st time step, then execute policy π for all further time steps
- Improves the VF from Stage 0: $\boldsymbol{V}^\pi$ to Stage 1: $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^\pi)$
- Stage 2: VF $(\boldsymbol{B}^{\pi'_D})^2(\boldsymbol{V}^\pi)$ means execute improved policy π'_D for first 2 time steps, then execute policy π for all further time steps
- Improves the VF from Stage 1: $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^\pi)$ to Stage 2: $(\boldsymbol{B}^{\pi'_D})^2(\boldsymbol{V}^\pi)$
- Each stage applies policy π'_D instead of π for an extra time step

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^\pi) | i = 0, 1, 2, \ldots]$
- Each stage of further application of $\boldsymbol{B}^{\pi'_D}$ improves the Value Function
- Stage 0: Value Function $\boldsymbol{V}^\pi$ means execute policy π throughout
- Stage 1: VF $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^\pi)$ means execute improved policy π'_D for the 1st time step, then execute policy π for all further time steps
- Improves the VF from Stage 0: $\boldsymbol{V}^\pi$ to Stage 1: $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^\pi)$
- Stage 2: VF $(\boldsymbol{B}^{\pi'_D})^2(\boldsymbol{V}^\pi)$ means execute improved policy π'_D for first 2 time steps, then execute policy π for all further time steps
- Improves the VF from Stage 1: $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^\pi)$ to Stage 2: $(\boldsymbol{B}^{\pi'_D})^2(\boldsymbol{V}^\pi)$
- Each stage applies policy π'_D instead of π for an extra time step
- These stages are the iterations of *Policy Evaluation* (using policy π'_D)

Intuitive Understanding of Policy Improvement Theorem

- Non-decreasing tower of Value Functions $[(\boldsymbol{B}^{\pi'_D})^i(\boldsymbol{V}^{\pi})|i = 0, 1, 2, \ldots]$
- Each stage of further application of $\boldsymbol{B}^{\pi'_D}$ improves the Value Function
- Stage 0: Value Function $\boldsymbol{V}^{\pi}$ means execute policy π throughout
- Stage 1: VF $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})$ means execute improved policy π'_D for the 1st time step, then execute policy π for all further time steps
- Improves the VF from Stage 0: $\boldsymbol{V}^{\pi}$ to Stage 1: $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})$
- Stage 2: VF $(\boldsymbol{B}^{\pi'_D})^2(\boldsymbol{V}^{\pi})$ means execute improved policy π'_D for first 2 time steps, then execute policy π for all further time steps
- Improves the VF from Stage 1: $\boldsymbol{B}^{\pi'_D}(\boldsymbol{V}^{\pi})$ to Stage 2: $(\boldsymbol{B}^{\pi'_D})^2(\boldsymbol{V}^{\pi})$
- Each stage applies policy π'_D instead of π for an extra time step
- These stages are the iterations of *Policy Evaluation* (using policy π'_D)
- Builds non-decreasing tower of VFs that converge to VF $\boldsymbol{V}^{\pi'_D}$ ($\geq \boldsymbol{V}^{\pi}$)

Repeating Policy Improvement and Policy Evaluation

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:
  - Start with Value Function $\boldsymbol{V}^{\pi}$ (for policy π)

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:
  - Start with Value Function $\boldsymbol{V}^\pi$ (for policy π)
  - Perform a "greedy policy improvement" to create policy $\pi'_D = G(\boldsymbol{V}^\pi)$

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:
  - Start with Value Function $\boldsymbol{V}^\pi$ (for policy π)
  - Perform a "greedy policy improvement" to create policy $\pi'_D = G(\boldsymbol{V}^\pi)$
  - Perform Policy Evaluation (for policy π'_D) with starting VF $\boldsymbol{V}^\pi$

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:
    - Start with Value Function $\boldsymbol{V}^{\pi}$ (for policy π)
    - Perform a "greedy policy improvement" to create policy $\pi'_D = G(\boldsymbol{V}^{\pi})$
    - Perform Policy Evaluation (for policy π'_D) with starting VF $\boldsymbol{V}^{\pi}$
    - This results in VF $\boldsymbol{V}^{\pi'_D} \geq$ starting VF $\boldsymbol{V}^{\pi}$

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:
  - Start with Value Function $\boldsymbol{V}^\pi$ (for policy π)
  - Perform a "greedy policy improvement" to create policy $\pi'_D = G(\boldsymbol{V}^\pi)$
  - Perform Policy Evaluation (for policy π'_D) with starting VF $\boldsymbol{V}^\pi$
  - This results in VF $\boldsymbol{V}^{\pi'_D} \geq$ starting VF $\boldsymbol{V}^\pi$
- We can repeat this process starting with $\boldsymbol{V}^{\pi'_D}$

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:
  - Start with Value Function $\boldsymbol{V}^{\pi}$ (for policy π)
  - Perform a "greedy policy improvement" to create policy $\pi'_D = G(\boldsymbol{V}^{\pi})$
  - Perform Policy Evaluation (for policy π'_D) with starting VF $\boldsymbol{V}^{\pi}$
  - This results in VF $\boldsymbol{V}^{\pi'_D} \geq$ starting VF $\boldsymbol{V}^{\pi}$
- We can repeat this process starting with $\boldsymbol{V}^{\pi'_D}$
- Creating an improved policy π''_D and improved VF $\boldsymbol{V}^{\pi''_D}$.

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:
  - Start with Value Function $\boldsymbol{V}^{\pi}$ (for policy π)
  - Perform a "greedy policy improvement" to create policy $\pi'_D = G(\boldsymbol{V}^{\pi})$
  - Perform Policy Evaluation (for policy π'_D) with starting VF $\boldsymbol{V}^{\pi}$
  - This results in VF $\boldsymbol{V}^{\pi'_D} \geq$ starting VF $\boldsymbol{V}^{\pi}$
- We can repeat this process starting with $\boldsymbol{V}^{\pi'_D}$
- Creating an improved policy π''_D and improved VF $\boldsymbol{V}^{\pi''_D}$.
- ... and we can keep going to create further improved policies/VFs

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:
  - Start with Value Function $\boldsymbol{V}^\pi$ (for policy π)
  - Perform a "greedy policy improvement" to create policy $\pi'_D = G(\boldsymbol{V}^\pi)$
  - Perform Policy Evaluation (for policy π'_D) with starting VF $\boldsymbol{V}^\pi$
  - This results in VF $\boldsymbol{V}^{\pi'_D} \geq$ starting VF $\boldsymbol{V}^\pi$
- We can repeat this process starting with $\boldsymbol{V}^{\pi'_D}$
- Creating an improved policy π''_D and improved VF $\boldsymbol{V}^{\pi''_D}$.
- ... and we can keep going to create further improved policies/VFs
- ... until there is no further improvement

Repeating Policy Improvement and Policy Evaluation

- Policy Improvement Theorem says:
  - Start with Value Function $\boldsymbol{V}^{\pi}$ (for policy π)
  - Perform a "greedy policy improvement" to create policy $\pi'_D = G(\boldsymbol{V}^{\pi})$
  - Perform Policy Evaluation (for policy π'_D) with starting VF $\boldsymbol{V}^{\pi}$
  - This results in VF $\boldsymbol{V}^{\pi'_D} \geq$ starting VF $\boldsymbol{V}^{\pi}$
- We can repeat this process starting with $\boldsymbol{V}^{\pi'_D}$
- Creating an improved policy π''_D and improved VF $\boldsymbol{V}^{\pi''_D}$.
- ... and we can keep going to create further improved policies/VFs
- ... until there is no further improvement
- This in fact is the *Policy Iteration* algorithm

Policy Iteration algorithm

Policy Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$

Policy Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $j = 0, 1, 2, \ldots$, calculate in each iteration:

$$\text{Deterministic Policy } \pi_{j+1} = G(\boldsymbol{V_j})$$

$$\text{Value Function } \boldsymbol{V_{j+1}} = \lim_{i \to \infty} (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j})$$

Policy Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $j = 0, 1, 2, \ldots$, calculate in each iteration:

$$\text{Deterministic Policy } \pi_{j+1} = G(\boldsymbol{V_j})$$

$$\text{Value Function } \boldsymbol{V_{j+1}} = \lim_{i \to \infty} (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j})$$

- Stop when $d(\boldsymbol{V_j}, \boldsymbol{V_{j+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_j} - \boldsymbol{V_{j+1}})(s)|$ is small enough

Policy Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $j = 0, 1, 2, \ldots$, calculate in each iteration:

$$\text{Deterministic Policy } \pi_{j+1} = G(\boldsymbol{V_j})$$

$$\text{Value Function } \boldsymbol{V_{j+1}} = \lim_{i \to \infty} (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j})$$

- Stop when $d(\boldsymbol{V_j}, \boldsymbol{V_{j+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_j} - \boldsymbol{V_{j+1}})(s)|$ is small enough

Policy Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $j = 0, 1, 2, \ldots$, calculate in each iteration:

$$\text{Deterministic Policy } \pi_{j+1} = G(\boldsymbol{V_j})$$

$$\text{Value Function } \boldsymbol{V_{j+1}} = \lim_{i \to \infty} (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j})$$

- Stop when $d(\boldsymbol{V_j}, \boldsymbol{V_{j+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_j} - \boldsymbol{V_{j+1}})(s)|$ is small enough

  At termination: $\boldsymbol{V_j} = (\boldsymbol{B}^{G(\boldsymbol{V_j})})^i(\boldsymbol{V_j}) = \boldsymbol{V_{j+1}}$ for all $i = 0, 1, 2, \ldots$

Policy Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $j = 0, 1, 2, \ldots$, calculate in each iteration:

$$\text{Deterministic Policy } \pi_{j+1} = G(\boldsymbol{V_j})$$

$$\text{Value Function } \boldsymbol{V_{j+1}} = \lim_{i \to \infty} (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j})$$

- Stop when $d(\boldsymbol{V_j}, \boldsymbol{V_{j+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_j} - \boldsymbol{V_{j+1}})(s)|$ is small enough

  At termination: $\boldsymbol{V_j} = (\boldsymbol{B}^{G(\boldsymbol{V_j})})^i(\boldsymbol{V_j}) = \boldsymbol{V_{j+1}}$ for all $i = 0, 1, 2, \ldots$

Specializing this to $i = 1$, we have for all $s \in \mathcal{N}$:

$$\boldsymbol{V_j}(s) = \boldsymbol{B}^{G(\boldsymbol{V_j})}(\boldsymbol{V_j})(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \boldsymbol{V_j}(s')\}$$

Policy Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $j = 0, 1, 2, \ldots$, calculate in each iteration:

$$\text{Deterministic Policy } \pi_{j+1} = G(\boldsymbol{V_j})$$

$$\text{Value Function } \boldsymbol{V_{j+1}} = \lim_{i \to \infty} (\boldsymbol{B}^{\pi_{j+1}})^i (\boldsymbol{V_j})$$

- Stop when $d(\boldsymbol{V_j}, \boldsymbol{V_{j+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_j} - \boldsymbol{V_{j+1}})(s)|$ is small enough

  At termination: $\boldsymbol{V_j} = (\boldsymbol{B}^{G(\boldsymbol{V_j})})^i (\boldsymbol{V_j}) = \boldsymbol{V_{j+1}}$ for all $i = 0, 1, 2, \ldots$

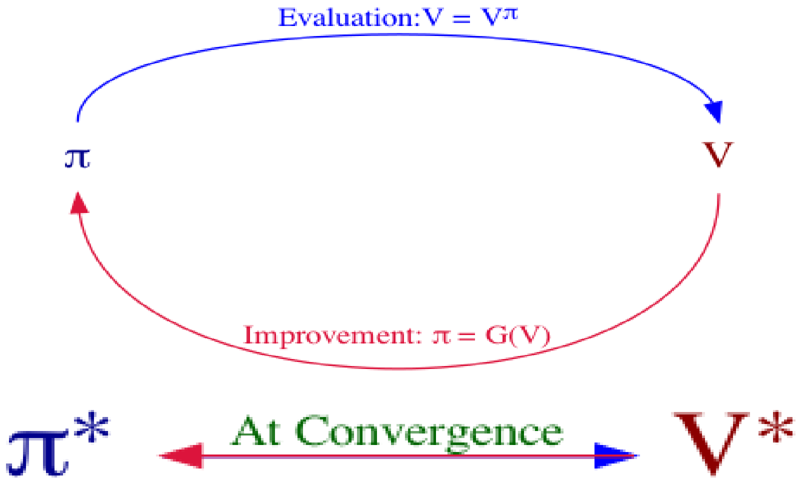
Specializing this to $i = 1$, we have for all $s \in \mathcal{N}$:

$$\boldsymbol{V_j}(s) = \boldsymbol{B}^{G(\boldsymbol{V_j})}(\boldsymbol{V_j})(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \boldsymbol{V_j}(s')\}$$

This means $\boldsymbol{V_j}$ satisfies the MDP Bellman Optimality Equation and so,

$$\boldsymbol{V_j} = \boldsymbol{V}^{\pi_j} = \boldsymbol{V}^*$$

Policy Iteration algorithm



Policy Iteration Convergence Theorem

Theorem (Policy Iteration Convergence Theorem)

For a Finite MDP with $|\mathcal{N}| = m$ and $\gamma < 1$, Policy Iteration algorithm converges to the Optimal Value Function $\boldsymbol{V}^* \in \mathbb{R}^m$ along with a Deterministic Optimal Policy $\pi_D^* : \mathcal{N} \rightarrow \mathcal{A}$, no matter which Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$ we start the algorithm with.

Policy Iteration Convergence Theorem

Theorem (Policy Iteration Convergence Theorem)

For a Finite MDP with $|\mathcal{N}| = m$ and $\gamma < 1$, Policy Iteration algorithm converges to the Optimal Value Function $\boldsymbol{V}^* \in \mathbb{R}^m$ along with a Deterministic Optimal Policy $\pi_D^* : \mathcal{N} \rightarrow \mathcal{A}$, no matter which Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$ we start the algorithm with.

Running time of Policy Improvement is $O(m^2 k)$ where $|\mathcal{N}| = m, |\mathcal{A}| = k$

Policy Iteration Convergence Theorem

Theorem (Policy Iteration Convergence Theorem)

For a Finite MDP with $|\mathcal{N}| = m$ and $\gamma < 1$, Policy Iteration algorithm converges to the Optimal Value Function $\boldsymbol{V}^* \in \mathbb{R}^m$ along with a Deterministic Optimal Policy $\pi_D^* : \mathcal{N} \rightarrow \mathcal{A}$, no matter which Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$ we start the algorithm with.

Running time of Policy Improvement is $O(m^2 k)$ where $|\mathcal{N}| = m, |\mathcal{A}| = k$

Running time of each iteration of Policy Evaluation is $O(m^2 k)$

Bellman Optimality Operator

Bellman Optimality Operator

- Tweak the definition of Greedy Policy Function ($\arg\max$ to $\max$)

Bellman Optimality Operator

- Tweak the definition of Greedy Policy Function ($\arg\max$ to $\max$)
- *Bellman Optimality Operator* $\boldsymbol{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$ defined as:

$$\boldsymbol{B}^*(\boldsymbol{V})(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}(s')\}$$

Bellman Optimality Operator

- Tweak the definition of Greedy Policy Function ($\arg\max$ to $\max$)
- *Bellman Optimality Operator* $\boldsymbol{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$ defined as:

$$\boldsymbol{B}^*(\boldsymbol{V})(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \boldsymbol{V}(s')\}$$

- Think of this as a non-linear transformation of a VF vector $\boldsymbol{V} \in \mathbb{R}^m$

Bellman Optimality Operator

- Tweak the definition of Greedy Policy Function (arg max to max)
- *Bellman Optimality Operator* $\boldsymbol{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$ defined as:

$$\boldsymbol{B}^*(\boldsymbol{V})(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}(s')\}$$

- Think of this as a non-linear transformation of a VF vector $\boldsymbol{V} \in \mathbb{R}^m$
- The action a producing the max is the action prescribed by $G(\boldsymbol{V})$. So,

$$\boldsymbol{B}^{G(\boldsymbol{V})}(\boldsymbol{V}) = \boldsymbol{B}^*(\boldsymbol{V}) \text{ for all } \boldsymbol{V} \in \mathbb{R}^m$$

Bellman Optimality Operator

- Tweak the definition of Greedy Policy Function (arg max to max)
- *Bellman Optimality Operator* $\boldsymbol{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$ defined as:

$$\boldsymbol{B}^*(\boldsymbol{V})(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \boldsymbol{V}(s')\}$$

- Think of this as a non-linear transformation of a VF vector $\boldsymbol{V} \in \mathbb{R}^m$
- The action a producing the max is the action prescribed by $G(\boldsymbol{V})$. So,

$$\boldsymbol{B}^{G(\boldsymbol{V})}(\boldsymbol{V}) = \boldsymbol{B}^*(\boldsymbol{V}) \text{ for all } \boldsymbol{V} \in \mathbb{R}^m$$

- Specializing $\boldsymbol{V}$ to be the Value Function $\boldsymbol{V}^\pi$ for a policy π, we get:

$$\boldsymbol{B}^{G(\boldsymbol{V}^\pi)}(\boldsymbol{V}^\pi) = \boldsymbol{B}^*(\boldsymbol{V}^\pi)$$

Bellman Optimality Operator

- Tweak the definition of Greedy Policy Function (arg max to max)
- *Bellman Optimality Operator* $\boldsymbol{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$ defined as:

$$\boldsymbol{B}^*(\boldsymbol{V})(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}(s')\}$$

- Think of this as a non-linear transformation of a VF vector $\boldsymbol{V} \in \mathbb{R}^m$
- The action a producing the max is the action prescribed by $G(\boldsymbol{V})$. So,

$$\boldsymbol{B}^{G(\boldsymbol{V})}(\boldsymbol{V}) = \boldsymbol{B}^*(\boldsymbol{V}) \text{ for all } \boldsymbol{V} \in \mathbb{R}^m$$

- Specializing $\boldsymbol{V}$ to be the Value Function $\boldsymbol{V}^\pi$ for a policy π, we get:

$$\boldsymbol{B}^{G(\boldsymbol{V}^\pi)}(\boldsymbol{V}^\pi) = \boldsymbol{B}^*(\boldsymbol{V}^\pi)$$

- This is the 1st stage of Policy Evaluation with improved policy $G(\boldsymbol{V}^\pi)$

Fixed-Point of Bellman Optimality Operator

Fixed-Point of Bellman Optimality Operator

- $\boldsymbol{B}^\pi$ was motivated by the MDP Bellman Policy Equation

Fixed-Point of Bellman Optimality Operator

- $\boldsymbol{B}^\pi$ was motivated by the MDP Bellman Policy Equation
- Similarly, $\boldsymbol{B}^*$ is motivated by the MDP Bellman Optimality Equation:

$$\boldsymbol{V}^*(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}^*(s')\} \text{ for all } s \in \mathcal{N}$$

Fixed-Point of Bellman Optimality Operator

- $\boldsymbol{B}^\pi$ was motivated by the MDP Bellman Policy Equation
- Similarly, $\boldsymbol{B}^*$ is motivated by the MDP Bellman Optimality Equation:

$$\boldsymbol{V}^*(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s, a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s, a, s') \cdot \boldsymbol{V}^*(s')\} \text{ for all } s \in \mathcal{N}$$

- So we can express the MDP Bellman Optimality Equation neatly as:

$$\boldsymbol{V}^* = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

Fixed-Point of Bellman Optimality Operator

- $\boldsymbol{B}^\pi$ was motivated by the MDP Bellman Policy Equation
- Similarly, $\boldsymbol{B}^*$ is motivated by the MDP Bellman Optimality Equation:

$$\boldsymbol{V}^*(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}^*(s')\} \text{ for all } s \in \mathcal{N}$$

- So we can express the MDP Bellman Optimality Equation neatly as:

$$\boldsymbol{V}^* = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

- Therefore, $\boldsymbol{V}^* \in \mathbb{R}^m$ is a Fixed-Point of $\boldsymbol{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$

Fixed-Point of Bellman Optimality Operator

- $\boldsymbol{B}^{\pi}$ was motivated by the MDP Bellman Policy Equation
- Similarly, $\boldsymbol{B}^*$ is motivated by the MDP Bellman Optimality Equation:

$$\boldsymbol{V}^*(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}^*(s')\} \text{ for all } s \in \mathcal{N}$$

- So we can express the MDP Bellman Optimality Equation neatly as:

$$\boldsymbol{V}^* = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

- Therefore, $\boldsymbol{V}^* \in \mathbb{R}^m$ is a Fixed-Point of $\boldsymbol{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$
- We want to prove that $\boldsymbol{B}^*$ is a contraction function (under L^{∞} norm)

Fixed-Point of Bellman Optimality Operator

- $\boldsymbol{B}^\pi$ was motivated by the MDP Bellman Policy Equation
- Similarly, $\boldsymbol{B}^*$ is motivated by the MDP Bellman Optimality Equation:

$$\boldsymbol{V}^*(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}^*(s')\} \text{ for all } s \in \mathcal{N}$$

- So we can express the MDP Bellman Optimality Equation neatly as:

$$\boldsymbol{V}^* = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

- Therefore, $\boldsymbol{V}^* \in \mathbb{R}^m$ is a Fixed-Point of $\boldsymbol{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$
- We want to prove that $\boldsymbol{B}^*$ is a contraction function (under L^∞ norm)
- So we can take advantage of Banach Fixed-Point Theorem

Fixed-Point of Bellman Optimality Operator

- $\boldsymbol{B}^\pi$ was motivated by the MDP Bellman Policy Equation
- Similarly, $\boldsymbol{B}^*$ is motivated by the MDP Bellman Optimality Equation:

$$\boldsymbol{V}^*(s) = \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot \boldsymbol{V}^*(s')\} \text{ for all } s \in \mathcal{N}$$

- So we can express the MDP Bellman Optimality Equation neatly as:

$$\boldsymbol{V}^* = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

- Therefore, $\boldsymbol{V}^* \in \mathbb{R}^m$ is a Fixed-Point of $\boldsymbol{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$
- We want to prove that $\boldsymbol{B}^*$ is a contraction function (under L^∞ norm)
- So we can take advantage of Banach Fixed-Point Theorem
- And solve the Control problem by iterative applications of $\boldsymbol{B}^*$

Proof that B^* is a contraction

Proof that B^* is a contraction

- We need to utilize two key properties of $\boldsymbol{B}^*$

$$\text{Monotonicity Property: } \boldsymbol{X} \geq \boldsymbol{Y} \Rightarrow \boldsymbol{B}^*(\boldsymbol{X}) \geq \boldsymbol{B}^*(\boldsymbol{Y})$$

$$\text{Constant Shift Property: } \boldsymbol{B}^*(\boldsymbol{X} + c) = \boldsymbol{B}^*(\boldsymbol{X}) + \gamma c$$

Proof that B^* is a contraction

- We need to utilize two key properties of $\boldsymbol{B}^*$

$$\text{Monotonicity Property: } \boldsymbol{X} \geq \boldsymbol{Y} \Rightarrow \boldsymbol{B}^*(\boldsymbol{X}) \geq \boldsymbol{B}^*(\boldsymbol{Y})$$

$$\text{Constant Shift Property: } \boldsymbol{B}^*(\boldsymbol{X} + c) = \boldsymbol{B}^*(\boldsymbol{X}) + \gamma c$$

- With these two properties, we can prove that:

$$\max_{s \in \mathcal{N}} |(\boldsymbol{B}^*(\boldsymbol{X}) - \boldsymbol{B}^*(\boldsymbol{Y}))(s)| \leq \gamma \cdot \max_{s \in \mathcal{N}} |(\boldsymbol{X} - \boldsymbol{Y})(s)|$$

Proof that B^* is a contraction

- We need to utilize two key properties of $\boldsymbol{B}^*$

$$\text{Monotonicity Property: } \boldsymbol{X} \geq \boldsymbol{Y} \Rightarrow \boldsymbol{B}^*(\boldsymbol{X}) \geq \boldsymbol{B}^*(\boldsymbol{Y})$$

$$\text{Constant Shift Property: } \boldsymbol{B}^*(\boldsymbol{X} + c) = \boldsymbol{B}^*(\boldsymbol{X}) + \gamma c$$

- With these two properties, we can prove that:

$$\max_{s \in \mathcal{N}} |(\boldsymbol{B}^*(\boldsymbol{X}) - \boldsymbol{B}^*(\boldsymbol{Y}))(s)| \leq \gamma \cdot \max_{s \in \mathcal{N}} |(\boldsymbol{X} - \boldsymbol{Y})(s)|$$

Proof that B^* is a contraction

- We need to utilize two key properties of $\boldsymbol{B}^*$

$$\text{Monotonicity Property: } \boldsymbol{X} \geq \boldsymbol{Y} \Rightarrow \boldsymbol{B}^*(\boldsymbol{X}) \geq \boldsymbol{B}^*(\boldsymbol{Y})$$

$$\text{Constant Shift Property: } \boldsymbol{B}^*(\boldsymbol{X} + c) = \boldsymbol{B}^*(\boldsymbol{X}) + \gamma c$$

- With these two properties, we can prove that:

$$\max_{s \in \mathcal{N}} |(\boldsymbol{B}^*(\boldsymbol{X}) - \boldsymbol{B}^*(\boldsymbol{Y}))(s)| \leq \gamma \cdot \max_{s \in \mathcal{N}} |(\boldsymbol{X} - \boldsymbol{Y})(s)|$$

Theorem (Value Iteration Convergence Theorem)

For a Finite MDP with $|\mathcal{N}| = m$ and $\gamma < 1$, if $\boldsymbol{V}^* \in \mathbb{R}^m$ is the Optimal Value Function, then $\boldsymbol{V}^*$ is the unique Fixed-Point of the Bellman Optimality Operator $\boldsymbol{B}^* : \mathbb{R}^m \rightarrow \mathbb{R}^m$, and

$$\lim_{i \rightarrow \infty} (\boldsymbol{B}^*)^i(\boldsymbol{V_0}) \rightarrow \boldsymbol{V}^* \textit{ for all starting Value Functions } \boldsymbol{V_0} \in \mathbb{R}^m$$

Value Iteration algorithm

Value Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$

Value Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \ldots$, calculate in each iteration:

$$\boldsymbol{V_{i+1}}(s) = \boldsymbol{B}^*(\boldsymbol{V_i})(s) \text{ for all } s \in \mathcal{N}$$

Value Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \ldots$, calculate in each iteration:

$$\boldsymbol{V_{i+1}}(s) = \boldsymbol{B}^*(\boldsymbol{V_i})(s) \text{ for all } s \in \mathcal{N}$$

- Stop when $d(\boldsymbol{V_i}, \boldsymbol{V_{i+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_i} - \boldsymbol{V_{i+1}})(s)|$ is small enough

Value Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \ldots$, calculate in each iteration:

$$\boldsymbol{V_{i+1}}(s) = \boldsymbol{B}^*(\boldsymbol{V_i})(s) \text{ for all } s \in \mathcal{N}$$

- Stop when $d(\boldsymbol{V_i}, \boldsymbol{V_{i+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_i} - \boldsymbol{V_{i+1}})(s)|$ is small enough

Value Iteration algorithm

- Start with any Value Function $\boldsymbol{V_0} \in \mathbb{R}^m$
- Iterating over $i = 0, 1, 2, \ldots$, calculate in each iteration:

$$\boldsymbol{V_{i+1}}(s) = \boldsymbol{B}^*(\boldsymbol{V_i})(s) \text{ for all } s \in \mathcal{N}$$

- Stop when $d(\boldsymbol{V_i}, \boldsymbol{V_{i+1}}) = \max_{s \in \mathcal{N}} |(\boldsymbol{V_i} - \boldsymbol{V_{i+1}})(s)|$ is small enough

Running time of each iteration of Value Iteration is $O(m^2 k)$ where $|\mathcal{N}| = m$ and $|\mathcal{A}| = k$

Optimal Policy from Optimal Value Function

Optimal Policy from Optimal Value Function

- Note that Value Iteration does not deal with any policy (only VFs)

Optimal Policy from Optimal Value Function

- Note that Value Iteration does not deal with any policy (only VFs)
- Extract Optimal Policy π^* from Optimal VF V^* such that $\boldsymbol{V}^{\pi^*} = \boldsymbol{V}^*$

Optimal Policy from Optimal Value Function

- Note that Value Iteration does not deal with any policy (only VFs)
- Extract Optimal Policy π^* from Optimal VF V^* such that $\boldsymbol{V}^{\pi^*} = \boldsymbol{V}^*$
- Use Greedy Policy function G. We know:

$$\boldsymbol{B}^{G(\boldsymbol{V})}(\boldsymbol{V}) = \boldsymbol{B}^*(\boldsymbol{V}) \text{ for all } \boldsymbol{V} \in \mathbb{R}^m$$

Optimal Policy from Optimal Value Function

- Note that Value Iteration does not deal with any policy (only VFs)
- Extract Optimal Policy π^* from Optimal VF V^* such that $\boldsymbol{V}^{\pi^*} = \boldsymbol{V}^*$
- Use Greedy Policy function G. We know:

$$\boldsymbol{B}^{G(\boldsymbol{V})}(\boldsymbol{V}) = \boldsymbol{B}^*(\boldsymbol{V}) \text{ for all } \boldsymbol{V} \in \mathbb{R}^m$$

- Specializing $\boldsymbol{V}$ to $\boldsymbol{V}^*$, we get:

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

Optimal Policy from Optimal Value Function

- Note that Value Iteration does not deal with any policy (only VFs)
- Extract Optimal Policy π^* from Optimal VF V^* such that $\boldsymbol{V}^{\pi^*} = \boldsymbol{V}^*$
- Use Greedy Policy function G. We know:

$$\boldsymbol{B}^{G(\boldsymbol{V})}(\boldsymbol{V}) = \boldsymbol{B}^*(\boldsymbol{V}) \text{ for all } \boldsymbol{V} \in \mathbb{R}^m$$

- Specializing $\boldsymbol{V}$ to $\boldsymbol{V}^*$, we get:

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

- But we know $\boldsymbol{V}^*$ is the Fixed-Point of $\boldsymbol{B}^*$, i.e., $\boldsymbol{B}^*(\boldsymbol{V}^*) = \boldsymbol{V}^*$. So,

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{V}^*$$

Optimal Policy from Optimal Value Function

- Note that Value Iteration does not deal with any policy (only VFs)
- Extract Optimal Policy π^* from Optimal VF V^* such that $\boldsymbol{V}^{\pi^*} = \boldsymbol{V}^*$
- Use Greedy Policy function G. We know:

$$\boldsymbol{B}^{G(\boldsymbol{V})}(\boldsymbol{V}) = \boldsymbol{B}^*(\boldsymbol{V}) \text{ for all } \boldsymbol{V} \in \mathbb{R}^m$$

- Specializing $\boldsymbol{V}$ to $\boldsymbol{V}^*$, we get:

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

- But we know $\boldsymbol{V}^*$ is the Fixed-Point of $\boldsymbol{B}^*$, i.e., $\boldsymbol{B}^*(\boldsymbol{V}^*) = \boldsymbol{V}^*$. So,

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{V}^*$$

- So $\boldsymbol{V}^*$ is the Fixed-Point of the Bellman Policy Operator $\boldsymbol{B}^{G(\boldsymbol{V}^*)}$

Optimal Policy from Optimal Value Function

- Note that Value Iteration does not deal with any policy (only VFs)
- Extract Optimal Policy π^* from Optimal VF V^* such that $\boldsymbol{V}^{\pi^*} = \boldsymbol{V}^*$
- Use Greedy Policy function G. We know:

$$\boldsymbol{B}^{G(\boldsymbol{V})}(\boldsymbol{V}) = \boldsymbol{B}^*(\boldsymbol{V}) \text{ for all } \boldsymbol{V} \in \mathbb{R}^m$$

- Specializing $\boldsymbol{V}$ to $\boldsymbol{V}^*$, we get:

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

- But we know $\boldsymbol{V}^*$ is the Fixed-Point of $\boldsymbol{B}^*$, i.e., $\boldsymbol{B}^*(\boldsymbol{V}^*) = \boldsymbol{V}^*$. So,

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{V}^*$$

- So $\boldsymbol{V}^*$ is the Fixed-Point of the Bellman Policy Operator $\boldsymbol{B}^{G(\boldsymbol{V}^*)}$
- But we know $\boldsymbol{B}^{G(\boldsymbol{V}^*)}$ has a unique Fixed-Point $(= \boldsymbol{V}^{G(\boldsymbol{V}^*)})$. So,

$$\boldsymbol{V}^{G(\boldsymbol{V}^*)} = \boldsymbol{V}^*$$

Optimal Policy from Optimal Value Function

- Note that Value Iteration does not deal with any policy (only VFs)
- Extract Optimal Policy π^* from Optimal VF V^* such that $\boldsymbol{V}^{\pi^*} = \boldsymbol{V}^*$
- Use Greedy Policy function G. We know:

$$\boldsymbol{B}^{G(\boldsymbol{V})}(\boldsymbol{V}) = \boldsymbol{B}^*(\boldsymbol{V}) \text{ for all } \boldsymbol{V} \in \mathbb{R}^m$$

- Specializing $\boldsymbol{V}$ to $\boldsymbol{V}^*$, we get:

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

- But we know $\boldsymbol{V}^*$ is the Fixed-Point of $\boldsymbol{B}^*$, i.e., $\boldsymbol{B}^*(\boldsymbol{V}^*) = \boldsymbol{V}^*$. So,

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{V}^*$$

- So $\boldsymbol{V}^*$ is the Fixed-Point of the Bellman Policy Operator $\boldsymbol{B}^{G(\boldsymbol{V}^*)}$
- But we know $\boldsymbol{B}^{G(\boldsymbol{V}^*)}$ has a unique Fixed-Point $(= \boldsymbol{V}^{G(\boldsymbol{V}^*)})$. So,

$$\boldsymbol{V}^{G(\boldsymbol{V}^*)} = \boldsymbol{V}^*$$

- Evaluating MDP with greedy policy extracted from $\boldsymbol{V}^*$ achieves $\boldsymbol{V}^*$

Optimal Policy from Optimal Value Function

- Note that Value Iteration does not deal with any policy (only VFs)
- Extract Optimal Policy π^* from Optimal VF V^* such that $\boldsymbol{V}^{\pi^*} = \boldsymbol{V}^*$
- Use Greedy Policy function G. We know:

$$\boldsymbol{B}^{G(\boldsymbol{V})}(\boldsymbol{V}) = \boldsymbol{B}^*(\boldsymbol{V}) \text{ for all } \boldsymbol{V} \in \mathbb{R}^m$$

- Specializing $\boldsymbol{V}$ to $\boldsymbol{V}^*$, we get:

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{B}^*(\boldsymbol{V}^*)$$

- But we know $\boldsymbol{V}^*$ is the Fixed-Point of $\boldsymbol{B}^*$, i.e., $\boldsymbol{B}^*(\boldsymbol{V}^*) = \boldsymbol{V}^*$. So,

$$\boldsymbol{B}^{G(\boldsymbol{V}^*)}(\boldsymbol{V}^*) = \boldsymbol{V}^*$$

- So $\boldsymbol{V}^*$ is the Fixed-Point of the Bellman Policy Operator $\boldsymbol{B}^{G(\boldsymbol{V}^*)}$
- But we know $\boldsymbol{B}^{G(\boldsymbol{V}^*)}$ has a unique Fixed-Point $(= \boldsymbol{V}^{G(\boldsymbol{V}^*)})$. So,

$$\boldsymbol{V}^{G(\boldsymbol{V}^*)} = \boldsymbol{V}^*$$

- Evaluating MDP with greedy policy extracted from $\boldsymbol{V}^*$ achieves $\boldsymbol{V}^*$
- So, $G(\boldsymbol{V}^*)$ is a (Deterministic) Optimal Policy

Value Function Progression in Policy Iteration

Value Function Progression in Policy Iteration

$$\pi_1 = G(\boldsymbol{V_0}) : \boldsymbol{V_0} \rightarrow \boldsymbol{B}^{\pi_1}(\boldsymbol{V_0}) \rightarrow \ldots (\boldsymbol{B}^{\pi_1})^i(\boldsymbol{V_0}) \rightarrow \ldots \boldsymbol{V}^{\pi_1} = \boldsymbol{V_1}$$

$$\pi_2 = G(\boldsymbol{V_1}) : \boldsymbol{V_1} \rightarrow \boldsymbol{B}^{\pi_2}(\boldsymbol{V_1}) \rightarrow \ldots (\boldsymbol{B}^{\pi_2})^i(\boldsymbol{V_1}) \rightarrow \ldots \boldsymbol{V}^{\pi_2} = \boldsymbol{V_2}$$

$$\ldots$$

$$\ldots$$

$$\pi_{j+1} = G(\boldsymbol{V_j}) : \boldsymbol{V_j} \rightarrow \boldsymbol{B}^{\pi_{j+1}}(\boldsymbol{V_j}) \rightarrow \ldots (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j}) \rightarrow \ldots \boldsymbol{V}^{\pi_{j+1}} = \boldsymbol{V}^*$$

Value Function Progression in Policy Iteration

$$\pi_1 = G(\boldsymbol{V_0}) : \boldsymbol{V_0} \rightarrow \boldsymbol{B}^{\pi_1}(\boldsymbol{V_0}) \rightarrow \ldots (\boldsymbol{B}^{\pi_1})^i(\boldsymbol{V_0}) \rightarrow \ldots \boldsymbol{V}^{\pi_1} = \boldsymbol{V_1}$$

$$\pi_2 = G(\boldsymbol{V_1}) : \boldsymbol{V_1} \rightarrow \boldsymbol{B}^{\pi_2}(\boldsymbol{V_1}) \rightarrow \ldots (\boldsymbol{B}^{\pi_2})^i(\boldsymbol{V_1}) \rightarrow \ldots \boldsymbol{V}^{\pi_2} = \boldsymbol{V_2}$$

$$\ldots$$

$$\ldots$$

$$\pi_{j+1} = G(\boldsymbol{V_j}) : \boldsymbol{V_j} \rightarrow \boldsymbol{B}^{\pi_{j+1}}(\boldsymbol{V_j}) \rightarrow \ldots (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j}) \rightarrow \ldots \boldsymbol{V}^{\pi_{j+1}} = \boldsymbol{V}^*$$

- Policy Evaluation and Policy Improvement alternate until convergence

Value Function Progression in Policy Iteration

$$\pi_1 = G(\boldsymbol{V_0}) : \boldsymbol{V_0} \rightarrow \boldsymbol{B}^{\pi_1}(\boldsymbol{V_0}) \rightarrow \ldots (\boldsymbol{B}^{\pi_1})^i(\boldsymbol{V_0}) \rightarrow \ldots \boldsymbol{V}^{\pi_1} = \boldsymbol{V_1}$$

$$\pi_2 = G(\boldsymbol{V_1}) : \boldsymbol{V_1} \rightarrow \boldsymbol{B}^{\pi_2}(\boldsymbol{V_1}) \rightarrow \ldots (\boldsymbol{B}^{\pi_2})^i(\boldsymbol{V_1}) \rightarrow \ldots \boldsymbol{V}^{\pi_2} = \boldsymbol{V_2}$$

$$\ldots$$

$$\ldots$$

$$\pi_{j+1} = G(\boldsymbol{V_j}) : \boldsymbol{V_j} \rightarrow \boldsymbol{B}^{\pi_{j+1}}(\boldsymbol{V_j}) \rightarrow \ldots (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j}) \rightarrow \ldots \boldsymbol{V}^{\pi_{j+1}} = \boldsymbol{V}^*$$

- Policy Evaluation and Policy Improvement alternate until convergence
- In the process, they simultaneously compete and try to be consistent

Value Function Progression in Policy Iteration

$$\pi_1 = G(\boldsymbol{V_0}) : \boldsymbol{V_0} \rightarrow \boldsymbol{B}^{\pi_1}(\boldsymbol{V_0}) \rightarrow \ldots (\boldsymbol{B}^{\pi_1})^i(\boldsymbol{V_0}) \rightarrow \ldots \boldsymbol{V}^{\pi_1} = \boldsymbol{V_1}$$

$$\pi_2 = G(\boldsymbol{V_1}) : \boldsymbol{V_1} \rightarrow \boldsymbol{B}^{\pi_2}(\boldsymbol{V_1}) \rightarrow \ldots (\boldsymbol{B}^{\pi_2})^i(\boldsymbol{V_1}) \rightarrow \ldots \boldsymbol{V}^{\pi_2} = \boldsymbol{V_2}$$

$$\ldots$$

$$\ldots$$

$$\pi_{j+1} = G(\boldsymbol{V_j}) : \boldsymbol{V_j} \rightarrow \boldsymbol{B}^{\pi_{j+1}}(\boldsymbol{V_j}) \rightarrow \ldots (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j}) \rightarrow \ldots \boldsymbol{V}^{\pi_{j+1}} = \boldsymbol{V}^*$$

- Policy Evaluation and Policy Improvement alternate until convergence
- In the process, they simultaneously compete and try to be consistent
- There are actually two notions of consistency:

Value Function Progression in Policy Iteration

$$\pi_1 = G(\boldsymbol{V_0}) : \boldsymbol{V_0} \rightarrow \boldsymbol{B}^{\pi_1}(\boldsymbol{V_0}) \rightarrow \ldots (\boldsymbol{B}^{\pi_1})^i(\boldsymbol{V_0}) \rightarrow \ldots \boldsymbol{V}^{\pi_1} = \boldsymbol{V_1}$$

$$\pi_2 = G(\boldsymbol{V_1}) : \boldsymbol{V_1} \rightarrow \boldsymbol{B}^{\pi_2}(\boldsymbol{V_1}) \rightarrow \ldots (\boldsymbol{B}^{\pi_2})^i(\boldsymbol{V_1}) \rightarrow \ldots \boldsymbol{V}^{\pi_2} = \boldsymbol{V_2}$$

$$\ldots$$

$$\ldots$$

$$\pi_{j+1} = G(\boldsymbol{V_j}) : \boldsymbol{V_j} \rightarrow \boldsymbol{B}^{\pi_{j+1}}(\boldsymbol{V_j}) \rightarrow \ldots (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j}) \rightarrow \ldots \boldsymbol{V}^{\pi_{j+1}} = \boldsymbol{V}^*$$

- Policy Evaluation and Policy Improvement alternate until convergence
- In the process, they simultaneously compete and try to be consistent
- There are actually two notions of consistency:
  - VF $\boldsymbol{V}$ being consistent with/close to VF $\boldsymbol{V}^\pi$ of the policy π.

Value Function Progression in Policy Iteration

$$\pi_1 = G(\boldsymbol{V_0}) : \boldsymbol{V_0} \rightarrow \boldsymbol{B}^{\pi_1}(\boldsymbol{V_0}) \rightarrow \ldots (\boldsymbol{B}^{\pi_1})^i(\boldsymbol{V_0}) \rightarrow \ldots \boldsymbol{V}^{\pi_1} = \boldsymbol{V_1}$$

$$\pi_2 = G(\boldsymbol{V_1}) : \boldsymbol{V_1} \rightarrow \boldsymbol{B}^{\pi_2}(\boldsymbol{V_1}) \rightarrow \ldots (\boldsymbol{B}^{\pi_2})^i(\boldsymbol{V_1}) \rightarrow \ldots \boldsymbol{V}^{\pi_2} = \boldsymbol{V_2}$$

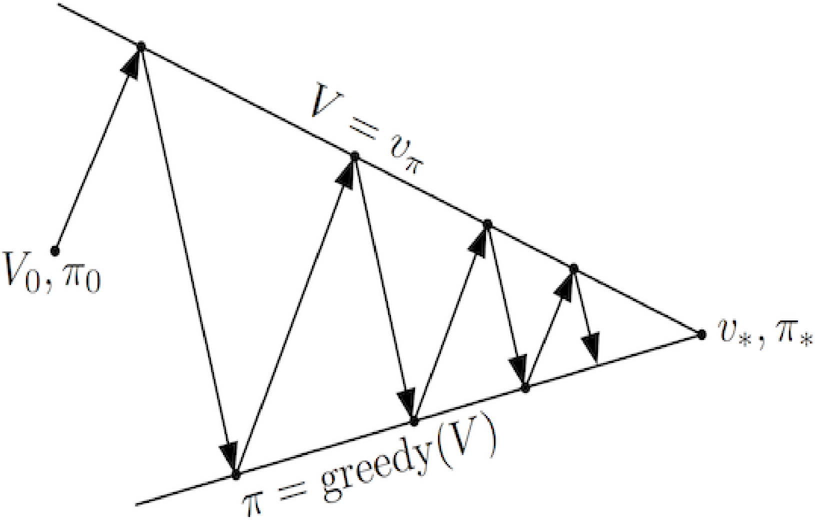
$$\ldots$$

$$\ldots$$

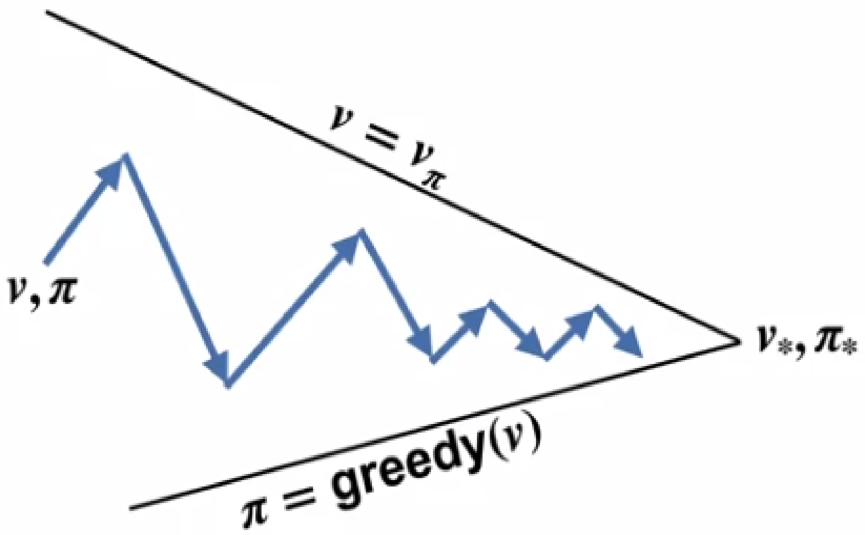
$$\pi_{j+1} = G(\boldsymbol{V_j}) : \boldsymbol{V_j} \rightarrow \boldsymbol{B}^{\pi_{j+1}}(\boldsymbol{V_j}) \rightarrow \ldots (\boldsymbol{B}^{\pi_{j+1}})^i(\boldsymbol{V_j}) \rightarrow \ldots \boldsymbol{V}^{\pi_{j+1}} = \boldsymbol{V}^*$$

- Policy Evaluation and Policy Improvement alternate until convergence
- In the process, they simultaneously compete and try to be consistent
- There are actually two notions of consistency:
  - VF $\boldsymbol{V}$ being consistent with/close to VF $\boldsymbol{V}^\pi$ of the policy π.
  - π being consistent with/close to Greedy Policy $G(\boldsymbol{V})$ of VF $\boldsymbol{V}$.

Policy Iteration



Generalized Policy Iteration (GPI)



Value Iteration and Reinforcement Learning as GPI

Value Iteration and Reinforcement Learning as GPI

- Value Iteration takes only one step of Policy Evaluation

$$\pi_1 = G(\boldsymbol{V_0}) : \ \boldsymbol{V_0} \rightarrow \boldsymbol{B}^{\pi_1}(\boldsymbol{V_0}) = \boldsymbol{V_1}$$

$$\pi_2 = G(\boldsymbol{V_1}) : \ \boldsymbol{V_1} \rightarrow \boldsymbol{B}^{\pi_2}(\boldsymbol{V_1}) = \boldsymbol{V_2}$$

$$\dots$$

$$\dots$$

$$\pi_{j+1} = G(\boldsymbol{V_j}) : \ \boldsymbol{V_j} \rightarrow \boldsymbol{B}^{\pi_{j+1}}(\boldsymbol{V_j}) = \boldsymbol{V}^*$$

Value Iteration and Reinforcement Learning as GPI

- Value Iteration takes only one step of Policy Evaluation

$$\pi_1 = G(\boldsymbol{V_0}) : \boldsymbol{V_0} \rightarrow \boldsymbol{B}^{\pi_1}(\boldsymbol{V_0}) = \boldsymbol{V_1}$$

$$\pi_2 = G(\boldsymbol{V_1}) : \boldsymbol{V_1} \rightarrow \boldsymbol{B}^{\pi_2}(\boldsymbol{V_1}) = \boldsymbol{V_2}$$

$$\ldots$$

$$\ldots$$

$$\pi_{j+1} = G(\boldsymbol{V_j}) : \boldsymbol{V_j} \rightarrow \boldsymbol{B}^{\pi_{j+1}}(\boldsymbol{V_j}) = \boldsymbol{V}^*$$

- RL updates either a subset of states or just one state at a time

Value Iteration and Reinforcement Learning as GPI

- Value Iteration takes only one step of Policy Evaluation

$$\pi_1 = G(\boldsymbol{V_0}) : \boldsymbol{V_0} \rightarrow \boldsymbol{B}^{\pi_1}(\boldsymbol{V_0}) = \boldsymbol{V_1}$$

$$\pi_2 = G(\boldsymbol{V_1}) : \boldsymbol{V_1} \rightarrow \boldsymbol{B}^{\pi_2}(\boldsymbol{V_1}) = \boldsymbol{V_2}$$

$$\ldots$$

$$\ldots$$

$$\pi_{j+1} = G(\boldsymbol{V_j}) : \boldsymbol{V_j} \rightarrow \boldsymbol{B}^{\pi_{j+1}}(\boldsymbol{V_j}) = \boldsymbol{V}^*$$

- RL updates either a subset of states or just one state at a time
- Large-scale RL updates function approximations of a VF

Value Iteration and Reinforcement Learning as GPI

- Value Iteration takes only one step of Policy Evaluation

$$\pi_1 = G(\boldsymbol{V_0}) : \boldsymbol{V_0} \rightarrow \boldsymbol{B}^{\pi_1}(\boldsymbol{V_0}) = \boldsymbol{V_1}$$

$$\pi_2 = G(\boldsymbol{V_1}) : \boldsymbol{V_1} \rightarrow \boldsymbol{B}^{\pi_2}(\boldsymbol{V_1}) = \boldsymbol{V_2}$$

$$\ldots$$

$$\ldots$$

$$\pi_{j+1} = G(\boldsymbol{V_j}) : \boldsymbol{V_j} \rightarrow \boldsymbol{B}^{\pi_{j+1}}(\boldsymbol{V_j}) = \boldsymbol{V}^*$$

- RL updates either a subset of states or just one state at a time
- Large-scale RL updates function approximations of a VF
- These can be thought of as *partial* Policy Evaluation/Improvement

Asynchronous Dynamic Programming

Asynchronous Dynamic Programming

The DP algorithms we've covered are qualified as *Synchronous DP*:

Asynchronous Dynamic Programming

The DP algorithms we've covered are qualified as *Synchronous DP*:

- *All* states' values are updated in each iteration
- "Simultaneous" state updates implemented by updating a copy of VF

Asynchronous Dynamic Programming

The DP algorithms we've covered are qualified as *Synchronous DP*:

- *All* states' values are updated in each iteration
- "Simultaneous" state updates implemented by updating a copy of VF

Asynchronous DP can update subset of states, or update in any order

Asynchronous Dynamic Programming

The DP algorithms we've covered are qualified as *Synchronous DP*:

- *All* states' values are updated in each iteration
- "Simultaneous" state updates implemented by updating a copy of VF

Asynchronous DP can update subset of states, or update in any order

1. *In-place* updates enable updated values to be used immediately

Asynchronous Dynamic Programming

The DP algorithms we've covered are qualified as *Synchronous DP*:

- *All* states' values are updated in each iteration
- "Simultaneous" state updates implemented by updating a copy of VF

Asynchronous DP can update subset of states, or update in any order

1. *In-place* updates enable updated values to be used immediately
2. *Prioritized Sweeping* keeps states sorted by their Value Function *gaps*

$$\text{Gaps } g(s) = |V(s) - \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot V(s')\}|$$

But this requires us to know the reverse transitions to resort queue

Asynchronous Dynamic Programming

The DP algorithms we've covered are qualified as *Synchronous DP*:

- *All* states' values are updated in each iteration
- "Simultaneous" state updates implemented by updating a copy of VF

Asynchronous DP can update subset of states, or update in any order

1. *In-place* updates enable updated values to be used immediately
2. *Prioritized Sweeping* keeps states sorted by their Value Function *gaps*

$$\text{Gaps } g(s) = |V(s) - \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot V(s')\}|$$

   But this requires us to know the reverse transitions to resort queue
3. *Real-Time Dynamic Programming (RTDP)* runs DP *while* the agent is experiencing real-time interaction with the environment

Asynchronous Dynamic Programming

The DP algorithms we've covered are qualified as *Synchronous DP*:

- *All* states' values are updated in each iteration
- "Simultaneous" state updates implemented by updating a copy of VF

Asynchronous DP can update subset of states, or update in any order

1. *In-place* updates enable updated values to be used immediately
2. *Prioritized Sweeping* keeps states sorted by their Value Function *gaps*

$$\text{Gaps } g(s) = |V(s) - \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot V(s')\}|$$

   But this requires us to know the reverse transitions to resort queue
3. *Real-Time Dynamic Programming (RTDP)* runs DP *while* the agent is experiencing real-time interaction with the environment
   - A state is updated when it is visited during the real-time interaction

Asynchronous Dynamic Programming

The DP algorithms we've covered are qualified as *Synchronous DP*:

- *All* states' values are updated in each iteration
- "Simultaneous" state updates implemented by updating a copy of VF

Asynchronous DP can update subset of states, or update in any order

1. *In-place* updates enable updated values to be used immediately
2. *Prioritized Sweeping* keeps states sorted by their Value Function *gaps*

$$\text{Gaps } g(s) = |V(s) - \max_{a \in \mathcal{A}} \{\mathcal{R}(s,a) + \gamma \cdot \sum_{s' \in \mathcal{N}} \mathcal{P}(s,a,s') \cdot V(s')\}|$$

   But this requires us to know the reverse transitions to resort queue
3. *Real-Time Dynamic Programming (RTDP)* runs DP *while* the agent is experiencing real-time interaction with the environment
   - A state is updated when it is visited during the real-time interaction
   - The choice of action is governed by real-time VF-extracted policy

Episodic MDPs with unique state visits

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:
  - All random sequences terminate within fixed time steps (episodic MDP)

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:
  - All random sequences terminate within fixed time steps (episodic MDP)
  - A state is encountered at most once in an episode

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:
  - All random sequences terminate within fixed time steps (episodic MDP)
  - A state is encountered at most once in an episode
- This can be conceptualized as a Directed Acyclic Graph (DAG)

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:
  - All random sequences terminate within fixed time steps (episodic MDP)
  - A state is encountered at most once in an episode
- This can be conceptualized as a Directed Acyclic Graph (DAG)
- Each node in the DAG is a (state, action) pair

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:
  - All random sequences terminate within fixed time steps (episodic MDP)
  - A state is encountered at most once in an episode
- This can be conceptualized as a Directed Acyclic Graph (DAG)
- Each node in the DAG is a (state, action) pair
- Prediction/Control solved by “backwards walk” from terminal nodes

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:
  - All random sequences terminate within fixed time steps (episodic MDP)
  - A state is encountered at most once in an episode
- This can be conceptualized as a Directed Acyclic Graph (DAG)
- Each node in the DAG is a (state, action) pair
- Prediction/Control solved by "backwards walk" from terminal nodes
- Bellman Equation enables simply *setting the VF* of a visited node

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:
  - All random sequences terminate within fixed time steps (episodic MDP)
  - A state is encountered at most once in an episode
- This can be conceptualized as a Directed Acyclic Graph (DAG)
- Each node in the DAG is a (state, action) pair
- Prediction/Control solved by "backwards walk" from terminal nodes
- Bellman Equation enables simply *setting the VF* of a visited node
- Avoids the expensive "iterate to convergence" method of classical DP

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:
  - All random sequences terminate within fixed time steps (episodic MDP)
  - A state is encountered at most once in an episode
- This can be conceptualized as a Directed Acyclic Graph (DAG)
- Each node in the DAG is a (state, action) pair
- Prediction/Control solved by "backwards walk" from terminal nodes
- Bellman Equation enables simply *setting the VF* of a visited node
- Avoids the expensive "iterate to convergence" method of classical DP
- States visited (and VFs set) in order of reverse Topological Sort

Episodic MDPs with unique state visits

- A fairly common specialization of MDPs enables great tractability:
  - All random sequences terminate within fixed time steps (episodic MDP)
  - A state is encountered at most once in an episode
- This can be conceptualized as a Directed Acyclic Graph (DAG)
- Each node in the DAG is a (state, action) pair
- Prediction/Control solved by "backwards walk" from terminal nodes
- Bellman Equation enables simply *setting the VF* of a visited node
- Avoids the expensive "iterate to convergence" method of classical DP
- States visited (and VFs set) in order of reverse Topological Sort
- Next we cover a special case of DAG MDPs: finite-horizon MDPs

Finite-Horizon MDPs

Finite-Horizon MDPs

- Finite-Horizon Markov Decision Processes are characterized by:

Finite-Horizon MDPs

- Finite-Horizon Markov Decision Processes are characterized by:
  - Each sequence terminates within a finite number of time steps T

Finite-Horizon MDPs

- Finite-Horizon Markov Decision Processes are characterized by:
    - Each sequence terminates within a finite number of time steps T
    - Each time step has it's own state space

Finite-Horizon MDPs

- Finite-Horizon Markov Decision Processes are characterized by:
  - Each sequence terminates within a finite number of time steps T
  - Each time step has it's own state space
- Denote states at time t as $\mathcal{S}_t$, terminal states as $\mathcal{T}_t$, non-terminal states as $\mathcal{N}_t = \mathcal{S}_t - \mathcal{T}_t$ (note: $\mathcal{N}_T = \emptyset$), actions as $\mathcal{A}_t$, rewards as $\mathcal{D}_t$

Finite-Horizon MDPs

- Finite-Horizon Markov Decision Processes are characterized by:
  - Each sequence terminates within a finite number of time steps T
  - Each time step has it's own state space
- Denote states at time t as $\mathcal{S}_t$, terminal states as $\mathcal{T}_t$, non-terminal states as $\mathcal{N}_t = \mathcal{S}_t - \mathcal{T}_t$ (note: $\mathcal{N}_T = \emptyset$), actions as $\mathcal{A}_t$, rewards as $\mathcal{D}_t$
- Augment each state to include time-index: augmented state is (t, s_t)

$$\text{Entire MDP's States } \mathcal{S} = \{(t, s_t) | t = 0, 1, \ldots, T, s_t \in \mathcal{S}_t\}$$

Finite-Horizon MDPs

- Finite-Horizon Markov Decision Processes are characterized by:
  - Each sequence terminates within a finite number of time steps T
  - Each time step has it's own state space
- Denote states at time t as $\mathcal{S}_t$, terminal states as $\mathcal{T}_t$, non-terminal states as $\mathcal{N}_t = \mathcal{S}_t - \mathcal{T}_t$ (note: $\mathcal{N}_T = \emptyset$), actions as $\mathcal{A}_t$, rewards as $\mathcal{D}_t$
- Augment each state to include time-index: augmented state is (t, s_t)

$$\text{Entire MDP's States } \mathcal{S} = \{(t, s_t) | t = 0, 1, \ldots, T, s_t \in \mathcal{S}_t\}$$

- Each t gets its own state-reward transition probability function

$$(\mathcal{P}_R)_t : \mathcal{N}_t \times \mathcal{A}_t \times \mathcal{D}_{t+1} \times \mathcal{S}_{t+1} \rightarrow [0, 1]$$

Finite-Horizon MDPs

- Finite-Horizon Markov Decision Processes are characterized by:
    - Each sequence terminates within a finite number of time steps T
    - Each time step has it's own state space
- Denote states at time t as $\mathcal{S}_t$, terminal states as $\mathcal{T}_t$, non-terminal states as $\mathcal{N}_t = \mathcal{S}_t - \mathcal{T}_t$ (note: $\mathcal{N}_T = \emptyset$), actions as $\mathcal{A}_t$, rewards as $\mathcal{D}_t$
- Augment each state to include time-index: augmented state is (t, s_t)

$$\text{Entire MDP's States } \mathcal{S} = \{(t, s_t) | t = 0, 1, \ldots, T, s_t \in \mathcal{S}_t\}$$

- Each t gets its own state-reward transition probability function

$$(\mathcal{P}_R)_t : \mathcal{N}_t \times \mathcal{A}_t \times \mathcal{D}_{t+1} \times \mathcal{S}_{t+1} \rightarrow [0, 1]$$

- Likewise, each t gets its own policy $\pi_t : \mathcal{N}_t \times \mathcal{A}_t \rightarrow [0, 1]$

Finite-Horizon MDPs

- Finite-Horizon Markov Decision Processes are characterized by:
  - Each sequence terminates within a finite number of time steps T
  - Each time step has it's own state space
- Denote states at time t as $\mathcal{S}_t$, terminal states as $\mathcal{T}_t$, non-terminal states as $\mathcal{N}_t = \mathcal{S}_t - \mathcal{T}_t$ (note: $\mathcal{N}_T = \emptyset$), actions as $\mathcal{A}_t$, rewards as $\mathcal{D}_t$
- Augment each state to include time-index: augmented state is (t, s_t)

$$\text{Entire MDP's States } \mathcal{S} = \{(t, s_t) | t = 0, 1, \ldots, T, s_t \in \mathcal{S}_t\}$$

- Each t gets its own state-reward transition probability function

$$(\mathcal{P}_R)_t : \mathcal{N}_t \times \mathcal{A}_t \times \mathcal{D}_{t+1} \times \mathcal{S}_{t+1} \rightarrow [0, 1]$$

- Likewise, each t gets its own policy $\pi_t : \mathcal{N}_t \times \mathcal{A}_t \rightarrow [0, 1]$
- An overall policy $\pi : \mathcal{N} \times \mathcal{A} \rightarrow [0, 1]$ composed of $(\pi_0, \pi_1, \ldots, \pi_{T-1})$

Backward Induction for Finite MRP with Finite-Horizon

Backward Induction for Finite MRP with Finite-Horizon

- VF for a given policy π can be represented by time-sequenced VFs

$$V_t^\pi : \mathcal{N}_t \rightarrow \mathbb{R}$$

Backward Induction for Finite MRP with Finite-Horizon

- VF for a given policy π can be represented by time-sequenced VFs

$$V_t^\pi : \mathcal{N}_t \rightarrow \mathbb{R}$$

- So Bellman Equation for π-implied Finite-Horizon MRP becomes:

$$V_t^\pi(s_t) = \sum_{s_{t+1} \in \mathcal{S}_{t+1}} \sum_{r_{t+1} \in \mathbb{D}_{t+1}} (\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^\pi(s_{t+1}))$$

$$(\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) = \sum_{a_t \in \mathcal{A}_t} \pi_t(s_t, a_t) \cdot (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1})$$

Backward Induction for Finite MRP with Finite-Horizon

- VF for a given policy π can be represented by time-sequenced VFs

$$V_t^\pi : \mathcal{N}_t \rightarrow \mathbb{R}$$

- So Bellman Equation for π-implied Finite-Horizon MRP becomes:

$$V_t^\pi(s_t) = \sum_{s_{t+1} \in \mathcal{S}_{t+1}} \sum_{r_{t+1} \in \mathbb{D}_{t+1}} (\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^\pi(s_{t+1}))$$

$$(\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) = \sum_{a_t \in \mathcal{A}_t} \pi_t(s_t, a_t) \cdot (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1})$$

- "Backward Induction" algorithm for *finite MRP* with Finite-Horizon

Backward Induction for Finite MRP with Finite-Horizon

- VF for a given policy π can be represented by time-sequenced VFs

$$V_t^\pi : \mathcal{N}_t \to \mathbb{R}$$

- So Bellman Equation for π-implied Finite-Horizon MRP becomes:

$$V_t^\pi(s_t) = \sum_{s_{t+1} \in \mathcal{S}_{t+1}} \sum_{r_{t+1} \in \mathbb{D}_{t+1}} (\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^\pi(s_{t+1}))$$

$$(\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) = \sum_{a_t \in \mathcal{A}_t} \pi_t(s_t, a_t) \cdot (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1})$$

- "Backward Induction" algorithm for *finite MRP* with Finite-Horizon
- Decrementing t from $T-1$ to 0, and calculating V_t^π from V_{t+1}^π

Backward Induction for Finite MRP with Finite-Horizon

- VF for a given policy π can be represented by time-sequenced VFs

$$V_t^\pi : \mathcal{N}_t \rightarrow \mathbb{R}$$

- So Bellman Equation for π-implied Finite-Horizon MRP becomes:

$$V_t^\pi(s_t) = \sum_{s_{t+1} \in \mathcal{S}_{t+1}} \sum_{r_{t+1} \in \mathbb{D}_{t+1}} (\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^\pi(s_{t+1}))$$

$$(\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) = \sum_{a_t \in \mathcal{A}_t} \pi_t(s_t, a_t) \cdot (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1})$$

- "Backward Induction" algorithm for *finite MRP* with Finite-Horizon
- Decrementing t from $T-1$ to 0, and calculating V_t^π from V_{t+1}^π
- Running time is $O(m^2 T)$ where $|\mathcal{N}_t|$ is $O(m)$

Backward Induction for Finite MRP with Finite-Horizon

- VF for a given policy π can be represented by time-sequenced VFs

$$V_t^\pi : \mathcal{N}_t \rightarrow \mathbb{R}$$

- So Bellman Equation for π-implied Finite-Horizon MRP becomes:

$$V_t^\pi(s_t) = \sum_{s_{t+1} \in \mathcal{S}_{t+1}} \sum_{r_{t+1} \in \mathbb{D}_{t+1}} (\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^\pi(s_{t+1}))$$

$$(\mathcal{P}_R^{\pi_t})_t(s_t, r_{t+1}, s_{t+1}) = \sum_{a_t \in \mathcal{A}_t} \pi_t(s_t, a_t) \cdot (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1})$$

- "Backward Induction" algorithm for *finite MRP* with Finite-Horizon
- Decrementing t from $T-1$ to 0, and calculating V_t^π from V_{t+1}^π
- Running time is $O(m^2 T)$ where $|\mathcal{N}_t|$ is $O(m)$
- $O(m^2 kT)$ to convert MDP to π-implied MRP ($|\mathcal{A}_t|$ is $O(k)$)

Backward Induction for Finite MDP with Finite-Horizon

Backward Induction for Finite MDP with Finite-Horizon

- Optimal VF V^* can be represented by time-sequenced Optimal VFs

$$V^*_t : \mathcal{N}_t \to \mathbb{R}$$

Backward Induction for Finite MDP with Finite-Horizon

- Optimal VF V^* can be represented by time-sequenced Optimal VFs

$$V_t^* : \mathcal{N}_t \rightarrow \mathbb{R}$$

- So MDP Bellman Optimality Equation becomes:

$$V_t^*(s_t) = \max_{a_t \in \mathcal{A}_t} \sum_{s_{t+1}} \sum_{r_{t+1}} (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^*(s_{t+1}))$$

Backward Induction for Finite MDP with Finite-Horizon

- Optimal VF V^* can be represented by time-sequenced Optimal VFs

$$V_t^* : \mathcal{N}_t \to \mathbb{R}$$

- So MDP Bellman Optimality Equation becomes:

$$V_t^*(s_t) = \max_{a_t \in \mathcal{A}_t} \sum_{s_{t+1}} \sum_{r_{t+1}} (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^*(s_{t+1}))$$

- "Backward Induction" (Control) for *finite MDP* with Finite-Horizon

Backward Induction for Finite MDP with Finite-Horizon

- Optimal VF V^* can be represented by time-sequenced Optimal VFs

$$V_t^* : \mathcal{N}_t \to \mathbb{R}$$

- So MDP Bellman Optimality Equation becomes:

$$V_t^*(s_t) = \max_{a_t \in \mathcal{A}_t} \sum_{s_{t+1}} \sum_{r_{t+1}} (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^*(s_{t+1}))$$

- "Backward Induction" (Control) for *finite MDP* with Finite-Horizon
- Decrementing t from $T-1$ to 0, and calculating V_t^* from V_{t+1}^*

Backward Induction for Finite MDP with Finite-Horizon

- Optimal VF V^* can be represented by time-sequenced Optimal VFs

$$V_t^* : \mathcal{N}_t \rightarrow \mathbb{R}$$

- So MDP Bellman Optimality Equation becomes:

$$V_t^*(s_t) = \max_{a_t \in \mathcal{A}_t} \sum_{s_{t+1}} \sum_{r_{t+1}} (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^*(s_{t+1}))$$

- "Backward Induction" (Control) for *finite MDP* with Finite-Horizon
- Decrementing t from $T-1$ to 0, and calculating V_t^* from V_{t+1}^*
- (Associated) Optimal (Deterministic) Policy $(\pi_D^*)_t : \mathcal{N}_t \rightarrow \mathcal{A}_t$ is

$$(\pi_D^*)_t(s_t) = \arg\max_{a_t \in \mathcal{A}_t} \sum_{s_{t+1}} \sum_{r_{t+1}} (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^*(s_{t+1}))$$

Backward Induction for Finite MDP with Finite-Horizon

- Optimal VF V^* can be represented by time-sequenced Optimal VFs

$$V_t^* : \mathcal{N}_t \rightarrow \mathbb{R}$$

- So MDP Bellman Optimality Equation becomes:

$$V_t^*(s_t) = \max_{a_t \in \mathcal{A}_t} \sum_{s_{t+1}} \sum_{r_{t+1}} (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^*(s_{t+1}))$$

- "Backward Induction" (Control) for *finite MDP* with Finite-Horizon
- Decrementing t from $T-1$ to 0, and calculating V_t^* from V_{t+1}^*
- (Associated) Optimal (Deterministic) Policy $(\pi_D^*)_t : \mathcal{N}_t \rightarrow \mathcal{A}_t$ is

$$(\pi_D^*)_t(s_t) = \arg\max_{a_t \in \mathcal{A}_t} \sum_{s_{t+1}} \sum_{r_{t+1}} (\mathcal{P}_R)_t(s_t, a_t, r_{t+1}, s_{t+1}) \cdot (r_{t+1} + \gamma \cdot V_{t+1}^*(s_{t+1}))$$

- Running time is $O(m^2kT)$ where $|\mathcal{N}_t|$ is $O(m)$ and $|\mathcal{A}_t|$ is $O(k)$

@abstractclass MarkovDecisionProcess

@abstractclass MarkovDecisionProcess

```
def optimal_vf_and_policy(
  steps: Sequence[StateActionMapping[S, A]],
  gamma: float
) -> Iterator[Tuple[
    Mapping[NonTerminal[S], float],
    FiniteDeterministicPolicy[S, A]
]]

def unwrap_finite_horizon_MDP(
  process: FiniteMarkovDecisionProcess[WithTime[S], A]
) -> Sequence[StateActionMapping[S, A]]

def finite_horizon_MDP(
  process: FiniteMarkovDecisionProcess[S, A],
  limit: int
) -> FiniteMarkovDecisionProcess[WithTime[S], A]
```

Dynamic Pricing for End-of-Life/End-of-Season

Dynamic Pricing for End-of-Life/End-of-Season

- Dynamic Pricing: Core to many businesses, flexing to supply/demand

Dynamic Pricing for End-of-Life/End-of-Season

- Dynamic Pricing: Core to many businesses, flexing to supply/demand
- We consider special case of products being sold at end of life/season

Dynamic Pricing for End-of-Life/End-of-Season

- Dynamic Pricing: Core to many businesses, flexing to supply/demand
- We consider special case of products being sold at end of life/season
- Assume we are T days from season-end and our inventory is M units

Dynamic Pricing for End-of-Life/End-of-Season

- Dynamic Pricing: Core to many businesses, flexing to supply/demand
- We consider special case of products being sold at end of life/season
- Assume we are T days from season-end and our inventory is M units
- Assume no more incoming inventory during these final T days

Dynamic Pricing for End-of-Life/End-of-Season

- Dynamic Pricing: Core to many businesses, flexing to supply/demand
- We consider special case of products being sold at end of life/season
- Assume we are T days from season-end and our inventory is M units
- Assume no more incoming inventory during these final T days
- Set prices daily to max *Expected Total Sales Revenue* over T days

Dynamic Pricing for End-of-Life/End-of-Season

- Dynamic Pricing: Core to many businesses, flexing to supply/demand
- We consider special case of products being sold at end of life/season
- Assume we are T days from season-end and our inventory is M units
- Assume no more incoming inventory during these final T days
- Set prices daily to max *Expected Total Sales Revenue* over T days
- Price for a given day picked from prices $P_1, P_2, \ldots, P_N \in \mathbb{R}$

Dynamic Pricing for End-of-Life/End-of-Season

- Dynamic Pricing: Core to many businesses, flexing to supply/demand
- We consider special case of products being sold at end of life/season
- Assume we are T days from season-end and our inventory is M units
- Assume no more incoming inventory during these final T days
- Set prices daily to max *Expected Total Sales Revenue* over T days
- Price for a given day picked from prices $P_1, P_2, \ldots, P_N \in \mathbb{R}$
- Customer daily demand is $Poisson(\lambda_i)$ if Price P_i is picked for the day

Dynamic Pricing for End-of-Life/End-of-Season

- Dynamic Pricing: Core to many businesses, flexing to supply/demand
- We consider special case of products being sold at end of life/season
- Assume we are T days from season-end and our inventory is M units
- Assume no more incoming inventory during these final T days
- Set prices daily to max *Expected Total Sales Revenue* over T days
- Price for a given day picked from prices $P_1, P_2, \ldots, P_N \in \mathbb{R}$
- Customer daily demand is $Poisson(\lambda_i)$ if Price P_i is picked for the day
- Note that demand can exceed inventory on any day, Sales $\leq$ Inventory

Dynamic Pricing Model for End-of-Life/End-of-Season

Dynamic Pricing Model for End-of-Life/End-of-Season

$$\mathcal{S}_t = \{(t, I_t) | I_t \in \mathbb{Z}, 0 \leq I_t \leq M\} \text{ for all } 0 \leq t \leq T$$

Dynamic Pricing Model for End-of-Life/End-of-Season

$$\mathcal{S}_t = \{(t, I_t) | I_t \in \mathbb{Z}, 0 \leq I_t \leq M\} \text{ for all } 0 \leq t \leq T$$

$$\mathcal{N}_t = \mathcal{S}_t \text{ and } \mathcal{A}_t = \{1, 2, \ldots, N\} \text{ for all } 0 \leq t < T, \text{ and } \mathcal{N}_T = \emptyset$$

Dynamic Pricing Model for End-of-Life/End-of-Season

$$\mathcal{S}_t = \{(t, I_t) | I_t \in \mathbb{Z}, 0 \leq I_t \leq M\} \text{ for all } 0 \leq t \leq T$$

$$\mathcal{N}_t = \mathcal{S}_t \text{ and } \mathcal{A}_t = \{1, 2, \ldots, N\} \text{ for all } 0 \leq t < T, \text{ and } \mathcal{N}_T = \emptyset$$

$$I_0 = M \text{ and } I_{t+1} = \max(0, I_t - d_t) \text{ where } d_t \sim Poisson(\lambda_i) \text{ if } a_t = i$$

Dynamic Pricing Model for End-of-Life/End-of-Season

$$\mathcal{S}_t = \{(t, I_t) | I_t \in \mathbb{Z}, 0 \leq I_t \leq M\} \text{ for all } 0 \leq t \leq T$$

$$\mathcal{N}_t = \mathcal{S}_t \text{ and } \mathcal{A}_t = \{1, 2, \ldots, N\} \text{ for all } 0 \leq t < T, \text{ and } \mathcal{N}_T = \emptyset$$

$$I_0 = M \text{ and } I_{t+1} = \max(0, I_t - d_t) \text{ where } d_t \sim Poisson(\lambda_i) \text{ if } a_t = i$$

$$\text{Sales Revenue on day } t \text{ is equal to } \min(I_t, d_t) \cdot P_i$$

Dynamic Pricing Model for End-of-Life/End-of-Season

$$\mathcal{S}_t = \{(t, I_t) | I_t \in \mathbb{Z}, 0 \leq I_t \leq M\} \text{ for all } 0 \leq t \leq T$$

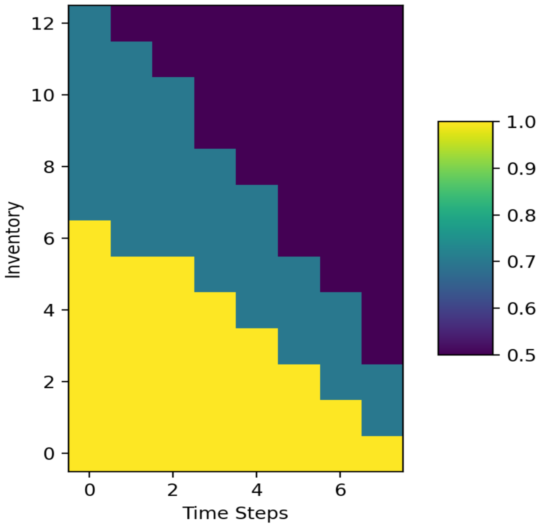
$$\mathcal{N}_t = \mathcal{S}_t \text{ and } \mathcal{A}_t = \{1, 2, \ldots, N\} \text{ for all } 0 \leq t < T, \text{ and } \mathcal{N}_T = \emptyset$$

$$I_0 = M \text{ and } I_{t+1} = \max(0, I_t - d_t) \text{ where } d_t \sim Poisson(\lambda_i) \text{ if } a_t = i$$

$$\text{Sales Revenue on day } t \text{ is equal to } \min(I_t, d_t) \cdot P_i$$

$$(\mathcal{P}_R)_t(I_t, i, r_{t+1}, I_t - k) = \begin{cases} \frac{e^{-\lambda_i} \lambda_i^k}{k!} & \text{if } k < I_t \text{ and } r_{t+1} = k \cdot P_i \\ \sum_{j=I_t}^{\infty} \frac{e^{-\lambda_i} \lambda_i^j}{j!} & \text{if } k = I_t \text{ and } r_{t+1} = k \cdot P_i \\ 0 & \text{otherwise} \end{cases}$$

Optimal Dynamic Pricing



Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as "tabular" algorithms

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as "tabular" algorithms
- "Tabular" means MDP is specified as a finite data structure

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as "tabular" algorithms
- "Tabular" means MDP is specified as a finite data structure
- More importantly, Value Function represented as a "table"

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as "tabular" algorithms
- "Tabular" means MDP is specified as a finite data structure
- More importantly, Value Function represented as a "table"
- These algorithms typically sweep through all states in each iteration

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as "tabular" algorithms
- "Tabular" means MDP is specified as a finite data structure
- More importantly, Value Function represented as a "table"
- These algorithms typically sweep through all states in each iteration
- Cannot do this for large finite spaces or for infinite spaces

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as "tabular" algorithms
- "Tabular" means MDP is specified as a finite data structure
- More importantly, Value Function represented as a "table"
- These algorithms typically sweep through all states in each iteration
- Cannot do this for large finite spaces or for infinite spaces
- Requires us to generalize to function approximation of Value Function

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as “tabular” algorithms
- “Tabular” means MDP is specified as a finite data structure
- More importantly, Value Function represented as a “table”
- These algorithms typically sweep through all states in each iteration
- Cannot do this for large finite spaces or for infinite spaces
- Requires us to generalize to function approximation of Value Function
    - Sample an appropriate subset of states

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as "tabular" algorithms
- "Tabular" means MDP is specified as a finite data structure
- More importantly, Value Function represented as a "table"
- These algorithms typically sweep through all states in each iteration
- Cannot do this for large finite spaces or for infinite spaces
- Requires us to generalize to function approximation of Value Function
  - Sample an appropriate subset of states
  - Calculate the Value Function for those states (Bellman calculation)

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as "tabular" algorithms
- "Tabular" means MDP is specified as a finite data structure
- More importantly, Value Function represented as a "table"
- These algorithms typically sweep through all states in each iteration
- Cannot do this for large finite spaces or for infinite spaces
- Requires us to generalize to function approximation of Value Function
  - Sample an appropriate subset of states
  - Calculate the Value Function for those states (Bellman calculation)
  - Create/Update a func approx with the sampled states' calculated values

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as “tabular” algorithms
- “Tabular” means MDP is specified as a finite data structure
- More importantly, Value Function represented as a “table”
- These algorithms typically sweep through all states in each iteration
- Cannot do this for large finite spaces or for infinite spaces
- Requires us to generalize to function approximation of Value Function
    - Sample an appropriate subset of states
    - Calculate the Value Function for those states (Bellman calculation)
    - Create/Update a func approx with the sampled states’ calculated values
- The fundamental structure of the algorithms is still the same

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as "tabular" algorithms
- "Tabular" means MDP is specified as a finite data structure
- More importantly, Value Function represented as a "table"
- These algorithms typically sweep through all states in each iteration
- Cannot do this for large finite spaces or for infinite spaces
- Requires us to generalize to function approximation of Value Function
    - Sample an appropriate subset of states
    - Calculate the Value Function for those states (Bellman calculation)
    - Create/Update a func approx with the sampled states' calculated values
- The fundamental structure of the algorithms is still the same
- Fundamental principles (Fixed-Point/Bellman Operators) still same

Generalizations to Non-Tabular Algorithms

- Finite MDP algorithms we covered known as "tabular" algorithms
- "Tabular" means MDP is specified as a finite data structure
- More importantly, Value Function represented as a "table"
- These algorithms typically sweep through all states in each iteration
- Cannot do this for large finite spaces or for infinite spaces
- Requires us to generalize to function approximation of Value Function
  - Sample an appropriate subset of states
  - Calculate the Value Function for those states (Bellman calculation)
  - Create/Update a func approx with the sampled states' calculated values
- The fundamental structure of the algorithms is still the same
- Fundamental principles (Fixed-Point/Bellman Operators) still same
- These generalizations known as *Approximate Dynamic Programming*

Key Takeaways from this Chapter

Key Takeaways from this Chapter

- Fixed-Point of Functions and Fixed-Point Theorem: Enables iterative algorithms to solve a variety of problems cast as Fixed-Point.

Key Takeaways from this Chapter

- Fixed-Point of Functions and Fixed-Point Theorem: Enables iterative algorithms to solve a variety of problems cast as Fixed-Point.
- Generalized Policy Iteration: Powerful idea of alternating between *any* method for Policy Evaluation and *any* method for Policy Improvement, including methods that are partial applications of Policy Evaluation or Policy Improvement. This generalized perspective unifies almost all of the algorithms that solve MDP Control problems.

Key Takeaways from this Chapter

- Fixed-Point of Functions and Fixed-Point Theorem: Enables iterative algorithms to solve a variety of problems cast as Fixed-Point.
- Generalized Policy Iteration: Powerful idea of alternating between *any* method for Policy Evaluation and *any* method for Policy Improvement, including methods that are partial applications of Policy Evaluation or Policy Improvement. This generalized perspective unifies almost all of the algorithms that solve MDP Control problems.
- Backward Induction: A straightforward method to solve finite-horizon MDPs by simply walking backwards and *setting* the Value Function from the horizon-end to the start.