- **27 Jan 2021 Notes:** Dynamic Programming Continued.

- Bellman Optimality Operator:

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

Action prescribed by Greedy policy is the one which satisfies Bellman optimality Equation

B^π motivated by MDP Bellman Policy Equation

Bellman Optimality Operator Equation can be expressed as

$$V^* = B^*(V^*)$$

$\therefore$ V^* is a fixed point

B^* is a contraction function

↳ Read Proof from slide 19 and textbook, will likely be similar to exam.

- Steps of proof:

① Monotonicity Property

$$X \geq Y \Rightarrow B^*(X) \geq B^*(Y)$$

② Constant Shift Property

$$B^*(X + c) = B^*(X) + \gamma c$$

$$\Rightarrow \max_{s \in N} \left| \left(B^*(X) - B^*(Y)\right)(s) \right| \leq \dots$$ see slide 19.

- **Value iteration algorithm**
  - Complexity is $O(n^2 k)$
  - Repeatedly apply B^* on X until convergence

- Optimal Policy from Optimal Value Function

$$V^{\pi^*} = V^*$$

↑ Searching for policy which achieves optimal value

↪ Use greedy policy function G

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

$$\Updownarrow$$

$$V^*$$

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

↪ V^* is a fixed point

↪ We know $B^{G(V^*)}$ has a unique fixed point at $V^{G(V^*)}$

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

↑ Evaluating MDP with greedy policy at V^* achieves V^*

$$\therefore \pi^* = G(V^*)$$

↑ We know how to get this

- Slide 31 has a good demonstration of algorithm.
  - ↳ Rows are policy evaluations
    - ↳ Repeatedly apply Bellman Operator
  - ↳ Between each row is policy improvement step.

- Two notions of consistency:
  1. Across rows, make VF V consistent with / close to V^{π} of the policy π
  2. Between rows, make π consistent with / close to Greedy policy $G(V)$ of VF V
     - ↳ Achieving one type of consistency loses the other type of consistency. Therefore, alternate between them

- Generalized Policy Iteration is the foundation of RL
  - ↳ Intuition: Go partially between the two notions of consistency

- Value iteration schematic on Slide 25
  - ↳ "Lazy" version of policy iterations (typo on slide)
- Large scale RL uses Partial policy evaluation and policy improvement

* Policy iteration is more precise, but slower than value iteration
- ↳ Both work for a small state space
- ↳ Value iterations can have convergence issues
- ↳ But for finite state spaces, both work and value iteration is better.

- To implement DP algorithms, see slide 26

  - ↳ Asynchronous DP algorithm uses prioritized sweeping, ranked based on gaps between states in the algorithm.

    - ↳ Address large gaps first.

    - ↳ You need links from destination states to start states to sort queue (disadvantage).

- Bathroom Break... Continue from slide 27 (~40 minute point of lecture)

- <u>Finite Horizon MDP</u>:

  - ↳ Each sequence terminates in a finite number of time steps T

  - ↳ Each time step has a separate (from other time steps) set of states

  - ↳ Walking back in time

  - ↳ MDP with fixed policy is equivalent to MRP

  - ↳ Backward Induction Algorithm: $O(m^2T)$

    Decrement t from T to 0 and calculate V_t^π from V_{t+1}^π

    - ↳ Backward induction applies to both prediction and control

      - ↳ For control, extract optimal policy using argmax

- Lots of problems in finance show up as finite horizon problem

  <u>E.g.</u> Dynamic Pricing: End-of-life / End-of-Season clearance

  - ↳ How much to reduce price at end of season/life

    T days away from end of season

M units of inventory

Maximize expected total sales revenue over T days

↳ Choose discrete prices P_t

↳ Each price associated with Poisson Distribution of demand

↳ Price elasticity of demand

Visualize as grid of time and inventory

↳ See slide 32 for model

↳ How do you model λ in Poisson distribution of demand based on pricing.

- Generalizations to non-tabular Algorithm

↳ Value function represented as a table in tabular algorithms

↳ You cannot do this for infinite state

↳ Instead, collect a sample of states and apply Bellman operator for those states. Create a function approximation with those samples (using interpolation).

↳ Fundamental principles are the same.

↳ Known as Approximate dynamic programming

- <u>Approximate Dynamic Programming</u> (New slide deck)

- Function Approximation:

↳ Predictor variable $x \in \mathcal{X}$, response $y \in \mathbb{R}$

$$P[y|x] = f(x; w)(y)$$

(w: Parameters)

↳ Assume data in form $[(x_i, y_i) \mid 1 \le i \le n]$ is empirical distribution D

↳ We want to construct model probability distribution M

↳ Reconcile D and M to minimize Cross-entropy loss:

$$H(D, M) = -E_D[\log M]$$

↳ Minimize using gradient descent

- First method in FunctionApprox is to solve for w given pairs of x, y

- Linear Function Approximation : Slides 7-9

- Deep Neural Network : Slide 10