A.I. for Optimal Decisioning under Uncertainty

Applied to Inventory and Price Optimization

Ashwin Rao

V.P. Artificial Intelligence at Target & Adjunct Faculty at Stanford

May 1, 2020

Meet the Speaker

- Here at Target, V.P. of Artificial Intelligence team in Data Sciences
- Other Job: Adjunct Faculty in Applied Math at Stanford University
- Director of Stanford's Mathematical & Comp. Finance program
- At Stanford, I teach a course on Reinforcement Learning for Finance
- I've written an educational codebase for Reinforcement Learning
- Previously, 14 years in Derivatives Trading at GS and MS in NY
- My teaching has been in Pure & Applied Math, Comp Sci, Finance
- My original background is Algorithms Theory & Abstract Algebra

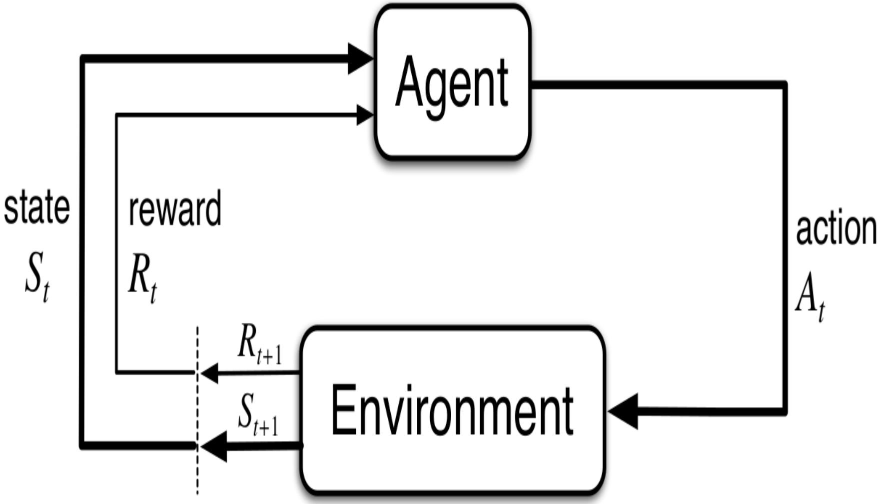
Overview

1. The Framework of Stochastic Control
2. Inventory Optimization
3. Clearance Price Optimization

A.I. for Optimal/Dynamic Decisioning under Uncertainty

- Let's look at some terms we use to characterize this branch of A.I.
- *Stochastic*: Uncertainty in key quantities, evolving over time
- *Optimization*: A well-defined metric to be maximized ("The Goal")
- *Dynamic*: Decisions need to a function of the changing situations
- *Control*: Overpower uncertainty by persistent steering towards goal
- Jargon overload due to confluence of Control Theory, O.R. and A.I.
- For language clarity, let's just refer to this area as *Stochastic Control*
- The core framework is called *Markov Decision Processes* (MDP)
- *Reinforcement Learning* is a class of algorithms to solve MDPs

The MDP Framework



Components of the MDP Framework

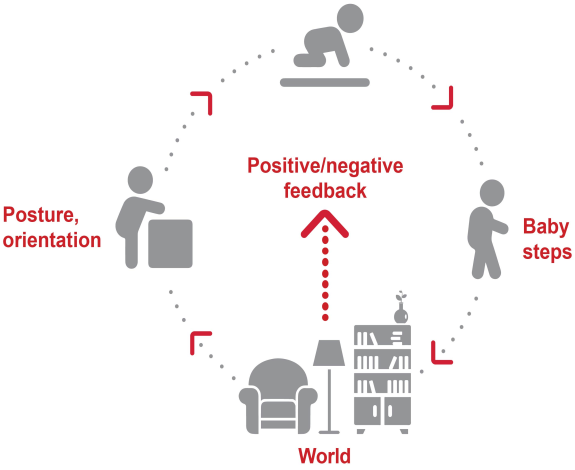
- The *Agent* and the *Environment* interact in a time-sequenced loop
- *Agent* responds to [*State*, *Reward*] by taking an *Action*
- *Environment* responds by producing next step's (random) *State*
- *Environment* also produces a (random) scalar denoted as *Reward*
- Each *State* is assumed to have the *Markov Property*, meaning:
  - Next *State/Reward* depends only on Current *State* (for a given *Action*)
  - Current *State* captures all relevant information from *History*
  - Current *State* is a sufficient statistic of the future (for a given *Action*)
- Goal of *Agent* is to maximize *Expected Sum* of all future *Reward*s
- By controlling the ($Policy : State \rightarrow Action$) function
- This is a dynamic (time-sequenced control) system under uncertainty

Formal MDP Framework

The following notation is for discrete time steps. Continuous-time formulation is analogous (often involving Stochastic Calculus)

- Time steps denoted as $t = 1, 2, 3, \ldots$
- Markov States $S_t \in \mathcal{S}$ where $\mathcal{S}$ is the State Space
- Actions $A_t \in \mathcal{A}$ where $\mathcal{A}$ is the Action Space
- Rewards $R_t \in \mathbb{R}$ denoting numerical feedback
- Transitions $p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$
- $\gamma \in [0, 1]$ is the Discount Factor for Reward when defining *Return*
- Return $G_t = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \ldots$
- Policy $\pi(a|s)$ is probability that Agent takes action a in states s
- The goal is find a policy that maximizes $\mathbb{E}[G_t|S_t = s]$ for all $s \in \mathcal{S}$

How a baby learns to walk



Many real-world problems fit this MDP framework

- Self-driving vehicle (speed/steering to optimize safety/time)
- Game of Chess (Boolean *Reward* at end of game)
- Complex Logistical Operations (eg: movements in a Warehouse)
- Make a humanoid robot walk/run on difficult terrains
- Manage an investment portfolio
- Control a power station
- Optimal decisions during a football game
- Strategy to win an election (high-complexity MDP)

Self-Driving Vehicle

Why are these problems hard?

- *State* space can be large or complex (involving many variables)
- Sometimes, *Action* space is also large or complex
- No direct feedback on "correct" *Actions* (only feedback is *Reward*)
- Time-sequenced complexity (*Actions* influence future *States/Actions*)
- *Action*s can have delayed consequences (late *Reward*s)
- *Agent* often doesn't know the *Model* of the *Environment*
- "Model" refers to probabilities of state-transitions and rewards
- So, *Agent* has to learn the *Model* AND solve for the Optimal *Policy*
- *Agent Action*s need to tradeoff between "explore" and "exploit"

Value Function and Bellman Equations

- Value function (under policy π) $V_\pi(s) = \mathbb{E}[G_t | S_t = s]$ for all $s \in \mathcal{S}$

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \cdot (r + \gamma V_\pi(s')) \text{ for all } s \in \mathcal{S}$$

- Optimal Value Function $V_*(s) = \max_\pi V_\pi(s)$ for all $s \in \mathcal{S}$

$$V_*(s) = \max_a \sum_{s',r} p(s', r|s, a) \cdot (r + \gamma V_*(s')) \text{ for all } s \in \mathcal{S}$$

- *There exists an Optimal Policy* π_* achieving $V_*(s)$ for all $s \in \mathcal{S}$
- Determining $V_\pi(s)$ known as *Prediction*, and $V_*(s)$ known as *Control*
- The above recursive equations are called *Bellman equations*
- In continuous time, refered to as *Hamilton-Jacobi-Bellman (HJB)*
- The algorithms based on Bellman equations are broadly classified as:
    - Dynamic Programming
    - Reinforcement Learning

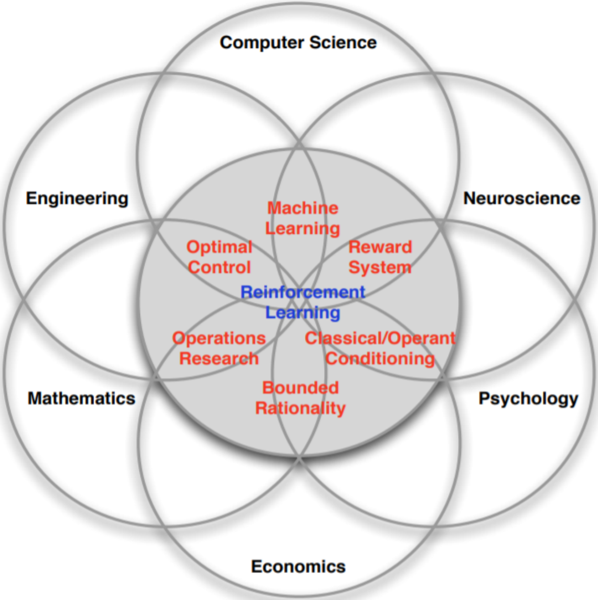
Dynamic Programming versus Reinforcement Learning

- When Probabilities Model is known $\Rightarrow$ *Dynamic Programming* (DP)
- DP Algorithms take advantage of knowledge of probabilities
- So, DP Algorithms do not require interaction with the environment
- In the Language of A.I, DP is a type of *Planning Algorithm*
- When Probabilities Model unknown $\Rightarrow$ *Reinforcement Learning* (RL)
- RL Algorithms interact with the Environment and incrementally learn
- Environment interaction could be *real* or *simulated* interaction
- RL approach: Try different actions & learn what works, what doesn't
- RL Algorithms' key challenge is to tradeoff "explore" versus "exploit"
- DP or RL, Good approximation of Value Function is vital to success
- Deep Neural Networks are typically used for function approximation

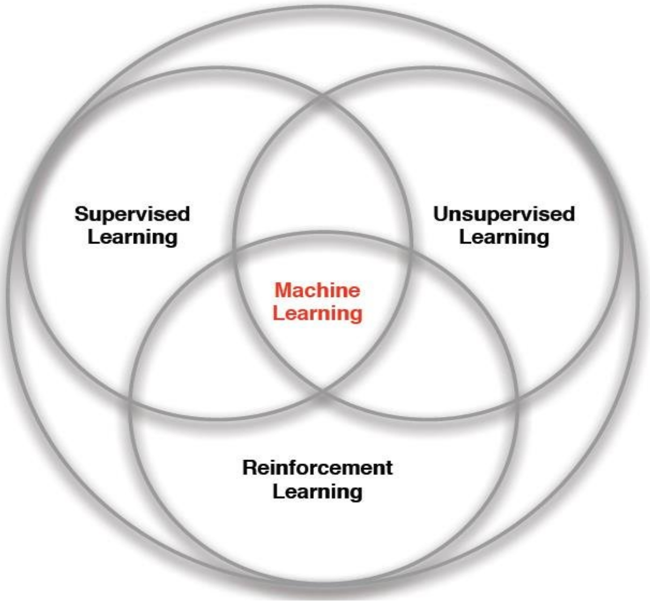
Why is RL interesting/useful to learn about?

- RL solves MDP problem when *Environment Probabilities* are unknown
- This is typical in real-world problems (complex/unknown probabilities)
- RL interacts with *Actual Environment* or with *Simulated Environment*
- **Promise of modern A.I. is based on success of RL algorithms**
- Potential for automated decision-making in many industries
- In 10-20 years: Bots that act or behave more optimal than humans
- RL already solves various low-complexity real-world problems
- RL might soon be the most-desired skill in the technical job-market
- Learning RL is a lot of fun! (interesting in theory as well as coding)

Many Faces of Reinforcement Learning



Vague (but in-vogue) Classification of Machine Learning



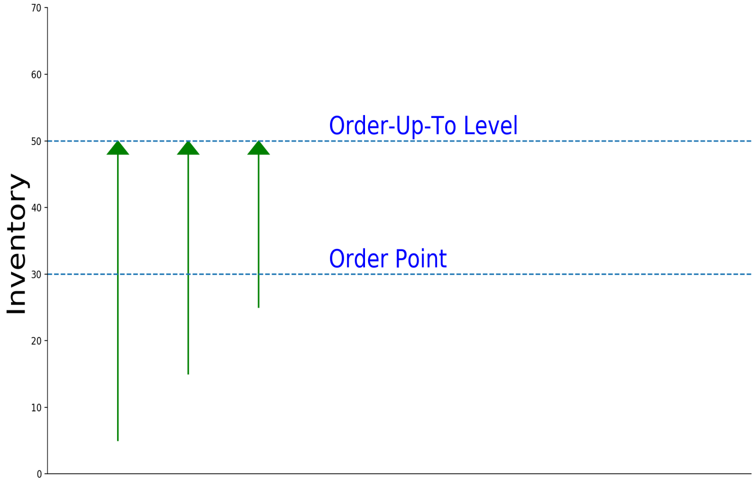
Inventory Optimization

- A fundamental problem in Retail is Inventory Optimization
- How to move inventory optimally from vendors to guests
- Guest Demand is fairly uncertain
- Nirvana is when Inventory appears "just in time" to satisfy Demand
- This is an example of a Stochastic Control problem

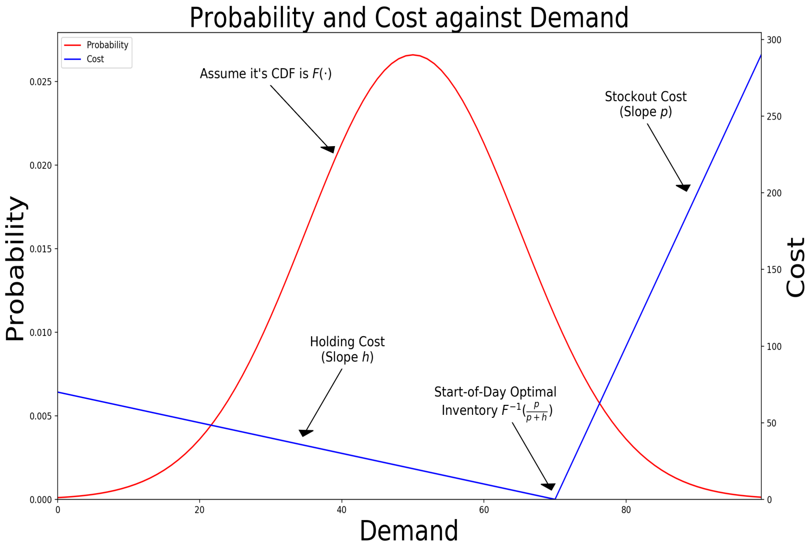
Single-store, Single-item Inventory Optimization

- The store experiences random daily demand given by PDF $f(x)$
- The store can order daily from a vendor carrying infinite inventory
- There's a cost associated with ordering, and order arrives in L days
- Holding Cost h for each unit of overnight inventory
- Stockout Cost p for each unit of lost sales due to empty shelf
- This is an MDP where *State* is current Inventory Position
- *Action* is quantity to Order
- *Reward* function has h, p, and ordering cost
- Transition probabilities are governed by demand distribution $f(x)$
- The Optimal (Ordering) Policy has a simple closed-form solution

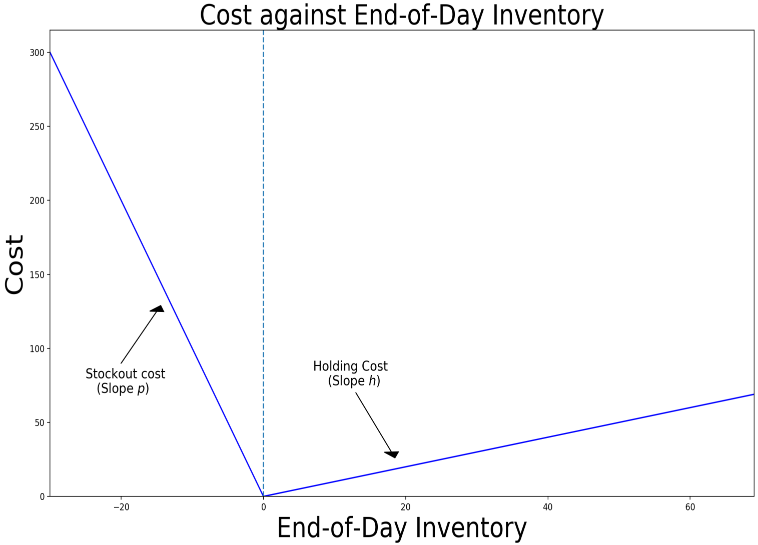
Optimal Ordering Policy



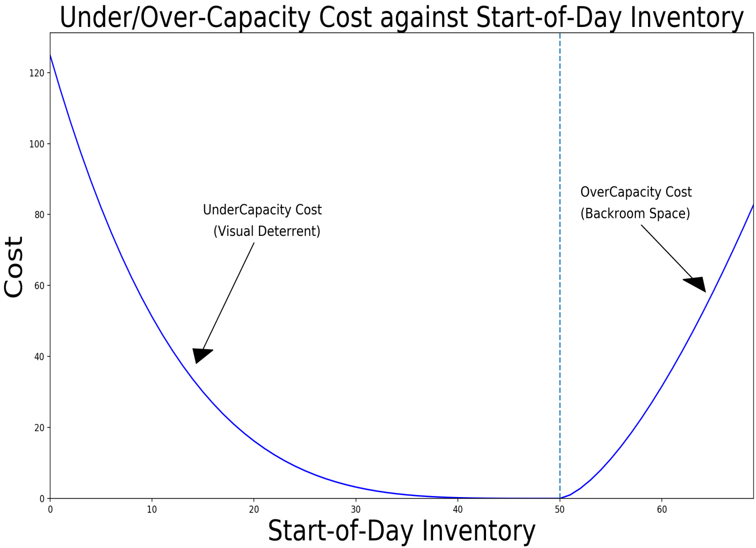
The Core of this Solution has this Pictorial Intuition



Costs viewed against End-of-Day Inventory



UnderCapacity and OverCapacity Costs



UnderCapacity Cost: Guest Psychology and Economics

- Retail Mantra: "Stack it high and watch it fly"
- Guests like to see shelves well stocked
- Visual emptiness is known to be a sales deterrent
- So, full-looking shelves are part of presentation strategy
- At a certain level of emptiness, the deterrent rises sharply
- Hence the convex nature of this cost curve
- Note that this curve varies from item to item
- It also varies from regular season to end of season
- Modeling/calibrating this is tricky!
- However, getting a basic model in place is vital

OverCapacity Cost: Backroom Space Constraints

- Retail store backrooms have limited capacity
- Typically tens of thousands of items compete for this space
- Retailers like to have clean and organized backrooms
- A perfect model is when all your inventory is on store shelves
- With backroom used purely as a hub for home deliveries
- Practically, some overflow from shelves is unavoidable
- Hence, the convex nature of this curve
- Modeling this is hard because it's a multi-item cost/constraint
- Again, getting a basic model in place is vital

What other costs are involved?

- Holding Cost: Interest on Inventory, Superficial Damage, Maintenance
- Stockout Cost: Lost Sales, sometimes Lost Customers
- Labor Cost: Replenishment involves movement from truck to shelf
- Spoilage Cost: Food & Beverages can have acute perishability
- End-of-Season/Obsolescence Cost: Intersects with Clearance Pricing

Practical Inventory Optimization as an MDP

- The store experiences random daily demand
- The store can place a replenishment order in casepack mutiples
- This is an MDP where *State* is current Inventory Position
- *Action* is the multiple of casepack to order (or not order)
- *Reward* function involves all of the costs we went over earlier
- State transitions governed by demand probability distribution
- Solve: Dynamic Programming or Reinforcement Learning Algorithms

Multi-node and Multi-item Inventory Optimization

- In practice, Inventory flows through a network of DCs/stores
- From source (vendors) to destination (stores or homes)
- So, we have to solve a multi-"node" Inventory Optimization problem
- *State* is joint inventory across all nodes (and between nodes)
- *Action* is recommended movements of inventory between nodes
- *Reward* is the aggregate of daily costs across the network
- Space and Throughput constraints are multi-item costs/constraints
- So, real-world problem is multi-node and multi-item (giant MDP)

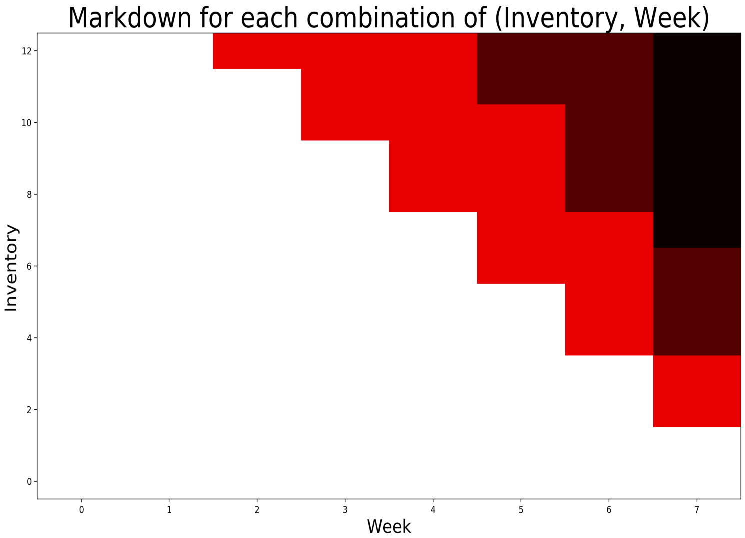
Clearance Price Optimization

- You are a few weeks away from end-of-season (eg: Christmas Trees)
- Assume you have too much inventory in your store
- What is the optimal sequence of price markdowns?
- Under (uncertain) demand responding to markdowns
- So as to maximize your total profit (sales revenue minus costs)
- Note: There is a non-trivial cost of performing a markdown
- If price markdowns are small, we end up with surplus at season-end
- Surplus often needs to be disposed at poor salvage price
- If price reductions are large, we run out of Christmas trees early
- "Stockout" cost is considered to be large during holiday season

MDP for Clearance Price Optimization

- *State* is [Days Left, Current Inventory, Current Price, Market Info]
- *Action* is Price Markdown
- *Reward* includes Sales revenue, markdown cost, stockout cost, salvage
- *Reward* & *State*-transitions governed by *Price Elasticity of Demand*
- Real-world *Model* can be quite complex (eg: competitor pricing)
- Big Idea: Blend Inventory and Price Optimization into one MDP

Optimal Markdown Frontier



Components of Clearance Pricing A.I.

- Statistical Estimation of *Price Elasticity of Demand*
- Backward Induction algorithm for Optimal Dynamic Pricing
- Simulation of the Optimal Policy to reveal various metrics to analyze

Inventory Rampdown as a function of Elasticity