A.I. for Dynamic Decisioning under Uncertainty

For Real-World Problems in Retail & Financial Trading

Ashwin Rao

VP Data Science at Target & Adjunct Faculty at Stanford

November 28, 2018

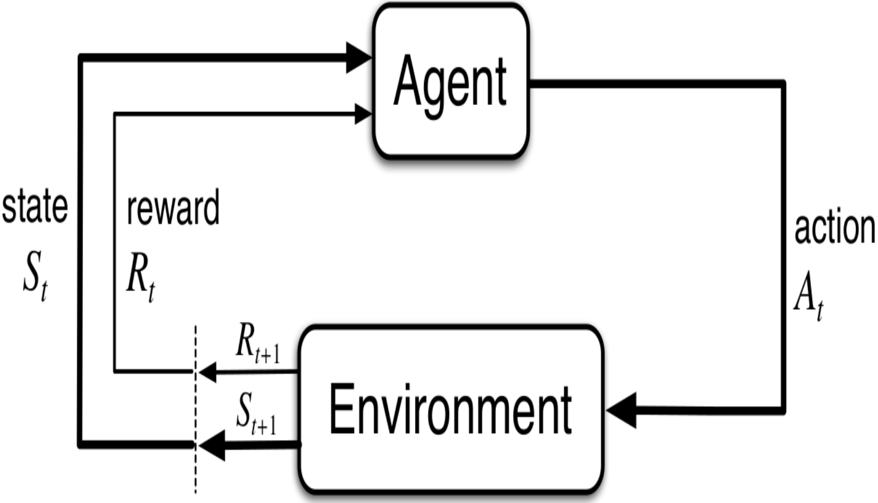
A.I. for Dynamic Decisioning under Uncertainty

- 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*
- I was introduced to this area through Inventory Control
- At Stanford, I teach a course on A.I. for Stochastic Control in Finance
- I'm developing an educational codebase for Stochastic Control
- Overarching Goal: Blend Theory, Algorithms & Real-World Modeling

Overview

1. The Framework of Stochastic Control
2. Core Problem in Retail: Inventory Control
3. Core Problem in Finance: Portfolio Optimization/Asset Allocation
4. Quick look at a few other problems in Retail and Finance
5. Perspective from the Trenches

The Stochastic Control Framework



Components of the 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) number denoted as *Reward*
- Goal of *Agent* is to maximize *Expected Sum* of all future *Rewards*
- By controlling the ($Policy : State \rightarrow Action$) function
- This is a dynamic (time-sequenced control) system under uncertainty
- Formally known as a Markov Decision Process (MDP)

Formal MDP Framework

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

- 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 $G_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+1} + \ldots$
- 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 $s \in \mathcal{S}$

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)

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*)
- *Action*s can have delayed consequences (late *Reward*s)
- Time-sequenced complexity (eg: *Actions* influence future *Actions*)
- *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*

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 $s \in \mathcal{S}$
- 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

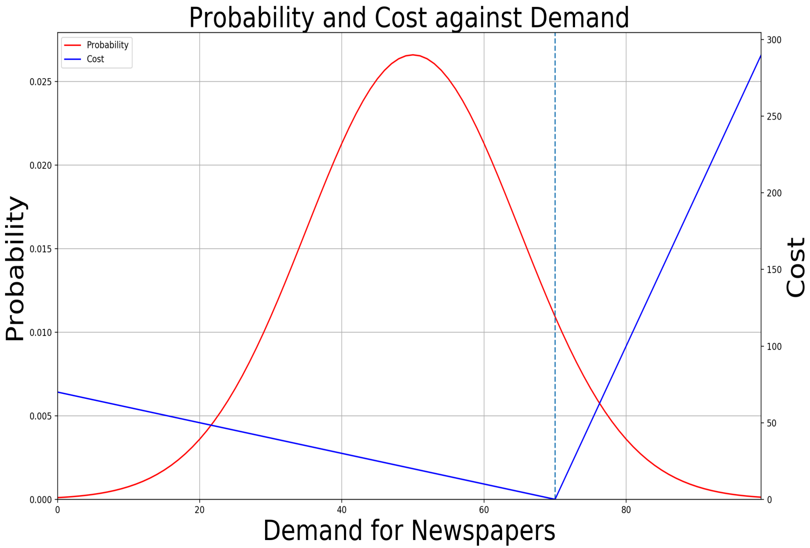
Dynamic Programming versus Reinforcement Learning

- When 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
- When Model is unknown $\Rightarrow$ *Reinforcement Learning* (RL)
- RL Algorithms interact with the Environment and incrementally learn
- Environment interaction could be real interaction or a simulator
- 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

Inventory Control starts with the Newsvendor Problem

- Newsvendor problem is a single-period Inventory Control problem
- Daily demand for newspapers is a random variable x
- The newsvendor has an estimate of the PDF $f(x)$ of daily demand
- For each newspaper that stays unsold, we suffer an *Excess cost* h
- Think of h as the purchase price minus salvage price
- For each newspaper we're short on, we suffer a *Deficit Cost* p
- Think of p as the missed profits (sale price minus purchase price)
- But p should also include potential loss of future customers
- What is the optimum # of newspapers to bring in the morning?
- To minimize the expected cost (function of f, h and p)

The Newsvendor Problem



Solution to the Newsvendor problem

- For tractability, we assume newspapers are a continuous variable x
- Then, we need to solve for the optimal supply S that maximizes

$$g(S) = h \int_0^S (S - x) \cdot f(x) \cdot dx + p \int_S^\infty (x - S) \cdot f(x) \cdot dx$$

- Setting $g'(S) = 0$, we get:

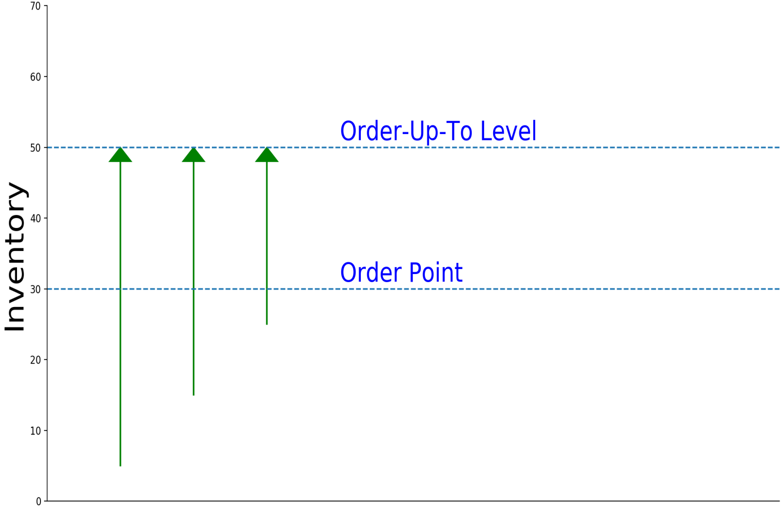
$$\text{Optimal Supply } S^* = F^{-1}(\frac{p}{p + h})$$

  where $F(y) = \int_0^y f(x) dx$ is the CDF of daily demand
- $\frac{p}{p+h}$ is known as the critical fractile
- It is the fraction of days when the newsvendor goes "out-of-stock"
- Assuming the newsvendor always brings this optimal supply S^*
- Solution details and connections with Financial Options Pricing here

Multi-period: Single-store, Single-item Inventory Control

- The store experiences random daily demand given by PDF $f(x)$
- The store can order daily from a supplier carrying infinite inventory
- There's a cost associated with ordering, and order arrives in L days
- Like newsvendor, there's an Excess Cost h and Deficit Cost p
- This is an MDP where *State* is current Inventory Level at the store
- *State* also includes current in-transit inventory (from supplier)
- *Action* is quantity to order in any given *State*
- *Reward* function has h, p (just like newsvendor), and ordering cost
- Transition probabilities are governed by demand distribution $f(x)$
- This has a closed-form solution, similar to newsvendor fomula

Optimal Policy: Order Point and Order-Up-To Level



Adding real-world frictions and constraints

- Inventory is integer-valued, and orders are in casepack units
- Excess/Deficit costs are not linear functions of inventory
- Perishability, Obsolescence, End-of-season involve big costs
- Need to factor in labor costs of handling cases and singles
- Limits on shipping/receiving dates/times
- Often, there is a constraint on minimum presentation quantities
- Store inventory cannot exceed a threshold (eg: Shelf space)
- Supplier has constraints on min and max order quantities
- Uncertainty with the time for order arrival
- There are approximate closed-form solutions in some cases
- But general case requires generic DP or RL Algorithms

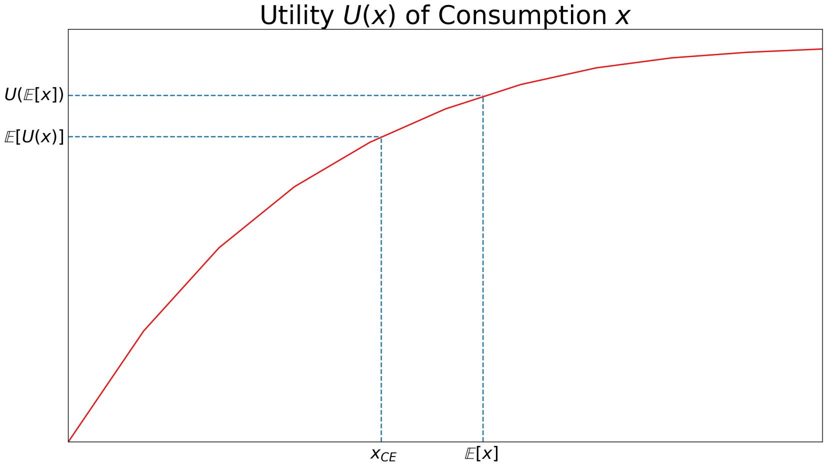
Multi-node and Multi-item Inventory Control

- In practice, Inventory flows through a network of warehouses
- From source (suppliers) to destination (stores or homes)
- So, we have to solve a multi-"node" Inventory Control 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
- In addition, we have multi-item constraints
- Space and Throughput constraints are multi-item constraints
- So, real-world problem is multi-node and multi-item (giant MDP)

Single-period Portfolio Optimization

- We start with a simple single-period portfolio optimization problem
- The simple setup helps develop understanding of the core concepts
- Assume we have one risky asset and one riskless asset
- The return (over the single-period) of the risky asset is $\mathcal{N}(\mu, \sigma^2)$
- The return of the riskless asset is deterministic ($= r < \mu$)
- Start with \$1, aim to maximize our period-ending Expected Wealth
- This means we invest fully in the risky asset (since $\mu > r$)
- But people are risk-averse and will trade higher returns for lower risk
- The exact Risk-Return tradeoff is specified through *Utility of Wealth*
- *Utility* is a concave function of Wealth describing *Risk-Aversion*
- For an intro to Risk-Aversion and Utility Theory, see here
- The goal is to maximize period-ending **Expected Utility of Wealth**

Concave Utility of Consumption $\Rightarrow \mathbb{E}[U(x)] < U(\mathbb{E}[x])$



Solution to Single-period Portfolio Optimization

- W denotes end-of-period Wealth
- α denotes fraction to invest in risky asset ($1 - \alpha$ is fraction in riskless)

$$W \sim \mathcal{N}(1 + r + \alpha(\mu - r), \alpha^2\sigma^2)$$

- Let Utility of Wealth function be $U(W) = -e^{-\beta W}$ for $\beta > 0$
- Where β is the coefficient (extent) of Risk-Aversion
- So we maximize over α, the Expected Utility

$$\mathbb{E}[-e^{-\beta W}] = -e^{-\beta(1+r+\alpha(\mu-r))+\frac{\beta^2\alpha^2\sigma^2}{2}} = g(\alpha)$$

- Setting $\frac{\partial\{\log g(\alpha)\}}{\partial\alpha} = 0$, we get:

$$\alpha^* = \frac{\mu - r}{\beta\sigma^2}$$

- This is the fundamental investment fraction in Portfolio Optimization
- This fraction generalizes to multi-period and multiple risky assets

Multi-Period: Merton's Portfolio Optimization Problem

- You will live for (deterministic) T more years (say 20 years)
- Current Wealth is W_0 (say \$ 10 million)
- Assume you have no further income and no debts
- You can invest in (allocate to) n risky assets and a riskless asset
- Each asset has known normal distribution of returns
- Allowed to long or short any fractional quantities of assets
- Trading in continuous time $0 \leq t < T$, with no transaction costs
- You can consume any fractional amount of wealth at any time
- Dynamic Decision: Optimal Allocation and Consumption at each time
- To maximize lifetime-aggregated Utility of Consumption
- Consumption Utility assumed to have constant Relative Risk-Aversion

Problem Notation

For ease of exposition, we state the problem for 1 risky asset

- Riskless asset: $dR_t = r \cdot R_t \cdot dt$
- Risky asset: $dS_t = \mu \cdot S_t \cdot dt + \sigma \cdot S_t \cdot dz_t$ (i.e. Geometric Brownian)
- $\mu > r > 0, \sigma > 0$ (for n assets, we work with a covariance matrix)
- Wealth at time t is denoted by $W_t > 0$
- Fraction of wealth allocated to risky asset denoted by $\pi(t, W_t)$
- Fraction of wealth in riskless asset will then be $1 - \pi(t, W_t)$
- Wealth consumption denoted by $c(t, W_t) \geq 0$
- Utility of Consumption function $U(x) = \frac{x^{1-\beta}}{1-\beta}$ for $0 < \beta \neq 1$
- β = (constant) Relative Risk-Aversion $\frac{-x \cdot U''(x)}{U'(x)}$

Continuous-Time Stochastic Control Problem

- Balance constraint implies the following process for Wealth W_t

$$dW_t = ((\pi_t \cdot (\mu - r) + r) \cdot W_t - c_t) \cdot dt + \pi_t \cdot \sigma \cdot W_t \cdot dz_t$$

- At any time t, determine optimal $[\pi(t, W_t), c(t, W_t)]$ to maximize:

$$E[\int_t^T \frac{\gamma^{s-t} \cdot c_s^{1-\beta}}{1-\beta} \cdot ds \mid W_t]$$

- where $0 \leq \gamma \leq 1$ is the annualized discount factor
- Think of this as a continuous-time Stochastic Control problem
- *State* is (t, W_t), *Action* is $[\pi_t, c_t]$, *Reward* per unit time is $U(c_t)$
- Stochastic process for W_t above defines the transition probabilities
- Find *Policy* $: (t, W_t) \rightarrow [\pi_t, c_t]$ that maximizes the *Expected Return*
- Note: $c_t \geq 0$, but π_t is unconstrained

Outline of Solution

- Optimal Value Function $V^*(t, W_t)$ has a recursive formulation

$$V^*(t, W_t) = \max_{\pi, c} E[\int_t^{t_1} \frac{\gamma^s \cdot c_s^{1-\beta}}{1-\beta} \cdot ds + V^*(t_1, W_{t_1})]$$

- Re-expressed as a Hamilton-Jacobi-Bellman (HJB) formulation

$$\max_{\pi_t, c_t} E[dV^*(t, W_t) + \frac{\gamma^t \cdot c_t^{1-\beta}}{1-\beta} \cdot dt] = 0$$

- Standard HJB calculus (Ito's Lemma followed by partial derivatives w.r.t. π_t, c_t) gives us a PDE for $V^*(t, W_t)$
- We can reduce the PDE to a tractable ODE with a guessed solution
- All the gory details in this lecture
- This solution outline is broadly applicable to continuous-time problems

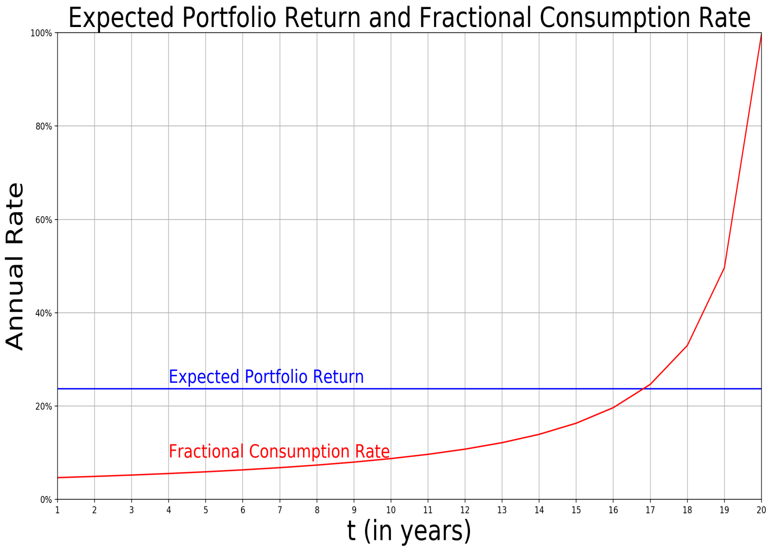
Gaining Insights into the Solution

- Optimal Allocation $\pi^*(t, W_t) = \frac{\mu - r}{\sigma^2 \beta}$ (independent of t and W_t)
- Optimal Fractional Consumption $\frac{c^*(t, W_t)}{W_t}$ depends only on t ($= f(t)$)
- With Optimal Allocation & Consumption, the Wealth process is:

$$\frac{dW_t}{W_t} = (r + \frac{(\mu - r)^2}{\sigma^2 \beta} - f(t)) \cdot dt + \frac{\mu - r}{\sigma \beta} \cdot dz_t$$

- Expected Portfolio Return is constant over time ($= r + \frac{(\mu - r)^2}{\sigma^2 \beta}$)
- Fractional Consumption $f(t)$ increases over time
- Expected Rate of Wealth Growth $r + \frac{(\mu - r)^2}{\sigma^2 \beta} - f(t)$ decreases over time
- If $r + \frac{(\mu - r)^2}{\sigma^2 \beta} > f(0)$, we start by Consuming < Expected Portfolio Growth and over time, we Consume > Expected Portfolio Growth
- Wealth Growth Volatility is constant ($= \frac{\mu - r}{\sigma \beta}$)

Portfolio Return versus Consumption Rate



Porting this to Real-World Portfolio Optimization

- Analytical tractability in Merton's formulation was due to:
  - Normal distribution of asset returns
  - Constant Relative Risk-Aversion
  - Frictionless, continuous trading
- However, real-world situation involves:
  - Discrete amounts of assets to hold and discrete quantities of trades
  - Transaction costs
  - Locked-out days for trading
  - Non-stationary/arbitrary/correlated processes of multiple assets
  - Changing/uncertain risk-free rate
  - Consumption constraints
  - Arbitrary Risk-Aversion/Utility specification
- $\Rightarrow$ Approximate Dynamic Programming or Reinforcement Learning
- Large Action Space points to Policy Gradient Algorithms

Overview of a few other problems

- Financial Trading: American Options Pricing
- Financial Trading: Trade Order Execution
- Retail: Clearance Pricing

American Options Pricing

- American option can be exercised anytime before option maturity
- Key decision at any time is to exercise or continue
- The default algorithm is Backward Induction on a tree/grid
- But it doesn't work for path-dependent options
- Also, it's not feasible when state dimension is large
- Industry-Standard: Longstaff-Schwartz's simulation-based algorithm
- RL is an attractive alternative to Longstaff-Schwartz
- RL is straightforward once Optimal Exercise is modeled as an MDP

MDP for Optimal Options Exercise

- *State* is [Current Time, History of Underlying Security Prices]
- *Action* is Boolean: Exercise (i.e., Payoff and Stop) or Continue
- *Reward* always 0, except upon Exercise (= Payoff)
- *State*-transitions governed by Underlying Prices' Stochastic Process
- Optimal Policy ⇒ Optimal Stopping ⇒ Option Price
- All the details in this lecture
- Can be generalized to other Optimal Stopping problems

Optimal Trade Order Execution (controlling Price Impact)

- You are tasked with selling a large qty of a (relatively less-liquid) stock
- You have a fixed horizon over which to complete the sale
- The goal is to maximize aggregate sales proceeds over the horizon
- If you sell too fast, *Price Impact* will result in poor sales proceeds
- If you sell too slow, you risk running out of time
- We need to model temporary and permanent *Price Impact*s
- Objective should incorporate penalty for variance of sales proceeds
- Which is equivalent to maximizing aggregate Utility of sales proceeds

MDP for Optimal Trade Order Execution

- *State* is [Time Remaining, Stock Remaining to be Sold, Market Info]
- *Action* is Quantity of Stock to Sell at current time
- *Reward* is Utility of Sales Proceeds (i.e., Variance-adjusted-Proceeds)
- *Reward* & *State*-transitions governed by *Price Impact Model*
- Real-world *Model* can be quite complex (Order Book Dynamics)

Clearance Pricing

- 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?
- 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 Pricing

- *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)

Perspective (and a bit of "advice") from the Trenches

- I always start with a simple version of problem to develop intuition
- My first line of attack is DP customized to the problem structure
- RL Algorithms that are my personal favorites (links to my lectures):
  - Deep Q-Network (DQN): Experience Replay, 2nd Target Network
  - Least Squares Policy Iteration (LSPI) - Batch Linear System
  - Exact Gradient Temporal-Difference (GTD)
  - Policy Gradient (esp. Natural Gradient, TRPO)
- I prefer to separate Model Estimation from Policy Optimization
- So we can customize RL algorithms to take advantage of:
  - Knowledge of transition probabilities
  - Knowledge of reward function
  - Any problem-specific structure that simplifies the algorithm
- Feature Engineering based on known closed-form approximations
- Many real-world, large-scale problems ultimately come down to suitable choices of DNN architectures and hyperparameter tuning ☹