Reinforcement Learning and it's Applications in Finance (A Thalesians Talk)

Ashwin Rao

Stanford University

July 14, 2021

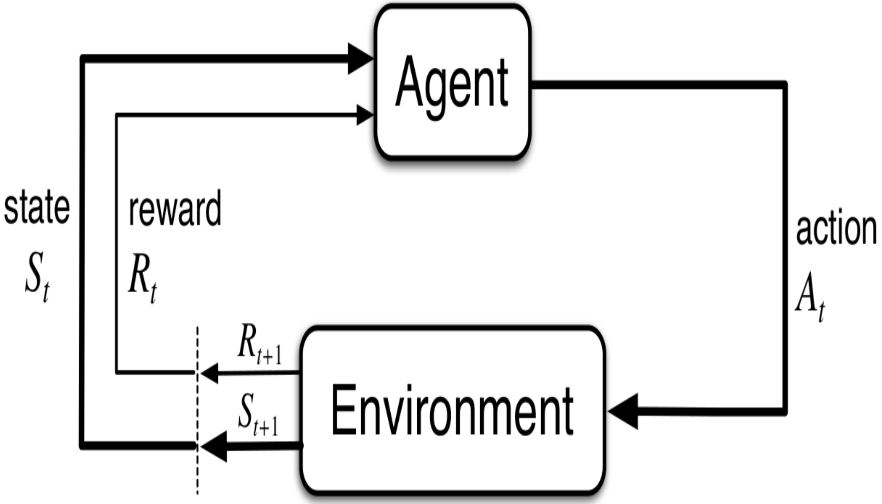
About Me

- VP of AI at Target Corporation ($\sim$ \$100B US Retail Company)
- Adjunct Professor, Applied Math (ICME), Stanford University
- Past: MD at Morgan Stanley, Trading Strategist at Goldman Sachs
- Wall Street career mostly in Rates and Mortgage Derivatives
- Educational background: Algorithms Theory and Abstract Algebra
- I direct Stanford's Mathematical & Computational Finance program
- Research & Teaching in: *RL and it's applications in Finance & Retail*
- In-progress book: Foundations of RL with Applications in Finance
- Book blends Theory, Modeling, Algorithms, Python, Trading problems
- Emphasis on broader principles in Applied Math & Software Design
- I spend a lot of time writing code associated with the book

AI for Dynamic Decisioning under Uncertainty

- Let's browse some terms used to characterize this branch of AI
- *Stochastic*: Uncertainty in key quantities, evolving over time
- *Optimization*: A well-defined metric to be maximized ("The Goal")
- *Dynamic*: Decisions need to be a function of the changing situations
- *Control*: Overpower uncertainty by persistent steering towards goal
- Jargon overload due to confluence of Control Theory, OR and AI
- 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

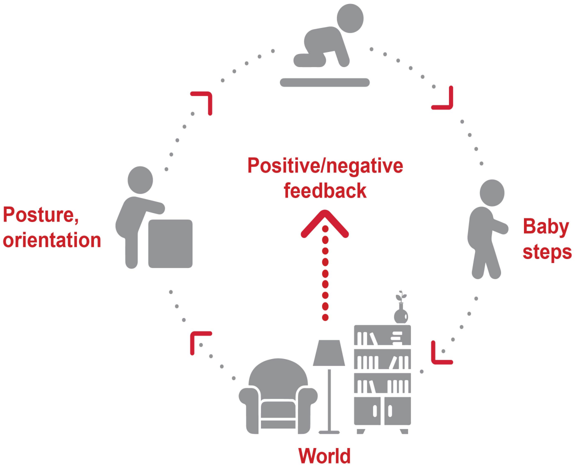
- The *Agent* and the *Environment* interact in a time-sequenced loop
- *Agent* responds to [*State*, *Reward*] by taking an *Action*
- *Environment* responds by producing next step's (random) *State*
- *Environment* also produces a (random) scalar denoted as *Reward*
- Each *State* is assumed to have 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 $p(r, s'|s, a) = \mathbb{P}[(R_{t+1} = r, S_{t+1} = s')|S_t = s, A_t = a]$
- $\gamma \in [0, 1]$ 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 $s \in \mathcal{S}$

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

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

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

- *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

- 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 AI, DP is a type of *Planning Algorithm*
- DP algorithms are iterative algorithms based on Fixed-Point Theorem
- Finding a *Fixed Point* of Operator based on Bellman Equation
- Why is DP not effective in practice?
  - Curse of Dimensionality
  - Curse of Modeling
- Curse of Dimensionality can be partially cured with Approximate DP
- To resolve both curses effectively, we need RL

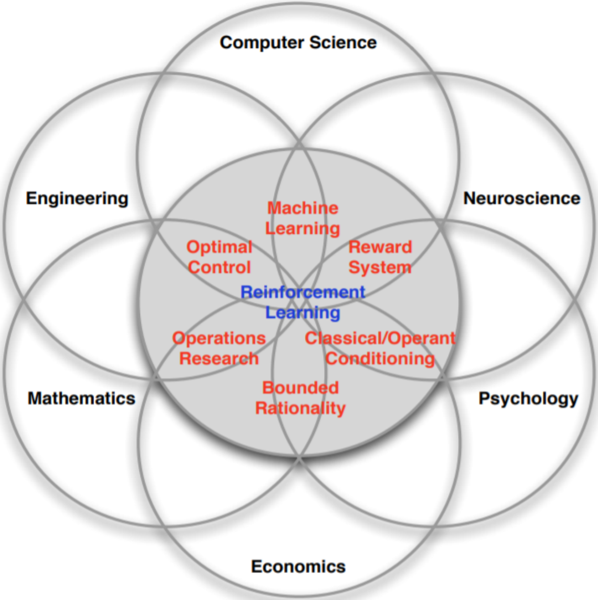
Reinforcement Learning

- Typically in real-world, we don't have access to a Probabilities Model
- All we have is access to an environment serving individual transitions
- Even if MDP model is available, model updates can be challenging
- Often real-world models end up being too large or too complex
- Sometimes estimating a *sampling model* is much more feasible
- So RL interacts with either *actual* or *simulated* environment
- Either way, we receive *individual transitions* to next state and reward
- RL is a "trial-and-error" approach linking *Actions* to *Returns*
- Try different actions & learn what works, what doesn't
- This is hard because actions have overlapping reward sequences
- Also, sometimes Actions result in *delayed Rewards*

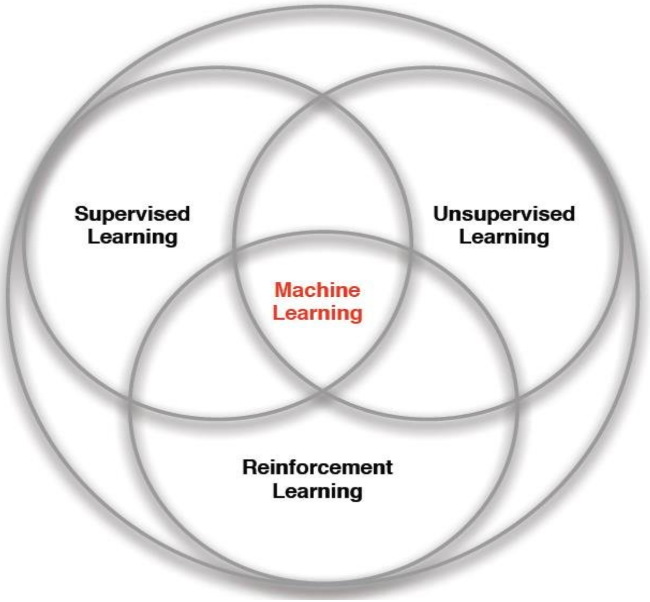
RL: Learning Value Function Approximation from Samples

- RL incrementally learns the Value Function from transitions data
- Appropriate Approximation of Value Function is key to success
- Deep Neural Networks are typically used for function approximation
- Big Picture: Sampling and Function Approximation come together
- RL algorithms are clever about balancing "explore" versus "exploit"
- Most RL Algorithms are founded on the Bellman Equations
- **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
- Possibilities in Finance are endless (we cover 5 important problems)
- Studying 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



P1: Dynamic Asset-Allocation and Consumption

- The broad topic is Investment Management
- Applies to Corporations as well as Individuals
- The two considerations are:
  - How to allocate money across assets in one's investment portfolio
  - How much to consume for one's needs/operations/pleasures
- We consider the dynamic version of these dual considerations
- Asset-Allocation and Consumption decisions at each time step
- Asset-Allocation decisions typically deal with Risk-Reward tradeoffs
- Consumption decisions are about spending now or later
- Objective: Horizon-Aggregated Expected Utility of Consumption

P1: Consider the simple example of Personal Finance

- Broadly speaking, Personal Finance involves the following aspects:
  - Receiving Money: Salary, Bonus, Rental income, Asset Liquidation etc.
  - Consuming Money: Food, Clothes, Rent/Mortgage, Car, Vacations etc.
  - Investing Money: Savings account, Stocks, Real-estate, Gold etc.
- Goal: Maximize lifetime-aggregated Expected Utility of Consumption
- This can be modeled as a Markov Decision Process
- *State:* Age, Asset Holdings, Asset Valuation, Career situation etc.
- *Action:* Changes in Asset Holdings, Optional Consumption
- *Reward:* Utility of Consumption of Money
- *Model:* Career uncertainties, Asset market uncertainties

P1: Merton's Frictionless Continuous-Time Formulation

- Assume: Current wealth is $W_0 > 0$, and you'll live for T more years
- You can invest in (allocate to) n risky assets and a riskless asset
- Each risky 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 Expected Utility of Consumption
- Consumption Utility assumed to have Constant Relative Risk-Aversion

P1: Problem Notation

For simplicity, consider the case of 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 per unit time denoted by $c(t, W_t) \geq 0$
- Utility of Consumption function $U(x) = \frac{x^{1-\gamma}}{1-\gamma}$ for $0 < \gamma \neq 1$
- $\gamma =$ (Constant) Relative Risk-Aversion $\frac{-x \cdot U''(x)}{U'(x)}$

P1: Formal Problem Statement

- Write π_t, c_t instead of $\pi(t, W_t), c(t, W_t)$ to lighten notation
- 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:

$$\mathbb{E}[\int_t^T \frac{e^{-\rho(s-t)} \cdot c_s^{1-\gamma}}{1-\gamma} \cdot ds + \frac{e^{-\rho(T-t)} \cdot B \cdot W_T^{1-\gamma}}{1-\gamma} \mid W_t]$$

  where $\rho \geq 0$ is the utility discount rate, B is the bequest
- We can solve this problem for arbitrary bequest but Merton considers $B = \epsilon^\gamma$ where $0 < \epsilon \ll 1$, meaning "zero" bequest

P1: Continuous-Time Stochastic Control

- Think of this as a continuous-time Stochastic Control problem
- The *State* at time t is (t, W_t)
- The *Action* at time t is $[\pi_t, c_t]$
- The *Reward* per unit time at time t is $U(c_t) = \frac{c_t^{1-\gamma}}{1-\gamma}$
- The *Return* at time t is the accumulated discounted *Reward*:

$$\int_t^T e^{-\rho(s-t)} \cdot \frac{c_s^{1-\gamma}}{1-\gamma} \cdot ds$$

- 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

P1: Optimal Allocation and Consumption

HJB-based solution yields:

$$\pi^*(t, W_t) = \frac{\mu - r}{\sigma^2 \gamma}$$

$$c^*(t, W_t) = \frac{W_t}{f(t)}$$

HJB Formulation is key and this solution approach provides a template for similar continuous-time stochastic control problems.

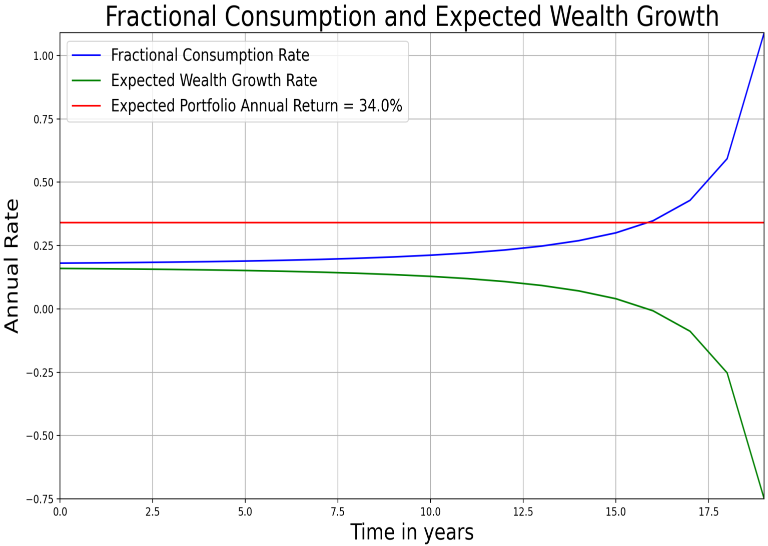
P1: Gaining Insights into the Solution

- Optimal Allocation $\pi^*(t, W_t)$ is constant (independent of t and W_t)
- Optimal Fractional Consumption $\frac{c^*(t,W_t)}{W_t}$ depends only on t ($= \frac{1}{f(t)}$)
- With Optimal Allocation & Consumption, the Wealth process is:

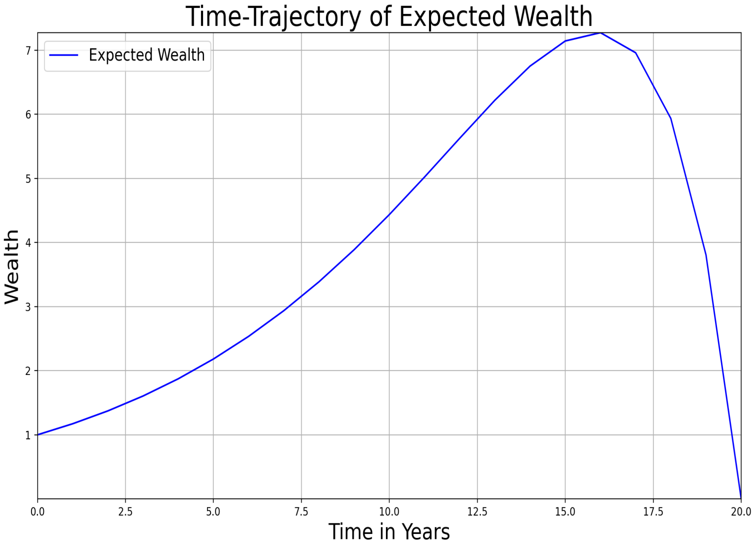
$$\frac{dW_t}{W_t} = (r + \frac{(\mu - r)^2}{\sigma^2 \gamma} - \frac{1}{f(t)}) \cdot dt + \frac{\mu - r}{\sigma \gamma} \cdot dz_t$$

- Expected Portfolio Return is constant over time ($= r + \frac{(\mu-r)^2}{\sigma^2\gamma}$)
- Fractional Consumption $\frac{1}{f(t)}$ increases over time
- Expected Rate of Wealth Growth $r + \frac{(\mu-r)^2}{\sigma^2\gamma} - \frac{1}{f(t)}$ decreases over time
- If $r + \frac{(\mu-r)^2}{\sigma^2\gamma} > \frac{1}{f(0)}$, we start by Consuming $<$ Expected Portfolio Growth and over time, we Consume $>$ Expected Portfolio Growth
- Wealth Growth Volatility is constant ($= \frac{\mu-r}{\sigma\gamma}$)

P1:Fractional Consumption and Expected Wealth Growth



P1: Time-Trajectory of Expected Wealth



P1: Real-World

- Analytical tractability in Merton's formulation is 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
- Practically we cannot hope to estimate a *Probabilities Model*
- Build a practical simulator of market/constraints and do RL
- Large Action Space points to Policy Gradient Algorithms

P2: Classical Pricing and Hedging of Derivatives

- Classical Pricing/Hedging Theory is based on a few core concepts:
  - **Arbitrage-Free Market** - where you cannot make money from nothing
  - **Replication** - when the payoff of a *Derivative* can be constructed by assembling (and rebalancing) a portfolio of the underlying securities
  - **Complete Market** - where payoffs of all derivatives can be replicated
  - **Risk-Neutral Measure** - Altered probability measure for movements of underlying securities for mathematical convenience in pricing
- Assumptions of arbitrage-free and completeness lead to (dynamic, exact, unique) replication of derivatives with the underlying securities
- Assumptions of frictionless trading provide these idealistic conditions
- Frictionless := continuous trading, any volume, no transaction costs
- Replication strategy gives us the pricing and hedging solutions
- This is the foundation of the famous Black-Scholes formulas
- However, the real-world has many frictions $\Rightarrow$ *Incomplete Market*
- ... where derivatives cannot be exactly replicated

P2: Pricing and Hedging in an Incomplete Market

- In an incomplete market, we have multiple risk-neutral measures
- So, multiple derivative prices (each consistent with no-arbitrage)
- The market/trader "chooses" a risk-neutral measure (hence, price)
- This "choice" is typically made in ad-hoc and inconsistent ways
- Alternative approach is for a trader to play *Portfolio Optimization*
- Maximizing "risk-adjusted return" of the derivative plus hedges
- Based on a specified preference for trading risk versus return
- This preference is equivalent to specifying a Utility function
- Reminiscent of the Portfolio Optimization problem we've seen before
- Likewise, we can set this up as a stochastic control (MDP) problem
- Where the decision at each time step is: *Trades in the hedges*
- So what's the best way to solve this MDP?

P2: Deep Reinforcement Learning (DRL)

- Dynamic Programming not suitable in practice due to:
  - Curse of Dimensionality
  - Curse of Modeling
- So we solve the MDP with *Deep Reinforcement Learning* (DRL)
- The idea is to use real market data and real market frictions
- Developing realistic simulations to derive the optimal policy
- The optimal policy gives us the (practical) hedging strategy
- The optimal value function gives us the price (valuation)
- Formulation based on Deep Hedging paper by J.P.Morgan researchers
- More details in the prior paper by some of the same authors

P2: Problem Setup

- Let's simplify the problem setup a bit for ease of exposition
- This model works for more complex, more frictionful markets too
- Assume time is in discrete (finite) steps $t = 0, 1, \dots, T$
- Assume we have a position (portfolio) D in m derivatives
- Assume each of these m derivatives expires in time $\leq T$
- Portfolio-aggregated *Contingent Cashflows* at time t denoted $X_t \in \mathbb{R}$
- Assume we have n underlying market securities as potential hedges
- Hedge positions (units held) at time t denoted $\boldsymbol{\alpha}_t \in \mathbb{R}^n$
- Cashflows per unit of hedges held at time t denoted $\boldsymbol{Y}_t \in \mathbb{R}^n$
- Prices per unit of hedges at time t denoted $\boldsymbol{P}_t \in \mathbb{R}^n$
- PnL position at time t is denoted as $\beta_t \in \mathbb{R}$

P2: States and Actions

- Among other things, *State* $s_t \in \mathcal{S}_t$ contains $\boldsymbol{\alpha}_t, \boldsymbol{P}_t, \beta_t, D$
- s_t will include any market information relevant to trading actions
- For simplicity, we assume s_t is just the tuple $(\boldsymbol{\alpha}_t, \boldsymbol{P}_t, \beta_t, D)$
- Denote *Action* at time t as $\boldsymbol{a}_t \in \mathcal{A}_t$
- a_t represents units of hedges traded (positive for buy, negative for sell)
- Trading restrictions (eg: no short-selling) define $\mathcal{A}_t$ as a function of s_t
- State transitions $\boldsymbol{P}_{t+1}|\boldsymbol{P}_t$ available from a *simulator*, whose internals are estimated from real market data and realistic assumptions

P2: Sequence of events at each time step $t = 0, \ldots, T$

1. Observe state $s_t = (\boldsymbol{\alpha}_t, \boldsymbol{P}_t, \beta_t, D)$
2. Perform action (trades) a_t to produce trading PnL $= -\boldsymbol{a}_t^T \cdot \boldsymbol{P}_t$
3. Trading transaction costs, eg. $= -\gamma \cdot abs(\boldsymbol{a}_t^T) \cdot \boldsymbol{P}_t$ for some $\gamma > 0$
4. Update $\boldsymbol{\alpha}_t$ as: $\boldsymbol{\alpha}_{t+1} = \boldsymbol{\alpha}_t + \boldsymbol{a}_t$ (force-liquidation at $T \Rightarrow \boldsymbol{a}_T = -\boldsymbol{\alpha}_T$)
5. Realize cashflows (from updated positions) $= X_{t+1} + \boldsymbol{\alpha}_{t+1}^T \cdot \boldsymbol{Y}_{t+1}$
6. Update PnL β_t as:

$$\beta_{t+1} = \beta_t - \boldsymbol{a}_t^T \cdot \boldsymbol{P}_t - \gamma \cdot abs(\boldsymbol{a}_t^T) \cdot \boldsymbol{P}_t + X_{t+1} + \boldsymbol{\alpha}_{t+1}^T \cdot \boldsymbol{Y}_{t+1}$$

7. Reward $r_t = 0$ for all $t = 0, \ldots, T-1$ and $r_T = U(\beta_{T+1})$ for an appropriate concave Utility function U (based on risk-aversion)
8. Simulator evolves hedge prices from $\boldsymbol{P}_t$ to $\boldsymbol{P}_{t+1}$

P2: Pricing and Hedging

- Assume we now want to enter into an incremental position (portfolio) D' in m' derivatives (denote the combined position as $D \cup D'$)
- We want to determine the *Price* of the incremental position D', as well as the hedging strategy for D'
- Denote the Optimal Value Function at time t as $V_t^* : \mathcal{S}_t \rightarrow \mathbb{R}$
- Pricing of D' is based on the principle that introducing the incremental position of D' together with a calibrated cashflow (Price) at $t = 0$ should leave the Optimal Value (at $t = 0$) unchanged
- Precisely, Price of D' is the value x such that

$$V_0^*((\boldsymbol{\alpha}_0, \boldsymbol{P}_0, \beta_0 - x, D \cup D')) = V_0^*((\boldsymbol{\alpha}_0, \boldsymbol{P}_0, \beta_0, D))$$

- This Pricing principle is known as the principle of *Indifference Pricing*
- The hedging strategy at time t for all $0 \leq t < T$ is given by the Optimal Policy $\pi_t^* : \mathcal{S}_t \rightarrow \mathcal{A}_t$

P2: DRL Approach relevant for Practical Trading?

- The industry practice/tradition has been to start with *Complete Market* assumption, and then layer ad-hoc/unsatisfactory adjustments
- There is some past work on pricing/hedging in incomplete markets
- But it's theoretical and not usable in real trading (eg: Superhedging)
- This DRL approach should be explored more for practical trading
- Key advantages of this DRL approach:
  - Algorithm for pricing/hedging independent of market dynamics
  - Computational cost scales efficiently with size m of derivatives portfolio
  - Enables one to faithfully capture practical trading situations/constraints
  - Deep Neural Networks provide great function approximation for RL

P3: Stopping Time

- Stopping time τ is a "random time" (random variable) interpreted as time at which a given stochastic process exhibits certain behavior
- Stopping time often defined by a "stopping policy" to decide whether to continue/stop a process based on present position and past events
- Deciding whether $\tau \leq t$ only depends on information up to time t
- Hitting time of a Borel set A for a process X_t is the first time X_t takes a value within the set A
- Hitting time is an example of stopping time. Formally,

$$T_{X,A} = \min\{t \in \mathbb{R} | X_t \in A\}$$

eg: Hitting time of a process to exceed a certain fixed level

P3: Optimal Stopping Problem

- Optimal Stopping problem for Stochastic Process X_t:

$$W(x) = \max_{\tau} \mathbb{E}[H(X_\tau)|X_0 = x]$$

  where τ is a set of stopping times of X_t, $W(\cdot)$ is called the Value function, and H is the Reward function.
- Note that sometimes we can have several stopping times that maximize $\mathbb{E}[H(X_\tau)]$ and we say that the optimal stopping time is the smallest stopping time achieving the maximum value.
- Example of Optimal Stopping: Optimal Exercise of American Options
  - X_t is risk-neutral process for underlying security's price
  - x is underlying security's current price
  - τ is set of exercise times corresponding to various stopping policies
  - $W(\cdot)$ is American option price as function of underlying's current price
  - $H(\cdot)$ is the option payoff function, adjusted for time-discounting

P3: Optimal Stopping Problems as MDPs

- We formulate Stopping Time problems as Markov Decision Processes
- *State* is X_t
- *Action* is Boolean: Stop or Continue
- *Reward* always 0, except upon Stopping (when it is $= H(X_\tau)$)
- *State*-transitions governed by the Stochastic Process X_t
- For discrete time steps, the Bellman Optimality Equation is:

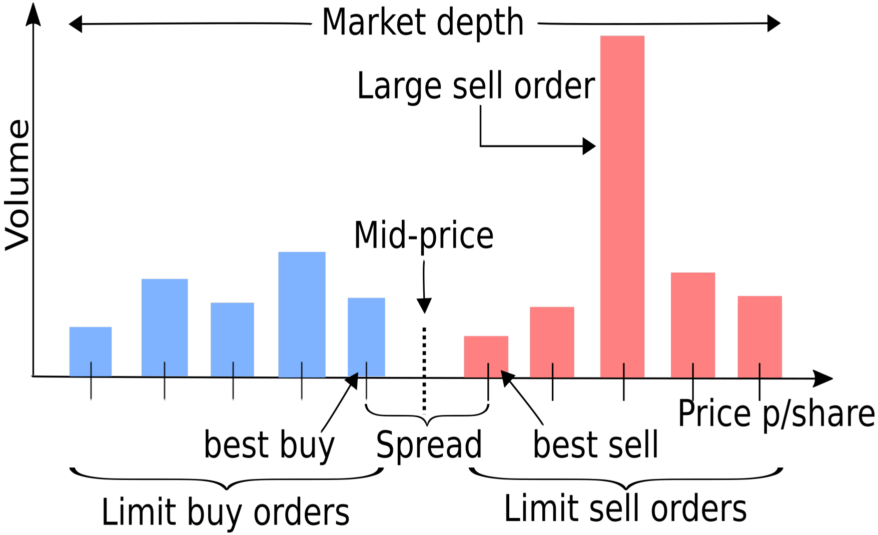
$$V^*(X_t) = \max(H(X_t), \mathbb{E}[V^*(X_{t+1})|X_t])$$

- For finite number of time steps, we can do a simple backward induction algorithm from final time step back to time step 0

P3: Mainstream approaches to American Option Pricing

- American Option Pricing is Optimal Stopping, and hence an MDP
- So can be tackled with Dynamic Programming or RL algorithms
- But let us first review the mainstream approaches
- For some American options, just price the European, eg: vanilla call
- When payoff is not path-dependent and state dimension is not large, we can do backward induction on a binomial/trinomial tree/grid
- Otherwise, the standard approach is Longstaff-Schwartz algorithm
- Longstaff-Schwartz algorithm combines 3 ideas:
    - Valuation based on Monte-Carlo simulation
    - Function approximation of continuation value for in-the-money states
    - Backward-recursive determination of early exercise states
- RL is an attractive alternative to Longstaff-Schwartz algorithm
- LSPI and Deep Q-Learning solutions sketched here

P4: Trading Order Book (abbrev. OB)



P4: Basics of Order Book (OB)

- Buyers/Sellers express their intent to trade by submitting bids/asks
- These are Limit Orders (LO) with a price P and size N
- Buy LO (P, N) states willingness to buy N shares at a price $\leq P$
- Sell LO (P, N) states willingness to sell N shares at a price $\geq P$
- Order Book aggregates order sizes for each unique price
- So we can represent with two sorted lists of (Price, Size) pairs

$$\text{Bids: } [(P_i^{(b)}, N_i^{(b)}) \mid 0 \leq i < m], P_i^{(b)} > P_j^{(b)} \text{ for } i < j$$

$$\text{Asks: } [(P_i^{(a)}, N_i^{(a)}) \mid 0 \leq i < n], P_i^{(a)} < P_j^{(a)} \text{ for } i < j$$

- We call $P_0^{(b)}$ as simply *Bid*, $P_0^{(a)}$ as *Ask*, $\frac{P_0^{(a)}+P_0^{(b)}}{2}$ as *Mid*
- We call $P_0^{(a)} - P_0^{(b)}$ as *Spread*, $P_{n-1}^{(a)} - P_{m-1}^{(b)}$ as *Market Depth*
- A Market Order (MO) states intent to buy/sell N shares at the *best possible price(s)* available on the OB at the time of MO submission

P4: Order Book (OB) Activity

- A new Sell LO (P, N) potentially removes best bid prices on the OB

$$\text{Removal: } [(P_i^{(b)}, \min(N_i^{(b)}, \max(0, N - \sum_{j=0}^{i-1} N_j^{(b)}))) \mid (i : P_i^{(b)} \geq P)]$$

- After this removal, it adds the following to the asks side of the OB

$$(P, \max(0, N - \sum_{i:P_i^{(b)} \geq P} N_i^{(b)}))$$

- A new Buy LO operates analogously (on the other side of the OB)
- A Sell Market Order N will remove the best bid prices on the OB

$$\text{Removal: } [(P_i^{(b)}, \min(N_i^{(b)}, \max(0, N - \sum_{j=0}^{i-1} N_j^{(b)}))) \mid 0 \leq i < m]$$

- A Buy Market Order N will remove the best ask prices on the OB

$$\text{Removal: } [(P_i^{(a)}, \min(N_i^{(a)}, \max(0, N - \sum_{j=0}^{i-1} N_j^{(a)}))) \mid 0 \leq i < n]$$

P4: Price Impact and Order Book Dynamics

- We focus on how a Market order (MO) alters the OB
- A large-sized MO often results in a big *Spread* which could soon be replenished by new LOs, potentially from either side
- So a large-sized MO moves the Bid/Ask/Mid (*Price Impact* of MO)
- Subsequent Replenishment activity is part of *OB Dynamics*
- Models for OB Dynamics can be quite complex

P4: Optimal Trade Order Execution Problem

- The task is to sell a large number N of shares
- We are allowed to trade in T discrete time steps
- We are only allowed to submit Market Orders
- Need to consider both *Temporary* and *Permanent* Price Impact
- For simplicity, consider a model of just the *Bid Price* Dynamics
- Goal is to maximize Expected Total Utility of Sales Proceeds
- By breaking N into appropriate chunks (timed appropriately)
- If we sell too fast, we are likely to get poor prices
- If we sell too slow, we risk running out of time
- Selling slowly also leads to more uncertain proceeds (lower Utility)
- This is a Dynamic Optimization problem
- We can model this problem as a Markov Decision Process (MDP)

P4: Problem Notation

- Time steps indexed by $t = 0, 1, \ldots, T$
- P_t denotes Bid Price at start of time step t
- N_t denotes number of shares sold in time step t
- $R_t = N - \sum_{i=0}^{t-1} N_i$ = shares remaining to be sold at start of step t
- $R_0 = N, R_{t+1} = R_t - N_t$ for all $t < T, N_{T-1} = R_{T-1} \Rightarrow R_T = 0$
- Price Dynamics given by:

$$P_{t+1} = f_t(P_t, N_t, \epsilon_t)$$

  where $f_t(\cdot)$ is an arbitrary function incorporating:
  - Permanent Price Impact of selling N_t shares
  - Impact-independent market-movement of Bid Price over time step t
  - ϵ_t denotes source of randomness in Bid Price market-movement
- Sales Proceeds in time step t defined as:

$$N_t \cdot Q_t = N_t \cdot (P_t - g_t(P_t, N_t))$$

  where $g_t(\cdot)$ is an arbitrary func representing Temporary Price Impact
- Utility of Sales Proceeds function denoted as $U(\cdot)$

P4: Markov Decision Process (MDP) Formulation

- This is a discrete-time, finite-horizon MDP
- MDP Horizon is time T, meaning all states at time T are terminal
- Order of MDP activity in each time step $0 \leq t < T$:
  - Observe *State* $s_t := (P_t, R_t) \in \mathcal{S}_t$
  - Perform *Action* $a_t := N_t \in \mathcal{A}_t$
  - Receive *Reward* $r_{t+1} := U(N_t \cdot Q_t) = U(N_t \cdot (P_t - g_t(P_t, N_t)))$
  - Experience Price Dynamics $P_{t+1} = f_t(P_t, N_t, \epsilon_t)$
- Goal is to find a Policy $\pi_t^*((P_t, R_t)) = N_t^*$ that maximizes:

$$\mathbb{E}[\sum_{t=0}^{T-1} \gamma^t \cdot U(N_t \cdot Q_t)] \text{ where } \gamma \text{ is MDP discount factor}$$

- Closed-form solutions by Bertsimas-Lo
- Risk-Aversion considerations by Almgren-Chriss

P4: Real-world Optimal Trade Order Execution

- Arbitrary Price Dynamics $f_t(\cdot)$ and Temporary Price Impact $g_t(\cdot)$
- Non-stationarity/non-linear dynamics/impact require (Numerical) DP
- Frictions: Discrete Prices/Sizes, Constraints on Prices/Sizes, Fees
- Incorporating various markets factors in the State bloats State Space
- We could also represent the entire OB within the State
- Practical route is to develop a simulator capturing all of the above
- Simulator is a *Market-Data-learnt Sampling Model* of OB Dynamics
- In practice, we'd need to also capture *Cross-Asset Market Impact*
- Using this simulator and neural-networks func approx, we can do RL
- References: Nevmyvaka, Feng, Kearns; 2006 and Vyetrenko, Xu; 2019
- Exciting area for Future Research as well as Engineering Design

P5: OB Dynamics and Market-Making

- Modeling OB Dynamics involves predicting arrival of MOs and LOs
- Market-makers are liquidity providers (providers of Buy and Sell LOs)
- Other market participants are typically liquidity takers (MOs)
- But there are also other market participants that trade with LOs
- Complex interplay between market-makers & other mkt participants
- Hence, OB Dynamics tend to be quite complex
- We view the OB from the perspective of a single market-maker who aims to gain with Buy/Sell LOs of appropriate width/size
- By anticipating OB Dynamics & dynamically adjusting Buy/Sell LOs
- Goal is to maximize *Utility of Gains* at the end of a suitable horizon
- If Buy/Sell LOs are too narrow, more frequent but small gains
- If Buy/Sell LOs are too wide, less frequent but large gains
- Market-maker also needs to manage potential unfavorable inventory (long or short) buildup and consequent unfavorable liquidation

P5: Notation for Optimal Market-Making Problem

- We simplify the setting for ease of exposition
- Assume finite time steps indexed by $t = 0, 1, \ldots, T$
- Denote $W_t \in \mathbb{R}$ as Market-maker's trading PnL at time t
- Denote $I_t \in \mathbb{Z}$ as Market-maker's inventory of shares at t ($I_0 = 0$)
- $S_t \in \mathbb{R}^+$ is the OB Mid Price at time t (assume stochastic process)
- $P_t^{(b)} \in \mathbb{R}^+, N_t^{(b)} \in \mathbb{Z}^+$ are market maker's Bid Price, Bid Size at t
- $P_t^{(a)} \in \mathbb{R}^+, N_t^{(a)} \in \mathbb{Z}^+$ are market-maker's Ask Price, Ask Size at t
- Assume market-maker can add or remove bids/asks costlessly
- Random var $X_t^{(b)} \in \mathbb{Z}_{\geq 0}$ denotes bid-shares "hit" *up to* time t
- Random var $X_t^{(a)} \in \mathbb{Z}_{\geq 0}$ denotes ask-shares "lifted" *up to* time t

$$W_{t+1} = W_t + P_t^{(a)} \cdot (X_{t+1}^{(a)} - X_t^{(a)}) - P_t^{(b)} \cdot (X_{t+1}^{(b)} - X_t^{(b)}) \; , \; I_t = X_t^{(b)} - X_t^{(a)}$$

- Goal to maximize $\mathbb{E}[U(W_T + I_T \cdot S_T)]$ for appropriate concave $U(\cdot)$

P5: Markov Decision Process (MDP) Formulation

- Order of MDP activity in each time step $0 \leq t \leq T-1$:
  - Observe *State* $:= (S_t, W_t, I_t) \in \mathcal{S}_t$
  - Perform *Action* $:= (P_t^{(b)}, N_t^{(b)}, P_t^{(a)}, N_t^{(a)}) \in \mathcal{A}_t$
  - Experience OB Dynamics resulting in:
    - random bid-shares hit $= X_{t+1}^{(b)} - X_t^{(b)}$ and ask-shares lifted $= X_{t+1}^{(a)} - X_t^{(a)}$
    - update of W_t to W_{t+1}, update of I_t to I_{t+1}
    - stochastic evolution of S_t to S_{t+1}
  - Receive next-step $(t+1)$ *Reward* R_{t+1}

$$R_{t+1} := \begin{cases} 0 & \text{for } 1 \leq t+1 \leq T-1 \\ U(W_{t+1} + I_{t+1} \cdot S_{t+1}) & \text{for } t+1 = T \end{cases}$$

- Goal is to find an *Optimal Policy* $\pi^* = (\pi_0^*, \pi_1^*, \ldots, \pi_{T-1}^*)$:

$$\pi_t^*((S_t, W_t, I_t)) = (P_t^{(b)}, N_t^{(b)}, P_t^{(a)}, N_t^{(a)}) \text{ that maximizes } \mathbb{E}[R_T]$$

P5: Real-world Market-Making and RL

- Avellaneda and Stoikov derive a closed-form solution for a simple continuous-time formulation
- Real-world OB dynamics are non-stationary, non-linear, complex
- Frictions: Discrete Prices/Sizes, Constraints on Prices/Sizes, Fees
- Need to capture various market factors in the *State* & OB Dynamics
- This leads to Curse of Dimensionality and Curse of Modeling
- The practical route is to develop a simulator capturing all of the above
- Simulator is a *Market-Data-learnt Sampling Model* of OB Dynamics
- Using this simulator and neural-networks func approx, we can do RL
- References: 2018 Paper from University of Liverpool and 2019 Paper from JP Morgan Research
- Exciting area for Future Research as well as Engineering Design