Integrating Machine Learning and Optimization for Inventory Management

Ziang Liu

Faculty of Environmental, Life, Natural Science and Technology
Okayama University

MOAI, January 26, 2024

Table of Contents

References

Books

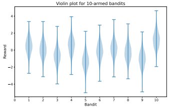
- Sutton, Richard S., and Andrew G. Barto. 2018. Reinforcement Learning, Second Edition: An Introduction. MIT Press.
- Snyder, Lawrence V., and Zuo-Jun Max Shen. 2019. Fundamentals of Supply Chain Theory. John Wiley & Sons.
- Puterman, Martin L. 1990. “Chapter 8 Markov Decision Processes.” In Handbooks in Operations Research and Management Science, 2:331-434. Elsevier.
- Kochenderfer, Mykel J., and Tim A. Wheeler. 2019. Algorithms for Optimization. MIT Press.
- Jin, Yaochu, Handing Wang, and Chaoli Sun. 2021. Data-Driven Evolutionary Optimization: Integrating Evolutionary Computation, Machine Learning and Data Science. Springer Nature.

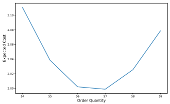
Webpages

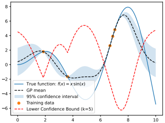
- “UCL Course on RL.” 2019. David Silver. December 19, 2019. https://www.davidsilver.uk/teaching/.
- Snyder, Larry. n.d. RL for Inventory Optimization. Github. Accessed January 4, 2024. https://github.com/LarrySnyder/RLforInventory.
- Poupart, Pascal. n.d. “CS885 Spring 2020 - Reinforcement Learning.” Accessed January 4, 2024. https://cs.uwaterloo.ca/ ppoupart/teaching/cs885-spring20/schedule.html.

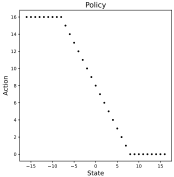
Source Code

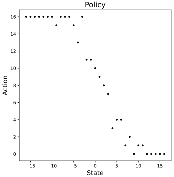
Source code is available on GitHub.
https://github.com/zi-ang-liu/ML-and-Opt-for-Inventory

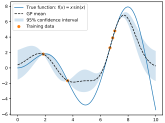












Mindmap

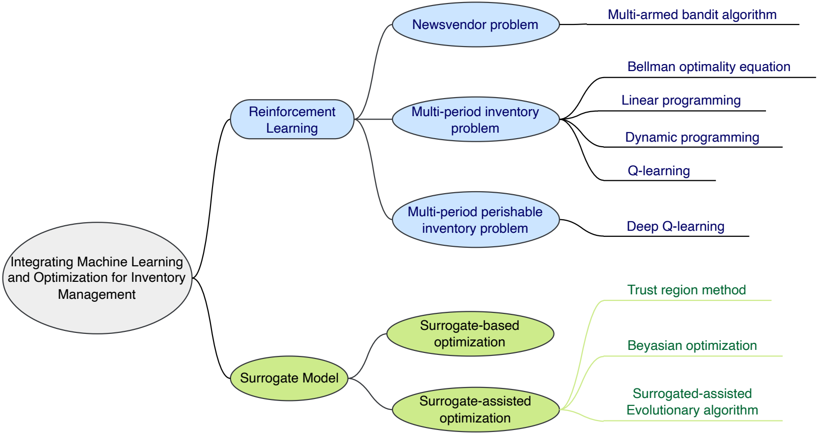

Integrating Machine Learning and Optimization for Inventory Management
Reinforcement Learning
Newsvendor problem
Multi-armed bandit algorithm
Multi-period inventory problem
Bellman optimality equation
Linear programming
Dynamic programming
Q-learning
Multi-period perishable inventory problem
Deep Q-learning
Surrogate Model
Surrogate-based optimization
Surrogate-assisted optimization
Trust region method
Beyasian optimization
Surrogated-assisted Evolutionary algorithm

Table of Contents

The newsvendor problem: Description

- A newspaper vendor must decide how many copies of a newspaper to order each morning
- The demand is uncertain
- Overage cost occurs when the demand is less than the order quantity
- Underage cost occurs when the demand is greater than the order quantity
- What is the optimal order quantity?

The newsvendor problem: Formulation

Parameters:

- D: Demand (random variable)
- h: Holding cost (overage cost)
- p: Stockout cost (shortage cost)

Decision variables:

- Q: Order quantity

Objective:

- $g(Q) = \mathbb{E}\left[h(Q-D)^+ + p(D-Q)^+\right]$
- $(x)^+ = \max\{x, 0\}$

The newsvendor problem: Optimization

Objective function:

$$\mathbb{E}\left[h \max\{Q-D,0\} + p \max\{D-Q,0\}\right]$$
$$= h\int_0^Q (Q-d)f(d)dd + p\int_Q^\infty (d-Q)f(d)dd$$

First order derivative: $hF(Q) - p(1-F(Q))$
Second order derivative: $hf(Q) + pf(Q)$
Optimal order quantity: $Q^* = F^{-1}\left(\frac{h}{h+p}\right)$

The newsvendor problem: Example

- $D \sim \mathcal{N}(50, 8^2)$
- $h = 0.18$, $p = 0.7$

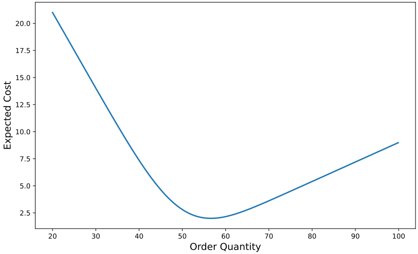


Figure: Continuous newsvendor problem

The discrete newsvendor problem: Example

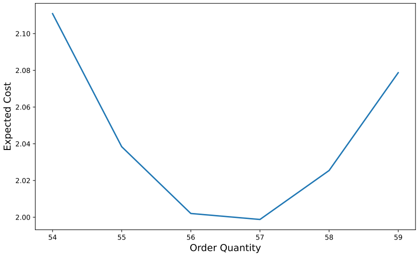


Figure: Discrete newsvendor problem

$g(Q)$ is still convex.

Multi-armed bandit problem

Problem statement:

- k different actions we can take
- After each action, we receive a reward from a stationary probability distribution
- The reward distribution is unknown
- Objective: maximize the total reward over some time period

The k-armed bandit problem: Example

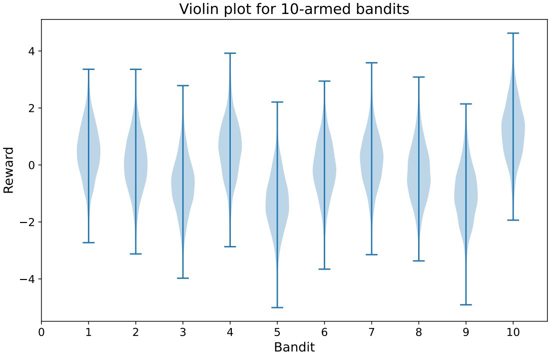


Figure: The 10-armed bandit problem

The k-armed bandit problem: Notations

- A_t: Action at time t
- R_t: Reward at time t
- $q_*(a)$: Expected reward for taking action a, $q_*(a) = \mathbb{E}\left[R_t | A_t = a\right]$
- $Q_t(a)$: Estimated value of action a at time t, $Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \mathbb{I}\{A_i = a\}}{\sum_{i=1}^{t-1} \mathbb{I}\{A_i = a\}}$

[0] $\mathbb{I}\left\{A_i = a\right\}$ is an indicator function, $\mathbb{I}\left\{A_i = a\right\} = 1$ if $A_i = a$, otherwise $\mathbb{I}\left\{A_i = a\right\} = 0$

Incremental implementation of sample averages

$$
\begin{aligned}
Q_{n+1} &= \frac{R_1 + R_2 + \cdots + R_n}{n} \\
&= \frac{1}{n} \sum_{i=1}^{n} R_i \\
&= \frac{1}{n} \left(R_n + \sum_{i=1}^{n-1} R_i \right) \\
&= \frac{1}{n} \left(R_n + (n-1) \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1} \right) \\
&= \frac{1}{n} (R_n + (n-1) Q_n) \\
&= Q_n + \frac{1}{n} (R_n - Q_n)
\end{aligned}
$$

The k-armed bandit problem: ϵ-greedy policy

- with probability $1 - \epsilon$, choose the action with the highest estimated value, $A_t = \arg\max_a Q_t(a)$
- with probability ϵ, choose an action randomly

A simple bandit algorithm

Algorithm 1: A simple bandit algorithm

Initialize, for $a = 1, \ldots, k$:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$
for $t = 1, 2, \ldots$ **do**
  Choose A_t using ϵ-greedy policy based on Q_t
  $R_t \leftarrow \text{bandit}(A_t)$
  $N(A_t) \leftarrow N(A_t) + 1$
  $Q(A_t) \leftarrow Q(A_t) + \frac{1}{N(A_t)}(R_t - Q(A_t))$
end

Solving newsvendor problem as a k-armed bandit problem

discrete newsvendor problem

- $h = 0.18$, $p = 0.7$, $D \sim \mathcal{N}(5, 1^2)$, D is discrete
- Optimal order quantity: 6, Expected Cost: 0.24446

simple bandit algorithm ($\epsilon = 0.01$, $k = 10$, $T = 2000$)

Optimal Action	Q Value
6	-0.24113
6	-0.24389
6	-0.24330
6	-0.24549
6	-0.23680

Table: Optimal actions and corresponding Q values

k-armed bandit problem and newsvendor problem

Summary

- k-armed bandit problem is nonassociative
- Only need to find the best action
- The newsvendor problem: Perishable, Lost sales
- The newsvendor problem can be solved as k-armed bandit problem

Next: Associative problem

- need to learn a policy (a mapping from states to actions)
- Multi-period inventory problem

Inventory Policy

- **Constant order quantity policy**: Order Q units at each period.
- **Base-stock policy**: Order up to S units when the inventory level is less than S, otherwise do not order.
- (s, S) **policy**: Order up to S units when the inventory level is less than s, otherwise do not order.

Inventory policy is a mapping from inventory level to order quantity.

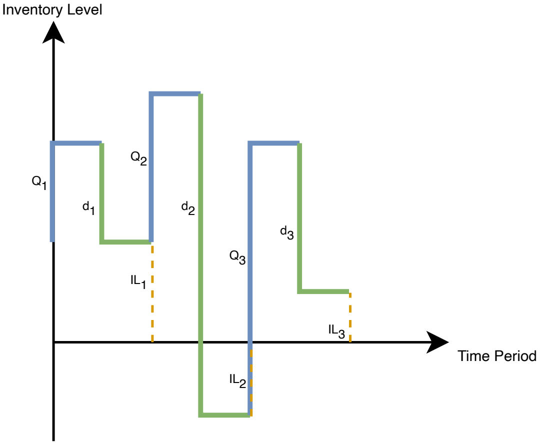
The multi-period inventory problem: Description

- Inventory is non-perishable
- Decide how much to order at each morning
- The demand is uncertain
- Unsatisfied demand is backordered
- What is the optimal order **policy**?

Notations

- D_t: Demand at period t
- h: Holding cost (overage cost)
- p: Stockout cost (shortage cost)
- Q_t: Order quantity at period t
- IL_t: Inventory level at the end of period t
  - $IL_t > 0$: Inventory on hand
  - $IL_t < 0$: Backorder
  - $IL_t = IL_{t-1} + Q_t - D_t$

The multi-period inventory problem



Objective

Cost at period t

- $g_t(Q_t) = h(IL_{t-1} + Q_t - d_t)^+ + p(d_t - IL_{t-1} - Q_t)^+$

Objective

- Total cost: $\sum_{t=1}^{T} g_t(Q_t)$
- Total cost, infinite horizon: $\sum_{t=1}^{\infty} g_t(Q_t)$
- Cumulative discounted cost, infinite horizon: $\sum_{t=1}^{\infty} \gamma^{t-1} g_t(Q_t)$

Problem statement

- Objective: minimize the discounted cumulative cost, $\sum_{t=1}^{\infty} \gamma^{t-1} g_t(Q_t)$
- Optimal inventory decision at period t depends on the ending inventory level in the previous period, IL_{t-1}
- We want to learn a policy π that maps IL_{t-1} to Q_t at each period t.
- The multi-period inventory problem is an associative problem

Markov Decision Process (MDP)

Definition

A Markov Decision Process (MDP) is a tuple (S, A, P, R, γ)

- S: a set of states
- A: a set of actions
- P: transition probability,
  $p(s', r|s, a) = \mathbb{P}[S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a]$,
  $p : S \times R \times S \times A \to [0, 1]$
- R: a reward function, $r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a]$,
  $r : S \times A \to \mathbb{R}$
- γ: a discount factor, $\gamma \in [0, 1]$

Multi-period inventory problem as an MDP

Multi-period inventory problem as an MDP

- $S = \{IL_t \in \mathbb{Z} : IL_t \text{ is a ending inventory levels}\}$
- $A = \{Q_t \in \mathbb{N} : Q_t \text{ is a order quantity}\}$
- $p(s', r|s, a) = \mathbb{P}\left[D_t = s + a - s'\right]$
- R: a reward function,
  $r(s, a) = -\mathbb{E}\left[h(s + a - D_t)^+ + p(D_t - s - a)^+\right]$
- γ: a discount factor, $\gamma \in [0, 1]$

Multi-period inventory problem as an MDP

Returns

- discount return: $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

Policies and value functions

- policy: $\pi(a|s) = \mathbb{P}\left[A_t = a | S_t = s\right]$
- state value function: $v_\pi(s) = \mathbb{E}_\pi\left[G_t | S_t = s\right]$
- action value function: $q_\pi(s, a) = \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right]$

Optimal policies and value functions

Policies

- $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s), \forall s \in S$
- At least one optimal policy exists, denoted as π_*

Value functions

- optimal state value function: $v_*(s) = \max_\pi v_\pi(s), \forall s \in S$
- optimal action value function: $q_*(s, a) = \max_\pi q_\pi(s, a)$, $\forall s \in S, a \in A$

Computing optimal policies and value functions

- Bellman optimality equation
- Linear programming
- Dynamic programming
  - Policy iteration
  - Value iteration

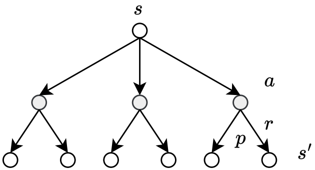
Bellman optimality equation (Optional)

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}\left[G_t | S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}_{\pi_*}\left[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a\right] \\
&= \max_a \sum_{s',r} p(s', r | s, a)\left[r + \gamma v_*(s')\right]
\end{aligned}
$$

Solving MDP: Bellman optimality equation

Bellman optimality equation

$$v_*(s) = \max_a \sum_{s',r} p(s', r | s, a) \left[r + \gamma v_*(s') \right]$$



Solving MDP: Linear programming

- A less frequently used method for solving MDP
- Idea:
  - If $v(s) \geq r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s')$ for all $s \in S$ and $a \in A$, then $v(s)$ is an upper bound on $v_*(s)$
  - $v_*(s)$ must be the smallest such solution

Solving MDP: Linear programming

Linear programming formulation

$$\text{minimize} \sum_{s \in S} \alpha_s v(s)$$

$$\text{s.t. } v(s) \geq r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s'), \forall s \in S, \forall a \in A$$

The constants α_s are arbitrary positive numbers.

Notes

- Linear programming methods can also be used to solve MDPs
- Linear programming methods become impractical at a much smaller number of states than do DP methods (by a factor of about 100).

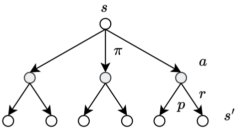
Policy iteration: Policy evaluation

- Given a policy π, compute $v_\pi(s)$ for all $s \in S$
- Linear system of $|S|$ equations in $|S|$ unknowns

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\,[r + \gamma v_\pi(s')]$$

- Iterative policy evaluation (converge to v_π as $k \to \infty$)

$$v_{k+1}(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\,[r + \gamma v_k(s')]$$



Policy iteration: Policy Improvement

- Given $v_\pi(s)$, compute $q_\pi(s, a)$

$$q_\pi(s, a) = \sum_{s', r} p(s', r | s, a) \left[r + \gamma v_\pi(s') \right]$$

- We can improve π by acting greedily

$$q_\pi(s, \pi'(s)) \geq v_\pi(s)$$

- π' is as good as or better than π

Policy Improvement Theorem

Policy Improvement Theorem

If $\forall s \in S$, $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, then π' is as good as or better than π, i.e., $v_{\pi'}(s) \geq v_\pi(s)$.

Greedy policy:

$$\pi'(s) = \arg\max_a q_\pi(s, a)$$

Policy iteration: pseudocode

Algorithm 2: Policy Iteration for estimating $\pi \approx \pi_*$

Initialize $V(s)$ and $\pi(s)$ arbitrarily, for all $s \in \mathcal{S}$;
while `True` **do**
    `// Policy Evaluation`
    while $\Delta \geq \theta$ **do**
        $\Delta \leftarrow 0$;
        for *each* $s \in \mathcal{S}$ **do**
            $v \leftarrow V(s)$;
            $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$;
            $\Delta \leftarrow \max(\Delta, |v - V(s)|)$;
    `// Policy Improvement`
    `policy-stable` $\leftarrow$ `True`;
    foreach $s \in \mathcal{S}$ **do**
        $a \leftarrow \pi(s)$;
        $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$;
        if $a \neq \pi(s)$ **then**
            `policy-stable` $\leftarrow$ `False`;
    if `policy-stable` **then**
        break;

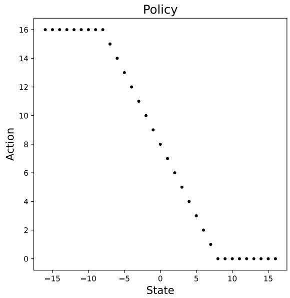
Solving Multi-period inventory problem

We can solve the multi-period inventory problem using policy iteration, value iteration (or Bellman optimality equation, linear programming)

Example:

- State space: $S = \{s \in \mathbb{Z} : -16 \leq s \leq 16\}$
- Action space: $A = \{a \in \mathbb{N} : 0 \leq a \leq 16\}$
- Overage cost: $h = 1$, Underage cost: $p = 10$
- Demand distribution: $D \sim \mathcal{P}(5)$
- Discount factor: $\gamma = 0.9$

Solving Multi-period inventory problem: Example



$$\pi_*(s) = \begin{cases} 0, & 8 \le s \le 16 \\ 8 - s, & -8 \le s < 8 \\ 16, & -16 \le s < -8 \end{cases}$$

We obtained a **Base-stock policy** for $-8 \le s \le 16$, which is exactly the optimal policy for this problem ($h = 1, p = 10, D \sim \mathcal{P}(5)$)

Multi-period inventory problem and MDP

Summary

- The multi-period inventory problem is an associative problem
- We need to find the best action for each state
- The multi-period inventory problem can be formulated as an MDP
- Methods for solving MDPs are introduced

Next

- What if we do not know the dynamics of the environment?
- What if the state space/action space is too large to solve?

TD Learning

- Temporal Difference (TD) Learning do not require complete knowledge of the environment.
- Although a model is required, the model need only generate sample transitions.
- In many cases, it is difficult to obtain the distribution in explicit form.
- TD learning:
  - Prediction problem (estimating v_π or q_π)
  - Control problem (estimating π_*)

TD Prediction

- Recall that the value function is the expected return:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s \right] \end{aligned}$$

- The Simplest TD method: TD(0)

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

- Update $V(S_t)$ towards estimated return $R_{t+1} + \gamma V(S_{t+1})$
- $R_{t+1} + \gamma V(S_{t+1})$ is called TD target
- $\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is called TD error

Q-learning

- Q-learning

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

- Use ϵ-greedy policy to select action

TD Control (Q-Learning): Pseudocode

Algorithm 3: Q-learning for estimating $\pi \approx \pi_*$

Input: a small $\epsilon > 0$, a small $\alpha \in (0, 1]$
Output: output a deterministic policy $\pi \approx \pi_*$

```
Initialize Q(s, a), for all s ∈ S+, a ∈ A(s), arbitrarily except that Q(terminal, ·) = 0;
while True do
    t ← 0;
    Initialize S_t;
    while S_t is not terminal do
        take action A_t using policy derived from Q (e.g., ε-greedy);
        observe R_{t+1} and S_{t+1};
        Q(S_t, A_t) ← Q(S_t, A_t) + α[R_{t+1} + γ max Q(S_{t+1}, a) − Q(S_t, A_t)];
        t ← t + 1;
```

TD Control (Q-Learning): Example

- State space: $S = \{s \in \mathbb{Z} : -16 \leq s \leq 16\}$
- Action space: $A = \{a \in \mathbb{N} : 0 \leq a \leq 16\}$
- Discount factor: $\gamma = 0.9$
- Environment: $h = 1$, $p = 10$, $D \sim \mathcal{P}(5)$

	$a_0 = 0$	$a_1 = 1$	...	$a_{\lvert A \rvert} = 16$
$s_0 = -16$	$Q(s_0, a_0)$	$Q(s_0, a_1)$	...	$Q(s_0, a_{\lvert A \rvert})$
$s_1 = -15$	$Q(s_1, a_0)$	$Q(s_1, a_1)$	...	$Q(s_1, a_{\lvert A \rvert})$
...	...	...	...	...
$s_{\lvert S \rvert} = 16$	$Q(s_{\lvert S \rvert}, a_0)$	$Q(s_{\lvert S \rvert}, a_1)$	...	$Q(s_{\lvert S \rvert}, a_{\lvert A \rvert})$

Table: Q-values for different state-action pairs

TD Control (Q-Learning): Results

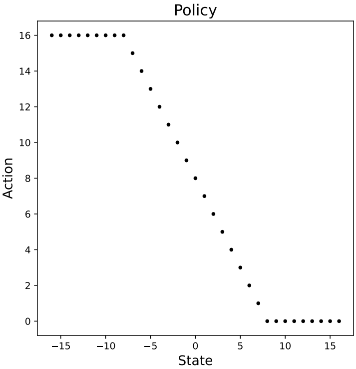


Figure: Value iteration

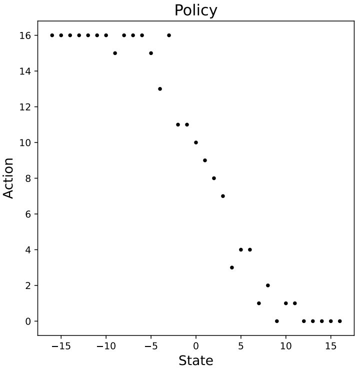


Figure: Q-learning

Multi-period inventory problem and TD learning

Summary

- Q-learning can be used to solve the multi-period inventory problem
- TD learning do not require complete knowledge of the environment
- A model is required to generate sample transitions
- A Q-table is required to store and update Q-values

Next

- The state can be very large or even continuous
- A function approximation method can be used

Deep Q-Network (DQN)

Recall TD error for Q-learning

$$\delta_t = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)$$

Basic idea of DQN

- Represent $Q(s, a)$ by a neural network with weights θ

$$Q(s, a; \theta)$$

- Update θ to minimize the TD error

$$\delta_t = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \theta) - Q(S_t, A_t; \theta)$$

Two issues of DQN

- Correlations between samples
  - Training NN requires independent and identically distributed (i.i.d.) samples
  - But, samples taken from an episode are not i.i.d.
- Non-stationary targets
  - The target $y_j = R_{j+1} + \gamma \max_{a'} Q(S_{j+1}, a'; \theta)$ changes as θ changes

$$\delta_t = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \theta) - Q(S_t, A_t; \theta)$$

Experience replay and target network

- Experience replay
  - Store transitions (S_t, A_t, R_t, S_{t+1}) in a replay buffer D
  - Sample random minibatch of transitions (S_j, A_j, R_j, S_{j+1}) from D
- Target network
  - Use a separate network with weights θ^- to compute the target

$$y_j = \begin{cases} r_j & \text{for terminal } S_{j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(S_{j+1}, a'; \theta^-) & \text{for non-terminal } S_{j+1} \end{cases}$$

  - Update θ^- every C steps

Deep Q-Network (DQN): Pseudocode

Algorithm 4: Deep Q-Network

```
Input: replay buffer capacity N, the number of steps C to perform a target update, a small ε > 0, a small
       α ∈ (0, 1]
Output: output a deterministic policy π ≈ π_*
Initialize empty replay memory D to capacity N;
Initialize action-value function Q with random weights θ;
Initialize target action-value function Q̂ with weights θ⁻ ← θ;
for episode = 1, 2, . . . M do
    t ← 0;
    Initialize S_t;
    while S_t is not terminal do
        with probability ε select a random action A_t;
        otherwise take action A_t using policy derived from Q;
        Execute action A_t and observe reward R_t and S_{t+1};
        Store transition (S_t, A_t, R_t, S_{t+1}) in D;
        t ← t + 1;
        Sample random minibatch of transitions (S_j, A_j, R_j, S_{j+1}) from D;
        Set y_j = { r_j                                  for terminal S_{j+1}
                  { r_j + γ max_{a'} Q̂(S_{j+1}, a'; θ⁻)   for non-terminal S_{j+1}  ;
        Perform a gradient descent step on (y_j − Q(S_j, a_j; θ))^2 with respect to the network parameters
          θ;
        Every C steps reset Q̂ ← Q;
```

DRL for inventory management: Example 1

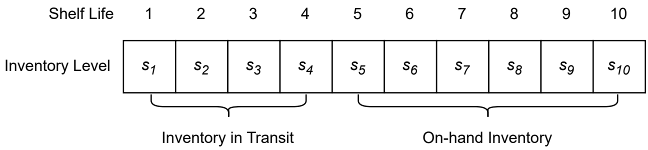


Figure: Perishable Inventory Problem

[0] De Moor, Bram J., Joren Gijsbrechts, and Robert N. Boute. 2022. "Reward Shaping to Improve the Performance of Deep Reinforcement Learning in Perishable Inventory Management." European Journal of Operational Research 301 (2): 535-45.

DRL for inventory management: Summary

Summary

- Machine learning models can be used to approximate the action-value function
- Integrating with machine learning enables solving more complex problems

Future Directions

- Offline Methods
- Explainability
- Privacy-preserving

Table of Contents

Surrogate-based Optimization

Motivation:

- The objective function f is highly nonlinear
- The objective function f only can be obtained by simulation

Basic Idea:

- Approximate the objective function f with a surrogate model $\hat{f}$
- Solve $\mathbf{x} = \arg\min_{\mathbf{x}} \hat{f}(\mathbf{x})$

Note:

- This method only builds surrogate model once
- The accuracy of $\hat{f}$ at the obtained solution is unknown

[0] Gurobi Machine Learning: https://gurobi-machinelearning.readthedocs.io/

Trust Region Methods

Basic Idea:

- Approximate the objective function f with $\hat{f}$ (Normally, a second-order taylor expansion approximation).
- Solve $\mathbf{x} = \arg\min_{\mathbf{x}} \hat{f}(\mathbf{x})$ within a trust region
- Expand or shrink the trust region based on the improvement
- Accept the new solution if the improvement is large enough

Trust Region Methods: Pseudocode

Algorithm 5: Trust Region Methods

Input: initial guess x_0, trust region radius δ, threshold η_1, η_2, scale factor γ_1, γ_2

Output: solution x

```
k ← 0;
solve f(x_k);
while not converged do
    solve min_x f̂(x_k + p) s.t. ||p|| ≤ δ_k;
    compute ρ_k = (f(x_k) − f(x_k + p_k)) / (f(x_k) − f̂(x_k + p_k));
    if ρ_k < η_1 then
        δ_{k+1} ← γ_1 δ_k;
    else
        x_{k+1} ← x_k + p_k;
        if ρ_k > η_2 then
            δ_{k+1} ← γ_2 δ_k;
    k ← k + 1;
```

Bayesian Optimization

Motivation:

- Evaluate f is expensive or time-consuming.
- Only a few evaluations of f are allowed.
- No first- or second-order derivatives.

Basic Idea:

- Approximate f with a surrogate model $\hat{f}$ (Gaussian process)
- Find the next evaluation point x by maximizing the acquisition function $\alpha(x)$
- Update $\hat{f}$ with the new evaluation point x and $f(x)$
- Repeat until the budget is exhausted

Gaussian Process

- A set of points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ is given.
- The corresponding $\mathbf{y} = \{y_1, y_2, \ldots, y_n\}$ are observed.
- Predict y_{n+1} at a new point $\mathbf{x}_{n+1}$.
  - $y_{n+1}|\mathbf{y} \sim \mathcal{N}(\mu(\mathbf{x}_{n+1}), \sigma^2(\mathbf{x}_{n+1}))$
- $\mu(\mathbf{x}_{n+1})$ is the estimated mean of y_{n+1}.
- $\sigma^2(\mathbf{x}_{n+1})$ is the estimated variance of y_{n+1}, which is a measure of uncertainty.

Gaussian Process: Example

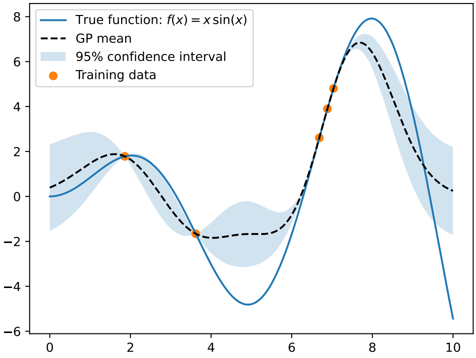


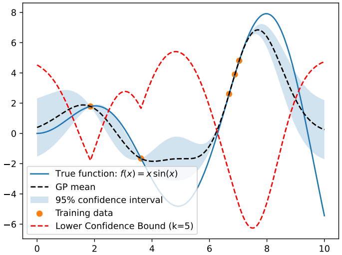
Figure: Gaussian Process Example

Acquisition Function

Find the next evaluation point $\mathbf{x}_{t+1}$ by maximizing $\alpha(\mathbf{x})$.

$$\mathbf{x}_{t+1} = \arg\max_{\mathbf{x}} \alpha(\mathbf{x})$$

- Prediction-based Exploration: $\alpha(\mathbf{x}) = -\mu(\mathbf{x})$
- Error-based Exploration: $\alpha(\mathbf{x}) = \sigma(\mathbf{x})$
- Lower Confidence Bound: $\alpha(\mathbf{x}) = -(\mu(\mathbf{x}) - k\sigma(\mathbf{x}))$
- ...



Bayesian Optimization: Pseudocode

Algorithm 6: Bayesian Optimization

Input: $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, $\mathbf{y} = \{y_1, y_2, \ldots, y_n\}$, acquisition function $\alpha(x)$, budget T

Output: solution x^*

```
while t < T do
    Update the Gaussian process model with X and y;
    x_{t+1} ← arg max_x α(x);
    Evaluate y_{t+1} = f(x_{t+1});
    Update X = X ∪ {x_{t+1}}, y = y ∪ {y_{t+1}};
x* = arg min_x y;
```

Properties of Bayesian Optimization

- $x \in \mathbb{R}^d$, d is not too large. Typically, $d \leq 20$.
- Feasible region $\mathcal{X}$ is a simple set (e.g., a hyper-rectangle).

[0] Frazier, Peter I. "A tutorial on Bayesian optimization." arXiv preprint arXiv:1807.02811 (2018).

Surrogate-assisted Evolutionary Optimization

Motivation:

- The objective function f is expensive ...
- Evolutionary computation needs many evaluations

Basic Idea:

- Approximate the objective function or constraint functions
  - Regression models
- Predict the feasibility, superiority.
  - Classification models
- Use the surrogate models to evaluate candidate solutions
- Only evaluate the ”promising” solutions
- Update the surrogate models with certain frequency

Pseudocode

Algorithm 7: Surrogate-assisted Evolutionary Optimization

Input: $X = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, $\mathbf{y} = y_1, y_2, \ldots, y_n$, surrogate model M, real function f, budget T

while *not terminate* **do**
    Update M using X and $\mathbf{y}$;
    Generate new candidate solutions X';
    Evaluate X' with M;
    Select promising solutions from X';
    Evaluate the selected solutions with the real function f;
    Add the evaluated solutions to X and $\mathbf{y}$;

Surrogate-assisted Optimization for Inventory

Motivation:

- Difficult to explicitly formulate supply chain simulation models.
- Simulation model under uncertainty requires a large number of samples, which is time-consuming.
- Supply chain digital twin can be expensive to evaluate.

Figure: Supply Chain Digital Twin Developed by anyLogistix

[0] Liu, Ziang, and Tatsushi Nishi. 2023. "Data-Driven Evolutionary Computation for Service Constrained Inventory Optimization in Multi-Echelon Supply Chains." Complex & Intelligent Systems, August. https://doi.org/10.1007/s40747-023-01179-0.

Surrogate-assisted Optimization for Inventory

- Multi-period multi-echelon perishable inventory problem
- Predict total cost/service level of an inventory policy with a surrogate model
  - (s, S) policy, (r, Q) policy, (s, Q) policy, ...
  - e.g., $\mathbf{x} = (s_1, S_1, s_2, S_2, \ldots, s_N, S_N)$

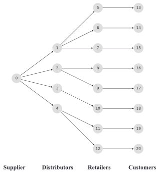


Figure: Perishable inventory problem in a distribution system

Surrogate-assisted Optimization for Inventory

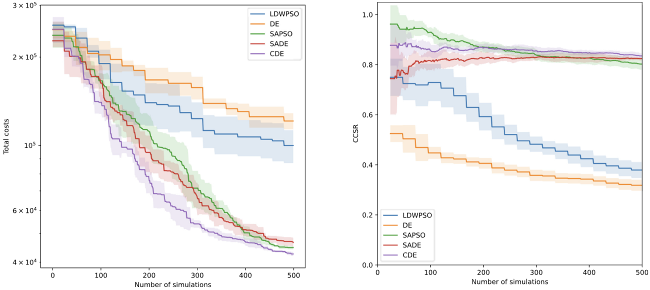


Figure: Convergence graph (Left), correct selection rate (Right)

[0] Liu, Ziang, and Tatsushi Nishi. 2023. “Surrogate-Assisted Evolutionary Optimization for Perishable Inventory Management in Multi-Echelon Distribution Systems.” Expert Systems with Applications, October, 122179.

Summary

Summary:

- Surrogate-assisted optimization is useful when the objective function is expensive.
- Machine learning models can be used to approximate the objective function or constraint functions.
- It is able to obtain a good solution with a small number of evaluations.

Table of Contents

Policy Improvement Theorem (Optional)

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s\right] \\
&\dots \\
&= v_{\pi'}(s)
\end{aligned}
$$

Policy iteration: Policy Improvement

- Suppose the new policy π' is as good as, but not better than, the old policy π
- Then, $v_{\pi'}(s) = v_{\pi}(s)$, for all $s \in S$

$$v_{\pi'}(s) = \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v_{\pi'}(s')\right]$$

- This is the Bellman optimality equation
- π' must be an optimal policy

Value Function ∞-Norm (Optional)

- We use ∞-norm to measure the difference between two value functions u and v
- ∞-norm is the maximum absolute difference between state values

$$\|u - v\|_\infty = \max_{s \in S} |u(s) - v(s)|$$

Bellman Expectation Backup is a Contraction (Optional)

- Define the Bellman operator T_π

$$T_\pi(v) = \mathcal{R}_\pi + \gamma \mathcal{P}_\pi v$$

- This operator is γ-Contraction, i.e., it makes values functions closer by at least γ

$$\begin{aligned} \| T_\pi(u) - T_\pi(v) \|_\infty &= \| \mathcal{R}_\pi + \gamma \mathcal{P}_\pi u - \mathcal{R}_\pi - \gamma \mathcal{P}_\pi v \|_\infty \\ &= \gamma \| \mathcal{P}_\pi u - \mathcal{P}_\pi v \|_\infty \\ &\leq \gamma \| u - v \|_\infty \end{aligned}$$

Value iteration

- Must we wait for policy evaluation to converge before improving the policy?
- Policy improvement after one update of each state

$$V_{k+1}(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')]$$

- For arbitrary v_0, v_k converges to v_*
- The output policy can be obtained

$$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_*(s')]$$

Value iteration: pseudocode

Algorithm 8: Value Iteration for estimating $\pi \approx \pi_*$

Input: input a small threshold $\theta > 0$ determining accuracy of estimation

Output: output a deterministic policy $\pi \approx \pi_*$, such that $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

```
Initialize V(s) arbitrarily, for all s ∈ S+;
Initialize V(terminal) = 0;
while Δ ≥ θ do
    Δ ← 0;
    foreach s ∈ S do
        v ← V(s);
        V(s) ← max_a Σ_{s',r} p(s', r|s, a)[r + γV(s')];
        Δ ← max(Δ, |v − V(s)|);
```

Gaussian Process (Optional)

$$\mu(\mathbf{x}) = \mathbf{K}(\mathbf{x}, X)\mathbf{K}(X, X)^{-1}(\mathbf{y} - m(X)) + m(\mathbf{x})$$
$$\sigma^2(\mathbf{x}_{n+1}) = \mathbf{K}(\mathbf{x}_{n+1}, \mathbf{x}_{n+1}) - \mathbf{K}(\mathbf{x}_{n+1}, X)\mathbf{K}(X, X)^{-1}\mathbf{K}(X, \mathbf{x}_{n+1})$$

Mean function and Kernel:

- $m(\mathbf{x})$: Mean function
- $\mathbf{K}(\mathbf{x}, \mathbf{x}')$: Kernel function

Commonly used functions:

- Zero mean function: $m(\mathbf{x}) = 0$
- Gaussian kernel: $\alpha \exp(-\|\mathbf{x} - \mathbf{x}'\|^2)$
- ...