# A Short Tutorial on Stochastic Dynamic Programming and Reinforcement Learning

Nan Chen

Systems Engineering & Engineering Management
The Chinese University of Hong Kong

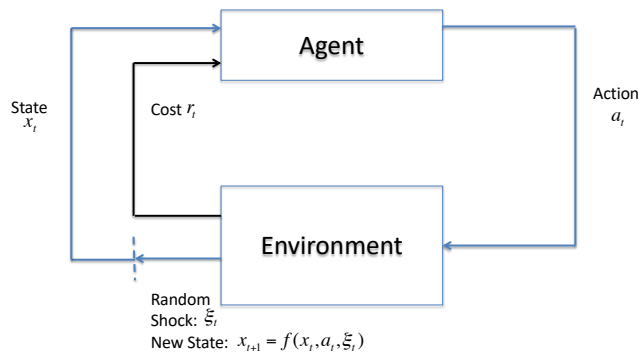
Institute for Mathematical Sciences, National University of Singapore
July 31 & August 1, 2019

# Outline

# Outline

# Mathematical Formulation

- Stochastic dynamic programming (SDP) provides a powerful framework for modeling and solving decision-making problems under a random environment where uncertainty is resolved and actions are taken sequentially over time.
- It also has become increasingly important to help us understand the general principle of reinforcement learning, a rapidly developing area of artificial intelligence.
- The agent-environment interaction:



# Mathematical Formulation (Continued)

- Time horizon: $t = 0, 1, ...$
  - A finite time horizon is possible and more relevant in financial applications
- State space: $\mathcal{S} = \{1, 2, \cdots, S\}$
- Action space: $\mathcal{A}(s)$, $s \in \mathcal{S}$
- System evolution:
  - Use $\{x_t : t = 0, 1, \cdots\}$ to denote the state evolution of the system
  - Transitional probability matrix is known: $P(x_{t+1}|x_t, a_t)$.
  - For simplicity, we consider a Markovian case in which there is only one-step dependence in the state sequence $\{x_t\}$. Sometimes people refer to this special case as Markov decision processes (MDP)
- Reward: $r(s, a)$

# Mathematical Formulation (Continued)

- Let $\mathcal{F}_t$ represent the $\sigma$-algebra spanned by the history of the system state up to time $t$, i.e., $\mathcal{F}_t = \sigma(x_0, x_1, \cdots, x_t)$, $\forall t$.
- Policy: a mapping from past observation of the system state to the set of actions, i.e., $\pi_t$ is an $\mathcal{F}_t-$measurable function valued in $\mathcal{A}$.
- Objective: Find a policy $\pi = (\pi_t, t = 0, 1, \cdots)$ to maximize

$$V(x) = \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t r(x_t, \pi_t) \middle| x_0 = x\right],$$

where $\gamma \in (0, 1)$ is a discount factor used by the decision maker.

# The Principle of Optimality: Bellman Equation

- We can easily show that the optimal value of the problem in the last slide is characterized by the following Bellman equation:
  - $$V(x) = \max_{a \in \mathcal{A}} \mathbb{E}\left[ r(x_t, a_t) + \gamma V(x_{t+1}) \middle| x_t = x \right]. \qquad (1)$$
  - The optimal policy $\pi_t^* = a_t^*(x)$ maximizes the right hand side of (1).
  - Intuitively, the term $V(x_{t+1})$ encodes the long-term impact when the agent takes actions to steer the system into a new state $x_{t+1}$.
  - Stationary policy

# Illustrative Examples: American Options

- American options entitle the holder the right to buy/sell a certain amount of assets for a pre-determined price by a time point in the future.
- American put
  - Time to maturity $T$
  - Underlying asset price follows a Markovian process living in a state space $\mathcal{S} = \{p_1, p_2, \cdots, p_S\}$
  - Strike price $K$
  - Value:

$$V_0(p) = \max_{\tau \in \{0,1,\cdots,T\}} \mathbb{E}\left[\gamma^\tau (K - x_\tau)^+ \middle| x_0 = p\right],$$

where $\tau$ is a stopping time.

# Illustrative Examples: American Options (Continued)

- We can easily see that the aforementioned general formulation applies to this problem.
  - Action space: $\mathcal{A} = \{0, 1\}$ where 0 represents "stop and exercise" and 1 "continue"
  - Let $\delta$ be an absorbing state and enlarge the state space to $\tilde{\mathcal{S}} = \mathcal{S} \cup \{\delta\}$.
  - The state variable evolves according to

$$P(\tilde{x}_{t+1}|\tilde{x}_t, a) = \begin{cases} P(x_{t+1}|x_t), & a = 1 \text{ and } \tilde{x}_t, \tilde{x}_{t+1} \in \mathcal{S}; \\ 1, & a = 0 \text{ and } \tilde{x}_{t+1} = \delta; \\ 0, & a = 0 \text{ and } x_{t+1} \in \mathcal{S}; \\ 1, & \tilde{x}_t = \delta. \end{cases}$$

  - Reward:

$$r(\tilde{x}_t, a) = \begin{cases} 0, & a = 1; \\ (K - \tilde{x}_t)^+, & a = 0 \text{ and } \tilde{x}_t \in \mathcal{S}; \\ 0, & \tilde{x}_t = \delta. \end{cases}$$

# Illustrative Examples: American Options (Continued)

- The objective:

$$V_t(x) = \max_{\pi \in \mathcal{A}} \mathbb{E}\left[\sum_{s=t}^{T} \gamma^t r(\tilde{x}_s, \pi_s) \Big| \tilde{x}_t = x\right].$$

- The Bellman equation:

$$V_t(x) = \max\left\{(K-x)^+, \quad \gamma \mathbb{E}[V_{t+1}(\tilde{x}_{t+1}) | \tilde{x}_t = x]\right\}.$$

# Illustrative Examples: Investment and Consumption

- Time horizon: $t = 0, 1, ..., T$
- Universe of investment: one risk free asset offering interest $r$ and $N$ stocks
- Stock processes: $\{S_t^1, \cdots, S_t^N\}$, $t = 0, 1, \cdots, T$, $N$-dim Markovian process
- In each period, the agent needs to determine
  - Consumption amount $c_t$
  - Investment portfolio: $\$y_t$ in risk free asset and $\{x_t^1, \cdots, x_t^N\}$ shares in the stocks
- These actions should satisfy

$$W_t = y_t + \sum_{n=1}^{N} x_t^n S_t^n - c_t$$

and some other constraints: no borrowing, no shorting, trading constraints, and so on.

# Illustrative Examples: Investment and Consumption (Continued)

- Objective: the agent intends to determine the investment-consumption plan $\pi_t = (c_t; y_t, x_t^1, \cdots, x_t^N)$, $t = 0, \cdots, T$ such that

$$\max_{\pi \in \mathcal{A}(W)} \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t U_t(c_t) + \gamma^T U_T(W_T) \middle| x_0 = x\right]$$

- Wealth process:

$$W_t = (1+r)y_{t-1} + \sum_{n=1}^{N} x_{t-1}^n S_t^n.$$

# Illustrative Examples: Investment and Consumption (Continued)

- The Bellman equation in this case should be

$$\begin{aligned} & V_t(W;S) \\ = & \max_{\pi} \left\{ U_t(c_t) + \gamma \mathbb{E}\left[ V_t(W_{t+1}, S_{t+1}) \middle| W_t = W; S_t = S \right] \right\}. \end{aligned}$$

# Outline

# Value Iteration

- Value iteration is perhaps the most widely used algorithm in MDP. It is based on the above principle of optimality.
- Recall that we actually need to solve a fixed-point problem to determine $V$ in an infinite-time horizon MDP. Then, the following recursive method is natural:
  - Step 0. Initialize $v^0$ and select a tolerance parameter $\epsilon$
  - Step 1. For each $x \in \mathcal{S}$, compute

$$\begin{aligned} v^n(x) &= \max_{a \in \mathcal{A}} \mathbb{E}\left[ r(x_t, a) + \gamma v^{n-1}(x_{t+1}) \middle| x_t = x \right] \\ &= \max_{a \in \mathcal{A}} \left[ r(x, a) + \gamma \sum_{x'} P(x'|x, a) v^{n-1}(x') \right] \end{aligned}$$

  - Step 2. If $\|v^n - v^{n-1}\| \leq \epsilon$, let $a^*$ be the resulting policy that solves the above equality and output $v^n$ as an approximation to $V$; Otherwise, set $n = n + 1$ and go to Step 1.

# Policy Iteration

- In policy iteration we start with a policy and then find its corresponding value. This value is used to help us to find another policy.
- Consider any stationary policy $\pi$. To evaluate this policy, we have

$$
\begin{aligned}
v^{\pi}(x) &= \mathbb{E}\left[\sum_{t=0}^{+\infty} \gamma^t r(x_t, \pi_t) \middle| x_0 = x\right] \\
&= \mathbb{E}\left[r(x, \pi(x)) + \sum_{t=1}^{+\infty} \gamma^t r(x_t, \pi(x_t)) \middle| x_0 = x\right] \text{ (stationarity)} \\
&= r(x, \pi(x)) + \gamma \mathbb{E}\left[\sum_{t=1}^{+\infty} \gamma^{t-1} r(x_t, \pi(x_t)) \middle| x_0 = x\right] \\
&= r(x, \pi(x)) + \gamma \sum_{x'} v^{\pi}(x') P(x'|x, \pi(x)) \qquad (2)
\end{aligned}
$$

# Policy Iteration (Continued)

- Notice that we can solve for $v^{\pi}$ using some numerical routines for linear equation systems.
- Policy iteration:
  - Step 0. Start with a policy $\pi^0$ and set $n = 1$
  - Step 1. Use (2) to compute $v^{\pi^{n-1}}$
  - Step 2. Update policy by

$$\pi^n(x) = \arg\max_{a \in \mathcal{A}} \mathbb{E}\left[ r(x_t, a) + \gamma v^{\pi^{n-1}}(x_{t+1}) \middle| x_t = x \right].$$

  - Step 2. If $\pi^n(x) = \pi^{n-1}(x)$ for all states $x$, then stop and output $\pi^* = \pi^n$; Otherwise, set $n = n + 1$ and go to Step 1.

# Policy Iteration (Continued)

- There are two ways to determine $v^\pi$ from (2)
  - via matrix inversion

$$v^\pi = (I - \gamma P^\pi)^{-1} r^\pi$$

  - via iteration, i.e., repeapting

$$v^{\pi,n} = r^\pi + \gamma P^\pi v^{\pi,n-1}.$$

# Outline

# Three Curses of Dimensionality

- Both value iteration and policy iteration are not computationally tractable for large scaled problems because of the "curse of dimensionality".
- Take the previous investment/consumption example.
  - State space: $(N+1)$-dim
  - Action space: $(N+2)$-dim
  - Transition matrix: $(N+1) \times (N+1)$
- The number of the combination of state, action, and stochastic transition that both iteration methods need to visit grows exponentially as the dimensionality of the problem increases.

# The Basic Idea of ADP

- The ADP algorithm steps forward in time to update the estimates of value function iteratively.
- Main Steps:
  - Step 0. Initialize $V^0(x)$ for all $x$; choose an initial state $x_0$; set $n = 1$;
  - Step 1. For $t = 0, 1, 2, \cdots, T$ do:
    - Step 1a. At $x_t^n$, solve

$$u_t^n = \max_{a \in \mathcal{A}} \left[ r(x_t, a_t) + \gamma \sum_{x'} V^{n-1}(x') P(x' | x_t^n, a) \right]. \qquad (3)$$

      and let $a_t^n$ be the value of $a$ that solves the above maximization problem.
    - Step 1b. Update

$$V^n(x) = \begin{cases} (1 - \alpha_{n-1}) V^{n-1}(x) + \alpha_{n-1} u_t^n, & \text{for } x = x_t^n \\ V^{n-1}(x), & \text{otherwise} \end{cases}$$

    - Step 1c. Simulate $x_{t+1}^n$ from the distribution $P(\cdot | x_t^n, a_t^n)$
  - Step 2. Let $n = n + 1$. If $n < N$, go to Step 1.

# Key Issues of ADP

- The algorithm is known as the "forward pass" algorithm, where we step forward in time, updating value functions as we progress. Another variation involves simulating forward through the horizon without updating the value function. Then, after the simulation is done, we step backward through time to update the value function, using information about the entire future trajectory.

# Key Issues of ADP (Continued)

- There are several issues we have to address to develop an effective ADP strategy:
  1. In the above ADP algorithm, we only update the value of states we visit, but we need the value of all the states to obtain $V^n$.
  2. We assume that we can compute the expectation, which is often not the case.
  3. We need to determine how to update the approximation.
  4. We can easily get caught in a circle of choosing actions that take us to states that we have already visited, simply because we may have pessimistic estimates of states that we have not yet visited. We need some mechanism to force us to visit states just to learn the values of those states.

# Approximate Value Function

- To address the first issue, we need to discuss some techniques of value approximation:
  - Regression and basis functions
  - Other nonparametric methods
    - $k$-nearest neighbors
    - Kernel regression
    - Local polynomial regression
    - Neural networks

# Approximate Value Function: Regression

- The regression method approximates functions as a linear combination of some pre-selected basis functions.
- Consider a set of basis functions $\{\phi_1, \cdots, \phi_m\}$ and suppose that we have observed the values (with noise) of $V$ at states $x_1, \cdots, x_n$, denoted by $v(x_1), \cdots, v(x_n)$.
- We can use the following least square method to build up an optimal approximation from the basis functions:

$$\min_{\theta} \sum_{i=1}^{n} \left( v(x_i) - \sum_{j=1}^{m} \theta_j \phi_j(x_i) \right)^2 .$$

- Tremendous possibilities of variations from here: robust, LASSO, logistic, ...

# Approximate Value Function: $k$-Nearest Neighbors

- How to select suitable basis functions in a given case still remains a (frustrating) art. Nonparametric alternatives more depend on observed data to build local approximations and can help avoid the art of specification of parametric models. The regression method approximates functions as a linear combination of some pre-selected basis functions.
- The $k$-nearest neighbor regression may be the simplest form of nonparametric regressions. It estimates functions by using a weighted average of their values at the $k$-nearest neighbors:

$$V(x) = \frac{1}{k} \sum_{i \in \mathcal{N}^k(x)} v(x_i).$$

  where $\mathcal{N}^k(x)$ contains $k$ indices of the nearest neighbors of $x$ among $\{x_1, \cdots, x_n\}$.
- metric, determination of $k$, weights

# Approximate Value Function: Kernel Regression

- Using the $k$-nearest neighbor model, the estimate can change abruptly as $x$ changes continuously, as the set of nearest neighbors changes.
- Kernel regression forms an estimate by using a weighted sum of observations

$$V(x) = \sum_{i=1}^{m} \frac{k_h(x, x_i)}{k_h(x)} v(x_i).$$

where $k_h(x, x^i)$ is a weightimg function that declines with the distance between $x$ and $x_i$ and

$$k_h(x) = \sum_i k_h(x, x_i).$$

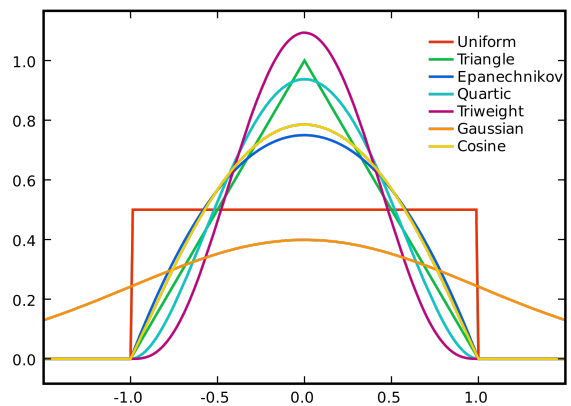
# Approximate Value Function: Kernel Regression (Continued)

- One of the most popular choice of the kernel function $k_h(x, x^i)$ is the following Gaussian kernel

$$k_h(x, x_i) = \exp(-\|x - x_i\|/h)^2$$

  Here $h$ plays the role of the bandwidth.

- Other popular choices of kernels:




- Choice of $h$, curse of dimensionality comes back.

# Approximate Value Function: Local Polynomial Regression

- Local polynomial regression estimates regression models locally around each observation $x_i$ by solving a least square problem that minimizes a weighted sum of least squares.
- That is, at each $x$, we solve

$$\min_{\theta} \left( \sum_{i=1}^{n} K_h(x, x_i) \cdot \left( v(x_i) - \sum_{j=1}^{m} \theta_j \phi_j(x_i) \right)^2 \right).$$

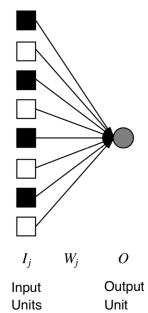
As the result of minimization, the optimal $\theta$ depends on $x$.

# Approximate Value Function: Neural Network

- Neural networks represent an unusually powerful and general class of approximation tools that have been widely used in machine learning.
- The simplest neural network is nothing more than a linear regression model. Consider a case that we have observed $(X_1, \cdots, X_l)$ and $Y$. Recall that if we want to fit a linear model from these data, we need to solve the following least square error:

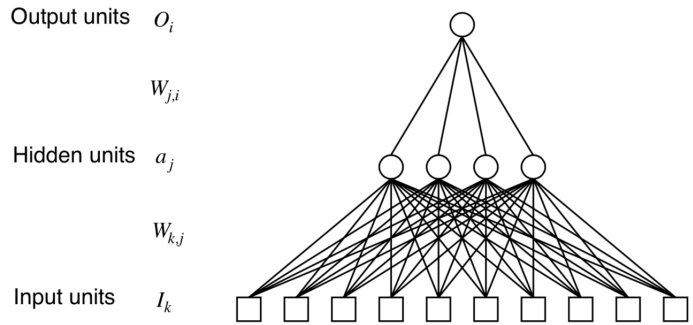
$$\min_w \sum_k (Y^k - (w_1 X_1^k + \cdots + w_l X_l^k))^2.$$

- Representing linear regression models as a single-layer feed-forward neural network.



# Approximate Value Function: Neural Network (Continued)

- We may extend the single layer neural network to multiple layered network.




  - Input layer: $(X_1, \cdots, X_I)$
  - Hidden layer:

$$a_j = \phi\left(\sum_{k=1} w_{k,j} X_k + b_j\right)$$

    for some activation function $\phi$.
  - Output layer: $o = \sum_j v_j a_j$

- Popular activation functions include tanh, arctan, rectified linear unit (ReLu), logistic (sigmoid).

# Approximate Value Function: Neural Network (Continued)

- Universal approximation theorem (Cybenko 1989, Hornik 1991)
  - Let activation function $\phi$ be non-constant, bounded, and continuous function. Let $I_m$ denote the m-dimensional unit hypercube $[0,1]^m$ and $C(I_m)$ denote the space of real-valued continuous functions on $I_m$. Then, the set of functions constructed from the two-layer neural network is dense in $C(I_m)$.
- One popular way to train the network is through the gradient descent method.

# Example: American Option Pricing

- We may adopt the variation of the ADP to approximately solve this optimal stopping problem. The method is based on the regression expansion (Longstaff and Schwartz 2001, Tsitsiklis and Van Roy 2001).
  - Forward to simulate states
  - Backward to update the value

# Example: American Option Pricing (Continued)

- More precisely,
  - Step 1. Simulate $b$ independent sample paths $\{x_{1j}, \cdots, x_{mj}\}$, $j = 1, 2, \cdots, b$, of the underlying asset price;
  - Step 2. At terminal nodes, set $V_{mj} = (K - x_{mj})^+$
  - Step 3. Apply backward induction to update value: for $i = m-1, \cdots, 1$
    - Step 3a. Use the regression method to estimate

$$C_i(x) = \mathbb{E}[\gamma V_{i+1} | x_i = x]$$

    - Step 3b. Set

$$V_{ij} = \max\{(K - x_{ij})^+, C_i(x_{ij})\}$$

    for all $j = 1, 2, \cdots, b$.
  - Step 4. Set

$$V_0 = (V_{11} + \cdots + V_{1b})/b.$$

# Example: American Option Pricing (Continued)

- American put, 4 periods, strike price \$1.10, discount factor 0.9417, the current price of the underlying \$1.00.
- Step 1: simulate sample paths of the underlying stock

Stock price paths

| Path | $t = 0$ | $t = 1$ | $t = 2$ | $t = 3$ |
|---|---|---|---|---|
| 1 | 1.00 | 1.09 | 1.08 | 1.34 |
| 2 | 1.00 | 1.16 | 1.26 | 1.54 |
| 3 | 1.00 | 1.22 | 1.07 | 1.03 |
| 4 | 1.00 | .93 | .97 | .92 |
| 5 | 1.00 | 1.11 | 1.56 | 1.52 |
| 6 | 1.00 | .76 | .77 | .90 |
| 7 | 1.00 | .92 | .84 | 1.01 |
| 8 | 1.00 | .88 | 1.22 | 1.34 |

- Step 2: determine ultimate payoff

Cash-flow matrix at time 3

| Path | $t = 1$ | $t = 2$ | $t = 3$ |
|---|---|---|---|
| 1 | — | — | .00 |
| 2 | — | — | .00 |
| 3 | — | — | .07 |
| 4 | — | — | .18 |
| 5 | — | — | .00 |
| 6 | — | — | .20 |
| 7 | — | — | .09 |
| 8 | — | — | .00 |

# Example: American Option Pricing (Continued)

- Step 3: move backwardly to time 2

Regression at time 2

| Path | $Y$ | $X$ |
|---|---|---|
| 1 | $.00 \times .94176$ | 1.08 |
| 2 | — | — |
| 3 | $.07 \times .94176$ | 1.07 |
| 4 | $.18 \times .94176$ | .97 |
| 5 | — | — |
| 6 | $.20 \times .94176$ | .77 |
| 7 | $.09 \times .94176$ | .84 |
| 8 | — | — |

Using $\{1, X, X^2\}$ as the basis functions to fit $E[Y|X]$, we have

$$c_2(x) = -1.070 + 2.983x - 1.813x^2.$$

- Step 4: determine the option value at time 2

Optimal early exercise decision at time 2

| Path | Exercise | Continuation |
|---|---|---|
| 1 | .02 | .0369 |
| 2 | — | — |
| 3 | .03 | .0461 |
| 4 | .13 | .1176 |
| 5 | — | — |
| 6 | .33 | .1520 |
| 7 | .26 | .1565 |
| 8 | — | — |

- Repeat the above steps until time 1.

# Example: Extension to a General Control Problem

- Bachouch et al. (2019) and Huŕe et al. (2019) suggest a deep neural network based approach to solve a general high-dimensional control problem
  - Objective: find the optimal control policy to maximize

$$\max_{\pi \in \mathcal{A}(W)} \mathbb{E}\left[\sum_{t=0}^{T-1} f_t(x_t, \pi_t) + g_T(x_T) \Big| x_0 = x\right]$$

with the underlying state process:

$$x_{n+1} = F(x_t, \pi_t, \xi_t)$$

  - The Bellman equation:

$$V_T(x) = g_T(x)$$

and

$$\begin{aligned} V_t(x) &= \max_{a_t} \mathbb{E}\left[f_t(x_t, a_t) + V_{t+1}(x_{t+1}) \Big| x_t = x\right] \\ &= \max_{a_t} \mathbb{E}\left[f_t(x, a_t) + V_{t+1}(F(x, a_t, \xi_t))\right] \end{aligned}$$

# Example: Extension to a General Control Problem (Continued)

- Main idea: Use two deep neural networks to approximate the policy and value function respectively.
- Algorithm:
  - Use training distributions $\mu_n$ to sample $K$ states $\{x_t^k : k = 1, \cdots, K\}$ at time $t = 1, \cdots, T$.
  - Set $\hat{V}_T(x) = g_T(x)$.
  - For $t = T-1, \cdots, 1$ do:
    - Let $\Pi_t(x_t; \beta_t)$ represent the neural network to approximate the optimal policy at time $t$. Train $\beta_t$, the parameters of the network.

$$\hat{\beta}_t = \arg\max_{\beta_t} \mathbb{E}\left[f_t(x, \Pi_t(x; \beta_t)) + \hat{V}_{t+1}(F(x, \Pi_t(x; \beta_t), \xi_t))\right]$$

    - Let $\hat{V}_t(x_t; \alpha_t)$ represent the neural network to approximate the value function at time $t$. Train $\alpha_t$ such that

$$\begin{aligned} & \hat{\alpha}_t \\ = \quad & \arg\min_{\alpha_t} \mathbb{E}\left[\left(f_t(x, \Pi_t(x; \hat{\beta}_t)) + \hat{V}_{t+1}(F(x, \Pi_t(x; \hat{\beta}_t), \xi_t)) - \hat{V}_t(x_t; \alpha_t)\right)^2\right] \end{aligned}$$

# Stochastic Optimization and Gradient Search

- On many occasions we discussed before, we have to solve the following optimization problem:

$$\min_{\theta} \mathbb{E}[f(\theta, W)]$$

where $W$ is a random quantity and the expectation is taken with respect to $W$.

- It is not computationally feasible in a lot of applications of practical interest to obtain the explicit expression of $\mathbb{E}[f(\theta, W)]$ (cf. the second issue about ADP). There are two major classes of approaches to solve the optimization problem.

  - Sample average approximation: draw independent samples of $W$ and estimate $\mathbb{E}[f(\theta, W)]$ by

$$\frac{1}{n}\sum_{i=1}^{n} f(\theta, W_i)$$

    Solve the above for the optimal $\theta^*$ using some deterministic optimization routines.
  - Stochastic approximation

# Stochastic Optimization and Gradient Search (Continued)

- Stochastic approximation finds local optima of the optimization problem for an unconstrained decision set using its noisy sub-gradient observations.
  - Batch gradient descent: update

$$\theta_{k+1} = \theta_k - \alpha_k \cdot \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta f(\theta_k, W_i)$$

    where $\alpha_k$ is the step size at step $k$.
  - Stochastic gradient descent:

$$\theta_{k+1} = \theta_k - \alpha_k \nabla_\theta f(\theta_k, W_k)$$

    where $W_k$ is a random copy of $W$ drawn from its distribution at step $k$.
  - Mini-batch gradient:

$$\theta_{k+1} = \theta_k - \alpha_k \cdot \frac{1}{n} \sum_{i=1}^{b} \nabla_\theta f(\theta_k, W_k^i)$$

    where a small amount of samples of $W$ is drawn to compute the gradient.

# Recursive Least Square

- We discussed how to approximate value functions using a batch of data. However, in the setting of ADP, it would be very expensive to use batch methods to estimate. To address the third issue of ADP, we develop a recursive least square to update the coefficient from one more observation.
- Suppose that we want to fit the following linear model

$$y = \theta_1 x_1 + \cdots + \theta_p x_p + \epsilon$$

We have observed $n-1$ pairs of $(y, \{x_1, \cdots, x_p\})$:

$$\mathbf{y}_{n-1} = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_{n-1} \end{pmatrix}$$

and

$$\mathbf{X}_{n-1} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \cdots & \cdots & \ddots & \cdots \\ x_{(n-1),1} & x_{(n-1),2} & \cdots & x_{(n-1),p} \end{pmatrix}$$

# Recursive Least Square (Continued)

- The least square estimate of $\theta$ should be given by

$$\hat{\theta}_{n-1} = (\mathbf{X}_{n-1}^T \mathbf{X}_{n-1})^{-1} \mathbf{X}_{n-1}^T \mathbf{y}_{n-1}.$$

- Now let's assume that we have one more observation of $(y, \{x_1, \cdots, x_p\})$

$$\mathbf{y}_n = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_{n-1} \\ y_n \end{pmatrix} =: \begin{pmatrix} \mathbf{y}_{n-1} \\ y_n \end{pmatrix}$$

and

$$\mathbf{X}_n = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \cdots & \cdots & \ddots & \cdots \\ x_{(n-1),1} & x_{(n-1),2} & \cdots & x_{(n-1),p} \\ x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{pmatrix} =: \begin{pmatrix} \mathbf{X}_{n-1} \\ x_n \end{pmatrix}$$

# Recursive Least Square (Continued)

- The new estimate with this new observation should be updated to

$$\begin{aligned}\hat{\theta}_n &= (\mathbf{X}_n^T\mathbf{X}_n)^{-1}\mathbf{X}_n^T\mathbf{y}_n \\ &= (\mathbf{X}_{n-1}^T\mathbf{X}_{n-1} + x_n^T x_n)^{-1}(\mathbf{X}_{n-1}^T\mathbf{y}_{n-1} + x_n^T y_n)\end{aligned}$$

- By Sherman-Morrison formula,

$$\begin{aligned}&(\mathbf{X}_{n-1}^T\mathbf{X}_{n-1} + x_n^T x_n)^{-1} \\ = \;& (\mathbf{X}_{n-1}^T\mathbf{X}_{n-1})^{-1} - \frac{1}{\gamma_n}\left[(\mathbf{X}_{n-1}^T\mathbf{X}_{n-1})^{-1}(x_n^T x_n)(\mathbf{X}_{n-1}^T\mathbf{X}_{n-1})^{-1}\right]\end{aligned}$$

  where

$$\gamma_n = 1 + x_n(\mathbf{X}_{n-1}^T\mathbf{X}_{n-1})^{-1}x_n^T.$$

- Some algebra leads to

$$\hat{\theta}_n = \hat{\theta}_{n-1} - \frac{1}{\gamma_n}(\mathbf{X}_{n-1}^T\mathbf{X}_{n-1})^{-1}x_n^T(x_n\hat{\theta}_{n-1} - y_n).$$

# Outline

# Motivation

- We use American option as an example to motivate the discussion on SDP duality. For notational simplicity, let's assume the discount factor $\gamma = 1$.
- Recall that the exercising policy used by the option holder must be a stopping time; that is, the decision she makes should depend on what has happened in the history. If we relax this requirement, allowing her to make the decision after she observed the entire trajectory of the underlying asset price, then

$$\begin{aligned} V_0(p) &= \max_{\tau \in \{0,1,\cdots,T\}} \mathbb{E}\left[(K - x_\tau)^+ \middle| x_0 = p\right] \\ &\leq \mathbb{E}\left[\max_{0 \leq t \leq T} (K - x_t)^+ \middle| x_0 = p\right] \end{aligned}$$

# Motivation (Continued)

- One analogy

$$\begin{array}{lll} \text{Primal}: & \min\ c^T x & \text{Dual}: \\ & \text{s.t.}\ \ Ax \le b\ (\lambda) \quad \Leftrightarrow & \max_{\lambda \ge 0} \min_x c^T x + \lambda^T (b - Ax) \end{array}$$

- Key idea of duality in solving constrained optimization problems:
  - Introduce a penalty $\lambda$ to relax the constraint
  - Formulate a saddle-point duality
  - Weak and strong duality
- Can we construct a proper penalty such that

$$V_0(p) = \inf_{M} \mathbb{E}\left[\max_{0 \le t \le T} \left\{(K - x_t)^+ - M_t\right\} \middle| x_0 = p\right]?$$

# Dual Theory for Optimal Stopping

- Rogers (2002) and Haugh and Kogan (2004):
  - Let
$$\mathcal{M} = \{M : M \text{ is a martingale and } M_0 = 0\}.$$
Then,
$$V_0(p) = \inf_{M \in \mathcal{M}} \mathbb{E}\left[\max_{0 \le t \le T} \left\{(K - x_t)^+ - M_t\right\} \middle| x_0 = p\right] \tag{4}$$
- Proof
  - For any martingale $M \in \mathcal{M}$ and stopping time $\tau$,
$$\begin{aligned} & \mathbb{E}\left[\max_{0 \le t \le T} \left\{(K - x_t)^+ - M_t\right\} \middle| x_0 = p\right] \\ \geq \quad & \mathbb{E}\left[\left\{(K - x_\tau)^+ - M_\tau\right\} \middle| x_0 = p\right] \\ = \quad & \mathbb{E}\left[(K - x_\tau)^+\right] .(\text{Optional sampling theorem}) \end{aligned}$$

# Dual Theory for Optimal Stopping (Continued)

- Proof (Continued)
  - Let $V_t$ represent the option value at time $t$ and define

$$\Delta_t = V_t(X_t) - \mathbb{E}[V_t(X_t)|X_{t-1}].$$

    Set

$$M_t = \sum_{i=1}^{t} \Delta_t.$$

    It is easy to check that $M_0 = 0$ and $M$ is a martingale.
  - Use induction to show that

$$V_t(X_t) = \max_{t \le i \le T} \left\{ (K - x_i)^+ - (M_i - M_t) \right\}.$$

# Strong and Weak Duality

- The relationship (4) implies that
  - weak duality: for any martingale $M$,

$$V_0(p) \leq \mathbb{E}\left[\max_{0\leq t\leq T}\left\{(K-x_t)^+ - M_t\right\}\middle| x_0 = p\right]$$

  - strong duality: there exists a martingale such that

$$V_0(p) = \mathbb{E}\left[\max_{0\leq t\leq T}\left\{(K-x_t)^+ - M_t^*\right\}\middle| x_0 = p\right]$$

# Martingale Construction

- It is not possible to obtain the exact optimal martingale because the proof shows that it is equivalent to solving the original optimal stopping problem.
- There are two main approaches proposed in the literature for constructing the optimal martingales used in the penalty:
  - martingales from approximate value functions (Rogers (2002) and Haugh and Kogan (2004)):
    - From the above proof, we know that one way to construct the optimal penalty is to define

$$M_t = \sum_{i=1}^{t} \left( V_t(X_t) - \mathbb{E}[V_t(X_t)|X_{t-1}] \right).$$

      We may use approximate value function $\hat{V}_t$ in it.

# Martingale Construction (Continued)

- (Continued)
  - martingales from approximate policies (Andersen and Broadie (2006)):
    - Sometimes we may have good approximation to the optimal exercising policies.
    - Note that
      $$V_t(X_t) = E[(K - X_{\tau_t})^+ | X_t]$$
      and
      $$\mathbb{E}[V_t(X_t)|X_{t-1}] = \mathbb{E}[V_t(X_t)|X_{t-1}] = E[(K - X_{\tau_t})^+ | X_{t-1}]$$
    - We may use simulations to estimate $\Delta_t$.

# Policy Assessment

- Given a value function/a policy obtained from some ADP methods, we want to assess their quality, i.e., how far it is away from the optimality. The duality provides us a systematic way to do it.
  - For any value/policy, they are suboptimal. We have
$$\hat{V}_0 \leq V_0.$$
  - On the other hand, we can use the value functions/polices to construct duality to obtain an upper bound
$$V_0 \leq \overline{V}_0.$$
  - Duality gap:
$$0 \leq V_0 - \hat{V}_0 \leq \overline{V}_0 - \hat{V}_0$$

# Duality in a General SDP

- Consider the optimal control policy to maximize

$$\max_{\pi \in \mathcal{A}(W)} \mathbb{E}\left[\sum_{t=0}^{T-1} f_t(x_t, \pi_t) + g_T(x_T) \middle| x_0 = x\right]$$

with the underlying state process:

$$x_{n+1} = F(x_t, \pi_t, \xi_t)$$

- Take any sequence of functions $W = (W_0(\cdot), ..., W_T(\cdot))$. Define a penalty function $z$ such that

$$z(\mathbf{a}, \xi) = \sum_{t=0}^{T-1} D_t(a_t, \xi_t)$$

where

$$\begin{aligned} & D_t(a_t, \xi_t) \\ = \quad & \mathbb{E}[f_t(x_t, a_t) + W_{t+1}(F_t(x_t, a_t, \xi_t))] - (f_t(x_t, a_t) + W_{t+1}(F_t(x_t, a_t, \xi_t))). \end{aligned}$$

# Information Relaxation and Duality

- Brown et al. (2009) show that the strong duality achieves if we choose $W_t = V_t$ for all $t$, i.e.,

$$\sup_{\mathbf{W}} \mathbb{E}\left[\inf_{\boldsymbol{a}}\left\{\sum_{t=0}^{T-1} r_t(x_t, a_t, \xi_t) + r_T(x_T) + z(\mathbf{a}, \xi)\right\}\middle| x_0\right] = V_0(x_0).$$

- Information relaxation

# Using Information Relaxation to Assess Control Quality

- The information relaxation-based duality indicates a systematic approach to assess the quality of a given control policy $\alpha$.
  - Evaluating the policy, we obtain

$$\overline{W}_0(x) := \mathbb{E}\left[\sum_{s=0}^{T-1} r_s(x_s, \alpha_s(x_s), \xi_s) + r_T(x_T) \middle| x_0 = x\right] \geq V_0(x)$$

    for all $t$ and $x$.
  - Using $\overline{W}$ to construct penalty to solve the duality in the last slide, we have

$$\underline{W}_0(x) \leq V_0(x)$$

    for all $t$ and $x$.
  - We may assess the control quality from the gap between $\underline{W}_0$ and $\overline{W}_0$. Note that $\overline{W}_0(x) - V_0 \leq \overline{W}_0(x) - \underline{W}_0(x)$.

# Outline

# Reinforcement Learning

- Reinforcement learning (RL) is an active area of machine learning concerned with how agents ought to take actions in an environment so as to maximize cumulative reward.
- The environment is typically formulated as a Markov decision process (MDP), as many reinforcement learning algorithms for this context utilize dynamic programming techniques. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP.

# Reinforcement Learning (Continued)

- In the RL community, the term "model" often refers to the transition probability of the underlying system. Some physical systems are so complex that their mathematical description is intractable. In such systems, we may make a decision but then have to observe the outcomes of the decision from a physical process, rather than depending on mathematical equations.

# Q-Learning

- Q-learning is one of the oldest algorithm from the RL community, named after the variable $Q(x, a)$, which is an estimate of the value of being in a state $s$ and taking action $a$.
  - Recall the Bellman equation in the case of infinite-time horizon SDP indicates that the optimal value function $V$ satisfies

$$V(x) = \max_{a \in \mathcal{A}(x)} \mathbb{E}\left[ r(x_t, a) + \gamma V(x_{t+1}^a) \middle| x_t = x \right].$$

  - Let

$$Q(x, a) = \mathbb{E}\left[ r(x_t, a) + \gamma V(x_{t+1}^a) \middle| x_t = x \right].$$

    Then, $Q(x, a)$ satisfies

$$V(x) = \max_a Q(x, a)$$

    and

$$Q(x, a) = \mathbb{E}\left[ r(x_t, a) + \gamma \max_{b \in \mathcal{A}(x_{t+1}^a)} Q(x_{t+1}^a, b) \middle| x_t = x \right].$$

# Q-Learning (Continued)

- Q-learning algorithm:
  - Step 1: Initialize $Q^0(s, a)$ for all $s$ and $a$ and set $n = 0$.
  - Step 2: At $x_n$, choose action $a_n$ according to

$$a_n = \arg\max_a Q^n(x_n, a)$$

  - Step 3: Observe that the state evolves to $x_{n+1}$ and collect the reward $r(x_n, a_n)$. Let

$$\delta^n = r(x_n, a_n) + \gamma \max_{b \in \mathcal{A}(x_{n+1})} Q^n(x_{n+1}, b).$$

    Update

$$Q^{n+1}(x_n, a_n) = (1 - \alpha_n) Q^n(x_n, a_n) + \alpha_n \delta^n$$

    and

$$Q^{n+1}(x, a) = Q^n(x, a)$$

    for the other state-action pair $(x, a)$. Let $n = n + 1$ and go to Step 2.

# Q-Learning (Continued)

- Remarks:
  - The power of $Q$-learning arises when we either do not have explicit transition probabilities of the system or do not know the explicit distribution of rewards. Some literature would like to refer to this formulation as model-free dynamic programming.
  - We adopt a "greedy" method in selecting actions in Step 2. That will lead to a serious exploration problem. A huge literature is proposed to deal with this problem.

# Outline

# Exploitation and Exploration

- The essence of ADP is that we try to visit states to estimate the value function of being in the state. Therefore, a fundamental challenge is: should we make a decision because we think it is the best decision based on our current estimate of the values of states, or do we make a decision just to try something new?
- This choice is known in the ADP literature as the exploitation vs. exploration problem.

# Example: Nomadic Trucker (Powell, 2011)

- A trucker moves around the US to meet random demands arising in different cities.
  - State: cities
  - Action: which city to move to in the next period
  - Reward: Some bonus earned from moving into a city to meet the stochastic demands in that city
  - Q-learning
- Different initial $Q$ values:
  - $Q^0 = 0$ in the top panel
  - $Q^0 = 2,000$ in the bottom panel

(a) Low initial estimate of the value function

(b) High initial estimate of the value function

**Figure 12.1** Effect of value function initialization on search process. (*a*) Low initial estimate, which produces limited exploration; (*b*) high initial estimate, which forces exploration of the entire state space.

# Heuristic Learning Policies

- The tradeoff between exploration and exploitation points to the importance of Step 2 in the generic Q-learning algorithm.
  - Pure exploitation (or greedy strategy): the one presented in the above generic Q-learning algorithm is known as a pure exploitation policy. We always try the best action based on the current estimation. As shown by the trucker example, it is easy to become struck in a local solution.
  - Pure exploration: use an exogenous process to choose either a state to visit or a state-action pair. As long as we can guarantee that we will visit every possible state, we can show that the $Q$ learning algorithm will converge to the optimal value with the help of some ergodic properties. However, such strategy is unlikely to work in large-scaled problems.

# Heuristic Learning Policies (Continued)

- (Continued)
  - $\epsilon$-greedy method: take an action according to

$$\begin{cases} \text{The action taken by the greedy method,} & \textit{prob.} = 1 - \epsilon; \\ \text{a random action,} & \textit{prob.} = \epsilon. \end{cases}$$

  - The $\epsilon$-greedy method maintains a certain degree of forced exploration, while the exploitation steps focus attention on the actions that appear to be most valuable.
  - Intuitively, we should use more exploratory strategies in the early stage of the Q-learning because we know little about the environment. As our knowledge increases, we should gradually switch to exploitation. However, a fixed $\epsilon$ does not reflect this consideration. In practice, people also use

$$\epsilon^n(s) = \frac{c}{N^n(s)}$$

where $N^n(s)$ is the number of times we have visited state $s$ by iteration $n$.

# Heuristic Learning Policies (Continued)

- (Continued)
  - Boltzmann exploration: In Step 2, the decision maker choose an action with "mistake". That is,

$$a^* = \arg\max_a [Q(x, a) + \epsilon^a]$$

where $\epsilon^a$ are independent random variables indexed by action.
  - It is easy to prove, under some assumptions of the distribution $\epsilon$ (say, it follows the double-exponential distribution), the probability to choose action $a$ in state $s$ is given by the following Boltzmann distribution:

$$P(s, a) = \frac{\exp(Q(x, a)/T)}{\sum_b \exp(Q(x, b)/T)}.$$

  - Here $T$ is known as the temperature. As $T \to +\infty$, the probability of choosing different actions becomes uniform (pure exploration). As $T \to 0$, the probability of choosing the action corresponding to the largest $Q(x, a)$ approaches 1 (pure exploitation).