ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA HỆ THỐNG THÔNG TIN

BÁO CÁO ĐỒ ÁN

LỚP: IS254.P11

Tìm hiểu về Policy Search

Giáo viên hướng dẫn: Ths. Nguyễn Hồ Duy Trí

Trương Vĩnh Thuận - 21522653

Bùi Văn Thái - 21522577

Nguyễn Thế Vinh – 21522794

Chế Duy Khang – 21522187

THÀNH PHỐ HỒ CHÍ MINH, 2024

MỤC LỤC

1. Giới thiệu bài toán

Trong các hệ thống thông minh hiện đại, hệ hỗ trợ ra quyết định (Decision Support System - DSS) đóng vai trò quan trọng trong việc cung cấp những gợi ý và quyết định tối ưu cho người dùng dựa trên dữ liệu và các mô hình phân tích. Một trong những bài toán nổi bật trong hệ hỗ trợ ra quyết định là tìm kiếm và tối ưu hóa chính sách (Policy Search), đặc biệt trong lĩnh vực học tăng cường (Reinforcement Learning).

Vấn đề đặt ra là khi không gian trạng thái (state space) lớn hoặc liên tục, việc đánh giá giá trị của mỗi trạng thái thông qua hàm giá trị (value function) trở nên rất phức tạp và tốn kém. Thay vào đó, chúng ta có thể tiếp cận bài toán theo một hướng khác: tìm kiếm chính sách mà không cần tính toán hàm giá trị cho toàn bộ không gian trạng thái. Điều này được gọi là Policy Search.

Mục tiêu xác định một chính sách tối ưu π^* mà đối tượng có thể tuân theo để đạt được giá trị kỳ vọng lớn nhất. Thay vì tối ưu hóa giá trị của từng trạng thái riêng lẻ, Policy Search tập trung vào việc tối ưu hóa trực tiếp không gian chính sách, giúp giảm bớt độ phức tạp khi làm việc với các bài toán có không gian trạng thái lớn.

Phương pháp tìm kiếm chính sách:

- Chính sách tham số hóa (Parameterized Policy π_θ): Một trong những phương pháp phổ biến là biểu diễn chính sách dưới dạng tham số hóa, nơi θ là các tham số mô hình. Điều này giúp quá trình tối ưu hóa diễn ra dễ dàng hơn, vì không gian chính sách thường có kích thước nhỏ hơn nhiều so với không gian trạng thái.
- Các phương pháp tìm kiếm không sử dụng gradient (Gradient-free search methods): Đối với những bài toán mà việc ước lượng gradient khó khăn hoặc không chính xác, các phương pháp tìm kiếm không dựa vào gradient sẽ được ưu tiên. Các phương pháp này có khả năng tránh được các cực trị cục bộ và khám phá không gian chính sách tốt hơn.

2. Approximate Policy Evaluation (Đánh giá chính sách xấp xỉ)

2.1 Markov Decision Process (MDP)

2.1.1 Khái niệm:

MDP (Markov Decision Process) là một mô hình toán học được sử dụng để mô tả các bài toán ra quyết định mà trong đó kết quả không chỉ phụ thuộc vào hành động của người ra quyết định mà còn phụ thuộc vào yếu tố ngẫu nhiên. MDP được sử dụng rộng rãi trong các lĩnh vực như trí tuệ nhân tạo, học tăng cường (Reinforcement Learning), lý thuyết trò chơi và tự động hóa.

2.1.2 Các thành phần của MDP

MDP thường được định nghĩa bởi 4 thành phần chính:

- **S - Không gian trạng thái (State space)**:
    - Tập hợp tất cả các trạng thái có thể xảy ra trong hệ thống.
    - Ví dụ: Trong một trò chơi cờ vua, mỗi bố trí quân cờ trên bàn là một trạng thái.
- **A - Không gian hành động (Action space)**:
    - Tập hợp các hành động có thể thực hiện tại mỗi trạng thái.
    - Ví dụ: Trong trò chơi cờ vua, các hành động có thể là di chuyển quân cờ từ vị trí này sang vị trí khác.
- **P - Chức năng chuyển trạng thái (Transition function)**:
    - P (s′|s, a) là xác suất chuyển từ trạng thái s sang trạng thái s′ khi thực hiện hành động a.
    - Ví dụ: Trong một trò chơi xúc xắc, nếu bạn tung xúc xắc và nhận được một con số, đây là một yếu tố ngẫu nhiên quyết định trạng thái tiếp theo.
- **R - Hàm phần thưởng (Reward function)**:
    - R (s, a) là phần thưởng nhận được khi thực hiện hành động a tại trạng thái s.

- Ví dụ: Trong trò chơi cờ vua, phần thưởng có thể là điểm số bạn nhận được sau khi thực hiện một nước đi tốt.

Hệ số Chiết khấu (γ)

- Hệ số chiết khấu γ (discount factor) là một giá trị nằm trong khoảng [0,1].
- Nó cho biết mức độ ưu tiên giữa phần thưởng hiện tại và phần thưởng tương lai.
    - $\gamma = 0$: Chỉ quan tâm đến phần thưởng hiện tại, không quan tâm đến tương lai.
    - $\gamma = 1$: Quan tâm nhiều hơn đến phần thưởng tương lai.

2.1.3 Ứng dụng

MDP có nhiều ứng dụng trong thực tế, bao gồm:

- Học tăng cường (Reinforcement Learning): Các thuật toán như Q-learning và chính sách Monte Carlo dựa trên MDP để tối ưu hóa quyết định của agent.
- Quản lý chuỗi cung ứng: Để tối ưu hóa quyết định đặt hàng và tồn kho.
- Điều khiển robot: Để tìm đường đi tối ưu cho robot.
- Trí tuệ nhân tạo trong trò chơi: Tối ưu hóa chiến lược của các agent trong trò chơi.

2.1.4 Ví dụ minh họa

Trong trò chơi Pac-Man:

- Trạng thái: Vị trí của Pac-Man, vị trí của ma quái, và vị trí của các điểm.
- Hành động: Di chuyển lên, xuống, trái, phải.
- Chuyển trạng thái: Pac-Man di chuyển và có thể va chạm với ma quái.
- Phần thưởng: Pac-Man nhận được điểm khi ăn điểm hoặc bị trừ điểm khi bị bắt.

2.2 Giới thiệu về Policy Evaluation

2.2.1 Tổng quan

Trong một Markov Decision Process (MDP), **policy evaluation** là quá trình tính toán giá trị kỳ vọng (value function) của một chính sách cố định π. Giá trị kỳ vọng này cho biết phần thưởng kỳ vọng mà một agent có thể nhận được khi bắt đầu từ một trạng thái nhất định và tuân theo chính sách π trong suốt quá trình.

Giá trị kỳ vọng của một trạng thái s khi theo chính sách π được ký hiệu là $U_\pi(s)$. Nó đo lường lợi nhuận kỳ vọng từ trạng thái đó khi hành động theo chính sách π.

2.2.2 Phương trình Bellman cho Policy Evaluation

Phương trình Bellman cho policy evaluation được biểu diễn như sau:

$$U_\pi(s) = R(s, \pi(s)) + \gamma s \sum_{s\prime} T\big(\, s' \mid s, \pi(s)\,\big) U_\pi(s')$$

Trong đó:

- $U_\pi(s)$: Giá trị kỳ vọng của trạng thái s theo chính sách π.
- $R(s, \pi(s))$: Phần thưởng nhận được khi thực hiện hành động $\pi(s)$ tại trạng thái s.
- γ: Hệ số chiết khấu (discount factor), nằm trong khoảng [0, 1], thể hiện mức độ ưu tiên giữa phần thưởng hiện tại và phần thưởng trong tương lai.
- $T(s' \mid s, \pi(s))$: Xác suất chuyển từ trạng thái s sang trạng thái s' khi thực hiện hành động $\pi(s)$.

Policy evaluation hội tụ vì phương trình cập nhật là một dạng contraction mapping. Điều này có nghĩa là mỗi lần cập nhật sẽ thu nhỏ khoảng cách giữa giá trị hiện tại và giá trị thực tế. Vì vậy, khi lặp lại quá trình cập nhật nhiều lần, giá trị sẽ hội tụ về giá trị chính xác của $U_\pi(s)$.

2.3. Approximate Policy Evaluation

2.3.1. Tổng quan

Policy Evaluation là một bước cơ bản trong việc tìm kiếm chính sách (policy search). Ý tưởng là tính toán giá trị kỳ vọng (expected return) khi theo một chính sách π từ một phân phối trạng thái ban đầu $b(s)$. Giá trị này, được ký hiệu là $U^{\pi}(s)$ có thể được tính theo công thức:

$$U(\pi) = \sum_{s} b(s)U^{\pi}(s)$$

Trong đó:

- $b(s)$: phân phối xác suất trạng thái ban đầu.
- $U^{\pi}(s)$: giá trị kỳ vọng tại trạng thái s khi theo chính sách π.

Tuy nhiên, khi không gian trạng thái lớn hoặc liên tục, không thể tính $U(\pi)$ chính xác. Thay vào đó, giá trị này được xấp xỉ bằng cách sử dụng các quỹ đạo (trajectories) của trạng thái, hành động và phần thưởng, giá trị U(π) được xấp xỉ bằng cách:

Lấy mẫu quỹ đạo (trajectories):

- Một quỹ đạo $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ gồm các trạng thái, hành động và phần thưởng, được tạo ra từ chính sách π.

Sử dụng hàm mật độ xác suất $p_{\pi}(\tau)$:

- $p_{\pi}(\tau)$ là xác suất của quỹ đạo τ khi áp dụng chính sách π, bắt đầu từ phân phối trạng thái ban đầu $b(s)$

Công thức kỳ vọng liên quan đến quỹ đạo:

$$U(\pi) = \mathrm{E}_{\tau}[R(\tau)] = \int p_{\pi}(\tau)R(\tau)d\tau$$

- $R(\tau)$: tổng phần thưởng chiết khấu của quỹ đạo τ.

2.4. Monte Carlo Policy Evaluation

2.4.1. Tổng quan

Monte Carlo Policy Evaluation là một phương pháp để xấp xỉ $U(\pi)$. Monte Carlo Policy Evaluation không yêu cầu biết chính xác phân phối $p_\pi(\tau)$. Thay vào đó, chỉ cần sinh các trajectory bằng cách thực thi chính sách π, điều này phù hợp khi không gian trạng thái hoặc hành động lớn.

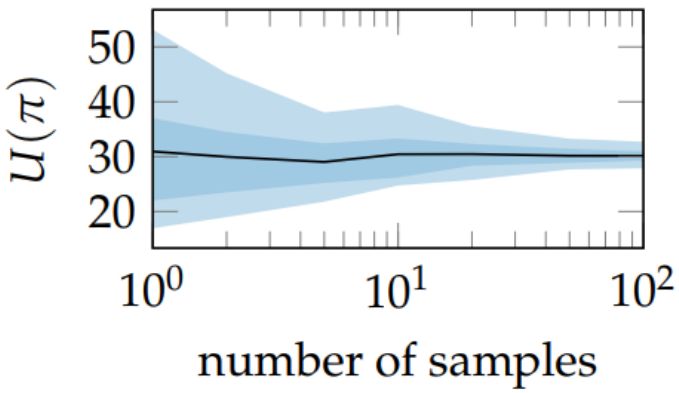
- Ý tưởng chính: Lấy mẫu m quỹ đạo độc lập từ chính sách π và tính giá trị trung bình tổng phần thưởng của các quỹ đạo này.
- Công thức xấp xỉ:

$$U(\pi) \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau^{(i)})$$

  - $R(\tau^{(i)})$: tổng phần thưởng của quỹ đạo thứ i.
  - m: số lượng quỹ đạo.

2.4.2 Đặc điểm:

- Phương pháp Monte Carlo là ngẫu nhiên, tức là giá trị $U(\pi)$ có thể thay đổi giữa các lần đánh giá.
- Khi tăng số mẫu m, phương sai trong kết quả đánh giá sẽ giảm, dẫn đến kết quả ổn định hơn (như minh họa trong Figure 10.2).



2.4.3. Monte Carlo Evaluation so với các phương pháp khác:

- Ưu điểm:
    - Đơn giản, dễ triển khai.
    - Không yêu cầu phải lưu trữ toàn bộ không gian trạng thái.
- Nhược điểm:
    - Tốn thời gian nếu số mẫu m lớn.
    - Kết quả phụ thuộc vào số lượng mẫu và chất lượng của việc lấy mẫu.

2.5. Ứng dụng của Approximate Policy Evaluation

- Lập kế hoạch trong robot:
    - Khi một robot cần đưa ra các hành động tối ưu dựa trên chính sách hiện tại, việc đánh giá chính sách là cần thiết để ước lượng giá trị các hành động và trạng thái trong tương lai.
    - Ví dụ: Robot giao hàng ước lượng tổng phần thưởng kỳ vọng (tương ứng với số đơn hàng hoàn thành) dựa trên trạng thái hiện tại (vị trí, pin còn lại).
- Điều khiển trong trò chơi:
    - Trong trò chơi (game AI), APE có thể được dùng để đánh giá hiệu quả của một chiến lược cụ thể mà không cần tính toán toàn bộ không gian trạng thái.
    - Ví dụ: Trong cờ vua, sử dụng APE để đánh giá nhanh giá trị của một nước đi dựa trên vị trí quân cờ mà không cần tính toàn bộ cây trạng thái.
- Quản lý chuỗi cung ứng:

- Được sử dụng để đánh giá các chính sách vận hành (ví dụ: chính sách tồn kho, chính sách vận chuyển) khi không thể mô hình hóa chính xác động lực học của chuỗi cung ứng.
- Ví dụ: Tối ưu hóa mức tồn kho dựa trên ước tính giá trị kỳ vọng của chính sách đặt hàng.

- Y học cá nhân hóa:
  - APE giúp đánh giá hiệu quả của các chính sách điều trị (ví dụ: lựa chọn loại thuốc) dựa trên dữ liệu bệnh nhân để tối ưu hóa lợi ích sức khỏe dài hạn.
  - Ví dụ: Ước lượng tổng giá trị kỳ vọng của một phác đồ điều trị dựa trên dữ liệu bệnh nhân trong bệnh viện.

2.6. Ví dụ cụ thể

Bài toán: Đánh giá một chính sách chơi trò Tic-Tac-Toe dựa trên một số lượng lớn các trò chơi giả lập.

Áp dụng APE:

- Sử dụng phương pháp Monte Carlo Policy Evaluation để ước tính giá trị kỳ vọng $U(\pi)$ của các trạng thái ban đầu bằng cách chơi nhiều trận giả lập và ghi lại điểm số của từng trận.
- Kết quả là một ước lượng của chính sách hiện tại, giúp bạn hiểu xem chiến lược chơi có hiệu quả hay không.
- Chạy 3 quỹ đạo:
  - Quỹ đạo 1: Trajectory thu được có tổng phần thưởng G(1)=5.0.
  - Quỹ đạo 2: Trajectory thu được có tổng phần thưởng G(2)=7.0.
  - Quỹ đạo 3: Trajectory thu được có tổng phần thưởng G(3)=6.0.
  - Giá trị kỳ vọng U(π) sẽ được tính bằng trung bình của tổng phần thưởng: U(π)= 1/3 * (5.0 + 7.0 + 6.0) = 6.0

3. Local search

3.1 Khái niệm

Local Search là một kỹ thuật tối ưu hóa được sử dụng để cải thiện các giải pháp thông qua việc tìm kiếm các lân cận của giải pháp hiện tại. Local Search bắt đầu với một giải pháp ban đầu và liên tục thăm dò các điểm lân cận của nó. Nếu một điểm lân cận mang lại giá trị tốt hơn cho hàm mục tiêu, giải pháp sẽ được cập nhật và tiếp tục thăm dò từ điểm này. Quá trình này lặp lại cho đến khi không còn cải thiện nào có thể thực hiện hoặc khi hội tụ.

Trong policy search, mục tiêu là tìm ra một bộ tham số chính sách tối ưu θ để tối đa hóa hàm tiện ích U(θ).

3.2 Ứng dụng của Local Search

❖ Tối ưu hóa cấu trúc mạng Bayesian

**Mạng Bayesian là một mô hình đồ thị xác suất biểu diễn các mối quan hệ nhân quả hoặc phụ thuộc giữa các biến.*

**Tìm kiếm cấu trúc tốt nhất của mạng Bayesian là quá trình thử nghiệm các cấu trúc mạng khác nhau để tìm ra cấu trúc tối ưu, có thể đại diện tốt nhất cho dữ liệu quan sát.*

Thay đổi cấu trúc lân cận: Tìm kiếm lân cận liên quan đến việc thực hiện các thay đổi nhỏ đối với cấu trúc hiện tại, ví dụ như thêm hoặc xóa một cạnh trong mạng. Bằng cách thay đổi các lân cận này và tính toán điểm số của mỗi cấu trúc, thuật toán có thể tiến dần đến cấu trúc mạng tốt nhất.

❖ Điều chỉnh tham số mô hình học máy

Trong học máy, các tham số của mô hình cần được tinh chỉnh để tối ưu hóa hiệu suất của mô hình trên dữ liệu.

Với Local Search, ta có thể thử nghiệm các giá trị khác nhau của các tham số, tìm ra cấu hình nào cho kết quả tốt nhất. Không cần phải tính đạo hàm (không cần gradient),

phương pháp này chỉ dựa trên việc so sánh các lân cận. Điều này hữu ích với các mô hình phức tạp, nơi tính gradient là khó khăn hoặc không thể thực hiện được.

❖ Bài toán tối ưu hóa lộ trình

Trong các bài toán như điều phối xe (như trong logistics, nơi xe phải giao hàng đến nhiều địa điểm) hoặc tìm đường (như trên bản đồ), mục tiêu là tìm ra lộ trình tốt nhất để tối ưu hóa chi phí, thời gian, hoặc quãng đường di chuyển.

Local Search trong trường hợp này giúp tối ưu hóa lộ trình bằng cách điều chỉnh một phần nhỏ của lộ trình hiện tại. Ví dụ: hoán đổi thứ tự của hai địa điểm hoặc chọn một lộ trình khác gần đó. Thuật toán sẽ thử nghiệm các phương án này và lựa chọn lộ trình tốt nhất dựa trên các thay đổi lân cận.

3.3 Phương pháp Hooke-Jeeves

```
struct HookeJeevesPolicySearch
    θ # initial parameterization
    α # step size
    c # step size reduction factor
    ϵ # termination step size
end

function optimize(M::HookeJeevesPolicySearch, π, U)
    θ, θ′, α, c, ϵ = copy(M.θ), similar(M.θ), M.α, M.c, M.ϵ
    u, n = U(π, θ), length(θ)
    while α > ϵ
        copyto!(θ′, θ)
        best = (i=0, sgn=0, u=u)
        for i in 1:n
            for sgn in (-1,1)
                θ′[i] = θ[i] + sgn*α
                u′ = U(π, θ′)
                if u′ > best.u
                    best = (i=i, sgn=sgn, u=u′)
                end
            end
            θ′[i] = θ[i]
        end
        if best.i != 0
            θ[best.i] += best.sgn*α
            u = best.u
        else
            α *= c
        end
    end
    return θ
end
```

Giả sử cần tối ưu hóa một hàm mục tiêu U(θ) với tham số θ là một vector nhiều chiều. Thuật toán Hooke-Jeeves thực hiện theo các bước sau:

Khởi Tạo: Chọn một điểm xuất phát θ_0 trong không gian tìm kiếm, thiết lập giá trị ban đầu cho kích thước bước α và các tham số như hệ số giảm bước c, ngưỡng dừng ε.

Bước Khám Phá:

Từ vị trí hiện tại θ, kiểm tra từng hướng của các trục tọa độ. Ví dụ, với vector $q = \theta_1, \theta_2, .. \theta_n$ lần lượt thử tăng hoặc giảm từng thành phần θ_i theo bước $\pm\alpha$

Sau khi thử tất cả các hướng, chọn điểm lân cận tốt nhất có giá trị hàm mục tiêu U(θ) cao hơn (đối với cực đại) hoặc thấp hơn (đối với cực tiểu).

Bước Di Chuyển:

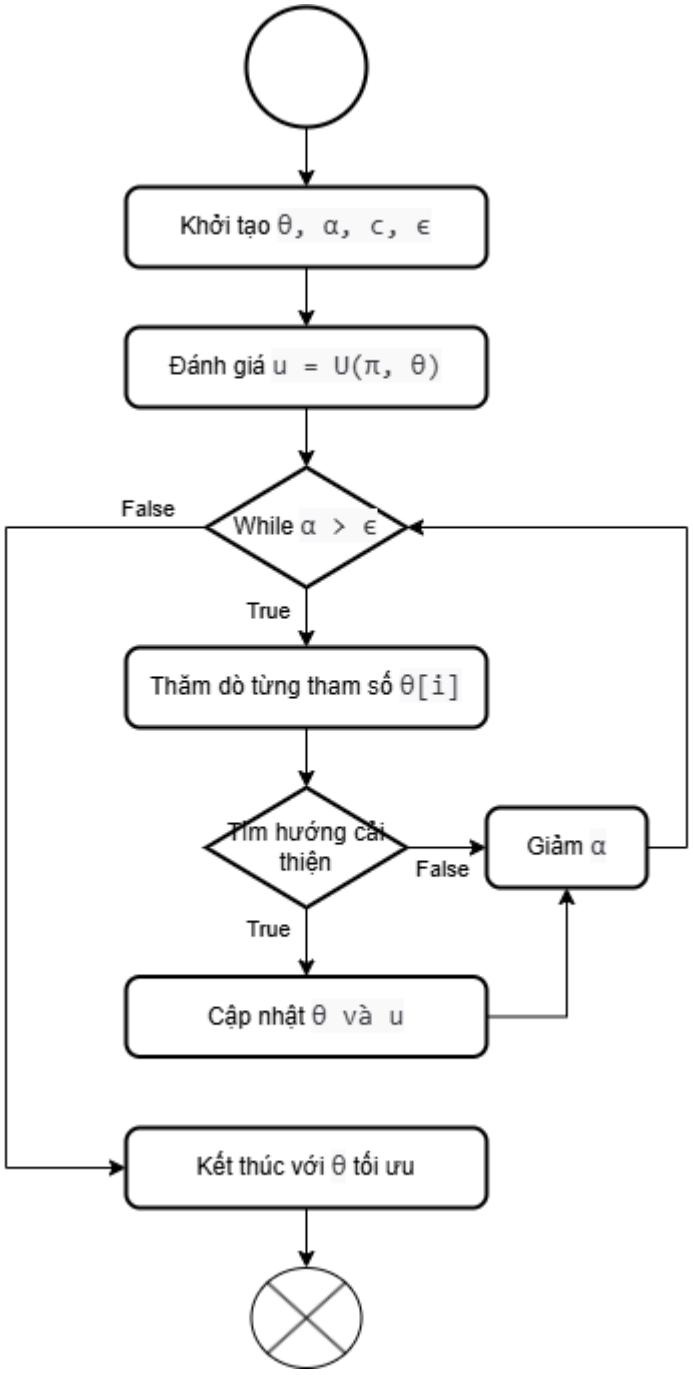
Nếu bước khám phá dẫn đến cải thiện, thực hiện một bước di chuyển lớn hơn theo hướng của điểm tốt nhất tìm được, hy vọng tìm ra vùng có giá trị tốt hơn.

Nếu không có cải thiện, giảm kích thước bước bằng cách nhân với hệ số c

Kiểm Tra Dừng:

Lặp lại các bước trên cho đến khi kích thước bước α nhỏ hơn ngưỡng ϵ, hoặc không còn cải thiện nào trong các điểm lân cận.

Kết Thúc: Trả về điểm tối ưu θ tìm được.



3.4 Ví dụ

Hàm mục tiêu cần tối đa hóa:

$$f(\theta_1, \theta_2) = -(\theta_1^2 + \theta_2^2)$$

Bước 1: Khởi tạo

Tham số khởi tạo

$$\theta = [1.0, 1.0]$$

Bước di chuyển ban đầu:

$$\alpha = 0.5$$

Hệ số giảm bước di chuyển:

$$c = 0.5$$

Ngưỡng dừng thuật toán:

$$\epsilon = 0.01$$

Bước 2: Đánh giá hàm mục tiêu tại điểm khởi tạo

$$f(1.0,1.0) = -(1.0^2 + 1.0^2) = -2.0$$

Khởi tạo biến tốt nhất:

$$best = (i = 0, sgn = 0, u = -2.0)$$

Bước 3: Khảo sát

Khảo sát θ_1 (với $\theta_2 = 1$):

1. Thử $\theta_1 = 1.5$ *và* $\theta_2 = 1$ $(sgn = 1)$

$$f(1.5, 1.0) = -(1.5^2 + 1^2) = -(2.25 + 1) = -3.25$$

   - Không tốt hơn giá trị cũ $(best.u = -2)$

2. Thử $\theta_1 = 0.5$ *và* $\theta_2 = 1$ $(sgn = -1)$

$$f(0.5, 1.0) = -(0.5^2 + 1^2) = -(0.25 + 1) = -1.25$$

   - Tốt hơn giá trị cũ $(best.u = -2)$
   - Cập nhật best:

$$best = (i = 1, sgn = -1, u = -1.25)$$

Khảo sát θ_2 (với $\theta_1 = 1$):

1. Thử $\theta_1 = 1$ và $\theta_2 = 1.5$ $(sgn = 1)$

$$f(1.5, 1.0) = -(1^2 + 1.5^2) = -(1 + 2.25) = -3.25$$

   - Không tốt hơn giá trị cũ ($best.u = -1.25$)

2. Thử $\theta_1 = 1$ và $\theta_2 = 0.5$ $(sgn = -1)$

$$f(1, 0.5) = -(1^2 + 0.5^2) = -(1 + 0.25) = -1.25$$

   - Không tốt hơn giá trị cũ ($best.u = -1.25$)

Bước 4: Cập nhật tham số và tiếp tục kiểm tra

Cập nhật θ với giá trị tốt nhất vừa tìm được trong khảo sát

$$\theta = [0.5,1.0]$$

Hàm mục tiêu hiện tại:

$$f(0.5,1.0) = -1.25$$

Giảm bước di chuyển ($\alpha = \alpha * c = 0.5 * 0.5 = 0.25$)

Tiếp tục các bước trên cho đến khi ta được $f(0,0) = 0, best.u = 0$

4. Genetic Algorithm

4.1. Khái niệm

Thuật toán di truyền (GA) là một lớp thuật toán tối ưu hóa lấy cảm hứng từ quá trình chọn lọc tự nhiên. Chúng mô phỏng quá trình tiến hóa, phát triển quần thể các giải pháp ứng viên để tìm ra giải pháp tối ưu hoặc gần tối ưu cho một vấn đề.

Thuật toán di truyền mô phỏng quá trình chọn lọc tự nhiên, nghĩa là những loài có thể thích nghi với những thay đổi trong môi trường của chúng có thể sống sót, sinh sản và chuyển sang thế hệ tiếp theo. Chúng mô phỏng "sự sống còn của những kẻ khỏe mạnh nhất" giữa các cá thể của các thế hệ liên tiếp để giải quyết một vấn đề. Mỗi thế hệ bao gồm một quần thể cá thể và mỗi cá thể đại diện cho một điểm trong không gian tìm kiếm và giải pháp khả thi. Mỗi cá thể được biểu diễn dưới dạng một chuỗi ký tự/số nguyên/số thực/bit. Chuỗi này tương tự như Nhiễm sắc thể.

4.2. Cơ chế hoạt động

Khởi tạo (Initialization):

Quần thể: Thuật toán bắt đầu với một quần thể được tạo ngẫu nhiên, thường được biểu diễn dưới dạng nhiễm sắc thể (chuỗi gen). Ví dụ, trong bối cảnh của bài toán Đường đi của người bán hàng (TSP), mỗi nhiễm sắc thể sẽ biểu diễn một đường đi có thể đến tất cả các thành phố.

Nhiễm sắc thể thường được mã hóa dưới dạng danh sách hoặc mảng các giá trị (chẳng hạn như số nguyên hoặc chữ số nhị phân) biểu diễn một giải pháp tiềm năng.

Hàm đánh giá (Fitness Evaluation):

Mỗi giải pháp (nhiễm sắc thể) trong quần thể được đánh giá bằng hàm đánh giá. Hàm đánh giá đo lường mức độ tốt của một giải pháp so với các giải pháp khác.

Trong trường hợp của TSP, hàm đánh giá tính toán tổng quãng đường di chuyển cho một tuyến đường nhất định và mục tiêu là giảm thiểu khoảng cách này.

Một giải pháp tốt hơn sẽ có giá trị đánh giá cao hơn (trong một số trường hợp, giá trị thấp hơn, chẳng hạn như khoảng cách hoặc thời gian, có thể được ưu tiên).

Chọn lọc (Selection):

Là quá trình mà các nhiễm sắc thể (giải pháp) được chọn để sinh sản và truyền genes của chúng cho thế hệ tiếp theo.

Một số phương pháp chọn lọc phổ biến, bao gồm:

- **Roulette Wheel Selection:** Các cá thể được chọn dựa trên giá trị hàm đánh giá của chúng, những nhiễm sắc thể khỏe mạnh hơn có cơ hội được chọn cao hơn. Xác suất chọn lọc tỷ lệ thuận với thể lực của nhiễm sắc thể.
- **Tourament Selection:** Một nhóm nhiễm sắc thể được chọn ngẫu nhiên và nhiễm sắc thể tốt nhất được chọn để sinh sản.

Mục tiêu của quá trình chọn lọc là ưu tiên các giải pháp tốt hơn, do đó, chúng có cơ hội cao hơn để truyền thông tin di truyền của mình cho thế hệ tiếp theo.

Lai tạo (Crossover):

Lai ghép là quá trình trong đó hai thế hệ bố mẹ kết hợp để tạo ra con cái, kết hợp các đặc điểm tốt từ hai cha mẹ có thể dẫn đến thế hệ con tốt hơn.

Một số phương pháp lai tạo phổ biến bao gồm:

Lai ghép một điểm (One-point crossover): Một điểm(gene) được chọn ngẫu nhiên tráo đổi phần đuôi từ điểm đó (gene) của 2 cha mẹ cho nhau để tạo ra 2 con

Lai ghép hai điểm (Two-points crossover): Hai điểm (genes) được chọn, tráo đổi phần giữa 2 vị trí điểm (genes) của cha mẹ.

Đột biến (Mutation):

Đột biến tạo ra tính ngẫu nhiên bằng cách thay đổi ngẫu nhiên một hoặc nhiều genes của một nhiễm sắc thể (giải pháp), giúp duy trì tính đa dạng trong quần thể và ngăn ngừa sự hội tụ sớm trên các giải pháp không tối ưu.

Trong TSP, điều này có thể liên quan đến việc hoán đổi hai thành phố trên tuyến đường hoặc đảo ngược một phần của tuyến đường.

- **Sau khi lai tạo (Crossover) và đột biến (Mutation), quần thể con mới thay thế quần thể cũ và quá trình này được lặp lại. Thông thường, các giải pháp tốt nhất từ quần thể hiện tại được giữ lại ở thế hệ tiếp theo để đảm bảo chất lượng của các giải pháp không bị suy giảm theo thời gian.**

4.3. Ưu điểm và nhược điểm

Ưu điểm:

- **Tính linh hoạt**: GA có thể áp dụng cho nhiều loại vấn đề khác nhau, từ tối ưu hóa liên tục đến rời rạc, từ đơn giản đến phức tạp.
- **Không yêu cầu kiến thức về đạo hàm**: Không cần phải tính toán đạo hàm của hàm mục tiêu, rất hữu ích khi hàm mục tiêu phức tạp hoặc không liên tục.
- **Khả năng tìm kiếm toàn cục**: GA có thể tìm kiếm trong không gian giải pháp rộng lớn và tránh bị kẹt trong các cực địa phương.
- **Dễ dàng song song hóa**: Các quá trình như chọn lọc, lai tạo và đột biến có thể được thực hiện song song, giúp tăng tốc độ tính toán.
- **Khả năng xử lý bài toán đa mục tiêu**: GA có thể được mở rộng để giải quyết các bài toán tối ưu hóa đa mục tiêu.

Nhược điểm:

- **Không đảm bảo tìm được giải pháp tối ưu**: GA chỉ tìm được các giải pháp gần đúng, không đảm bảo tìm được giải pháp tối ưu tuyệt đối.
- **Tham số phức tạp**: Cần phải điều chỉnh nhiều tham số (như tỷ lệ đột biến, tỷ lệ lai tạo, kích thước quần thể) để đạt hiệu quả tốt.
- **Tốn tài nguyên tính toán**: Để đạt được kết quả tốt, GA thường yêu cầu nhiều thế hệ và tính toán phức tạp, đòi hỏi tài nguyên tính toán lớn.
- **Khó khăn trong việc xác định điều kiện dừng**: Điều kiện dừng của GA có thể không rõ ràng, dẫn đến việc dừng thuật toán quá sớm hoặc quá muộn.

- **Khả năng mất đa dạng**: Nếu không có cơ chế bảo vệ đa dạng, quần thể có thể nhanh chóng bị mất đa dạng, dẫn đến kết quả không tốt.

4.4. Ứng dụng

- **Bài toán tối ưu hóa**: GA được sử dụng để giải quyết các bài toán tối ưu hóa, tìm kiếm giải pháp tốt nhất trong một không gian lớn các khả năng. Ví dụ: tối ưu hóa hàm toán học, điều chỉnh tham số, phân bổ tài nguyên, v.v.
- **Tối ưu hóa tổ hợp**: GA hiệu quả trong các bài toán tối ưu tổ hợp, như bài toán người bán hàng (TSP), phân phối xe (VRP), xếp lịch công việc, đóng gói thùng, và căn chỉnh chuỗi DNA.
- **Học máy**: GA được áp dụng trong việc tối ưu hóa các tham số của mô hình học máy, chẳng hạn như tốc độ học, tham số điều chỉnh, và kiến trúc mạng nơ-ron. Nó cũng có thể được dùng để chọn lọc đặc trưng.
- **Robot học tiến hóa**: GA được sử dụng để phát triển chiến lược điều khiển và hành vi của robot. Các tham số điều khiển robot được đại diện dưới dạng nhiễm sắc thể, và GA tìm kiếm giải pháp tối ưu.
- **Xử lý ảnh và tín hiệu**: GA tối ưu hóa các tham số trong các thuật toán xử lý ảnh như tái tạo ảnh, loại bỏ nhiễu, và nhận dạng mẫu. Nó cũng giúp tối ưu hóa bộ lọc tín hiệu trong việc loại bỏ nhiễu.
- **Thiết kế và sáng tạo**: GA có thể được áp dụng trong việc tạo ra các thiết kế nghệ thuật, sáng tác âm nhạc và thiết kế trò chơi. GA giúp tạo ra các thiết kế hoặc bản nhạc mới, sáng tạo dựa trên các hàm fitness đặc thù.
- **Mô hình tài chính**: GA được sử dụng trong tối ưu hóa danh mục đầu tư, giao dịch thuật toán, và quản lý rủi ro. Nó giúp phân bổ tài sản tối ưu để tối đa hóa lợi nhuận và giảm thiểu rủi ro, và có thể điều chỉnh các chiến lược giao dịch để thích ứng với điều kiện thị trường.

5. Cross-Entropy Method

5.1. Tổng quan

- Phương pháp Cross-Entropy (CE) là một kỹ thuật tối ưu hóa xác suất, thường được sử dụng để tìm chính sách tốt nhất hoặc lời giải tối ưu cho các bài toán phức tạp.
- Hoạt động bằng cách học một phân phối xác suất từ các mẫu tốt nhất và dần dần cập nhật phân phối này để tập trung vào các giải pháp tối ưu hơn qua các lần lặp.

$$\psi^* = \arg\max_{\psi} E_{\theta \sim p(\cdot|\psi)}[U(\theta)] = \arg\max_{\psi} \int U(\theta) \mathrm{p}(\theta \mid \psi) \mathrm{d}\theta$$

Tối ưu kỳ vọng (Expectation):

$$\psi^* = \arg\max_{\psi} E_{\theta \sim p(\cdot|\psi)}[U(\theta)]$$

- Mục tiêu là tìm giá trị ψ để **kỳ vọng** của hàm $\mathrm{U}(\theta)$, khi θ được lấy từ phân phối $\mathrm{p}(\theta \mid \psi)$, đạt giá trị lớn nhất.
- $\mathrm{p}(\theta \mid \psi)$,: Phân phối xác suất của các mẫu θ dựa trên tham số ψ.
- E: Kỳ vọng, biểu diễn giá trị trung bình của hàm $\mathrm{U}(\theta)$ qua tất cả các giá trị θ có thể.

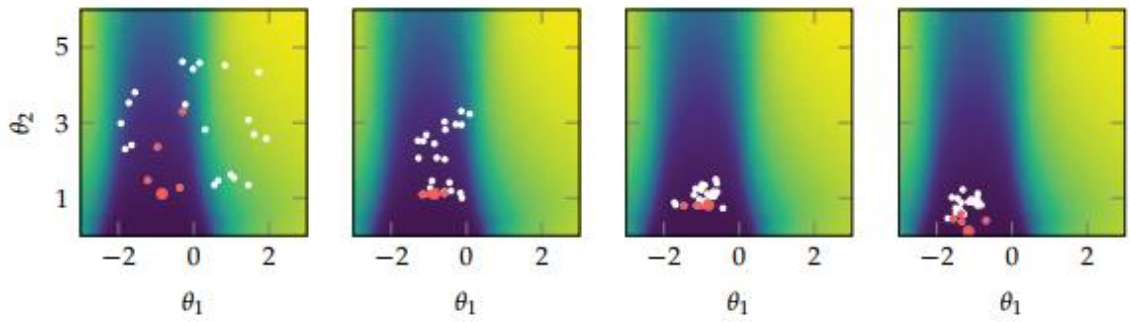
- **Dạng tích phân của kỳ vọng:**

$$\psi^* = \arg\max_{\psi} \int U(\theta) \mathrm{p}(\theta \mid \psi) \mathrm{d}\theta$$

  - $\int \mathrm{U}(\theta) \mathrm{p}(\theta \mid \psi) \mathrm{d}\theta$: Là kỳ vọng được viết dưới dạng tích phân, biểu diễn sự "đóng góp" của tất cả giá trị (θ) trong không gian xác suất.

- **Ý nghĩa thực tiễn:**
  - $\mathrm{U}(\theta)$: Hàm mục tiêu cần tối ưu hóa.

- ψ: Tham số của phân phối tìm kiếm $p(\theta \mid \psi)$.
- Mục tiêu: Tìm giá trị ψ^* để phân phối $p(\theta \mid \psi)$ tập trung nhiều nhất vào các giá trị θ có $U(\theta)$ cao (gần giá trị tối ưu).



5.2. Ưu điểm và hạn chế của phương pháp CE

Ưu điểm:

- Đơn giản và hiệu quả:
  - CE dễ triển khai vì chỉ yêu cầu lấy mẫu và cập nhật tham số, không phụ thuộc vào gradient hay mô hình phức tạp.
- Phù hợp với bài toán phức tạp:
  - Được dùng trong các bài toán tối ưu tổ hợp, tối ưu không khả vi, hoặc các không gian chính sách lớn.
- Khả năng khám phá toàn không gian:
  - Phân phối ban đầu ngẫu nhiên đảm bảo khám phá toàn bộ không gian giải pháp, trong khi cập nhật tập trung dần vào khu vực chứa lời giải tốt.

Hạn chế:

- Chi phí tính toán cao:
  - Cần sinh và đánh giá một lượng lớn mẫu tại mỗi lần lặp, đòi hỏi tài nguyên tính toán đáng kể.

- Nguy cơ hội tụ sớm:
    - Nếu nhóm mẫu tốt nhất (elite) không được chọn đúng cách hoặc không đủ đa dạng, phương pháp có thể hội tụ vào giải pháp kém.
- Phụ thuộc vào phân phối ban đầu:
    - Nếu phân phối khởi tạo không đủ rộng, phương pháp có thể bỏ qua các vùng không gian chứa giải pháp tối ưu.

5.3. Ứng dụng

Phương pháp CE được sử dụng rộng rãi trong nhiều lĩnh vực:

- Robot học: Tối ưu hóa chính sách điều khiển robot.
- Tối ưu tổ hợp: Bài toán TSP (Travelling Salesman Problem), lập lịch công việc.
- Học tăng cường (Reinforcement Learning): Tìm chính sách tốt nhất trong môi trường phức tạp.
- Thiết kế mạng nơ-ron: Cải thiện kiến trúc mô hình hoặc tham số huấn luyện.

5.5. Ví dụ minh họa

Xem θ là các giải pháp của một bài toán tối ưu (như trọng số của mô hình) và $U(\theta)$ đo hiệu suất của các giải pháp đó (ví dụ: độ chính xác của mô hình).

1. **Khởi tạo phân phối:**
    - Ban đầu, chọn một phân phối $p(\theta \mid \psi)$ (ví dụ: Gaussian) với tham số ban đầu ψ (trung bình và độ lệch chuẩn).
2. **Sinh mẫu:**
    - Sinh các giá trị θ từ phân phối $p(\theta \mid \psi)$.
3. **Đánh giá:**
    - Tính giá trị hàm $U(\theta)$ cho từng mẫu θ.

4. **Cập nhật ψ:**
    - Dựa vào các mẫu θ tốt nhất (elite samples), cập nhật ψ để phân phối $p(\theta \mid \psi)$ tập trung vào các giá trị θ có $U(\theta)$ cao hơn.
5. **Lặp lại:**
    - Quá trình lặp lại cho đến khi ψ hội tụ hoặc đạt hiệu suất mong muốn.

6. Evolution Strategies

6.1. Khái niệm

Evolution Strategies (ES) là một nhóm các thuật toán tối ưu hóa dựa trên các nguyên lý tiến hóa sinh học. ES thuộc về lĩnh vực học tăng cường tiến hóa (Evolutionary Reinforcement Learning) và được sử dụng rộng rãi trong các bài toán tối ưu hóa không khả vi, đa mục tiêu hoặc không có thông tin về gradient. ES tập trung vào việc tối ưu hóa các tham số của một hàm mục tiêu thông qua sự lặp đi lặp lại của các bước tiến hóa như đột biến (mutation), chọn lọc (selection), và tái tổ hợp (recombination).

Các thành phần chính của Evolution Strategies:

1. **Biểu diễn (Representation):**
    - Một tập hợp các cá thể (individuals) được biểu diễn dưới dạng vector trong không gian tham số.
2. **Đột biến (Mutation):**
    - Là bước thay đổi ngẫu nhiên các tham số của cá thể để tạo ra cá thể mới. Quá trình này thường dựa trên việc thêm nhiễu Gaussian vào vector tham số.
3. **Chọn lọc (Selection):**

- Lựa chọn các cá thể tốt nhất từ thế hệ hiện tại để giữ lại và tạo ra thế hệ mới. Các cá thể được đánh giá dựa trên **fitness function** (hàm đánh giá).

4. **Tái tổ hợp (Recombination - tùy chọn):**
    - Một số ES áp dụng tái tổ hợp để kết hợp thông tin từ nhiều cá thể (cha mẹ) để tạo ra cá thể mới.
5. **Cập nhật chiến lược (Strategy Update):**
    - Một đặc điểm nổi bật của ES là các tham số như độ lệch chuẩn (standard deviation) của phân phối Gaussian trong đột biến cũng được tối ưu hóa, giúp cải thiện khả năng tìm kiếm trong không gian tham số.

6.2. Cơ chế hoạt động

Khởi tạo quần thể ban đầu (Initialization)

- Một quần thể ban đầu gồm các cá thể được tạo ra ngẫu nhiên. Mỗi cá thể đại diện cho một **nghiệm tiềm năng** của bài toán tối ưu hóa.
- Khởi tạo các tham số giải pháp ban đầu, đặt vector trung bình cho các tham số giải pháp và độ lệch chuẩn

Đánh giá mức độ phù hợp (Fitness Evaluation)

- Mỗi cá thể trong quần thể được đánh giá dựa trên một **hàm mục tiêu (objective function)** hoặc **hàm fitness**.
- Giá trị fitness cho biết mức độ "tốt" của một cá thể trong việc giải quyết bài toán tối ưu hóa.

Lặp lại quá trình tiến hóa (Evolution Process)

Quá trình tiến hóa lặp lại nhiều lần để cải thiện chất lượng của quần thể. Các bước chính trong quá trình tiến hóa bao gồm:

a. Đột biến (Mutation)

- Đột biến là quá trình tạo ra các cá thể con (offspring) bằng cách thêm nhiễu ngẫu nhiên vào các cá thể cha mẹ (parents).
- Nhiễu này thường được lấy từ một phân phối xác suất, một phân phối tham số ban đầu, ví dụ như phân phối Gaussian. Phân phối này sẽ tạo ra các mẫu tham số θ.

b. Chọn lọc tự nhiên (Selection)

- Sau khi đột biến, các cá thể con sẽ được đánh giá dựa trên hàm fitness.
- Dựa vào giá trị fitness, các cá thể tốt nhất (có fitness cao hơn) sẽ được chọn để tiếp tục tham gia vào các thế hệ tiếp theo.

Hai chiến lược phổ biến trong chọn lọc:

1. **(μ, λ)-selection:**
    - Chỉ chọn các cá thể con (offspring) để tạo ra thế hệ mới.
    - μ: Số lượng cá thể cha mẹ.
    - λ: Số lượng cá thể con.
    - λ>μλ : Số lượng cá thể con thường lớn hơn cha mẹ để tăng tính đa dạng.
2. **(μ + λ)-selection:**
    - Chọn cả cá thể cha mẹ và con để tạo ra thế hệ mới.
    - Điều này giúp bảo tồn các cá thể tốt nhất từ thế hệ cũ.

c. Cập nhật phân phối tham số

- Gán trọng số dựa trên thứ hạng:

$$w(i) = \max\left(0, \log\left(\frac{m}{2} + 1\right) - \log(i)\right)$$

  Mẫu có thứ hạng cao sẽ có trọng số lớn hơn, ảnh hưởng nhiều hơn đến gradient. Mẫu kém sẽ ít hoặc không ảnh hưởng.

- Ước lượng gradient của kỳ vọng tiện ích:

$$\nabla_{\psi} \boldsymbol{E}_{\boldsymbol{\theta}\sim \boldsymbol{p}(\boldsymbol{\theta}|\boldsymbol{\psi})}[\mathbf{U}(\boldsymbol{\theta})] = \boldsymbol{E}_{\boldsymbol{\theta}\sim \boldsymbol{p}(\boldsymbol{\theta}|\boldsymbol{\psi})}[\mathbf{U}(\boldsymbol{\theta}) \nabla_{\psi} \log \boldsymbol{p}(\boldsymbol{\theta}|\boldsymbol{\psi})]$$

Công thức này tính toán cách thay đổi tham số ψ để tăng xác suất sinh ra các mẫu θi có tiện ích U(θi) cao hơn.

- Cập nhật tham số phân phối

$$\psi \leftarrow \psi + \alpha . \nabla_{\psi} \boldsymbol{E}_{\boldsymbol{\theta}\sim \boldsymbol{p}(\boldsymbol{\theta}|\boldsymbol{\psi})}[\mathbf{U}(\boldsymbol{\theta})]$$

Tham số ψ của phân phối được điều chỉnh để tập trung hơn vào khu vực chứa các mẫu θ có giá trị tiện ích cao

d. Dừng thuật toán (Termination)

- Quá trình tiến hóa dừng lại khi một trong các điều kiện sau được thỏa mãn:
    - Đạt được số thế hệ tối đa.
    - Sự cải thiện của fitness giữa các thế hệ nhỏ hơn một ngưỡng nhất định.
    - Fitness của một cá thể đạt đến giá trị mong muốn.

6.3. Ưu điểm và nhược điểm của Evolution Strategies

Ưu điểm:

- **Không cần thông tin gradient:** ES hoạt động tốt trên các hàm mục tiêu không khả vi, phi tuyến.
- **Tính song song:** Có thể đánh giá nhiều cá thể cùng lúc, phù hợp với các hệ thống tính toán song song.
- **Đa dạng hóa giải pháp:** Khám phá được nhiều khu vực trong không gian tìm kiếm.

Nhược điểm:

- **Hiệu quả thấp với không gian lớn:** Khi không gian tìm kiếm quá lớn, ES có thể đòi hỏi nhiều tính toán.
- **Đòi hỏi điều chỉnh tham số:** Việc lựa chọn σ, kích thước quần thể, và số thế hệ có thể yêu cầu thử nghiệm nhiều lần.

6.4. Ứng dụng của Evolution Strategies

- Tối ưu hóa các hàm phi tuyến trong kỹ thuật và khoa học.
- Tối ưu hóa các hệ thống AI và học máy (ví dụ: huấn luyện mạng nơ-ron không sử dụng gradient).
- Thiết kế tự động trong các lĩnh vực kỹ thuật, như tối ưu hóa hình dáng khí động học, thiết kế mạch điện tử.

7. Isotropic Evolutionary Strategies

7.1. Giới thiệu

Isotropic Evolutionary Strategies là một phương pháp tìm kiếm tối ưu trong không gian tham số, nơi giả định rằng phân phối tìm kiếm là một phân phối Gauss đồng dạng với ma trận hiệp phương sai có dạng $\sigma^2 I$. Phương pháp này được sử dụng rộng rãi trong các bài toán học máy, đặc biệt trong việc tối ưu hóa các tham số của mô hình, chẳng hạn như các tham số trong học sâu

7.2. Mô tả thuật toán

Thuật toán Isotropic Evolutionary Strategies (IES) hoạt động trên nguyên lý tìm kiếm trong không gian tham số theo cách tối ưu hóa hàm lợi ích của một mô hình học máy. Cách thức tối ưu hóa này không yêu cầu tính toán gradient của hàm mục tiêu mà thay vào đó sử dụng các mẫu ngẫu nhiên để ước lượng gradient.

Công thức:

Giả sử phân phối tìm kiếm $\theta \sim N(\psi, \sigma^2 I)$, trong đó ψ là giá trị trung bình và σ^2 là phương sai. Hàm lợi ích $U(\theta)$ được tính như sau:

$$E_{\theta \sim N(\psi,\sigma^2 I)}[U(\theta)] = E_{\epsilon \sim N(0,I)}[U(\psi + \sigma\epsilon)]$$

Trong đó:

$\theta = \psi + \sigma\epsilon$ *với* $\epsilon \sim N(0, I)$, nghĩa là ϵ là một vector ngẫu nhiên với phân phối chuẩn chuẩn tắc.

ψ là giá trị trung bình của phân phối tham số

σ là độ lệch chuẩn, điều chỉnh mức độ lan rộng của các mẫu

Công thức tính Gradients cho hàm lợi ích:

$$\nabla_{\psi} E_{\theta \sim N(\psi,\sigma^2 I)}[U(\theta)] = \frac{1}{\sigma} E_{\epsilon \sim N(\mathbb{0},I)}[U(\psi + \sigma\epsilon)\epsilon]$$

Công thức này cho thấy rằng gradient phụ thuộc vào các mẫu ϵ được lấy từ phân phối chuẩn chuẩn tắc, và ta cần tính toán hiệu quả của hàm lợi ích với mỗi mẫu, sau đó nhân với ϵ để ước lượng gradient.

```
struct IsotropicEvolutionStrategies
    ψ       # initial mean
    σ       # initial standard deviation
    m       # number of samples
    α       # step factor
    k_max   # number of iterations
end

function optimize_dist(M::IsotropicEvolutionStrategies, π, U)
    ψ, σ, m, α, k_max = M.ψ, M.σ, M.m, M.α, M.k_max
    n = length(ψ)
    ws = evolution_strategy_weights(2*div(m,2))
    for k in 1:k_max
        ϵs = [randn(n) for i in 1:div(m,2)]
        append!(ϵs, -ϵs) # weight mirroring
        us = [U(π, ψ + σ.*ϵ) for ϵ in ϵs]
        sp = sortperm(us, rev=true)
        ∇ = sum(w.*ϵs[i] for (w,i) in zip(ws,sp)) / σ
        ψ += α.*∇
    end
    return MvNormal(ψ, σ)
end
```

Giải thích mã giả:

- ψ là trung bình ban đầu của phân phối, thường được khởi tạo ngẫu nhiên hoặc có ý nghĩa gì đó trong bối cảnh cụ thể.
- σ là độ lệch chuẩn của phân phối, điều chỉnh phạm vi thay đổi của các tham số trong quá trình tối ưu hóa.
- m là số lượng mẫu ngẫu nhiên được sinh ra trong mỗi vòng lặp.
- α là hệ số bước, ảnh hưởng đến tốc độ cập nhật tham số.

- k_max là số vòng lặp tối đa, tức là số lần mà quá trình tối ưu hóa sẽ lặp lại.
- n = length(ψ) là số chiều của tham số ψ (số lượng tham số trong mô hình)
- ws = evolution_strategy_weights(2*div(m,2)): Tạo ra một vector trọng số cho các mẫu ngẫu nhiên. Trọng số này sẽ được sử dụng để ưu tiên các mẫu có hiệu suất cao hơn trong quá trình cập nhật tham số.
- Vòng lặp bắt đầu từ k = 1 đến k_max, thực hiện tối ưu hóa trong các vòng lặp này.
- ϵs là danh sách các mẫu ngẫu nhiên, được tạo ra bằng cách sinh ngẫu nhiên các vector có chiều dài n (số chiều của tham số ψ) từ phân phối chuẩn (randn(n)).
- append!(ϵs, -ϵs) thực hiện phản chiếu trọng số: Tạo ra các mẫu ngược dấu so với các mẫu ban đầu để làm tăng tính đa dạng trong quá trình tối ưu hóa.
- us là một danh sách chứa hiệu suất (utility) của các mẫu ψ + σ * ϵ, tức là các tham số được thay đổi theo từng mẫu ngẫu nhiên. Hàm U(π, ψ + σ.*ϵ) tính toán hiệu suất của chính sách với các tham số đã thay đổi.
- sp = sortperm(us, rev=true) sắp xếp các giá trị hiệu suất us theo thứ tự giảm dần, tức là các mẫu có hiệu suất cao nhất được sắp xếp lên đầu.
- ∇ là gradient ước lượng: Tính toán gradient bằng cách tính tổng của các mẫu ϵs trọng số theo các trọng số ws (được tính từ trước) và sắp xếp theo thứ tự hiệu suất.
- ψ += α.*∇ cập nhật tham số ψ bằng cách cộng vào gradient đã được tính toán, nhân với hệ số bước α để điều chỉnh tốc độ cập nhật.
- Cuối cùng, hàm trả về một phân phối chuẩn mới (MvNormal(ψ, σ)), với trung bình là ψ và độ lệch chuẩn là σ. Đây là phân phối mô tả các tham số tối ưu sau quá trình tối ưu hóa.

7.3. Ý nghĩa và ứng dụng thực tế

Isotropic Evolutionary Strategies có thể được áp dụng trong các bài toán tối ưu hóa, đặc biệt là trong việc tối ưu hóa các mô hình học máy mà không cần phải tính toán gradient phức tạp. Trong học sâu, các chiến lược này có thể được sử dụng để tối ưu hóa các tham số của mạng neuron mà không cần sử dụng phương pháp lan truyền ngược.

Ứng dụng:

Tối ưu hóa tham số trong học máy: Thuật toán này có thể được sử dụng để tối ưu hóa tham số trong các bài toán học máy như học sâu (deep learning), tăng cường học (reinforcement learning).

Tìm kiếm chính sách trong học máy tăng cường: Trong các bài toán học máy tăng cường, thuật toán này có thể được sử dụng để tìm kiếm các chính sách tối ưu cho tác vụ cụ thể.

8. Chương trình minh hoạ

8.1. Approximate Policy Evaluation

VÍ DỤ GRIDWORLD:

- Một grid 5×5 với trạng thái bắt đầu ở góc trên trái (0,0) và mục tiêu ở góc dưới phải (4,4)
- Phần thưởng là 10 khi đến mục tiêu (4,4) các trạng thái khác không có phần thưởng.
- Agent di chuyển theo chính sách ngẫu nhiên (random policy).

=> Tính trung bình giá trị kì vọng của chính sách trên 1000 tập.

❖ **Cách cài đặt Monte Carlo Policy**

```python
# Monte Carlo Policy Evaluation
def monte_carlo_policy_evaluation(env, policy, num_episodes=1000, max_steps=50):
    total_rewards = []

    for _ in range(num_episodes):
        state = env.reset()
        trajectory = []
        total_reward = 0
        discount = 1

        # Generate a trajectory
        for _ in range(max_steps):
            action = policy(state)
            next_state, reward, done = env.step(state, action)
            trajectory.append((state, action, reward))
            total_reward += discount * reward
            discount *= env.gamma
            state = next_state
            if done:
                break

        total_rewards.append(total_reward)

    # Estimate U(π) as the average return across all episodes
    U_pi = np.mean(total_rewards)
    return U_pi
```

❖ **Thiết lập grid**

```python
# Initialize the environment
grid_size = 5
start_state = (0, 0)
end_state = (4, 4)
rewards = {(4, 4): 10}  # Reward of 10 for reaching the goal state
gamma = 0.9

env = GridWorld(grid_size, start_state, end_state, rewards, gamma)

# Perform Monte Carlo Policy Evaluation
U_pi = monte_carlo_policy_evaluation(env, random_policy, num_episodes=1000, max_steps=50)
```

❖ **Thiết lập policy**

```
    def step(self, state, action):
        # Possible actions: 'up', 'down', 'left', 'right'
        x, y = state
        if action == 'up':
            next_state = (max(x - 1, 0), y)
        elif action == 'down':
            next_state = (min(x + 1, self.grid_size - 1), y)
        elif action == 'left':
            next_state = (x, max(y - 1, 0))
        elif action == 'right':
            next_state = (x, min(y + 1, self.grid_size - 1))
        else:
            next_state = state  # Invalid action: no movement

        reward = self.rewards.get(next_state, 0)  # Get reward for the next state
        done = next_state == self.end_state
        return next_state, reward, done

    def reset(self):
        return self.start_state

# Define a random policy
def random_policy(state):
    return random.choice(['up', 'down', 'left', 'right'])
```

❖ **Kết quả:**

```
Estimated value of the policy (U(π)): 0.24
```

8.2. Local Search

❖ **Cách cài đặt hooke_jeeves**

```python
def hooke_jeeves(theta_init, alpha_init, c, epsilon, max_iter=1000):
    theta = np.array(theta_init, dtype=float)
    alpha = alpha_init
    n = len(theta)
    u = objective_function(theta)
    iter_count = 0

    while alpha > epsilon and iter_count < max_iter:
        iter_count += 1
        best_u = u
        best_move = None

        # Thăm dò
        for i in range(n):
            for sgn in [-1, 1]:
                theta_new = theta.copy()
                theta_new[i] += sgn * alpha
                u_new = objective_function(theta_new)
                if u_new > best_u:
                    best_u = u_new
                    best_move = theta_new.copy()

        # Cập nhật
        if best_move is not None:
            theta = best_move.copy()
            u = best_u
        else:
            alpha *= c

        # In thông tin mỗi bước (tùy chọn)
        print(f"Iteration {iter_count}: theta = {theta}, alpha = {alpha}, u = {u}")

    return theta
```

❖ **Thiết lập các tham số ban đầu:**

```python
theta_init = [1, 1]   # Điểm khởi đầu
alpha_init = 0.5      # Kích thước bước ban đầu
c = 0.5               # Hệ số giảm kích thước bước
epsilon = 0.01        # Ngưỡng dừng
```

❖ **Kết quả:**

```
    optimal_theta = hooke_jeeves(theta_init, alpha_init, c, epsilon)
```

```
Iteration 1: theta = [0.5 1. ], alpha = 0.5, u = -1.25
Iteration 2: theta = [0.5 0.5], alpha = 0.5, u = -0.5
Iteration 3: theta = [0.  0.5], alpha = 0.5, u = -0.25
Iteration 4: theta = [0. 0.], alpha = 0.5, u = -0.0
Iteration 5: theta = [0. 0.], alpha = 0.25, u = -0.0
Iteration 6: theta = [0. 0.], alpha = 0.125, u = -0.0
Iteration 7: theta = [0. 0.], alpha = 0.0625, u = -0.0
Iteration 8: theta = [0. 0.], alpha = 0.03125, u = -0.0
Iteration 9: theta = [0. 0.], alpha = 0.015625, u = -0.0
Iteration 10: theta = [0. 0.], alpha = 0.0078125, u = -0.0
```

```
    print(f"\nGiá trị tối ưu tìm được: theta = {optimal_theta}")
    print(f"Giá trị hàm mục tiêu tại theta tối ưu: u = {objective_function(optimal_theta)}")
```

```
Giá trị tối ưu tìm được: theta = [0. 0.]
Giá trị hàm mục tiêu tại theta tối ưu: u = -0.0
```

8.3. Genetic Algorithm

Bài toán người bán hàng (TSP)

Mô tả bài toán: Tìm đường đi ngắn nhất để đi qua tất cả các thành phố

Dữ liệu bài toán: Tọa độ (x,y) của 16 thành phố mà người bán hàng cần đi qua

- **Dataset**

```
# Tọa độ các thành phố
coordinates = [
    (-100.0, -50.0), (150.0, -40.0),
    (-150.0, -60.0), (-50.0, -30.0),
    (-200.0, 220.0), (130.0, 100.0),
    (-250.0, 40.0), (-250.0, -170.0),
    (400.0, 370.0), (500.0, 670.0),
    (-80.0, 20.0), (-580.0, 600.0),
    (-520.0, 650.0), (300.0, 600.0),
    (-10.0, -80.0), (70.0, -20.0),
]
```

- **Cài đặt thuật toán Genetic Algorithm**

```python
class GeneticAlgorithm:
    def __init__(self, coords: List[Tuple[float, float]], pop_size: int = 100, mutation_rate: float = 0.01):
        self.coords = coords
        self.pop_size = pop_size
        self.mutation_rate = mutation_rate
        self.n_cities = len(coords)
        self.population = self.initialize_population()

        # Lưu lịch sử fitness
        self.best_fitness_history = []
        #self.avg_fitness_history = []
```

❖ Khởi tạo quần thể và hàm thích nghi

```python
    def initialize_population(self) -> List[List[int]]:
        """Khởi tạo quần thể ngẫu nhiên"""
        return [random.sample(range(self.n_cities), self.n_cities) for _ in range(self.pop_size)]

    def fitness(self, route: List[int]) -> float:
        """Tính tổng khoảng cách của một lộ trình"""
        return sum(
            np.sqrt((self.coords[route[i]][0] - self.coords[route[(i+1) % self.n_cities]][0]) ** 2 +
                    (self.coords[route[i]][1] - self.coords[route[(i+1) % self.n_cities]][1]) ** 2)
            for i in range(self.n_cities)
        )
```

❖ Hàm chọn lọc và hàm lai ghép

```python
    def select_parents(self) -> List[List[int]]:
        """Chọn cha mẹ dựa trên tournament selection"""
        parents = []
        for _ in range(self.pop_size):
            tournament = random.sample(self.population, 5)
            best = min(tournament, key=self.fitness)
            parents.append(best)
        return parents

    def crossover(self, parent1: List[int], parent2: List[int]) -> List[int]:
        """Lai ghép (crossover) giữa hai cá thể cha mẹ"""
        start, end = sorted(random.sample(range(self.n_cities), 2))
        child = parent1[start:end+1]
        child += [city for city in parent2 if city not in child]
        return child
```

❖ Hàm đột biến và hàm tiến hóa

```python
def mutate(self, route: List[int]) -> List[int]:
    """Đột biến (mutation) bằng cách hoán đổi hai thành phố"""
    if random.random() < self.mutation_rate:
        i, j = random.sample(range(self.n_cities), 2)
        route[i], route[j] = route[j], route[i]
    return route

def evolve(self):
    """Tiến hóa qua một thế hệ"""
    parents = self.select_parents()
    new_population = []
    for i in range(0, len(parents), 2):
        parent1, parent2 = parents[i], parents[(i + 1) % len(parents)]
        child = self.crossover(parent1, parent2)
        child = self.mutate(child)
        new_population.append(child)
    self.population = new_population

    # Lưu lịch sử fitness
    population_fitness = [self.fitness(route) for route in self.population]
    self.best_fitness_history.append(min(population_fitness))
    #self.avg_fitness_history.append(np.mean(population_fitness))
```

❖ Hàm khởi chạy thuật toán và điều kiện dừng

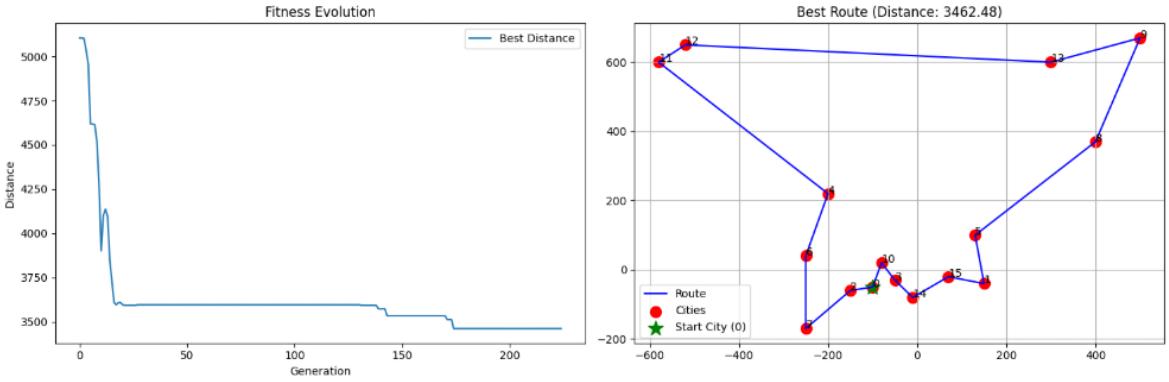
```python
def run(self, generations: int = 500, tolerance: float = 1e-3, patience: int = 50) -> Tuple[List[int], float]:
    """Chạy thuật toán qua số thế hệ, thêm điều kiện dừng sớm"""
    best_route = None
    best_distance = float('inf')
    no_improvement_generations = 0  # Đếm số thế hệ không có cải thiện

    for generation in range(generations):
        self.evolve()
        current_best = min(self.population, key=self.fitness)
        current_distance = self.fitness(current_best)

        # Hiển thị kết quả mỗi thế hệ
        print(f"Generation {generation + 1}: Best Distance = {current_distance:.2f}")

        # Kiểm tra cải thiện
        if abs(best_distance - current_distance) <= tolerance:
            no_improvement_generations += 1
        else:
            no_improvement_generations = 0

        # Cập nhật kết quả tốt nhất
        if current_distance < best_distance:
            best_distance = current_distance
            best_route = current_best

        # Điều kiện dừng sớm
        if no_improvement_generations >= patience:
            print(f"Stopping early at generation {generation + 1} due to no improvement for {patience} generations.")
            break

    return best_route, best_distance
```

❖ Trực quan hóa kết quả và khởi chạy thuật toán

```python
    def visualize_results(self, best_route: List[int], best_distance: float):
        """Hiển thị kết quả"""
        plt.figure(figsize=(15, 5))

        # Biểu đồ tiến hóa fitness
        plt.subplot(1, 2, 1)
        plt.plot(self.best_fitness_history, label='Best Distance')
        #plt.plot(self.avg_fitness_history, label='Average Distance')
        plt.title('Fitness Evolution')
        plt.xlabel('Generation')
        plt.ylabel('Distance')
        plt.legend()

        # Biểu đồ lộ trình tốt nhất
        plt.subplot(1, 2, 2)
        x_coords = [self.coords[i][0] for i in best_route] + [self.coords[best_route[0]][0]]
        y_coords = [self.coords[i][1] for i in best_route] + [self.coords[best_route[0]][1]]

        plt.plot(x_coords, y_coords, 'b-', label="Route")
        plt.scatter([c[0] for c in self.coords], [c[1] for c in self.coords], c='red', s=100, label="Cities")
        for i, (x, y) in enumerate(self.coords):
            plt.text(x, y, str(i), fontsize=10, color='black')

        plt.scatter(self.coords[0][0], self.coords[0][1], c='green', s=200, label="Start City (0)", marker='*')
        plt.title(f'Best Route (Distance: {best_distance:.2f})')
        plt.legend()
        plt.grid(True)

        plt.tight_layout()
        plt.show()

# Chạy thuật toán
ga = GeneticAlgorithm(coordinates, pop_size=100, mutation_rate=0.01)
best_route, best_distance = ga.run(generations=350)

# Hiển thị kết quả
print(f"Best Route: {best_route}")
print(f"Total Distance: {best_distance:.2f}")
ga.visualize_results(best_route, best_distance)
```

❖ Kết quả chạy thuật toán

```
Stopping early at generation 225 due to no improvement for 50 generations.
Best Route: [8, 9, 13, 12, 11, 4, 6, 7, 2, 0, 10, 3, 14, 15, 1, 5]
Total Distance: 3462.48
```



8.4. Cross-Entropy Method

Hàm mục tiêu: f(x) = sin(5x) · (1 − tanh(x^2))

- Mean ban đầu: Sinh ngẫu nhiên trong khoảng giới hạn [−2,2]
- Std Dev ban đầu: Được đặt là (upper−lower) / 4

- Số lượng mẫu mỗi vòng: n(samples) = 100.
- Phần trăm mẫu tốt nhất: 20% (tức chọn 20 mẫu trong số 100 mẫu tốt nhất).
- Điều kiện hội tụ: $\epsilon = 0.001$

Kiểm tra hội tụ: Nếu $|\Delta\mu| < 0.001$ và $|\Delta\sigma| < 0.001$ dừng lại

❖ **Cách cài đặt Cross-Entropy Method**

```python
def optimize(CEM, obj_func, n_iterations=50):
    lower, upper = CEM.bounds
    mean, std_dev = CEM.mean, CEM.std_dev

    for iteration in range(n_iterations):
        samples = np.random.normal(mean, std_dev, CEM.n_samples)
        samples = np.clip(samples, lower, upper)

        scores = obj_func(samples)

        #Chọn các mẫu tốt nhất (elite samples)
        n_elite = int(CEM.elite_frac * CEM.n_samples)
        elite_indices = np.argsort(scores)[-n_elite:]
        elite_samples = samples[elite_indices]

        # Cập nhật tham số phân phối Gaussian
        mean_new = np.mean(elite_samples)
        std_dev_new = np.std(elite_samples)

        #Kiểm tra điều kiện hội tụ
        if np.abs(mean - mean_new) < CEM.epsilon and np.abs(std_dev - std_dev_new) < CEM.epsilon:
            print(f"Converged at iteration {iteration }")
            break

        mean, std_dev = mean_new, std_dev_new

        print(f"Iteration {iteration + 1}: Mean = {mean:.3f}, Std Dev = {std_dev:.3f}")


    return mean
```

❖ **Hàm mục tiêu và các tham số đầu vào**

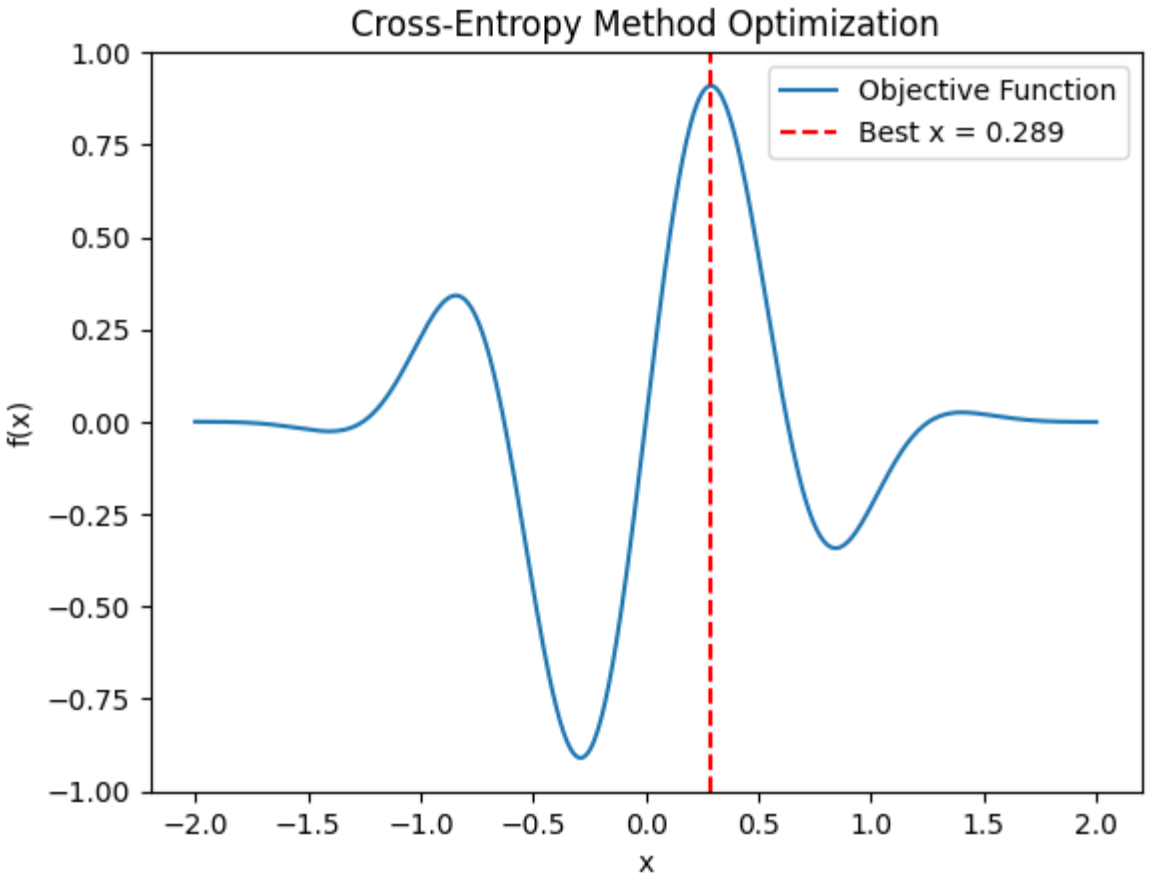
```python
def objective_function(x):
    return np.sin(5 * x) * (1 - np.tanh(x**2))


bounds = (-2, 2)
initial_mean = np.random.uniform(bounds[0], bounds[1])
initial_std_dev = (bounds[1] - bounds[0]) / 4
cem = CrossEntropyMethod(
    mean=initial_mean,
    std_dev=initial_std_dev,
    bounds=bounds,
    n_samples=100,
    elite_frac=0.2,
    epsilon=1e-3
)
```

❖ **Nhận các đầu vào**

```python
class CrossEntropyMethod:
    def __init__(self, mean, std_dev, bounds, n_samples, elite_frac, epsilon):
        self.mean = mean
        self.std_dev = std_dev
        self.bounds = bounds
        self.n_samples = n_samples
        self.elite_frac = elite_frac
        self.epsilon = epsilon
```

❖ **Kết quả:**

```
Iteration 1: Mean = 1.219, Std Dev = 0.491
Iteration 2: Mean = 0.651, Std Dev = 0.452
Iteration 3: Mean = 0.354, Std Dev = 0.135
Iteration 4: Mean = 0.291, Std Dev = 0.029
Iteration 5: Mean = 0.291, Std Dev = 0.008
Iteration 6: Mean = 0.289, Std Dev = 0.001
Converged at iteration 6
```



8.5. Evolution Strategies

Bài toán tối ưu hóa thêm số model SVM

Mô tả bài toán: Tối ưu hóa tham số để model SVM có thể dự đoán chính xác nhất

Dữ liệu bài toán: Dữ liệu giả lập make_classification trong scikit-learn

❖ **Khởi tạo các giá trị trung bình, phương sai**

```
def evolution_strategy_theoretical(objective_function, dim, population_size, sigma, alpha, max_iters, early_stop_threshold=None):
    μ_mean = np.random.uniform(0.1, 10, dim)  # Khởi tạo giá trị trung bình
    μ_std = np.full(dim, sigma)  # Khởi tạo phương sai
    best_score = -np.inf
    best_params = None
    history_scores = []  # Lưu lại các giá trị độ chính xác qua từng vòng lặp
    history_params = []  # Lưu lại các giá trị tham số tốt nhất qua từng vòng lặp
```

❖ **Thiết lập hàm đánh giá hiệu năng**

```python
def evaluate_hyperparameters(params):
    C, gamma = params  # Các siêu tham số
    if C <= 0 or gamma <= 0:  # Ràng buộc không hợp lệ
        return -np.inf  # Trả về giá trị rất thấp
    model = SVC(C=C, gamma=gamma)
    scores = cross_val_score(model, X, y, cv=5, scoring="accuracy")
    return scores.mean()  # Trả về độ chính xác trung bình
```

❖ **Sinh mẫu từ phân phối Gaussian, xếp hàng và gán trọng số**

```python
for iteration in range(max_iters):
    # Sinh mẫu từ phân phối Gaussian
    population = np.random.normal(ψ_mean, ψ_std, (population_size, dim))
    scores = np.array([objective_function(ind) for ind in population])

    # Xếp hạng và gán trọng số
    ranks = np.argsort(scores)[::-1]
    weights = np.maximum(0, np.log(population_size / 2 + 1) - np.log(np.arange(1, population_size + 1)))
    weights /= np.sum(weights)
    weights -= 1 / population_size
```

❖ **Tính gradient log lil-hood, cập nhập giá trị trung bình và cập nhật giá trị phương sai**

```python
# Tính gradient log-likelihood và cập nhật mean
gradients = np.zeros(dim)
for i in range(population_size):
    gradients += weights[i] * ((population[i] - ψ_mean) / ψ_std**2)
ψ_mean += alpha * gradients

# Cập nhật std để giữ phân phối ổn định
ψ_std = np.maximum(0.01, ψ_std + alpha * np.std(population, axis=0))

# Lưu lại giá trị độ chính xác và tham số tốt nhất
history_scores.append(scores[ranks[0]])
history_params.append(population[ranks[0]])
```

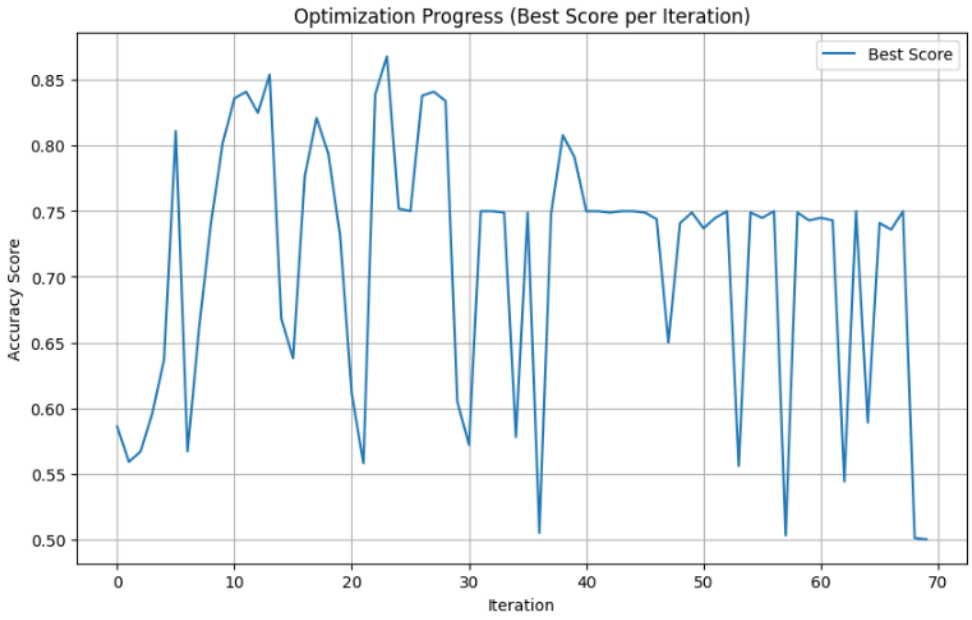
❖ **Thêm điều kiện dừng và trực quan hóa qua các vòng lặp**

```
        # Kiểm tra điều kiện dừng sớm
        if scores[ranks[0]] > best_score:
            best_score = scores[ranks[0]]
            best_params = population[ranks[0]]
        if early_stop_threshold and best_score >= early_stop_threshold:
            print(f"Early stopping at iteration {iteration + 1}")
            break

        # Logging
        print(f"Iteration {iteration + 1}, Best Score: {best_score}, Best Params: {best_params}")

    # Vẽ biểu đồ độ chính xác qua các vòng lặp
    plt.figure(figsize=(10, 6))
    plt.plot(history_scores, label="Best Score")
    plt.xlabel("Iteration")
    plt.ylabel("Accuracy Score")
    plt.title("Optimization Progress (Best Score per Iteration)")
    plt.grid(True)
    plt.legend()
    plt.show()

    return best_params, best_score
```

❖ Cấu hình và khởi chạy kết quả

```
# Tạo dataset mẫu
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

# Cấu hình Evolution Strategies
dim = 2  # Tối ưu hóa 2 siêu tham số: C và gamma
population_size = 20
sigma = 0.5
alpha = 0.1
max_iters = 70
early_stop_threshold = 0.95  # Dừng sớm nếu đạt độ chính xác >= 95%

# Tối ưu hóa
best_params, best_score = evolution_strategy_theoretical(
    objective_function=evaluate_hyperparameters,
    dim=dim,
    population_size=population_size,
    sigma=sigma,
    alpha=alpha,
    max_iters=max_iters,
    early_stop_threshold=early_stop_threshold
)

print(f"Best Parameters: {best_params}, Best Score: {best_score}")
```

❖ Kết quả




```
Best Parameters: [1.36566550e+01 5.44019582e-03], Best Score: 0.868
```

8.6. Isotropic Evolutionary Strategies

Mở một cửa hàng trong thành phố sao cho vị trí của nó tối ưu hóa khoảng cách trung bình đến tất cả các khách hàng tiềm năng.

- Các tọa độ của khách hàng (xi, yi) đã được biết.
- Với điểm bắt đầu là [0,0]

⇨ **Tìm vị trí tối ưu của cửa hàng (dạng toạ độ [x, y]).**

❖ **Cài đặt Isotropic Evolutionary Strategies**

```python
#Hàm gradient
def compute_gradient(psi, sigma, m):

    n = len(psi)
    gradients = np.zeros(n)

    epsilon_samples = [np.random.randn(n) for _ in range(m)]

    for epsilon in epsilon_samples:
        sample_positive = psi + sigma * epsilon
        sample_negative = psi - sigma * epsilon

        u_positive = objective(sample_positive)
        u_negative = objective(sample_negative)

        gradients += (u_positive - u_negative) * epsilon

    gradients /= (2 * m * sigma)

    return gradients

#Hàm tối ưu sử dụng Evolutionary Strategies với Gradient
def optimize_with_gradient(initial_position, sigma, alpha, k_max, m):

    psi = np.array(initial_position)

    for k in range(k_max):
        gradient = compute_gradient(psi, sigma, m)

        # Cập nhật psi dựa trên gradient
        psi += alpha * gradient

        current_value = -objective(psi)
        print(f"Iteration {k+1}: Position = {psi}, Objective value = {current_value:.2f}")

    return psi
```

❖ **Dataset và hàm mục tiêu:**

```python
#Tọa độ khách hàng
customer_locations = np.array([
    [2, 3],
    [5, 1],
    [6, 8],
    [7, 7],
    [3, 6],
    [8, 3],
    [9, 9],
    [4, 5],
    [2, 8],
    [1, 1]
])

#Hàm mục tiêu (tối thiểu hóa khoảng cách trung bình)
def objective(position):

    x, y = position
    distances = np.sqrt((customer_locations[:, 0] - x)**2 + (customer_locations[:, 1] - y)**2)
    return -np.mean(distances)
```

❖ **Khởi tạo vị trí ban đầu và các tham số, điều kiện.**

```python
initial_position = [0.0, 0.0]  # Điểm bắt đầu
sigma = 1.0  # Độ lệch chuẩn
alpha = 0.5  # Hệ số bước
k_max = 50  # Số vòng lặp tối đa
m = 20  # Số mẫu cho gradient
```

❖ **Kết quả**

```
Iteration 1: Position = [0.11262334 0.49978975], Objective value = 6.85
Iteration 2: Position = [0.48819172 0.75661627], Objective value = 6.43
Iteration 3: Position = [0.77764826 1.11755372], Objective value = 6.02
Iteration 4: Position = [0.89134323 1.57517046], Objective value = 5.72
Iteration 5: Position = [1.14491348 1.67060358], Objective value = 5.53
Iteration 6: Position = [1.19396145 1.80305842], Objective value = 5.44
Iteration 7: Position = [1.50569045 1.88255076], Objective value = 5.24
Iteration 8: Position = [1.64467932 1.97700268], Objective value = 5.12
Iteration 9: Position = [2.02128902 2.16448119], Objective value = 4.86
Iteration 10: Position = [2.33040629 2.55183855], Objective value = 4.56
Iteration 11: Position = [2.73246389 2.92153356], Objective value = 4.29
Iteration 12: Position = [3.00046415 3.20914165], Objective value = 4.12
Iteration 13: Position = [3.25393987 3.35259165], Objective value = 4.02
Iteration 14: Position = [3.42897408 3.43132344], Objective value = 3.96
Iteration 15: Position = [3.53797682 3.68530448], Objective value = 3.87
Iteration 16: Position = [3.63301289 3.84853496], Objective value = 3.81
Iteration 17: Position = [3.82639996 4.06882657], Objective value = 3.72
Iteration 18: Position = [3.83798694 4.14953881], Objective value = 3.70
Iteration 19: Position = [3.87690636 4.26660712], Objective value = 3.67
Iteration 20: Position = [3.90673507 4.37500591], Objective value = 3.64
Iteration 21: Position = [3.94205673 4.41446668], Objective value = 3.63
Iteration 22: Position = [3.94309069 4.46007766], Objective value = 3.62
Iteration 23: Position = [4.0136405  4.52941175], Objective value = 3.59
Iteration 24: Position = [4.0445322 4.5691042], Objective value = 3.58
Iteration 25: Position = [4.04386532 4.62421532], Objective value = 3.57
...
Iteration 48: Position = [4.41264832 5.15459814], Objective value = 3.52
Iteration 49: Position = [4.41531432 5.16369506], Objective value = 3.52
Iteration 50: Position = [4.43116283 5.18666215], Objective value = 3.52
Optimal store location: [4.43116283 5.18666215]
```

9. Kết luận

9.1. Ưu điểm

Ưu điểm: Phù hợp với các bài toán phức tạp, không gian trạng thái lớn, chính sách ngẫu nhiên, và không yêu cầu mô hình động lực học chính xác.

9.2. Hạn chế

Chi phí tính toán cao, dễ mắc kẹt trong cực tiểu cục bộ, và yêu cầu thiết kế chính sách tham số hóa và hàm mục tiêu tốt.

10. Hướng phát triển

- Tăng cường hiệu quả tính toán

  Mục tiêu: Giảm chi phí tính toán và tăng tốc độ hội tụ trong các bài toán lớn và phức tạp.

- Ứng dụng vào lĩnh vực mới

  Mục tiêu: Mở rộng ứng dụng của Policy Search vào các lĩnh vực như y tế, tài chính, tự động hóa, và năng lượng.

- Ví dụ:

  Y tế: Hỗ trợ ra quyết định trong điều trị bệnh nhân hoặc quản lý tài nguyên.

  Tài chính: Phát triển các chiến lược giao dịch tự động và quản lý rủi ro.

  Năng lượng: Tối ưu hóa việc sử dụng năng lượng trong các hệ thống phân phối.

11. Bảng phân công công việc của các thành viên trong nhóm:

STT	Họ và tên	Công việc	Hoàn thành công việc
1	Bùi Văn Thái	Tìm hiểu về Approximate Policy Evaluation	80%
2	Chế Duy Khang	Tìm hiểu về Cross-Entropy Method, demo và demo về Isotropic Evolutionary Strategies	100%
3	Trương Vĩnh Thuận	Tìm hiểu về Local Search, demo và lý thuyết về	100%

		Isotropic Evolutionary Strategies	
4	Nguyễn Thế Vinh	Tìm hiểu về Genetic Algorithm, demo và tìm hiểu về Evolution Strategies, demo	100%

12. Bảng đánh giá chéo các thành viên trong nhóm (thang điểm 10)

	Vinh	Khang	Thái	Thuận
Vinh	X	10	10	10
Khang	10	X	10	10
Thái	7	10	X	10
Thuận	10	10	10	X

13. NGUỒN THAM KHẢO

1. GeeksforGeeks. (2024, March 8). *Genetic algorithms*. GeeksforGeeks. https://www.geeksforgeeks.org/genetic-algorithms/
2. Nhs. (2018, September 15). *Tóm tắt giải thuật di truyền*. Nhs000 Blog. https://nhs000.github.io/posts/tech/2018_09_16-genetic-algorithm/#headline-14
3. Kanade, V. (2023, September 6). *Genetic Algorithms - meaning, working, and applications*. Spiceworks Inc. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-are-genetic-algorithms/#:~:text=The%20genetic%20algorithm%20begins%20with,selection%2C%20crossover%2C%20and%20mutation.
4. Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine Learning*, *3*(2/3), 95–99. https://doi.org/10.1023/a:1022602019183
5. Weng, L. (2019, September 5). Evolution strategies. Lil'Log. https://lilianweng.github.io/posts/2019-09-05-evolution-strategies/