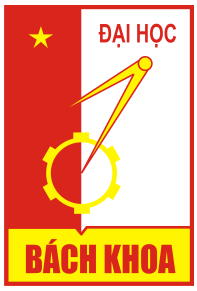
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN TOÁN ỨNG DỤNG VÀ TIN HỌC

—————————o0o—————————



ỨNG DỤNG DEEP REINFORCEMENT LEARNING GIẢI BÀI TOÁN NGƯỜI DU LỊCH ĐA MỤC TIÊU

ĐỒ ÁN I

Chuyên ngành: TOÁN TIN

Giảng viên hướng dẫn: **TS. TRẦN NGỌC THĂNG**
Bộ môn: TOÁN TIN
Sinh viên thực hiện: **NGUYỄN ANH MINH**
Lớp: CTTN Toán Tin - K64

Chữ kí của GVHD

Hà Nội, 7/2022

Nhận xét của giảng viên	Điểm

Hà Nội, ngày ... tháng ... năm 2022

Giảng viên hướng dẫn

TS. Trần Ngọc Thăng

Lời mở đầu

Tối ưu tổ hợp là một lớp bài toán trong lĩnh vực khoa học máy tính với nhiều ứng dụng cụ thể trong đời sống như những bài toán tìm đường đi ngắn nhất, bài toán Knapsack,... Một ví dụ điển hình cho các bài toán tối ưu tổ hợp đó là bài toán Traveling Saleman (TSP). Việc tìm kiếm lời giải tối ưu cho bài toán TSP là bài toán thuộc lớp $NP - hard$ [12], kể cả trong trường hợp trọng số giữa các đỉnh trong đồ thị được tính bằng khoảng cách Euclide.

Hiện nay có rất nhiều giải thuật từ giải thuật chính xác đến các giải thuật heuristic để giải bài toán TSP, trong báo cáo này tác giả xin phép trình bày một hướng tiếp cận giải bài toán TSP và TSP đa mục tiêu thông qua phương pháp *Học tăng cường Reinforcement Learning* và kết hợp với các *mạng neural* của *Học sâu Deep Learning*. Bằng việc xây dựng mô hình dựa trên *Học tăng cường* và *Học sâu*, mô hình tối ưu sau qua trình huấn luyện sẽ có thể đưa ra lời giải tối ưu cho các bộ dữ liệu khác mà không cần tốn chi phí và thời gian huấn luyện lại. Các kết quả thực nghiệm đã cho thấy kết quả vô cùng tốt của hướng tiếp cận này.

Báo cáo này sẽ trình bày mô hình giải bài toán TSP đa mục tiêu (MOTSP) bằng phương pháp *Deep Reinfocement Learning* [9] thông qua việc chia nhỏ bài toán ban đầu thành N bài toán vô hướng con và kết hợp nghiệm tối ưu của N bài toán con đó sẽ tìm được *Pareto Front* của bài toán ban đầu. Báo cáo sẽ trình bày cụ thể kiến trúc mạng được sử dụng để giải bài toán MOTSP đó là *Pointer Network* và đồng thời sẽ tiến hành cài đặt mô hình và tiến hành so sánh giữa *Pointer Network* với thuật toán chính xác giải bằng quy hoạch động cũng như giải thuật tiến hóa đa mục tiêu giải bài toán MOTSP.

Quy tắc viết đồ án và ký hiệu

Trong báo cáo này, tác giả sử dụng một số tên viết tắt cho các thuật ngữ như sau:

DRL	Deep Reinforcement Learning
MOP	Multi-objective Optimization Problems
TSP	Travelling Salesman Problem
MOTSP	Multi-objective Travelling Salesman Problem

Bảng 1: Bảng thuật ngữ viết tắt

Đồng thời, tác giả sẽ sử dụng các ký hiệu sau trong báo cáo:

$\lambda \in \mathbb{R}^p$	vector trọng số
$y^1 < y^2$	$y_k^1 < y_k^2$ với $k = 1, \ldots, p$
$y^1 \leqq y^2$	$y_k^1 \leq y_k^2$ với $k = 1, \ldots, p$
$y^1 \leq y^2$	$y^1 \leqq y^2$ nhưng $y^1 \neq y^2$
$\mathbb{R}^p_>$	$\{y \in \mathbb{R}^p : y > 0\}$
$\mathbb{R}^p_\geq$	$\{y \in \mathbb{R}^p : y \geq 0\}$
$\mathbb{R}^p_\leqq$	$\{y \in \mathbb{R}^p : y \leqq 0\}$
i.i.d	biến ngẫu nhiên độc lập có phân phối đồng nhất

Bảng 2: Bảng ký hiệu

Cấu trúc của đồ án

Đồ án I này sẽ được trình bày gồm 4 chương chính như sau:

- **Chương 1: Cơ sở lý thuyết** sẽ trình bày các kiến thức cơ bản liên quan đến tối ưu đa mục tiêu, kiến trúc *mạng Neural*, lý thuyết về *Học tăng cường* cũng như trình bày về các bài toán TSP, MOTSP và một phương pháp đã được áp dụng để giải 2 bài toán trên là các giải thuật tiến hóa đa mục tiêu.
- **Chương 2: Ứng dụng Deep Reinforcement Learning giải bài toán TSP** sẽ trình bày về kiến trúc mạng *Pointer Network* cũng như quy tắc luyện *Actor-Critic* của *Học tăng cường* được sử dụng để tối ưu mạng *Pointer*.
- **Chương 3: Cài đặt và tính toán thử nghiệm** sẽ tiến hành cài đặt mô hình, tính toán thử nghiệm và so sánh với những cách tiếp cận khác để giải bài toán MOTSP như thuật toán Quy hoạch động giải chính xác, các giải thuật tiến hóa đa mục tiêu *MOEA, NSGA-II*,..
- **Chương 4: Kết luận và hướng phát triển đề tài**.

Dù đã rất cố gắng xong đồ án này vẫn không tránh khỏi những hạn chế cần khắc phục. Vì vậy, tác giả rất mong quý thầy cô đưa ra những ý kiến góp ý bổ ích để đồ án này tiếp tục được phát triển và có những kết quả mới tốt hơn.

Lời cảm ơn

Báo cáo này được thực hiện và hoàn thành tại Trường Đại học Bách Khoa Hà Nội, nằm trong nội dung học phần *Đồ án I* của kì học 2021-2.

Tác giả xin được dành lời cảm ơn chân thành tới TS. Trần Ngọc Thăng, là giảng viên đã trực tiếp hướng dẫn và gợi ý cho tác giả đề tài rất thú vị này, đồng thời thầy cũng đã giúp đỡ tận tình và có những đóng góp bổ ích để tác giả có thể hoàn thành báo cáo này một cách tốt nhất.

Hà Nội, ngày ... tháng 07 năm 2022.

Tác giả đồ án

Nguyễn Anh Minh

Mục lục

Chương 1

Cơ sở lý thuyết

1.1 Tối ưu đa mục tiêu

1.1.1 Kiến thức liên quan đến tối ưu đa mục tiêu

Trong mục này, tác giả sẽ trình các kiến thức liên quan của tối ưu đa mục tiêu như khái niệm điểm Pareto, điểm hữu hiệu yếu, phương pháp vô hướng hóa,... Các kiến thức trên được tham khảo từ sách [4].

Bài toán tối ưu đa mục tiêu MOP được phát biểu như sau:

$$\begin{aligned} \text{Min} \quad & f(x) = f_k(x) \quad k = 1, \ldots, p \quad \text{(MOP)} \\ \text{v.đ.k} \quad & x \in \mathcal{X} \subset \mathbb{R}^n_+ \end{aligned}$$

trong đó, n là số chiều của biến, $f_k(x) : \mathbb{R}^n_+ \to \mathbb{R}$ biểu diễn hàm mục tiêu thứ k và $\mathcal{X}$ là tập chấp nhận được. Với những phép toán quan hệ hai ngôi được định nghĩa ở Bảng 2, ta có một số định nghĩa như sau

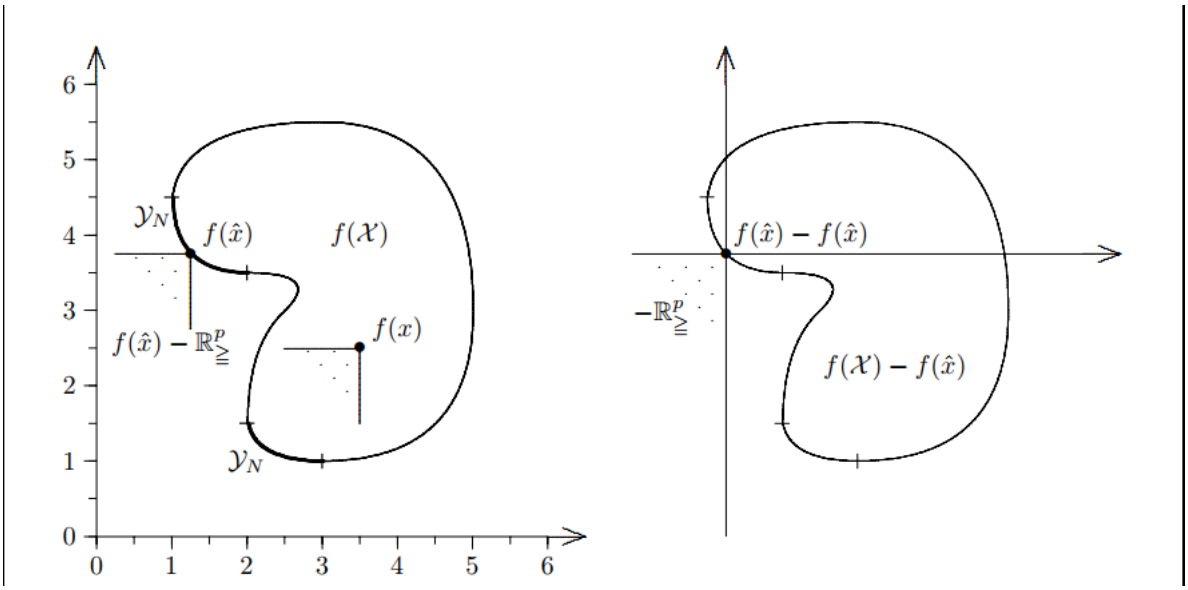
Định nghĩa 1.1.1 *Một điểm chấp nhận được $\hat{x} \in \mathcal{X}$ được gọi là một điểm hữu hiệu (efficient point) hay điểm tối ưu Pareto nếu không còn điểm $x \in \mathcal{X}$ nào khác sao cho $f(x) \leq f(\hat{x})$. Nếu $\hat{x}$ là điểm hữu hiệu thì $f(\hat{x})$ được gọi là điểm không trội (nondominated point). Nếu $x^1, x^2 \in \mathcal{X}$ và $f\left(x^1\right) \leq f\left(x^2\right)$ thì ta phát biểu rằng x^1 trội hơn x^2 và $f\left(x^1\right)$ trội hơn $f\left(x^2\right)$. Tập tất cả các điểm hữu hiệu $\hat{x} \in \mathcal{X}$ được ký hiệu là $\mathcal{X}_E$ và được gọi là tập điểm hữu hiệu. Tập tất cả các điểm không trội $\hat{y} = f(\hat{x}) \in \mathcal{Y}$, với $\hat{x} \in \mathcal{X}_E$ được ký hiệu là $\mathcal{Y}_N$ và được gọi là tập điểm không trội.*

Cụ thể, một điểm $\hat{x}$ được gọi là điểm hữu hiệu nếu thỏa mãn một trong các điều kiện sau đây:

1. Không tồn tại $x \in \mathcal{X}$ thỏa mãn $f_k(x) \leq f_k(\hat{x})$ với mọi $k = 1, \ldots, p$ và $f_i(x) < f_i(\hat{x})$ với một vài trường hợp $i \in \{1, \ldots, k\}$.
2. Không tồn tại $x \in X$ sao cho $f(x) - f(\hat{x}) \in -\mathbb{R}^p_{\leqq} \backslash \{0\}$.
3. $f(x) - f(\hat{x}) \in \mathbb{R}^p \backslash \left\{-\mathbb{R}^p_{\leqq} \backslash \{0\}\right\}$ với mọi $x \in \mathcal{X}$.
4. $f(\mathcal{X}) \cap \left(f(\hat{x}) - \mathbb{R}^p_{\leqq}\right) = \{f(\hat{x})\}$.

5. Không tồn tại $f(x) \in f(\mathcal{X}) \backslash \{f(\hat{x})\}$ với $f(x) \in f(\hat{x}) - \mathbb{R}^p_{\leqq}$.

6. Nếu $f(x) \leqq f(\hat{x})$ với $x \in \mathcal{X}$ thì $f(x) = f(\hat{x})$.

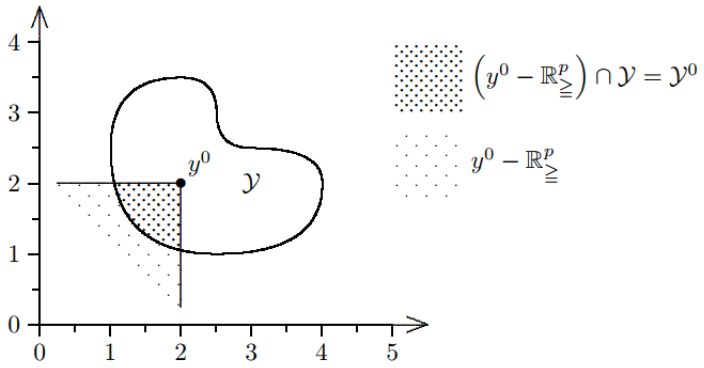


Hình 1.1: Minh họa định nghĩa điểm hữu hiệu

Hình 1.1 miêu tả các tính chất của điểm hữu hiệu tương ứng với 6 tính chất trên. Phần hình bên trái của Hình 1.1 biểu thị cho các tính chất $1, 4, 5$ và hình bên phải biểu thị cho tính chất $2, 3$.

Khi khảo sát tính chất của tập $\mathcal{X}_E$ và tập $\mathcal{Y}_N$, ta cần khảo sát sự tồn tại đầu tiên, ta cần xét xem 2 tập đó có khác rỗng hay chứa các *điểm cô lập* (isolated points) hay không. Để khảo sát được các tính chất của tập điểm hữu hiệu và tập điểm không trội ta cần một số định nghĩa và định lý sau.

Định lý 1.1.2 *(Borwein (1983)). Cho $\mathcal{Y}$ là một tập khác rỗng và giả sử rằng tồn tại $y^0 \in \mathcal{Y}$ sao cho bộ phận $\mathcal{Y}^0 = \{y \in \mathcal{Y} : y \leqq y^0\} = \left(y^0 - \mathbb{R}^p_{\geq}\right) \cap \mathcal{Y}$ có tính compact (khi đó, ta gọi $\mathcal{Y}$ là compact bộ phận (compact section)) thì $\mathcal{Y}_N$ khác rỗng.*

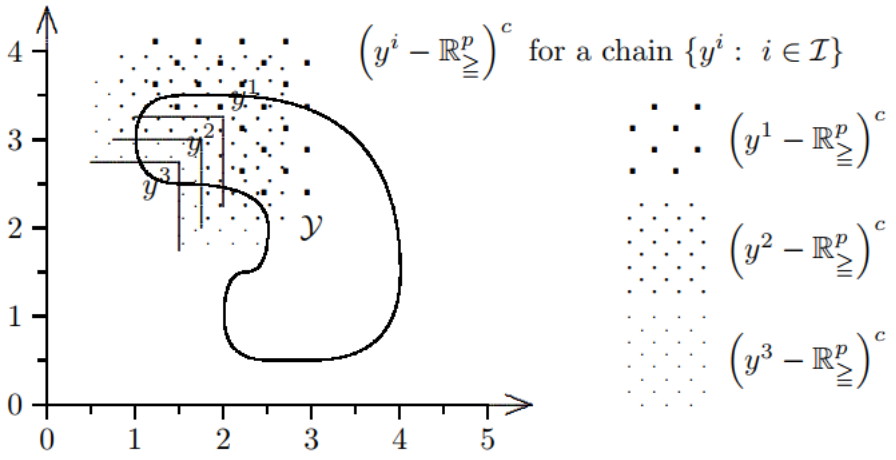


Hình 1.2: $\mathcal{Y}^0$ là compact bộ phận của $\mathcal{Y}$

Định nghĩa 1.1.3 *Tập $\mathcal{Y} \subset \mathbb{R}^p$ được gọi là $\mathbb{R}^p_{\leqq}$-semicompact nếu mọi phủ mở $\mathcal{Y}$: $\left\{\left(y^i - \mathbb{R}^p_{\leqq}\right)^c : y^i \in \mathcal{Y}, i \in \mathcal{I}\right\}$ có một phủ con (subcover) hữu hạn. Điều đó có nghĩa là, với mọi $\mathcal{Y} \subset \cup_{i\in\mathcal{I}} \left(y^i - \mathbb{R}^p_{\leqq}\right)^c$ thì tồn tại $m \in \mathbb{N}$ và $\{i_1, \ldots, i_m\} \subset \mathcal{I}$ sao cho*

$$\mathcal{Y} \subset \bigcup_{k=1}^{m} \left(y^{i_k} - \mathbb{R}^p_{\leqq}\right)^c.$$

Ở đây, $\left(y^i - \mathbb{R}^p_{\leqq}\right)^c$ ký hiệu phần bù $\mathbb{R}^p \backslash \left(y^i - \mathbb{R}^p_{\leqq}\right)$ của $y^i - \mathbb{R}^p_{\leqq}$. Chú ý các tập này luôn mở.



Hình 1.3: Minh họa phủ của tập $\mathcal{Y}$

Định lý 1.1.4 *(Corley (1980)) Nếu $\mathcal{Y} \neq \emptyset$ là tập $\mathbb{R}^p_{\leqq}$-semicompact thì $\mathcal{Y}_N \neq \emptyset$.*

Định nghĩa 1.1.5 *Một tập $\mathcal{Y} \subset \mathbb{R}^p$ được gọi là $\mathbb{R}^p_{\leqq}$-compact, nếu với mọi $y \in \mathcal{Y}$ phần $\left(y - \mathbb{R}^p_{\leqq}\right) \cap \mathcal{Y}$ là tập compact.*

Bổ đề 1.1.6 *Nếu $\mathcal{Y}$ là tập $\mathbb{R}^p_{\leqq}$-compact thì $\mathcal{Y}$ là tập $\mathbb{R}^p_{\leqq}$-semicompact.*

Định lý 1.1.7 *(Hartley (1978)). Nếu $\mathcal{Y} \subset \mathbb{R}^p$ khác rỗng và $\mathbb{R}^p_{\leqq}-$ compact thì $\mathcal{Y}_N \neq \emptyset$.*

Trên đây, ta đã liệt kê một số định lý giúp chứng minh sự tồn tại của tập điểm không trội $\mathcal{Y}_N$. Sau đây, tác giả xin trình bày một số định lý liên quan tới việc chứng minh sự tồn tại của tập điểm hữu hiệu $\mathcal{X}_E$.

Định nghĩa 1.1.8 *Một ánh xạ $f : \mathbb{R}^n \to \mathbb{R}^p$ là $\mathbb{R}^p_{\leqq}$-nửa liên tục (semicontinuous) nếu*

$$f^{-1}\left(y - \mathbb{R}^p_{\leqq}\right) = \left\{x \in \mathbb{R}^n : y - f(x) \in \mathbb{R}^p_{\leqq}\right\}$$

là tập đóng với mọi $y \in \mathbb{R}^p$

Bổ đề 1.1.9 *Cho tập $\mathcal{X} \subset \mathbb{R}^n$ khác rỗng và compact, $f : \mathbb{R}^n \to \mathbb{R}^p$ là $\mathbb{R}^p_{\leqq}$-nửa liên tục thì $\mathcal{Y} = f(\mathcal{X})$ là tập $\mathbb{R}^p_{\leqq}$-semicompact.*

Chứng minh: Đặt $\left\{\left(y^i - \mathbb{R}^p_{\geq}\right)^c : y^i \in \mathcal{Y}, i \in \mathcal{I}\right\}$ là phủ mở của $\mathcal{Y}$. Bằng tính chất $\mathbb{R}^p_{\geq}-$ nửa liên tục của f thì $\left\{f^{-1}\left(\left(y^i - \mathbb{R}^p_{\geqq}\right)^c\right) : y^i \in \mathcal{Y}, i \in \mathcal{I}\right\}$ là một phủ mở của $\mathcal{X}$. Vì $\mathcal{X}$ là tập compact nên tồn tại một phủ con hữu hạn. Ảnh của phủ con hữu hạn này là một phủ con hữu hạn của $\mathcal{Y}$, do đó $\mathcal{Y}$ là tập $\mathbb{R}^p_{\geqq}$-semicompact.

Định lý 1.1.10 *Cho tập $\mathcal{X} \subset \mathbb{R}^n$ khác rỗng và compact. Ánh xạ f là $\mathbb{R}^p_{\geqq}-$ semicontinuous thì $\mathcal{X}_E \neq \emptyset$.*

Sau đây, ta xét tới khái niệm và tính chất của điểm hữu hiệu yếu, điểm không trội yếu và tập điểm hữu hiệu, tập điểm không trội yếu.

Định nghĩa 1.1.11 *Một nghiệm chấp nhận được $\hat{x} \in \mathcal{X}$ được gọi là điểm hữu hiệu yếu hay điểm tối ưu Pareto yếu nếu không tồn tại $x \in \mathcal{X}$ sao cho $f(x) < f(\hat{x})$ nghĩa là $f_k(x) < f_k(\hat{x})$ với mọi $k = 1, \ldots, p$. Điểm $\hat{y} = f(\hat{x})$ được gọi là điểm không trội yếu. Tập điểm hữu hiệu yếu và tập điểm không trội yếu được ký hiệu lần lượt là $\mathcal{X}_{wE}$ và $\mathcal{Y}_{wE}$.*

Từ định nghĩa, ta chỉ ra được tính chất sau: $\mathcal{Y}_N \subset \mathcal{Y}_{wN}$ và $\mathcal{X}_E \subset \mathcal{X}_{wE}$.

Sau đây, ta chỉ ra một số tính chất của điểm hữu hiệu yếu. Một điểm $\hat{x} \in \mathcal{X}$ được gọi là điểm hữu hiệu yếu khi có những tính chất sau:

1. Không tồn tại $x \in \mathcal{X}$ sao chi $f(\hat{x}) - f(x) \in \operatorname{int} \mathbb{R}^p_{\geqq} = \mathbb{R}^p_{>}$
2. $(f(\hat{x}) - \mathbb{R}^p_{>}) \cap \mathcal{Y} = \emptyset$.

Một số định lý thông dụng chỉ ra sự tồn tại của tập $\mathcal{X}_{wE}$ và tập $\mathcal{Y}_{wN}$ như sau:

Định lý 1.1.12 *Với tập $\mathcal{Y} \subset \mathbb{R}^p$ khác rỗng và compact thì $\mathcal{Y}_{wN} \neq \emptyset$.*

Chứng minh: Giả sử $\mathcal{Y}_{wN} = \emptyset$, khi đó với mọi $y \in \mathcal{Y}$ tồn tại $y' \in \mathcal{Y}$ sao cho $y \in y' + \mathbb{R}^p_{>}$. Xét trên toàn bộ $y \in \mathcal{Y}$ ta có:

$$\mathcal{Y} \subset \bigcup_{y'} (y' + \mathbb{R}^p_{>}) \tag{1.1}$$

vì $\mathbb{R}^p_{>}$ là tập mở nên 1.1 xác định một phủ mở của $\mathcal{Y}$. Bằng tính compact của tập $\mathcal{Y}$ tồn tại một phủ con hữu hạn:

$$\mathcal{Y} \subset \bigcup_{i=1}^{k} \left(y^i + \mathbb{R}^p_{>}\right)$$

Chọn y^i về phía bên tay trái của mặt phẳng, điều này dẫn tới với mọi $i = 1, \ldots, k$ tồn tại $1 \leq j \leq k$ với $y^i \in y^j + \mathbb{R}^p_{>}$ nghĩa là với mọi i thì tồn tại j sao cho $y^j < y^i$. Bằng tính bắc cầu của quan hệ hai ngôi $<$ và có hữu hạn y^i nên tồn tại i^*, m và ta xây dựng được chuỗi sau $y^{i^*} < y^{i_1} < \ldots < y^{i_m} < y^{i^*}$, vô lý. Do đó giả sử sai và định lý đã được chứng minh.

Bổ đề 1.1.13 *Cho tập $\mathcal{X} \subset \mathbb{R}^n$ khác rỗng và compact. Giả sử rằng $f : \mathbb{R}^n \to \mathbb{R}^p$ liên tục thì $\mathcal{X}_{wE} \neq \emptyset$.*

1.1.2 Phương pháp vô hướng hóa

Bài toán (MOP) đã trình bày ở trên có thể giải được thông qua phương pháp vô hướng hóa tức ta đi giải bài toán tối ưu sau:

$$\min \sum_{k=1}^{p} \lambda_k f_k(x)$$

$$\text{v.đ.k} \quad x \in X$$

với $\lambda \in \mathbb{R}^p_+$ là vector trọng số với các thành phần $\lambda_k, k = 1, \ldots, p$ dương. Để giải các bài toán (MOP) ta thường tiến hành giải trên không gian ảnh với tập $\mathcal{Y}$ trước do số chiều của không gian ảnh thường nhỏ hơn số chiều của không gian quyết định. Trong phần này, tác giả sẽ trình bày mối liên hệ giữa các điểm không trội với giá trị $\sum_{k=1}^{p} \lambda_k y_k$ cũng như chứng minh lời giải tối ưu cho bài toán vô hướng hóa cũng là nghiệm của bài toán (MOP) với các điều kiện nhất định.

Cho $\mathcal{Y} \subset \mathbb{R}^p$ với một $\lambda \in \mathbb{R}^p_{\leqq}$ cho trước, ta ký hiệu:

$$\mathcal{S}(\lambda, \mathcal{Y}) := \left\{ \hat{y} \in \mathcal{Y} : \langle \lambda, \hat{y} \rangle = \min_{y \in \mathcal{Y}} \langle \lambda, y \rangle \right\}$$

là tập nghiệm tối ưu của $\mathcal{Y}$ ứng với λ.

$$\mathcal{S}(\mathcal{Y}) := \bigcup_{\lambda \in \mathbb{R}^p_{>}} \mathcal{S}(\lambda, \mathcal{Y}) = \bigcup_{\left\{\lambda > 0 : \sum_{k=1}^{p} \lambda_k = 1\right\}} \mathcal{S}(\lambda, \mathcal{Y})$$

$$\text{và } \mathcal{S}_0(\mathcal{Y}) := \bigcup_{\lambda \in \mathbb{R}^p_{\geq}} \mathcal{S}(\lambda, \mathcal{Y}) = \bigcup_{\left\{\lambda \geq 0 : \sum_{k=1}^{p} \lambda_k = 1\right\}} \mathcal{S}(\lambda, \mathcal{Y}).$$

Định nghĩa 1.1.14 *Một tập $\mathcal{Y} \in \mathbb{R}^p$ được gọi là $\mathbb{R}^p_{\leqq}$-lồi nếu $\mathcal{Y} + \mathbb{R}^p_{\leqq}$ là tập lồi. Mọi tập lồi $\mathcal{Y}$ hiển nhiên cũng là tập $\mathbb{R}^p_{\leqq}$-lồi.*

Sau đây là một số định lý chứng minh tính đúng đắn của phương pháp vô hướng hóa

Định lý 1.1.15 *Với mọi tập $\mathcal{Y} \subset \mathbb{R}^p$ ta có $\mathcal{S}_0(\mathcal{Y}) \subset \mathcal{Y}_{wN}$.*

Chứng minh: Cho $\lambda \in \mathbb{R}^p_{\geq}$ và $\hat{y} \in \mathcal{S}(\lambda, \mathcal{Y})$. Khi đó

$$\sum_{k=1}^{p} \lambda_k \hat{y}_k \leq \sum_{k=1}^{p} \lambda_k y_k \text{ for all } y \in \mathcal{Y}.$$

Giả sử rằng $\hat{y} \notin \mathcal{Y}_{wN}$. Khi đó tồn tại $y' \in \mathcal{Y}$ với $y'_k < \hat{y}_k, k = 1, \ldots, p$. Do đó:

$$\sum_{k=1}^{p} \lambda_k y'_k < \sum_{k=1}^{p} \lambda_k \hat{y}_k,$$

do ít nhất một trong các trọng số λ_k phải dương. Điều vô lý này dẫn tới giả thiết sai và định lý đã được chứng minh.

Định lý 1.1.16 *Nếu $\mathcal{Y}$ là tập $\mathbb{R}^p_{\leqq}$-lồi khi đó $\mathcal{Y}_{wN} = \mathcal{S}(\mathcal{Y})$.*

Định lý 1.1.17 *Cho tập $\mathcal{Y} \subset \mathbb{R}^p$ khi đó $\mathcal{S}(\mathcal{Y}) \subset \mathcal{Y}_N$*

Chứng minh: Cho $\hat{y} \in \mathcal{S}(\mathcal{Y})$ khi đó tồn tại $\lambda \in \mathbb{R}^p_>$ thỏa mãn $\sum_{k=1}^p \lambda_k \hat{y}_k \leq \sum_{k=1}^p \lambda_k y_k$ với mọi $y \in \mathcal{Y}$. Giả sử $\hat{y} \notin \mathcal{Y}_N$ khi đó tồn tại $y' \in \mathcal{Y}$ với $y' \leq y$. Nhân các thành phần với các trọng số $\lambda_k y'_k \leq \lambda_k \hat{y}_k$ với mọi $k = 1, \ldots, p$ và thỏa mãn rằng buộc chặt với một k cụ thể. Các rằng buộc chặt với λ_k dương dẫn tới $\sum_{k=1}^p \lambda_k y'_k < \sum_{k=1}^p \lambda_k \hat{y}_k$ mâu thuẫn do $\hat{y} \in \mathcal{S}(\mathcal{Y})$. Do đó điều giả sử sai và định lý đã được chứng minh.

Bổ đề 1.1.18 *$\mathcal{Y}_N \subset \mathcal{S}_{(\mathcal{Y})}$ nếu $\mathcal{Y}$ là một tập $\mathbb{R}^p_{\geqq}$-lồi.*

Sau đây là một số tính chất quan trọng về nghiệm tối ưu của bài toán vô hướng hóa

Bổ đề 1.1.19 *Giả sử rằng $\hat{x}$ là nghiệm tối ưu của bài toán vô hướng hóa*

$$\min_{x \in \mathcal{X}} \sum_{k=1}^p \lambda_k f_k(x). \tag{1.2}$$

với $\lambda \in \mathbb{R}^p_>$. Khi đó, các tính chất sau đúng:

1. *Nếu $\lambda \in \mathbb{R}^p_>$ thì $\hat{x} \in \mathcal{X}_{wE}$.*
2. *Nếu $\lambda \in \mathbb{R}^p_\geq$ thì $\hat{x} \in \mathcal{X}_E$.*
3. *Nếu $\lambda \in \mathbb{R}^p_>$ và $\hat{x}$ là nghiệm tối ưu duy nhất của 1.2 khi đó $\hat{x} \in \mathcal{X}_{sE}$.*

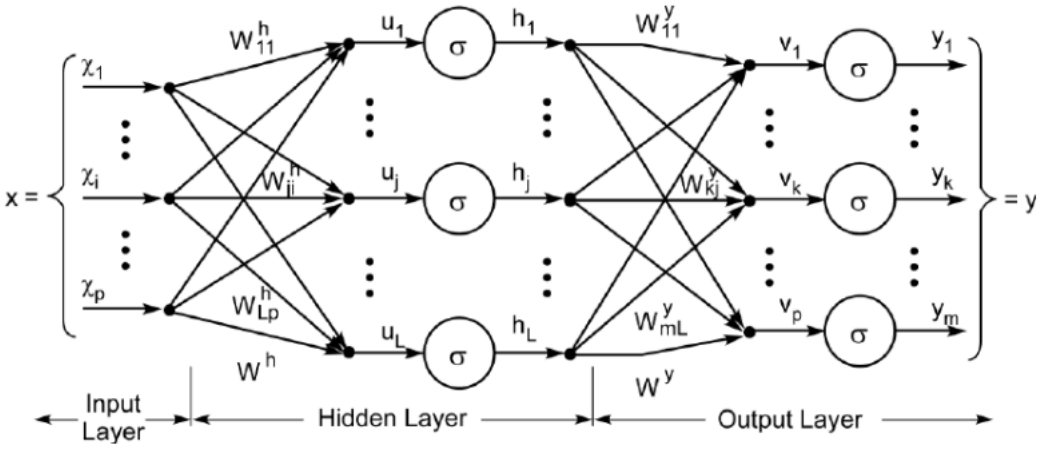
Bổ đề 1.1.20 *Cho $\mathcal{X}$ là tập lồi và f_k là hàm lồi, $k = 1, \ldots, p$. Nếu $\hat{x} \in \mathcal{X}_{wE}$ thì tồn tại $\lambda \in \mathbb{R}^p_>$ sao cho $\hat{x}$ là nghiệm tối ưu của (1.2).*

1.2 Mô hình mạng Neural

Trong phần này, tác giả sẽ trình bày các kiến thức cơ bản về mạng Neural và kiến trúc của mạng Recurrent Neural. Các kiến thức trong phần này được tham khảo chính trong sách [6]. Các ký hiệu in đậm như $\boldsymbol{v}$ biểu thị cho các vector.

1.2.1 Tổng quan mạng Neural

Mô hình mạng Neural được xây dựng dựa trên mô hình tổng quát Multilayer Perceptrion-MLP (theo [6]). Một MLP là ánh xạ $y = f*(x, \theta)$ với input là dữ liệu và output là giá trị dự đoán. Hàm f được xây dựng từ các hàm tuyến tính, hàm tanh, ... Bằng việc lựa chọn các hàm thích hợp và dữ liệu đầu vào mà mô hình MLP có thể xấp xỉ được tham số θ và từ đó giúp máy tính xác định đầu ra.



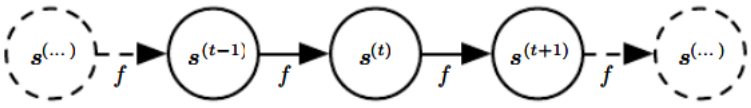
Hình 1.4: Kiến trúc cơ bản của mạng MLP

Kiến trúc của mạng Neural được xây dựng tương tự như ở Hình 1.4 với 3 thành phần chính là:

1. Input Layer: Lớp input dữ liệu.
2. Hidden Layer: Lớp tính toán, lưu trữ thông tin trung gian gồm nhiều các *node* mạng với các *activation function* f khác nhau.
3. Output Layer: Lớp đầu ra dự đoán của mô hình.

Mỗi *layer* là một tầng trong mạng Neural, một *Hidden Layer và Output Layer* có thể gồm nhiều *node* (tương ứng các σ trong Hình 1.4) là thành phần lưu trữ thông tin trong mạng, mỗi *node* sẽ liên kết với toàn bộ các *node* của *layer* trước đó với các hệ số w riêng và ở mỗi *node* sẽ diễn ra 2 bước tính toán: tính tổng tuyến tính với các hệ số w và sau đó sử dụng *activation function* riêng để tính toán ra đầu ra dùng cho *layer* tiếp theo.

Việc mô hình hóa tính toán của mạng Neural được biểu diễn thông qua *computational graph* như sau:



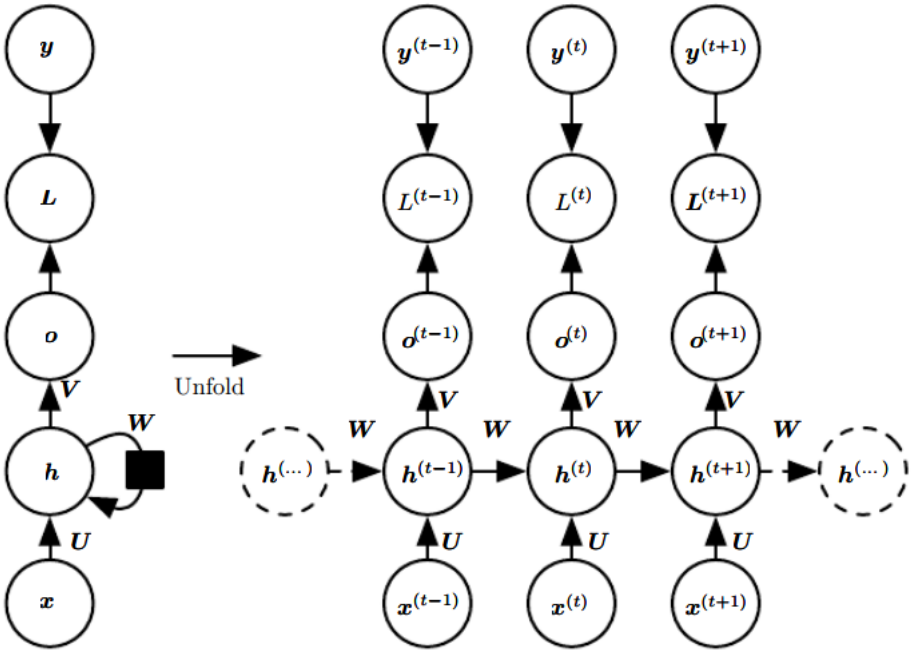
Hình 1.5: Computational Graph

Mỗi một *node* có một *activation function* f riêng (ví dụ như hàm tuyến tính, hàm *relu*, hàm *tanh*, hàm *softmax*,...) và được minh họa trong *computational graph* như trên. Ví dụ trong hình 1.5, ta có biểu diễn cách tính giá trị *state* $\boldsymbol{s}^{(t)}$ của *layer* sau được tính thông qua *layer* trước:

$$\boldsymbol{s}^{(t)} = f\left(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}\right)$$

1.2.2 Sequence Modeling: Mạng LSTM và GRU

Mạng *Long-short term memory LSTM* và mạng *Gated Recurrent Unit GRU* đều có kiến trúc chung là mạng *Recurrent Neural RNN*, chúng được sử dụng rộng rãi trong các mô hình *Natural Language Processing NLP*, cụ thể là dịch và xử lý các đoạn văn bản. Kiến trúc mạng RNN tổng quát có thể được biểu diễn như sau:



Hình 1.6: Computational graph biểu diễn RNN

Việc tính toán *forward* trong mạng RNN được tổng quát thông qua hàm sau:

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, x^{(t)}; \boldsymbol{\theta})$$

Như trong hình 1.6 biểu diễn, mạng RNN sẽ gồm các ánh xạ nối chuỗi input $\boldsymbol{x}$ đến các giá trị output tương ứng của từng *node* là $\boldsymbol{o}$. Một hàm mất mát *loss* L sẽ tính toán độ sai khác giữa mỗi $\boldsymbol{o}$ với giá trị *training* $\boldsymbol{y}$ thông qua đại lượng dự đoán của mạng là $\hat{\boldsymbol{y}} = softmax(\boldsymbol{o})$. Mạng RNN gồm các ma trận trọng số sau: ma trận trọng số $\boldsymbol{U}$ tính toán từ input $\boldsymbol{x}$ tạo ra các giá trị ẩn *hidden* $\boldsymbol{h}$ của mạng, các giá trị $\boldsymbol{h}$ sẽ được dùng làm input cho bước lặp tiếp theo trong *forward* mạng RNN và các giá trị $\boldsymbol{h}$ đó được kết nối với nhau thông qua ma trận trọng số $\boldsymbol{W}$, việc tính toán từ $\boldsymbol{h}$ tới các giá trị output $\boldsymbol{o}$ của mạng thông qua ma trận trọng số $\boldsymbol{V}$. Cụ thể, quá trình tính toán trong mạng RNN sẽ tiến hành như sau:

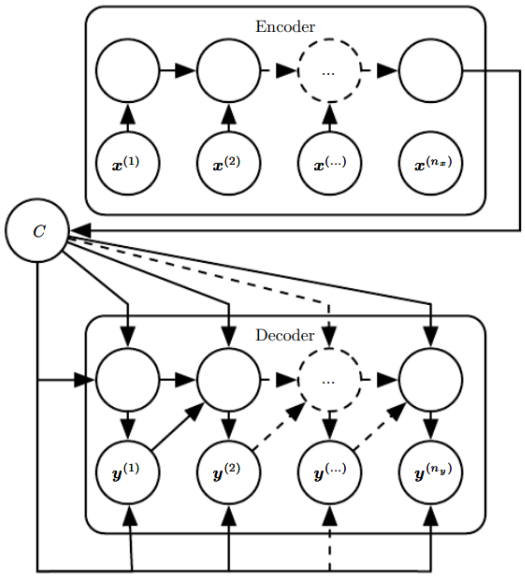
$$\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh\left(\boldsymbol{a}^{(t)}\right) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \text{softmax}\left(\boldsymbol{o}^{(t)}\right)
\end{aligned}$$

Các tham số *bias* của mạng được biểu diễn thông qua vectors $\boldsymbol{b}$ và $\boldsymbol{c}$. Mô hình trên được sử dụng trong trường hợp chuỗi input $\boldsymbol{x}$ và chuỗi output $\boldsymbol{y}$ có cùng độ dài. Tổng giá trị hàm mất mát khi biết trước chuỗi input $\boldsymbol{x}$ và chuỗi output $\boldsymbol{y}$ là tổng các hàm mất mát của tất cả các bước lặp. Ví dụ $L^{(t)}$ là hàm mất mát *negative log-likelihood* của $y^{(t)}$ cho trước chuỗi $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}$ thì:

$$\begin{aligned} & L\left(\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\right\}, \left\{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\right\}\right) \\ =& \sum_t L^{(t)} \\ =& -\sum_t \log p_{\text{model}}\left(y^{(t)} \mid \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\right\}\right) \end{aligned}$$

với $p_{\text{model}}\left(y^{(t)} \mid \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\right\}\right)$ được tính thông qua mô hình khi so sánh giá trị của $y^{(t)}$ với giá trị dự đoán của mô hình là $\hat{\boldsymbol{y}}^{(\boldsymbol{t})}$.

Kiến trúc Encoder-Decoder Sequence-to-Sequence

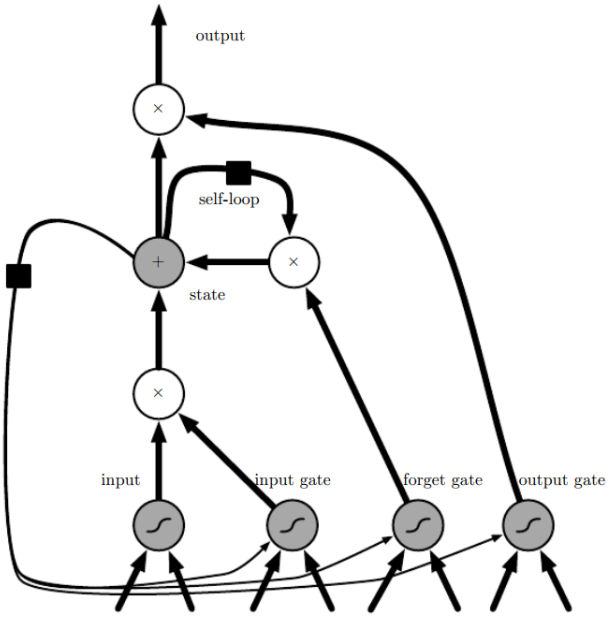


Hình 1.7: Kiến trúc Encoder-Decoder

Hình 1.7 miêu tả kiến trúc kiểu sequence-to-sequence của mạng RNN trong việc học tạo ra chuỗi output $\left(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n_y)}\right)$ khi biết trước chuỗi input $\left(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n_x)}\right)$. Nó là sự kết hợp của một khối Encoder RNN đọc chuỗi dữ liệu đầu vào và một khối Decoder RNN tạo ra chuỗi output đầu ra (hoặc tính xác suất xảy ra một sự kiện nào đó (bài toán phân loại) khi cho trước chuỗi đầu vào). Lớp ẩn cuối cùng của Encoder RNN được dùng để tính một biến "ngữ cảnh" (context) cố định C biểu diễn thông tin được trích xuất ra từ chuỗi đầu vào để là đầu vào cho khối Decoder RNN.

Mạng LSTM

Về mặt lý thuyết mạng RNN có thể mang thông tin từ các *layer* trước đến các *layer* sau, nhưng thực tế là thông tin chỉ mang được qua một số lượng *state* nhất định, sau đó thì sẽ xảy ra hiện tượng *vanishing gradient* (tức đạo hàm hàm *loss* theo tham số của mạng xấp xỉ bằng 0) hay nói cách khác là mạng RNN chỉ học được từ các state gần nó, đây là hiện tượng *short term memory.* Với bài toán dịch văn bản trong xử lý ngôn ngữ tự nhiên thì input của mạng là những câu văn dài và ta cần thông tin của những câu văn trước đó để output đầu ra dịch đúng ngữ cảnh nhất. Chính vì vậy, để khắc phục hiện tượng *vanishing gradient* một mạng RNN cải tiến ra đời chính là mạng *Long-short term memory* LSTM.



Hình 1.8: Kiến trúc tế bào của mạng LSTM

Điểm đặc biệt trong cấu trúc tế bào của mạng LSTM là nó có cơ chế *self-loop* để làm giảm hiện tượng *vanishing gradient*, tế bào LSTM sẽ quyết định tỷ lệ lưu giữ thông tin của các *state* trước đó, các tham số của *self-loop* được điều khiển thông qua cổng "quên" (forget gate) $f_i^{(t)}$ (ứng với bước lặp thứ t và tế bào thứ i), cổng này sẽ điều chỉnh tỷ lệ quên thông tin trước trong khoảng từ 0 tới 1 thông qua hàm *sigmoid*:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right)$$

trong đó $\boldsymbol{x}^{(t)}$ là vector input hiện tại và $\boldsymbol{h}^{(t)}$ là vector *hidden layer* hiện tại chứa output của toàn bộ tế bào LSTM trước đó. $b^f, \boldsymbol{U}^f, \boldsymbol{W}^f$ lần lượt là tham số *bias* của mạng, trọng số của dữ liệu đầu vào và trọng số cho tầng cổng quên. Thành phần quan trọng nhất của tế bào là *state unit* $s_i^{(t)}$ lưu trữ thông tin

của mạng. Khi đó quá trình cập nhật giá trị trong mạng diễn ra như sau:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right)$$

với $\boldsymbol{b}, \boldsymbol{U}$ và $\boldsymbol{W}$ lần lượt ký hiệu cho hệ số *bias*, tham số của dữ liệu input và tham số *recurrent* của tế bào LSTM. Ta có thể thấy giá trị của s_i^t được cập nhật kết hợp với giá trị của tầng cổng quên $f_i^{(t)}$. Giá trị *external input gate* $g_i^{(t)}$ được tính toán tương tự như tầng cổng quên với các tham số riêng để tính toán tỷ lệ ảnh hưởng của dữ liệu đầu vào ở bước thứ t là $x_j^{(t)}$ và thông tin từ *state* trước đó là $h_j^{(t-1)}$:

$$g_i^{(t)} = \sigma \left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)} \right).$$

Giá trị output dự đoán $h_i^{(t)}$ của tế bào LSTM có thể bị bỏ qua thông qua cổng *output* $q_i^{(t)}$ cũng được tính thông qua hàm *sigmoid*:

$$h_i^{(t)} = \tanh \left(s_i^{(t)} \right) q_i^{(t)}$$
$$q_i^{(t)} = \sigma \left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)} \right)$$

có chứa các vector tham số $\boldsymbol{b}^o, \boldsymbol{U}^o, \boldsymbol{W}^o$ lần lượt ứng với *bias*, trọng số với dữ liệu input, trọng số của tầng cổng quên đối với kết quả output dự đoán của các *state* trước đó.

Mạng GRU

Mạng GRU *Gated Recurrent Unit* là một loại mạng RNN cũng được thiết kế để khắc phục hiện tượng *vanishing gradient*. Mạng GRU có kiến trúc tương tự mạng LSTM nhưng có sự thay đổi trong cách thiết kế tế bào *cell*, cụ thể là thay đổi cách thiết lập hàm để "quên" thông tin từ các *state* trước đó cũng như cách tính trọng số trong việc sử dụng thông tin từ các *state* trước đó:

$$h_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + \left(1 - u_i^{(t-1)} \right) \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t-1)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)} \right)$$

với $\boldsymbol{u}$ biểu thị cho *cổng cập nhật (update gate)* và $\boldsymbol{r}$ biểu thị cho *cổng reset*. Các cổng này được thiết lập tính toán như sau:

$$u_i^{(t)} = \sigma \left(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)} \right)$$

và

$$r_i^{(t)} = \sigma \left(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)} \right)$$

Cổng *reset* cũng như cổng *cập nhật* có thể bỏ qua thông tin của các *state* trước đó, cổng cập nhật có cơ chế tương tự như tầng cổng quên, cổng reset sẽ điều khiển phần thông tin nào của *state* trước đó được dùng để sử dụng trong việc tính toán của *state* này.

1.3 Học tăng cường

Trong phần này, tác giả sẽ trình bày các kiến thức liên quan tới Học tăng cường như mô hình toán học của Học tăng cường là quá trình quyết định Markov MDP, các khái niệm trong Học tăng cường như phần thưởng, chính sách, các cách xấp xỉ cho chính sách cũng như cập nhật chính sách để đạt phần thưởng tối đa,... Các kiến thức trong phần này sẽ được tham khảo chính tại sách [14].

1.3.1 Quá trình quyết định Markov (MDP)

Một quá trình quyết định Markov MDP được định nghĩa thông qua 5 thành phần như sau $(\mathcal{S}, \mathcal{A}, P(\cdot \mid \cdot, \cdot), R(\cdot, \cdot\,\cdot), \gamma)$, trong đó:

- $\mathcal{S}$ là tập hữu hạn biểu thị tất cả các trạng thái của quá trình. Ký hiệu trạng thái tại thời điểm t của quá trình là S_t.
- $\mathcal{A}$ là tập hữu hạn các hành động. Với mỗi trạng thái $s \in \mathcal{S}$ ta có tập hữu hạn các hành động tại s là $\mathcal{A}_s$. Ký hiệu hành động tại thời điểm t là A_t.
- $P(s' \mid s, a) = P(S_{t+1} = s' \mid S_t = s, A_t = a)$ là xác suất quá trình chuyển sang trạng thái s' tại thời điểm $t+1$ với $S_t = s$ và $A_t = a$.
- $R(s', a, s)$ là phần thưởng khi quá trình chuyển từ trạng thái s chọn hành động a và chuyển sang trạng thái s'.
- $0 \leq \gamma \leq 1$ là độ chiết khấu giữa phần thưởng hiện tại và phần thưởng trong tương lai.

Tại mỗi thời điểm t, quá trình đang ở trạng thái S_t và đưa ra quyết định chọn hành động $A_t \in \mathcal{A}_{S_t}$. Quá trình sẽ chuyển đến trạng thái S_{t+1} với xác suất chuyển $P(S_{t+1} \mid S_t, A_t)$ và được nhận phần thưởng tương ứng $R_t = R(S_{t+1}, A_t, S_t)$.

Ta có định nghĩa một chính sách π là một phương án lựa chọn hành động mà quá trình tuân theo. Chính sách π gồm hai loại sau:

1. **Xác định**: khi quá trình đang ở trạng thái bất kì $s \in \mathcal{S}$, hành động được chọn tiếp theo sẽ là $\pi(s)$;
2. **Ngẫu nhiên**: xác suất lựa chọn hành động $a \in \mathcal{A}_s$ là $\pi(a|s)$.

Mục tiêu đặt ra cho MDP: Cho một quá trình Markov quyết định $(\mathcal{S}, \mathcal{A}, P(\cdot \mid \cdot, \cdot), R(\cdot, \cdot, \cdot), \gamma)$ hãy đi xây dựng chính sách π nhằm tối ưu các phần thưởng thu được trong thời gian dài hạn (còn gọi là *lợi nhuận (return)*), trong đó công thức của lợi nhuận G_t với chiết khấu γ là:

$$G_t = R_t + \gamma R_{t+1} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

G_t là tổng tất cả các phần thưởng nhận được kể từ trạng thái s_t đến tương lai, với $0 < \gamma < 1$ thì các phần thưởng càng xa hiện tại sẽ càng bị *giảm giá (discount)* nhiều và ngược lại.

Định nghĩa 1.3.1 *Xét chuỗi biến ngẫu nhiên* $\{S_t\}_{t\in\mathbb{N}}$*, trạng thái* s *được gọi là có tính Markov nếu* $\forall s' \in \mathcal{S}$ *thì*

$$P\left(S_{t+1} = s' \mid S_t = s\right) = P\left(S_{t+1} = s' \mid h_{t-1}, S_t = s\right)$$

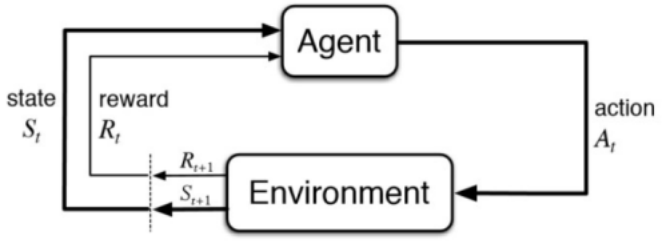
với mọi khả năng các sự kiện xảy ra trong quá khứ $h_{t-1} = \{S_1, \ldots, S_{t-1}, A_1, \ldots, A_{t-1}, R_1, \ldots, R_{t-1}\}$.

Trong MDP thì mọi trạng thái đều được giả sử có tính Markov, tức là trạng thái s biểu thị đầy đủ các thông tin liên quan từ các trạng thái trong quá khứ.

1.3.2 Mô hình hóa cho RL

MDP là biểu diễn mô hình toán học của Học tăng cường, ứng với các thành phần của MDP ta định nghĩa một số thuật ngữ được sử dụng trong RL như sau

- *Agent* (Tác tử): Tác tử quan sát môi trường và thực hiện các hành động tương ứng. Ta cần điều khiển tác tử sao cho đạt mục tiêu đề ra là tối đa hóa lợi nhuận.
- *Environment* (Môi trường): là không gian mà tác tử tương tác.
- *Policy* (Chính sách): Tác tử sẽ thực hiện các hành động theo chính sách để đạt được mục đích cụ thể với từng nhiệm vụ khác nhau.
- *Reward* (Phần thưởng): phần thưởng tương ứng từ môi trường mà tác tử nhận được khi thực hiện một hành động.
- *State* (Trạng thái): trạng thái của môi trường mà tác tử nhận được.
- *Episode* (tập): một chuỗi các trạng thái và hành động cho đến trạng thái kết thúc : $s_1, a_1, s_2, a_2, ...s_T, a_T$
- *Accumulative Reward* (phần thưởng tích lũy/lợi nhuận): tổng phần thưởng tích lũy từ trạng thái đầu tiên đến trạng thái cuối cùng. Như vậy, tại s_t, tác tử tương tác với môi trường và thực hiện hành động a_t, dẫn đến trạng thái mới s_{t+1} và nhận được phần thưởng tương ứng r_{t+1}. Quá trình lặp diễn ra như vậy cho đến trạng thái cuối cùng s_T.



Hình 1.9: Mô hình học tăng cường

Chính sách và hàm giá trị (value function)

Chiến sách là cốt lõi của tác tử trong việc xác định hành vi. Các thuật toán RL sẽ chỉ định cách mà tác tử thay đổi chính sách dựa vào chính những trải nghiệm (các bước đi và hành động trong quá khứ) của nó. Trong một số trường hợp, chiến lược có thể là một hàm hoặc bảng tra cứu đơn giản. Trong một số trường hợp khác, chiến lược có thể liên quan đến tính toán mở rộng, ví dụ như quá trình tìm kiếm.

Giá trị của trạng thái s tuân theo chính sách π, kí hiệu là $V^\pi(s)$ được định nghĩa kỳ vọng lợi nhuận khi bắt đầu từ trạng thái s và đi theo chính sách π:

$$V^\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right], \text{ với mọi } s \in \mathcal{S}$$

Chú ý rằng ta luôn có $V^\pi(s) = 0$ khi s là trạng thái dừng (terminal state). Hàm $V^\pi(s)$ được xác định như trên được gọi là hàm giá trị trạng thái (state-value function) của chính sách π.

Tương tự, ta định nghĩa giá trị của hành động a ở trạng thái s theo chính sách π, kí hiệu là $Q^\pi(s, a)$, là kỳ vọng lợi nhuận khi bắt đầu từ trạng thái s, chọn hành động a và sau đó tuân theo chính sách π:

$$Q^\pi(s, a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$$

Ta gọi Q^π là hàm giá trị hành động của policy π (action-value function) của chính sách π).

Dựa theo công thức xác suất đầy đủ ta có quan hệ giữa $V^\pi(s)$ và $Q^\pi(s, a)$:

$$V^\pi(s) = \sum_{a \in A} \pi_\theta(s, a) Q^\pi(s, a),$$

Mặt khác, với mọi $s \in \mathcal{S}$, ta có:

$$\begin{aligned} V^\pi(s) &\doteq \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \mid S_t = s \right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \\ &= \sum_a \pi(a \mid s) \sum_{s', r} p\left(s', r \mid s, a\right) \left[r + \gamma v_\pi\left(s'\right) \right] \quad \textit{(Bellman equation)} \end{aligned} \tag{1.3}$$

Phương trình (1.3) được gọi là phương trình Bellman và sẽ là phương trình cốt lõi cho việc cập nhật các hàm giá trị trong các thuật toán tối ưu chính sách của RL.

Chính sách tối ưu (Optimal policies) và hàm giá trị tối ưu (optimal value functions)

Mục đích của RL là tìm ra một chính sách giúp đạt được nhiều phần thưởng về mặt dài hạn. Với quá trình quyết định Markov hữu hạn, ta có thể định nghĩa chính xác chính sách tối ưu theo cách sau đây:

- Các hàm giá trị xác định thứ tự từng phần đối với các chính sách.
- Chính sách π được định nghĩa là tốt hơn hoặc chất lượng hơn một chính sách π' nếu lợi nhuận kỳ vọng của nó lớn hơn hoặc bằng π' cho tất cả các trạng thái.

- $\pi \geq \pi'$ khi và chỉ khi $V^{\pi}(s) \geq V^{\pi'}(s)$ với mọi $s \in \mathcal{S}$.

Luôn có ít nhất một chính sách tốt hơn hơn hoặc bằng tất cả các chính sách khác và được định nghĩa là chính sách tối ưu. Có thể có nhiều hơn một chính sách tối ưu, ký hiệu bằng π_*. Ứng với π_*, tác tử sẽ thực hiện các hành động và dẫn tới có một hàm giá trị trạng thái tối ưu (optimal state-value function), được ký hiệu là V_*, và được định nghĩa bởi:

$$V_*(s) \doteq \max_{\pi} V^{\pi}(s)$$

với mọi $s \in \mathcal{S}$.

Các chính sách tối ưu sẽ tương ứng hàm giá trị hành động tối ưu (optimal action-value function), kí hiệu là Q_* và được xác định bởi:

$$Q_*(s,a) \doteq \max_{\pi} Q^{\pi}(s,a)$$

với mọi $s \in \mathcal{S}$ và $a \in \mathcal{A}(s)$. Với mỗi cặp trạng thái-hành động (s,a), Q_* cho ta giá trị lợi nhuận kỳ vọng lớn nhất khi thực hiện hành động a ỏ trạng thái s và các hành động sau đó tuân theo chính sách tối ưu. Do đó, ta có thể viết Q_* ở dạng sau:

$$Q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma V_*\left(S_{t+1}\right) \mid S_t = s, A_t = a\right]$$

Dựa vào phương trình Bellman và định nghĩa của $V_*(s)$, ta có:

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

Hai công thức cuối chính là hai dạng của phương trình tối ưu Bellman cho V_*. Một cách trực quan, phương trình tối ưu Bellman diễn tả thực tế rằng giá trị của một trạng thái theo một chính sách tối ưu phải bằng với lợi nhuận mong đợi cho hành động tốt nhất từ trạng thái đó. Tương tự, ta cũng có phương trình tối ưu Bellman cho Q_* như sau:

$$\begin{aligned}
Q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} Q_*\left(S_{t+1}, a'\right) \mid S_t = s, A_t = a\right] \\
&= \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma \max_{a'} Q_*\left(s', a'\right)\right]
\end{aligned}$$

Đối với MDP hữu hạn, phương trình tối ưu Bellman cho v_π có một nghiệm duy nhất độc lập với chính sách. Phương trình tối ưu Bellman thực chất là một hệ phương trình, mỗi phương trình ứng với mỗi trạng thái, vì vậy nếu có n trạng thái, thì sẽ có n phương trình với n ẩn số. Nếu biết $p(.,.|.,.)$ của môi trường, thì về nguyên tắc ta có thể giải hệ phương trình này với giá v_* bằng cách sử dụng bất kỳ

một trong các phương pháp giải hệ phương trình phi tuyến. Tuy nhiên điều này trên thực tế khó thực hiện được do chi phí tính toán giải hệ phương trình tốn kém với bài toán phức tạp có nhiều trạng thái và việc xác định $p(.,.|.,.)$ của môi trường là điều rất khó khăn.

1.3.3 Các phương pháp Policy gradient

Trong phần này, chúng ta xem xét các phương pháp tìm hiểu một chính sách được tham số hóa có thể chọn các hành động mà không cần tham khảo một hàm giá trị. Hàm giá trị vẫn có thể được sử dụng để tìm hiểu tham số chính sách, nhưng không bắt buộc đối với lựa chọn hành động.

Kí hiệu $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ là vector tham số chính sách. Khi đó ta có: $\pi(a \mid s, \boldsymbol{\theta}) = \Pr\{A_t = a \mid S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$ là xác suất để hành động a được chọn tại thời điểm t khi môi trường đang ở trạng thái s với tham số θ.

Trong phần này, ta xem xét các phương pháp học tham số chính sách dựa trên gradient của một số hàm độ đo hiệu suất $J(\theta)$ đối với tham số chính sách. Các phương pháp này tìm cách tối đa hóa hiệu suất, vì vậy công thức cập nhật xấp xỉ dựa trên hướng tăng của gradient trong J:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla J(\boldsymbol{\theta}_t)}\,.$$

trong đó $\widehat{\nabla J(\boldsymbol{\theta}_t)}$ là một ước lượng ngẫu nhiên mà kỳ vọng của nó xấp xỉ gradient của độ đo hiệu suất đối với $\boldsymbol{\theta}_t$. Các phương pháp tuân theo lược đồ chung này được gọi là phương pháp gradient chiến lược (policy gradient methods).

Xấp xỉ policy (policy approximation)

Mục tiêu của xấp xỉ chính sách $\pi_\theta(s, a)$ là ta cần tìm tham số tối ưu $\boldsymbol{\theta}$, do đó ta cần một hàm số đo độ tốt của chính sách $\pi_{\boldsymbol{\theta}}$ đang được ước lượng bởi các giá trị thuộc vector $\boldsymbol{\theta}$. Trong báo cáo này, ta xét với môi trường chia theo từng tập (tức luôn có một trạng thái kết thúc *terminal state* ở cuối mỗi tập). Đối với môi trường loại này, một hàm dùng để đánh giá độ tốt của chính sách $\pi_{\boldsymbol{\theta}}$ là hàm lợi nhuận theo tập *Episodic-return objective:*

$$\begin{aligned} J_{\mathrm{G}}(\boldsymbol{\theta}) &= \mathbb{E}_{S_0 \sim d_0, \pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right] \\ &= \mathbb{E}_{S_0 \sim d_0, \pi_\theta}\left[G_0\right] \\ &= \mathbb{E}_{S_0 \sim d_0}\left[\mathbb{E}_{\pi_\theta}\left[G_t \mid S_t = S_0\right]\right] \\ &= \mathbb{E}_{S_0 \sim d_0}\left[v_{\pi_\theta}\left(S_0\right)\right] \end{aligned} \tag{1.4}$$

với d_0 là phân phối xác suất biểu diễn trạng thái bắt đầu. Giá trị của $J_{\mathrm{G}}(\boldsymbol{\theta})$ chính là kỳ vọng của hàm giá trị trạng thái bắt đầu từ S_0.

Policy Gradient

Bài toán đặt ra là ta cần tìm một tham số $\boldsymbol{\theta}$ để cực đại hàm $J(\boldsymbol{\theta})$. Một cách tiếp cận đơn giản và hiệu quả là dùng phương pháp *stochastic gradient ascent* phương pháp hướng tăng ngẫu nhiên. Ý tưởng

chính của phương pháp là tăng giá trị đạo hàm của $J(\boldsymbol{\theta})$:

$$\Delta\boldsymbol{\theta} = \alpha\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

với $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ là gradient của chính sách được định nghĩa như sau:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_n} \end{pmatrix}$$

và α là tham số thể hiện bước nhảy của thuật toán. Công việc còn lại là tính toán $\nabla_{\boldsymbol{\theta}}(J(\boldsymbol{\theta})$ dựa trên hàm khả vi $\boldsymbol{\theta}$ (tính khả vi có được khi ta tham số hóa $\boldsymbol{\theta}$ bằng mạng *neural*). Dựa vào công thức (1.4), ta có:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R] \tag{1.5}$$

Contextual Bandit Policy Gradient

Ta xét trong trường hợp 1 bước lặp tức chuyển từ thời gian t sang $t+1$, với giá trị cho cặp state-action (S, A) cụ thể thì giá trị $\nabla_{\boldsymbol{\theta}}\mathbb{E}[R(S, A)]$ được tính thông qua đại lượng $r_{sa} = \mathbb{E}[R(S, A)|S = s, A = a]$ như đã trình bày ở thuật toán REINFORCE [16] như sau:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R(S, A)] &= \nabla_{\boldsymbol{\theta}} \sum_s d(s) \sum_a \pi_{\boldsymbol{\theta}}(a \mid s) r_{sa} \\
&= \sum_s d(s) \sum_a r_{sa} \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a \mid s) \\
&= \sum_s d(s) \sum_a r_{sa} \pi_{\boldsymbol{\theta}}(a \mid s) \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_{\boldsymbol{\theta}}(a \mid s)} \\
&= \sum_s d(s) \sum_a \pi_{\boldsymbol{\theta}}(a \mid s) r_{sa} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a \mid s) \\
&= \mathbb{E}_{d,\pi_{\boldsymbol{\theta}}} \left[R(S, A) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A \mid S)\right]
\end{aligned}$$

Với $d(s)$ là phân phối biểu thị xác suất tác tử đến các trạng thái s. Từ việc lấy mẫu và các phân phối d, ta có thể đơn giản hóa $\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R(S, A)]$ như sau:

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}[R(S, A)] = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A \mid S) R(S, A)\right]$$

Thuật toán chính sách gradient ngẫu nhiên sẽ cập nhật tham số $\boldsymbol{\theta}$ sau mỗi bước lặp như sau:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha R_{t+1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}_t}\left(A_t \mid S_t\right).$$

Để làm giảm phương sai của việc dự đoán các $\mathbb{E}[R(S, A)]$ người ta thường dùng một hàm *baseline* cơ sở như sau:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha\left(R_{t+1} - b\left(S_t\right)\right) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}_t}\left(A_t \mid S_t\right).$$

Định lý Policy Gradient

Định lý 1.3.2 *Với mọi hàm chính sách khả vi $\pi_{\boldsymbol{\theta}}(s,a)$, cho d_0 là phân phối xác suất cho trạng thái bắt đầu mỗi tập. Khi đó gradient của chính sách $J(\boldsymbol{\theta}) = \mathbb{E}\left[G_0 \mid S_0 \sim d_0\right]$ là:*

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T} \gamma^t q_{\pi_{\boldsymbol{\theta}}}\left(S_t, A_t\right) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(A_t \mid S_t\right) \mid S_0 \sim d_0\right]$$

với

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma q_\pi\left(S_{t+1}, A_{t+1}\right) \mid S_t = s, A_t = a\right] \end{aligned}$$

Chứng minh: Xét chuỗi lịch sử tương tác giữa tác tử và môi trường là: $\tau = S_0, A_0, R_1, S_1, A_1, R_1, S_2, \ldots$ với lợi nhuận ký hiệu là $G(\tau)$, ta có:

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi) = \nabla_{\boldsymbol{\theta}} \mathbb{E}[G(\tau)] = \mathbb{E}\left[G(\tau) \nabla_{\boldsymbol{\theta}} \log p(\tau)\right]$$

Ta lại có:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \log p(\tau) &= \nabla_{\boldsymbol{\theta}} \log\left[p\left(S_0\right) \pi\left(A_0 \mid S_0\right) p\left(S_1 \mid S_0, A_0\right) \pi\left(A_1 \mid S_1\right) \cdots\right] \\ &= \nabla_{\boldsymbol{\theta}}\left[\log p\left(S_0\right) + \log \pi\left(A_0 \mid S_0\right) + \log p\left(S_1 \mid S_0, A_0\right) + \log \pi\left(A_1 \mid S_1\right) + \cdots\right] \\ &= \nabla_{\boldsymbol{\theta}}\left[\log \pi\left(A_0 \mid S_0\right) + \log \pi\left(A_1 \mid S_1\right) + \cdots\right] \end{aligned}$$

Do đó:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi) &= \mathbb{E}_\pi\left[G(\tau) \nabla_{\boldsymbol{\theta}} \sum_{t=0}^{T} \log \pi\left(A_t \mid S_t\right)\right] \\ &= \mathbb{E}_\pi\left[G(\tau) \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi\left(A_t \mid S_t\right)\right] \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{T} G(\tau) \nabla_{\boldsymbol{\theta}} \log \pi\left(A_t \mid S_t\right)\right] \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{T}\left(\sum_{k=0}^{T} \gamma^k R_{k+1}\right) \nabla_{\boldsymbol{\theta}} \log \pi\left(A_t \mid S_t\right)\right] \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{T}\left(\sum_{k=t}^{T} \gamma^k R_{k+1}\right) \nabla_{\boldsymbol{\theta}} \log \pi\left(A_t \mid S_t\right)\right] \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{T}\left(\gamma^t \sum_{k=t}^{T} \gamma^{k-t} R_{k+1}\right) \nabla_{\boldsymbol{\theta}} \log \pi\left(A_t \mid S_t\right)\right] \\ &= \mathbb{E}_\pi\left[\sum_{t=0}^{T} \gamma^t q_\pi\left(S_t, A_t\right) \nabla_{\boldsymbol{\theta}} \log \pi\left(A_t \mid S_t\right)\right] \end{aligned}$$

Định lý đã được chứng minh.

Để làm giảm phương sai của việc dự đoán hàm $q_\pi(S_t, A_t)$, người ta thường dùng hàm cơ sở là $v_\pi(S_t)$ như sau: $v_\pi\left(S_t\right)$

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t\left(q_\pi\left(S_t, A_t\right) - v_\pi\left(S_t\right)\right) \nabla_{\boldsymbol{\theta}} \log \pi\left(A_t \mid S_t\right)\right]$$

Từ đó, ta bỏ tổng xích ma ra ngoài và chia *gradient* thành các *gradient* thành phần như sau

$$\Delta\boldsymbol{\theta}_t = \gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi\left(A_t \mid S_t\right)$$

sao cho $\mathbb{E}_\pi\left[\sum_t \Delta\boldsymbol{\theta}_t\right] = \nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi)$. Và từ đó ta có công thức để cập nhật cho $\boldsymbol{\theta}$.

Thuật toán Actor-Critic

Ý tưởng của thuật toán *Actor-Critic* trong việc tìm chính sách tối ưu như sau: Khối *Actor* có nhiệm vụ *exploitation* tận dụng các kiến thức đã biết của tác tử để thực hiện các hành động đem lại phần thưởng lớn nhất cho từng bước sau đó dùng khối *Critic* để tối ưu chính sách một cách lâu dài dựa trên việc cập nhật các hàm giá trị trạng thái đã được tham số hóa.

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

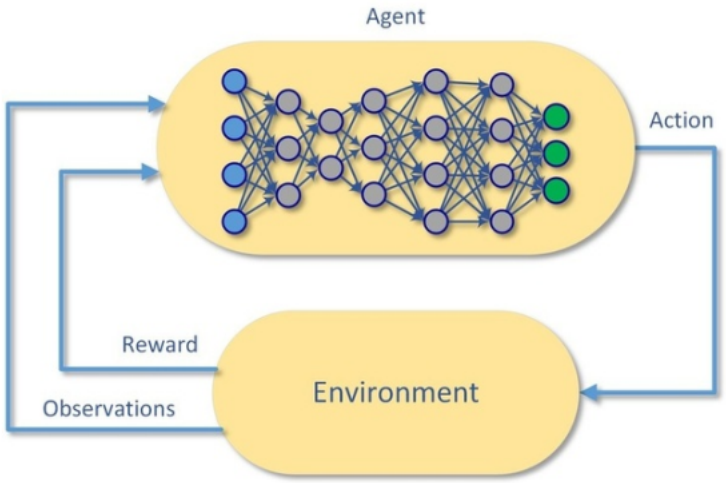
Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S,\boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Hình 1.10: Thuật toán Actor-Critic 1 bước lặp

Xét phiên bản đơn giản của thuật toán Actor-Critic cho 1 bước lặp được trình bày như ở Hình (1.10), hàm chính sách sẽ được điều khiển thông qua tham số $\boldsymbol{\theta}$ và hàm giá trị trạng thái sẽ được tham số hóa bởi $\boldsymbol{w}$. Trong đó, ta xấp xỉ G_t bởi $R_{t+1} + \gamma V(S_{t+1})$ và đại lượng δ được gọi là sai số *Temporal difference* trong RL.

1.3.4 Deep Reinforcement Learning

Mô hình *Học sâu tăng cường* là sự kết hợp của Học sâu và Học tăng cường với đặc điểm chính của mô hình là thay vì lưu trữ các cặp trạng thái - hành động và phần thưởng thì ta sẽ mô hình hóa các thông tin đó dưới dạng các mạng Neural. Việc làm này giúp giảm thiểu dung lượng lưu trữ trong trường hợp số lượng trạng thái và hành động lớn, cũng như tăng tốc độ huấn luyện của mô hình.



Hình 1.11: Deep Reinforcement Learning

Để có thể giải các bài toán điều khiển tối ưu trên thực tế, các phương pháp *Học tăng cường* cần có một phương pháp lưu trữ hiệu quả số lượng trạng thái lớn của môi trường cũng như cần tăng độ hiểu quả tính toán trong việc "học" giá trị từ các trạng thái. Chính vì lý do đó, các hướng tiếp cận xấp xỉ sử dụng các mạng *Học sâu* ra đời với đặc điểm:

- Ánh xạ các trạng thái s đến các hàm mô tả"đặc trưng" $\phi(s)$ một cách hợp lý.
- Tiến hành tham số hóa các hàm đặc trưng $v_\theta(\phi)$.
- Cập nhật tham số θ dẫn tới cập nhật các *value function* cho chính sách $v_\pi(s) \sim v_\theta(\phi(s))$

Mục tiêu của DRL là tìm ra tham số θ cực tiểu sai lệch giữa v_π và v_θ. Độ sai lệch giữa hai đại lượng trên được minh họa bởi:

$$L(\theta) = E_{S\sim d}\left[\left(v_\pi(S) - v_\theta(S)\right)^2\right]$$

với d là phân phối lượt truy cập các trạng thái được thực hiện theo chính sách π. Chúng ta có thể dùng các phương pháp hướng giảm để tối ưu hóa hàm mục tiệu:

$$\Delta\theta = -\frac{1}{2}\alpha\nabla_\theta L(\theta) = \alpha E_{S\sim d}\left[\left(v_\pi(S) - v_\theta(S)\nabla_\theta v_\theta(S)\right]\right.$$

Việc tham số v_θ bởi các *mạng Neural* để dễ dàng tính toán, lưu trữ là ý tưởng chính cho phương pháp DRL. Từ đó, việc đánh giá các chính sách sử dụng phương pháp *stochastic gradient descent* sẽ được thực hiện với các hàm giá trị v_θ.

1.4 Bài toán tối ưu tổ hợp đa mục tiêu

1.4.1 Giới thiệu chung

Bài toán tối ưu tổ hợp được miêu tả bởi 3 thành phần (S, f, ω) trong đó:

- S là tập các lời giải.
- f là hàm đánh giá (hàm này gán giá trị f(s) cho mỗi lời giải ứng cử viên $s \in S$).
- ω là tập các ràng buộc của bài toán.

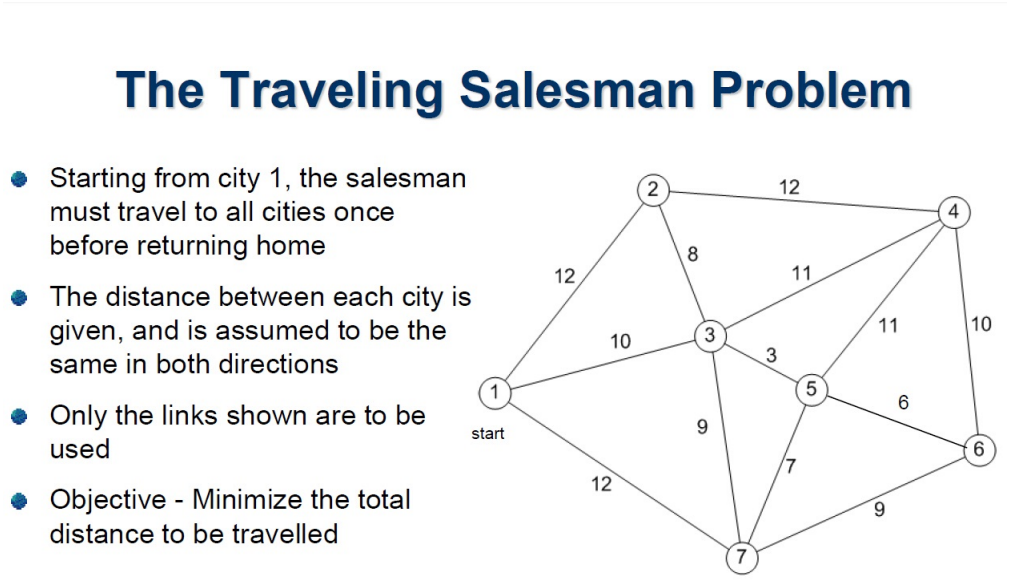
Các lời giải thuộc tập $S^* \subseteq S$ thỏa mãn tập các ràng buộc ω gọi là lời giải khả thi. Mục tiêu bài toán là tìm ra một lời giải khả thi tối ưu toàn cục s^*. Với các bài toán tối ưu cực tiểu là tìm lời giải s^* với giá nhỏ nhất, nghĩa là $f(s^*) \leq f(s)$ với mọi lời giải $s \subseteq S$. Ngược lại bài toán tối ưu cực đại là tìm lời giải s^* với giá lớn nhất, nghĩa là $f(s^*) \geq f(s)$ với mọi lời giải $s \subseteq S$. Bài toán tối ưu tổ hợp có thể chia 2 loại: Bài toán tĩnh và bài toán động.

1. **Bài toán tối ưu tổ hợp tĩnh (Static Combinatorial optimization):** Là bài toán tối ưu tổ hợp trong đó cấu trúc (topology) và giá (cost) không thay đổi khi bài toán được giải quyết. Ví dụ là bài toán TSP khi thực hiện thuật toán để giải bài toán vị trí các thành phố, khoảng cách giữa các thành phố là không thay đổi.
2. **Bài toán tối ưu tổ hợp động (Dynamic Combinatorial optimization):** Là bài toán tối ưu tổ hợp trong đó cấu trúc (topology) và giá (cost) có thể thay đổi khi bài toán đang được giải quyết. Ví dụ bài toán định tuyến trong mạng viễn thông, trong đó mô hình mạng và dung lượng yêu cầu luôn thay đổi.

Bài toán tối ưu tổ hợp đa mục tiêu là phiên bản khó hơn của bài toán tối ưu tổ hợp thông thường, khi đó các hàm f đánh giá sẽ ở dạng đa mục tiêu: $f = (f_1, f_2, \ldots, f_p)$ với p là số hàm đánh giá cần tối ưu.

1.4.2 Bài toán người du lịch (TSP)

Bài toán *TSP* là một bài toán lâu đời và có nhiều ứng dụng trong thực tế, đặc biệt là đối với ngành khoa học dữ liệu và phân tích tài chính. Các công ty vận tải sẽ thu thập thông tin những hành trình đã hoàn thành của nhân viên giao hàng và tiến hành phân tích dữ liệu từ đó đưa ra các tuyến đường ngắn hơn cho nhân viên thông qua việc sử dụng các mô hình *Học máy* để phân tích dữ liệu.



Hình 1.12: Minh họa TSP

Như mô tả tại Hình (1.12), bài toán *TSP* được phát biểu cụ thể như sau: Cho trước một đồ thị n đỉnh $\{0, 1, \ldots, n-1\}$ và giá trị trọng số $C_{i,j}$ trên mỗi cặp đỉnh i, j, ta cần tìm một chu trình đi qua tất cả các cạnh của đồ thị, mỗi đỉnh đúng một lần sao cho tổng các trọng số trên chu trình đó là nhỏ nhất. Đây là bài toán thuộc lớp *NP-Hard* nên không tồn tại thuật toán tất định thời gian đa thức hiện có để giải bài toán này.

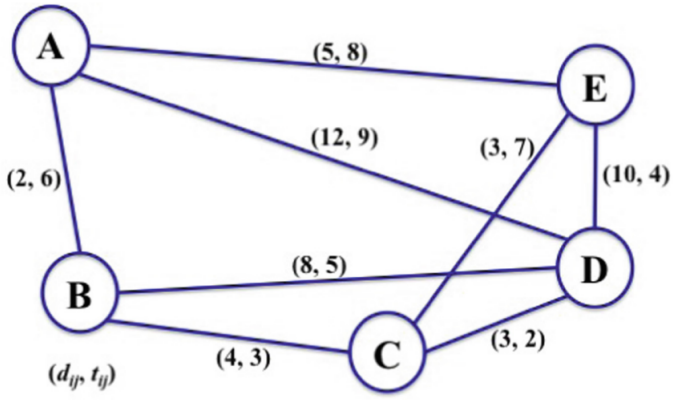
Một số thuật toán giải chính xác thường được sử dụng để bài toán TSP ví dụ như:

- Thuật toán duyệt toàn bộ *Brute Force* tiến hành duyệt toàn bộ các hoán vị các đỉnh cho độ phức tạp là $\Theta(n!)$ nhưng chỉ hiệu quả với số thành phố $n \leq 11$.
- Thuật toán giải chính xác tốt nhất hiện nay cho bài toán TSP là thuật toán Quy hoạch động với độ phức tạp thuật toán là $\Theta(2^n n^2)$. Việc truy vết các thành phố đã đi qua và chạy thuật toán Quy hoạch động là vô cùng tốn kém khi số thành phố $n \geq 20$ do hiện tượng bùng nổ tổ hợp.

Đặc điểm chung của những thuật toán giải chính xác trên là chi phí tính toán rất lớn trong trường hợp số thành phố $n \geq 20$. Do đó, các thuật toán heuristic và các mô hình *Học máy* đang là hướng tiếp cận mới để giải bài toán này. Trong báo cáo này, tác giả sẽ trình bày phương pháp *Học tăng cường* để giải phiên bản khó hơn của bài toán TSP là bài toán TSP đa mục tiêu (MOTSP).

1.4.3 Bài toán người du lịch đa mục tiêu (MOTSP)

Bài toán MOTSP là phiên bản đa mục tiêu của bài toán TSP ở đó việc di chuyển giữa các thành phố không chỉ xét tới yếu tố khoảng cách mà còn xét tới nhiều loại chi phí khác nhau.



Hình 1.13: Minh họa MOTSP

Bài toán MOTSP được phát biểu cụ thể như sau: Cho một đồ thị n đỉnh tương ứng với các thành phố trên thực tế. Việc di chuyển giữa 2 thành phố i và j tốn M loại hàm chi phí khác nhau. Ta cần tìm 1 chu trình đi qua n thành phố sao cho tổng giá trị của M hàm chi phí là nhỏ nhất.

Các phương pháp tiếp cận để giải bài toán MOTSP thông thường sẽ gồm các *giải thuật tiến hóa MOEA* và các giải thuật tìm kiếm địa phương *Local Search* [11]. Trong báo cáo này, tác giả sẽ trình bày hướng tiếp cận giải bài toán MOTSP bằng phương pháp *Học tăng cường*.

1.4.4 Giới thiệu giải thuật tiến hóa giải các bài toán tối ưu tổ hợp đa mục tiêu

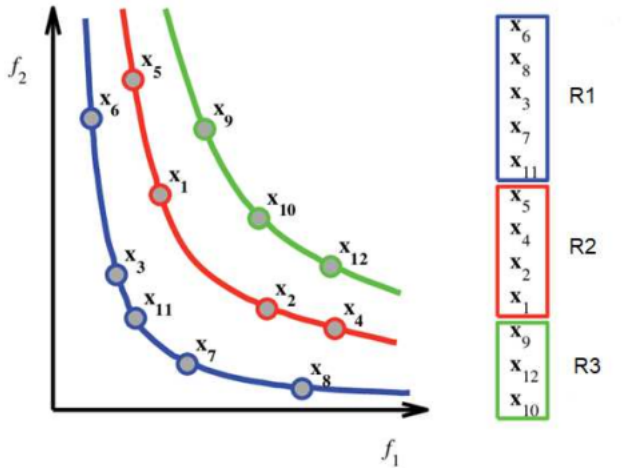
Trong phần này, tác giả sẽ giới thiệu hai thuật toán tiến hóa giải bài toán tối ưu đa mục tiêu phổ biến đó là NSGA-II [3] và MOEA/D [17] và giới thiệu các thành phần trong mô thức chung của các thuật toán tính toán tiến hóa như toán tử chọn lọc, toán tử lai ghép,...

Thuật toán NSGA-II

NSGA-II là phương pháp giải bài toán tối ưu hóa đa mục tiêu sử dụng giải thuật di truyền kết hợp sắp xếp nghiệm không trội. Các đặc điểm nổi bật của thuật toán nằm ở: phương pháp dùng để gán độ thích nghi, cách duy trì quần thể ưu tú, cách tiếp cận nhằm đa dạng hóa quần thể.

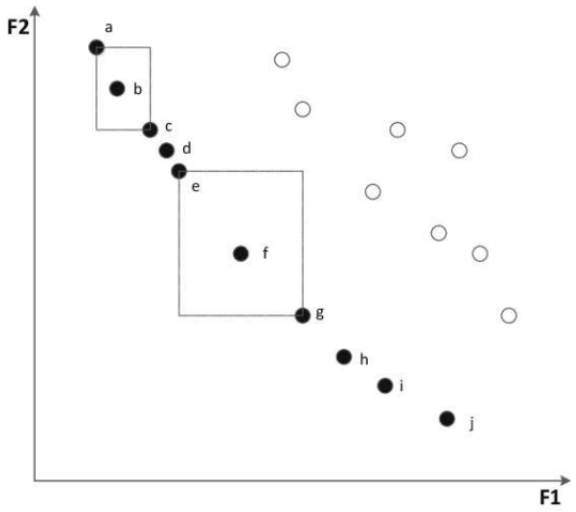
Để gán độ thích nghi cho một cá thể ta gán cho cá thể thứ hạng theo các biên không trội mà cá thể đó thuộc vào dựa vào sắp xếp không trội (*Non-dominated sorting*). Độ thích nghi của một cá thể sau khi sắp xếp được xác định như sau:

- Nghiệm hay cá thể nào nằm trên biên thứ nhất - R1 thì có độ thích nghi cao nhất và tất cả các nghiệm này được gán là hạng thứ 1.
- Tất cả các nghiệm nằm trên cùng một biên chứa các nghiệm không trội thì có cùng độ thích nghi và chúng có cùng thứ hạng.



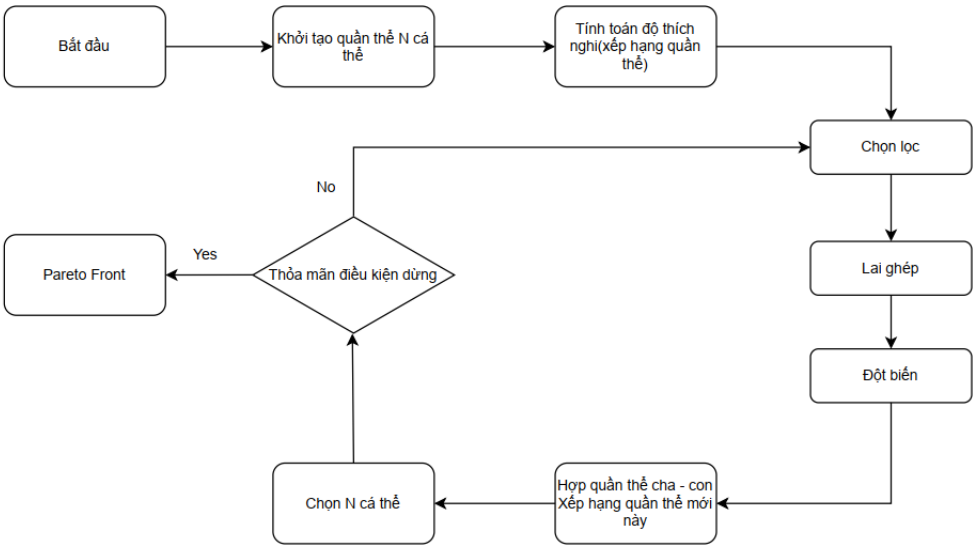
Hình 1.14: Sắp xếp không trội

Để xem xét các nghiệm liền kề với nó trên cùng biên không trội có gần nó hay không, ta sẽ sử dụng độ đo theo khoảng cách quy tụ *(Crowding Distance)*. Khoảng cách quy tụ của cá thể hay nghiệm nằm trên một biên là chiều dài trung bình các cạnh của một hình hộp được minh họa như hình vẽ:



Hình 1.15: Khoảng cách quy tụ

Sau đây là lược đồ thuật toán NSGA-II:



Hình 1.16: Lược đồ thuật toán NSGA-II

Các thành phần của lược đồ (1.16) cụ thể như sau:

- **Toán tử chọn lọc**: Trong bước chọn ra N cá thể cho quần thể mới, ta có quy tắc chọn như sau: Giữa 2 nghiệm không trội chọn nghiệm có thứ hạng thấp hơn vàk hi 2 nghiệm không trội có cùng thứ hạng nghĩa là 2 nghiệm này nằm trên cùng một biên, nghiệm nào có khoảng cách quy tụ lớn hơn thì sẽ được ưu tiên lựa chọn

- **Phương pháp bảo tồn cá thể tối ưu:** Trong thuật toán NSGA II có áp dụng phương pháp bảo tồn và nhân bản những cá thể ưu tú, trong sơ đồ trên ta có thể thấy phương pháp này được thể hiện ở bước gộp 2 quần thể cha ban đầu và quần thể con sau khi lai ghép và đột biến sau đó chọn ra N cá thể. Bằng cách này, những cá thể ưu tú nhất sẽ không bị loại khỏi quần thể sau bước chọn lọc và do đó giữ lại được những gen tốt cho thế hệ tiếp theo.

- **Cách mã hóa một cá thể trong thuật toán NSGA-II:** Cách mã hóa thường được sử dụng là mã hóa số thực (real-coded GA). Với một nghiệm có số chiều là D sẽ được biểu diễn như cách biểu diễn một vectơ thực $x = (x_1, x_2, ..., x_D)$. Với mỗi cách biểu diễn nghiệm, cần phải có một toán tử lai ghép và đột biến phù hợp. Một trong những toán tử lai ghép và đột biến thường được sử dụng trong NSGA-II với cách mã hóa thực là Simulated Binary Crossover (SBX) và Polynomial Mutation.

- **Các toán tử trong NSGA-II:**

  1. **Toán tứ lai ghép SBX:** Quy trình tính toán cá thể con $x_i^{(1,t+1)}$ và $x_i^{(2,t+1)}$ từ cá thể cha mẹ $x_i^{(1,t)}$ và $x_i^{(2,t)}$ được mô tả như sau. Một hệ số chênh lệch β là tỉ lệ giữa sự khác biệt tuyệt đối

của của hai cá thể con và hai cá thể cha mẹ được tính bằng:

$$\beta = \left| \frac{x_i^{2,t+1} - x_i^{1,t+1}}{x_i^{2,t} - x_i^{1,t}} \right|.$$

Trước tiên, một số u ngẫu nhiên được sinh trong đoạn $[0, 1]$. Sau đó sử dụng một hàm phân phối xác suất dưới đây để tìm $\bar{\beta}$ sao cho diện tích dưới đường cong phân phối từ 0 đến $\bar{\beta}$ bằng với u :

$$\mathbb{P}(\beta) = \begin{cases} 0.5(n+1)\beta^n, & \text{nếu } \beta \leq 1 \\ 0.5(n+1)\frac{1}{\beta^{n+2}}, & \text{nếu } \beta > 1 \end{cases}$$

n là một số thực không âm bất kỳ. Giá trị n lớn cho xác suất cao hơn để tạo các giá trị con gần cha mẹ và giá trị nhỏ của n cho phép các điểm ở xa được chon làm cá thể con.

Sau khi thu được $\bar{\beta}$ từ phân phối trên, các cá thể con được sinh ra như sau:

$$x_i^{(1,t+1)} = 0.5\left[(1+\bar{\beta})x_i^{(1,t)} + (1-\bar{\beta})x_i^{(2,t)}\right]$$
$$x_i^{(2,t+1)} = 0.5\left[(1-\bar{\beta})x_i^{(1,t)} + (1+\bar{\beta})x_i^{(2,t)}\right]$$

Hai cá thể con đối xứng nhau về các nghiệm cha mẹ, giúp tránh sự thiên vị đối với bất kỳ cá thể cha mẹ cụ thể nào. Một khía cạnh thú vị khác của toán tử chéo này là đối với một phân phối xác suất cố định nếu các giá trị cha mẹ ở xa nhau thì có thể tạo ra các giá trị con ở xa nhau, nhưng nếu các giá trị của cha mẹ gần nhau, các giá trị con sẽ gần nhau. Ban đầu khi các cá thể được khởi tạo là ngẫu nhiên, điều này cho phép khảo sát toàn bộ không gian tìm kiếm, nhưng khi các cá thể có xu hướng hội tụ do tác động của các toán tử di truyền, các cá thể ở xa không được sinh ra, do đó tập trung tìm kiếm vào một vùng hẹp.

2. **Toán tử đột biến:**Đối với một thành phần của vectơ x, giá trị hiện tại của biến được thay đổi thành giá trị lân cận bằng cách sử dụng một phân phối xác suất, một yếu tố nhiễu loạn được xác định như sau:

$$\delta = \frac{c-p}{\Delta_{\max}},$$

trong đó c là giá trị đột biến và $\Delta_{\max}$ là đại lượng cố định, đại diện cho nhiễu tối đa cho phép ở giá trị cha mẹ p. Tương tự với toán tử lai ghép, giá trị đột biến được tính toán với phân phối xác suất phụ thuộc vào hệ số nhiễu δ như sau:

$$\mathbb{P}(\delta) = 0.5(n+1)(1-|\delta|)^n.$$

Phân phối xác suất trên thỏa mãn với $\delta \in (-1, 1)$ và n là một số thực không âm.

Để tạo một giá trị đột biến, một số ngẫu nhiên u được tạo trong khoảng $(0, 1)$. Sau đó, tính hệ số nhiễu loạn $\bar{\delta}$ tương ứng với u bằng cách sử dụng phân phối xác suất ở trên:

$$\bar{\delta} = \begin{cases} (2u)^{\frac{1}{n+1}} - 1, & \text{nếu } u < 0.5 \\ 1 - [2(1-u)]^{\frac{1}{n+1}}, & \text{nếu } u \geq 0.5 \end{cases}$$

Sau đó giá trị đột biến được tính toán như sau:

$$c = p + \bar{\delta}\Delta_{\max}.$$

Độ phức tạp của thuật toán với M là số hàm mục tiêu, N là kích cỡ quần thể ta có: Thuật toán sắp xếp không trội có độ phức tạp là $O(M(2N)^2)$ Thuật toán tính khoảng cách quy tụ có độ phức tạp là $O(M(2N)\log(2N))$. Sắp xếp theo khoảng cách quy tụ có độ phức tạp là $O(2N\log(2N))$ Như vậy độ phức tạp tổng thể của thuật toán NSGA II trong một lần tạo quần thể là $O(MN^2)$.

Thuật toán MOEA/D

Phát biểu bài toán MOP đang xét:

$$\text{maximize } F(x) = (f_1(x), \ldots, f_m(x))^T \text{ v.đ.k } x \in \Omega$$

Trong phần này, tác giả giới thiệu thuật toán tiến hóa đa mục tiêu dựa trên phân rã *(Multiobjective evolutionary algorithm based on decomposition)* (MOEA/D). Ý tưởng chính của thuật toán là việc phân rã bài toán tối ưu đa mục tiêu thành bài toán vô hướng với các điều kiện rằng buộc. Trong bài báo [17], các tác giả sử dụng phương pháp phân rã Tchebycheff như sau:

Cho $\lambda^1, \ldots, \lambda^N$ là tập vector trọng số và z^* là một điểm tham chiếu. Bài toán tối ưu đa mục tiêu ban đầu sẽ được phân rã thành N bài toán con vô hướng, hàm mục tiêu của bài toán con thứ j được định nghĩa là:

$$g^{te}\left(x \mid \lambda^j, z^*\right) = \max_{1 \le i \le m} \left\{ \lambda_i^j \left| f_i(x) - z_i^* \right| \right\}$$

với $\lambda^j = \left(\lambda_1^j, \ldots, \lambda_m^j\right)^T$. Thuật toán MOEA/D sẽ cực tiểu hóa N hàm mục tiêu g.

Bằng thực nghiệm cho thấy, giá trị tối ưu của $g^{te}\left(x \mid \lambda^i, z^*\right)$ sẽ gần với nghiệm tối ưu của $g^{te}\left(x \mid \lambda^j, z^*\right)$ nếu λ^i và λ^j gần nhau. Do đó, các thông tin từ các hàm g^{te} có vector trọng số gần với λ^i sẽ có ích trong việc giải bài toán $g^{te}\left(x \mid \lambda^i, z^*\right)$.

Trong thuật toán MOEA/D, một lân cận của vector trọng số λ^i được định nghĩa là tập các vector trọng số gần nó nhất xét trong tập $\{\lambda_1, \ldots, \lambda_N\}$. Lân cận của bài toán thứ i bao gồm tất cả các bài toán con có vector trọng số là lân cận của λ^i. Các nghiệm tối ưu của bài toán lân cận sẽ là nghiệm khởi đầu cho bài toán con đang xét.

Ở mỗi bước lặp t, thuật toán MOEA/D với phương pháp phân rã Tchebycheff sẽ gồm các thành phần:

- Một tập N điểm $x^1, \ldots, x^N \in \Omega$ với Ω là tập chấp nhận được và x^i là lời giải tốt nhất hiện thời cho bài toán con thứ i;
- $FV^1, \ldots, FV^N$, với FV^i là giá trị hàm F tại điểm x^i ví dụ $FV^i = F\left(x^i\right)$ với mỗi $i = 1, \ldots, N$;
- $z = (z_1, \ldots, z_m)^T$ với z_i là giá trị tốt nhất tìm được bởi hàm f_i;
- Một tập bên ngoài *external population* (EP) chứa các điểm không trội tìm thấy trong quá trình lặp.

Khi đó thuật toán thực hiện như sau:

Input:

- Điều kiện dừng của thuật toán;
- N : Số bài toán con sau khi phân rã;
- Phân phối đều N vector trọng số: $\lambda^1, \ldots, \lambda^N$;
- T : Số lượng vector trọng số lân cận với vector trọng số đang xét.

Output: Tập FP.

- **Bước 1 (Thiết lập Initialization):** Đặt EP $= \emptyset$, sau đó tính khoảng cách Euclide cho các cặp vector trọng số và chọn ra T vector trọng số gần vector trọng số đang xét nhất với mỗi $i = 1, \ldots, N$, đặt tập $B(i) = \{i_1, \ldots, i_T\}$, với $\lambda^{i_1}, \ldots, \lambda^{i_T}$ là T vector trọng số gần λ^i nhất. Tiếp theo khởi tạo ngẫu nhiên hoặc bằng một phương pháp cụ thể tập $x^1, \ldots, x^N$. Đặt $FV^i = F\left(x^i\right)$. Cuối cùng, thiết lập $z = (z_1, \ldots, z_m)^T$ bằng phương pháp cụ thể.
- **Bước 2 (Cập nhật):** Lặp từ $i = 1, \ldots, N$, các hành động sau:
  1. Sinh sản (*Reproduction*): Chọn ngẫu nhiên 2 chỉ số k, l từ $B(i)$ và sinh ra một lời giải y mới từ x^k và x^l bằng toán tử lai ghép.
  2. Cải thiện (*Improvement*): Sử dụng một giải thuật cụ thể để chỉnh sửa và nâng cấp y thành lời giải y' của bài toán.
  3. Cập nhật giá trị z : Với mỗi $j = 1, \ldots, m$, nếu $z_j < f_j(y')$ thì cập nhật $z_j = f_j(y')$.
  4. Cập nhật các lời giải lân cận: Với mỗi chỉ số $j \in B(i)$ nếu $g^{te}\left(y' \mid \lambda^j, z\right) \leq g^{te}\left(x^j \mid \lambda^j, z\right)$, thì đặt $x^j = y'$ và $FV^j = F(y')$.
  5. Cập nhật tập EP :Loại bỏ khỏi EP tất cả các vector bị trội bởi $F(y')$. Thêm $F(y')$ vào tập EP nếu không vector nào trong EP trội hơn $F(y')$.
- **Bước 3 (Kiểm tra điều kiện dừng)**: Nếu điều kiện dừng thỏa mãn thì xuất ra tập EP và dừng vòng lặp. Ngược lại, quay lại **Bước 2**.

Chương 2

Ứng dụng Deep Reinforcement Learning giải bài toán MOTSP

Trong chương này, tác giả sẽ trình bày cách giải bài toán TSP và MOTSP bằng phương pháp Deep Reinforcement Learning, cụ thể ta đi tìm hiểu cách xây dựng mạng Pointer và cách thức luyện Actor-Critic của DRL để xây dựng chính sách tối ưu hay tìm được đường đi có tổng trọng số nhỏ nhất trên đồ thị có số thành phố lớn. Cách giải bài toán TSP bằng phương pháp DRL được tham khảo từ bài báo [1] và cách giải bài toán MOTSP bằng DRL được tham khảo từ bài báo [9].

2.1 Deep Reinforcement Learning giải bài toán TSP

Chúng ta xét bài toán TSP trong không gian Euclidean 2D chiều. Cho trước một đồ thị đầu vào biểu diễn như là 1 chuỗi n thành phố trong không gian 2 chiều như sau: $s = \{\mathbf{x}_i\}_{i=1}^{n}$ với mỗi $\mathbf{x}_i \in \mathbb{R}^2$. Bài toán đặt ra là chúng ta cần tìm một tổ hợp của chuỗi là π hay thuật ngữ gọi là *chu trình* sao cho mỗi thành phố chỉ được thăm đúng một lần và chu trình đó có tổng độ dài (trọng số) nhỉ nhất. Ta định nghĩa độ dài của 1 chu trình π là:

$$L(\pi \mid s) = \left\|\mathbf{x}_{\pi(n)} - \mathbf{x}_{\pi(1)}\right\|_2 + \sum_{i=1}^{n-1} \left\|\mathbf{x}_{\pi(i)} - \mathbf{x}_{\pi(i+1)}\right\|_2,$$

với $\|\cdot\|_2$ biểu thị cho chuẩn ℓ_2. Ta cần xác định tham số của chính sách ngẫu nhiên $P(\pi \mid s)$ sao cho khi cho dữ liệu đầu vào là chuỗi s thì chính sách đó gán cho các chu trình có độ dài ngắn xác suất cao xảy ra và ngược lại. Mô hình Pointer Network sẽ sử dụng công thức tích các sự kiện để biểu diễn xác suất xảy ra 1 chu trình như sau:

$$P(\pi \mid s) = \prod_{i=1}^{n} P(\pi(i) \mid \pi(< i), s)$$

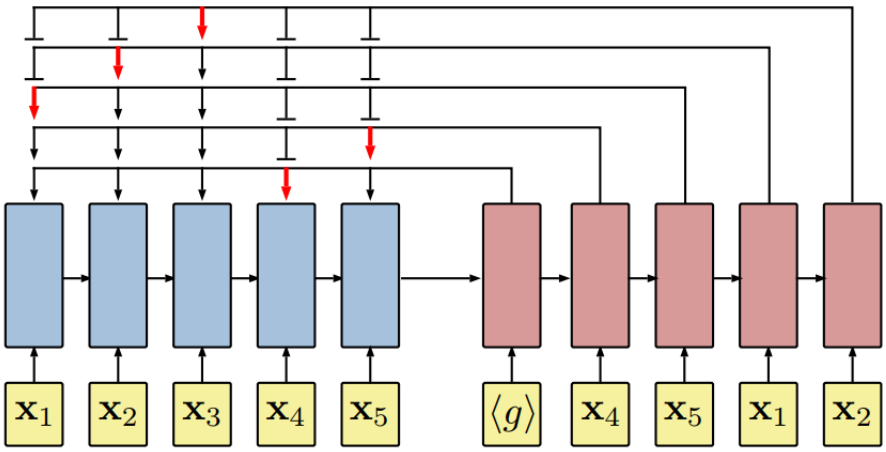
Ta có thể tham số hóa $P(\pi|s)$ và học được chính sách tối ưu thông qua mạng Pointer được trình bày cụ thể ở phần tiếp theo.

Kiến trúc mạng Pointer

Ý tưởng chính của mạng Pointer : Mạng Pointer sẽ tiến hành tham số hóa các thành phần sau:

- θ : là bộ tham số mạng Pointer cần tối ưu bằng cách sử dụng mô hình không dựa trên model mà dựa vào các quy tắc (model-free policy-based Reinforcement Learning) [14].
- $\pi(i)$: là thành phố đi tới ở bước thứ i được quy định bởi chính sách π.
- $\pi_i \leftarrow x_j$: Gán thành phố x_j vào chu trình ở bước thứ i theo chính sách π.
- L: là độ dài chu trình, trong bài toán DRL thì độ dài giữa hai thành phố chính là phần thưởng nhận được, độ dài hai thành phố càng ngắn thì hàm phần thưởng có giá trị càng lớn.
- $P(\pi(i) \mid \pi(< i), s)$: là xác suất xảy ra việc thăm quan các thành phố theo chính sách π.
- π : là một chính sách và mạng Pointer qua huấn luyện sẽ học được chính sách tối ưu.
- s: là môi trường hay chính là input đầu vào gồm tọa độ 2 chiều của các thành phố.

Mục đích của mạng Pointer là để học được bộ tham số $P(\pi(i) \mid \pi(< i), s)$ với chỉ số $i = 1, \ldots, n$ bằng cách tối ưu tham số θ.



Hình 2.1: Kiến trúc mạng Pointer

Mạng Pointer sẽ gồm 2 khối mạng *RNN* là Enconder dùng để đọc chuỗi input và Decoder dùng để dự đoán chuỗi output chính là thứ tự thăm các thành phố. Cả 2 khối trên đều bao gồm các mạng LSTM, cụ thể như sau:

- **Khối Encoder:** Khối Encoder đọc chuỗi đầu vào s, mỗi lần 1 thành phố và chuyển nó thành chuỗi các trạng thái nhớ ẩn *(latent memory)* dưới dạng $\{enc_i\}_{i=1}^{n}$ với $enc_i \in \mathbb{R}^d$. Đầu vào cho mạng Encoder ở bước thứ i là một mảng d chiều ứng với thành phố x_i. Bước biến đổi này thông qua một hàm biến đổi *(Embedding)* tuyến tính được dùng cho mọi bước lặp của mạng biến đổi từ tọa độ $\mathbf{x}_i$ 2 chiều sang mảng d chiều.

- **Khối Decoder:** Khối Decoder cũng lưu trữ các trạng thái nhớ ẩn dưới dạng $\{\text{dec}_i\}_{i=1}^{n}$ với $dec_i \in \mathbb{R}^d$ và tại mỗi bước lặp i mạng sử dụng một cơ chế trỏ *pointing mechanism* để sinh ra một phân phối đưa ra xác suất thăm các thành phố tiếp theo. Khi một thành phố đã được thăm thì nó sẽ được chuyển thành input cho bước lặp tiếp theo của khối Decoder. Input dữ liệu đầu tiên của khối Decoder, được ký hiệu bởi $\langle g \rangle$ trong Hình 2.1) là một vector d chiều được coi như tham số huấn luyện của mạng Neural chứa thông tin tóm gọn của khối Encoder.

Cơ chế trỏ là đặc trưng của mạng *Pointer* giúp đưa ra phân phối xác suất thăm các thành phố cho từng bước lặp. Đồng thời, để tăng hiệu suất tính phân phối xác suất cho mô hình bằng việc kết hợp mức độ mô hình "quan tâm" hay trỏ tới một phần của dữ liệu input thì mạng *Pointer* sẽ sử dụng thêm hàm *glimpse* được trình bày ở phần dưới đây.

Cơ chế trỏ và cơ chế chú ý

Cơ chế trỏ được biểu thị thông qua hàm *attention* nhận input đầu vào là 1 vector truy vấn $q = dec_i \in \mathbb{R}^d$ (dữ liệu trạng thái ẩn của khối Decoder) và 1 dãy các vector tham chiếu ref $= \{enc_1, \ldots, enc_k\}$ với $enc_i \in \mathbb{R}^d$ (dữ liệu trạng thái ẩn của khối Encoder).

Cơ chế trỏ: Việc tính toán của nó được tham số hóa qua 2 ma trận *attention* $W_{ref}, W_q \in \mathbb{R}^{d\times d}$ và một vector *attention* $v \in \mathbb{R}^d$ được tính như sau:

$$u_i = \begin{cases} v^\top \cdot \tanh\left(W_{\text{ref}} \cdot ref_i + W_q \cdot q\right) & \text{nếu } i \neq \pi(j) \text{ với mọi } j < i \\ -\infty & \text{ngược lại} \end{cases} \quad \text{với } i = 1, 2, \ldots, k \tag{2.1}$$

Hàm *attention* được định nghĩa như sau:

$$A\left(\text{ref}, q; W_{ref}, W_q, v\right) \stackrel{\text{def}}{=} \text{softmax}(u).$$

Hàm này có tác dụng tính xác suất thăm thành phố $\pi(j)$ được quy định bởi chính sách π, cụ thể như sau:

$$P(\pi(j) \mid \pi(< j), s) \stackrel{\text{def}}{=} A\left(enc_{1:n}, \text{dec}_j\right)$$

Việc đặt trường hợp $u_i = -\infty$ như ở (2.1) biểu thị mô hình của chúng ta chỉ trỏ tới các thành phố chưa được thăm trong chu trình giúp thỏa mãn yêu cầu của bài toán TSP.

Cơ chế chú ý (Attending mechanism): Hàm *attention* sẽ tiến hành dự đoán phân phối xác suất thăm các thành phố được biểu thị qua $A(ref, q)$ khi có k truy vấn. Đồng thời, phân phối này được tính thông qua hàm *glimpse* với sự kết hợp các tham chiếu ref_i biểu thị tỷ lệ mô hình "quan tâm" tới truy vấn q. Hàm *Glimpse* $G(ref, q)$ nhận dữ liệu input truyền vào như hàm *attention* A và được tham số hóa bởi các ma trận $W_{ref}^g, W_q^g \in \mathbb{R}^{d\times d}$ và $v^g \in \mathbb{R}^d$ và được tính toán như sau:

$$P = A\left(\text{ ref }, q; W_{ref}^g, W_q^g, v^g\right)$$

$$G\left(\text{ ref }, q; W_{ref}^g, W_q^g, v^g\right) \stackrel{\text{def}}{=} \sum_{i=1}^{k} ref_i P_i.$$

Hàm *glimpse* G được tính thông qua tổ hợp tuyến tính trọng số các vector tham chiếu với xác suất P_i được tính thông qua hàm *attention* từ đó hàm *glimpse* giúp mô hình tính tỷ lệ "quan tâm" tới truy vấn q. Hàm này có thể được sử dụng nhiều lần trên cùng bộ tham chiếu ref:

$$g_0 \overset{\text{def}}{=} q$$
$$g_l \overset{\text{def}}{=} G\left(ref, g_{l-1}; W^g_{ref}, W^g_q, v^g\right)$$

Cuối cùng, giá trị vector g_l vector được nạp vào hàm *attention* $A\left(ref, g_l; W_{ref}, W_q, v\right)$ để tính xác suất cho cơ chế trỏ, quá trình thực nghiệm chỉ ra rằng việc sử dụng cơ chế tham dự làm tăng hiệu suất của mô hình.

Quy tắc luyện Actor-Critic

Việc huấn luyện mô hình Pointer để giải bài toán TSP theo kiểu *Học có giám sát supervised* thường tốn chi phí cực lớn trong việc gán nhãn dữ liệu, việc gán nhãn các chu trình tốt với đồ thị dày với nhiều đỉnh là điều cực kỳ tốn kém chính vì vậy ta cần huấn luyện mô hình mạng *Pointer* bằng các phương pháp *Học không giám sát unsupervised* và trong bài báo [1], họ tiến hành huấn luyện theo phướng pháp *model-free policy-based*, cụ thể là phương pháp *policy gradient* để huấn luyện mạng *Pointer* đưa ra tham số $\boldsymbol{\theta}$ tối ưu. Hàm mục tiêu của mô hình là hàm trung bình độ dài chu trình khi cho trước chuỗi input s, được định nghĩa như sau:

$$J(\boldsymbol{\theta} \mid s) = \mathbb{E}_{\pi \sim p_\theta(.|s)} L(\pi \mid s). \tag{2.2}$$

Trong suốt quá trình huấn luyện, các chuỗi input được lấy mẫu từ phân phối $\mathcal{S}$, do đó hàm tổng mục tiêu huấn luyện được tính như sau:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{s \sim \mathcal{S}} J(\boldsymbol{\theta} \mid s).$$

Theo thuật toán REINFORCE [16], đạo hàm của (2.2) được tính như sau:

$$\nabla_\theta J(\theta \mid s) = \mathbb{E}_{\pi \sim p_\theta(.|s)} \left[(L(\pi \mid s) - b(s)) \nabla_\theta \log p_\theta(\pi \mid s) \right], \tag{2.3}$$

Bằng việc lấy B mẫu i.i.d. ta được các chuỗi biểu diễn các đồ thị input như sau: $s_1, s_2, \ldots, s_B \sim \mathcal{S}$ sau đó tiến hành sinh ra một chu trình ứng với mỗi đồ thị, ví dụ $\pi_i \sim p_\theta\left(. \mid s_i\right)$, đạo hàm ở công thức (2.3) được tính thông qua phương pháp Monte Carlo như sau:

$$\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{i=1}^{B} \left(L\left(\pi_i \mid s_i\right) - b\left(s_i\right) \right) \nabla_\theta \log p_\theta\left(\pi_i \mid s_i\right).$$

Ta chọn hàm *baseline* $b(s)$ là dạng trung bình tăng theo hàm mũ *(exponential moving average)* của các phần thưởng mà mạng thu được để biểu diễn chính sách được huấn luyện tốt dần theo thời gian huấn luyện. Việc sử dụng hàm baseline để xấp xỉ độ dài trung bình của chu trình $\mathbb{E}_{\pi \sim p_\theta(\cdot|s)} L(\pi \mid s)$ bằng thực nghiệm đã chứng tỏ tăng độ hiệu quả cho mô hình. Do đó, chúng ta sẽ xây dựng thêm mạng *Critic* được tham số hóa bởi θ_v để xấp xỉ độ dài chu trình trung bình đang được xét tới bởi chính sách p_θ khi cho trước chuỗi dữ liệu s. Mạng *critic* được huấn luyện bởi phương pháp *stochastic gradient descent* thông

qua hàm trung bình bình phương sai số *(mean squared error)* được tính giữa giá trị hàm baseline $b_{\theta_v}(s)$ mà mạng *Critic* đang dự đoán và độ dài thật sự của chu trình được tính thông qua chính sách gần nhất vừa được lấy mẫu, cụ thể:

$$\mathcal{L}(\theta_v) = \frac{1}{B}\sum_{i=1}^{B} \|b_{\theta_v}(s_i) - L(\pi_i \mid s_i)\|_2^2 .$$

Kết hợp quá trình lấy mẫu cho s, π, xây dựng mạng *Pointer* và mạng *Critic* qua T bước huấn luyện cũng như sử dụng phương pháp *policy gradient* để huấn luyện mô hình thì ta thu được quy tắc luyện *Actor-Critic* được trình bày tại thuật toán (2.2) dưới đây:

Algorithm 1 Actor-critic training

procedure TRAIN(training set S, number of training steps T, batch size B)
  Initialize pointer network params θ
  Initialize critic network params θ_v
  for $t = 1$ to T **do**
    $s_i \sim$ SAMPLEINPUT(S) for $i \in \{1, \ldots, B\}$
    $\pi_i \sim$ SAMPLESOLUTION($p_\theta(.|s_i)$) for $i \in \{1, \ldots, B\}$
    $b_i \leftarrow b_{\theta_v}(s_i)$ for $i \in \{1, \ldots, B\}$
    $g_\theta \leftarrow \frac{1}{B}\sum_{i=1}^{B}(L(\pi_i|s_i) - b_i)\nabla_\theta \log p_\theta(\pi_i|s_i)$
    $\mathcal{L}_v \leftarrow \frac{1}{B}\sum_{i=1}^{B}\|b_i - L(\pi_i)\|_2^2$
    $\theta \leftarrow$ ADAM(θ, g_θ)
    $\theta_v \leftarrow$ ADAM($\theta_v, \nabla_{\theta_v}\mathcal{L}_v$)
  end for
  return θ
end procedure

Hình 2.2: Thuật toán luyện Actor-Critic

2.2 Deep Reinforcement Learning giải bài toán MOTSP

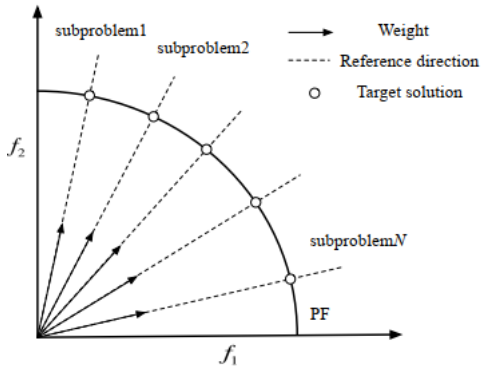
Trong phần này, tác giả xin trình bày mô thức chung để giải bài toán MOTSP được trình bày trong bài báo [9] đó là DLR-MOA. Mô thức này gồm hai bước chính là sử dụng phương pháp phân rã (decomposition) để chuyển bài toán tối ưu đa mục tiêu thành các bài toán con và mỗi bài toán con đó sẽ được mô hình hóa bằng một mạng neural. Các tham số của các mạng neural sau đó được kết hợp lại thông qua chiến lược tham số lân cận (neighborhood-based parameter transfer) và cuối cùng được huấn luyện thông qua quy tắc luyện Actor-Critic của Học tăng cường.

2.2.1 Mô thức chung của thuật toán DRL-MOA

Chiến lược phân rã

Chiến lược phân rã là một chiến lược thường được dùng trong việc giải các bài toán tối ưu đa mục tiêu ([10], [2]). Ý tưởng chính của chiến lược này là chuyển bài toán tối ưu đa mục tiêu ban đầu thành các bài toán vô hướng con và giải lần lượt các bài toán con đó và kết hợp nghiệm Pareto của các bài toán con này để thu được Pareto Front của bài toán ban đầu. Trong bài báo [9], các tác giả sử dụng

phương pháp vô hướng hóa để tiến hành giải bài toán MOTSP. Tiến hành thiết lập bộ vector trọng số đều $\lambda^1, \ldots, \lambda^N$ cho N bài toán con sau khi phân rã. Ví dụ đối với bài toán tối ưu 2 mục tiêu như trong Hình 2.3, bộ trọng số có thể là $(1,0), (0.9,0.1), \ldots, (0,1)$.. Cụ thể, $\lambda^j = \left(\lambda_1^j, \ldots, \lambda_M^j\right)^T$, với M biểu diễn số mục tiêu cần tối ưu.



Hình 2.3: Chiến lược phân rã

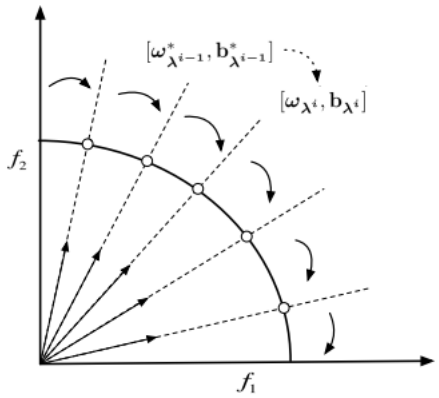
Bài toán tối ưu đa mục tiêu ban đầu được phân rã thành N bài toán tối ưu vô hướng con bằng phương pháp vô hướng hóa. Hàm mục tiêu của bài toán con thứ j được biểu diễn như sau:

$$\min g^{ws}\left(x \mid \lambda_i^j\right) = \sum_{i=1}^{M} \lambda_i^j f_i(x) \tag{2.4}$$

Cuối cùng, đường cong nghiệm *Pareto Front* được xác định bằng cách kết hợp nghiệm tối ưu của N bài toán con này.

Chiến lược tham số lân cận

Tiến hành giải N bài toán con vô hướng bằng cách mô hình chúng dưới dạng các mạng *Neural*. Bằng các chứng minh và thực nghiệm của bài báo [17] thì việc giải các bài toán con ở ((2.4)) cho nghiệm tối ưu gần nhau nếu như các vector trọng số liền kề. Do đó, một bài toán con có thể được giải khi biết trước nghiệm tối ưu từ bài toán con lân cận của nó.



Hình 2.4: Chiến lược tham số lân cận

Các tham số của mạng *Neural* tương ứng với bài toán con thứ $(i-1)$ có thể được biểu diễn dưới dạng $[\boldsymbol{\omega}_{\lambda^{i-1}}, \mathbf{b}_{\lambda^{i-1}}]$. Ký hiệu $[\boldsymbol{\omega}^*, \mathbf{b}^*]$ biểu thị tham số của mạng *Neural* đã tối ưu và $[\boldsymbol{\omega}, \mathbf{b}]$ biểu thị cho các tham số chưa tối ưu. Giả sử bài toán $i-1$ đã được giải và cho ra nghiệm tối ưu xấp xỉ khi đó tham số $\left[\boldsymbol{\omega}^*_{\lambda^{i-1}}, \mathbf{b}^*_{\lambda^{i-1}}\right]$ được dùng làm tham số ban đầu khi bắt đầu huấn luyện mạng *Neural* tương ứng cho bài toán con thứ i. Bằng thực nghiệm đã chỉ ra rằng chiến lược tham số lân cận giúp giải N bài toán tối ưu con nhanh chóng hiệu quả hơn phương pháp giải lần lượt N bài toán con đó. Dưới đây là thuật toán giải bài toán MOTSP theo mô thức chung của thuật toán DRL-MOA

Algorithm 1 General Framework of DRL-MOA

Input: The model of the subproblem $\mathcal{M} = [\boldsymbol{w}, \mathbf{b}]$, weight vectors $\boldsymbol{\lambda^1}, \ldots, \boldsymbol{\lambda^N}$

Output: The optimal model $\mathcal{M}^* = [\boldsymbol{w}^*, \mathbf{b}^*]$

1: $[\boldsymbol{\omega}_{\boldsymbol{\lambda}^1}, \mathbf{b}_{\boldsymbol{\lambda}^1}] \leftarrow Random\_Initialize$
2: **for** $i \leftarrow 1 : N$ **do**
3: **if** $i == 1$ **then**
4: $[\boldsymbol{\omega}^*_{\boldsymbol{\lambda}^1}, \mathbf{b}^*_{\boldsymbol{\lambda}^1}] \leftarrow Actor\_Critic([\boldsymbol{\omega}_{\boldsymbol{\lambda}^1}, \mathbf{b}_{\boldsymbol{\lambda}^1}], g^{ws}(\boldsymbol{\lambda^1}))$
5: **else**
6: $[\boldsymbol{\omega}_{\boldsymbol{\lambda}^i}, \mathbf{b}_{\boldsymbol{\lambda}^i}] \leftarrow [\boldsymbol{\omega}^*_{\boldsymbol{\lambda}^{i-1}}, \mathbf{b}^*_{\boldsymbol{\lambda}^{i-1}}]$
7: $[\boldsymbol{\omega}^*_{\boldsymbol{\lambda}^i}, \mathbf{b}^*_{\boldsymbol{\lambda}^i}] \leftarrow Actor\_Critic([\boldsymbol{\omega}_{\boldsymbol{\lambda}^i}, \mathbf{b}_{\boldsymbol{\lambda}^i}], g^{ws}(\boldsymbol{\lambda^i}))$
8: **end if**
9: **end for**
10: **return** $[\boldsymbol{w}^*, \mathbf{b}^*]$
11: Given inputs of the MOP, the PF can be directly calculated by $[\boldsymbol{w}^*, \mathbf{b}^*]$.

Hình 2.5: Mô thức chung của thuật toán DRL-MOA

Khi đã giải được N bài toán con thì đường cong nghiệm *Pareto Front* sẽ được giải thông qua việc chạy *forward* mô hình một lần, đồng thời với bộ tham số tối ưu của mô hình thì ta không cần phải huấn luyện lại mô hình cho các *test case* khác hay các đồ thị mới giúp tiết kiệm chi phí tính toán. Việc mô

hình hóa cho các bài toán con và tiến hành giải chúng thông qua thuật toán *Actor-Critic* được trình bày cụ thể trong phần dưới.

2.2.2 Mô hình hóa các bài toán con của MOTSP

Tương tự như trong bài báo [1] và phần 2.1 đã trình bày, mạng *Pointer* được điều chỉnh để mô hình hóa cho các bài toán vô hướng con đồng thời quy tắc luyện *Actor-Critic* sẽ được sử dụng.

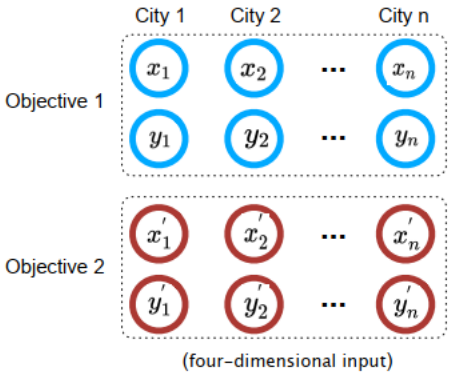
Ta cần tìm chu trình đi qua n thành phố được ký hiệu dưới dạng 1 hoán vị tuần hoàn ρ, để tối ưu M hàm mục tiêu khác nhau, hàm chi phí (*cost function*) sẽ được phát biểu như sau:

$$\min z_k(\rho) = \sum_{i=1}^{n-1} c^k_{\rho(i),\rho(i+1)} + c^k_{\rho(n),\rho(1)}, k = 1, \ldots, M. \tag{2.5}$$

với $c^k_{\rho(i),\rho(i+1)}$ là hàm chi phí thứ k_{th} của việc chuyển từ thành phố $\rho(i)$ đến thành phố $\rho(i+1)$ theo chính sách ρ.

Cấu trúc Input-Output dữ liệu

Tương tự trong bài báo [1], mô hình sẽ có dữ liệu đầu vào là $X \doteq \left\{s^i, i = 1, \cdots, n\right\}$ với n là số thành phố. Mỗi phần tử s^i được biểu diễn dưới dạng chuỗi $\left\{s^i = \left(s^i_1, \cdots, s^i_M\right)\right\}$. Trong đó, s^i_j là thuộc tính của thành phố thứ i được dùng để tính hàm chi phí thứ j, luôn luôn $s^i_1 = (x_i, y_i)$ biểu diễn cặp tọa độ (x, y) của thành phố thứ i và được dùng để tính độ dài giữa hai thành phố.



Hình 2.6: Cấu trúc Input dữ liệu của mô hình

Ví dụ với trường hợp có 2 hàm mục tiêu, ta có biểu diễn cấu trúc input dữ liệu như Hình 2.6.

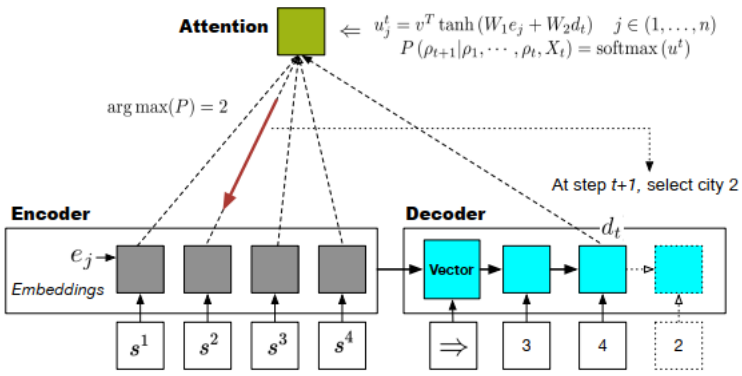
Đầu ra của mô hình là các tổ hợp chu trình di chuyển giữa các thành phố như sau: $Y = \{\rho_1, \cdots, \rho_n\}$. Để ánh xạ Input X đến Output Y thì xác suất được tính thông qua tích các sự kiện như sau:

$$P(Y \mid X) = \prod_{t=1}^{n} P\left(\rho(t+1) \mid \rho(1), \cdots, \rho(t), X_t\right) \tag{2.6}$$

Thành phố đầu tiên được chọn ngẫu nhiên là ρ_1. Ở mỗi bước lặp *decode* $t = 1, 2, \cdots$, ta chọn $\rho(t+1)$ từ thành phố X_t và các thành phố đã thăm $\rho(1), \ldots, \rho(t)$. Thành phố X_t được cập nhật sau mỗi bước lặp của

mô hình. Để tiến hành mô hình hóa cho công thức 2.6, ta sử dụng cấu trúc mạng *Sequence-to-Sequence* gồm 2 khối *Encoder* và *Decoder* tương tự như trong phần 2.1 đã trình bày. Khối Encoder sẽ đọc chuỗi input đầu vào và tổng hợp tạo ra 1 vector *ngữ cảnh (context)* tổng hợp tri thức ban đầu và chuyển vào cho khối *Decoder* để từ đó khối *Decoder* sẽ tạo ra chuỗi mong muốn, đối với bài toán MOTSP là chu trình ρ tối ưu.

Kiến trúc của mạng Pointer



Hình 2.7: Kiến trúc của mạng Pointer

Khối Encoder

Điểm khác biệt trong cách thiết kế khối Encoder này so với phần 2.1 đó là việc sử dụng 1 khối tích chập 1 chiều *1-D convolution layer* để chuyển dữ liệu đầu vào thành 1 vector nhiều chiều d_h (cụ thể, $d_h = 128$). Đầu ra của khối Encoder là mô $n \times d_h$ vector. Việc chỉ sử dụng 1 lớp tích chấp 1 chiều giúp tiết kiệm chi phí tính toán do thứ tự các thành phố trong chuỗi Input không có ý nghĩa nên ta không cần dùng các khối RNN để đọc chuỗi Input đầu vào mà chỉ cần 1 khối tích chập để lưu trữ thông tin. Tham số của khối tích chập 1-D được dùng cho toàn bộ các thành phố.

Khối Decoder

Khối Decoder có tác dụng chuyển thông tin từ chuỗi thành phố đã có $\rho(1), \rho(2), \ldots, \rho(t)$ để tiến hành chọn ra thành phố $\rho(t+1)$. Do đó, khối Decoder cần sử dụng 1 mạng RNN để chắt lọc thông tin từ chuỗi đã dự đoán và input để dự đoán thành phố tiếp theo sẽ tới. Trong bài báo [9], tác giả sử dụng khối GRU để tiết kiệm chi phí tính toán do GRU vẫn giảm được hiện tượng *vanishing gradient* và cũng có ít tham số cần luyện hơn khối LSTM.

Để có thể tính được công thức (2.6), mạng *Pointer* sẽ sử dụng thông tin của các trạng thái nhớ ẩn lưu trữ dữ liệu từ khối Encoder là $e_1, \ldots, e_n$ và thông tin của các trạng thái nhớ ẩn lưu trữ dự liệu tại bước thứ t là d_t của khối Decoder. Việc tính toán được mô hình thực hiện thông qua cơ chế tham dự *Attention mechanism* được trình bày dưới đây.

Cơ chế chú ý

Cơ chế này có tác dụng tính xem độ quan tâm của mô hình tới dữ liệu input đầu vào trong việc sử dụng dữ liệu để dự đoán thành phố ở bước *decode* thứ t. Thành phố càng được "quan tâm" bởi mô hình thì càng dễ được chọn. Cụ thể việc tính toán như sau:

$$u_j^t = v^T \tanh\left(W_1 e_j + W_2 d_t\right) \quad j \in (1, \ldots, n)$$
$$P\left(\rho_{t+1} \mid \rho_1, \cdots, \rho_t, X_t\right) = \text{softmax}\left(u^t\right)$$

trong đó sử dụng các tham số học như vector v và hai ma trận trọng số tương ứng với dữ liệu trạng thái ẩn của khối Encoder và Decoder là W_1, W_2 là các tham số học của mô hình. Dữ liệu trạng thái ẩn d_t là chìa khóa trong việc tính xác suất $P\left(\rho_{t+1} \mid \rho_1, \cdots, \rho_t, X_t\right)$ do nó lưu trữ dữ liệu từ các bước lặp trước $\rho_1, \cdots, \rho_t$.

Trong quá trình huấn luyện, thành phố tiếp theo được chọn khi có xác suất cao nhất được tính theo cơ chế tham dự tuy nhiên mô hình vẫn sẽ chọn thành phố tiếp theo thông qua việc chọn mẫu từ một phân phối xác suất biểu diễn tỷ lệ thăm thành phố. Quá trình này mang ý nghĩa *khám phá exploration* trong *Học tăng cường*.

Quy tắc luyện Actor-Critic

Mô hình sẽ gồm 2 khối *Actor* và *Critic*. Nhiệm vụ của khối *Actor* là chọn các thành phố tối ưu cho từng bước lặp theo chính sách $\epsilon-greedy$ của *Học tăng cường* mang ý nghĩa cho việc *khai thác exploitation*. Khối *Critic* đóng vai trò tối ưu cho toàn bộ chính sách ρ mang ý nghĩa tối ưu dài hạn và đóng vai tròn cho quá trình *khám phá exploration*. Cụ thể như sau:

- Khối Actor tính xác suất thăm thành phố tiếp theo. Khối Actor được mô hình hóa bằng mạng *Pointer*.
- Khối Critic đánh giá phần thưởng trung bình thu được trong quá trình huấn luyện mạng và được mô hình hóa cũng bởi mạng *Pointer*.

Mô hình sẽ được huấn luyện theo kiểu *Học không giám sát unsupervised* do việc gán nhãn dữ liệu như đã trình bày ở phần 2.1 là vô cùng tốn kém. Trong quá trình huấn luyện, ta sẽ tiến hành tạo ra các phân phối xác suất $\{\Phi_{\mathcal{M}_1}, \cdots, \Phi_{\mathcal{M}_M}\}$để dữ liệu được chọn từ các mẫu này. Trong đó, $\mathcal{M}$ biểu diễn cho các đặc trưng khác nhau của dữ liệu input, ví dụ như tọa độ hay độ an toàn của các thành phố. Trong trường hợp cụ thể như bài toán MOTSP 2 mục tiêu cần tối ưu là $\mathcal{M}_1, \mathcal{M}_2$ đều là các đặc trưng biểu thị tọa độ của thành phố được biểu diễn dưới dạng tọa độ Euclide thì chúng có thể có phân phối đều $[0, 1] \times [0, 1]$.

Hai mạng *Actor* và *Critic* được tham số hóa thông qua hai tham số lần lượt là θ và ϕ. Để huấn luyện 2 mạng này thì trong quá trình *training* N ví dụ *(instance)* hay dữ liệu input biểu thị cho N bài toán con được lấy mẫu từ các phân phối $\{\Phi_{\mathcal{M}_1}, \cdots, \Phi_{\mathcal{M}_M}\}$. Với mỗi ví dụ, ta sử dụng khối *Actor* với tham số θ hiện tại để tạo ra chu trình qua n thành phố và tính giá trị phần thưởng thu được. Sau đó sử dụng *policy*

gradient (chi tiết công thức đạo hàm có thể xem tại bài báo [8]) để cập nhật lại tham số θ. $V(X_0^k;\phi)$ là phần thưởng xấp xỉ cho bài toán con thứ k được tính bởi khối *Critic*.

Algorithm 2 Actor-Critic training algorithm

Input: $\theta, \phi \leftarrow$ initialized parameters given in Algorithm 1
Output: The optimal parameters θ, ϕ
for $iteration \leftarrow 1, 2, \cdots$ **do**
  generate T problem instances from $\{\Phi_{\mathcal{M}_1}, \cdots, \Phi_{\mathcal{M}_M}\}$ for the MOTSP.
  for $k \leftarrow 1, \cdots, T$ **do**
    $t \leftarrow 0$
    while not terminated **do**
      select the next city ρ_{t+1}^k according to $P\left(\rho_{t+1}^k | \rho_1^k, \cdots, \rho_t^k, X_t^k\right)$
      Update X_t^k to X_{t+1}^k by leaving out the visited cities.
    end while
    compute the reward R^k
  end for
  $d\theta \leftarrow \frac{1}{N} \sum_{k=1}^{N} \left(R^k - V\left(X_0^k;\phi\right)\right) \nabla_\theta \log P\left(Y^k | X_0^k\right)$
  $d\phi \leftarrow \frac{1}{N} \sum_{k=1}^{N} \nabla_\phi \left(R^k - V\left(X_0^k;\phi\right)\right)^2$
  $\theta \leftarrow \theta + \eta d\theta$
  $\phi \leftarrow \phi + \eta d\phi$
end for

Hình 2.8: Quy tắc luyện Actor-Critic

Thời gian độ phức tạp (*time complexity)* thuật toán DRL-MOA để giải bài toán MOTSP xấp xỉ là $O\left(Nnd_h^2\right)$ với N là số bài toán con.

Chương 3

Cài đặt và tính toán thử nghiệm

Trong chương này, tác giả sẽ tiến hành cài đặt thuật toán Quy hoạch động giải chính xác bài toán TSP, cài đặt thuật toán DRL-MOA và các thuật toán tiến hóa đa mục tiêu để giải bài toán MOTSP và so sánh kết quả chạy của các thuật toán trên.

3.1 Cài đặt mô hình

3.1.1 Cài đặt thuật toán Quy hoạch động giải đúng TSP

Thuật toán Quy hoạch động được cài đặt bằng ngôn ngữ $C++$ dựa vào việc sử dụng *BitMask*. Thuật toán Quy hoạch động được khái quát qua các bước sau:

- Không mất tính tổng quát giả sử chu trình bắt đầu và kết thúc tại đỉnh 0.
- Gọi $TSP(i, S)$ là cách sử dụng chi phí ít nhất để đi qua toàn các đỉnh và quay trở lại đỉnh 0, nếu như hiện tại hành trình đang ở tại đỉnh i và người du lịch đã thăm tất cả các đỉnh trong S.
- Bước cơ sở: $TSP(i, \text{tập mọi đỉnh}) = C_{i,0}$.
- Bước chuyển quy nạp: $TSP(i, S) = \min_{j \notin S}(C_{i,j} + TSP(j, S \cup \{j\}))$.

Cài đặt:

```
double tsp(double arr[][V], int dp[][V], int n, int visited_all, int mask, int pos)
{
   if (mask == visited_all){ //Khi da tham toan bo thanh pho
      return arr[pos][0];
   }
   if (dp[mask][pos] != -1){
      return dp[mask][pos];
   }
```

```
    double ans = 10000000;
    for (int city =0; city < n; city++){
        if((mask & (1<<city))== 0){
            double newAns = arr[pos][city] + tsp(arr, dp, n, visited_all, mask|(1<<city), city);
            ans = min(ans, newAns);
        }
    }
    return dp[mask][pos] = ans;
}
```

Bằng thực nghiệm cho thấy thuật toán làm việc kém hiệu quả với các đồ thị dày với số đỉnh lớn hơn hoặc bằng 20.

3.1.2 Cài đặt thuật toán DRL-MOA

Để có thể triển khai thuật toán DRL-MOA trên thực tế, ta cần thực hiện hai nhiệm vụ:

1. Thiết kế khối Actor-Crtic.
2. Khởi tạo dữ liệu "không nhãn" do ta đang tiến hành huấn luyện mạng theo kiểu học không giám sát.

Tác giả đã tiến hành tinh chỉnh lại code nguồn của bài báo [9] để cài đặt code Python cho bài toán tại link sau `https://bit.ly/3oi4mEk`.

Thành phần mạng Neural của khối Actor-Critic

Thông số chi tiết của các khối mạng Neural như khối tích chập 1 chiều, khối GRU được trình bày dưới đây:

Actor network(Pointer Network)	
Encoder:	1D-Conv(D_{input}, 128, kernel size=1, stride=1)
Decoder:	GRU(hidden size=128, number of layer=1)
	Attention(No hyper parameters)
Critic network	
1D-Conv(D_{input}, 128, kernel size=1, stride =1)	
1D-Conv(128, 20, kernel size=1, stride =1)	
1D-Conv(20, 20, kernel size=1, stride =1)	
1D-Conv(20, 1, kernel size=1, stride =1)	

Hình 3.1: Tham số của mô hình

Việc tiến hành huấn luyện 2 khối Actor và Critic đều sử dụng phương pháp tối ưu Adam [7] với hệ số học *learning rate* $\eta = 0.0001$ và độ dài 1 *batch* là 200. Việc khởi tạo tham số ban đầu cho mạng được tiến hành khởi tạo theo phương pháp Xavier [5].

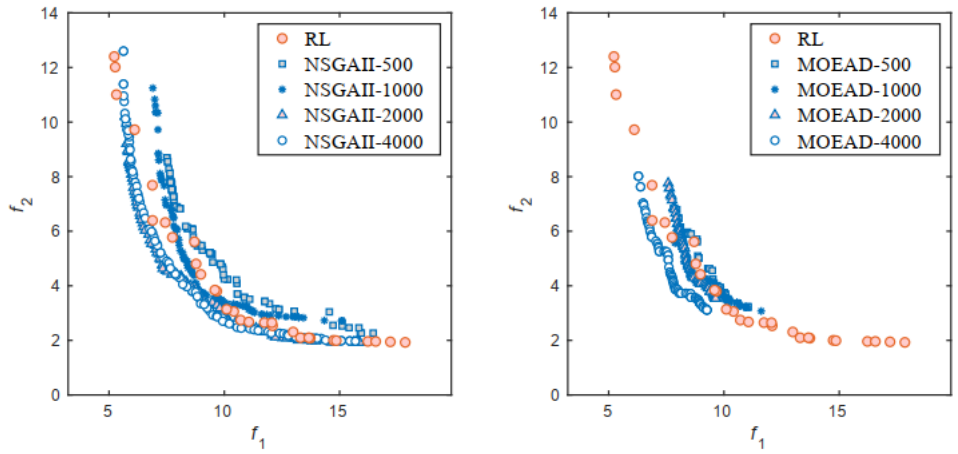
Khởi tạo dữ liệu

Trong báo cáo này, tác giả sẽ tiến hành cài đặt mô hình và thử nghiệm với loại ví dụ trộn *Mixed instances* để thử nghiệm khả năng thích ứng của mô hình với cấu trúc input dữ liệu khác nhau, cụ thể là dữ liệu 3 chiều với số thành phố cần đi qua lần lượt là $40, 100, 150, 200$ thành phố. Mô hình sẽ tiến hành tối ưu 2 hàm mục tiêu với hàm mục tiêu thứ nhất là độ dài giữa hai thành phố được đo bằng cặp tọa độ (x, y) ứng với 2 chiều đầu tiên của dữ liệu, hàm mục tiêu thứ hai là đo độ cao giữa hai thành phố h ứng với chiều dữ liệu cuối cùng.

Tập dữ liệu huấn luyện *training set* sẽ được thiết kế dựa trên việc lấy mẫu từ mẫu phân phối đều trên đoạn $[0, 1]$. Do mô hình được luyện theo kiểu không giám sát nên quá trình huấn luyện chỉ cần dữ liệu 3 chiều mà chúng ta tự tạo và hàm phần thưởng được tính toán dựa trên khoảng cách Euclide giữa 2 thành phố và độ cao giữa 2 thành phố và không cần nhãn của dữ liệu. Tập dữ liệu kiểm tra *Test set* sẽ được lấy từ hai bộ dữ liệu chuyên dụng cho bài toán TSB là kroA và kroB trong thư viện TSPLIB [13], chúng được dùng để xây dựng các ví dụ *instances* cho biết khoảng cách Euclide giữa 2 thành phố A và B. Chiều cao của các thành phố sẽ được sinh ngẫu nhiên.

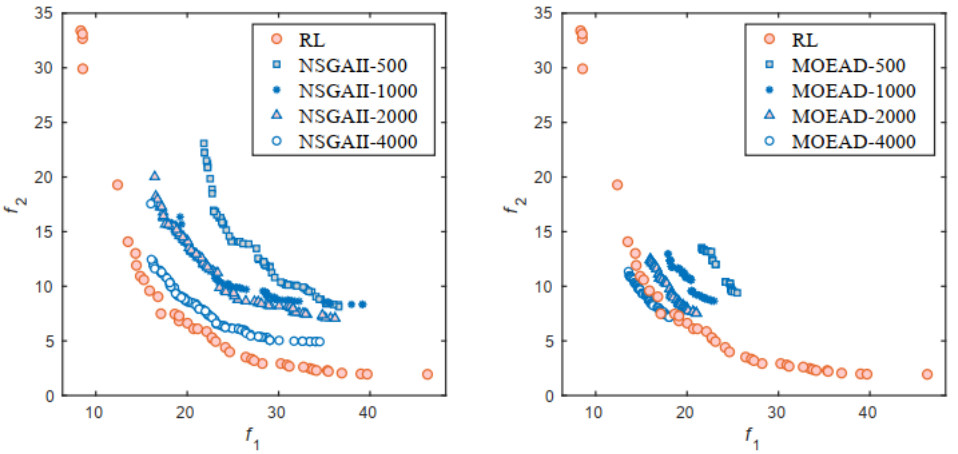
3.2 Kết quả thực nghiệm và đánh giá mô hình

Trong báo cáo này, tác giả sẽ tiến hành cài đặt lại mô hình và tính toán trong trường hợp *Mixed instances* của dữ liệu với 2 hàm mục tiêu cần tối ưu. Mô hình sẽ tìm *Pareto Front* trong các trường hợp đồ thị có $40, 100, 150, 200$ đỉnh. Thuật toán DRL-MOA sẽ được so sánh với hai loại thuật toán NSGA-II [3] và MOEA/D [17]. Việc tính toán thử nghiệm 2 thuật toán NSGA-II và MOEA/D sẽ được thực hiện trên nền tảng PLATEMO [15] được chạy trên nền ngôn ngữ lập trình MATLAB và 2 thuật toán trên sẽ được thực thi với số lần lặp lần lượt là $500, 1000, 2000$ và 4000. Sau đây là các hình ảnh so sánh thuật toán DRL-MOA với hai thuật toán còn lại.



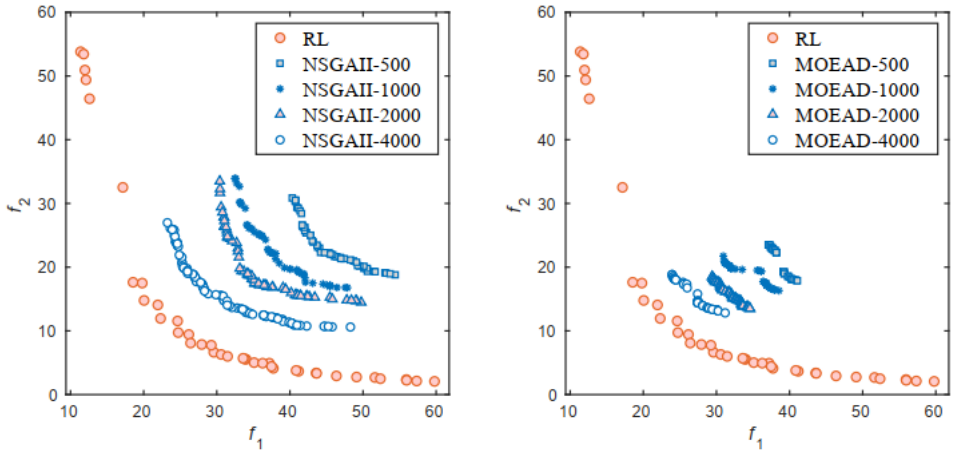
(a) DRL-MOA and NSGA-II (b) DRL-MOA and MOEA/D

Hình 3.2: Kết quả so sánh giữa các thuật toán trong trường hợp 40 thành phố



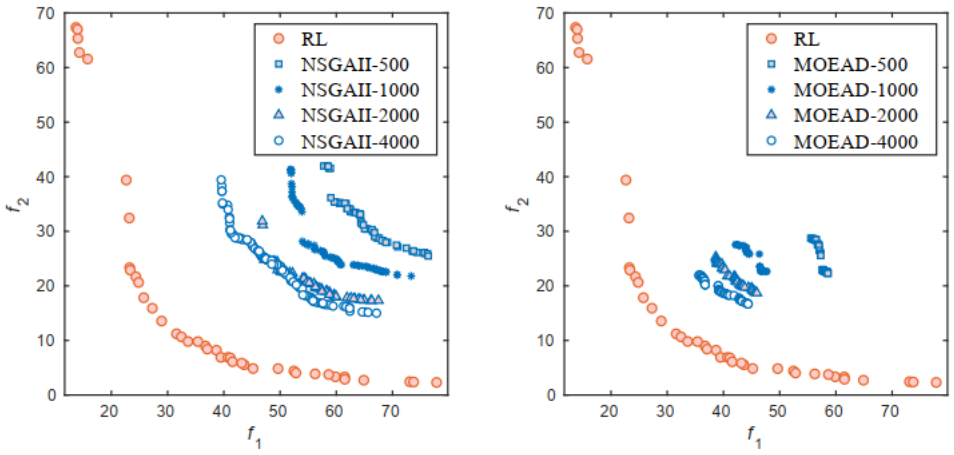
(a) DRL-MOA and NSGA-II (b) DRL-MOA and MOEA/D

Hình 3.3: Kết quả so sánh giữa các thuật toán trong trường hợp 100 thành phố



(a) DRL-MOA and NSGA-II (b) DRL-MOA and MOEA/D

Hình 3.4: Kết quả so sánh giữa các thuật toán trong trường hợp 150 thành phố



(a) DRL-MOA and NSGA-II (b) DRL-MOA and MOEA/D

Hình 3.5: Kết quả so sánh giữa các thuật toán trong trường hợp 40 thành phố

Đối với trường hợp 40 thành phố, cả 3 thuật toán đều cho ra *Pareoto Front* tốt, thậm chí thuật toán MOEA/D và NSGA-II còn cho kết quả hội tụ tốt hơn so với DRL-MOA. Tuy nhiên khi số bài toán tăng dần tức đồ thị trở lên dày hơn thì dù qua quá trình lặp lớn 4000 vòng lặp thì cả 2 thuật toán MOEA/D và NSGA-II cho kết quả hội tụ tệ trong khi đó đường cong Pareto được tạo ra bởi thuật toán DRL-MOA đều tốt hơn cả về độ hội tụ và tính đa dạng *diversity*. Từ đó khẳng định độ hiệu quả của thuật toán DRL-MOA trong việc giải bài toán MOTSP.

Chương 4

Kết luận và hướng phát triển đề tài

4.1 Kết luận

Đồ án này đã trình bày các kiến thức cơ sở của tối ưu đa mục tiêu, *Học sâu, Học tăng cường* để từ đó trình bày kiến trúc mạng *Pointer* và quy tắc luyện *Actor-Critic* để giải hai bài toán tổ hợp có nhiều ứng dụng trong thực tế là TSP và MOTSP. Một số kết quả mà tác giả đạt được trong quá trình hoàn thiện đồ án này như sau:

- Trình bày nội dung lý thuyết về tối ưu đa mục tiêu, mạng *Neural* và *Học tăng cường* cơ bản.
- Tiến hành khảo sát các phương pháp giải bài toán TSP và phiên bản đa mục tiêu là bài toán MOTSP.
- Trình bày một số phương pháp để giải các bài toán tối ưu tổ hợp đa mục tiêu như các giải thuật tiến hóa đa mục tiêu, các mô hình *Học tăng cường*,...
- Trình bày chi tiết nội dung hai bài báo [9] và [1] là hai bài báo ứng dụng *Deep Reinforcement Learning* để giải quyết bài toán tối ưu tổ hợp người đi du lịch và người đi du lịch đa mục tiêu.
- Đồ án đã đưa ra các kết quả thực nghiệm và đánh giá mô hình sau quá trình cài đặt các mô hình *Học tăng cường* trên thực tế.

4.2 Hướng phát triển đề tài

Trong thời gian tới, tác giả sẽ tiếp tục nghiên cứu đề tài và tiến hành áp dụng các kiến thức của *Deep Reinforcement Learning* để cải tiến đề tài thông qua các ý tưởng sau:

- Giải các bài toán tối ưu tổ hợp đa mục tiêu khác như: bài toán Knapsack, bài toán Capacitated vehicle routing,..

- Thay đổi chiến lược *decomposition*, thay phương pháp vô hướng hóa bằng các phương pháp giải bài toán tối ưu đa mục tiêu phức tạp hơn.
- Nghiên cứu các xây dựng cách thiết kế hàm mục tiêu mới, các kiến trúc mạng neural mới nhằm tăng hiệu suất của quy tắc luyện *Actor-Critic*.
- Sử dụng các mô hình *Học tăng cường đa mục tiêu (Multi-Objective Reinforcement Learning)* để giải các bài toán tổ hợp động như bài toán định tuyến trong mạng viễn thông.

Tài liệu tham khảo

[1] Irwan Bello et al. "Neural combinatorial optimization with reinforcement learning". In: *arXiv preprint arXiv:1611.09940* (2016).

[2] Kalyanmoy Deb and Himanshu Jain. "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints". In: *IEEE transactions on evolutionary computation* 18.4 (2013), pp. 577–601.

[3] Kalyanmoy Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[4] Matthias Ehrgott. *Multicriteria optimization*. Vol. 491. Springer Science & Business Media, 2005.

[5] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[7] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[8] Vijay Konda and John Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems* 12 (1999).

[9] Kaiwen Li, Tao Zhang, and Rui Wang. "Deep reinforcement learning for multiobjective optimization". In: *IEEE transactions on cybernetics* 51.6 (2020), pp. 3103–3114.

[10] Ke Li et al. "An evolutionary many-objective optimization algorithm based on dominance and decomposition". In: *IEEE transactions on evolutionary computation* 19.5 (2014), pp. 694–716.

[11] Thibaut Lust and Jacques Teghem. "The multiobjective traveling salesman problem: a survey and a new approach". In: *Advances in Multi-Objective Nature Inspired Computing*. Springer, 2010, pp. 119–141.

[12] Christos H Papadimitriou and Kenneth Steiglitz. "On the complexity of local search for the traveling salesman problem". In: *SIAM Journal on Computing* 6.1 (1977), pp. 76–83.

[13] Gerhard Reinelt. "TSPLIB—A traveling salesman problem library". In: *ORSA journal on computing* 3.4 (1991), pp. 376–384.

[14] Richard S Sutton, Andrew G Barto, et al. "Introduction to reinforcement learning". In: (1998).

[15] Ye Tian et al. "PlatEMO: A MATLAB platform for evolutionary multi-objective optimization". In: *IEEE Computational Intelligence Magazine* 12.4 (2017), pp. 73–87.

[16] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3 (1992), pp. 229–256.

[17] Qingfu Zhang and Hui Li. "MOEA/D: A multiobjective evolutionary algorithm based on decomposition". In: *IEEE Transactions on evolutionary computation* 11.6 (2007), pp. 712–731.