

# Tổng quan lĩnh vực tìm kiếm kiến trúc mạng thần kinh nhân tạo

N. V. Tuấn Kiệt <sup>1,2</sup>    N. Hồng Đăng <sup>1</sup>    H. Quốc Hùng <sup>1</sup>    Đ. Phúc Long <sup>1</sup>

<sup>1</sup>Chương trình tài năng - Khoa học máy tính K67  
Trường Công nghệ Thông tin và Truyền thông

<sup>2</sup>Neural Architecture Search  
Optimization Research Group, BK.AI

Ngày 2 tháng 1 năm 2024

# Mục lục

Khát quát về Học sâu

Giới thiệu các kiến trúc mạng

Phát biểu bài toán

Các phương pháp tiếp cận

## Khát quát về Học sâu

### Giới thiệu các kiến trúc mạng

Kiến trúc mạng tích chập

Các kiến trúc mạng tích chập hiện đại

Kiến trúc mạng hồi quy

Các kiến trúc mạng hồi quy hiện đại

### Phát biểu bài toán

Khái quát lĩnh vực nghiên cứu

Bộ dữ liệu kiến trúc làm chuẩn

Ứng dụng và các thành tựu đạt được

### Các phương pháp tiếp cận

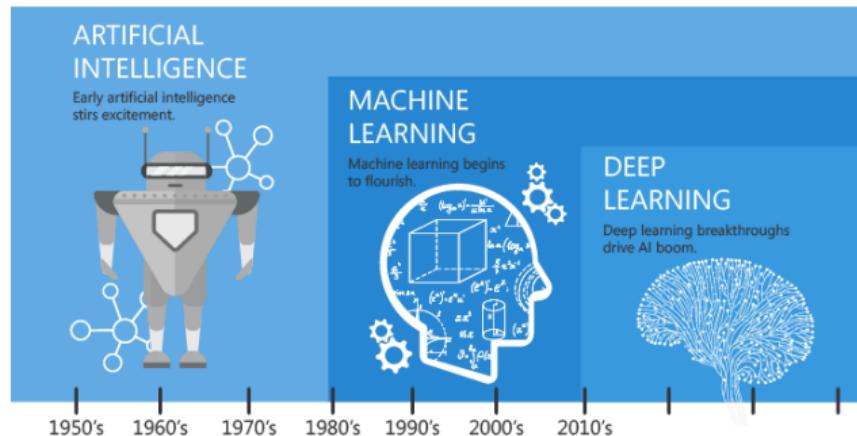
Các giải thuật phỏng tự nhiên

Các giải thuật dựa trên vi phân

Kỹ thuật học tăng cường

## Khát khao về Học sâu

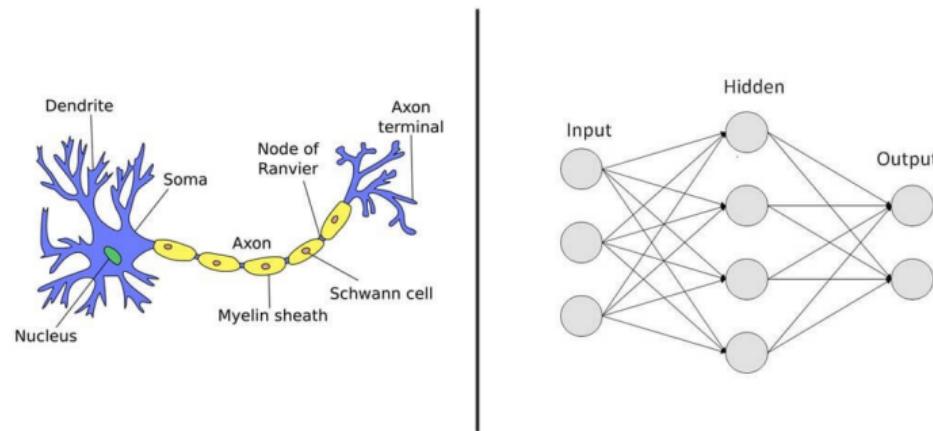
Trong lĩnh vực Trí tuệ nhân tạo, Học sâu (deep learning) là một nhánh của Học máy (machine learning) tập trung vào việc xây dựng và huấn luyện các mô hình máy tính, thường là mạng nơ-ron (neural networks), để tự động học biểu diễn cấp cao từ dữ liệu.



Hình 1: Lịch sử phát triển Trí tuệ nhân tạo, Học máy và Học sâu

# Mạng thần kinh nhân tạo

**Mạng thần kinh nhân tạo** (hay mạng nơ-ron) là một hệ thống tích hợp các nút (nơ-ron) kết nối với nhau qua các liên kết (trọng số) để thực hiện xử lý thông tin.

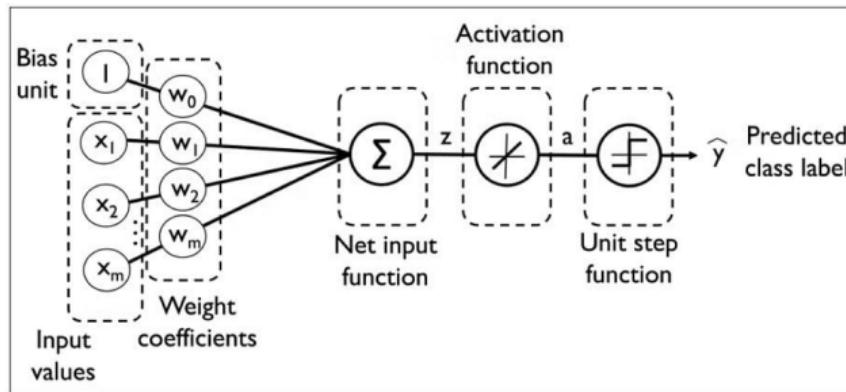


Hình 2: Mạng nơ-ron được thiết kế dựa trên cấu trúc tương tự như tế bào thần kinh trong não người

## Phân loại mạng thần kinh nhân tạo

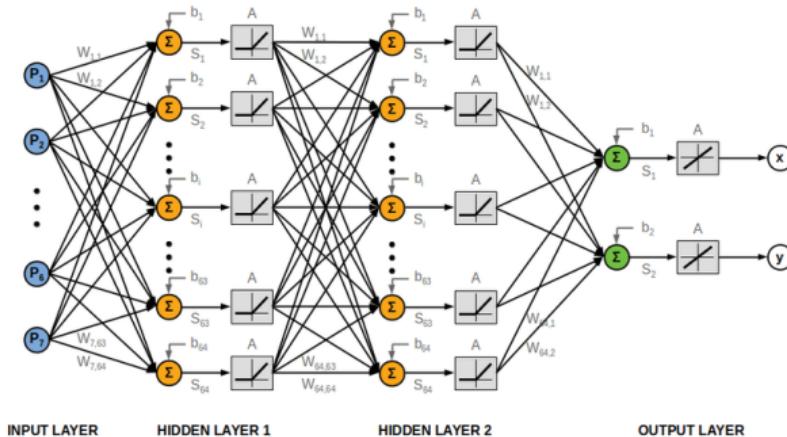
Các mạng nơ-ron có ứng dụng đa dạng và được thiết kế để giải quyết các loại nhiệm vụ khác nhau. Dưới đây là một số loại mạng nơ-ron phổ biến:

- **Perceptron** bao gồm một lớp đầu vào với các trọng số tương ứng và một đầu ra duy nhất.
- **Perceptron đa lớp** (Multilayer Perceptron - MLP) là một dạng phức tạp hơn của Perceptron, có nhiều lớp ẩn giữa lớp đầu vào và lớp đầu ra.



Hình 3: Minh họa kiến trúc Perceptron gồm: trọng số, toán tử lấy tổng và hàm kích hoạt

- **Mạng truyền thẳng** (Feedforward Neural Network - FNN) có dữ liệu di chuyển theo hướng một chiều từ lớp đầu vào đến lớp đầu ra, không có chu kỳ hoặc lưu trữ trạng thái.
- **Mạng tích chập** (Convolutional Neural Network - CNN) là một thiết kế đặc biệt cho xử lý hình ảnh và dữ liệu lưới, sử dụng các tầng tích chập để trích xuất đặc trưng không gian.
- **Mạng hồi quy** (Recurrent Neural Network - RNN) là một mô hình có khả năng lưu trữ trạng thái trước đó, phù hợp cho xử lý dữ liệu chuỗi như văn bản và âm thanh.



**Hình 4:** Minh họa kiến trúc FNN như một Perceptron đa lớp

# Huấn luyện mạng thần kinh nhân tạo

Huấn luyện mạng nơ-ron là quá trình điều chỉnh trọng số trên kết nối giữa các nơ-ron để giảm thiểu sai số giữa dự đoán của mạng và giá trị thực tế. Phương pháp hiệu quả nhất trong huấn luyện mạng nơ-ron là **Giải thuật Lan truyền ngược**<sup>1</sup> (backpropagation).

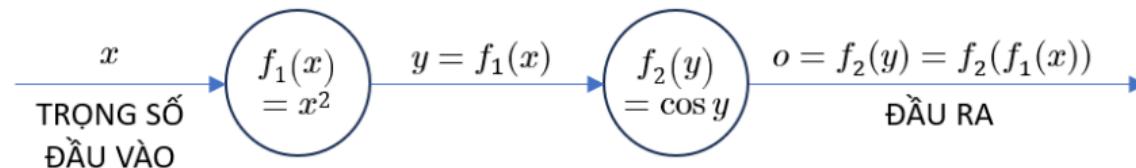
- Quá trình này bắt đầu với việc đưa dữ liệu qua mạng (lan truyền tiên), sau đó so sánh kết quả dự đoán với giá trị thực tế để tính toán sai số.
- Tiếp theo, thực hiện lan truyền ngược từ lớp đầu ra về lớp đầu vào, điều chỉnh trọng số trên đường đi để giảm sai số.

**Quy tắc dây chuyền** (chain rule) là nền tảng của Lan truyền ngược, giúp tính toán đạo hàm của hàm mất mát theo trọng số của mạng.

---

<sup>1</sup>Neural Networks and Deep Learning - A Textbook

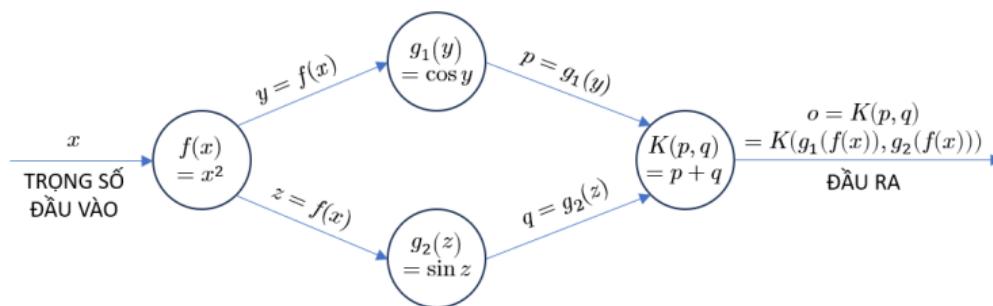
# Quy tắc dây chuyền



Hình 5: Một đồ thị tính toán có 2 nút

Trong Hình 5, lấy đạo hàm đầu ra theo đầu vào ta có:

$$\frac{\partial o}{\partial x} = \frac{\partial f_2(f_1(x))}{\partial x} = \underbrace{\frac{\partial f_2(f_1(x))}{f_1(x)}}_{-\sin f_1(x)} \cdot \underbrace{\frac{\partial f_1(x)}{\partial x}}_{2x} = -\sin(x^2) \cdot 2x$$



Hình 6: Một đồ thị tính toán có 2 đường đi

Trong Hình 6, lấy đạo hàm đầu ra là một hàm nhiều biến ta có:

$$\begin{aligned}
 \frac{\partial o}{\partial x} &= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial x} + \frac{\partial o}{\partial q} \cdot \frac{\partial q}{\partial x} \\
 &= \frac{\partial o}{\partial p} \cdot \frac{\partial p}{\partial y} \cdot \frac{\partial y}{\partial x} + \frac{\partial o}{\partial p} \cdot \frac{\partial q}{\partial z} \cdot \frac{\partial z}{\partial x} \\
 &= \underbrace{\frac{\partial K(p, q)}{\partial p}}_1 \cdot \underbrace{\cancel{g'_1(y)}_{-\sin y} \cdot \cancel{f'(x)}_{2x}} + \underbrace{\frac{\partial K(p, q)}{\partial q}}_1 \cdot \underbrace{\cancel{g'_2(y)}_{\cos z} \cdot \cancel{f'(x)}_{2x}} \\
 &= -\sin(x^2) \cdot 2x + \cos(x^2) \cdot 2x
 \end{aligned}$$

## Bổ đề Tổng hợp đường đi - Pathwise Aggregation Lemma

Xét một đồ thị tính toán có hướng không chu trình mà nút  $i$  chứa biến  $y^{(i)}$ . Đạo hàm địa phương  $z(i, j)$  của cạnh có hướng  $(i, j)$  được định nghĩa là  $z(i, j) = \frac{\partial y^{(j)}}{\partial y^{(i)}}$ . Gọi  $\mathcal{P}$  là tập tất cả các đường đi từ biến  $x$  đến biến đầu ra  $o$  trong đồ thị, khi đó:

$$S(x, o) = \frac{\partial o}{\partial x} = \sum_{P \in \mathcal{P}} \prod_{(i,j) \in P} z(i, j)$$

Gọi  $A(i)$  là tập hợp các nút tại các điểm cuối của các cạnh đi ra từ nút  $i$ . Khi đó, việc tính toán giá trị tổng hợp  $S(i, o)$  cho mỗi nút trung gian  $i$  (giữa  $x$  và  $o$ ) bằng cách sử dụng cập nhật **Quy hoạch động** (dynamic programming):

$$S(i, o) \leftarrow \sum_{j \in A(i)} S(j, o)z(i, j)$$

Quá trình tính toán này được thực hiện ngược lại, bắt đầu từ các nút trực tiếp liên quan đến  $o$ , vì đã biết  $S(o, o) = 1$ .

## Giải thuật Lan truyền ngược

Xét một đồ thị tính toán trong đó các nút chứa các biến sau kích hoạt trong mạng nơ-ron. Đây chính là các biến ẩn của các tầng khác nhau.

Giải thuật Lan truyền ngược đầu tiên sử dụng *giai đoạn tiến* để tính giá trị đầu ra và măt mát. Giai đoạn tiến thiết lập khởi tạo cho đệ quy quy hoạch động và cũng là các biến trung gian sẽ được sử dụng trong *giai đoạn lùi*. Mô tả các giai đoạn này như sau:

- **Giai đoạn tiến** (Forward phase): Một vector đầu vào cụ thể được sử dụng để tính toán giá trị của mỗi tầng ẩn dựa trên giá trị hiện tại của các trọng số. Mục tiêu của giai đoạn tiến là tính toán tất cả các biến ẩn và đầu ra trung gian cho một đầu vào cụ thể.
- **Giai đoạn lùi** (Backward phase): Giai đoạn lùi tính đạo hàm của hàm măt mát đối với các trọng số khác nhau. Bước đầu tiên là tính đạo hàm  $\frac{\partial L}{\partial o}$ .

## Lan truyền ngược sau kích hoạt

Xét một đường đi được ký hiệu bằng dãy các đơn vị ẩn  $h_1, h_2, \dots, h_k$  tiếp theo là đầu ra  $o$ . Trọng số của kết nối từ đơn vị ẩn  $h_r$  đến  $h_{r+1}$  được ký hiệu bởi  $w_{(h_r, h_{r+1})}$ .

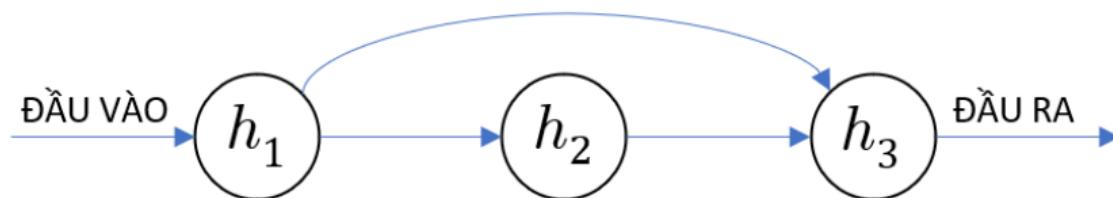
Khi tồn tại một tập  $\mathcal{P}$  gồm tất cả đường đi từ  $h_r$  đến  $o$ , đạo hàm matsu có thể được biểu diễn như sau:

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \underbrace{\frac{\partial L}{\partial o} \cdot \left[ \sum_{[h_r, h_{r+1}, \dots, h_k, o] \in \mathcal{P}} \frac{\partial o}{\partial h_k} \prod_{i=r}^{k-1} \frac{\partial h_{i+1}}{\partial h_i} \right]}_{\text{Lan truyền ngược tính toán } \Delta(h_r, o) = \frac{\partial L}{\partial h_r}}$$

Trước tiên tính toán  $\Delta(h_k, o)$  cho các nút  $h_k$  gần  $o$  nhất, sau đó đệ quy tính toán những giá trị này cho các nút ở các tầng trước dựa trên các nút ở các tầng sau.

$$\Delta(h_r, o) = \frac{\partial L}{\partial h_r} = \sum_{h:h_r \rightarrow h} \frac{\partial L}{\partial h} \cdot \frac{\partial h}{\partial h_r} = \sum_{h:h_r \rightarrow h} \frac{\partial h}{\partial h_r} \cdot \underbrace{\Delta(h, o)}_{\text{Đã tính toán}}$$

Tổng này lấy trên tất cả các đơn vị ẩn  $h$  mà có kết nối xuất phát từ  $h_r$  và kết thúc tại nó.



$$\Delta(h_1, o) = \frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_2} \frac{\partial h_2}{\partial h_1} + \frac{\partial L}{\partial h_3} \frac{\partial h_3}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \Delta(h_2, o) + \frac{\partial h_3}{\partial h_1} \Delta(h_3, o)$$

Hình 7: Minh họa cập nhật đệ quy

Gọi  $a_h$  là giá trị tính toán tại đơn vị ẩn  $h$  trước khi áp dụng hàm kích hoạt  $\Phi(\cdot)$ . Ta có:

$$a_h = w_{(h_r, h)}^\top h_r \quad \text{và} \quad h = \Phi(a_h)$$

Khi đó:

$$\frac{\partial h}{\partial h_r} = \frac{\partial h}{\partial a_h} \cdot \frac{\partial a_h}{\partial h_r} = \frac{\partial \Phi(a_h)}{\partial a_h} \cdot w_{(h_r, h)} = \Phi'(a_h) \cdot w_{(h_r, h)}$$

Như vậy:

$$\Delta(h_r, o) = \sum_{h: h_r \rightarrow h} \Phi'(a_h) \cdot w_{(h_r, h)} \cdot \Delta(h, o)$$

## Lan truyền ngược trước kích hoạt

Viết lại công thức đạo hàm matsu theo  $a_h$ , ta có:

$$\frac{\partial L}{\partial w_{(h_{r-1}, h_r)}} = \underbrace{\frac{\partial L}{\partial o} \cdot \Phi'(a_o) \cdot \left[ \sum_{[h_r, h_{r+1}, \dots, h_k, o] \in \mathcal{P}} \frac{\partial a_o}{\partial a_{h_k}} \prod_{i=r}^{k-1} \frac{\partial a_{h_{i+1}}}{\partial a_{h_i}} \right] h_{r-1}}_{\text{Lan truyền ngược tính toán } \delta(h_r, o) = \frac{\partial L}{\partial a_{h_r}}}$$

Tương tự, cập nhật đệ quy như sau:

$$\delta(h_r, o) = \frac{\partial L}{\partial a_{h_r}} = \sum_{h: h_r \rightarrow h} \frac{\partial L}{\partial a_h} \cdot \frac{\partial a_h}{\partial a_{h_r}} = \sum_{h: h_r \rightarrow h} \frac{\partial a_h}{\partial a_{h_r}} \cdot \delta(h, o)$$

Trong đó, áp dụng Quy tắc dây chuyền:

$$\frac{\partial a_h}{\partial a_{h_r}} = \frac{\partial a_h}{\partial h_r} \cdot \frac{\partial h_r}{\partial a_{h_r}} = w_{(h_r, h)} \cdot \frac{\partial \Phi(a_{h_r})}{\partial a_{h_r}} = w_{(h_r, h)} \cdot \Phi'(a_{h_r})$$

Như vậy:

$$\delta(h_r, o) = \Phi'(a_{h_r}) \sum_{h: h_r \rightarrow h} w_{(h_r, h)} \cdot \delta(h, o)$$

- Một lợi ích của việc sử dụng biến trước kích hoạt so với sau kích hoạt là đạo hàm kích hoạt ở bên ngoài tổng, từ đó đơn giản hóa tính toán.
- Phương pháp này đại diện cho cách lan truyền ngược thực sự được triển khai trong các hệ thống thực tế.

## Khát quát về Học sâu

### Giới thiệu các kiến trúc mạng

Kiến trúc mạng tích chập

Các kiến trúc mạng tích chập hiện đại

Kiến trúc mạng hồi quy

Các kiến trúc mạng hồi quy hiện đại

### Phát biểu bài toán

Khái quát lĩnh vực nghiên cứu

Bộ dữ liệu kiến trúc làm chuẩn

Ứng dụng và các thành tựu đạt được

### Các phương pháp tiếp cận

Các giải thuật phỏng tự nhiên

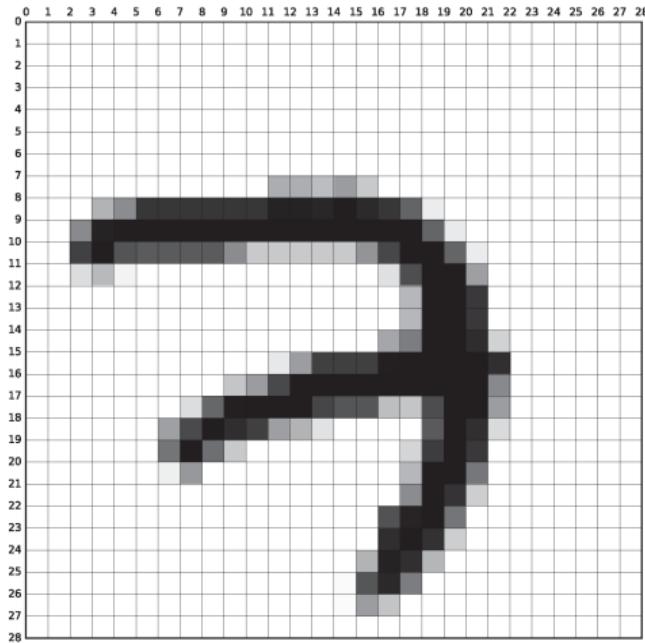
Các giải thuật dựa trên vi phân

Kỹ thuật học tăng cường

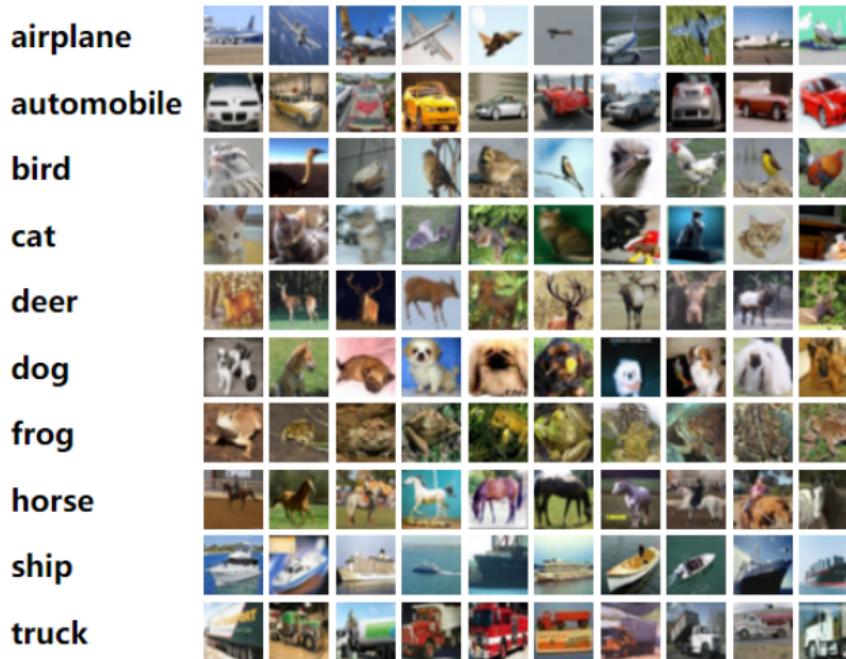
## Khái niệm mở đầu

**Mạng nơ-ron tích chập** (Convolutional Neural Network - CNN) là một loại mạng nơ-ron sâu, thường được sử dụng trong xử lý hình ảnh và video.

- Trong mạng nơ-ron tích chập, các trạng thái ở mỗi lớp được tổ chức theo cấu trúc lưới không gian (spatial grid).
- Đầu vào của mỗi lớp trong mạng tích chập có cấu trúc lưới 3 chiều, gồm chiều cao, chiều rộng, và chiều sâu.
- Giá sử đầu vào của lớp  $q$  có kích thước  $L_q \times B_q \times d_q$  trong đó  $L_q$  là chiều cao (height) hoặc chiều dài (length),  $B_q$  là chiều rộng (width) hoặc chiều ngang (breadth),  $d_q$  là chiều sâu (depth). Đối với lớp đầu tiên (lớp đầu vào), những giá trị này được quyết định bởi bản chất của dữ liệu và quá trình tiền xử lý của nó.



Hình 8: Minh họa bộ dữ liệu chữ số viết tay MNIST gồm các ảnh đen trắng kích thước  $28 \times 28 \times 1$ .



Hình 9: Minh họa bộ dữ liệu CIFAR-10 gồm các ảnh màu kích thước  $32 \times 32 \times 3$ .

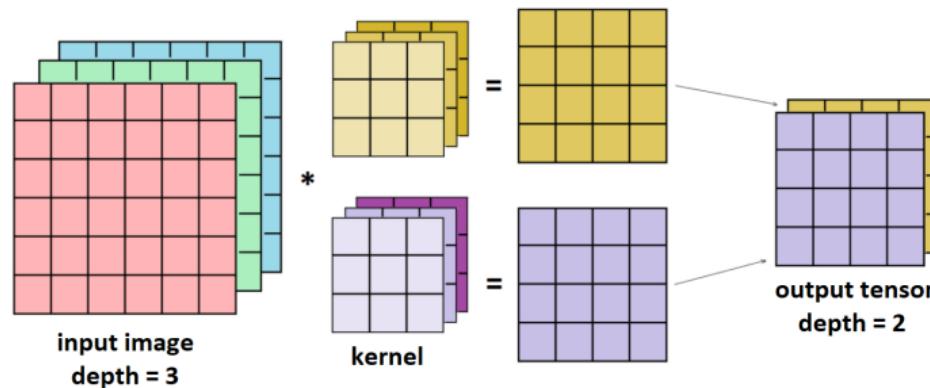
## Bản đồ đặc trưng

Khi (số thứ tự lớp)  $q > 1$  (tức không kể đến lớp đầu tiên), thì mỗi lưới không gian 2 chiều nằm dọc theo chiều sâu được gọi là bản đồ đặc trưng (feature map) hay bản đồ kích hoạt (activation map).

- Mỗi bản đồ đặc trưng được tạo ra bằng cách áp dụng một bộ lọc (filter hoặc kernel) lên hình ảnh đầu vào hoặc bản đồ đặc trưng từ lớp trước.
- Bản đồ đặc trưng đại diện cho nơi mà các đặc điểm (được bộ lọc đặc trưng hóa) xuất hiện trong hình ảnh đầu vào.

Các biểu diễn tại lớp sau trong CNN đều có cùng cấu trúc 3 chiều như lớp đầu vào, nhưng ý nghĩa của các giá trị trong lưới này đã thay đổi:

- Chiều cao  $L_q$  và chiều rộng  $B_q$  của lưới đại diện cho không gian 2 chiều của ảnh hoặc bản đồ đặc trưng.
- Chiều sâu của lưới không gian  $d_q$  không còn đại diện cho các kênh màu như trong lớp đầu vào, mà nó đại diện cho số lượng các bản đồ đặc trưng khác nhau.

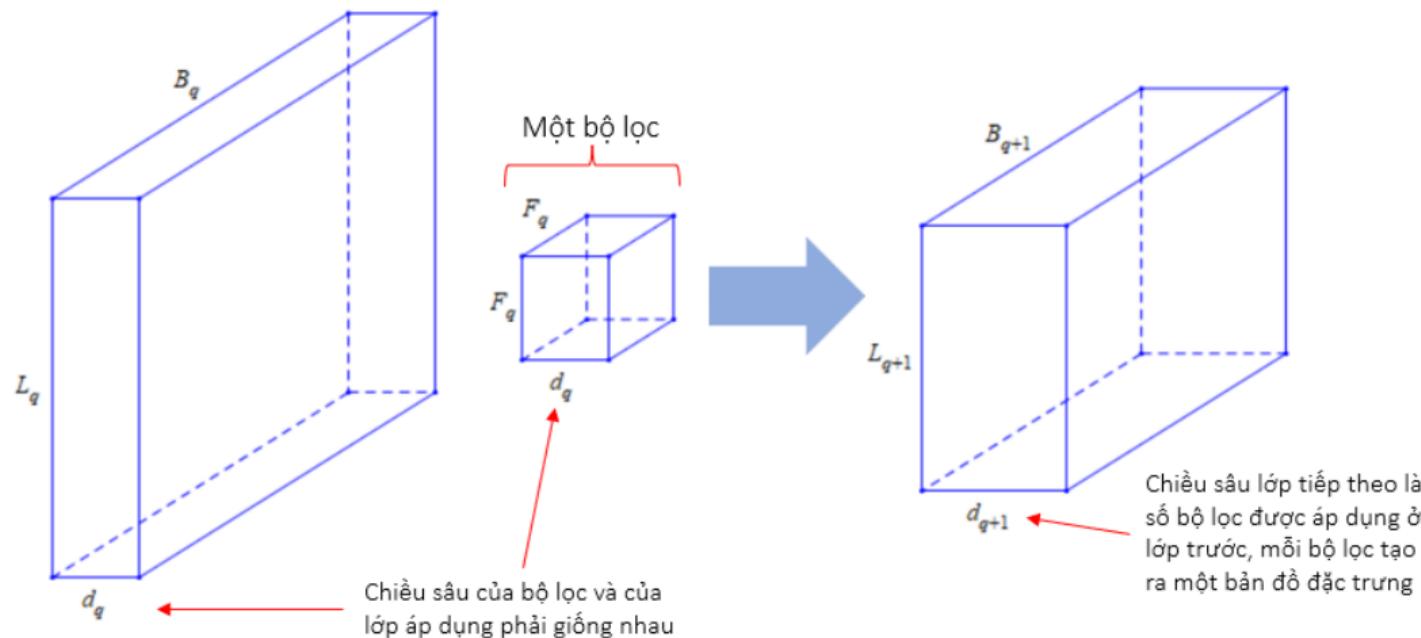


**Hình 10:** Minh họa toán tử tích chập tác động lên đầu vào tạo ra bản đồ đặc trưng.

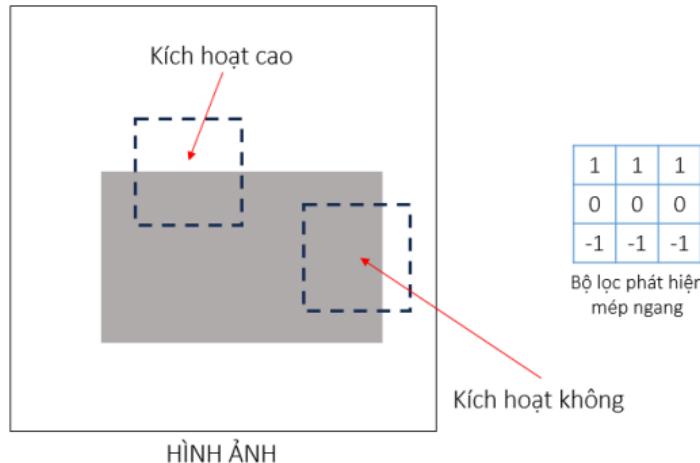
## Phép toán tích chập

Trong mạng nơ-ron tích chập, các tham số được tổ chức thành các bộ đơn vị cấu trúc 3 chiều (bộ lọc, filter hay kernel). Giả sử kích thước bộ lọc trong lớp  $q$  là  $F_q \times F_q \times d_q$ , trong đó  $F_q$  thường nhỏ và là số lẻ.

- **Phép toán tích chập** (convolution) đặt bộ lọc tại mỗi vị trí có thể có trong hình ảnh (hoặc lớp ẩn) để bộ lọc trùng hợp hoàn toàn với hình ảnh, và thực hiện một tích vô hướng giữa các tham số  $F_q \times F_q \times d_q$  trong bộ lọc với vùng địa phương của lưới phù hợp có cùng kích thước.
- Khi thực hiện các phép toán tích chập ở lớp thứ  $q$ , người ta có thể căn chỉnh bộ lọc tại  $L_{q+1} = L_q - F_q + 1$  vị trí đọc theo chiều cao và  $B_{q+1} = B_q - F_q + 1$  vị trí đọc theo chiều rộng của hình ảnh. Xác định một bản đồ đặc trưng làm đầu ra có kích thước  $L_{q+1} \times B_{q+1}$ .



**Hình 11:** Minh họa hoạt động của phép toán tích chập. Lưu ý, có thể có nhiều bộ lọc và việc áp dụng mỗi một bộ lọc sẽ xác định một bản đồ đặc trưng độc theo chiều sâu của đầu ra. Thông thường, các lớp sau cùng có kích thước không gian nhỏ hơn, nhưng độ sâu lớn hơn về số lượng bản đồ đặc trưng.



HÌNH ẢNH

**Hình 12:** Đặc trưng kết quả sẽ có kích hoạt cao tại mỗi vị trí mà một mép ngang được nhìn thấy. Một mép hoàn toàn dọc sẽ cho kích hoạt bằng không, trong khi một mép nghiêng có thể cho kích hoạt ở giữa. Do đó, việc trượt bộ lọc ở mọi nơi trên hình ảnh sẽ ngay lập tức phát hiện ra một số đường viền chính của hình ảnh trong một bản đồ đặc trưng duy nhất của khối lượng đầu ra. Nhiều bộ lọc được sử dụng để tạo ra một khối lượng đầu ra với nhiều hơn một bản đồ đặc trưng. Ví dụ, một bộ lọc khác có thể được dùng để tạo ra một bản đồ đặc trưng của kích hoạt mép dọc.

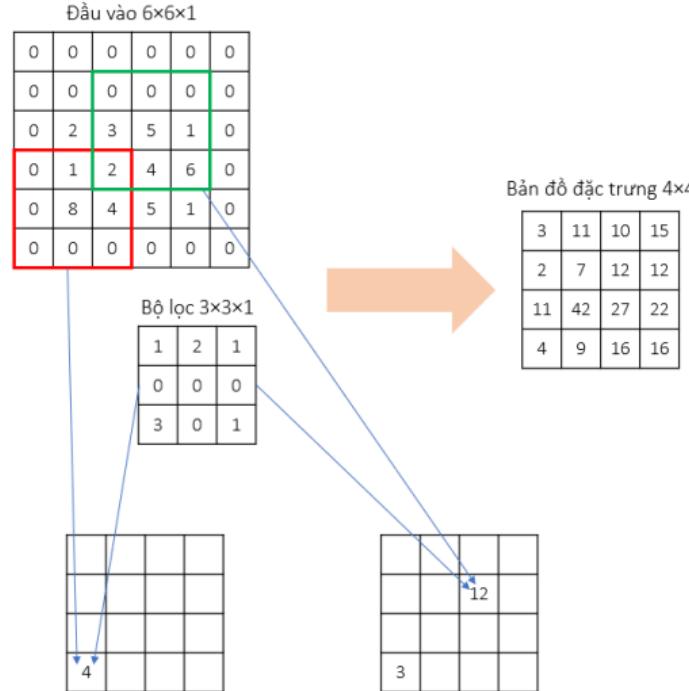
Bộ lọc thứ  $p$  trong lớp thứ  $q$  được biểu diễn bởi tensor 3 chiều  $W = [w_{ijk}^{(p,q)}]$ . Các chỉ số  $i, j, k$  chỉ vị trí đọc theo chiều cao, chiều rộng, và chiều sâu của bộ lọc.

Các bản đồ đặc trưng trong lớp thứ  $q$  được biểu diễn bởi tensor 3 chiều  $H^{(q)} = [h_{ijk}^{(q)}]$ , trong trường hợp  $q = 1$  tương ứng với ký hiệu  $H^{(1)}$  đơn giản chỉ lớp đầu vào (không ẩn).

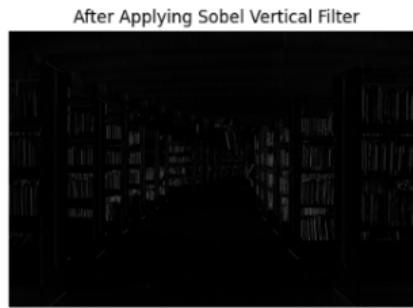
Khi đó, phép toán tích chập từ lớp thứ  $q$  đến lớp thứ  $q + 1$  thông qua bộ lọc thứ  $p$  là:

$$h_{ijp}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} h_{i+r-1, j+s-1, k}^{(q)}$$

với mọi  $i \in \{1, 2, \dots, L_q - F_q + 1\}$ ,  $j \in \{1, 2, \dots, B_q - F_q + 1\}$  và  $p \in \{1, 2, \dots, d_{p+1}\}$ .



Hình 13: Minh họa tính toán tích chập



**Hình 14:** Minh họa bản đồ đặc trưng được tạo thành khi tác động bộ lọc Sobel phát hiện mép ngang (horizontal) và mép dọc (vertical) lên ảnh.

# Đệm

Phép toán tích chập làm giảm kích thước của lớp  $q + 1$  so với lớp  $q$ . Việc giảm kích thước này nói chung không được mong muốn, bởi vì nó có xu hướng mất một số thông tin dọc theo các biên của hình ảnh (hoặc của bản đồ đặc trưng). Vấn đề này có thể được giải quyết bằng cách sử dụng **đệm** (padding). Một số dạng đệm:

- **Không sử dụng đệm** (Valid Padding).
- **Nửa đệm** (Half-Padding, Same Padding): Số lượng pixel được thêm vào mép của hình ảnh là  $\frac{F_q - 1}{2}$ , với  $F_q$  là kích thước không gian của filter. Đảm bảo rằng hình ảnh đầu ra sau phép tích chập có cùng kích thước không gian với hình ảnh đầu vào.
- **Đệm toàn phần** (Full-Padding): Hình ảnh được đệm với  $F_q - 1$  pixel trên mỗi mép, làm tăng kích thước không gian của bản đồ đặc trưng sau phép tích chập.

Đầu vào  $6 \times 6 \times 1$  đã đệm

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	2	3	5	1	0	0	0
0	0	1	2	4	6	0	0	0
0	0	8	4	5	1	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Bộ lọc  $3 \times 3 \times 1$

1	2	1
0	0	0
3	0	1

Tích chập

Bản đồ đặc trưng  $6 \times 6$

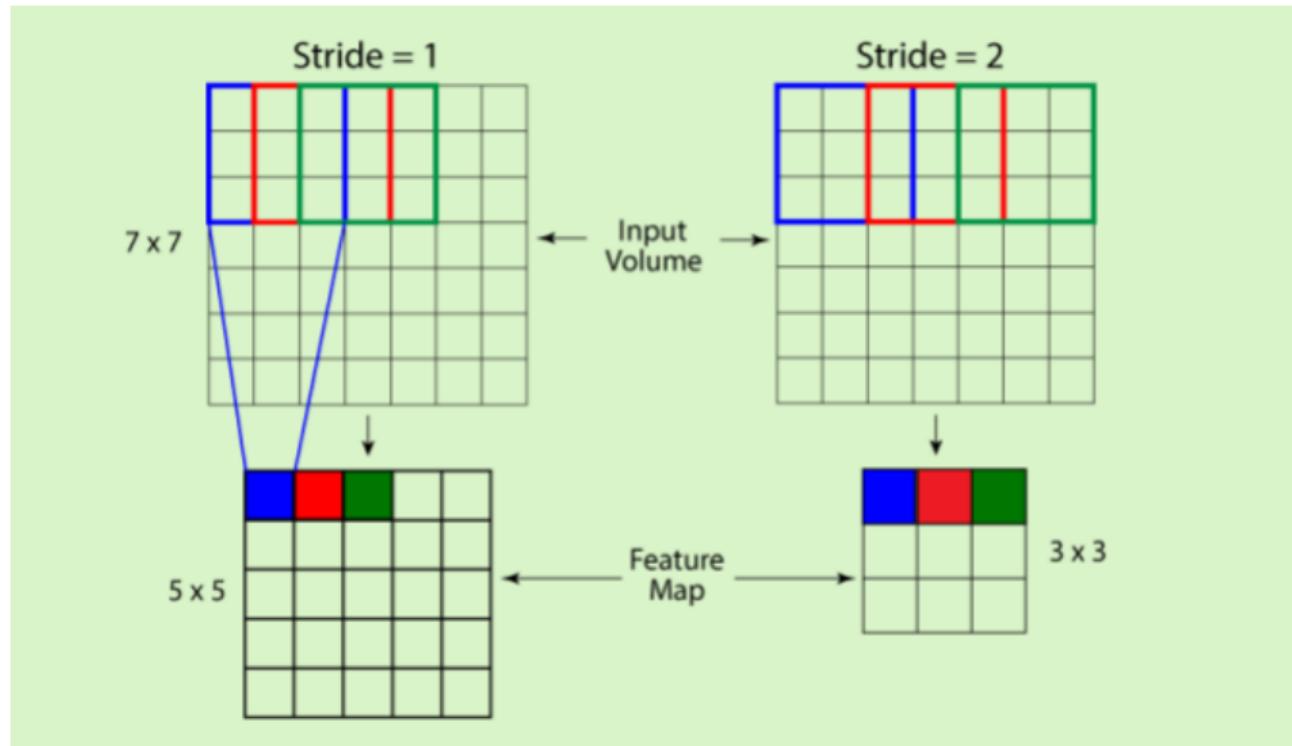
0	0	0	0	0	0	0
2	3	11	10	15	3	
1	2	7	12	12	18	
10	11	42	27	22	4	
1	3	9	16	16	6	
8	20	21	15	17	1	

Hình 15: Minh họa nửa đệm để bảo toàn kích thước không gian của đầu vào.

## Bước nhảy

Không nhất thiết phải thực hiện tích chập ở mỗi vị trí không gian trong lớp. Có thể giảm độ tinh tế của tích chập bằng cách sử dụng khái niệm về **bước nhảy** (strides).

- Khi sử dụng bước nhảy  $S_q$  trong lớp thứ  $q$ , tích chập được thực hiện ở các vị trí  $1, S_q + 1, 2S_q + 1, \dots$  đọc theo cả 2 chiều không gian của lớp.
- Kích thước không gian của đầu ra sau khi thực hiện tích chập dạng này có chiều cao  $L_{q+1} = \frac{L_q - F_q}{S_q} + 1$  và chiều rộng  $B_{q+1} = \frac{B_q - F_q}{S_q} + 1$ .



Hình 16: Minh họa bước nhảy khi áp dụng bộ lọc

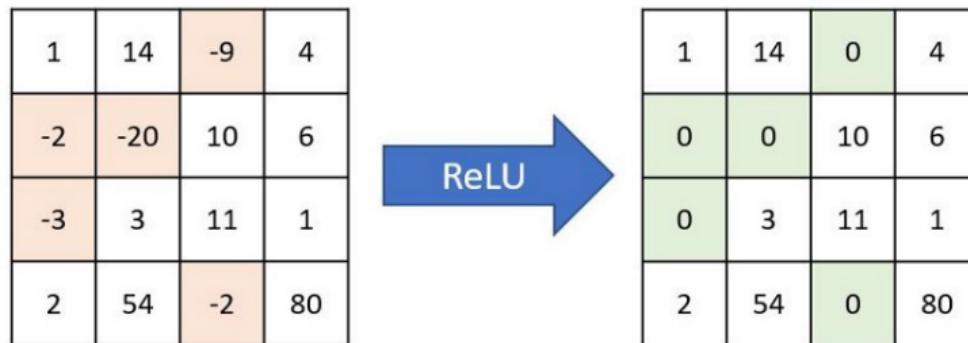
- Mô tả tích chập mặc định tương ứng với trường hợp khi sử dụng bước nhảy là 1
- Bước nhảy lớn có thể hữu ích trong các tình huống bị giới hạn bộ nhớ hoặc để giảm quá khứp (overfitting) nếu độ phân giải không gian không cần thiết quá cao.
- Bước nhảy có tác dụng tăng cường nhanh chóng trường nhận thức (receptive field) của từng đặc trưng trong lớp ẩn, trong khi giảm kích thước không gian của toàn bộ lớp. Một trường nhận thức tăng cường là hữu ích để nắm bắt một đặc trưng phức tạp trong một khu vực không gian lớn hơn của hình ảnh.

# ReLU

Phép toán tích chập được xen kẽ với các phép toán gộp và ReLU.

$$\text{ReLU}(x) = \begin{cases} 0, & \text{nếu } x \leq 0 \\ x, & \text{nếu } x > 0 \end{cases}$$

ReLU thường theo sau một toán tử tích chập. Việc áp dụng ReLU không thay đổi kích thước của một lớp.



Hình 17: Minh họa kích hoạt ReLU lên một tensor.

## Phép toán gộp

**Phép toán gộp** (pooling) hoạt động trên các vùng lưới nhỏ có kích thước  $P_q \times P_q$  trong mỗi lớp, và tạo ra một lớp khác với chiều sâu giống hệt.

- Đối với mỗi vùng vuông có kích thước  $P_q \times P_q$  của một trong số  $d_q$  bản đồ đặc trưng, giá trị lớn nhất trong số những giá trị này được trả lại. Phương pháp này được gọi là gộp tối đại (max-pooling).
- Nếu sử dụng bước nhảy là 1, sau khi gộp sẽ tạo ra một lớp mới có kích thước  $(L_q - P_q + 1) \times (B_q - P_q + 1) \times d_q$ .
- Tuy nhiên, phổ biến hơn là sử dụng bước nhảy  $S_q > 1$  trong gộp. Với trường hợp như vậy, chiều dài của lớp mới sẽ là  $L_{q+1} = \frac{L_q - P_q}{S_q} + 1$  và chiều rộng là  $B_{q+1} = \frac{B_q - P_q}{S_q} + 1$ , chiều sâu vẫn là  $d_q$ . Do đó, gộp giảm đáng kể kích thước không gian của mỗi bản đồ kích hoạt.

Bản đồ đặc trưng  $6 \times 6$ 

6	5	1	2	4	0
1	0	1	3	0	3
4	2	3	5	1	0
0	1	2	4	6	0
0	8	4	5	1	2
6	0	2	0	3	0

Gộp  $3 \times 3$   
Bước nhảy 1

8			

Gộp  $3 \times 3$  và bước nhảy 1

Gộp  $3 \times 3$   
Bước nhảy 2

6			

Bản đồ đặc trưng  $4 \times 4$ 

6	5	5	5
4	5	6	6
8	8	6	6
8	8	6	6

Bản đồ đặc trưng  $2 \times 2$ 

6	6
8	6

Hình 18: Minh họa phép toán gộp

- Khác với các phép toán tích chập, phép gộp được thực hiện ở cấp độ của từng bản đồ đặc trưng.
- Trong khi một phép toán tích chập sử dụng đồng thời tất cả  $d_q$  bản đồ đặc trưng kết hợp với một bộ lọc để tạo ra một giá trị đặc trưng đơn, còn gộp hoạt động độc lập trên mỗi bản đồ đặc trưng để tạo ra một bản đồ đặc trưng khác.
- Do đó, phép toán gộp không thay đổi số lượng bản đồ đặc trưng. Nói cách khác, chiều sâu của lớp được tạo ra bằng cách sử dụng gộp giống với chiều sâu của lớp mà phép toán gộp được thực hiện.

## POOLING

Max pooling

32	19
20	27

Average pooling

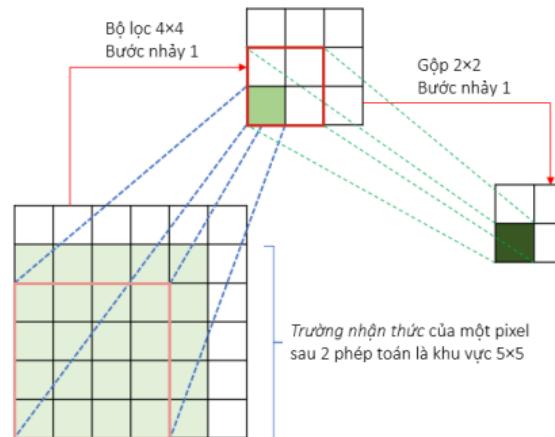
15	14
11	16

32	10	11	17
4	14	9	19
20	4	16	27
8	12	7	14

Hình 19: Minh họa gộp tối đại và gộp trung bình, kích thước  $2 \times 2$  với bước nhảy 2.

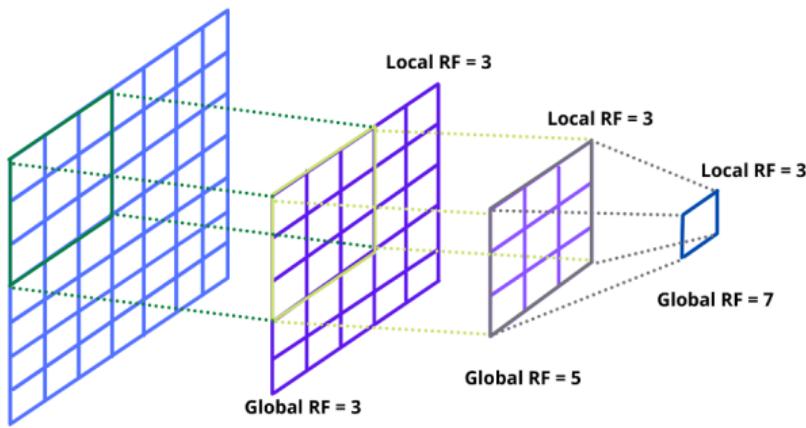
## Trường nhận thức

Một khái niệm cơ bản trong CNN sâu là **trường nhận thức** (receptive field), hoặc vùng nhìn thấy, của một đơn vị trong một lớp nhất định của mạng. Khác với trong các mạng kết nối đầy đủ, nơi giá trị của mỗi đơn vị phụ thuộc vào toàn bộ đầu vào của mạng, một đơn vị trong các mạng tích chập chỉ phụ thuộc vào một khu vực con của đầu vào. Khu vực này trong đầu vào là trường nhận thức cho đơn vị đó.



Hình 20: Minh họa về trường nhận thức

- Khái niệm về trường nhận thức quan trọng để hiểu và chẩn đoán cách mạng nơ-ron tích chập sâu hoạt động.
- Trong nhiều nhiệm vụ, đặc biệt là các nhiệm vụ dự đoán dày đặc như đoạn ngữ nghĩa hình ảnh, ước lượng stereo và luồng quang học, nơi chúng ta thực hiện dự đoán cho từng pixel đơn lẻ trong hình ảnh đầu vào, việc quan trọng là mỗi pixel đầu ra phải có một trường nhận thức lớn, sao cho không có thông tin quan trọng nào bị bỏ sót khi thực hiện dự đoán.



Hình 21: Bất cứ nơi nào trong một hình ảnh đầu vào nằm ngoài trường nhận thức của một đơn vị không ảnh hưởng đến giá trị của đơn vị đó, việc kiểm soát cẩn thận trường nhận thức là cần thiết để đảm bảo rằng nó bao phủ toàn bộ khu vực hình ảnh liên quan.

Nghiên cứu của Luo và Li<sup>2</sup> chỉ ra rằng:

- Mặc dù mỗi đơn vị trong lớp tích chập của CNN có khả năng nhận thông tin từ toàn bộ trường nhận thức lý thuyết (theoretical receptive field), trường nhận thức hiệu quả (effective receptive field) thực tế lại có phân phối theo hình chuông Gaussian, tập trung chủ yếu ở trung tâm và giảm dần về phía rìa. Điều này dẫn đến việc không phải tất cả các điểm ảnh trong trường nhận thức lý thuyết đều đóng góp như nhau vào phản ứng của một đơn vị.
- Kiểm soát ERF cũng giúp giảm hiện tượng quá khớp, nơi mạng quá mức điều chỉnh theo dữ liệu huấn luyện và mất khả năng tổng quát hóa. Đồng thời khiến mạng tập trung vào các khu vực quan trọng và loại bỏ thông tin không liên quan, từ đó tăng độ chính xác.

---

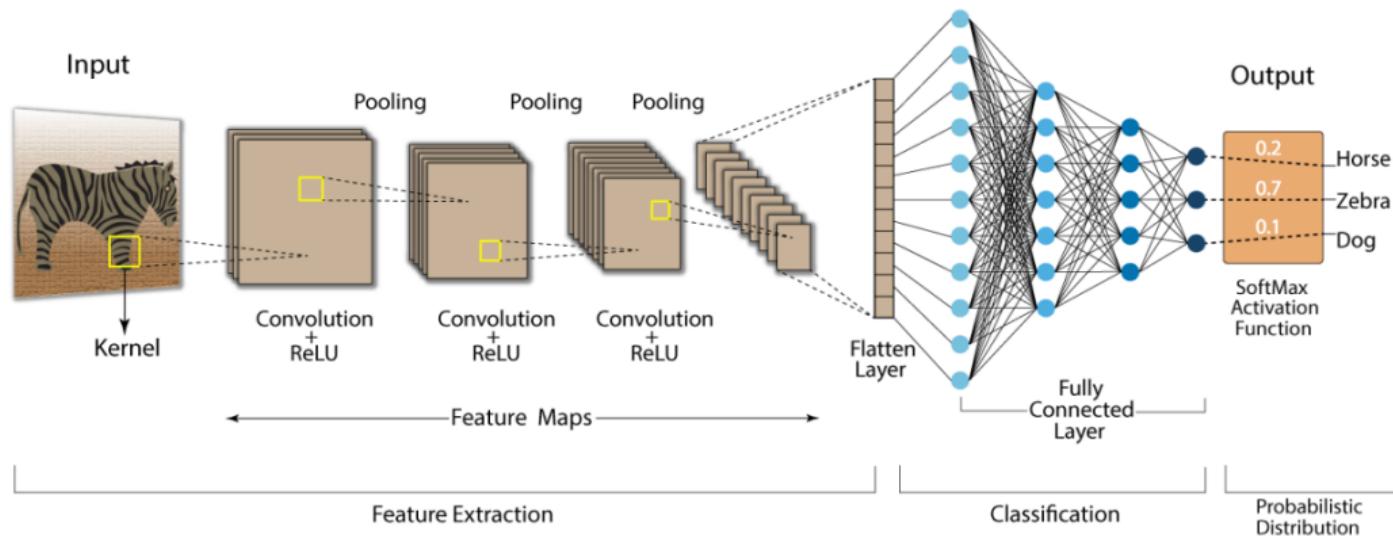
<sup>2</sup>Understanding the Effective Receptive Field in Deep Convolutional Neural Networks

## Lớp kết nối đầy đủ

**Lớp kết nối đầy đủ** (fully connected layer) thường được đặt ở cuối cùng và đóng vai trò quan trọng trong việc đưa ra quyết định cho mạng.

- Sau quá trình trích xuất đặc trưng qua các lớp tích chập và gộp, các đặc trưng sau cùng được “duỗi thẳng” (hay làm phẳng) để tạo thành một vector đặc trưng, mà từ đó lớp kết nối đầy đủ sẽ học cách đưa ra dự đoán dựa trên những thông tin đã được học.
- Mỗi đơn vị trong lớp kết nối đầy đủ kết nối với tất cả các đơn vị ở lớp trước, cho phép lớp này học được các mối quan hệ phức tạp từ toàn cục từ dữ liệu đầu vào, đồng thời làm giảm kích thước chiều của dữ liệu bằng cách chuyển đổi vector đặc trưng vào một số lượng đầu ra cố định, thường tương ứng với số lượng các nhãn cần phân loại.

## Convolution Neural Network (CNN)



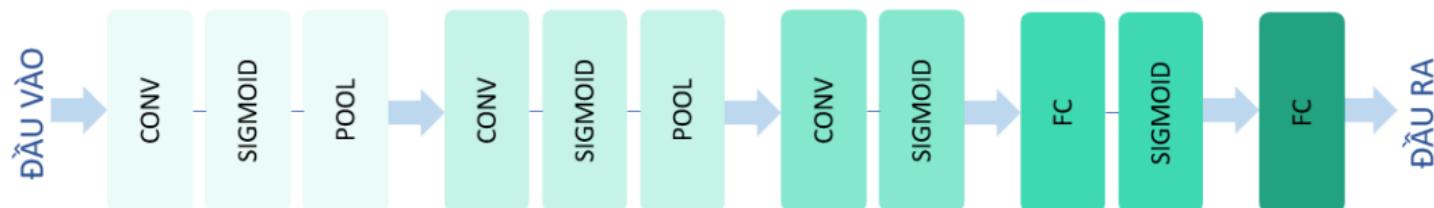
Hình 22: Minh họa một mạng CNN trong đó các lớp kết nối đầy đủ sau cùng của kiến trúc.

- Điểm mạnh của lớp kết nối đây đủ không chỉ nằm ở khả năng tổng hợp thông tin, mà còn ở việc cung cấp một không gian đặc trưng mới, trong đó các đặc trưng có thể được tối ưu hóa để phục vụ cho việc phân loại.
- Lớp này không chỉ là cầu nối giữa các đặc trưng được trích xuất và kết quả đầu ra, mà còn là nơi mà các mối quan hệ phi tuyến tính và toàn cục được hình thành và tối ưu hóa để cải thiện hiệu suất của mô hình.

## LeNet<sup>3</sup>: Kiến trúc

LeNet được phát triển bởi Yann LeCun vào năm 1998 và là một trong những CNN đầu tiên.

- Hàm kích hoạt ReLU chưa xuất hiện vào lúc ấy, sigmoid hoặc tanh được sử dụng.
- Khác với gộp tối đại lấy giá trị lớn nhất tại vùng mà pooling trượt qua, gộp trung bình trong LeNet lấy giá trị trung bình của tất cả phần tử trong khu vực pooling trượt qua.
- Việc sử dụng gộp trung bình (average-pooling) ngày nay là cực kỳ hiếm khi so với gộp tối đại (max-pooling).



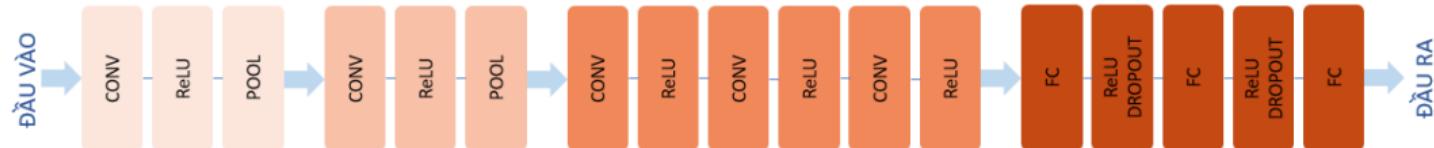
Hình 23: Minh họa kiến trúc LeNet với hàm kích hoạt sigmoid.

<sup>3</sup>Gradient-Based Learning Applied to Document Recognition, 1998

## AlexNet<sup>4</sup>: Kiến trúc

AlexNet, được giới thiệu vào năm 2012 bởi Alex Krizhevsky, Ilya Sutskever, và Geoffrey Hinton, đã đánh dấu một bước ngoặt trong lĩnh vực học sâu.

- AlexNet sử dụng hàm kích hoạt ReLU (Rectified Linear Unit) thay vì sigmoid hoặc tanh.
- AlexNet sử dụng lớp gộp tối đa để giảm kích thước không gian của các đặc trưng.
- Để giảm overfitting trong huấn luyện, các lớp kết nối đầy đủ có dropout.
- Một kỹ thuật mới trong AlexNet là chuẩn hóa phản hồi địa phương (Local Response Normalization - LRN), nhằm tăng cường sự kích thích của các đặc trưng nổi bật.



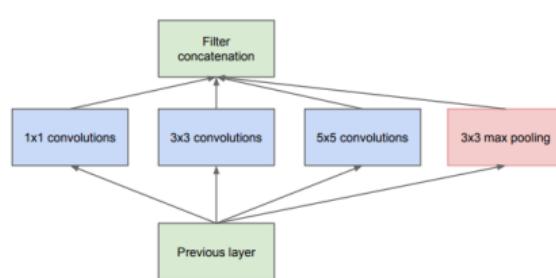
Hình 24: Minh họa kiến trúc AlexNet.

<sup>4</sup>ImageNet Classification with Deep Convolutional Neural Networks, 2012

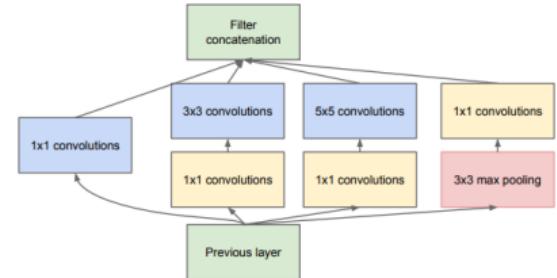
# GoogLeNet<sup>5</sup> (Inception V1): Lớp phân nhánh

GoogLeNet sử dụng các mô-đun Inception, nơi mà một lớp được phân thành nhiều nhánh khác nhau với các loại bộ lọc tích chập và gộp riêng nhau, sau đó kết hợp đầu ra của các nhánh này.

- Điều này giúp mạng có khả năng trích xuất thông tin ở nhiều quy mô khác nhau từ cùng một vùng hình ảnh.
- Mô-đun Inception sử dụng các bộ lọc  $1 \times 1$  để giảm chiều không gian trước khi thực hiện các tích chập kích thước lớn hơn, giúp giảm số lượng tham số và tính toán cần thiết.



(a) Inception module, naïve version



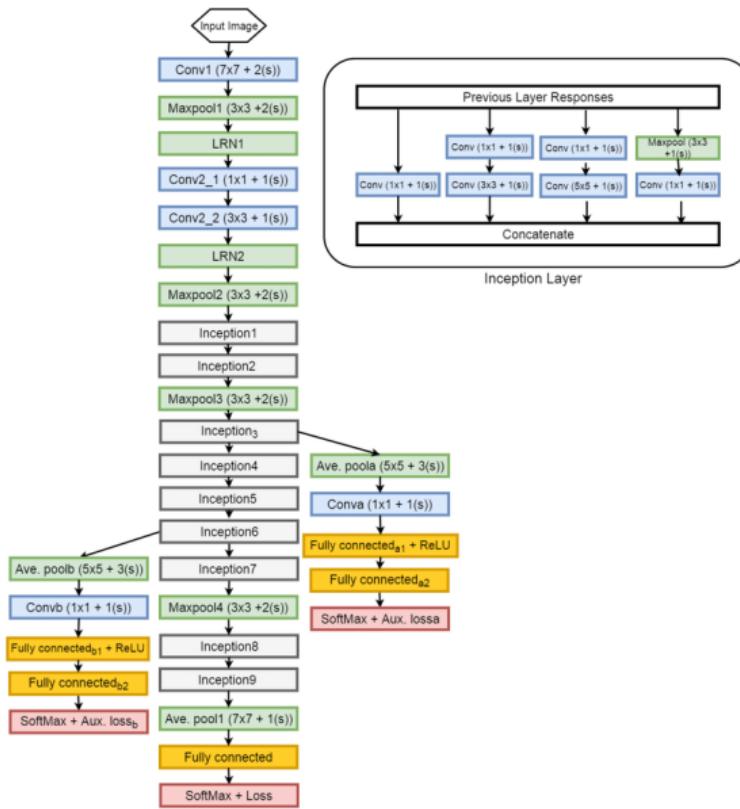
(b) Inception module with dimension reductions

**Hình 25:** Minh họa: (bên trái) kiến trúc Inception ngây thơ và (bên phải) kiến trúc Inception với giảm chiều không gian.

<sup>5</sup>Going Deeper with Convolutions, 2014

## GoogLeNet: Kiến trúc

- GoogLeNet có tổng cộng 22 lớp chứa tham số và là một trong những mạng sâu nhất tại thời điểm đó.
- Mặc dù có độ sâu lớn, số lượng tham số lại khá ít so với các mạng sâu khác như AlexNet hay VGGNet, nhờ vào việc sử dụng hiệu quả các mô-đun Inception.
- GoogLeNet sử dụng các phân loại phụ (auxiliary classifiers) ở các lớp trung gian.



Hình 26: Kiến trúc GoogLeNet.

## ResNet<sup>6</sup>: Hiện tượng thoái biến

Hiện tượng **Thoái biến** (Degradation) trong mạng nơ-ron sâu đề cập đến hiện tượng không mong đợi khi tăng độ sâu của mạng: độ chính xác của mạng bắt đầu giảm mạnh mẽ, ngay cả khi không có vấn đề quá khớp (overfitting).

- Các mạng sâu hơn và phức tạp hơn có thể khó tối ưu hóa hơn bởi vì gradient có thể biến mất (vanishing) hoặc bùng nổ (exploding) khi lan truyền ngược qua nhiều lớp, làm cho việc cập nhật trọng số trở nên không ổn định.
- Khi thêm các lớp, không gian các hàm số mà mô hình có thể biểu diễn tăng lên, có thể làm cho quá trình học tập trở nên khó khăn hơn.

Trong trường hợp của ResNet, vấn đề Thoái biến được giải quyết thông qua việc sử dụng các “đường dẫn tắt” (shortcut connections) hay các ánh xạ thặng dư, giúp đưa thông tin qua các lớp mà không bị mất mát, cho phép mạng học tốt hơn, ngay cả khi số lượng lớp tăng lên.

---

<sup>6</sup>Deep Residual Learning for Image Recognition, 2015

## ResNet: Học thặng dư

Mục tiêu chính của **Học thặng dư** (Residual Learning) là giải quyết vấn đề thoái biến và giúp huấn luyện các mạng sâu một cách hiệu quả hơn.

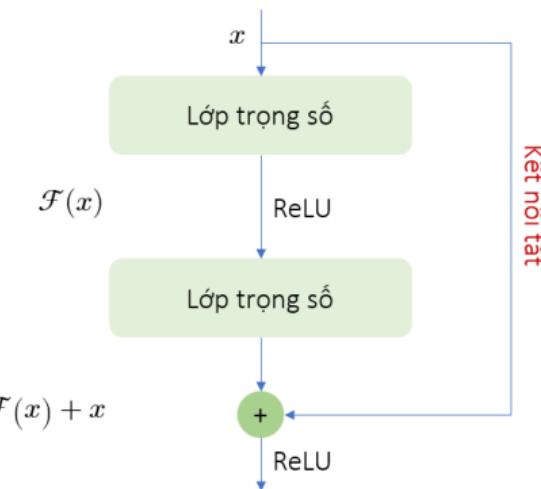
Xem xét  $\mathcal{H}(\mathbf{x})$  là một ánh xạ cơ bản cần được khớp bằng vài lớp xếp chồng (không nhất thiết là toàn bộ mạng), với  $\mathbf{x}$  ký hiệu đầu vào cho lớp đầu tiên trong số này.

- Giả định rằng nhiều lớp phi tuyến tính có thể xấp xỉ các hàm phức tạp theo cách tiệm cận, thì cũng tương đương với việc giả định rằng, chúng hoàn toàn có thể xấp xỉ các hàm dư, tức là  $\mathcal{H}(\mathbf{x}) - \mathbf{x}$ .
- Thay vì mong đợi các lớp xếp chồng xấp xỉ  $\mathcal{H}(\mathbf{x})$ , chúng ta sẽ để những lớp này xấp xỉ một hàm dư  $\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}$ . Hàm gốc do đó trở thành  $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$ .

Với việc tái cấu trúc thông qua Học thặng dư, nếu ánh xạ đồng nhất là tối ưu, các lời giải có thể đơn giản điều hướng trọng số của nhiều lớp phi tuyến tính về không để tiếp cận ánh xạ đồng nhất.

## ResNet: Khối thặng dư

**Khối thặng dư** (Residual Block) là đơn vị cấu trúc của mạng nơ-ron thặng dư (Residual Network). Trong một khối thặng dư cơ bản, thông thường sẽ có hai hay nhiều lớp tích chập, hàm kích hoạt ReLU và một kết nối tắt.

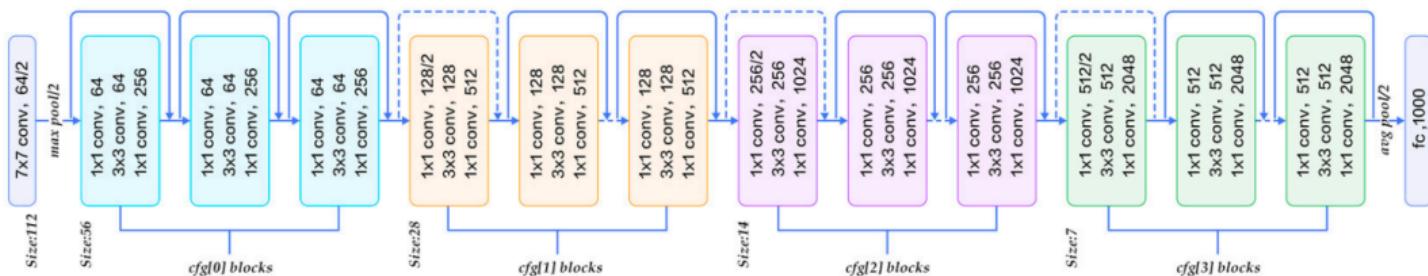


Hình 27: Minh họa kiến trúc một khối thặng dư với 2 lớp trọng số.

## ResNet: Kiến trúc

ResNet được giới thiệu bởi Kaiming He, Xiangyu Zhang, Shaoqing Ren, và Jian Sun vào năm 2015. ResNet đã đạt được sự chú ý lớn trong cộng đồng học sâu do khả năng xây dựng các CNN với độ sâu đáng kinh ngạc, lên tới hàng trăm lớp.

- ResNet được xây dựng từ nhiều khối thặng dư.
- Sau mỗi lớp tích chập là lớp chuẩn hóa theo lô (Batch Normalization) và hàm kích hoạt ReLU. Lớp này giúp mô hình giảm sự phụ thuộc vào khởi tạo trọng số, tăng tốc độ hội tụ trong khi huấn luyện.

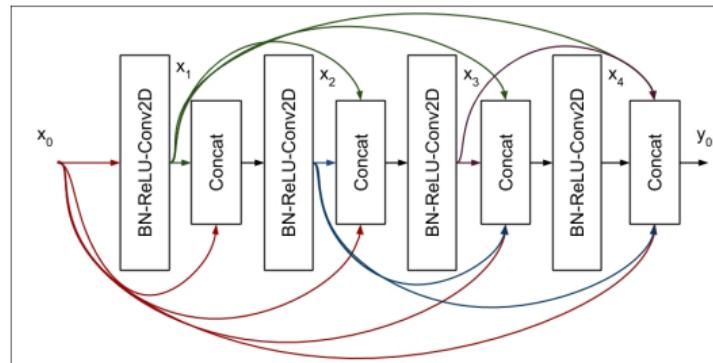


Hình 28: Minh họa kiến trúc ResNet-50.

## DenseNet<sup>7</sup>: Khối dày đặc

DenseNet đã được giới thiệu vào năm 2016 và đã nhanh chóng trở thành một trong những kiến trúc quan trọng trong lĩnh vực thị giác máy tính.

- Kiến trúc của DenseNet bao gồm nhiều **Khối dày đặc** (Dense Block) và các **Lớp chuyển tiếp** (Transition Layer) giữa chúng.
- Trong một **Khối dày đặc**, mỗi lớp nhận đầu vào từ tất cả các lớp trước đó.

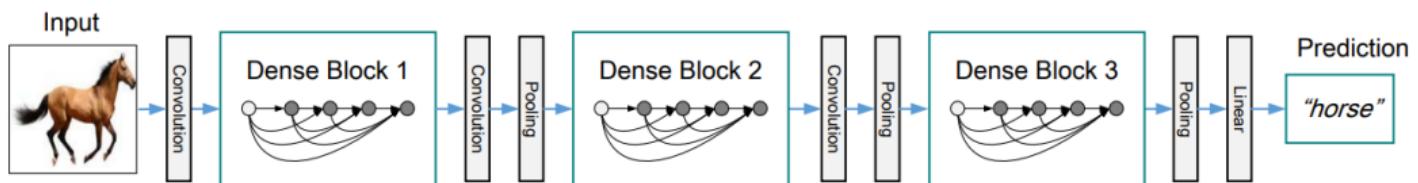


Hình 29: Minh họa một khối dày đặc.

<sup>7</sup>Densely Connected Convolutional Networks

## DenseNet: Kiến trúc

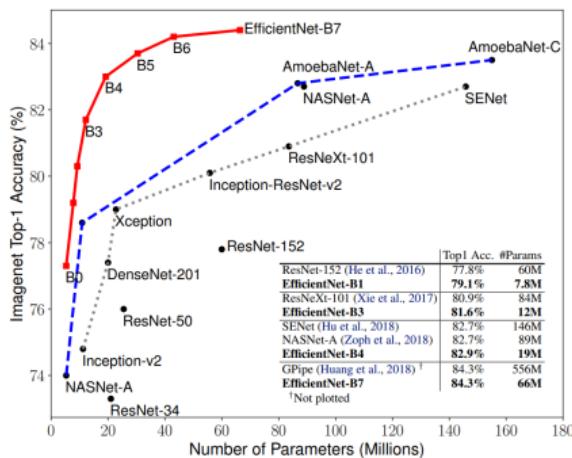
- Giữa các Khối dày đặc, DenseNet sử dụng các lớp chuyển tiếp để điều chỉnh kích thước của đặc trưng và kênh.
- Mục đích của các lớp chuyển tiếp là giảm kích thước không gian của đặc trưng, qua đó giúp giảm số lượng tham số và tính toán cần thiết, đồng thời duy trì thông tin quan trọng.
- Mạng thường kết thúc bằng lớp **Gộp trung bình toàn cục** (Global Average Pooling) và một lớp kết nối đầy đủ để phân loại.



Hình 30: Minh họa kiến trúc DenseNet với 3 khối dày đặc.

## EfficientNet<sup>8</sup>: Giới thiệu

EfficientNet là một kiến trúc CNN tiên tiến, được thiết kế để cung cấp hiệu suất xử lý hình ảnh cao với hiệu quả tài nguyên tối ưu. Được giới thiệu bởi Mingxing Tan và Quoc V. Le trong một bài báo năm 2019, EfficientNet đã nhanh chóng trở thành một trong những mô hình phổ biến cho các ứng dụng thị giác máy tính.



Hình 31: Kích thước mô hình các mạng CNN nổi tiếng và độ chính xác trên tập dữ liệu ImageNet (14 triệu bức ảnh).

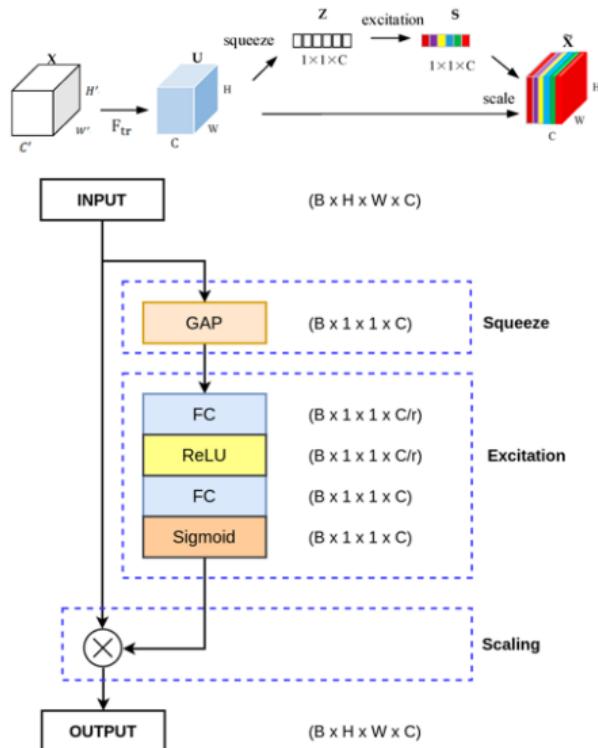
<sup>8</sup>EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, 2019

## EfficientNet: Khối nén kích thích

EfficientNet bắt đầu với phiên bản cơ bản là EfficientNet-B0, được xây dựng dựa trên kiến trúc cơ sở với một số lớp tích chập và **Khối nén kích thích** (Squeeze-and-Excitation Block - SE).

Khối nén kích thích là một thành phần đổi mới trong kiến trúc CNN. Cấu trúc này nhằm tăng cường khả năng của mạng để tập trung vào các đặc trưng quan trọng:

- **Squeeze** (nén): đầu ra của mỗi lớp tích chập được tổng hợp thông qua gộp trung bình toàn cục (Global Average Pooling). Điều này tạo ra một vector với kích thước bằng số lượng kênh, mỗi phần tử biểu diễn tính năng tổng quát của mỗi kênh.
- **Excitation** (kích thích): vector thu được sau quá trình nén sau đó được chuyển qua một mạng kết nối đầy đủ, thường bao gồm 2 lớp, để học một tập trọng số. Cho phép mô hình “tập trung” hoặc “kích thích” những kênh quan trọng hơn trong khi giảm sự chú ý đối với những kênh ít quan trọng.
- **Scaling** (định cỡ): cuối cùng, vector trọng số được nhân với đầu ra gốc của lớp tích chập để điều chỉnh các kênh dựa trên mức độ quan trọng của chúng.



Hình 32: Minh họa cấu trúc của khối nén kích thích.

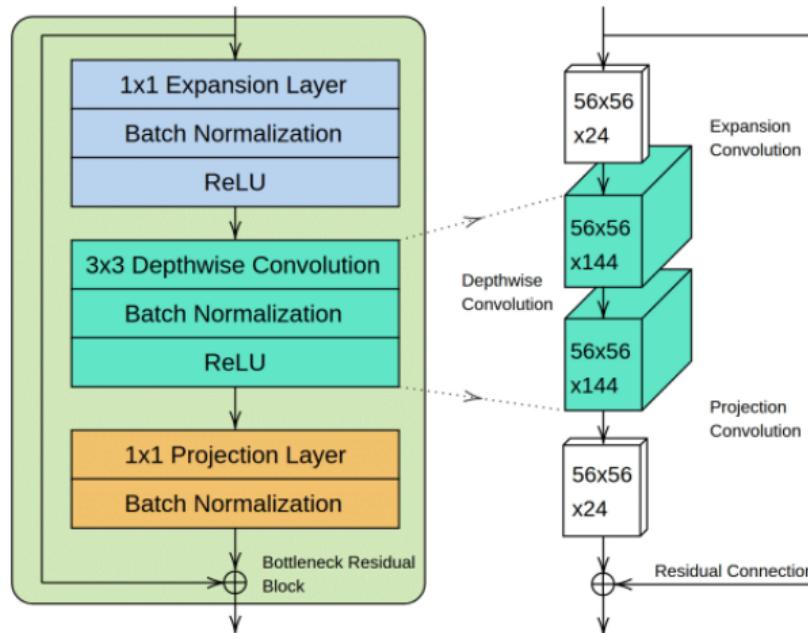
## EfficientNet: Lớp nghịch đảo cổ chai

**Lớp nghịch đảo cổ chai<sup>9</sup>** (Mobile Inverted Bottleneck Convolution - MBConv) là một sự cải tiến của các khối tích chập truyền thống, cung cấp hiệu quả tài nguyên tốt hơn cho các mạng sâu. Cấu trúc của MBConv bao gồm:

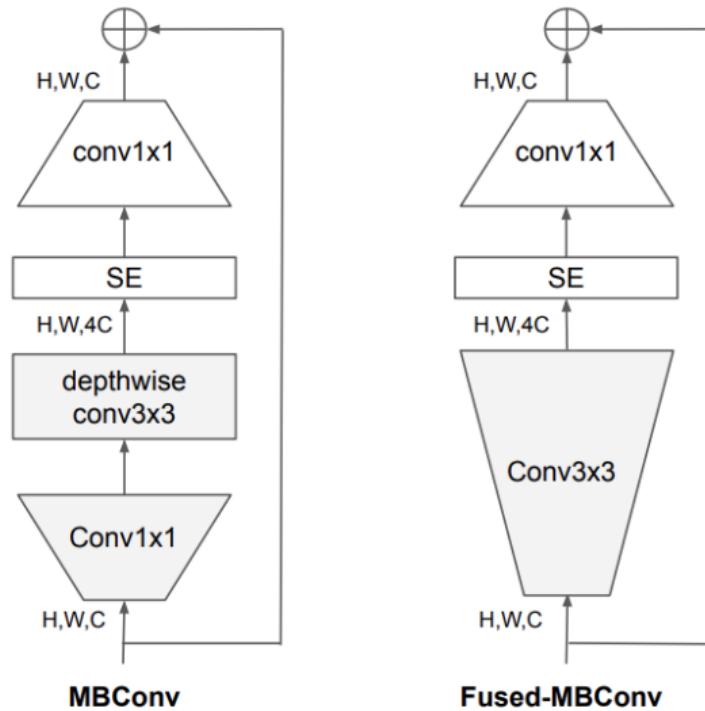
- **Inverted Residuals** (thặng dư nghịch đảo): Điểm khác biệt chính của MBConv so với các khối tích chập thông thường là nó sử dụng cấu trúc thặng dư nghịch đảo. Trong cấu trúc này, việc mở rộng kích thước kênh diễn ra trước khi áp dụng tích chập theo chiều sâu (depthwise convolution), sau đó kích thước kênh được thu hẹp lại.
- **Depthwise Convolution** (Tích chập theo chiều sâu): MBConv sử dụng tích chập sâu để xử lý mỗi kênh đầu vào một cách độc lập. Điều này giúp giảm số lượng tham số và tính toán cần thiết.
- **Linear Bottleneck** (Cổ chai tuyến tính): Phần cuối của MBConv thường bao gồm một lớp tích chập tuyến tính (linear convolution) để giảm số lượng kênh, qua đó giảm độ phức tạp và tăng hiệu quả.

---

<sup>9</sup>MobileNetV2: Inverted Residuals and Linear Bottlenecks



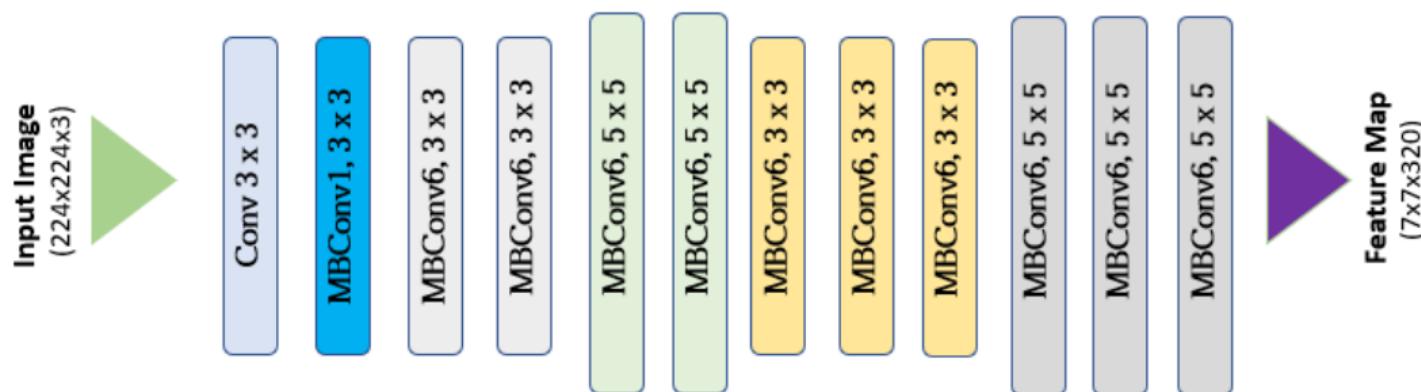
Hình 33: Minh họa cấu trúc của khối MBConv.



Hình 34: Minh họa cấu trúc MBConv (trong EfficientNetV1) và Fused-MBConv (trong EfficientNetV2).

## EfficientNet: Kiến trúc

EfficientNet sử dụng một hệ số phức hợp (compound coefficient) để mở rộng mô hình theo cả 3 chiều (độ sâu, độ rộng, độ phân giải), giúp cải thiện hiệu suất mà không làm tăng quá nhiều về kích thước mô hình hoặc yêu cầu tính toán.



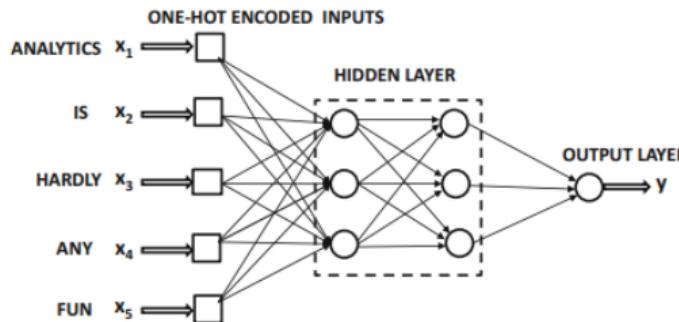
Hình 35: Minh họa kiến trúc EfficientNet xử lý hình ảnh kích cỡ  $224 \times 224 \times 3$ .

## Dạng dữ liệu phụ thuộc tuần tự

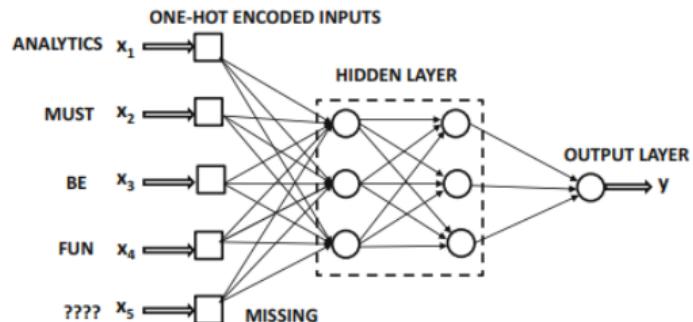
Một số mạng nơ-ron được thiết kế vốn dĩ cho dữ liệu đa chiều mà trong đó các thuộc tính phần lớn độc lập với nhau. Tuy nhiên, một số loại dữ liệu như *dữ liệu chuỗi thời gian*, *văn bản* và *dữ liệu sinh học* chứa sự phụ thuộc tuần tự giữa các thuộc tính. Ví dụ:

- Trong một tập dữ liệu chuỗi thời gian, các giá trị tại các mốc thời gian liên tiếp có liên quan chặt chẽ với nhau. Nếu sử dụng các giá trị của các mốc thời gian này như các đặc trưng độc lập, thì thông tin quan trọng về các mối quan hệ giữa các giá trị của các mốc thời gian này sẽ bị mất.
- Mặc dù văn bản thường được xử lý như một túi từ, người ta có thể nhận được thông tin ngữ nghĩa tốt hơn khi sử dụng thứ tự của các từ.
- Dữ liệu sinh học thường chứa các chuỗi, trong đó các biểu tượng có thể tương ứng với các axit amin hoặc một trong các nucleobase tạo thành các tế bào xây dựng của DNA.

Các giá trị riêng lẻ trong một chuỗi có thể là giá trị thực hoặc biểu tượng. Chuỗi giá trị thực cũng được gọi là chuỗi thời gian. Mạng hồi quy có thể được sử dụng cho cả hai loại dữ liệu.



(a) 5-word sentence  
“Analytics is hardly any fun.”



(b) 4-word sentence  
“Analytics must be fun.”

**Hình 36:** Sử dụng một mạng nơ-ron thông thường cho phân tích cảm xúc gặp khó khăn khi đầu vào có độ dài biến đổi. Kiến trúc mạng cũng không chứa bất kỳ thông tin hữu ích nào về các phụ thuộc tuần tự giữa các từ liên tiếp.

Nhiều ứng dụng tập trung vào chuỗi như văn bản thường được xử lý như “túi từ”, phương pháp này bỏ qua thứ tự của các từ, và hoạt động tốt cho các tài liệu có kích thước hợp lý.

Tuy nhiên, trong các ứng dụng mà việc giải thích ngữ nghĩa của câu là quan trọng, hoặc trong đó kích thước của phần văn bản tương đối nhỏ (ví dụ: một câu đơn), phương pháp như vậy đơn giản là không đủ. Xét cặp câu sau:

Con mèo đuổi theo con chuột.

Con chuột đuổi theo con mèo.

Hai câu rõ ràng rất khác nhau (và câu thứ hai là không thường).

- Biểu diễn túi từ sẽ coi chúng là giống nhau, loại biểu diễn này hoạt động tốt cho các ứng dụng đơn giản hơn (như phân loại).
- Một mức độ thông minh ngôn ngữ lớn hơn là cần thiết cho các ứng dụng phức tạp hơn trong các cài đặt khó khăn như *phân tích cảm xúc*, *dịch máy*, hoặc *trích xuất thông tin*.

Hai yêu cầu chính cho việc xử lý chuỗi bao gồm:

- i Khả năng nhận và xử lý đầu vào theo cùng một thứ tự như chúng xuất hiện trong chuỗi.
- ii Xử lý đầu vào tại mỗi mốc thời gian theo cách tương tự liên quan đến lịch sử đầu vào trước đó.

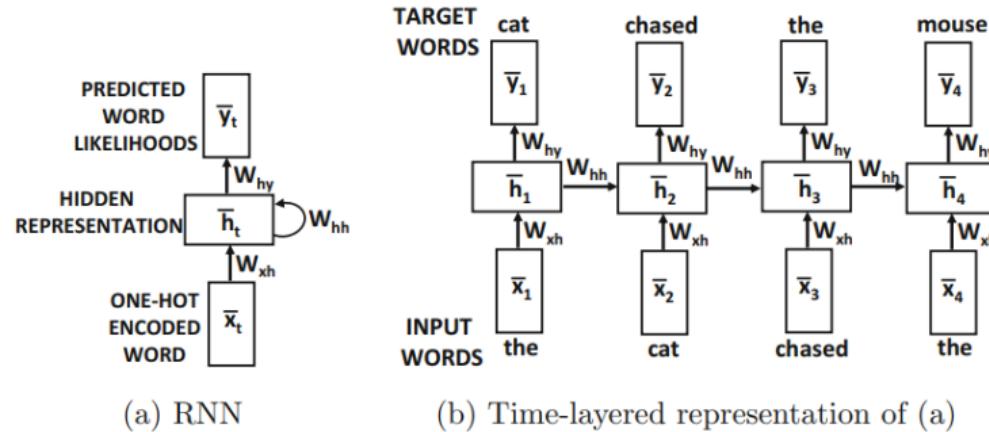
Một thách thức chính là chúng ta cần xây dựng một cách nào đó một mạng nơ-ron với một số lượng tham số cố định, nhưng với khả năng xử lý một số lượng đầu vào biến đổi.

## Mạng nơ-ron hồi quy

Những yêu cầu trước đó được đáp ứng một cách tự nhiên với việc sử dụng mạng nơ-ron hồi quy (RNN). Trong RNN, có một sự tương ứng một-một giữa các lớp trong mạng và các vị trí cụ thể trong chuỗi. Vị trí trong chuỗi cũng được gọi là mốc thời gian của nó.

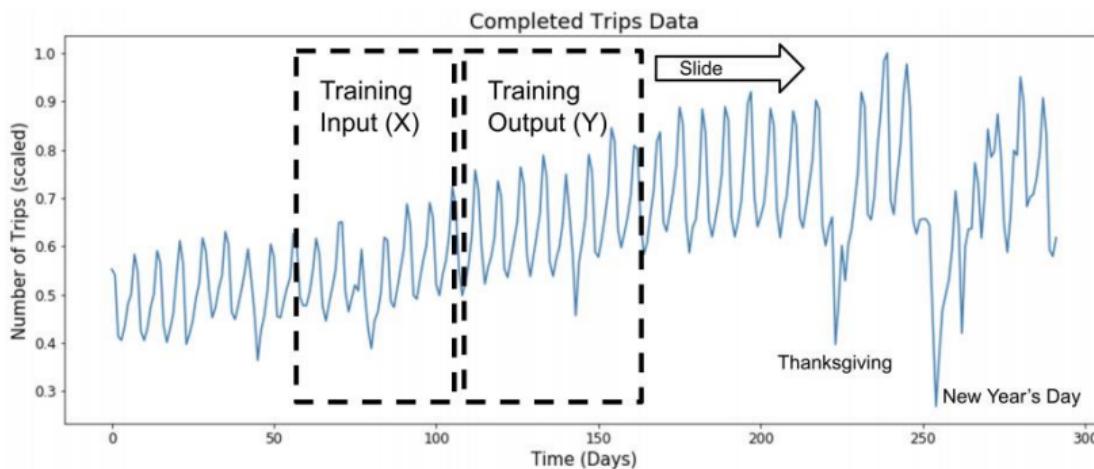
- Thay vì có một số lượng đầu vào biến đổi trong một lớp đầu vào duy nhất, mạng chứa một số lượng lớp biến đổi, và mỗi lớp có một đầu vào duy nhất tương ứng với mốc thời gian đó.
- Các đầu vào được phép tương tác trực tiếp với các lớp ẩn hạ lưu (down-stream hidden layers) tùy thuộc vào vị trí của chúng trong chuỗi.
- Mỗi lớp sử dụng cùng một bộ tham số để đảm bảo mô hình tương tự tại mỗi mốc thời gian, và do đó số lượng tham số cũng được cố định.

- Đầu vào có thể là một chuỗi từ, và đầu ra có thể là cùng một chuỗi được dịch chuyển 1, để chúng ta đang dự đoán từ tiếp theo tại bất kỳ điểm nào.
- Đây là một mô hình ngôn ngữ cổ điển trong đó chúng ta đang cố gắng dự đoán từ tiếp theo dựa trên lịch sử tuần tự của các từ.
  - Các mô hình ngôn ngữ có rất nhiều ứng dụng trong khai thác văn bản và truy xuất thông tin.



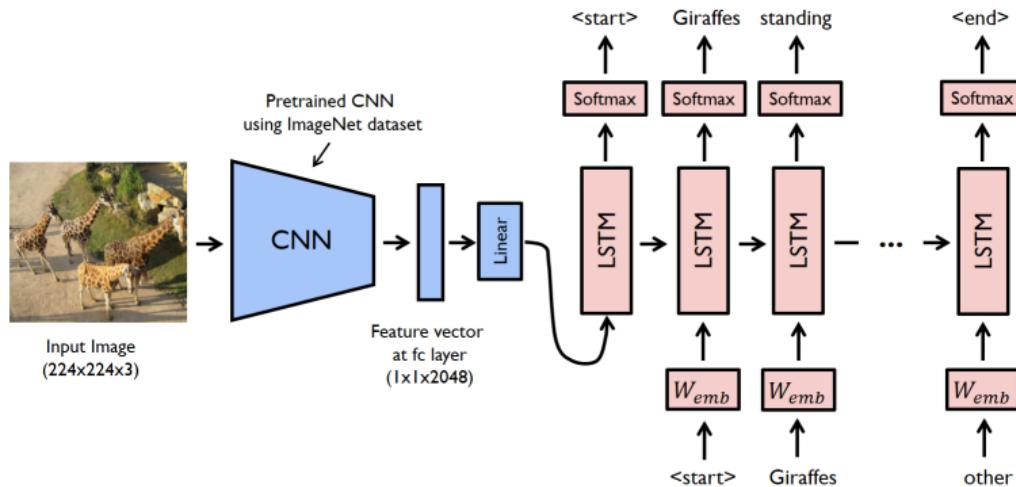
Hình 37: Một RNN với biểu diễn lớp thời gian.

2. Trong một chuỗi thời gian giá trị thực, vấn đề học phần tử tiếp theo tương đương với phân tích tự hồi quy. Tuy nhiên, một RNN có thể học được nhiều mô hình phức tạp hơn so với những mô hình thu được với mô hình hóa chuỗi thời gian truyền thống.



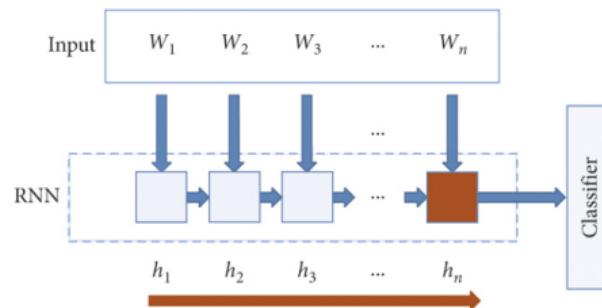
Hình 38: Phân tích tự hồi quy chuỗi thời gian

3. Đầu vào có thể là một câu trong một ngôn ngữ, và đầu ra có thể là một câu trong một ngôn ngữ khác. Trong trường hợp này, người ta có thể kết nối hai RNN để học các mô hình dịch giữa hai ngôn ngữ. Người ta thậm chí có thể kết nối một RNN với một loại mạng khác (ví dụ: CNN) để học các chú thích của hình ảnh.



Hình 39: Mô hình tạo chú thích hình ảnh.

4. Đầu vào có thể là một chuỗi (ví dụ: câu), và đầu ra có thể là một vector của các xác suất lớp, được kích hoạt bởi cuối câu. Phương pháp này hữu ích cho các ứng dụng phân loại tập trung vào câu như phân tích cảm xúc.



Hình 40: Minh họa quy trình phân tích cảm xúc

Có những thách thức đáng kể trong việc học các tham số của RNN. Một trong những vấn đề chính về tiêu biến và bùng nổ gradient. Do đó, một số biến thể của mạng neural tuần tự, như bộ nhớ dài ngắn hạn (LSTM) và đơn vị hồi quy có công (GRU), đã được đề xuất.

## Khả năng biểu diễn mạng hồi quy

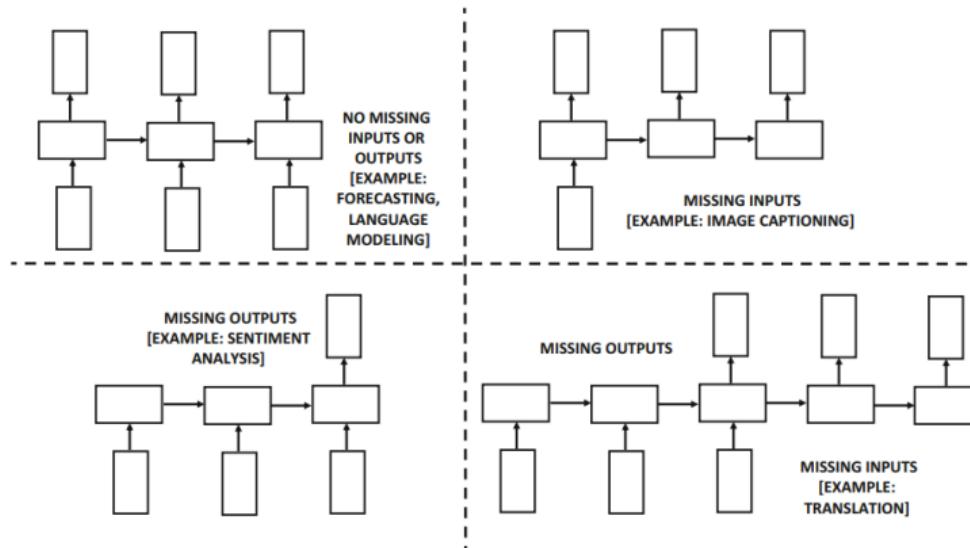
Mạng nơ-ron hồi quy được biết đến là *máy Turing hoàn chỉnh* (Turing complete), nghĩa là một mạng nơ-ron hồi quy có thể mô phỏng bất kỳ thuật toán nào, miễn là có đủ dữ liệu và tài nguyên tính toán.

- Tuy nhiên, tính chất này không thực sự hữu ích trong thực tế vì lượng dữ liệu và tài nguyên tính toán cần thiết để đạt được mục tiêu này trong các cài đặt tùy ý có thể không thực tế.
- Hơn nữa, có những vấn đề thực tế trong việc huấn luyện RNN, chẳng hạn như các vấn đề về đạo hàm biến mất và bùng nổ. Những vấn đề này tăng lên với độ dài của chuỗi, và các biến thể ổn định hơn như LSTM chỉ có thể giải quyết vấn đề này một cách có hạn.

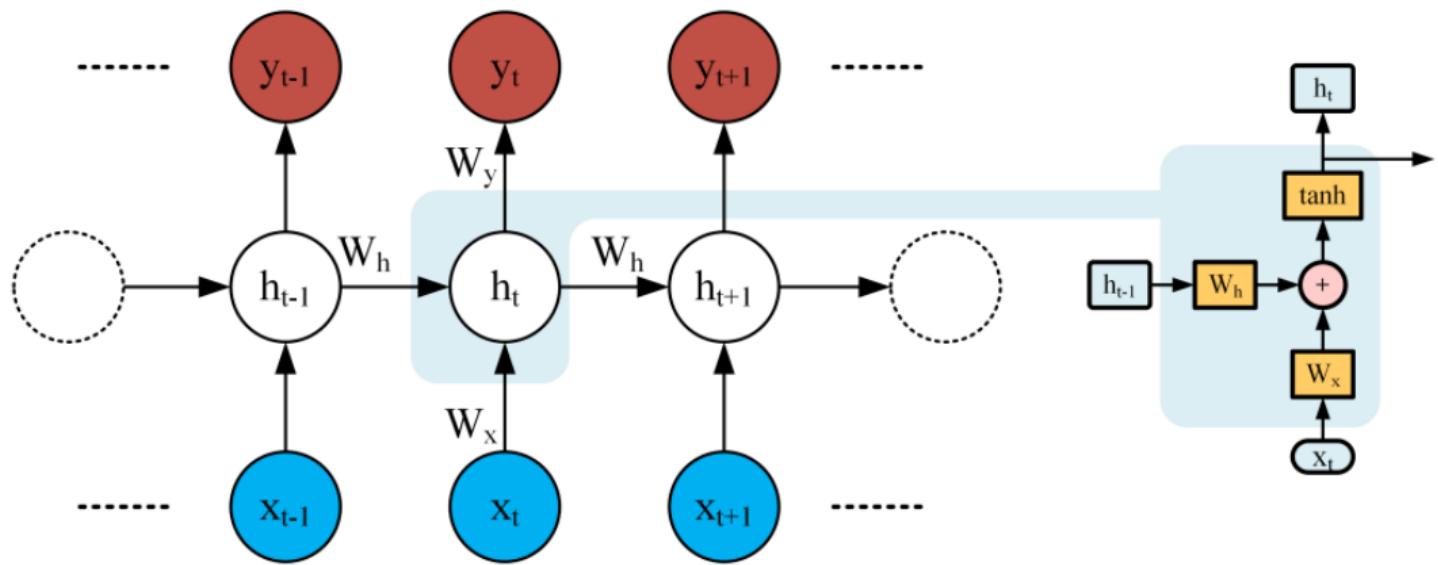
Một máy Turing có thể được chứng minh là tương đương với một mạng nơ-ron hồi quy, và nó thường sử dụng một mạng hồi quy truyền thống hơn, được gọi là bộ điều khiển (controller), như một thành phần quyết định hành động quan trọng.

# Kiến trúc mạng hồi quy

Mỗi mốc thời gian có một đầu vào, đầu ra, và đơn vị ẩn. Trên thực tế, có thể có đầu vào hoặc đầu ra bị thiếu tại bất kỳ mốc thời gian cụ thể nào.



Hình 41: Các ví dụ về trường hợp với đầu vào và đầu ra bị thiếu



Hình 42: Mô hình mạng hồi quy

- Tại mốc thời gian thứ  $t$  có vector đầu vào  $x_t$ , trạng thái ẩn  $h_t$  và vector đầu ra  $y_t$ .
- Số chiều của  $x_t$  và  $y_t$  là  $d$ , của  $h_t$  là  $p$  (giả sử chúng là các vector cột).
- Trạng thái ẩn tại mốc thời gian  $t$  là một hàm tổng hợp bởi trạng thái ẩn tại mốc thời gian  $(t - 1)$  và vector đầu vào tại  $t$ , hay:

$$h_t = f(h_{t-1}, x_t)$$

Hàm này được định nghĩa với việc sử dụng các ma trận trọng số và các hàm kích hoạt, và *cùng một trọng số được sử dụng tại mỗi mốc thời gian*. Do đó, mặc dù trạng thái ẩn thay đổi theo thời gian, các trọng số và hàm cơ bản  $f(\cdot, \cdot)$  vẫn cố định qua tất cả các mốc thời gian sau khi mạng đã được huấn luyện.

- Một hàm riêng biệt  $y_t = g(h_t)$  được sử dụng để học các xác suất đầu ra từ trạng thái ẩn.

Để định nghĩa các hàm  $f(\cdot, \cdot)$  và  $g(\cdot)$  cụ thể, ta gọi:

- $W_x$  là ma trận đầu vào - ẩn kích thước  $p \times d$ .
- $W_h$  là ma trận ẩn - ẩn kích thước  $p \times p$ .
- $W_y$  là ma trận ẩn - đầu ra kích thước  $d \times p$ .

Khi đó:

$$\begin{aligned} h_t &= \tanh(W_h h_{t-1} + W_x x_t) \\ y_t &= W_y h_t \end{aligned}$$

Ký hiệu tanh được sử dụng một cách lỏng lẻo, với ý nghĩa rằng hàm được áp dụng cho vector cột  $p$  chiều theo cách từng phần tử để tạo ra một vector  $p$  chiều với mỗi phần tử trong  $[-1, 1]$ . Ngoài ra, có thể dùng hàm sigmoid.

Tại mốc thời gian đầu tiên,  $h_{t-1}$  được giả định là một vector hằng số mặc định nào đó (như 0), vì không có đầu vào từ lớp ẩn ở đầu câu. Người ta cũng có thể học vector này, nếu muốn.

Do bản chất đệ quy, mạng hồi quy có khả năng tính toán một hàm của đầu vào độ dài biến đổi. Nói cách khác, người ta có thể mở rộng sự đệ quy  $h_t = f(h_{t-1}, x_t)$  để định nghĩa hàm cho  $h_t$  theo  $t$  đầu vào.

Ví dụ, bắt đầu từ  $h_0$ , thường được cố định thành một vector hằng số nào đó (như vector 0), chúng ta có  $h_1 = f(h_0, x_1)$  và  $h_2 = f(f(h_0, x_1), x_2)$ . Lưu ý rằng  $h_1$  là một hàm của chỉ  $x_1$ , trong khi  $h_2$  là một hàm của cả  $x_1$  và  $x_2$ . Do đó,  $h_t$  là một hàm của  $x_1, x_2, \dots, x_t$ . Vì đầu ra  $y_t$  là một hàm của  $h_t$ , những tính chất này được kế thừa bởi  $y_t$  cũng vậy. Nói chung, chúng ta có thể viết như sau:

$$y_t = F_t(x_1, x_2, \dots, x_t)$$

Hàm  $F_t(\cdot)$  thay đổi với giá trị của  $t$  mặc dù mối quan hệ của nó với trạng thái ngay trước đó luôn giống nhau.

## Lan truyền ngược trong mạng hồi quy

Các logarit âm của xác suất softmax của các từ chính xác tại các mốc thời gian khác nhau được tổng hợp để tạo ra hàm mất mát. Nếu vector đầu ra  $y_t$  có thể được viết dưới dạng  $[\hat{y}_t^1, \hat{y}_t^2, \dots, \hat{y}_t^d]$ , nó đầu tiên được chuyển đổi thành một vector của  $d$  xác suất bằng cách sử dụng hàm softmax:

$$[\hat{p}_t^1, \hat{p}_t^2, \dots, \hat{p}_t^d] = \text{softmax}([\hat{y}_t^1, \hat{y}_t^2, \dots, \hat{y}_t^d])$$

Nếu  $j_t$  là chỉ số của từ đúng tại thời điểm  $t$  trong dữ liệu huấn luyện, thì hàm mất mát  $L$  cho tất cả  $T$  mốc thời gian được tính như sau:

$$L = - \sum_{t=1}^T \ln (\hat{p}_t^{j_t})$$

## Tối ưu hàm mất mát

Đạo hàm hàm mất mát theo đầu ra được tính bởi:

$$\frac{\partial L}{\partial \hat{y}_t^k} = \hat{p}_t^k - I(k, j_t)$$

Ở đây,  $I(k, j_t)$  là một hàm chỉ thị, bằng 1 khi  $k$  và  $j_t$  giống nhau, và 0, nếu không. Bắt đầu với đạo hàm riêng này, người ta có thể sử dụng cập nhật lan truyền ngược đơn giản để tính toán các gradient liên quan đến trọng số trong các lớp khác nhau.

Mẹo chính để xử lý trọng số chia sẻ là đầu tiên “giả vờ” rằng các tham số trong các lớp thời gian khác nhau là độc lập với nhau. Vì mục đích này, chúng ta giới thiệu các biến thời gian  $W_x^{(t)}, W_h^{(t)}, W_y^{(t)}$  cho mỗi thời gian  $t$ .

- Lan truyền ngược thông thường đầu tiên được thực hiện bằng cách làm việc dưới sự giả định rằng các biến này khác nhau.
- Sau đó, các đóng góp của các mốc thời gian khác nhau của các trọng số đến gradient được cộng lại để tạo ra một cập nhật thống nhất cho mỗi trọng số.

Loại thuật toán lan truyền ngược đặc biệt này được gọi là *Lan truyền ngược qua thời gian* (Backpropagation Through Time - BPTT)

## Tóm tắt thuật toán

- Chạy đầu vào tuần tự theo hướng tiến qua thời gian và tính toán các lỗi tại mỗi mốc thời gian.
- Tính toán các gradient của trọng số cạnh theo hướng ngược lại trên mạng đã được mở rộng mà không quan tâm đến việc trọng số trong các lớp thời gian khác nhau được chia sẻ. Nói cách khác, giả định rằng các trọng số  $W_x^{(t)}, W_h^{(t)}, W_y^{(t)}$  tại mốc thời gian  $t$  khác biệt so với các mốc thời gian khác. Do đó, người ta có thể sử dụng lan truyền ngược thông thường để tính  $\frac{\partial L}{\partial W_x^{(t)}}, \frac{\partial L}{\partial W_h^{(t)}}, \frac{\partial L}{\partial W_y^{(t)}}$ .
- Cộng tất cả các trọng số (chia sẻ) tương ứng với các lan truyền ngược khác nhau của một cạnh theo thời gian. Nghĩa là:

$$\frac{\partial L}{\partial W_x} = \sum_{t=1}^T \frac{\partial L}{\partial W_x^{(t)}}, \quad \frac{\partial L}{\partial W_h} = \sum_{t=1}^T \frac{\partial L}{\partial W_h^{(t)}}, \quad \frac{\partial L}{\partial W_y} = \sum_{t=1}^T \frac{\partial L}{\partial W_y^{(t)}}$$

## Mạng hồi quy hai chiều

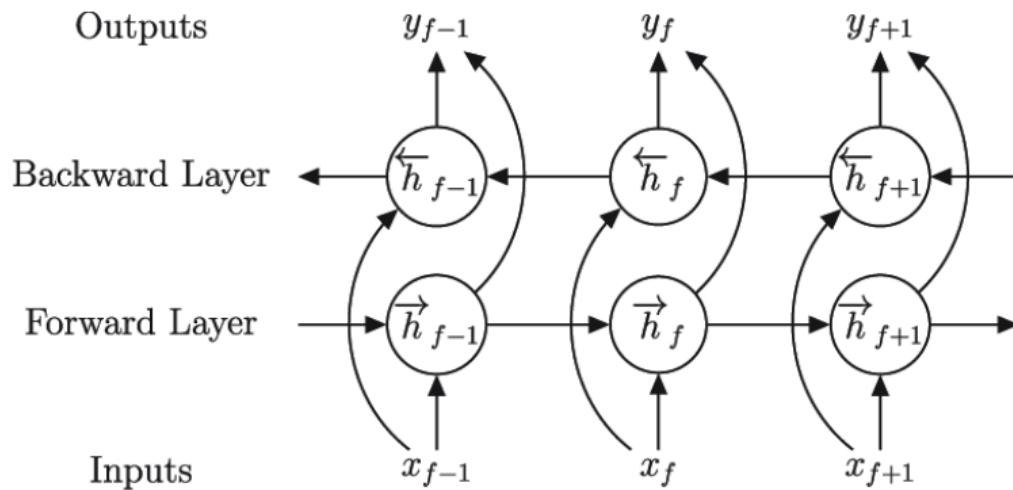
Mạng hồi quy mang nhược điểm là chỉ có kiến thức về các đầu vào trong quá khứ đến một điểm nhất định trong một câu, nhưng không có kiến thức về các trạng thái tương lai. Điều này gây hạn chế trong một số ứng dụng như mô hình hóa ngôn ngữ hay nhận dạng chữ viết tay.

- Trong mạng lưới tái phát hai chiều, chúng ta có các trạng thái ẩn riêng biệt  $h_t^{(f)}$  và  $h_t^{(b)}$  cho các hướng tiến và lùi.
- Các trạng thái ẩn tiến chỉ tương tác với nhau và điều này cũng đúng với các trạng thái ẩn lùi. Sự khác biệt chính là các trạng thái tiến tác động theo hướng tiến, trong khi các trạng thái lùi tác động theo hướng lùi.
- Tuy nhiên, cả  $h_t^{(f)}$  và  $h_t^{(b)}$  đều nhận đầu vào từ cùng một vector  $x_t$  và chúng tương tác với cùng một vector đầu ra  $y_t$ .

$$h_t^{(f)} = \tanh(W_h^{(f)} h_{t-1}^{(f)} + W_x^{(f)} x_t)$$

$$h_t^{(b)} = \tanh(W_h^{(b)} h_{t+1}^{(b)} + W_x^{(b)} x_t)$$

$$y_t = W_y^{(f)} h_t^{(f)} + W_y^{(b)} h_t^{(b)}$$



Hình 43: Mô hình mạng hồi quy hai chiều

## Mạng hồi quy đa lớp

Trong các ứng dụng thực tế, kiến trúc đa lớp được sử dụng để xây dựng các mô hình phức tạp hơn. Đầu tiên, chúng ta viết lại phương trình hồi quy của các lớp ẩn (đối với các mạng một lớp) theo một hình thức có thể được thích ứng dễ dàng với các mạng đa lớp:

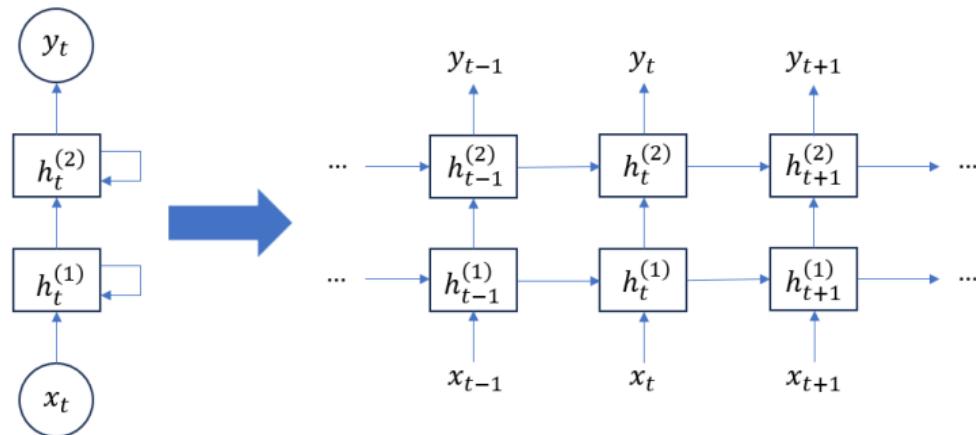
$$\begin{aligned} h_t &= \tanh(W_h h_{t-1} + W_x x_t) \\ &= \tanh\left(\underbrace{\begin{bmatrix} W_x & W_h \end{bmatrix}}_W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}\right) \end{aligned}$$

Hãy thêm một chỉ số phụ vào trạng thái ẩn và ký hiệu vector cho các trạng thái ẩn tại thời điểm  $t$  và lớp ẩn thứ  $k$  bằng  $h_t^{(k)}$ . Tương tự, hãy để ma trận trọng số cho lớp ẩn thứ  $k$  được ký hiệu bằng  $W^{(k)}$ . Các trọng số này được chia sẻ qua các thời điểm khác nhau (như trong mạng hồi quy một lớp), nhưng chúng không được chia sẻ qua các lớp khác nhau.

Ma trận  $W^{(1)}$  có kích thước  $p \times (d + p)$ , với  $k \geq 2$  ta có:

$$h_t^{(k)} = \tanh \left( W^{(k)} \begin{bmatrix} h_t^{(k-1)} \\ h_{t-1}^{(k)} \end{bmatrix} \right)$$

Trong trường hợp này  $W^{(k)}$  có kích thước  $p \times 2p$ .



Hình 44: Mô hình mạng hồi quy đa lớp

## Bộ nhớ dài - ngắn hạn

- Tiêu biến hoặc bùng nổ gradient là một vấn đề phổ biến trong cập nhật mạng nơ-ron, nơi mà việc nhân liên tiếp bởi ma trận  $W^{(k)}$  là không ổn định
- Loại bất ổn này là kết quả trực tiếp của việc nhân liên tiếp với ma trận trọng số (hồi quy) tại các dấu thời gian khác nhau.

Vì vậy, một mạng nơ-ron sử dụng cập nhật nhân là tốt chỉ trong việc học trên các chuỗi ngắn, và do đó nó vốn dĩ có bộ nhớ ngắn hạn tốt nhưng bộ nhớ dài hạn kém. Đây là một đặc tính có hữu của RNN:

- RNN truyền thông tự nhiên có khả năng hiệu quả trong việc nắm bắt và ghi nhớ thông tin từ các đầu vào gần đây trong một chuỗi. Kiến trúc của mạng cho phép nó duy trì thông tin qua vài bước thời gian, làm cho nó giỏi trong các nhiệm vụ cần xử lý thông tin ngắn hạn.
- Mặt khác, RNN truyền thông gặp khó khăn trong việc lưu giữ và nhớ thông tin từ các bước đầu tiên của chuỗi dữ liệu trong thời gian dài. Nghĩa là khi thông tin được truyền qua nhiều lớp hoặc qua nhiều bước thời gian, sự ảnh hưởng của nó dần dần giảm đi, làm cho mạng không thể học hiệu quả từ dữ liệu có mối liên quan dài hạn.

Để giải quyết vấn đề này, một giải pháp là thay đổi phương trình hồi quy cho vector ẩn bằng cách sử dụng LSTM. Các hoạt động của LSTM được thiết kế để có kiểm soát tinh vi đối với dữ liệu được ghi vào bộ nhớ dài hạn.

Ký hiệu  $h_t^{(k)}$  đại diện cho các trạng thái ẩn của lớp thứ  $k$  trong LSTM đa lớp. Để thuận tiện, giả định rằng lớp đầu vào  $x_t$  có thể được ký hiệu bởi  $h_t^{(0)}$  (mặc dù không phải ẩn).

- Vector đầu vào  $x_t$  là  $d$  chiều.
- Các trạng thái ẩn là  $p$  chiều.

LSTM là một cải tiến của kiến trúc mạng nơ-ron hồi quy đa lớp, trong đó chúng ta thay đổi điều kiện hồi quy của cách các trạng thái ẩn  $h_t^{(k)}$  được truyền đi. Để đạt được mục tiêu này, chúng ta có thêm một vector ẩn  $p$  chiều, được ký hiệu là  $c_t^{(k)}$  và được gọi là trạng thái tế bào (cell state).

Có thể xem trạng thái tế bào như một loại bộ nhớ dài hạn giữ lại ít nhất một phần thông tin trong các trạng thái trước đó bằng cách sử dụng sự kết hợp của các toán tử “quên một phần” (forgetting) và “tăng cường” (increment) trên các trạng thái tế bào trước đó.

Việc xác định vector trạng thái ẩn  $h_t^{(k)}$  và vector trạng thái tế bào  $c_t^{(k)}$  sử dụng một quá trình nhiều bước, đầu tiên là tính toán các biến trung gian  $i, f, o, \tilde{c}$  và sau đó là tính toán các biến ẩn từ các biến trung gian này. Công thức cập nhật như sau:

Input Gate (Cổng vào):

Forget Gate (Cổng quên):

Output Gate (Cổng ra):

New Cell State (Trạng thái mới):

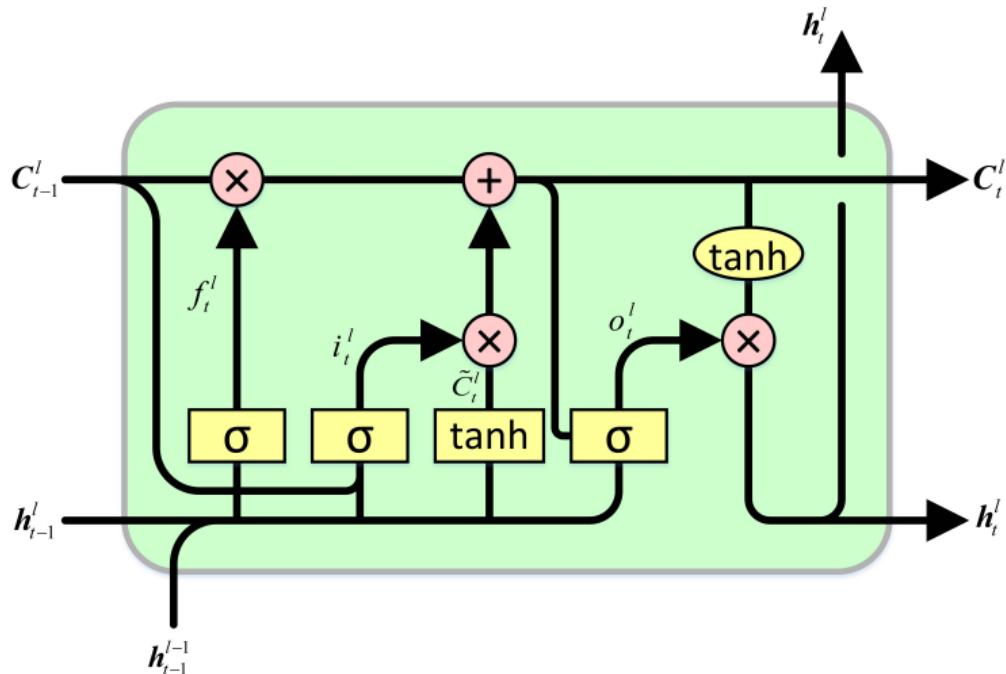
$$\begin{bmatrix} i \\ f \\ o \\ \tilde{c} \end{bmatrix} = \begin{pmatrix} \text{sigmoid} \\ \text{sigmoid} \\ \text{sigmoid} \\ \tanh \end{pmatrix} \left( W^{(k)} \begin{bmatrix} h_t^{(k-1)} \\ h_{t-1}^{(k)} \end{bmatrix} \right)$$

Chọn quên và thêm vào bộ nhớ dài:

$$c_t^{(k)} = f \odot c_{t-1}^{(k)} + i \odot \tilde{c}$$

Chọn lọc rò rỉ bộ nhớ dài sang trạng thái ẩn:

$$h_t^{(k)} = o \odot \tanh c_t^{(k)}$$



Hình 45: Cấu trúc một tế bào của Bộ nhớ ngắn - dài hạn.

Các biến trung gian  $i, f, o$  kiểm soát thông tin được lưu giữ, quên, hoặc chuyển qua mỗi tế bào của LSTM. Chúng hoạt động như các cổng logic, quyết định thông tin nào được thông qua.

Trong công thức cập nhật trạng thái tế bào:

$$c_t^{(k)} = \underbrace{f \odot c_{t-1}^{(k)}}_{\text{bị quên}} + \underbrace{i \odot \tilde{c}}_{\text{thông qua}}$$

- Phần đầu tiên  $(f \odot c_{t-1}^{(k)})$ : Sử dụng cổng quên  $f$  để quyết định phần nào của trạng thái tế bào trước đó sẽ được “quên” đi. Điều này được thực hiện bằng cách nhân từng phần tử của  $f$  (cổng quên) với  $c_{t-1}^{(k)}$  (trạng thái tế bào tại bước thời gian trước). Những phần tử có giá trị cao trong  $f$  cho thấy các phần tử tương ứng trong  $c_{t-1}^{(k)}$  nên được giữ lại, trong khi giá trị thấp cho thấy nó nên bị đặt lại về 0 (quên đi).
- Phần thứ hai  $(i \odot \tilde{c})$ : Sử dụng cổng đầu vào  $i$  để quyết định thông tin mới  $\tilde{c}$  (đề xuất về trạng thái tế bào mới) nào sẽ được thêm vào trạng thái tế bào hiện tại. Lại một lần nữa, phép nhân từng phần tử giữa  $i$  và  $\tilde{c}$  quyết định thông tin nào sẽ “được thông qua”.

## Khát quát về Học sâu

### Giới thiệu các kiến trúc mạng

Kiến trúc mạng tích chập

Các kiến trúc mạng tích chập hiện đại

Kiến trúc mạng hồi quy

Các kiến trúc mạng hồi quy hiện đại

### Phát biểu bài toán

Khái quát lĩnh vực nghiên cứu

Bộ dữ liệu kiến trúc làm chuẩn

Ứng dụng và các thành tựu đạt được

### Các phương pháp tiếp cận

Các giải thuật phỏng tự nhiên

Các giải thuật dựa trên vi phân

Kỹ thuật học tăng cường

## Khát quát về Tìm kiếm kiến trúc mạng<sup>10,11</sup>

Học sâu đã trở thành mô hình chủ đạo trong Học máy cho nhiều ứng dụng, đạt được nhiều đột phá trong thị giác máy tính, hiểu ngôn ngữ tự nhiên, nhận dạng giọng nói và học tăng cường.

Nhiều yếu tố đã đóng góp vào sự tăng trưởng của các phương pháp học sâu:

- **Khả năng tự động hóa** việc trích xuất đặc trưng.
- **Sự tăng lên** của dữ liệu.
- **Sự phát triển** của nguồn lực tính toán.

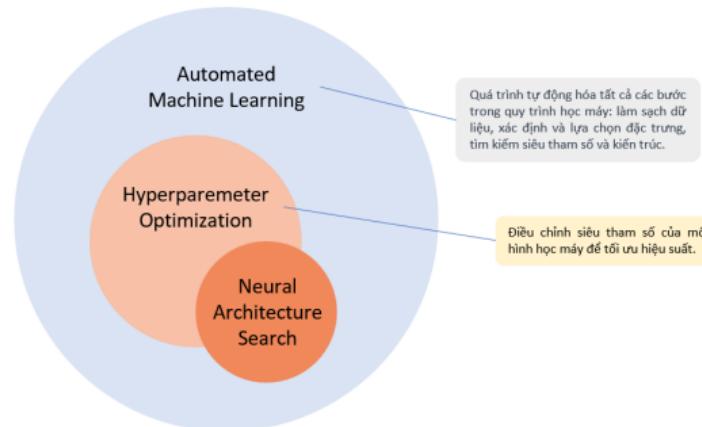
Tuy nhiên, thiết kế các kiến trúc mạng hiệu suất cao quyết định đến sự thành công của học sâu.

---

<sup>10</sup>Neural Architecture Search: Insights from 1000 Papers, 1/2023

<sup>11</sup>Neural Architecture Search: A Survey, 4/2019

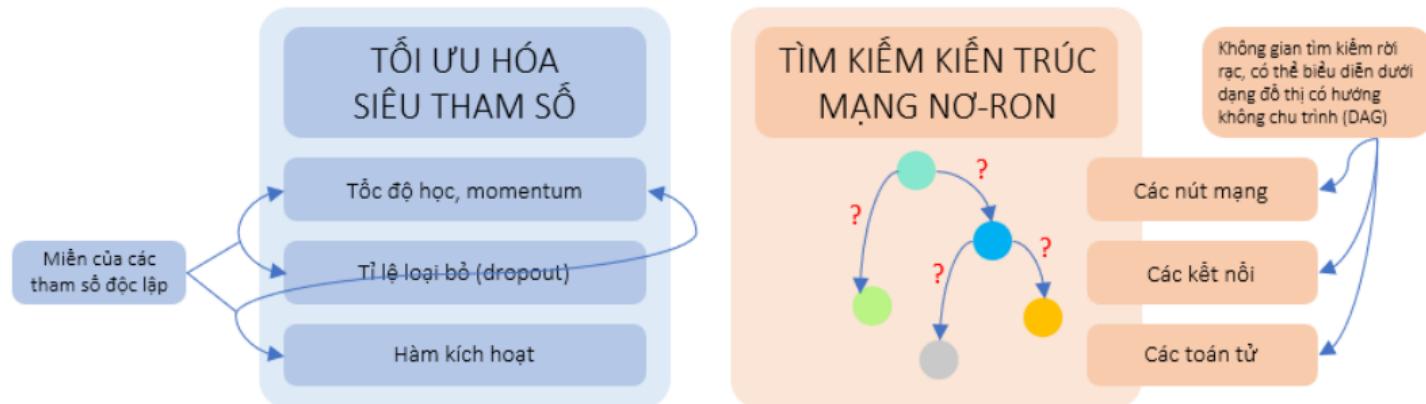
**Tìm kiếm kiến trúc mạng nơ-ron** (Neural Architecture Search - NAS) là quá trình tự động hóa việc thiết kế kiến trúc mạng nơ-ron cho một nhiệm vụ cụ thể, đã vượt qua những kiến trúc tốt nhất thiết kế bởi con người trên nhiều nhiệm vụ.



Hình 46: Mô tả lĩnh vực Tìm kiếm kiến trúc mạng

NAS là lĩnh vực con của Tối ưu hóa siêu tham số (HPO), trong đó NAS tập trung tìm kiếm các siêu tham số tương ứng với kiến trúc. Do đó, NAS cũng là một phần của Tự động hóa học máy (AutoML). Tuy nhiên, kỹ thuật NAS và HPO thường khác nhau.

HPO tiêu biểu tối ưu hóa sự kết hợp của các siêu tham số liên tục và rời rạc, như tốc độ học, tỷ lệ loại bỏ (dropout), kích thước lô (batch size), mô-men (momentum), hàm kích hoạt, chiến lược chuẩn hóa,... NAS tập trung cụ thể vào việc tối ưu hóa kiến trúc, và phức tạp hơn nhiều.

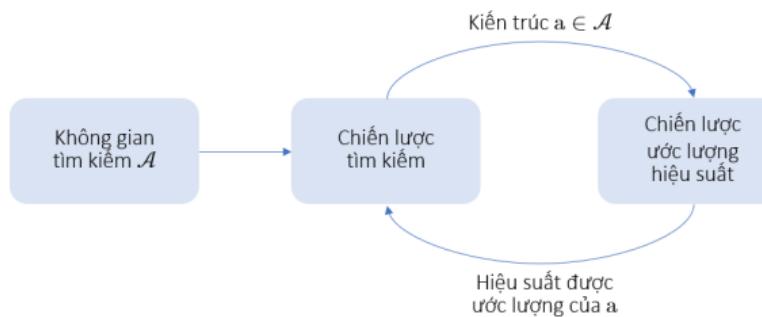


Hình 47: Các mục tiêu tìm kiếm của HPO nói chung và NAS nói riêng

# Phân loại các phương pháp NAS

Các phương pháp NAS có thể phân loại dựa trên 3 khía cạnh:

- **Không gian tìm kiếm** (Search Space) là tập hợp tất cả các kiến trúc mà thuật toán NAS được phép chọn. Không gian tìm kiếm NAS thường có kích thước vô cùng lớn.
- **Chiến lược tìm kiếm** (Search Strategy) là kỹ thuật tối ưu hóa được sử dụng để tìm một kiến trúc hiệu suất cao trong không gian tìm kiếm.
- **Chiến lược ước lượng hiệu suất** (Performance Estimation Strategy) là bất kỳ phương pháp nào được sử dụng để dự đoán nhanh chóng hiệu suất của các kiến trúc nơ-ron nhằm tránh việc phải huấn luyện hoàn toàn kiến trúc đó.



Hình 48: Minh họa phương pháp NAS tổng quát

## Phát biểu bài toán

### Dịnh nghĩa 1: Bài toán Tìm kiếm kiến trúc mạng nơ-ron

- **Đầu vào:** Không gian tìm kiếm  $\mathcal{A}$ , tập dữ liệu  $\mathcal{D}$ , quy trình huấn luyện  $\mathcal{P}$ , và một ngân sách thời gian hoặc tính toán  $t$ .
- **Đầu ra:** Một kiến trúc  $a \in \mathcal{A}$  trong ngân sách  $t$  có độ chính xác trên tập xác thực cao nhất có thể khi được huấn luyện sử dụng tập dữ liệu  $\mathcal{D}$  và quy trình huấn luyện  $\mathcal{P}$ .

Tóm lại, NAS giải quyết gần đúng Bài toán tối ưu hai cấp (Bilevel Optimization Problem - BiO) sau trong điều kiện hạn chế  $t$ :

$$\min_{a \in \mathcal{A}} \mathcal{L}_{\text{validation}}(w^*(a), a)$$

với  $w^*(a) = \arg \min_w \mathcal{L}_{\text{train}}(w, a)$

Với  $\mathcal{L}_{\text{validation}}$  và  $\mathcal{L}_{\text{train}}$  là lỗi xác thực và lỗi huấn luyện. Ngoài định nghĩa cơ bản này, NAS có thể tìm kiếm kiến trúc với các tiêu chí khác: tối thiểu số lượng tham số, độ trễ cùng lúc với tối đa độ chính xác.

# KHÔNG GIAN TÌM KIẾM

Thiết kế không gian tìm kiếm đại diện cho sự cân nhắc quan trọng giữa thiên vị của con người và hiệu quả tìm kiếm:

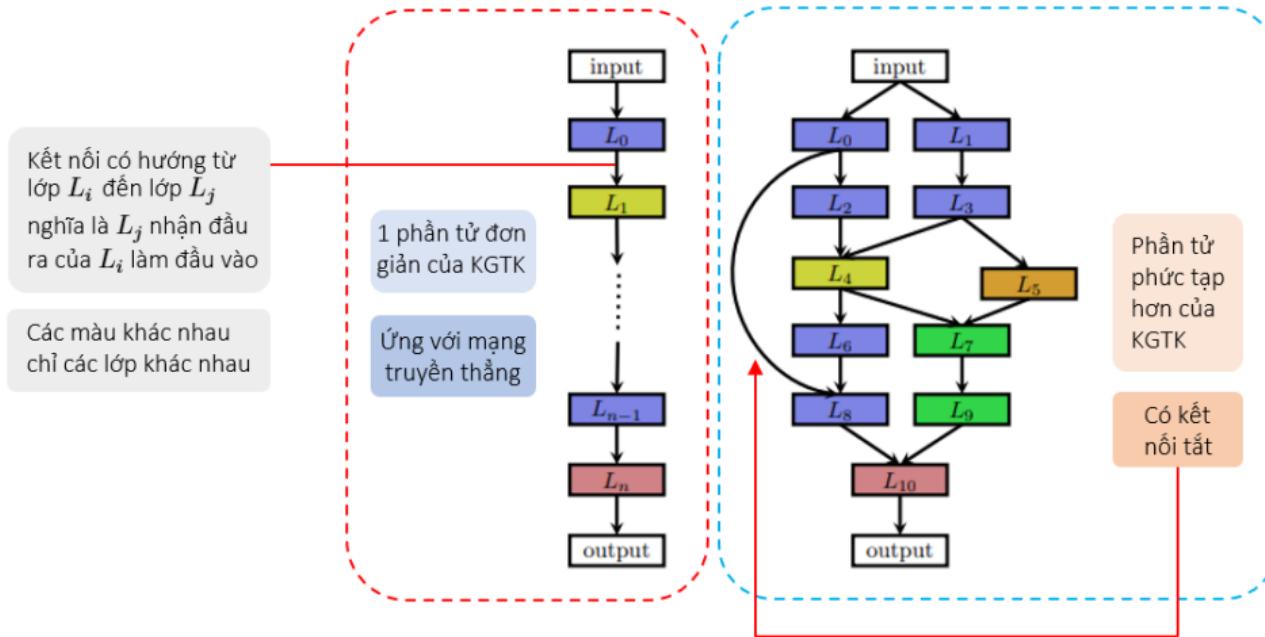
- Nếu kích thước của không gian tìm kiếm nhỏ và bao gồm nhiều quyết định được chọn bằng tay, thì các thuật toán NAS sẽ dễ dàng tìm ra một kiến trúc hiệu suất cao.
- Mặt khác, nếu không gian tìm kiếm lớn với nhiều khối hơn, một thuật toán NAS sẽ cần chạy lâu hơn, nhưng có khả năng phát hiện ra các kiến trúc hoàn toàn mới.

KGTK	Cấu trúc	Siêu tham số tìm kiếm	Cấp tô pô
<b>KGTK vĩ mô</b> VD: NASBOT, EfficientNet	DAG	Toán tử, DAG tô pô, siêu tham số vĩ mô	1
<b>KGTK cấu trúc chuỗi</b> VD: MobileNetV2	Chuỗi	Toán tử, siêu tham số vĩ mô	1
<b>KGTK dựa trên tế bào</b> VD: DARTS	Nhân bản tế bào	Toán tử, tô pô tế bào	1
<b>KGTK phân cấp</b> VD: Hier. Repr., Auto-DeepLab	Đa dạng	Toán tử, DAG/tế bào tô pô, siêu tham số vĩ mô	> 1

Bảng 1: Tóm tắt các kiểu không gian tìm kiếm (KGTK) trong NAS

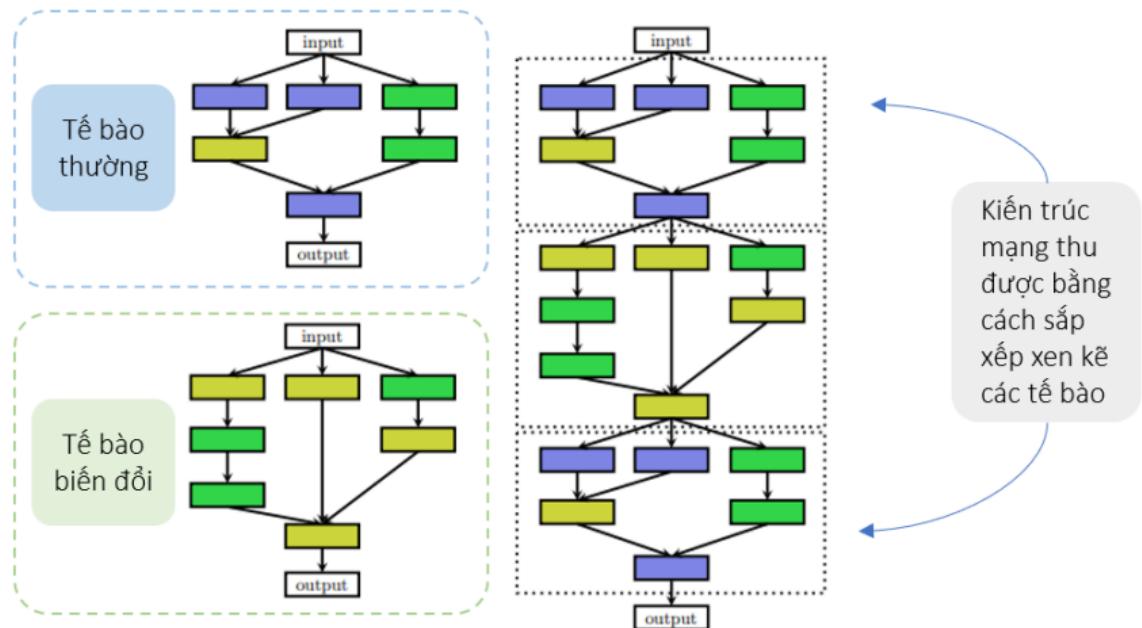
## Thuật ngữ

- **Toán tử** (Operation/Primitive): đơn vị nguyên tử của KGTK. Đôi với hầu như tất cả các KGTK phổ biến, đây là một bộ 3 gồm một kích hoạt cố định, toán tử và chuẩn hóa cố định, như ReLU-conv\_1x1-batchnorm, với ReLU và batchnorm là cố định, và toán tử ở giữa là tùy chỉnh.
- **Lớp** (Layer): thường sử dụng trong KGTK cấu trúc chuỗi hoặc vĩ mô để chỉ cùng một thứ với toán tử. Đôi khi nó cũng chỉ đến các kết hợp toán tử nổi tiếng, như inverted bottleneck residual.



Hình 49: Minh họa phần tử (đơn giản và phức tạp hơn) của không gian tìm kiếm

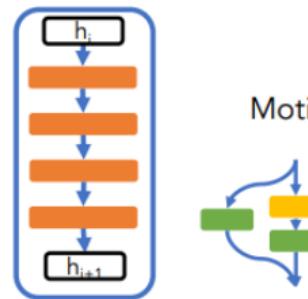
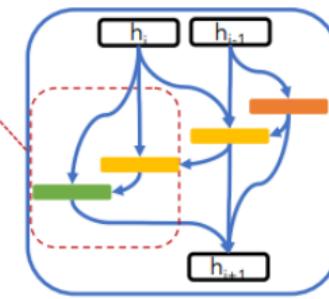
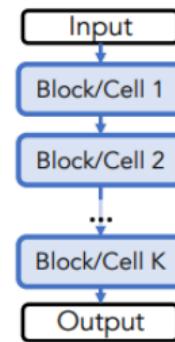
- **Khối** (Block/Module): chỉ một chồng tuần tự của các lớp.
- **Tế bào** (Cell): chỉ một đồ thị có hướng không chu trình của các toán tử trong KGTK dựa trên tế bào. Số lượng toán tử tối đa trong một tế bào thường được cố định.
- **Mô típ** (Motif): một mẫu con được tạo thành từ nhiều toán tử trong một kiến trúc. Một số tài liệu coi một tế bào như một motif cấp cao hơn và một tập hợp nhỏ các toán tử như một motif cơ sở.



**Hình 50:** Minh họa không gian tìm kiếm dựa trên tế bào, gồm tế bào thường (normal cell) và tế bào biến đổi (reduction cell). Ngoài sắp xếp tuần tự các tế bào, còn có thể thiết kế đa nhánh, coi một tế bào tương tự như một lớp

**Operation Layer/Unit/Primitive**

- █ 3x3 depthwise-separable convolution
- █ 1x1 convolution
- █ Inverted bottleneck residual layer

**Block/Module****Cell****Architecture**

Hình 51: Minh họa các thuật ngữ

## Không gian tìm kiếm vĩ mô

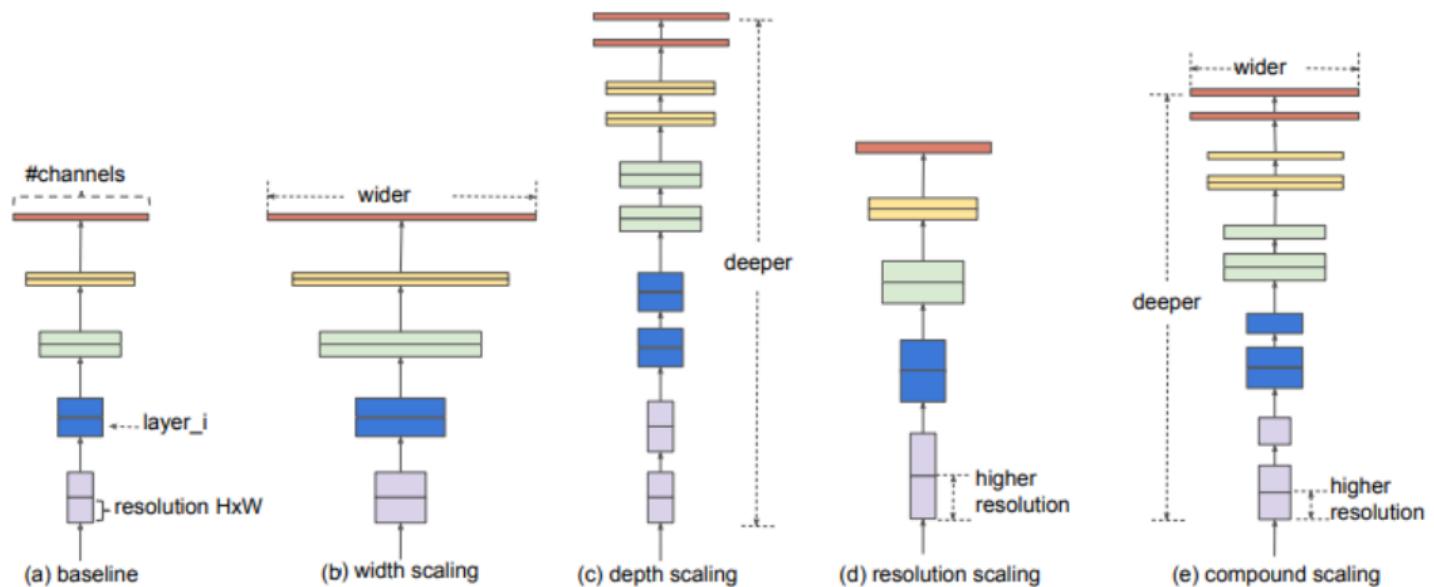
KGTK vĩ mô (Macro Search Spaces) chỉ một trong 2 loại:

- KGTK mã hóa toàn bộ kiến trúc chỉ ở một cấp độ: toàn bộ kiến trúc được biểu diễn như một DAG duy nhất. Các KGTK này thường có sự lựa chọn về toán tử ở mỗi nút trên đồ thị và cấu trúc DAG.
- KGTK chỉ tập trung vào các siêu tham số vĩ mô<sup>12</sup> (như chiều rộng, chiều sâu và độ phân giải): ở đâu và bao nhiêu cần giảm mẫu (downsample) không gian phân giải qua toàn bộ kiến trúc, trong khi giữ nguyên cấu trúc kiến trúc và toán tử.

So với các KGTK khác, KGTK vĩ mô có khả năng biểu diễn cao: cấu trúc linh hoạt của chúng cho phép khả năng khám phá các kiến trúc mới. Tuy nhiên, nhược điểm chính là rất chậm để tìm kiếm.

---

<sup>12</sup>EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks



**Hình 52:** Minh họa kĩ thuật **Mở rộng mô hình** (Model Scaling) gồm: (b) mở rộng chiều rộng (tăng số lượng kênh trong mỗi lớp), (c) mở rộng chiều sâu (tăng số lượng lớp trong mạng), (d) tăng độ phân giải, (e) mở rộng tổ hợp (mở rộng toàn diện, đồng đều trên cả 3 chiều)

## Không gian tìm kiếm cấu trúc chuỗi

KGTK cấu trúc chuỗi, có một tô pô kiến trúc đơn giản: một chuỗi tuần tự các lớp toán tử. Thường lấy các thiết kế thủ công tiên tiến như ResNet<sup>13</sup> hoặc MobileNets<sup>14</sup> làm xương sống.

Có nhiều KGTK cấu trúc chuỗi dựa trên mạng tích chập:

- ProxylessNAS<sup>15</sup> bắt đầu với kiến trúc MobileNetV2<sup>16</sup> và tìm kiếm trên các kích cỡ bộ lọc (kernel sizes) và tỷ lệ mở rộng (expansion ratios) trong các *lớp thăng dư nghịch đảo cỡ chai* (inverted bottleneck residual layers).
- XD<sup>17</sup> và DASH<sup>18</sup> bắt đầu với LeNet, ResNet, hoặc WideResNet,<sup>19</sup> và tìm kiếm trên một *sự tổng quát hóa diễn cảm của các tích chập* (expressive generalization of convolutions) dựa trên ma trận Kaleidoscope, hoặc kích cỡ bộ lọc (kernel sizes) và sự mở rộng (dilations) tương ứng.

---

<sup>13</sup>Deep residual learning for image recognition, 12/2015

<sup>14</sup>Mobilenets: Efficient convolutional neural networks for mobile vision applications, 4/2017

<sup>15</sup>Proxylessnas: Direct neural architecture search on target task and hardware, 12/2018

<sup>16</sup>Mobilenetv2: Inverted residuals and linear bottlenecks, 1/2018

<sup>17</sup>Rethinking neural operations for diverse tasks, 3/2021

<sup>18</sup>Efficient architecture search for diverse tasks, 4/2022

<sup>19</sup>Wide residual networks, 5/2016

KGTK cấu trúc chuỗi cũng phổ biến trong KGTK dựa trên mô hình transformer:

- KGTK từ Lightweight Transformer Search (LTS)<sup>20</sup> bao gồm một cấu hình cấu trúc chuỗi của dòng kiến trúc GPT phổ biến cho mô hình ngôn ngữ tự hồi quy, với các lựa chọn tìm kiếm cho số lượng lớp, kích thước mô hình, kích thước nhúng linh hoạt, kích thước của mạng nơ-ron truyền thẳng trong một lớp transformer, và số lượng đầu trong mỗi lớp transformer.
- KGTK từ NAS-BERT<sup>21</sup> và MAGIC<sup>22</sup> đều bao gồm một KGTK cấu trúc chuỗi trên kiến trúc BERT<sup>23</sup> với tới 26 lựa chọn toán tử bao gồm các biến thể của sự chú ý đa đầu (multi-head attention), các lớp lan truyền và các tích chập với kích cỡ bộ lọc khác nhau.

---

<sup>20</sup>Litetransformersearch: Training-free on-device search for efficient autoregressive language models, 2/2022

<sup>21</sup>NAS-BERT: Task-Agnostic and Adaptive-Size BERT Compression with Neural Architecture Search, 5/2021

<sup>22</sup>Analyzing and Mitigating Interference in Neural Architecture Search, 8/2021

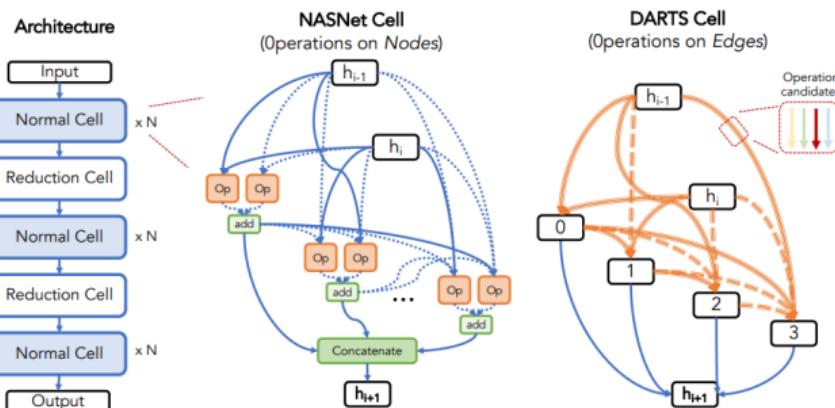
<sup>23</sup>BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 10/2018

KGTK cấu trúc chuỗi về mặt khái niệm là đơn giản, làm cho chúng dễ thiết kế và thực hiện.

- Thường chứa các kiến trúc mạnh mẽ có thể được tìm thấy tương đối nhanh chóng.
- Nhược điểm chính là do tốn pô kiến trúc đơn giản, có cơ hội tương đối thấp để khám phá một kiến trúc thực sự mới.

# Không gian tìm kiếm dựa trên tế bào

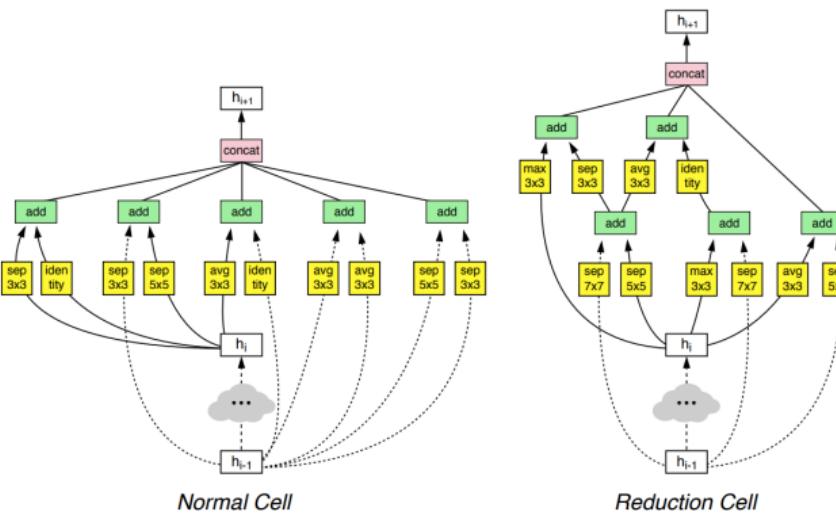
KGTK dựa trên tế bào là loại KGTK phổ biến nhất trong NAS. Thay vì tìm kiếm toàn bộ kiến trúc mạng từ đầu, Zoph và đồng nghiệp (2018) đề xuất chỉ tìm kiếm trên các tế bào nhỏ, và xếp chúng nhiều lần theo thứ tự để tạo thành kiến trúc tổng thể. Các tế bào có thể tìm kiếm tạo nên *cấu trúc vi mô* của KGTK, trong khi khung ngoài (*cấu trúc vĩ mô*) được cố định.



**Hình 53:** Minh họa của KGTK dựa trên tế bào. Khung ngoài chung cho các tế bào (bên trái) được cố định, trong khi các tế bào có thể tìm kiếm được. NASNet gán các toán tử cho các nút (ở giữa) trong khi DARTS gán các toán tử cho các cạnh (bên phải)

KGTK dựa trên tế bào hiện đại đầu tiên, NASNet gồm 2 loại tế bào: tế bào thường (normal cell) và tế bào biến đổi (reduction cell). Cả 2 có cùng cấu trúc, nhưng các toán tử ban đầu trong tế bào biến đổi có bước nhảy 2 để giảm một nửa độ phân giải không gian đầu vào.

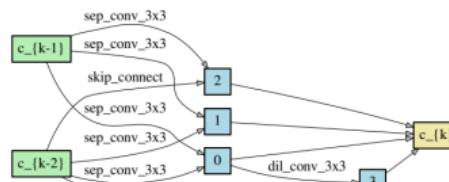
- Mỗi tế bào NASNet có thể được biểu diễn như một DAG.
- Các nút được sắp xếp thành bộ 3 gồm: 2 nút toán tử và 1 nút kết hợp.



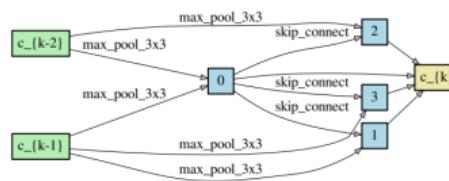
**Hình 54:** Đầu vào (màu trắng) là trạng thái ẩn từ các kích hoạt trước đó (hoặc hình ảnh đầu vào). Đầu ra (màu hồng) là kết quả của một toán tử nối qua tất cả các nhánh kết quả.

KGTK của DARTS khác biệt hơn, trong khi nó cũng có 2 tế bào có thể tìm kiếm thì:

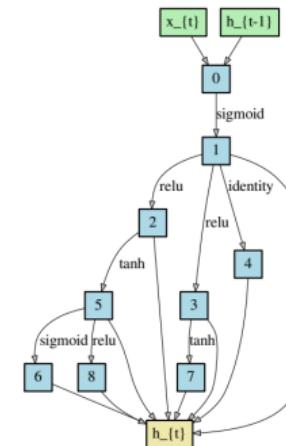
- Các tế bào DARTS có các lựa chọn toán tử trên các cạnh của đồ thị chứ không phải trên các nút.
- Các nút đại diện cho các biểu diễn tiềm ẩn và các cạnh là toán tử.



Tế bào thường của mạng tích chập



Tế bào biến đổi của mạng tích chập



Tế bào hồi quy của mạng hồi quy

Hình 55: Minh họa tế bào tìm kiếm được bằng DARTS

Ngoài phân loại hình ảnh, các thiết kế tế bào tương tự cũng đã được áp dụng cho các mô hình ngôn ngữ. Ngoài ra, KGTK dựa trên tế bào phổ biến gần đây bởi:

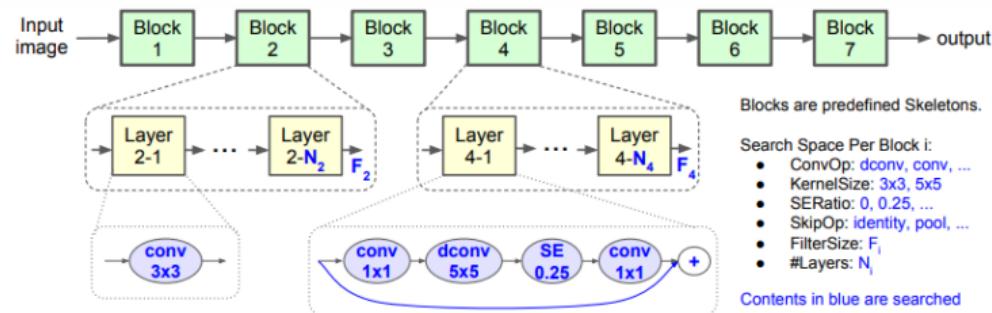
- Thiết kế dựa trên tế bào đã giảm đáng kể độ phức tạp của KGTK, trong khi thường tạo ra một kiến trúc cuối cùng hiệu suất cao.
- Bằng cách tách rời độ sâu của kiến trúc khỏi quá trình tìm kiếm, cấu trúc dựa trên tế bào có thể chuyển giao: các tế bào tối ưu học được trên một bộ dữ liệu nhỏ (ví dụ, CIFAR-10) thường chuyển giao tốt sang một bộ dữ liệu lớn (ví dụ, ImageNet) bằng cách tăng số lượng tế bào và bộ lọc trong kiến trúc tổng thể.

Việc giới hạn tìm kiếm vào một tế bào làm giảm độ phức tạp của việc tìm kiếm, nhưng thực hành này làm giảm sự đa dạng của KGTK NAS, khiến việc tìm kiếm các kiến trúc mới lạ cao cấp trở nên khó khăn.

# Không gian tìm kiếm phân cấp

Các KGTK trước đều có biểu diễn phẳng. KGTK phân cấp (Hierarchical Search Spaces) liên quan đến việc thiết kế các motif ở các cấp độ khác nhau, nơi mỗi motif cấp độ cao hơn thường được biểu diễn dưới dạng DAG của các motif cấp độ thấp hơn.

- Một loại đơn giản của KGTK phân cấp có 2 cấp độ tìm kiếm bằng cách thêm các siêu tham số kiến trúc vĩ mô vào KGTK dựa trên tế bào hoặc cấu trúc chuỗi. Chẳng hạn, KGTK MnasNet<sup>24</sup> có mô tả như dưới đây:

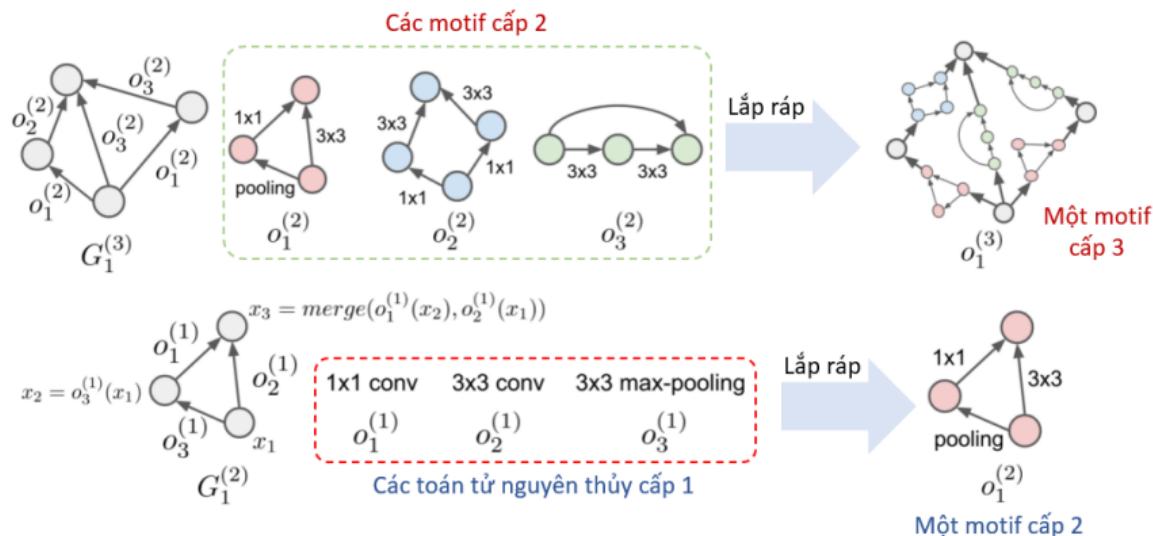


**Hình 56:** Chia nhỏ một mô hình CNN thành các khối đơn nhất và sau đó tìm kiếm các toán tử và kết nối cho từng khối một, cho phép kiến trúc lớp khác nhau trong các khối khác nhau

<sup>24</sup>Mnasnet: Platform-aware neural architecture search for mobile, 5/2019

- Vượt qua 2 cấp độ, hệ thống phân cấp 3 cấp độ được đề xuất.

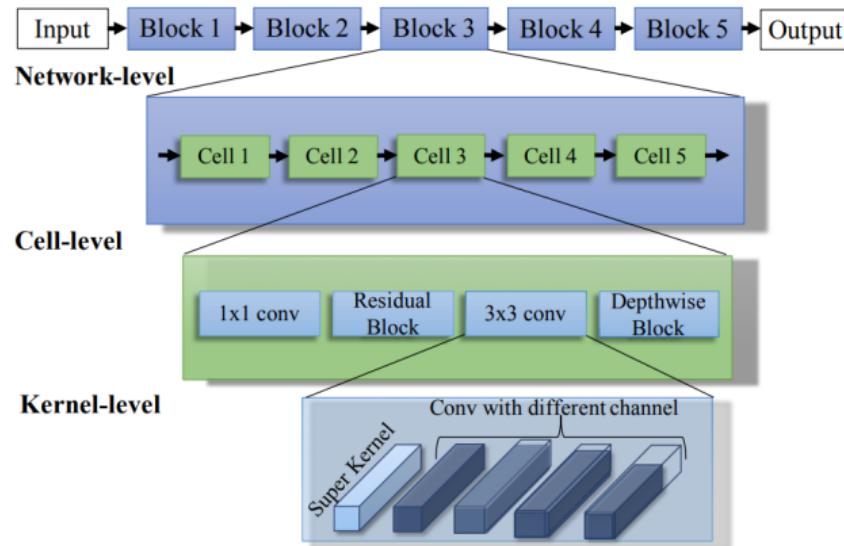
1. Liu và đồng nghiệp đề xuất mỗi cấp độ là một đồ thị được tạo nên từ các thành phần của cấp độ trước.<sup>25</sup>



Hình 57: Minh họa KGTK 3 cấp độ, dựa trên đề xuất của Liu.

<sup>25</sup>Hierarchical Representations for Efficient Architecture Search, 2/2018

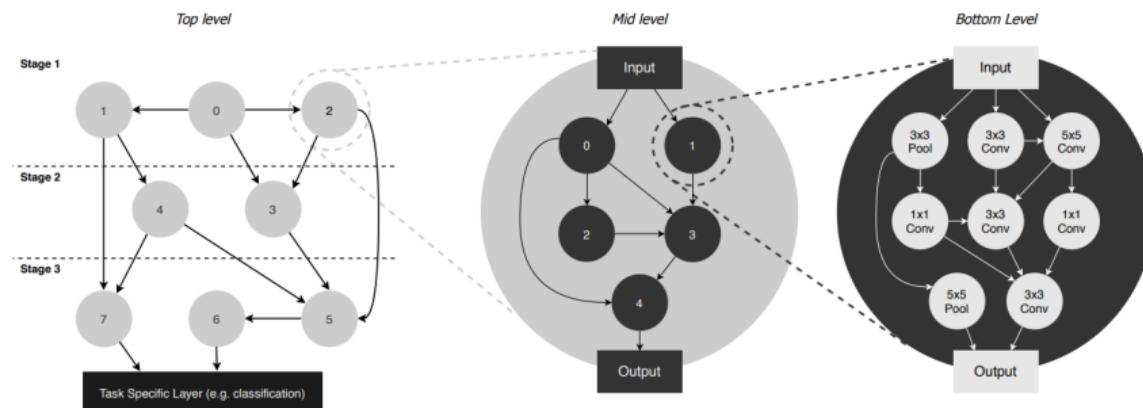
2. Wu và đồng nghiệp đề xuất một hệ thống phân cấp 3 cấp độ khác nhau, bao gồm các siêu tham số bộ lọc, siêu tham số dựa trên tế bào, và siêu tham số vĩ mô.<sup>26</sup>



**Hình 58:** (1) KGTK cấp mạng bao gồm tất cả các đường dẫn mạng ứng cử. (2) KGTK cấp tế bào chứa một sự kết hợp của tất cả các toán tử ứng cử có thể. (3) KGTK cấp bộ lọc tìm kiếm toán tử và kích thước bộ lọc tốt nhất.

<sup>26</sup>Trilevel Neural Architecture Search for Efficient Single Image Super-Resolution, 4/2021

3. Phần thiết kế trước đó đã được mở rộng ra ngoài 2 cấp độ trong một vài công trình sau: Ru và đồng nghiệp<sup>27</sup> đã đề xuất một thiết kế phân cấp 4 cấp độ, được điều khiển bởi một tập siêu tham số ứng với một trình tạo đồ thị ngẫu nhiên, Chrostoforidis và đồng nghiệp<sup>28</sup> đã giới thiệu quá trình xây dựng đệ quy để cho phép số lượng cấp độ phân cấp biến đổi cũng như một định dạng linh hoạt trong các motif cấp trên.



Hình 59: Công trình của Ru về một trình tạo mạng ngẫu nhiên có thể đưa ra một loạt rộng lớn các mạng chưa từng thấy trước đó về mức độ phức tạp của việc kết nối

<sup>27</sup> Neural Architecture Generator Optimization, 4/2020

<sup>28</sup> A Novel Evolutionary Algorithm for Hierarchical Neural Architecture Search, 5/2023

Có nhiều lợi ích khi sử dụng KGTK phân cấp:

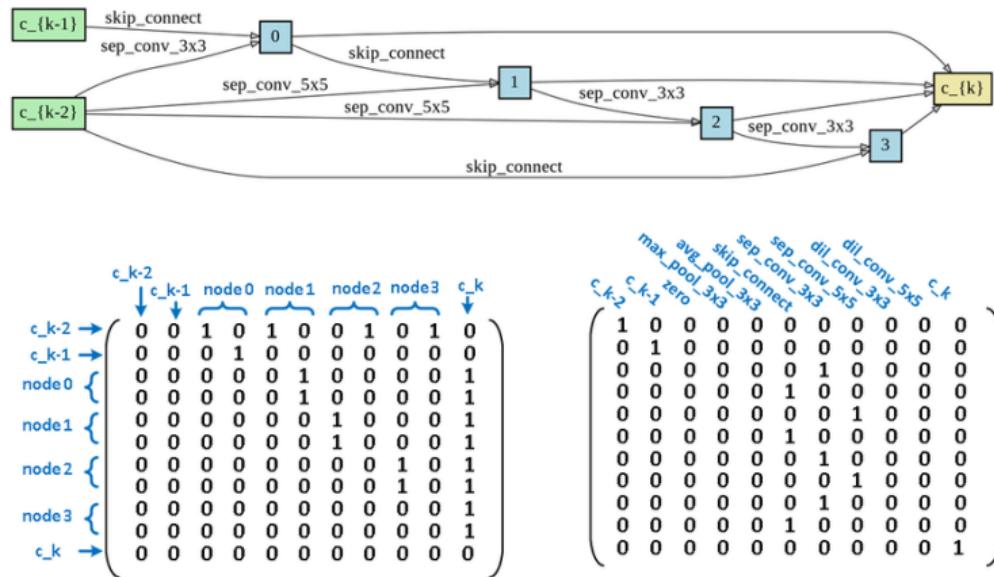
- KGTK phân cấp thường có tính biểu đạt cao hơn.
- Hầu hết các KGTK theo dạng chuỗi, dựa trên tê bào hoặc cấu trúc vĩ mô có thể được xem như một KGTK phân cấp với một cấp độ tìm kiếm duy nhất, nhưng có 2 hoặc nhiều mức cho phép tìm kiếm qua nhiều thiết kế kiến trúc phức tạp và đa dạng hơn.
- Việc biểu diễn phân cấp của một kiến trúc lớn là một cách hiệu quả để giảm độ phức tạp của quá trình tìm kiếm, có thể dẫn đến hiệu suất tìm kiếm tốt hơn

## Mã hóa kiến trúc

Các chương trình NAS cần có một biểu diễn ngắn gọn cho mỗi kiến trúc, hoặc mã hóa, để thực hiện các hoạt động như: biến đổi kiến trúc, định lượng sự tương tự giữa 2 kiến trúc hoặc dự đoán hiệu suất thử nghiệm.

- Trong hầu hết các KGTK, kiến trúc có thể được biểu diễn một cách gọn gàng dưới dạng DAG, trong đó mỗi nút hoặc cạnh đại diện cho một toán tử. Có thể áp dụng cho KGTK cấu trúc chuỗi và dựa trên tế bào.
- KGTK phân cấp không thể được biểu diễn hoàn toàn bằng một DAG và cần một mã hóa có cấu trúc với điều kiện, trong đó số lượng cấp độ siêu tham số điều kiện tương ứng với số cấp độ của phân cấp.

- Với KGTK dựa trên tê bào, mã hóa phổ biến nhất là ma trận liên thuộc cùng với danh sách các toán tử, của các tê bào có thể tìm kiếm. VD: NAS-Bench-101,<sup>29</sup> EcoNAS.<sup>30</sup>

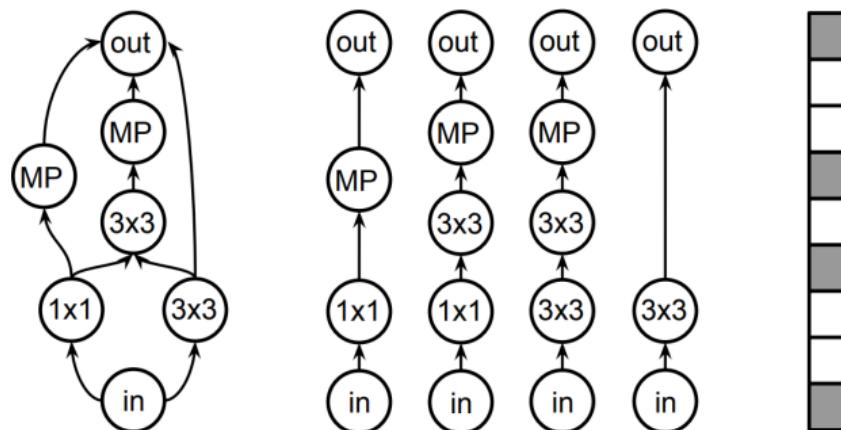


Hình 60: Minh họa mã hóa tê bào trong không gian tìm kiếm DARTS

<sup>29</sup>NAS-Bench-101: Towards Reproducible Neural Architecture Search, 5/2019

<sup>30</sup>EcoNAS: Finding Proxies for Economical Neural Architecture Search, 2/2020

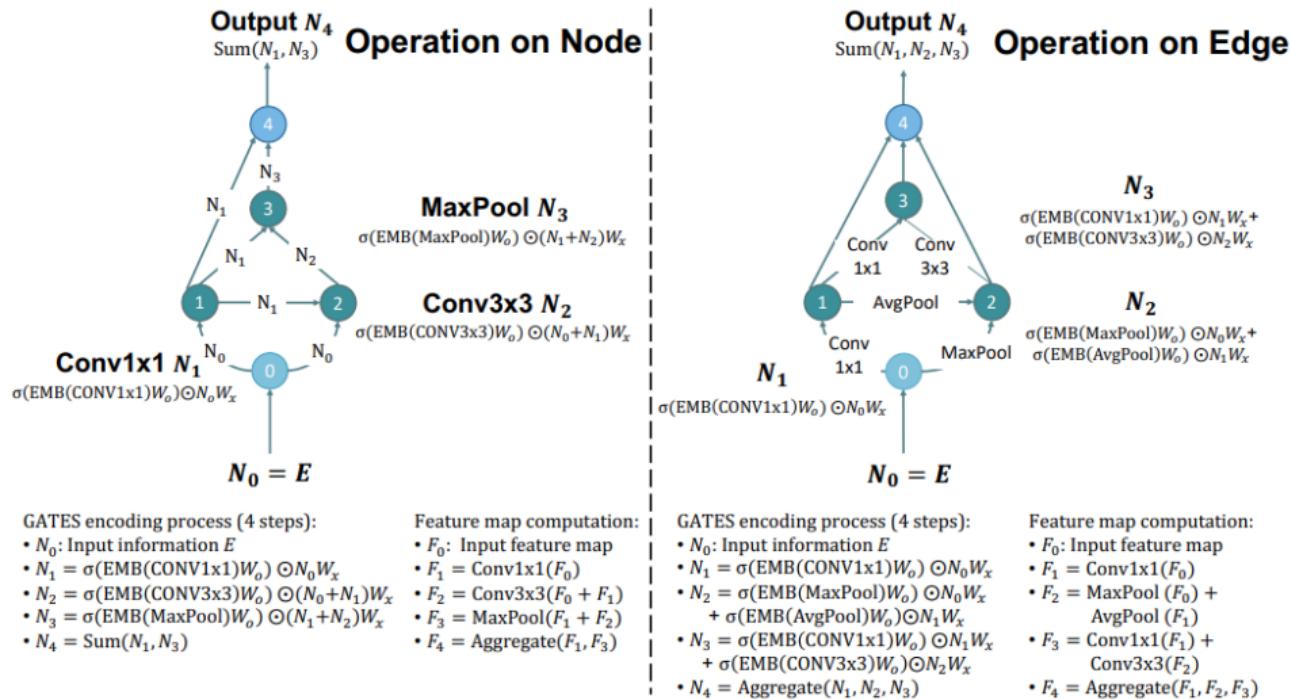
- Để có tính tổng quát tốt hơn, Ning và đồng nghiệp đã đề xuất một hệ thống mã hóa dựa trên đồ thị<sup>31</sup>; White và đồng nghiệp đã đề xuất một hệ thống mã hóa dựa trên đường dẫn,<sup>32</sup> cả 2 đều mô hình hóa luồng thông tin truyền tải trong mạng.



**Hình 61:** Minh họa **Mã hóa đường dẫn** (path encoding): mỗi đường dẫn được mã hóa thành một đặc trưng nhị phân, thể hiện sự có mặt của đường dẫn đó trong kiến trúc.

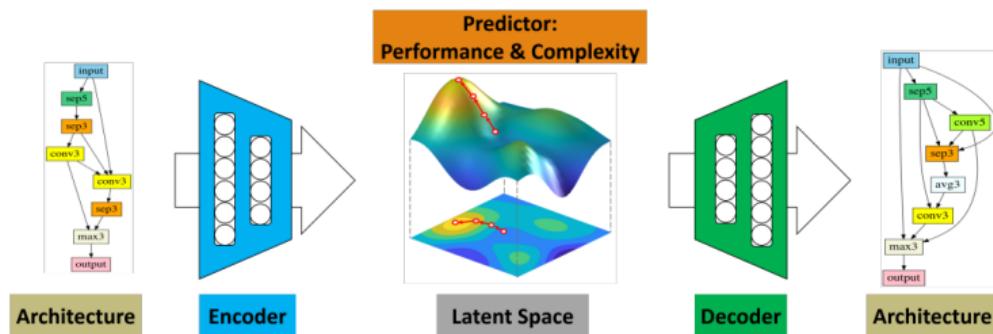
<sup>31</sup>A Generic Graph-based Neural Architecture Encoding Scheme for Predictor-based NAS, 9/2020

<sup>32</sup>BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search, 11/2020



**Hình 62:** Minh họa **GATES** (Graph-based neural ArchiTecture Encoding Scheme). Quá trình tính toán bản đồ đặc trưng ( $F_i$ ) và mã hóa ( $N_i$ ). Trong GATES, các toán tử trong kiến trúc mạng được mô hình hóa như là các biến đổi của thông tin, không chỉ đơn giản là thuộc tính của nút.

- Một loại mã hóa khác cho tất cả các KGTK là mã hóa *được học* bằng cách tiền đào tạo không giám sát. Trước khi chúng ta chạy NAS, chúng ta sử dụng một tập hợp các kiến trúc chưa được đào tạo để học mã hóa kiến trúc, ví dụ, bằng cách sử dụng Bộ tự động mã hóa<sup>33,34</sup> (Autoencoder) hoặc Bộ biến hình<sup>35</sup> (Transformer).

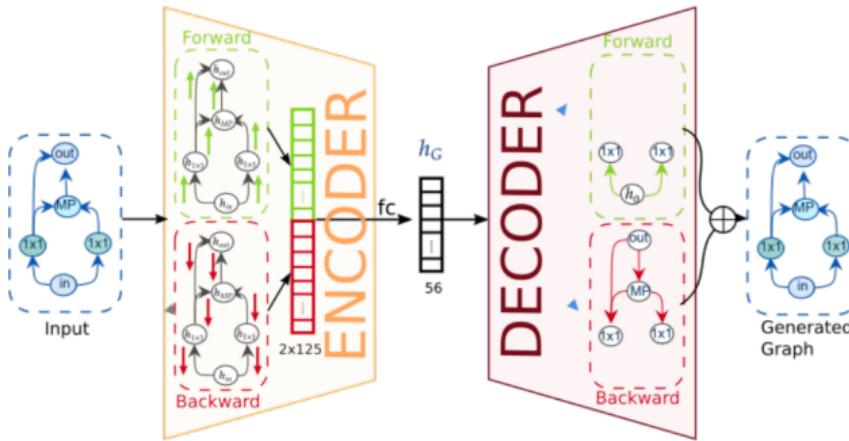


**Hình 63:** Khung làm việc của **NGAE** (Graph variational AutoEncoder): Kiến trúc mạng được ánh xạ thành một biểu diễn liên tục thông qua bộ mã hóa. Sử dụng thuật toán tối ưu hóa gradient ngẫu nhiên, biểu diễn liên tục được tối ưu hóa thông qua việc tối đa hóa bộ dự đoán. Biểu diễn liên tục tối ưu hóa sau đó được giải mã thành một kiến trúc mạng bằng cách sử dụng bộ giải mã.

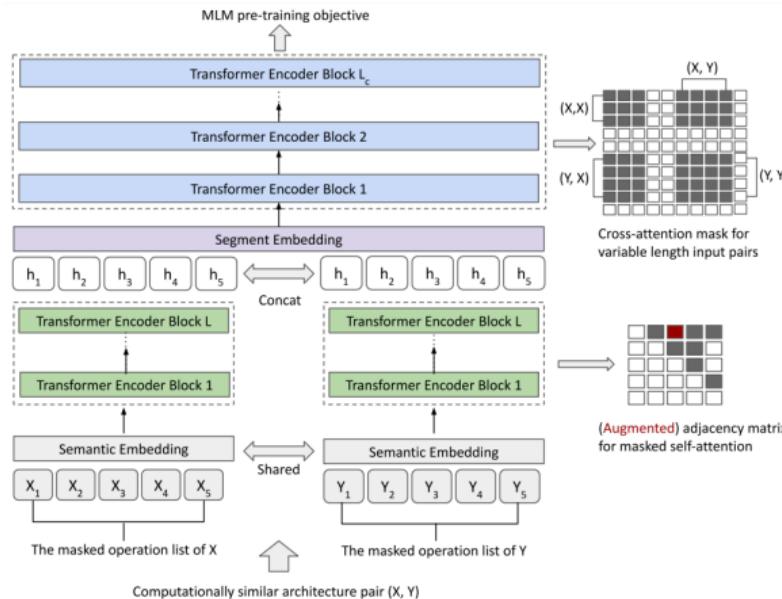
<sup>33</sup>Neural Architecture Optimization with Graph VAE, 6/2020

<sup>34</sup>Smooth Variational Graph Embeddings for Efficient Neural Architecture Search, 5/2021

<sup>35</sup>CATE: Computation-aware Neural Architecture Encoding with Transformers, 6/2021



**Hình 64:** Kiến trúc của mô hình **SVGe** (Smooth Variational Graph embeddings) để xuất: Nhận dữ liệu đầu vào là một đồ thị kiến trúc mạng. Bộ mã hóa (bên trái) sử dụng 2 mô-đun GNN (Graph Neural Networks), bộ mã hóa xuôi (màu xanh) và bộ mã hóa ngược (màu đỏ), để tạo ra một biểu diễn tiềm tàng có thông tin. Vector tiềm tàng này là đầu vào cho bộ giải mã (bên phải), bộ giải mã giải mã các bước tiến (màu xanh) và bước lùi (màu đỏ) một cách riêng biệt, tạo ra 2 đồ thị theo cách tuần tự. Sự kết hợp của chúng là đầu ra của SVGe.



**Hình 65:** Tổng quan về **CATE** (ComputationAware Transformer-based Encoding), một phương pháp mã hóa kiến trúc mạng nơ-ron dựa trên khả năng tính toán. Nó hoạt động bằng cách lựa chọn các cặp kiến trúc có tính toán tương tự nhau và sử dụng kiến trúc Transformer để mã hóa chúng. Trong quá trình huấn luyện, một số phần của mỗi kiến trúc được che giấu (masked), và mô hình được huấn luyện để dự đoán những phần này.

# CHIẾN LƯỢC TÌM KIẾM

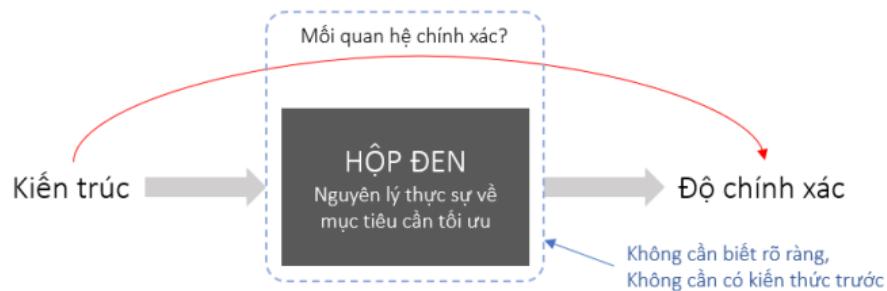
Chiến lược tìm kiếm rơi vào 2 loại: Kỹ thuật tối ưu hóa hộp đen (Black-box Optimization Techniques) và Kỹ thuật một lần (One-shot Techniques) hoặc không phải loại nào.

- **Kỹ thuật tối ưu hóa hộp đen:** coi quá trình tìm kiếm kiến trúc mạng nơ-ron như một “hộp đen”, giả định không có bất kỳ kiến thức trước đó về cấu trúc nội bộ.
- **Kỹ thuật một lần:** tập trung vào việc đào tạo một mạng lớn duy nhất có khả năng biểu diễn nhiều kiến trúc khác nhau.

## Kỹ thuật tối ưu hóa hộp đen

Kỹ thuật tối ưu hóa hộp đen được sử dụng rộng rãi và nghiên cứu ngày nay, do hiệu suất mạnh mẽ và dễ sử dụng của chúng.

- Kỹ thuật tối ưu hóa hộp đen thường sử dụng nhiều tài nguyên tính toán hơn so với kỹ thuật một lần, do phải đào tạo nhiều kiến trúc một cách độc lập.
- Chúng cũng có nhiều ưu điểm hơn so với kỹ thuật một lần, như sự kiên định (duy trì hoạt động ổn định và đáng tin cậy dưới nhiều điều kiện khác nhau), tối ưu hóa đơn giản hơn cho các mục tiêu không khả vi, tính song song hóa đơn giản hơn, tối ưu hóa chung với các tham số siêu học khác, và dễ thích nghi hơn với không gian tìm kiếm mới, dữ liệu mới.



Hình 66: Minh họa Tối ưu hóa hộp đen

## Tìm kiếm ngẫu nhiên

Trong phương pháp **Tìm kiếm ngẫu nhiên** (Random Search):

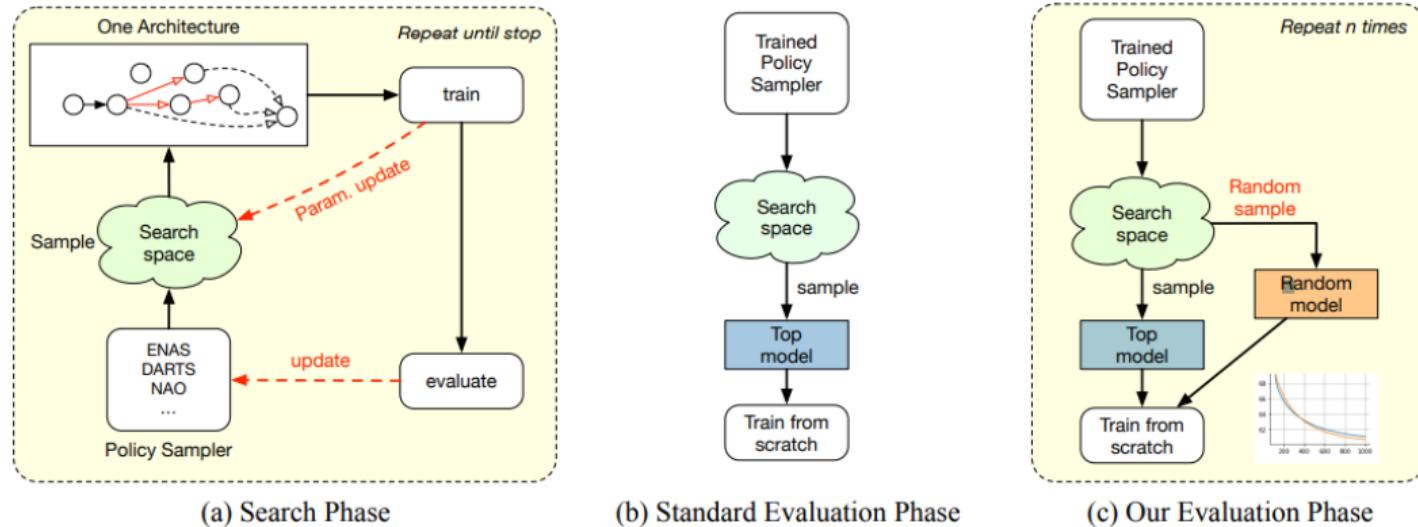
- Các kiến trúc được lựa chọn ngẫu nhiên từ KGTK và sau đó được đào tạo hoàn toàn.
- Cuối cùng, kiến trúc có độ chính xác xác thực tốt nhất được xuất ra.
- Mặc dù ngày thơ, nhiều bài báo đã chỉ ra rằng tìm kiếm ngẫu nhiên hoạt động đáng ngạc nhiên tốt.<sup>36,37,38</sup>

---

<sup>36</sup>Random Search and Reproducibility for Neural Architecture Search, 7/2019

<sup>37</sup>Evaluating the search phase of neural architecture search, 11/2019

<sup>38</sup>NAS evaluation is frustratingly hard, 2/2020



Hình 67: Thực hiện so sánh giai đoạn tìm kiếm của các thuật toán NAS hiện có với một chính sách tìm kiếm ngẫu nhiên. Một thuật toán tìm kiếm hiệu quả nên tạo ra một giải pháp rõ ràng vượt trội so với chính sách ngẫu nhiên.

- Tìm kiếm ngẫu nhiên sẽ tốt với những KGTK được thiết kế cao cấp với tỷ lệ cao của các kiến trúc mạnh.
- Tuy nhiên, tìm kiếm ngẫu nhiên không hoạt động tốt trên không gian tìm kiếm lớn và đa dạng.
- Tìm kiếm ngẫu nhiên được khuyến nghị mạnh mẽ làm cơ sở so sánh cho các thuật toán NAS mới, và có thể trở nên cạnh tranh cao bằng cách tích hợp chia sẻ trọng số (weight sharing), proxy không tốn kém (zero-cost proxies), hoặc ngoại suy đường cong học tập (learning curve extrapolation).

## Tìm kiếm địa phương

- Các bài báo gần đây đã chỉ ra rằng **Tìm kiếm địa phương** (Local Search) là một cơ sở mạnh mẽ cho NAS trên cả KGTK nhỏ<sup>39,40</sup> và KGTK lớn.<sup>41</sup>
- Sử dụng khái niệm *phạm vi láng giềng* của kiến trúc là tất cả các kiến trúc khác biệt một toán tử hoặc một cạnh. Hình thức đơn giản nhất cũng mang lại hiệu quả tốt: lặp lại đào tạo và đánh giá tất cả các kiến trúc láng giềng tốt nhất được tìm thấy.
- Tìm kiếm địa phương có thể được tăng tốc độ đáng kể bằng cách sử dụng cấu xạ mạng (network morphism) để khởi động ấm (warm-start) các kiến trúc láng giềng.<sup>42</sup>

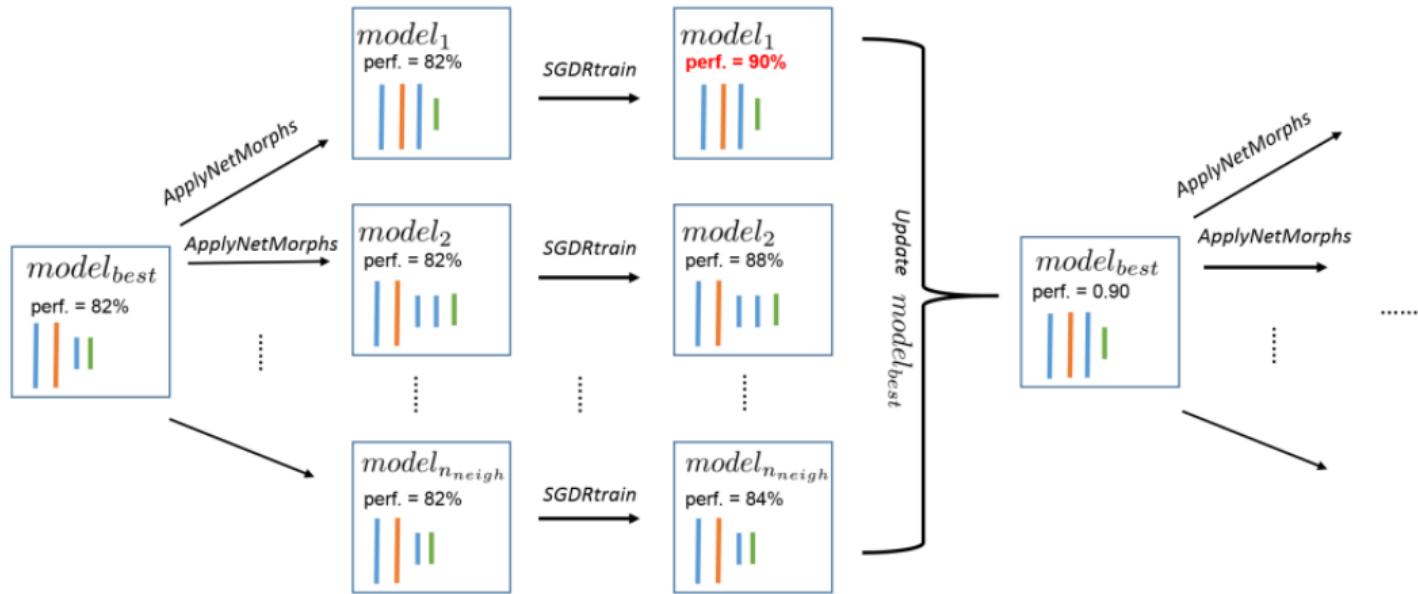
---

<sup>39</sup>Local Search is a Remarkably Strong Baseline for Neural Architecture Search, 7/2020

<sup>40</sup>Exploring the Loss Landscape in Neural Architecture Search, 6/2021

<sup>41</sup>NAS-Bench-301 and the Case for Surrogate Benchmarks for Neural Architecture Search, 9/2020

<sup>42</sup>Simple And Efficient Architecture Search for Convolutional Neural Networks, 11/2017



Hình 68: Bắt đầu bằng một mạng nhỏ, có thể đã được đào tạo trước. Sau đó áp dụng câu xạ mạng vào mạng ban đầu này để tạo ra các mạng lớn hơn có thể hoạt động tốt hơn khi được đào tạo tiếp. Những mạng “con” mới này có thể được xem như là hàng xóm của mạng “cha” ban đầu trong không gian kiến trúc mạng.

## Học tăng cường

**Học tăng cường** (Reinforcement Learning - RL) nổi bật trong giai đoạn đầu NAS hiện đại.

- Hầu hết các phương pháp RL mô hình kiến trúc dưới dạng một chuỗi các hành động được tạo ra bởi một bộ điều khiển (controller).<sup>43,44</sup>
- Độ chính xác kiểm tra của các kiến trúc được lấy mẫu sau quá trình đào tạo được sử dụng làm phần thưởng để cập nhật bộ điều khiển nhằm tối đa hóa giá trị kỳ vọng của nó.
- Bộ điều khiển thường là một mạng nơ-ron hồi quy tạo ra một chuỗi thành phần tương ứng với một kiến trúc.
- Sau khi mỗi kiến trúc được tạo ra được đào tạo và đánh giá, các tham số của RNN được cập nhật để tối đa hóa độ chính xác kiểm tra kỳ vọng của các kiến trúc được tạo ra, bằng cách sử dụng phương pháp REINFORCE, chính sách tối ưu gần (proximal policy optimization).

Gần đây, RL đã không được sử dụng phổ biến trong NAS, vì nó đã được chứng minh thua kém trong các so sánh trực tiếp bởi các phương pháp tiến hóa, và tối ưu hóa Bayes.

<sup>43</sup>Designing neural network architectures using reinforcement learning, 3/2017

<sup>44</sup>Neural architecture search with reinforcement learning, 2/2017

## Thuật toán 1: Thuật toán RL - NAS tổng quát

**Đầu vào:** Không gian kiến trúc  $\mathcal{A}$ , số vòng lặp  $T$ .

1. Khởi tạo ngẫu nhiên tham số  $\theta$  cho bộ điều khiển kiến trúc.
2. Vòng lặp  $t = 1, 2, \dots, T$ :
  - Huấn luyện kiến trúc  $a \sim \pi(a; \theta)$ , lấy mẫu ngẫu nhiên từ chính sách  $\pi(a; \theta)$ .
  - Cập nhật  $\theta$  bằng  $\nabla_{\theta} \mathbb{E}_{a \sim \pi(a; \theta)} [\mathcal{L}_{\text{validation}}(a)]$ .

**Đầu ra:** Kiến trúc chọn từ chính sách đã huấn luyện  $\pi(a; \theta^*)$ .

## Các giải thuật tiến hóa và di truyền

- Nhiều thập kỷ trước sự phục hồi gần đây của NAS, một trong những công trình đầu tiên trong NAS đã sử dụng thuật toán tiến hóa.<sup>45</sup>
- Ngày nay, thuật toán tiến hóa vẫn được ưa chuộng để tối ưu hóa kiến trúc do tính linh hoạt, đơn giản về khái niệm và kết quả cạnh tranh.
- Một thuật toán tiến hóa đặc biệt thành công là **Tiến hóa hiệu chỉnh** (regularized evolution) của Real và đồng nghiệp.<sup>46</sup> Phương pháp này đã vượt qua tìm kiếm ngẫu nhiên và RL trong cuộc so sánh trực tiếp và đạt được hiệu suất hàng đầu trên ImageNet vào thời điểm công bố.

---

<sup>45</sup>Designing neural networks using genetic algorithms, 1989

<sup>46</sup>Regularized evolution for image classifier architecture search, 2/2019

## Thuật toán 2: Thuật toán Tiền hóa - NAS tổng quát

**Đầu vào:** Không gian kiến trúc  $\mathcal{A}$ , số vòng lặp  $T$ .

1. Lấy mẫu ngẫu nhiên và huấn luyện quần thể các kiến trúc từ không gian  $\mathcal{A}$ .
2. Vòng lặp  $t = 1, 2, \dots, T$ :
  - Lấy mẫu (dựa trên độ chính xác) một tập kiến trúc cha mẹ từ quần thể.
  - Đột biến kiến trúc cha mẹ để tạo ra kiến trúc con và huấn luyện chúng.
  - Thêm các kiến trúc con vào quần thể và loại bỏ những kiến trúc già nhất (hoặc kém nhất).

**Đầu ra:** Kiến trúc có độ chính xác trên tập xác thực cao nhất.

## Tối ưu Bayes

**Tối ưu hóa Bayes** (Bayesian Optimization - BO) là một phương pháp mạnh mẽ để tối ưu hóa các hàm đắt đỏ, và nó đã đạt được thành công đáng kể trong NAS.

Có 2 thành phần chính trong BO:

1. Xây dựng một ước tính xác suất (probabilistic surrogate) để mô hình hóa mục tiêu không xác định dựa trên các quan sát trước đây.
2. Xác định một hàm thu nhận (acquisition function) để cân bằng sự khám phá (exploration) và khai thác (exploitation) trong quá trình tìm kiếm.

Để sử dụng ước tính xác suất theo **Quá trình Gauss** (Gaussian Process - GP), NAS dựa trên BO đã phát triển các phương pháp tính “khoảng cách” (distance metric) giữa các kiến trúc:

- Hạt nhân kiến trúc đặc biệt.<sup>47</sup>
- Hàm khoảng cách dựa trên tối ưu vận tải.<sup>48</sup>
- Hàm khoảng cách dựa trên cây-Wasserstein.<sup>49</sup>

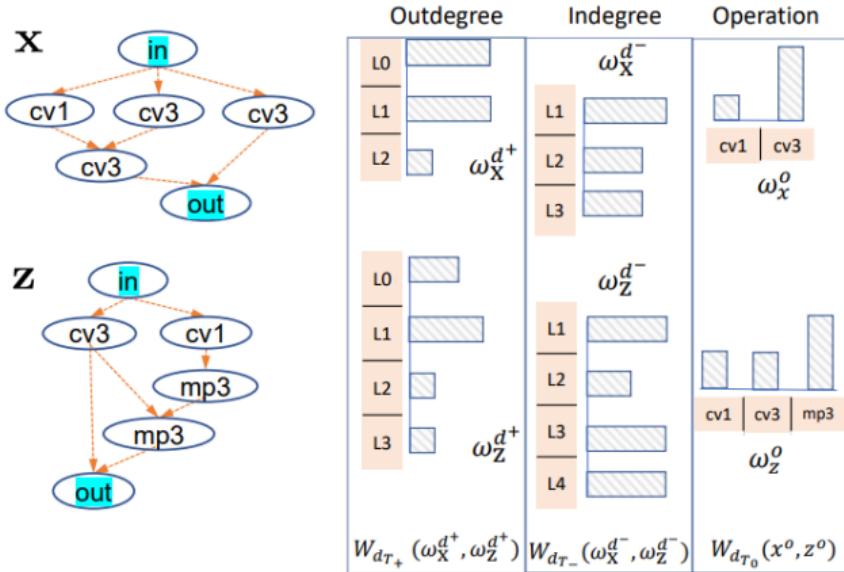
Tuy nhiên, việc sử dụng ước tính xác suất GP tiêu chuẩn thường không hiệu quả cho NAS, vì không gian tìm kiếm thường có số chiều cao, không liên tục và có cấu trúc dạng đồ thị.

---

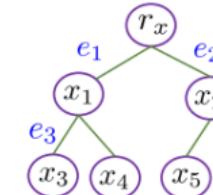
<sup>47</sup> Raiders of the Lost Architecture: Kernels for Bayesian Optimization in Conditional Parameter Spaces, 9/2014

<sup>48</sup> Neural Architecture Search with Bayesian Optimisation and Optimal Transport, 3/2019

<sup>49</sup> Optimal Transport Kernels for Sequential and Parallel Neural Architecture Search, 6/2021



$$d_{NN}(x, z) = \alpha_1 W_{d_{T_0}}(x^o, z^o) + \alpha_2 W_{d_{T_+}}(\omega_X^{d+}, \omega_Z^{d+}) + (1 - \alpha_1 - \alpha_2) W_{d_{T_-}}(\omega_X^{d-}, \omega_Z^{d-})$$

A tree for computing  $W$ 

$W(x, z)$	transport $x$ to $z$
frequency	frequency
$cv1$	$operation$
$L1$	$bin over layer 1$
$cv1/3$	$convolution1x1/3x3$
$mp3$	$maxpooling3x3$

**Hình 69:** Biểu diễn 2 kiến trúc  $x$  và  $z$  bằng cấu trúc mạng (qua số lượng đỉnh ra và số lượng đỉnh vào) và toán tử mạng. Độ tương tự giữa mỗi biểu diễn tương ứng được ước tính bằng cây-Wasserstein để tính toán chi phí tối thiểu của việc vận chuyển một đối tượng đến đối tượng khác.

Để khắc phục hạn chế của GP, một số phương pháp sau được đề xuất:

- Mã hóa các kiến trúc trước tiên và sau đó đào tạo một mô hình chặng hạn, công cụ ước tính cây-Parzen,<sup>50,51</sup> rừng ngẫu nhiên,<sup>52,53</sup> hay mạng nơ-ron.<sup>54,55</sup>
- Một hướng khác là chiểu thông tin kiến trúc vào không gian ẩn liên tục có chiều thấp mà BO truyền thống có thể được áp dụng một cách hiệu quả.<sup>56,57</sup>
- Một lớp các mô hình ước tính xác suất khác sử dụng mạng thần kinh đồ thị<sup>58,59</sup> hoặc một hạt nhân dựa trên đồ thị<sup>60</sup> để tự nhiên xử lý biểu diễn đồ thị của các kiến trúc mà không cần mã hóa tường minh.

<sup>50</sup>Algorithms for Hyper-Parameter Optimization, 2011

<sup>51</sup>BOHB: Robust and Efficient Hyperparameter Optimization at Scale, 7/2018

<sup>52</sup>Sequential model-based optimization for general algorithm configuration, 2011

<sup>53</sup>Nas-bench-101: Towards reproducible neural architecture search, 5/2019

<sup>54</sup>Bayesian Optimization with Robust Bayesian Neural Networks

<sup>55</sup>Bananas: Bayesian optimization with neural architectures for neural architecture search, 11/2020

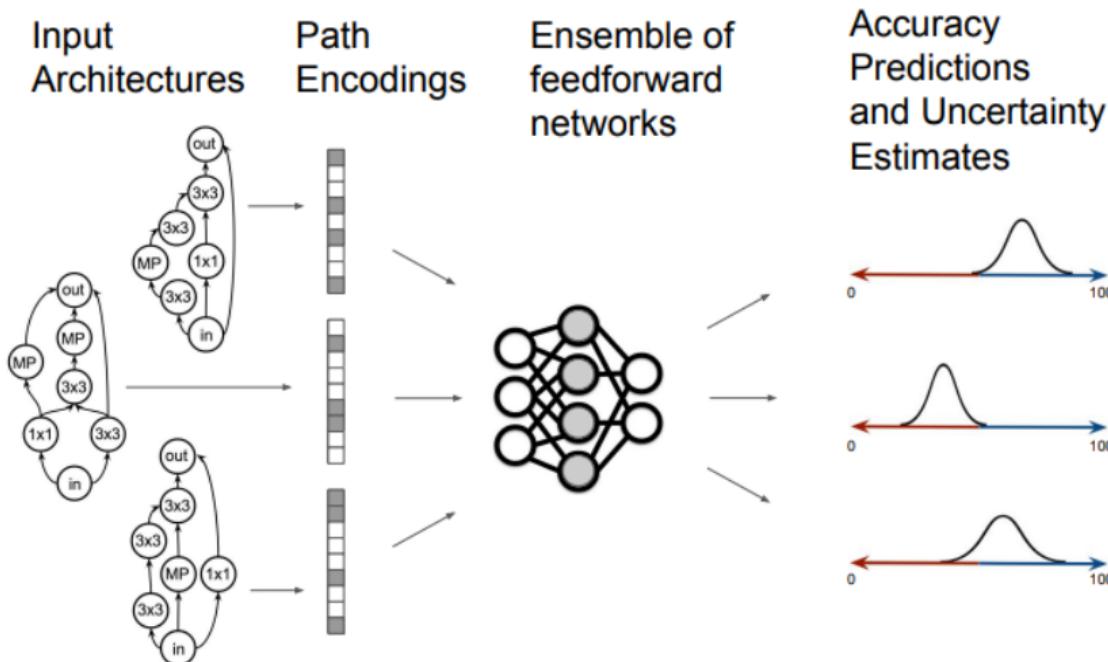
<sup>56</sup>Neural architecture generator optimization, 2020

<sup>57</sup>Approximate neural architecture search via operation distribution learning, 2022

<sup>58</sup>Deep neural architecture search with deep graph bayesian optimization, 2019

<sup>59</sup>Bridging the gap between sample-based and one-shot neural architecture search with bonas, 2020

<sup>60</sup>Neural architecture search using bayesian optimisation with weisfeiler-lehman kernel, 2021



Hình 70: Minh họa quy trình làm việc của BANANAS

*Hàm thu nhận* (acquisition function), có nhiệm vụ cân đối sự khám phá và khai thác trong quá trình tìm kiếm, là một thành phần thiết kế quan trọng khác của BO.

Có nhiều loại hàm thu nhận được sử dụng trong NAS:

- Cải thiện kỳ vọng (expected improvement).<sup>61</sup>
- Ngưỡng tin cậy cao (upper confidence bound).<sup>62</sup>
- Các hàm lý thuyết thông tin.<sup>63</sup>

---

<sup>61</sup>Efficient global optimization of expensive black-box functions, 1998

<sup>62</sup>Gaussian process optimization in the bandit setting: No regret and experimental design, 2010

<sup>63</sup>Predictive entropy search for efficient global optimization of black-box functions, 2014

Trong NAS, việc tối ưu hóa hàm thu nhận trong mỗi vòng BO là một thách thức do:

- KGTK không liên tục.
- Việc đánh giá giá trị hàm thu nhận trên tất cả các kiến trúc có thể có là không khả thi tính toán.

Phương pháp phổ biến nhất để tối ưu hóa hàm thu nhận trong NAS là bằng cách ngẫu nhiên biến đổi một nhóm nhỏ các kiến trúc tốt nhất được truy vấn cho đến nay, và trong số các kiến trúc đã biến đổi, chọn ra các kiến trúc có giá trị hàm thu nhận cao nhất.

Các phương pháp khác để tối ưu hóa hàm thu nhận bao gồm tìm kiếm địa phương, tìm kiếm tiến hóa và tìm kiếm ngẫu nhiên.

## Thuật toán 3: Thuật toán BO - NAS tổng quát

**Đầu vào:** Không gian tìm kiếm  $\mathcal{A}$ , số vòng lặp  $T$ , hàm thu nhận  $\phi$ .

1. Lấy mẫu ngẫu nhiên và huấn luyện một tập các kiến trúc từ  $\mathcal{A}$ .
2. Vòng lặp  $t = 1, 2, \dots, L$ :
  - Huấn luyện mô hình ước tính xác suất bằng tập các kiến trúc hiện có.
  - Chọn kiến trúc  $a_t = \arg \max_{a \in \mathcal{A}} \phi(a)$  dựa trên mô hình ước tính.
  - Huấn luyện kiến trúc  $a_t$  và thêm nó vào tập kiến trúc.

**Đầu ra:** Kiến trúc có độ chính xác trên tập xác thực cao nhất.

## Tìm kiếm cây Monte Carlo

Một lớp khác của các phương pháp NAS dựa trên **Tìm kiếm cây Monte Carlo** (Monte Carlo Tree Search - MCTS). MCTS tìm ra các quyết định tối ưu bằng cách:

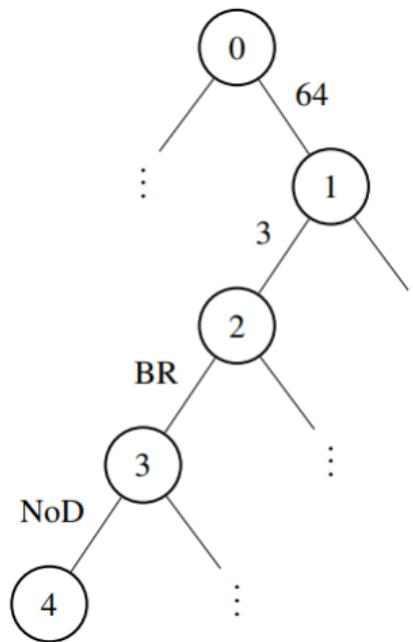
- Lấy mẫu để quy các quyết định mới, chạy các lượt mô phỏng ngẫu nhiên để nhận được phản thưởng và sau đó thực hiện lan truyền ngược để cập nhật trọng số của quyết định ban đầu.
- Qua các vòng lặp, thuật toán xây dựng một cây quyết định để điều chỉnh tìm kiếm vào các vùng triển vọng hơn bằng cách cân đối sự khám phá và khai thác trong quá trình đưa ra quyết định.

MCTS lần đầu được áp dụng vào NAS bởi Negrinho và Gordon,<sup>64</sup> người đã biểu diễn KGTK và siêu tham số của nó bằng một mô-đun ngôn ngữ (modular language).

- Dẫn đến một KGTK có cấu trúc cây, có khả năng mở rộng, ngược lại với KGTK cố định trong các nghiên cứu trước đó.
- Trong NAS, mỗi nút trên cây Monte Carlo biểu diễn một quyết định trong quá trình thiết kế kiến trúc mạng neural, như lựa chọn số lượng lớp, kích thước lớp, loại kích hoạt,...
- Cây được mở rộng từng bước một, với mỗi nút mới biểu diễn một lựa chọn kiến trúc cụ thể. Các chính sách trong MCTS - *chính sách cây* (tree policy) và *chính sách lượt mở phỏng* (rollout policy) - hỗ trợ quá trình điều hướng qua không gian kiến trúc để tìm ra các kiến trúc mạng có hiệu suất tốt.

---

<sup>64</sup>Deeparchitect: Automatically designing and training deep architectures, 2017



**Hình 71:** Minh họa mô hình được mã hóa bởi đường bao gồm một lớp tích chập với 64 bộ lọc kích thước  $3 \times 3$  và bước nhảy 1, tiếp theo là lớp chuẩn hóa theo lô, ReLU và affine. Mô hình này không sử dụng dropout. Các nhánh biểu diễn các siêu tham số chỉ có một lựa chọn đã được bỏ qua.

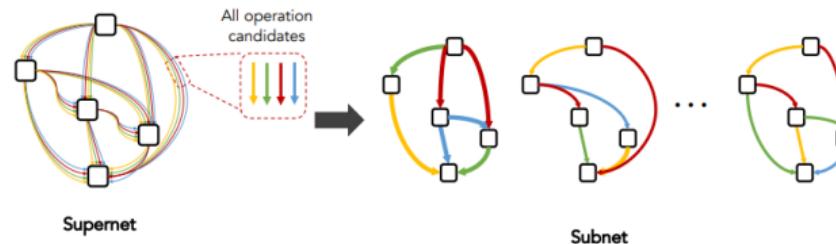
Các phương pháp được giới thiệu là chủ đạo trong giai đoạn đầu của nghiên cứu về NAS.

- Nguyên tắc chung: lặp đi lặp lại việc lấy mẫu các kiến trúc từ không gian tìm kiếm, huấn luyện chúng, và sử dụng hiệu suất của chúng để hướng dẫn việc tìm kiếm.
- Nhược điểm chính của các phương pháp này, khi áp dụng mà không có kỹ thuật tăng tốc, là chi phí tính toán cực lớn, đôi khi lên đến hàng nghìn ngày GPU do nhu cầu phải huấn luyện hàng nghìn kiến trúc một cách độc lập và từ đầu.

## Kỹ thuật một lần

**Kỹ thuật một lần** (One-shot Technique) đã được giới thiệu để tránh việc huấn luyện từng kiến trúc từ đầu, qua đó giảm bớt gánh nặng tính toán liên quan. Tính đến năm 2022, chúng hiện là một trong những kỹ thuật phổ biến nhất trong nghiên cứu NAS.

- Thay vì huấn luyện từng kiến trúc từ đầu, các phương pháp one-shot ngầm huấn luyện tất cả các kiến trúc trong KGTK thông qua một lần huấn luyện duy nhất cho hypernetwork (siêu mạng) hoặc supernet (tổng mạng).
- Hypernetwork (siêu mạng) là một mạng nơ-ron tạo ra trọng số cho các mạng nơ-ron khác.
- Supernet (tổng mạng, đồng nghĩa với *mô hình one-shot*) là một kiến trúc quá tham số hóa chứa tất cả các kiến trúc có thể trong không gian tìm kiếm dưới dạng các mạng con.

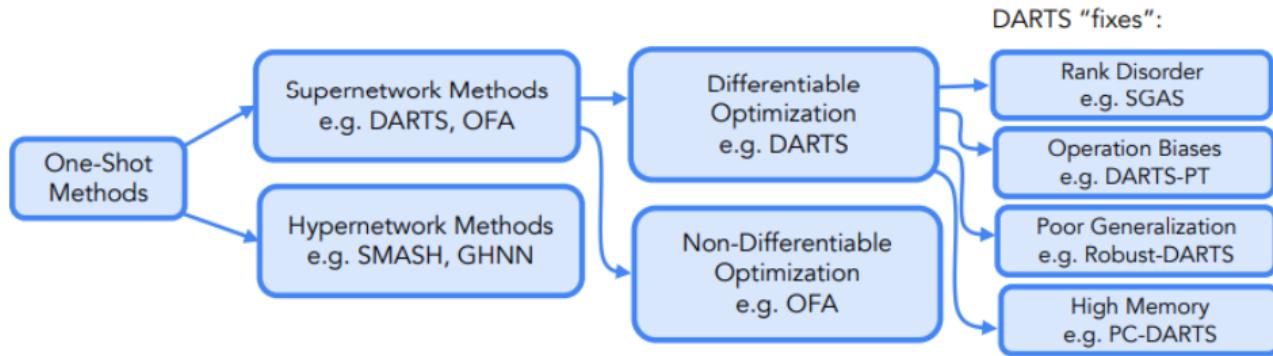


Hình 72: Minh họa về toàn mạng.

- Một khi supernet (tổng mạng) được huấn luyện, mỗi kiến trúc từ KGTK có thể được đánh giá bằng cách thừa hưởng trọng số từ subnet tương ứng trong supernet.
- Lý do cho khả năng mở rộng và hiệu quả của supernets là việc tăng truyền tính số lượng toán tử ứng viên chỉ gây ra sự tăng truyền tính trong chi phí tính toán cho việc huấn luyện, nhưng số lượng subnet trong supernet tăng theo cấp số nhân.
- Do đó, *supernets cho phép chúng ta huấn luyện một số lượng kiến trúc theo cấp số nhân với một chi phí tính toán tuyến tính*.

Giả định chính được đưa ra trong các phương pháp one-shot là khi sử dụng mô hình one-shot để đánh giá các kiến trúc, thứ hạng của các kiến trúc tương đối nhất quán với thứ hạng mà người ta sẽ nhận được từ việc huấn luyện chúng một cách độc lập.

- Giả định đã được tranh luận rộng rãi, với nhiều công trình chỉ ra bằng chứng nhưng cũng có phần chống lại.
- Tính hợp lệ của giả định phụ thuộc vào thiết kế KGTK, các kỹ thuật sử dụng để huấn luyện mô hình one-shot, và chính bộ dữ liệu, và khó dự đoán mức độ đúng giả định được duy trì trong một trường hợp cụ thể.



Hình 73: Phân loại họ các kỹ thuật one-shot

## Phương pháp Toàn mạng không khả vi

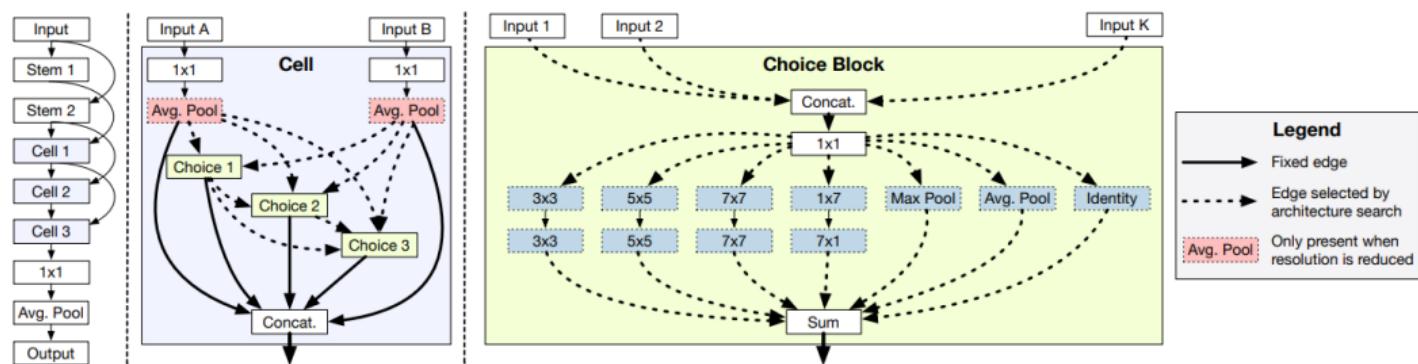
Họ các phương pháp **Toàn mạng không khả vi** (Non-Differentiable Supernet-Based Methods) gồm một số mà tách biệt việc huấn luyện supernet và tìm kiếm kiến trúc:

- i Đầu tiên huấn luyện một supernet.
- ii Sau đó chạy một thuật toán tối ưu hóa hộp đen để tìm kiến trúc tốt nhất.

Các phương pháp khác huấn luyện supernet đồng thời chạy một thuật toán tìm kiếm không khả vi, như học tăng cường, để chọn các mạng con.

Bender và đồng nghiệp<sup>65</sup> xây dựng supernet bằng cách:

- Tạo các nút riêng biệt tương ứng với các toán tử, ở mọi nơi có sự lựa chọn toán tử.
- Sau đó họ huấn luyện supernet như thể nó là một mạng nơ-ron chuẩn.
- Với một ngoại lệ: các nút được loại bỏ ngẫu nhiên trong quá trình huấn luyện, với mức độ loại bỏ tăng dần theo thời gian.



**Hình 74:** Đường连线 biểu thị các thành phần có mặt trong mọi kiến trúc, trong khi các đường đứt nét biểu thị các thành phần tùy chọn là một phần của KGTK.

<sup>65</sup> Understanding and simplifying one-shot architecture search, 2018

Trong công trình nối tiếp, Li và Talwalkar<sup>66</sup> và Guo và đồng nghiệp<sup>67</sup> đưa ý tưởng này xa hơn:

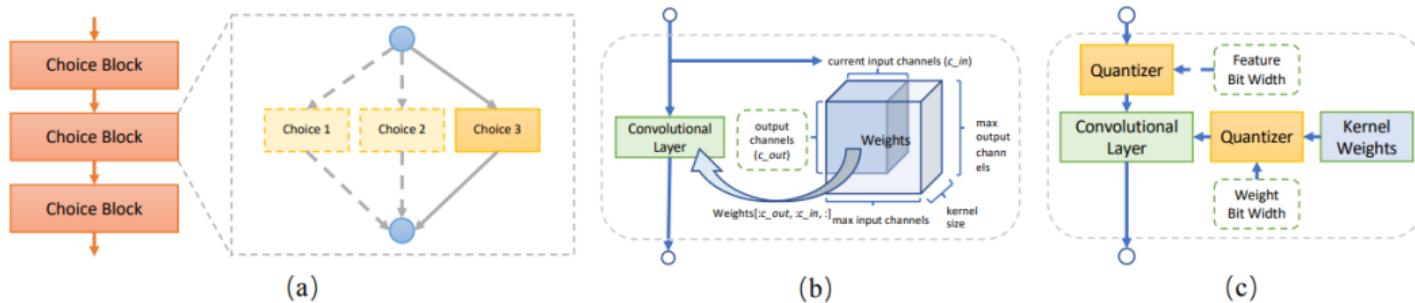
- Trong mỗi bước huấn luyện, họ ngẫu nhiên chọn một kiến trúc và chỉ cập nhật trọng số của supernet tương ứng với kiến trúc đó.
- Các kỹ thuật này mô phỏng tốt hơn những gì diễn ra khi đánh giá: chỉ đánh giá một mạng con thay vì toàn bộ supernet.
- Hơn nữa, các quy trình này sử dụng ít bộ nhớ hơn nhiều so với việc huấn luyện tất cả các trọng số của một supernet.

Mỗi phương pháp kết thúc bằng việc sử dụng supernet đã huấn luyện để nhanh chóng đánh giá các kiến trúc khi tiến hành tìm kiếm ngẫu nhiên (Bender; Li và Talwalkar) hoặc tìm kiếm tiến hóa (Guo). Kiến trúc được xác định cuối cùng sau đó được huấn luyện từ đầu.

---

<sup>66</sup>Random search and reproducibility for neural architecture search, 2019

<sup>67</sup>Single path one-shot neural architecture search with uniform sampling, 2020



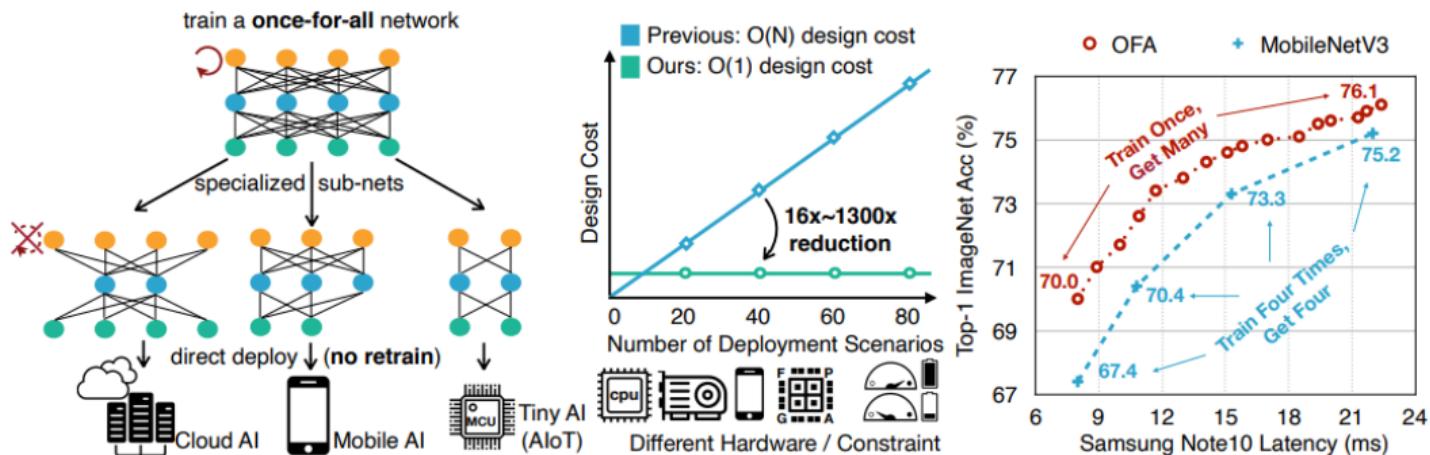
**Hình 75:** Supernet trong phương pháp **SPOS** (Single Path One-Shot) được thiết kế để mỗi kiến trúc chỉ bao gồm một đường dẫn duy nhất, trong quá trình huấn luyện supernet, mỗi kiến trúc được lấy mẫu một cách đồng đều. Để xây dựng supernet, phương pháp này sử dụng các “khối lựa chọn” (choice blocks), trong đó mỗi khối bao gồm: (a) khối cho đường dẫn đơn, (b) khối cho tìm kiếm số kênh, (c) khối cho tìm kiếm lượng tử chính xác hỗn hợp (mixed-precision quantization search).

Việc triển khai mạng nơ-ron trong thực tế thường đi kèm với các ràng buộc về độ trễ hoặc bộ nhớ. Trong khi các supernet được xem xét cho đến nay thường chỉ chứa các kiến trúc có kích thước xấp xỉ như nhau, Cai và đồng nghiệp<sup>68</sup> đề xuất một supernet chứa các mạng con với kích thước khác nhau.

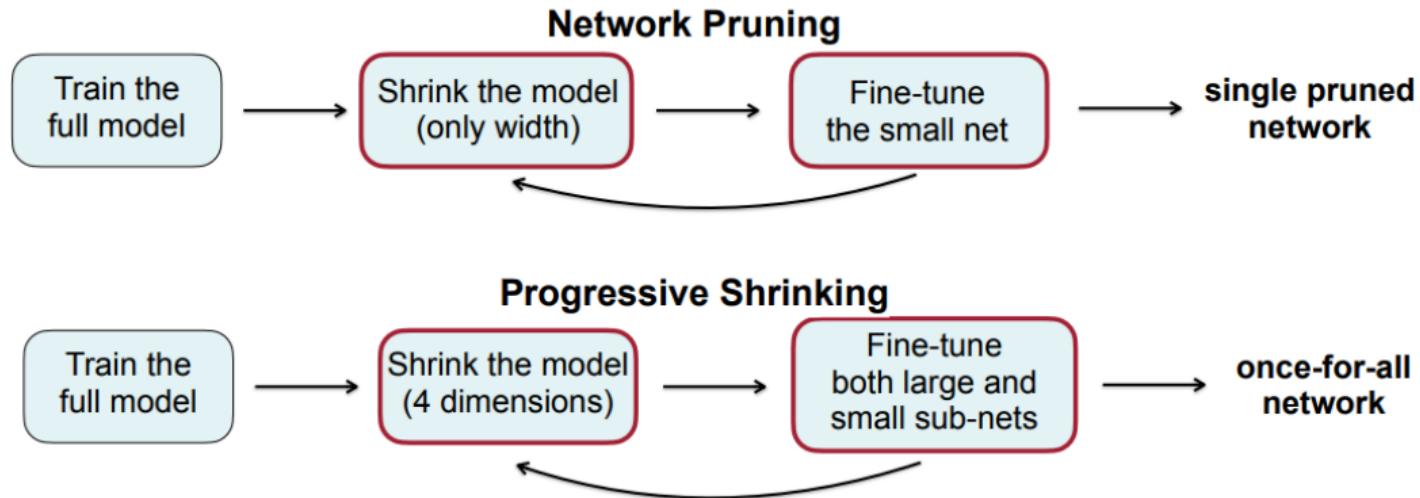
- Phương pháp Once-for-all (OFA) này sử dụng một chiến lược co gọn tiên tiến (progressive shrinking strategy) bắt đầu bằng việc lấy mẫu các mạng con lớn nhất, sau đó chuyển sang các mạng con nhỏ hơn, nhằm giảm thiểu sự đồng thích nghi (co-adaptation) giữa các mạng con và huấn luyện hiệu quả các mạng có kích thước khác nhau “một lần cho tất cả”.
- Sử dụng tìm kiếm ngẫu nhiên hoặc tìm kiếm tiên hóa cho giai đoạn sau đó.

---

<sup>68</sup>Once-for-all: Train one network and specialize it for efficient deployment, 2020



**Hình 76:** Bên trái: một mạng once-for-all duy nhất được huấn luyện để hỗ trợ các cấu hình kiến trúc linh hoạt bao gồm độ sâu, độ rộng, kích thước kernel và độ phân giải. Trong một kịch bản triển khai cụ thể, một mạng con chuyên biệt được trực tiếp lựa chọn từ mạng once-for-all mà không cần huấn luyện. Ở giữa: phương pháp này giảm chi phí triển khai học sâu chuyên biệt từ  $O(N)$  xuống  $O(1)$ . Bên phải: mạng once-for-all sau bối lựa chọn mô hình có thể tạo ra nhiều sự đánh đổi giữa độ chính xác và độ trễ chỉ bằng cách huấn luyện một lần, so với các phương pháp thông thường yêu cầu huấn luyện lặp đi lặp lại.



**Hình 77:** Co gọn tiên tiến có thể được xem như một kỹ thuật cắt tỉa mạng nơ-ron tổng quát hóa với độ linh hoạt cao hơn nhiều. So với cắt tỉa mạng, nó thu hẹp nhiều chiều hơn (không chỉ là độ rộng) và cung cấp một mạng once-for-all mạnh mẽ hơn nhiều có thể phù hợp với các kịch bản triển khai khác nhau thay vì chỉ một mạng đã được cắt tỉa.

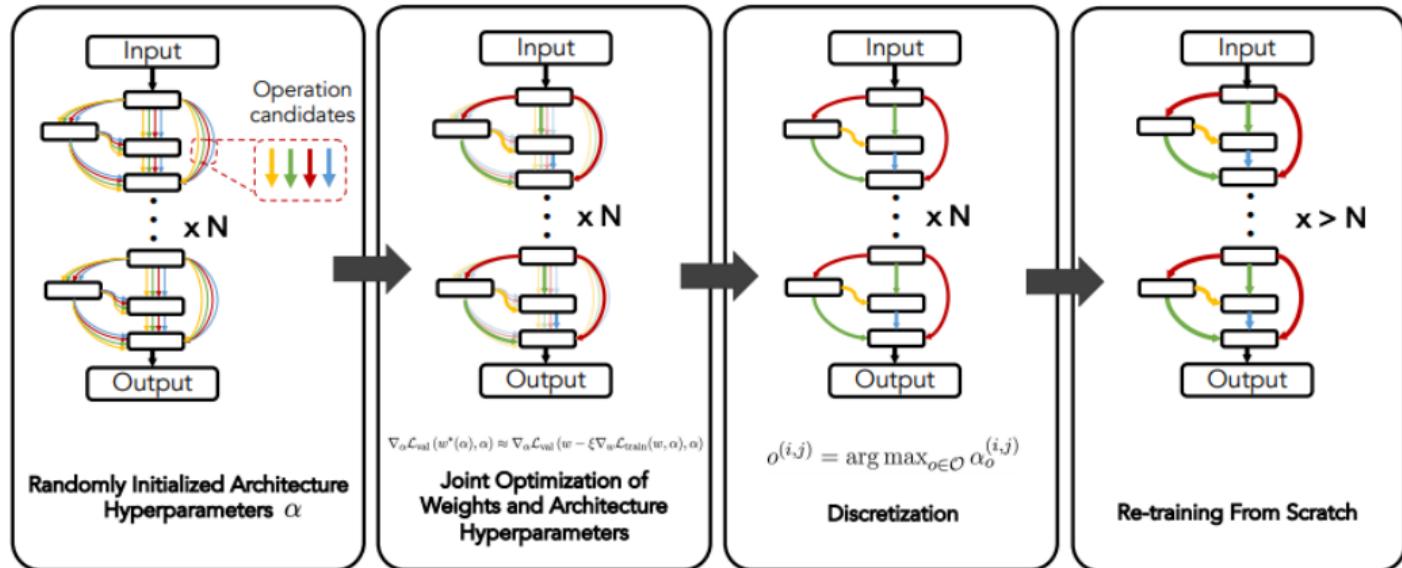
## Phương pháp Toàn mạng khả vi

Công trình tiên phong trong phương pháp này của Liu và đồng nghiệp,<sup>69</sup> gọi là **Tìm kiếm kiến trúc khả vi** (Differentiable architecture search - DARTS).

- Phương pháp DARTS sử dụng một giản lược liên tục (continuous relaxation) của KGTK kiến trúc rời rạc, cho phép sử dụng gradient descent để tìm kiếm một cực tiểu địa phương hiệu suất cao nhanh hơn đáng kể so với các phương pháp tối ưu hóa hộp đen.
- Nó thể được áp dụng cho bất kỳ KGTK dựa trên đồ thị (DAG) nào có các lựa chọn khác nhau về toán tử trên mỗi cạnh bằng cách sử dụng toán tử “zero” để mô phỏng sự vắng mặt của một cạnh.

---

<sup>69</sup>DARTS: Differentiable architecture search, 2019



Hình 78: Minh họa các bước của giải thuật DARTS.

- Ban đầu, mỗi cạnh  $(i, j)$  trong KGTK DARTS bao gồm nhiều toán tử ứng viên  $o$ , mỗi toán tử có liên quan đến một siêu tham số liên tục  $\alpha_o^{(i,j)} \in [0, 1]$ .
- Trong quá trình huấn luyện supernet, cạnh  $(i, j)$  bao gồm một sự kết hợp của tất cả các toán tử ứng viên, được gán trọng số bởi  $\alpha_o^{(i,j)}$  tương ứng.
- Các tham số kiến trúc  $\alpha$  được tối ưu hóa cùng với trọng số mô hình supernet  $w$  thông qua việc sử dụng gradient descent xen kẽ.

- Cụ thể, để cập nhật các trọng số kiến trúc  $\alpha$  bằng gradient descent, DARTS sử dụng xấp xỉ sau đây:

$$\nabla_{\alpha} \mathcal{L}_{\text{validation}}(w^*(\alpha), \alpha) \approx \nabla_{\alpha} \mathcal{L}_{\text{validation}}(w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha), \alpha)$$

Với  $\mathcal{L}_{\text{validation}}$  và  $\mathcal{L}_{\text{train}}$  tương ứng là lỗi trên tập xác thực và tập huấn luyện,  $\xi$  là tốc độ học,  $w^*(\alpha)$  là trọng số làm tối thiểu lỗi huấn luyện với kiến trúc  $\alpha$ .

- Mặc dù chiến lược này không đảm bảo sẽ hội tụ, Liu và đồng nghiệp đã chứng minh rằng nó hoạt động tốt trong thực tế với sự lựa chọn thích hợp của  $\xi$ .
- Sau giai đoạn huấn luyện, DARTS thu được một kiến trúc rời rạc bằng cách chọn phép hoạt động có giá trị  $\alpha$  lớn nhất trên mỗi cạnh và sau đó huấn luyện lại từ đầu.

## Thuật toán 4: Thuật toán DARTS cơ bản

**Đầu vào:** Không gian tìm kiếm  $\mathcal{A}$ , số vòng lặp  $T$ , tốc độ học  $\xi$ .

1. Khởi tạo ngẫu nhiên mô hình one-shot dựa trên không gian tìm kiếm  $A$  với trọng số  $w$  và siêu tham số kiến trúc  $\alpha$ .
2. Vòng lặp  $t = 1, 2, \dots, T$ :
  - Cập nhật  $\alpha$ :  $\alpha \leftarrow \alpha - \nabla_\alpha \mathcal{L}_{\text{validation}}(w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha), \alpha)$ .
  - Cập nhật  $w$ :  $w \leftarrow w - \nabla_w \mathcal{L}_{\text{train}}(w, \alpha)$ .

**Đầu ra:** Chọn toán tử cho cạnh  $(i, j)$  là  $o^{(i,j)} = \arg \max_o \alpha_o^{(i,j)}$  và huấn luyện kiến trúc thu được từ đầu.

## Phương pháp Siêu mạng

Công trình đầu tiên sử dụng siêu mạng cho NAS là SMASH<sup>70</sup> (One-Shot Model Architecture Search through Hypernetworks). SMASH gồm 2 giai đoạn:

1. Đầu tiên, huấn luyện một siêu mạng để tạo ra các trọng số cho bất kỳ kiến trúc nào trong KGTK.
2. Tiếp theo, lấy mẫu ngẫu nhiên một tập hợp lớn các kiến trúc, tạo ra các trọng số của chúng bằng cách sử dụng siêu mạng và xuất ra kiến trúc có độ chính xác trên tập xác thực tốt nhất.

Siêu mạng, một mạng nơ-ron tích chập, nhận đầu vào là một mã hóa kiến trúc và tạo ra một bộ trọng số cho kiến trúc đó, và được huấn luyện bằng cách lấy mẫu ngẫu nhiên một kiến trúc, tạo ra các trọng số của nó, tính toán lỗi huấn luyện và sau đó thực hiện lan truyền ngược qua toàn bộ hệ thống (bao gồm cả trọng số của siêu mạng).

---

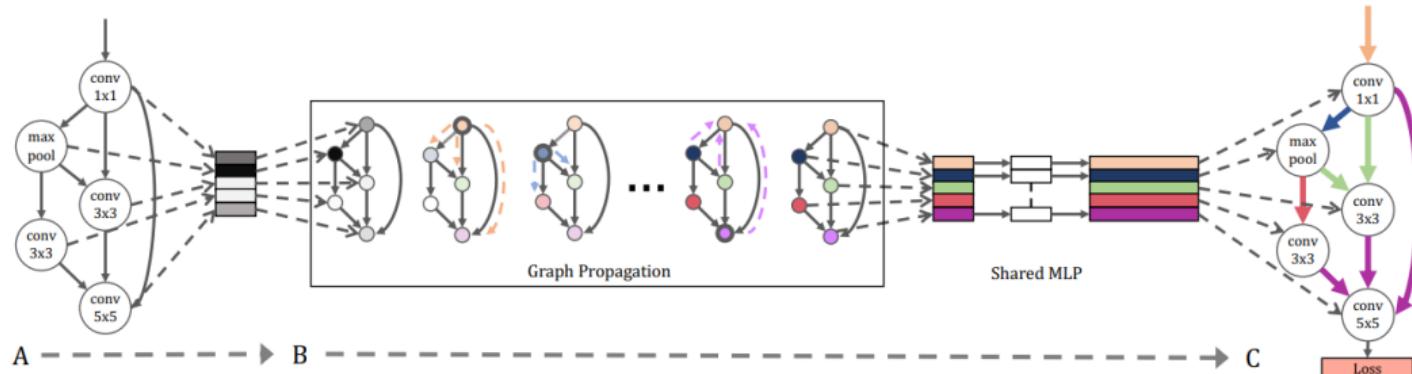
<sup>70</sup>SMASH: One-Shot Model Architecture Search through HyperNetworks, 2018

Một thuật toán NAS khác dựa trên mạng siêu là GHN<sup>71</sup> (Graph Hypernetworks).

- Sự khác biệt chính giữa SMASH và GHN là cách mã hóa kiến trúc và kiến trúc của siêu mạng.
- Siêu mạng GHN là sự kết hợp giữa mạng nơ-ron đồ thị (GNN) và siêu mạng tiêu chuẩn: nhận đầu vào là đồ thị tính toán của một kiến trúc và sử dụng các phép toán truyền thông điệp phổ biến trong GNN để tạo ra các trọng số cho kiến trúc đó.

---

<sup>71</sup>Graph hypernetworks for neural architecture search, 2018



**Hình 79:** Sơ đồ quy trình GHN. A: Một kiến trúc mạng nơ-ron được lấy mẫu ngẫu nhiên, tạo thành một GHN. B: Sau khi lan truyền đồ thị, mỗi nút trong GHN tạo ra các tham số trọng số riêng của nó. C: GHN được huấn luyện để giảm thiểu sự mất mát trong quá trình huấn luyện của mạng đã được lấy mẫu với các trọng số được tạo ra. Các mạng ngẫu nhiên được xếp hạng dựa trên hiệu suất của chúng sử dụng các trọng số được tạo ra bởi GHN.

# CHIẾN LƯỢC ĐÁNH GIÁ HIỆU SUẤT

Các kỹ thuật tăng tốc tổng quát trong NAS bao gồm:

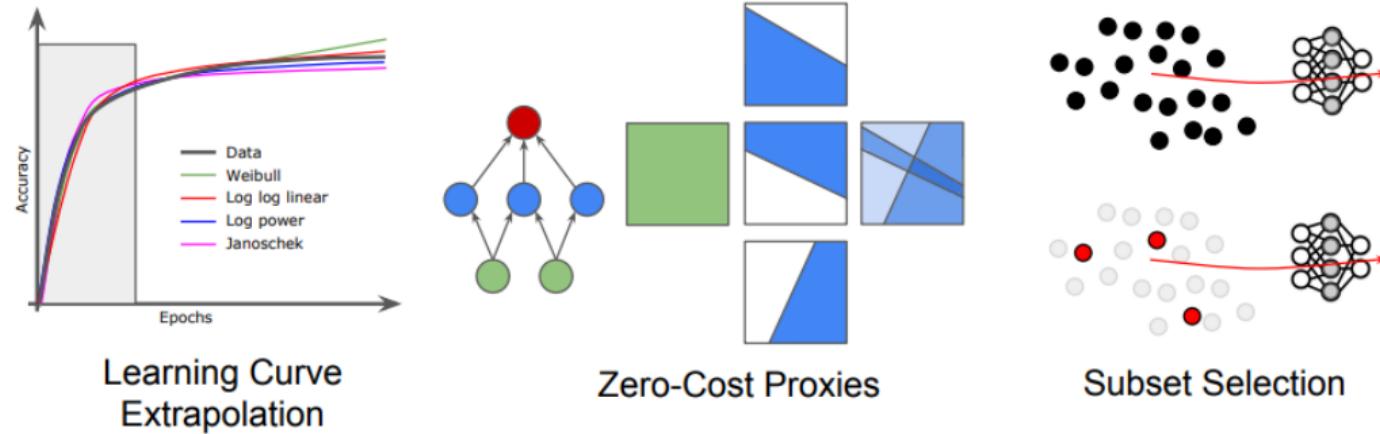
- Dự đoán hiệu suất (performance prediction).
- Phương pháp đa tin cậy (multi-fidelity methods).
- Học siêu (meta-learning approaches).
- Kế thừa trọng số (weight inheritance).

## Dự đoán hiệu suất

Các kỹ thuật này có tiềm năng làm tăng tốc độ thực thi của các thuật toán NAS, vì chúng loại bỏ nhu cầu huấn luyện hoàn toàn mỗi kiến trúc được xem xét. Việc áp dụng chúng cải thiện gần như tất cả các loại thuật toán NAS, từ tối ưu hóa hộp đen đến one-shot.

Một cách hình thức, cho một không gian tìm kiếm  $\mathcal{A}$  và kiến trúc  $a \in \mathcal{A}$ , ký hiệu độ chính xác cuối cùng thu được với một quy trình huấn luyện cố định là  $f(a)$ .

- Một bộ dự đoán hiệu suất  $f^*$  được định nghĩa là bất kỳ hàm nào dự đoán độ chính xác hoặc độ chính xác tương đối của các kiến trúc, mà không cần huấn luyện hoàn toàn chúng.
- Nói cách khác, việc đánh giá  $f^*(a)$  tốn ít thời gian hơn so với việc đánh giá  $f(a)$ , và  $\{f^*(a) : a \in \mathcal{A}\}$  lý tưởng có tương quan cao hoặc tương quan hạng với  $\{f(a) : a \in \mathcal{A}\}$ .



**Hình 80:** Minh họa về các loại chính của các dự đoán hiệu suất: ngoại suy đường cong học tập về độ chính xác xác thực thông qua một mô hình tham số hóa (bên trái), đánh giá tính tổng quát của một kiến trúc thông qua một lan truyền tiên duy nhất của một minibatch dữ liệu (ở giữa), và huấn luyện kiến trúc trên một phần dữ liệu (bên phải).

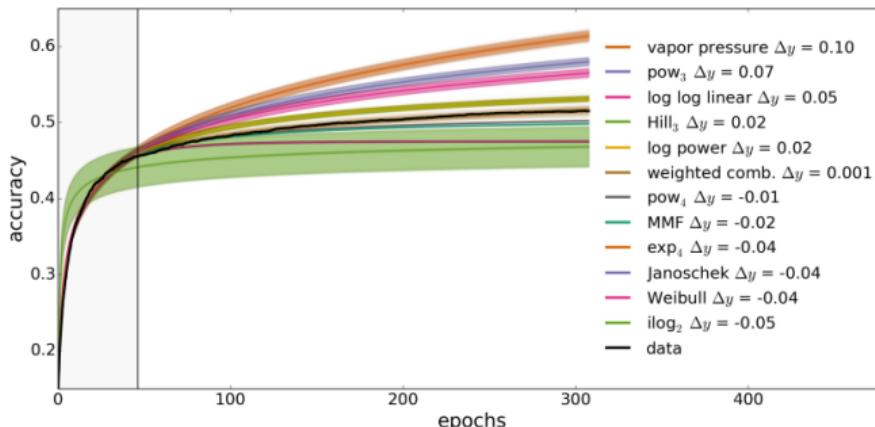
## Ngoại suy đường cong học tập

Các phương pháp **ngoại suy đường cong học tập** (learning curve extrapolation) có gắng dự đoán hiệu suất cuối cùng của một kiến trúc cụ thể sau khi huấn luyện một phần, bằng cách ngoại suy từ đường cong học tập một phần của nó (chuỗi độ chính xác trên tập xác thực ở tất cả các vòng lặp huấn luyện cho đến nay).<sup>72</sup>

Các phương pháp ngoại suy đường cong học tập cũng có thể được sử dụng kết hợp với một mô hình ước lượng (surrogate model): trong trường hợp đó, mô hình nhận đầu vào là cả mã hóa của  $a$  và đường cong học tập một phần của  $a$ , và xuất ra một dự đoán  $f^*(a)$ .

---

<sup>72</sup>Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves, 2015



Reference name	Formula
vapor pressure	$\exp(a + \frac{b}{x} + c \log(x))$
pow <sub>3</sub>	$c - ax^{-\alpha}$
log log linear	$\log(a \log(x) + b)$
Hill <sub>3</sub>	$\frac{y_{\max} x^{\eta}}{\kappa^{\eta} + x^{\eta}}$
log power	$\frac{a}{1 + \left(\frac{x}{e^b}\right)^c}$
pow <sub>4</sub>	$c - (ax + b)^{-\alpha}$
MMF	$\alpha - \frac{\alpha - \beta}{1 + (\kappa x)^{\delta}}$
exp <sub>4</sub>	$c - e^{-ax^{\alpha} + b}$
Janoschek	$\alpha - (\alpha - \beta)e^{-\kappa x^{\delta}}$
Weibull	$\alpha - (\alpha - \beta)e^{-(\kappa x)^{\delta}}$
ilog <sub>2</sub>	$c - \frac{a}{\log x}$

**Hình 81:** Bên trái: Một đường học điển hình và các dự đoán ngoại suy từ phần đầu tiên của nó, với mỗi trong 11 mô hình tham số riêng biệt. Chú thích được sắp xếp theo phần dư của các dự đoán tại epoch 300. Bên phải: các công thức cho 11 mô hình tham số.

## Chỉ số không tốn kém

**Chỉ số không tốn kém** (zero-cost proxies) là một họ các kỹ thuật dự đoán hiệu suất được phát triển gần đây.

- Ý tưởng là thực hiện tính toán rất nhanh (như chạy qua một lượt tiến và lùi của một minibatch dữ liệu duy nhất) trên một tập hợp các kiến trúc để gán một điểm số (score) cho mỗi kiến trúc, với hy vọng rằng các điểm này tương quan với độ chính xác cuối cùng.<sup>73</sup>
- Một số chỉ số khác không phụ thuộc vào dữ liệu, tính điểm chỉ từ trọng số đã khởi tạo hoặc số lượng tham số của mạng nơ-ron.
- Nhược điểm chính của zero-cost proxies là chúng có thể không đáng tin cậy và có thể có thiên lệch, đặc biệt trên các không gian tìm kiếm lớn hơn.

---

<sup>73</sup>Neural architecture search without training, 2021

## Các dự đoán kém tin cậy khác

Ngoài việc huấn luyện ít epoch, các công trình **dự đoán kém tin cậy** (low-fidelity prediction) cung cấp một ước tính về độ chính xác cuối cùng thông qua việc huấn luyện trên một tập con của dữ liệu huấn luyện (hoặc một tập dữ liệu nhỏ hơn được tạo tổng hợp).

Nhiều công trình đã nghiên cứu các thuật toán lựa chọn tập con khác nhau:

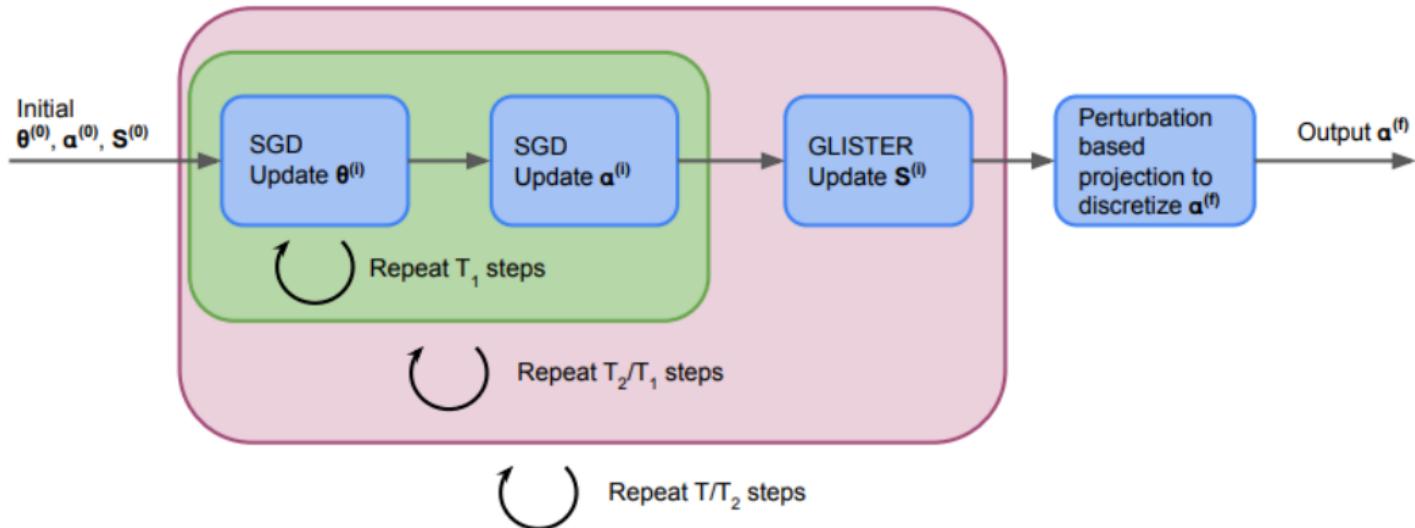
- Mẫu ngẫu nhiên (random sampling).
- Mẫu dựa trên entropy (entropy-based sampling).<sup>74</sup>
- Phân cụm thông qua tập hạt nhân (clustering via core-sets).<sup>75</sup>
- Vị trí cơ sở (facility location).<sup>76</sup>
- $k$ -tâm ( $k$ -center).

---

<sup>74</sup>Accelerating neural architecture search via proxy data, 2021

<sup>75</sup>Core-set sampling for efficient neural architecture search, 2021

<sup>76</sup>Speeding up NAS with adaptive subset selection, 2022



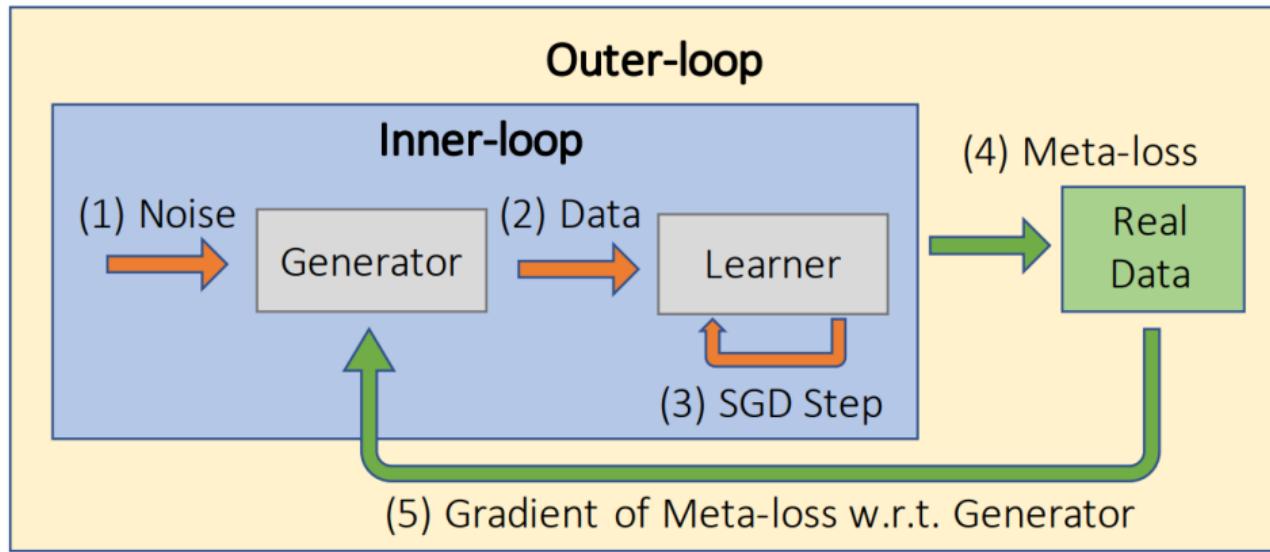
**Hình 82:** Tổng quan về ADAPTIVE-DPT. Thuật toán bắt đầu với tập trọng số ban đầu  $\theta^{(0)}$ , các tham số kiến trúc  $\alpha^{(0)}$  và tập con của dữ liệu huấn luyện  $S^{(0)}$ . Trong suốt quá trình tìm kiếm, các trọng số  $\theta^{(i)}$  và các tham số kiến trúc  $\alpha^{(i)}$  được cập nhật bằng SGD, và tập con  $S^{(i)}$  được cập nhật bằng **GLISTER** (Generalization based data subset selection for efficient and robust learning), theo các lịch trình thời gian khác nhau. Sau đó, kiến trúc cuối cùng  $\alpha^{(f)}$  được rời rạc hóa và trả về.

Such và đồng nghiệp<sup>77</sup> giới thiệu **mạng truyền đạt tạo sinh** (generative teaching networks):

- Sử dụng một tập nhỏ dữ liệu giả tạo (synthetic data) để huấn luyện mạng nơ-ron nhanh hơn nhiều so với việc sử dụng dữ liệu huấn luyện thực tế ban đầu.
- Dữ liệu giả tạo được tạo ra bằng cách sử dụng một mạng dữ liệu tạo sinh (data-generating network) để khớp với độ chính xác của một mạng đã được huấn luyện trên dữ liệu thực tế.

---

<sup>77</sup> Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data, 2020



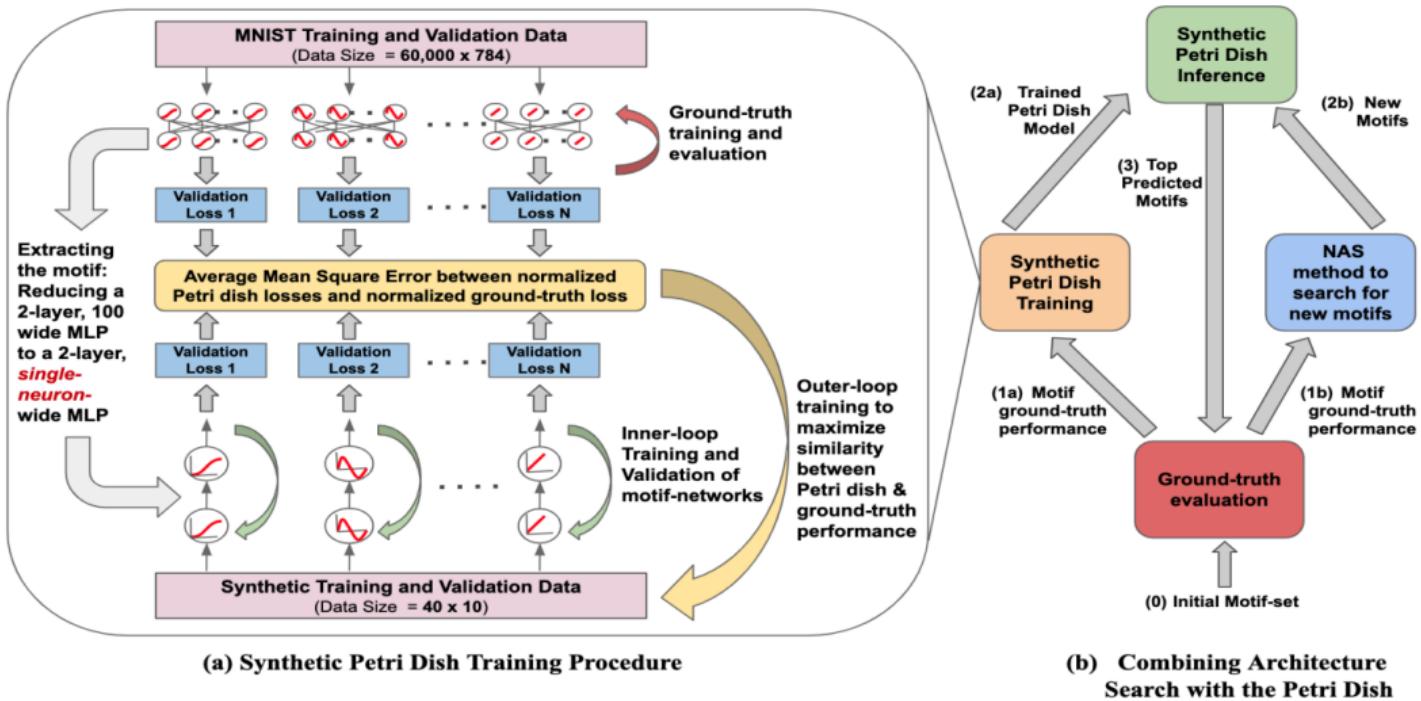
**Hình 83:** Phương pháp GTN (Generative Teaching Network). (1) nhiễu được đưa vào Bộ tạo dữ liệu. (2) sử dụng nó để tạo ra dữ liệu mới, (3) người học được huấn luyện (ví dụ, bằng cách sử dụng SGD hoặc Adam) để hoạt động tốt trên dữ liệu được tạo ra. (4) người học đã được huấn luyện được đánh giá trên dữ liệu huấn luyện thực tế trong vòng lặp bên ngoài để tính toán siêu mốt mót (meta-loss). (5) gradient của các tham số của bộ tạo dữ liệu được tính đổi với siêu mốt mót để cập nhật bộ tạo dữ liệu.

Một phương pháp liên quan là **đĩa petri giả tạo**<sup>78</sup> (synthetic petri dish):

- Đánh giá các mẫu kiến trúc bằng cách đặt chúng vào một mạng nơ-ron nhỏ và sau đó huấn luyện chúng bằng một tập dữ liệu tổng hợp nhỏ.
- Phương pháp này cũng tường minh tối ưu hóa sự tương quan giữa xếp hạng kiến trúc với sự xấp xỉ và huấn luyện đầy đủ.

---

<sup>78</sup>Synthetic petri dish: A novel surrogate model for rapid architecture search, 2020



Hình 84: (a) quá trình huấn luyện đĩa petri giả tạo và (b) kết hợp tìm kiếm kiến trúc với đĩa petri.

## Các thuật toán đa tin cậy

Chúng ta sẽ xem xét các thuật toán sử dụng những phương pháp dự đoán hiệu suất để chạy NAS một cách hiệu quả.

Một cách chính thức, hàm mục tiêu  $f : \mathcal{X} \rightarrow \mathbb{R}$ , thường tồn kém để đánh giá hoàn toàn, có thể được xấp xỉ một cách rẻ tiền bởi phiên bản kém tin cậy hơn  $\hat{f}(\cdot, b)$  của  $f(\cdot)$ , được tham số hóa bởi tham số độ tin cậy  $b$ . Khi  $b = b_{\max}$ , chúng ta nhận được hàm thật sự  $f(\cdot) = \hat{f}(\cdot, b_{\max})$ .

Đây là một sự tổng quát hóa của định nghĩa trước. Tham số độ tin cậy  $b$  có thể biểu thị số epoch huấn luyện, kích thước tập con dữ liệu huấn luyện.

**Liên tục chia đôi**<sup>79</sup> (Successive Halving - SH) là một trong những thuật toán đa tin cậy đơn giản nhất.

- Nó bắt đầu huấn luyện một số lượng lớn các kiến trúc, từ từ loại bỏ càng lúc càng nhiều các kiến trúc không triển vọng dựa trên các đánh giá có độ tin cậy thấp hơn, cho đến khi chỉ còn lại các kiến trúc triển vọng nhất được đánh giá với độ tin cậy cao nhất.
- Ngưỡng độ tin cậy và số lượng kiến trúc được thăng chức lên độ tin cậy cao hơn được điều khiển bằng một siêu tham số.

---

<sup>79</sup>Non-stochastic best arm identification and hyperparameter optimization, 2016

Một cải tiến phổ biến của SH là **Siêu dài**<sup>80</sup> (Hyperband - HB):

- Là một chiến lược **máy đánh bạc nhiều tay** (multi-armed bandit) mà lặp đi lặp lại việc gọi SH như một chương trình con.
- Sử dụng các giá trị khác nhau cho ngân sách tối thiểu (giới hạn về tài nguyên) trong mỗi lần gọi.

Do đó, HB đánh cược cho nhiều lựa chọn về ngân sách tối thiểu thay vì dựa vào một lựa chọn cụ thể.

---

<sup>80</sup>Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018

Các công trình tiêu biểu gần đây áp dụng SH/HB:

- **BOHB**<sup>81,82</sup> (Bayesian Optimization Hyperband): hoạt động tương tự HB trong lần lặp đầu tiên, và trong các lần lặp sau đó, nó khớp một mô hình ước lượng xác suất cho mỗi độ tin cậy theo thứ tự để đưa ra quyết định lấy mẫu sáng suốt.
- **DEHB**<sup>83</sup>: kết hợp DE (Tiến hóa vi phân - Differential Evolution) với HB, cải thiện đáng kể các lần lặp sau của HB.
- **ASHA**<sup>84</sup> và **ABOHB**<sup>85</sup>: cải thiện SH và BOHB thêm nữa bằng cách sử dụng tính toán song song bất đồng bộ (parallel asynchronous computation) lớn và chiến lược dừng sớm.
- **EcoNAS**<sup>86</sup>: đề xuất một phương pháp tìm kiếm tiến hóa phân cấp chia KGTK thành các tập con và phân bổ tăng cấp độ tin cậy cho các kiến trúc triển vọng nhất trong mỗi tập.

---

<sup>81</sup>BOHB: Robust and efficient hyperparameter optimization at scale, 2018

<sup>82</sup>SMAC3: A versatile bayesian optimization package for hyperparameter optimization, 2022

<sup>83</sup>DEHB: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization, 2021

<sup>84</sup>A system for massively parallel hyperparameter tuning, 2020

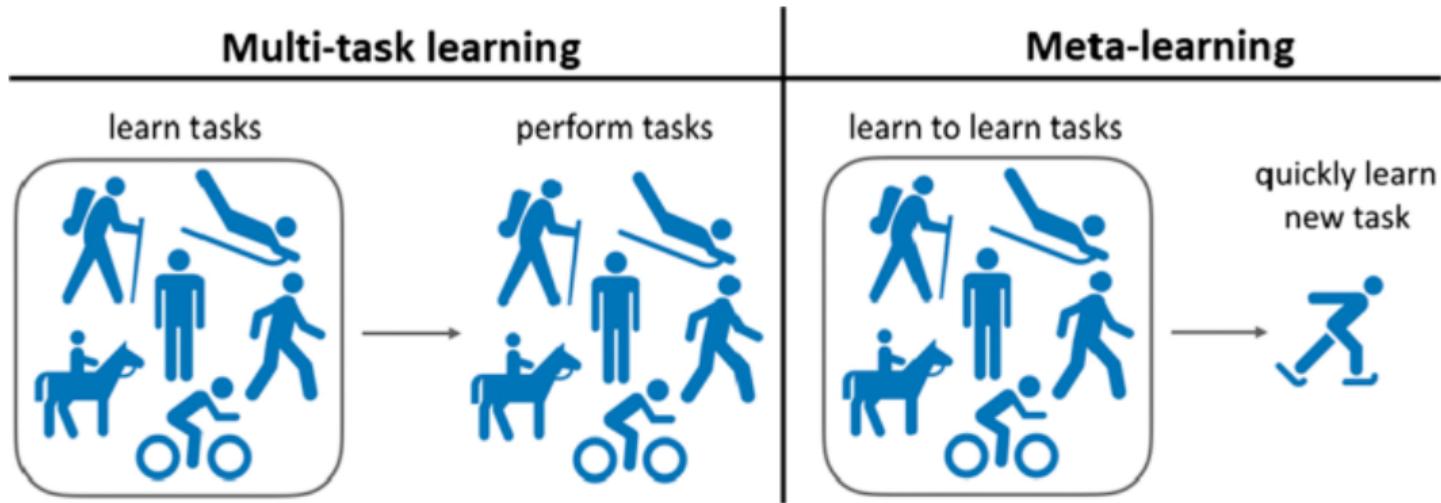
<sup>85</sup>Model-based asynchronous hyperparameter and neural architecture search, 2020

<sup>86</sup>Econas: Finding proxies for economical neural architecture search, 2020

## Học siêu

- Đa số các phương pháp NAS xem xét việc giải quyết một nhiệm vụ duy nhất từ đầu, bỏ qua các giải pháp đã khám phá trước đó.
- Vì vậy, có thể đặt câu hỏi: *tại sao lại chạy NAS từ đầu thay vì tái sử dụng thông tin từ các thí nghiệm trước đó?*

Câu hỏi này tự nhiên dẫn đến ý tưởng về **học siêu** (meta-learning) hoặc **học về cách học** (learning to learn), mục tiêu là cải thiện một thuật toán bằng cách tận dụng thông tin từ các thí nghiệm liên quan trước đó.



**Hình 85: Bên trái:** *Học đa nhiệm* là quá trình học nhiều nhiệm vụ cùng lúc với mục tiêu là để mô hình có khả năng tổng quát hóa và chia sẻ kiến thức giữa các nhiệm vụ, giúp cải thiện hiệu suất trên từng nhiệm vụ cụ thể. **Bên phải:** *Học siêu thì* mô hình được huấn luyện để nhanh chóng thích ứng với các nhiệm vụ mới dựa trên kinh nghiệm từ các nhiệm vụ đã học trước đó.

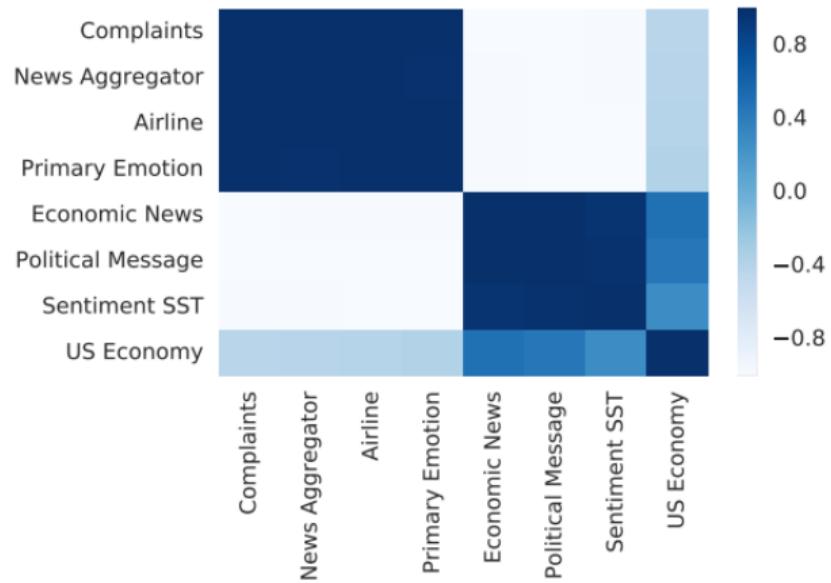
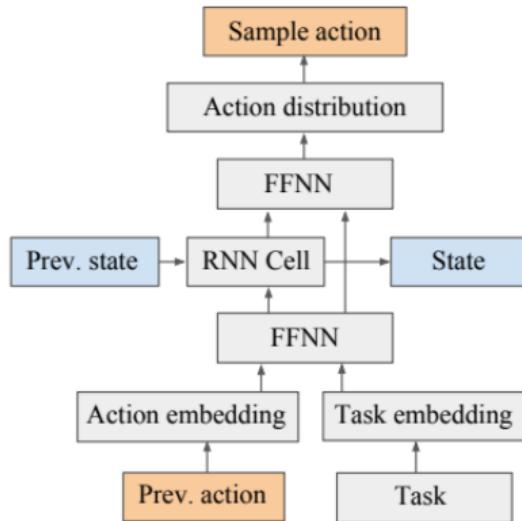
Wong cùng đồng nghiệp<sup>87</sup> và Zimmer cùng đồng nghiệp<sup>88</sup> áp dụng các chiến lược học siêu trong một cài đặt AutoML chung hơn.

- Với Wong, các nhiệm vụ được mã hóa tương tự như cách mã hóa từ trong NLP.
- Zimmer đơn giản chỉ khởi động lại tìm kiếm dựa trên các cấu hình đã hoạt động tốt trước đó.

---

<sup>87</sup>Transfer learning with neural automl, 2018

<sup>88</sup>Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl, 2021



**Hình 86:** **Trái:** Một bước thời gian đơn lẻ của bộ điều khiển AutoML đa nhiệm vụ hồi quy, trong đó một hành động đơn lẻ được thực hiện. Mã hóa nhiệm vụ được nối với mã hóa của hành động được lấy mẫu tại bước thời gian trước đó và chuyển vào bộ điều khiển RNN. Tất cả các tham số, ngoại trừ mã hóa nhiệm vụ, được chia sẻ qua các nhiệm vụ. **Phải:** Độ tương đồng cosine giữa các mã hóa nhiệm vụ học được bởi mô hình AutoML dựa trên mạng nơ-ron đa nhiệm vụ.

Lian cùng đồng nghiệp<sup>89</sup> và Elsken cùng đồng nghiệp<sup>90</sup> tập trung vào **học nhanh** (few-shot learning): vẫn đề học một nhiệm vụ mới với chỉ một vài điểm dữ liệu cho việc đào tạo.

Các tác giả mở rộng các phương pháp học siêu dựa trên gradient, phương pháp tiếp cận học siêu theo mô hình bất khả tri (model-agnostic) như MAML<sup>91</sup> và REPTILE<sup>92</sup> không chỉ để học siêu một tập trọng số ban đầu cho một kiến trúc mạng nơ-ron cố định, mà còn tìm ra kiến trúc tối ưu bằng cách kết hợp một phương pháp khả vi như DARTS vào thuật toán học siêu.

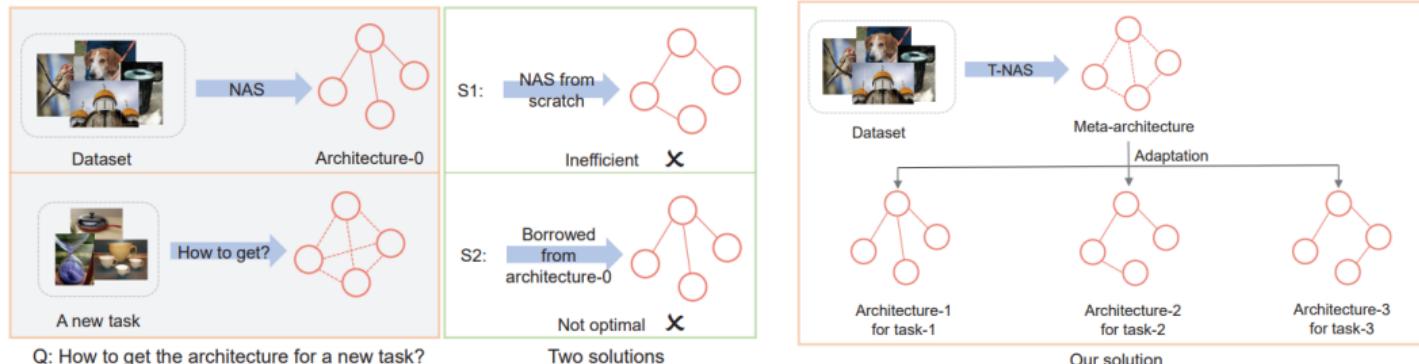
---

<sup>89</sup>Towards fast adaptation of neural architectures with meta learning, 2020

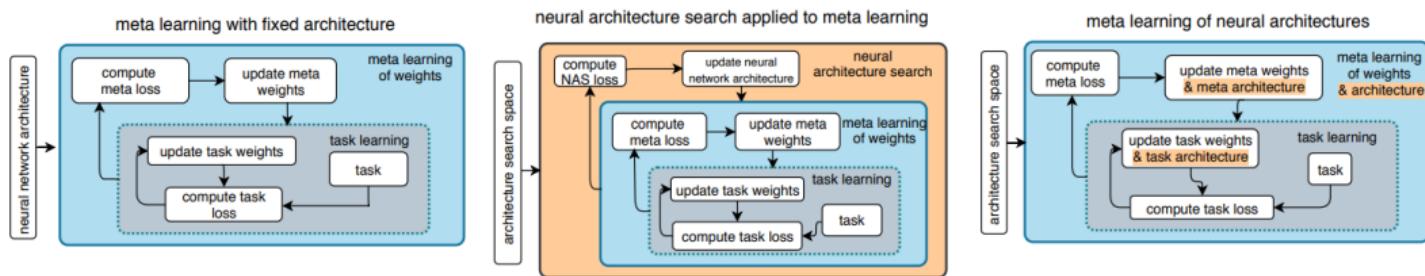
<sup>90</sup>Meta-learning of neural architectures for few-shot learning, 2020

<sup>91</sup>Model-agnostic meta-learning for fast adaptation of deep networks, 2017

<sup>92</sup>On first-order meta-learning algorithms, 2018



**Hình 87:** **Bên trái:** làm thế nào để tìm kiến trúc mạng khi có một nhiệm vụ mới? **Ở giữa:** hai giải pháp đơn giản nhưng không hiệu quả hoặc không tối ưu. **Bên phải:** phương pháp T-NAS để có được một siêu kiến trúc (meta-architecture), có khả năng thích ứng dễ dàng và nhanh chóng với các nhiệm vụ khác nhau.



**Hình 88:** **Bên trái:** học siêu dựa trên gradient với kiến trúc cố định như MAML hoặc REPTILE. **Ở giữa:** áp dụng NAS cho học siêu như AutoMeta. **Bên phải:** Đề xuất kết hợp học siêu về kiến trúc và trọng số với METANAS. Vì kiến trúc được thích nghi trong quá trình học nhiệm vụ, phương pháp được đề xuất có thể học các kiến trúc cụ thể cho từng nhiệm vụ.

Công trình của Lee và đồng nghiệp<sup>93</sup> sử dụng các KGTK điển hình.

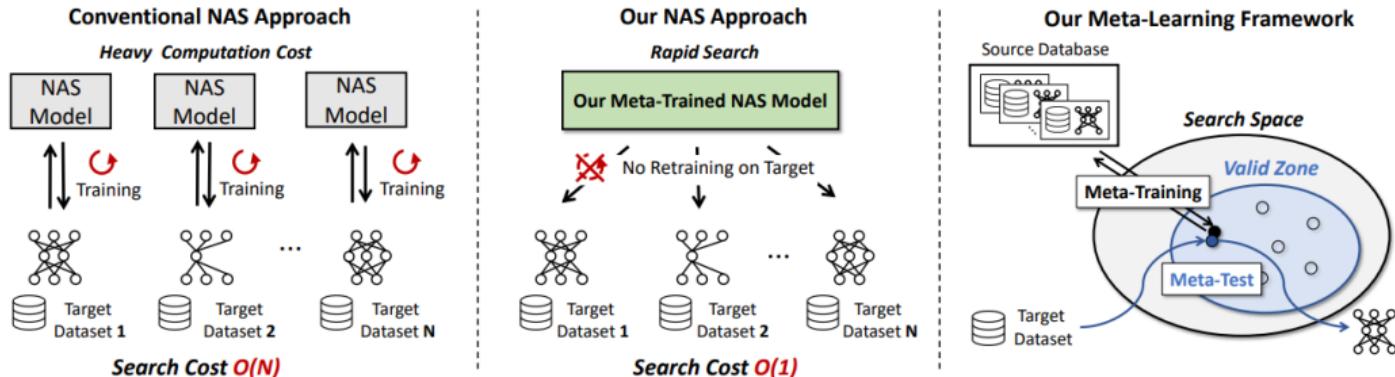
- Đề xuất một bộ mã hóa tập mới để cải thiện so với các **tập sâu**<sup>94</sup> và **bộ biến đổi tập**<sup>95</sup> (set transformers).
- Một bộ giải mã dựa trên mạng nơ-ron đồ thị được sử dụng để tạo ra các kiến trúc nơ-ron dựa trên mã hóa tập. Một mạng nơ-ron đồ thị cũng được sử dụng để mã hóa các kiến trúc được tạo ra.
- Mã hóa kiến trúc kết hợp với mã hóa tập sau đó được sử dụng để học siêu một mô hình thay thế nhằm dự đoán hiệu suất của cặp kiến trúc, tập dữ liệu.

---

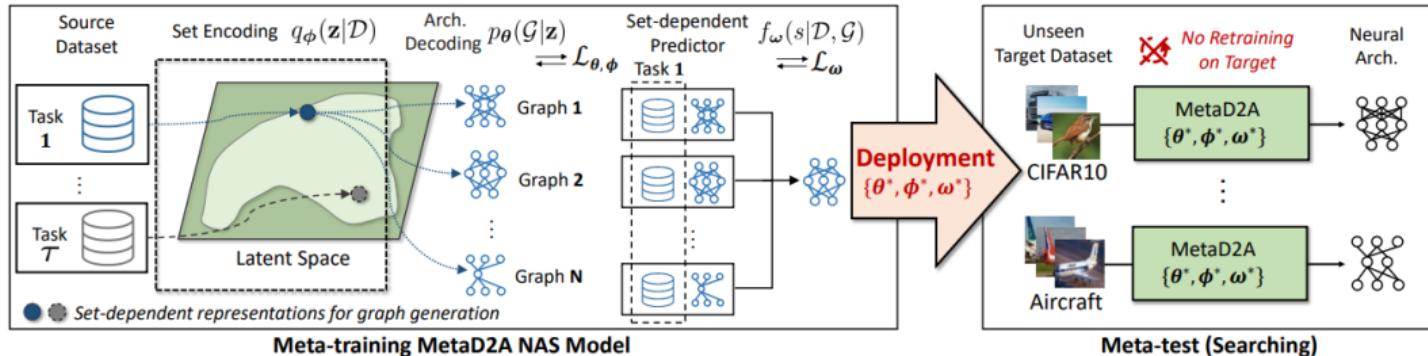
<sup>93</sup>Rapid neural architecture search by learning to generate graphs from datasets, 2021

<sup>94</sup>Deep sets, 2017

<sup>95</sup>Set transformer: A framework for attention-based permutation-invariant neural networks, 2019

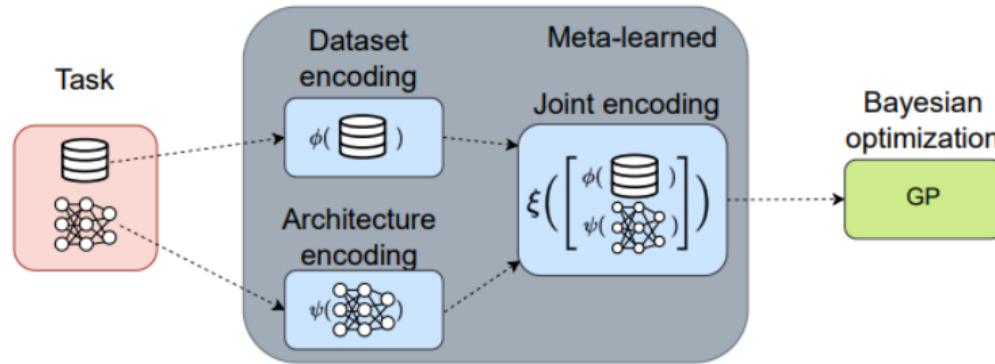


**Hình 89:** Bên trái: Hầu hết các phương pháp NAS thông thường cần huấn luyện lặp đi lặp lại mô hình NAS trên mỗi tập dữ liệu mục tiêu cụ thể, điều này dẫn đến tổng thời gian tìm kiếm lớn trên nhiều tập dữ liệu. Ở giữa: Lee và đồng nghiệp đề xuất một khung NAS mới mẻ có khả năng tổng quát hóa đối với bất kỳ tập dữ liệu mục tiêu nào để tạo ra kiến trúc nơ-ron chuyên biệt mà không cần huấn luyện thêm mô hình NAS sau chỉ sau quá trình học siêu trên cơ sở dữ liệu nguồn. Do đó, phương pháp của này giảm chi phí tìm kiếm để huấn luyện mô hình NAS trên nhiều tập dữ liệu từ  $O(N)$  xuống  $O(1)$ . Bên phải: Đối với tập dữ liệu mục tiêu chưa thấy trước, sử dụng kiến thức meta được ghi là các biểu diễn kiến trúc phụ thuộc vào tập để tạo ra kiến trúc.



**Hình 90:** Tổng quan về **MetaD2A**. Bộ tạo sinh đề xuất với  $\theta$  và  $\phi$  học meta để tạo ra các biểu diễn đồ thị phụ thuộc vào tập trên các nhiệm vụ học meta, trong đó mỗi nhiệm vụ chứa một tập hợp con của ImageNet-1K và kiến trúc chất lượng cao cho tập con đó. Bộ dự đoán đề xuất với  $\omega$  học meta để dự đoán hiệu suất, xem xét cả tập dữ liệu cũng như đồ thị. Trong giai đoạn học meta thử nghiệm (tìm kiếm), MetaD2A đã học tổng quát để tạo ra kiến trúc nơ-ron chuyên biệt cho các tập dữ liệu mục tiêu mới mà không cần huấn luyện thêm mô hình NAS.

Shala cùng đồng nghiệp<sup>96</sup> mở rộng công việc của Lee bằng cách sử dụng mã hóa tập dữ liệu và kiến trúc trong một khuôn khổ tối ưu hóa Bayesian, dẫn đến một bộ dự đoán ước lượng xác suất.



**Hình 91:** Minh họa về TNAS. Sử dụng một mạng nơ-ron đồ thị (GNN)  $\psi$  để mã hóa kiến trúc, một transformer  $\varphi$  để mã hóa một tập dữ liệu, và một mạng nơ-ron đa tầng (MLP)  $\xi$  để kết hợp các mã hóa này lại. Mã hóa chung này sau đó được đưa vào một mô hình ước lượng (GP) được sử dụng trong tối ưu hóa Bayesian (BO). Tất cả các mã hóa đều được học trong quá trình học meta.

<sup>96</sup>Transfer NAS with metalearned bayesian surrogates, 2022

## Kế thừa trọng số & Cấu xạ mạng

**Kế thừa trọng số** (weight inheritance) với ý tưởng: tái sử dụng các trọng số của các kiến trúc đã được đào tạo trên các kiến trúc tương tự chưa được đào tạo. Ví dụ, Real và đồng nghiệp<sup>97</sup> đề xuất sao chép các trọng số của tất cả các lớp chưa bị ảnh hưởng bởi các đột biến đã áp dụng từ kiến trúc cha sang kiến trúc con.

Ý tưởng này được mở rộng thông qua khái niệm về **cấu xạ mạng** (network morphisms):

- Cấu xạ mạng là các toán tử hoạt động trong không gian kiến trúc mạng nơ-ron.
- Chúng thay đổi kiến trúc của một mạng nơ-ron mà không thay đổi hàm (function) mà chúng đại diện, tức là: với một đầu vào tùy ý, đầu ra vẫn giống nhau cho kiến trúc gốc và kiến trúc đã được thay đổi bằng cấu xạ mạng.

---

<sup>97</sup> Large-scale evolution of image classifiers, 2017

- Với  $\mathcal{N}(\mathcal{X})$  là không gian các mạng nơ-ron mà mỗi phần tử  $N \in \mathcal{N}(\mathcal{X})$  ứng với một mạng nơ-ron hay ánh xạ từ  $\mathcal{X} \subseteq \mathbb{R}^n$  đến không gian nào đó (chẳng hạn ánh xạ từ tensor của hình ảnh đến tập các nhãn).
- Một toán tử mạng  $T : \mathcal{N}(\mathcal{X}) \times \mathbb{R}^i \rightarrow \mathcal{N}(\mathcal{X}) \times \mathbb{R}^j$  là phép biến đổi một mạng  $N^w \in \mathcal{N}(\mathcal{X})$  với tham số  $w \in \mathcal{R}^i$  thành một mạng  $(TN)^{\tilde{w}} \in \mathcal{N}(\mathcal{X})$  với tham số  $\tilde{w} \in \mathbb{R}^j$ .
- Cầu xạ mạng (NM) là toán tử mạng thỏa mãn:  $N^w(x) = (TN)^{\tilde{w}}(x), \forall x \in \mathcal{X}$ .
- Cầu xạ mạng xấp xỉ (ANM) là toán tử mạng thỏa mãn:  $N^w(x) \approx (TN)^{\tilde{w}}(x), \forall x \in \mathcal{X}$ .

Cấu xạ mạng cho phép thay đổi kiến trúc của một mạng mà không làm mất mát thông tin học được từ mạng gốc. Điều này có nghĩa là khi thực hiện các thay đổi trên kiến trúc, hiệu năng ban đầu của mạng được bảo toàn, không cần phải huấn luyện lại từ đầu.

Cấu xạ mạng đã được áp dụng trong một số công trình:

- Thuật toán tiến hóa.<sup>98,99,100</sup>
- Học tăng cường.<sup>101</sup>
- Tối ưu hóa Bayes.<sup>102</sup>
- Kỹ thuật một lần.<sup>103</sup>

---

<sup>98</sup>Efficient multi-objective neural architecture search via lamarckian evolution, 2019

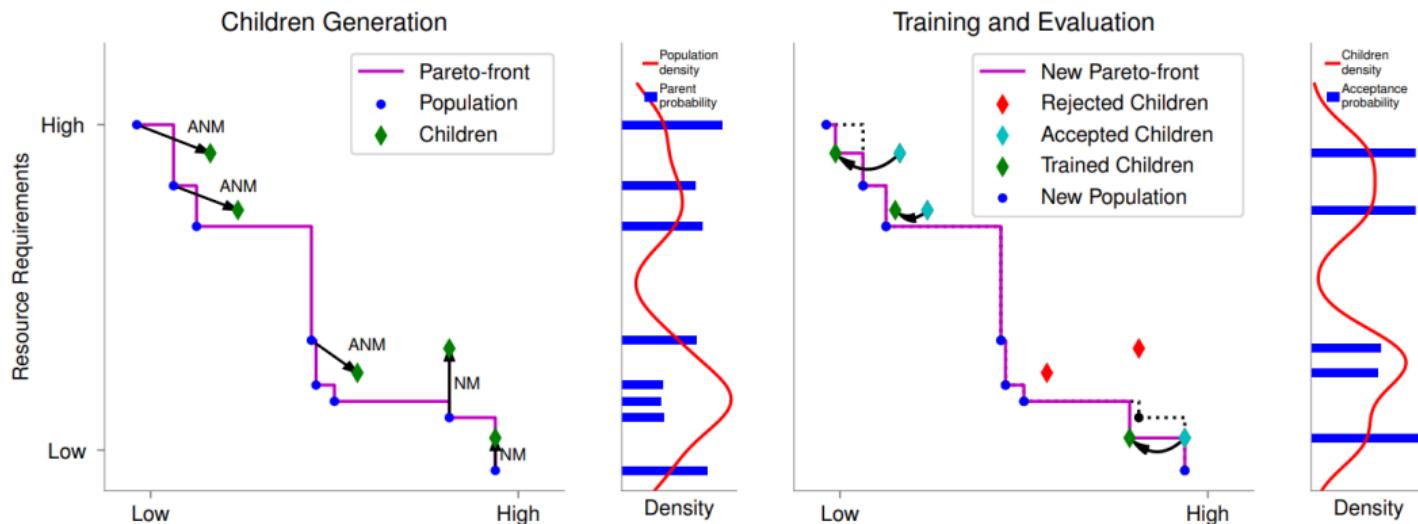
<sup>99</sup>Automated design of error-resilient and hardware-efficient deep neural networks, 2020

<sup>100</sup>Deep learning architecture search by neuro-cell-based evolution with function-preserving mutations, 2019

<sup>101</sup>Efficient architecture search by network transformation, 2018

<sup>102</sup>An efficient neural architecture search system, 2019

<sup>103</sup>Fast neural network adaptation via parameter remapping and architecture search, 2020



**Hình 92:** Hình minh họa về **LEMONADE** (Lamarckian Evolutionary algorithm for Multi-Objective Neural Architecture DEsign). **Bên trái:** LEMONADE duy trì một quần thể của các mạng đã được đào tạo, tạo thành một biên Pareto trong không gian đa mục tiêu. Cha mẹ được lựa chọn từ quần thể tỷ lệ nghịch với mật độ của họ. Các con cái được tạo ra bằng các toán tử đột biến có sự thừa kế Lamarckian được thực hiện bằng NM và ANM. Các toán tử NM tạo ra con cái với cùng lỗi ban đầu như cha mẹ của chúng. Các con cái được tạo ra bằng các toán tử ANM có thể gây ra sự tăng lỗi (nhỏ) so với cha mẹ của chúng. **Bên phải:** Chỉ một phần con cái được tạo ra được chấp nhận để đào tạo. Sau khi đào tạo, hiệu suất của con cái được đánh giá và quần thể được cập nhật để tạo thành biên Pareto.

## Các tiêu chuẩn đánh giá

- Trong những ngày đầu của nghiên cứu về NAS, các tiêu chí phổ biến nhất là **độ chính xác kiểm tra cuối cùng** trên CIFAR-10 và ImageNet. Điều này gây ra sự không nhất quán về KGTK và quy trình đào tạo giữa các bài báo, đồng thời làm tăng chi phí tính toán.
- Gần đây, các tiêu chuẩn đánh giá NAS có thể truy vấn đã giúp lĩnh vực giảm bớt tính toán khi phát triển các kỹ thuật NAS và thực hiện so sánh công bằng, có ý nghĩa thống kê giữa các phương pháp.

- Một *tiêu chuẩn đánh giá NAS*<sup>104</sup> được định nghĩa là một tập dữ liệu với phân chia huấn luyện - kiểm tra cố định, một KGTK và một quy trình đánh giá cố định để đào tạo các kiến trúc.
- Một *tiêu chuẩn đánh giá NAS dạng bảng* là một tiêu chuẩn đánh giá cung cấp đánh giá được tính toán trước cho tất cả các kiến trúc có thể trong KGTK.
- Một *tiêu chuẩn đánh giá NAS dựa trên mô hình thay thế* là một tiêu chuẩn đánh giá NAS kèm theo một mô hình thay thế có thể được sử dụng để dự đoán hiệu suất của bất kỳ kiến trúc nào trong KGTK.
- Một *tiêu chuẩn đánh giá NAS có thể truy vấn* nếu nó là một tiêu chuẩn dạng bảng hoặc dựa trên mô hình thay thế.

---

<sup>104</sup> Best practices for scientific research on neural architecture search, 2020

Tiêu chuẩn	Mô tả
NAS-Bench-101 <sup>105</sup>	Gồm 423,624 kiến trúc với độ chính xác đã tính trước trên CIFAR-10.
NAS-Bench1Shot1 <sup>106</sup>	Mô phỏng thuật toán one-shot trên không gian NAS-Bench-101 với số lượng nút cố định.
NAS-Bench-201 <sup>107</sup>	Bao gồm 6466 kiến trúc với độ chính xác đã tính trước trên CIFAR-10, CIFAR-100, và ImageNet-16-120.
NATS-Bench <sup>108</sup>	Mở rộng của NAS-Bench-201 bao gồm không gian tìm kiếm vĩ mô.
HW-NAS-Bench-201 <sup>109</sup>	Cung cấp chi phí phần cứng đo lường hoặc ước lượng cho kiến trúc trên sáu thiết bị.

Bảng 2: Tiêu chuẩn đánh giá NAS dạng bảng.

<sup>105</sup>Nas-bench-101: Towards reproducible neural architecture search, 2019<sup>106</sup>Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search, 2020<sup>107</sup>Nas-bench-201: Extending the scope of reproducible neural architecture search, 2020<sup>108</sup>Nats-bench: Benchmarking nas algorithms for architecture topology and size, 2021<sup>109</sup>HW-nas-bench: Hardware-aware neural architecture search benchmark, 2021

Tiêu chuẩn	Mô tả
Surr-NAS-Bench-DARTS <sup>110</sup> (NAS-Bench-301)	Bảng đánh giá NAS thay thế đầu tiên, được tạo ra bằng cách huấn luyện 60,000 kiến trúc từ không gian tìm kiếm DARTS trên CIFAR-10 và sau đó huấn luyện một mô hình thay thế.
NAS-Bench-x11 <sup>111</sup>	Công trình phát triển một kỹ thuật để dự đoán đường cong học tập đầy đủ, cho phép truy vấn độ chính xác xác nhận ở các epoch tùy ý, điều này là cần thiết để mô phỏng thuật toán NAS đa tầng.

Bảng 3: Tiêu chuẩn đánh giá NAS dựa trên mô hình thay thế.

<sup>110</sup>Nas-bench-301 and the case for surrogate benchmarks for neural architecture search, 2020

<sup>111</sup>Nas-bench-x11 and the power of learning curves, 2021

Tiêu chuẩn	Mô tả
TransNAS-Bench-101 <sup>112</sup>	Đánh giá dạng bảng với 7 nhiệm vụ thị giác máy tính từ bộ dữ liệu Taskonomy.
NAS-Bench-NLP <sup>113</sup>	Đánh giá dạng bảng với không gian tìm kiếm lấy cảm hứng từ LSTM cho NLP.
NAS-Bench-ASR <sup>114</sup>	Đánh giá dạng bảng cho nhận dạng tiếng nói tự động.
NAS-Bench-360 <sup>115</sup>	Bộ kiểm thử cung cấp bảng đánh giá NAS trên 10 vấn đề đa dạng như kiểm soát cánh tay giả, giải phương trình vi phân, gấp protein và hình ảnh thiên văn.
NAS-Bench-Suite <sup>116</sup>	Bộ kiểm thử kết hợp hầu hết các bảng đánh giá NAS truy vấn được hiện nay, tổng cộng 28 nhiệm vụ, vào một giao diện đồng nhất.
NAS-Bench-Suite-Zero <sup>117</sup>	Mở rộng của NAS-Bench-Suite cung cấp các giá trị thay thế không chi phí đã tính trước trên tất cả các nhiệm vụ.

Bảng 4: Một số tiêu chuẩn đánh giá NAS khác.

<sup>112</sup> Transnas-bench-101: Improving transferability and generalizability of cross-task neural architecture search, 2021

<sup>113</sup> Nas-bench-nlp: neural architecture search benchmark for natural language processing, 2022

<sup>114</sup> Nas-bench-asr: Reproducible neural architecture search for speech recognition, 2021

<sup>115</sup> NAS-bench-360: Benchmarking neural architecture search on diverse tasks, 2022

<sup>116</sup> Nas-bench-suite: Nas evaluation is (now) surprisingly easy, 2022

<sup>117</sup> Nas-bench-suite-zero: Accelerating research on zero cost proxies, 2022

## Úng dụng

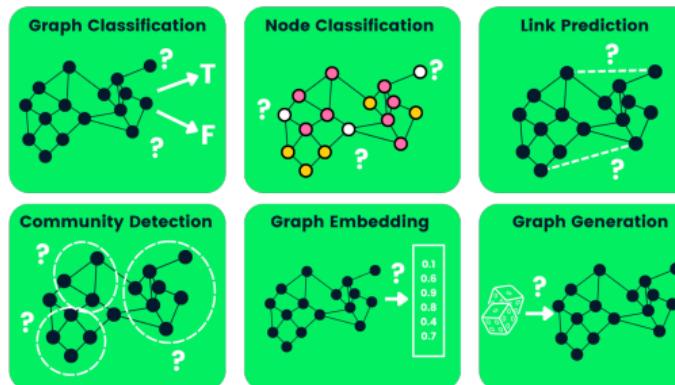
Đa số nghiên cứu NAS tập trung vào phân loại hình ảnh, nhưng cũng có nhiều câu chuyện thành công với việc áp dụng NAS cho các cài đặt ít nổi tiếng hơn, bao gồm:

- Mạng nơ-ron đồ thị (Graph Neural Networks).
- Mạng đối nghịch tạo sinh (Generative Adversarial Network).
- Nhiệm vụ dự đoán dày đặc (Dense Prediction Tasks).
- Bộ biến hình (Transformers).

# Mạng thần kinh đồ thị

**Mạng thần kinh đồ thị** (Graph Neural Networks - GNNs) được thiết kế để xử lý dữ liệu được biểu diễn bằng đồ thị. Việc sử dụng NAS để thiết kế GNN đặt ra các vấn đề đặc biệt:

- KGTK cho GNN phức tạp hơn so với KGTK tích chập thông thường.
- Cả NAS và GNNs đều nổi tiếng với khả năng tính toán lớn.



Hình 93: Một số ứng dụng của mạng thần kinh đồ thị.

Zhou và đồng nghiệp<sup>118</sup> khởi xướng một chuỗi nghiên cứu áp dụng NAS cho GNN bằng cách *định nghĩa một không gian tìm kiếm mới* với các *toán tử đặc biệt* cho GNN và sau đó sử dụng chiến lược học tăng cường.

Nghiên cứu theo chân tiếp theo thiết kế không gian tìm kiếm tương tự<sup>119,120</sup> với các đặc điểm chuyên biệt như siêu-đường (meta-paths),<sup>121</sup> đặc trưng cạnh<sup>122</sup> hoặc các toán tử lấy mẫu nhanh.<sup>123</sup>

Nhìn chung, sự khác biệt chính giữa NAS cho GNNs và cài đặt NAS tiêu biểu hơn nằm ở cấu trúc của không gian tìm kiếm. Các chiến lược tìm kiếm chính bao gồm: học tăng cường, phương pháp một lần, và thuật toán tiến hóa.

---

<sup>118</sup>Dha: End-to-end joint optimization of data augmentation policy, hyperparameter and architecture, 2019

<sup>119</sup>Graph neural architecture search, 2020

<sup>120</sup>Automated machine learning on graphs: A survey, 2021

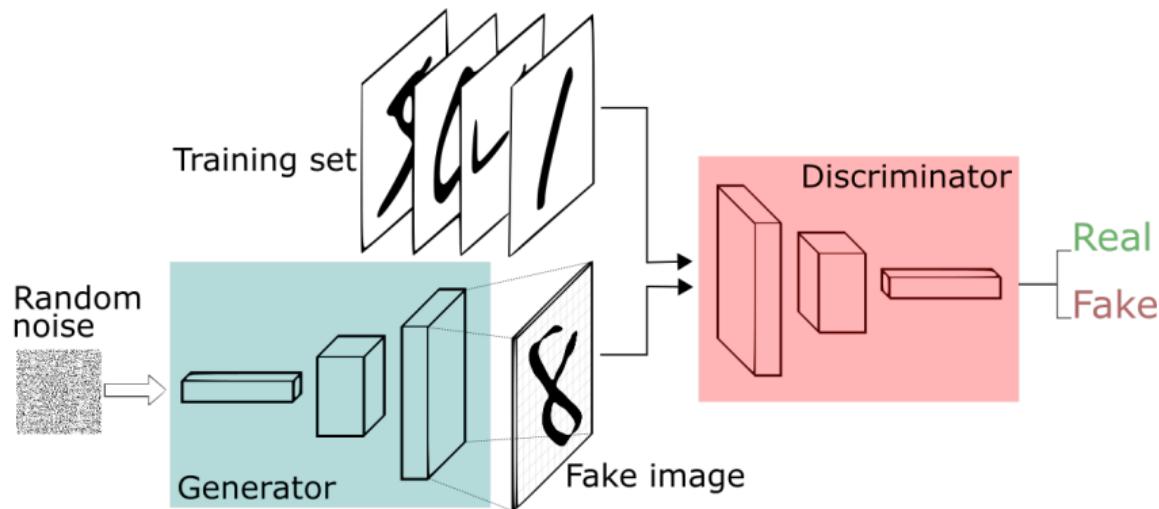
<sup>121</sup>Diffmg: Differentiable meta graph search for heterogeneous graph neural networks, 2021

<sup>122</sup>Graph neural network architecture search for molecular property prediction, 2020

<sup>123</sup>Graph neural architecture search, 2020

## Mạng đối nghịch tạo sinh

**Mạng đối nghịch tạo sinh** (Generative Adversarial Network - GANs) là một lựa chọn phổ biến cho mô hình tạo sinh trong các nhiệm vụ như thị giác máy tính. GANs sử dụng 2 mạng riêng biệt đào tạo cùng nhau: một bộ tạo sinh (generator) và một bộ so sánh (discriminator).



Hình 94: Minh họa mạng đối nghịch tạo sinh.

Các công trình khác nhau đã đạt được hiệu suất cải thiện thông qua NAS bằng cách:

- Tìm kiếm chỉ kiến trúc của bộ tạo sinh với một bộ so sánh cố định.<sup>124</sup>
- Tìm kiếm bộ tạo sinh với bộ so sánh ngày càng phát triển được xác định trước.<sup>125</sup>
- Tìm kiếm cả 2 kiến trúc của bộ tạo sinh và bộ so sánh đồng thời.<sup>126</sup>

Lựa chọn không gian tìm kiếm phổ biến nhất là không gian tìm kiếm dựa trên tế bào. Tế bào cho bộ tạo sinh bao gồm một tế bào tích chập tiêu chuẩn, kèm theo các toán tử nâng phân giải (upsampling operations) đa dạng.<sup>127,128</sup>

Các kỹ thuật tìm kiếm giống như các kỹ thuật được sử dụng cho NAS tiêu biểu: học tăng cường, mô hình NAS một lần, và thuật toán tiến hóa, với điểm số dựa trên Inception Score (IS) hoặc Fréchet Inception Distance (FID).

---

<sup>124</sup>Degas: differentiable efficient generator search, 2020

<sup>125</sup>Autogan-distiller: searching to compress generative adversarial networks, 2020

<sup>126</sup>Autogan: Neural architecture search for generative adversarial networks, 2019

<sup>127</sup>Automating generative adversarial networks using neural architecture search: A review, 2021

<sup>128</sup>Off-policy reinforcement learning for efficient and effective gan architecture search, 2020

## Nhiệm vụ dự đoán dày đặc

Dự đoán dày đặc cho thị giác máy tính bao gồm nhiều nhiệm vụ phổ biến như:

- Phân đoạn ngữ nghĩa (semantic segmentation).
- Phát hiện đối tượng (object detection).
- Luồng quang học (optical flow).
- Ước lượng chênh lệch (disparity estimation).

Nó yêu cầu các kiến trúc phức tạp hơn so với các vấn đề phân loại hình ảnh tiêu biểu.

Ví dụ, các kiến trúc thường bao gồm một bộ giải mã, các mô-đun để tạo ra đặc trưng đa quy mô hoặc các đầu cụ thể cho nhiệm vụ ngoài mạng chính. Do đó, các thuật toán NAS đã được áp dụng để tìm kiếm các thành phần này, hoặc là độc lập hoặc cùng nhau hoặc bằng cách khám phá các mẫu thiết kế mới.

Các kỹ thuật NAS tiêu biểu được sử dụng: dựa trên gradient với DARTS,<sup>129,130</sup> học tăng cường<sup>131</sup> hoặc lấy cảm hứng từ ProxylessNAS và ENAS.<sup>132</sup>

- Các phương pháp cho nhiệm vụ dự đoán dày đặc thường xây dựng không gian tìm kiếm dựa trên các mạng phân loại hình ảnh tiên tiến, với các thành phần cụ thể cho kiến trúc dự đoán dày đặc đang hoạt động hiệu quả.
- Thường sử dụng các kiến trúc cột sống được đào tạo trước hoặc thậm chí là lưu trữ các đặc trưng được tạo ra bởi một kiến trúc cột sống để tăng tốc quá trình tìm kiếm kiến trúc.
- Các phương pháp đôi khi cũng sử dụng nhiều giai đoạn tìm kiếm, với mục tiêu đầu tiên là loại bỏ các kiến trúc hoặc phần của không gian tìm kiếm hoạt động kém và từ từ cải thiện các kiến trúc còn lại

---

<sup>129</sup>Autodispnet: Improving disparity estimation with automl, 2019

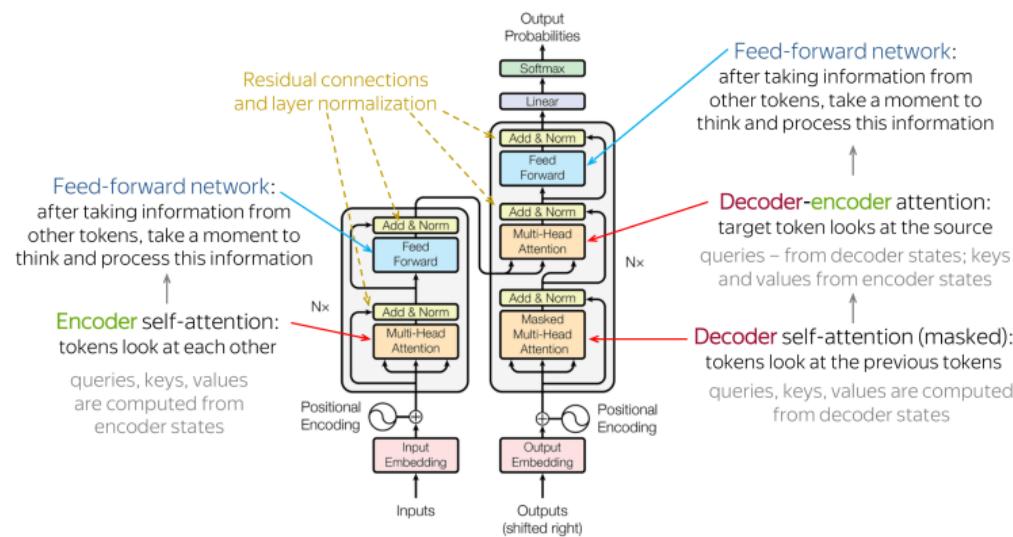
<sup>130</sup>Auto-fpn: Automatic network architecture adaptation for object detection beyond classification, 2019

<sup>131</sup>Nas-fpn: Learning scalable feature pyramid architecture for object detection, 2019

<sup>132</sup>Can weight sharing outperform random architecture search? an investigation with tunas, 2020

# Bộ biến hình

**Bộ biến hình** (Transformers) được sử dụng để giải quyết vấn đề của các chuỗi dài mà RNNs gặp khó khăn trong việc mô hình hóa, bằng cách sử dụng cơ chế *tự chú ý* (self-attention) và *chú ý chéo* (cross-attention) sao cho biểu diễn của mỗi token trong một chuỗi đầu vào được tính từ trung bình có trọng số của biểu diễn của tất cả các token khác.



Hình 95: Kiến trúc bộ biến hình.

Thiết kế transformer cốt lõi được giới thiệu cho dịch máy, nhưng nó đã được sử dụng phổ biến trong mô hình ngôn ngữ nhân quả (causal language modeling), mô hình ngôn ngữ che mặt (masked language modeling), và gần đây, trong thị giác máy tính.

Kể từ khi ra mắt, đã có nhiều nỗ lực để cải thiện transformers thông qua NAS. Các chiến lược tìm kiếm phổ biến nhất cho transformers là tiến hóa<sup>133,134</sup> hoặc mô hình một lần.<sup>135,136,137,138</sup>

Tổng thể, lĩnh vực NAS cho transformers chưa hội tụ thành một loại không gian tìm kiếm tốt nhất. Có 4 loại transformers áp dụng NAS: chỉ có bộ giải mã (decoder-only), chỉ có bộ mã hóa (encoder-only), có cả bộ mã hóa và giải mã (encoder-decoder), và vision transformers.

---

<sup>133</sup>Autoformer: Searching transformers for visual recognition, 2021

<sup>134</sup>The evolved transformer, 2019

<sup>135</sup>Hr-nas: Searching efficient high-resolution neural architectures with lightweight transformers, 2021

<sup>136</sup>Nasvit: Neural architecture search for efficient vision transformers with gradient conflict aware supernet training, 2021

<sup>137</sup>Bosnnas: Exploring hybrid cnn-transformers with block-wisely selfsupervised neural architecture search, 2021

<sup>138</sup>Vitas: Vision transformer architecture search, 2021

- Các kiến trúc chỉ có bộ giải mã, như dòng **kiến trúc GPT**, trực tiếp tiêu thụ dòng gợi ý văn bản đầu vào và xuất ra chuỗi các token văn bản có khả năng cao nhất để theo sau. Primer<sup>139</sup> là một thuật toán NAS sử dụng tìm kiếm tiền hóa trên một không gian tìm kiếm vĩ mô lớn chỉ có bộ giải mã.
- Kiến trúc chỉ có bộ mã hóa, như **BERT**, mã hóa văn bản đầu vào thành một biểu diễn có thể được sử dụng cho nhiều loại nhiệm vụ hạ lưu. Nhiều công trình<sup>140, 141</sup> cố gắng khám phá các phiên bản nén của BERT, trong đó người dùng chỉ định độ trễ mong muốn và nhiệm vụ. Phương pháp điển hình là huấn luyện một siêu mạng trên một nhiệm vụ tự giám sát tiêu biểu (mô hình ngôn ngữ phủ), sau đó có thể được sử dụng để khám phá các mô hình nén cho một nhiệm vụ ngôn ngữ cụ thể.

---

<sup>139</sup>Primer: Searching for efficient transformers for language modeling, 2021

<sup>140</sup>Nasbert, 2021

<sup>141</sup>Autotinybert: Automatic hyper-parameter optimization for efficient pre-trained language models, 2021

- Kiến trúc có cả bộ mã hóa và giải mã, như **T5**, được sử dụng trong các nhiệm vụ từ chuỗi đến chuỗi như dịch máy, trong đó ngôn ngữ nguồn được mã hóa thành một biểu diễn, sau đó giải mã thành ngôn ngữ đích. So và đồng nghiệp<sup>142</sup> sử dụng tìm kiếm tiến hóa kết hợp với một kỹ thuật mới để phân bổ động tài nguyên thêm cho các mô hình ứng viên hứa hẹn hơn, trong khi Zhao và đồng nghiệp<sup>143</sup> đề xuất một thuật toán dựa trên DARTS với một kỹ thuật mới về hiệu suất bộ nhớ trong quá trình lan truyền ngược.
- Một loạt lớn các thuật toán NAS đã được nghiên cứu cho không gian tìm kiếm vision transformer, với đa số sử dụng phương pháp một lần. AutoFormer<sup>144</sup> tìm kiếm qua các kiến trúc và siêu tham số vision transformer bằng cách sử dụng chiến lược một đường-một lần<sup>145</sup> (single-path-one-shot) và sau đó chạy tìm kiếm tiến hóa trên siêu mạng được huấn luyện. Một công trình theo sau, AutoFormerv2,<sup>146</sup> tự động hóa thiết kế của không gian tìm kiếm bằng cách tiến hóa từng bước các chiều tìm kiếm khác nhau.

---

<sup>142</sup>The evolved transformer, 2019

<sup>143</sup>Memoryefficient differentiable transformer architecture search, 2021

<sup>144</sup>Autoformer: Searching transformers for visual recognition, 2021

<sup>145</sup>Single path one-shot neural architecture search with uniform sampling, 2020

<sup>146</sup>Searching the search space of vision transformer, 2021

## Khát quát về Học sâu

### Giới thiệu các kiến trúc mạng

Kiến trúc mạng tích chập

Các kiến trúc mạng tích chập hiện đại

Kiến trúc mạng hồi quy

Các kiến trúc mạng hồi quy hiện đại

### Phát biểu bài toán

Khái quát lĩnh vực nghiên cứu

Bộ dữ liệu kiến trúc làm chuẩn

Ứng dụng và các thành tựu đạt được

### Các phương pháp tiếp cận

Các giải thuật phỏng tự nhiên

Các giải thuật dựa trên vi phân

Kỹ thuật học tăng cường

## GA: Giải thuật Di truyền

Xét mạng với số lượng lớp hạn chế, ta sẽ mô tả không gian các kiến trúc có thể có của nó.

- Mạng bao gồm tất cả  $S$  giai đoạn (stage).
- Giai đoạn thứ  $s = 1, 2, \dots, S$  có  $K_s$  nút là  $v_{s,k_s}$  với  $k_s = 1, 2, \dots, K_s$ .
- Mỗi nút là liên tiếp các phép toán: tích chập, chuẩn hóa theo lô<sup>147</sup> và kích hoạt ReLU.
- Liền sau mỗi giai đoạn là lớp gộp và kết thúc mạng là lớp kết nối đầy đủ.

---

<sup>147</sup>Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

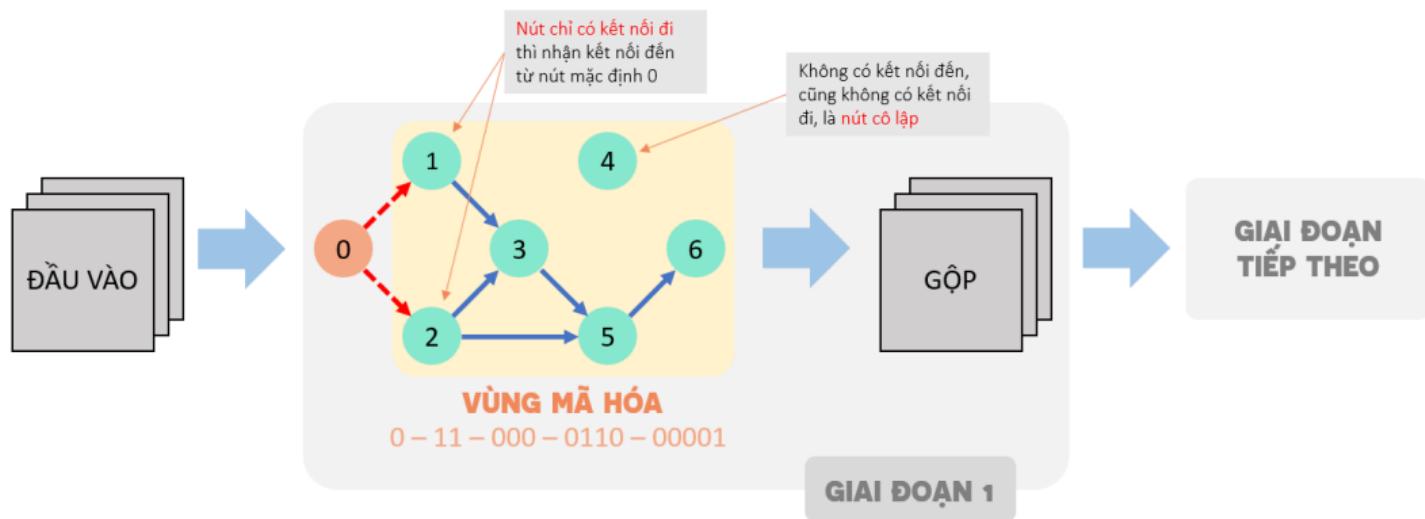
## Mã hóa nhị phân kiến trúc mạng

Trong giai đoạn  $s$ , ta sử dụng  $\frac{1}{2}K_s(K_s - 1)$  bit nhị phân để biểu diễn.

- 1 bit đầu tiên thể hiện kết nối  $v_{s,1}$  đến  $v_{s,2}$ .
- 2 bit tiếp theo thể hiện kết nối  $v_{s,1}, v_{s,2}$  đến  $v_{s,3}$ .
- ...
- $K_s - 1$  bit cuối cùng thể hiện kết nối  $v_{s,1}, v_{s,2}, \dots, v_{s,K_s-1}$  đến  $v_{s,K_s}$ .

Để đảm bảo kiến trúc hợp lệ, ta bổ sung nút thứ 0 bắt đầu mỗi giai đoạn:

- Nút 0 vẫn có cấu tạo gồm: tích chập, chuẩn hóa theo lô, ReLU.
- Đầu ra của nút 0 chuyển đến những nút mà chỉ có kết nối đi, chưa có kết nối đến.
- Nếu mã hóa là một dãy chỉ số 0, thì giai đoạn vẫn có ít nhất một nút là nút 0.



Hình 96: Minh họa giai đoạn trong kiến trúc mạng và mã hóa.

## Nguyên tắc khởi tạo

Quá trình di truyền bắt đầu với một quần thể gồm  $N$  cá thể ngẫu nhiên. Sau đó, thực hiện  $T$  vòng lặp (hay  $T$  thế hệ), mỗi thế hệ gồm 3 hoạt động: chọn lọc, lai ghép, đột biến. Hàm thích nghi của mỗi cá thể được đánh giá thông qua việc đào tạo từ đầu trên tập dữ liệu tham chiếu.

- Khởi tạo quần thể các mô hình  $\{\mathbb{M}_{0,n} : n = 1, 2, \dots, N\}$
- Mỗi mô hình là một chuỗi nhị phân  $L$  bit, tức:

$$\mathbb{M}_{0,n} : \mathbf{b}_{0,n} = [b_{0,n}^1, b_{0,n}^2, \dots, b_{0,n}^L] \in \{0, 1\}^L$$

- Mỗi bit trong cá thể được lấy mẫu độc lập từ một phân phối Bernoulli:

$$\ell = 1, 2, \dots, L : b_{0,n}^\ell \sim \mathcal{B}(L, 0.5)$$

Các chiến lược khởi tạo khác nhau không ảnh hưởng nhiều đến hiệu suất di truyền. Ngay cả khi khởi tạo ngây thơ (tất cả các cá thể đều là chuỗi 0), quá trình di truyền có thể khám phá ra các kiến trúc cạnh tranh nếu số thế hệ là đủ lớn.

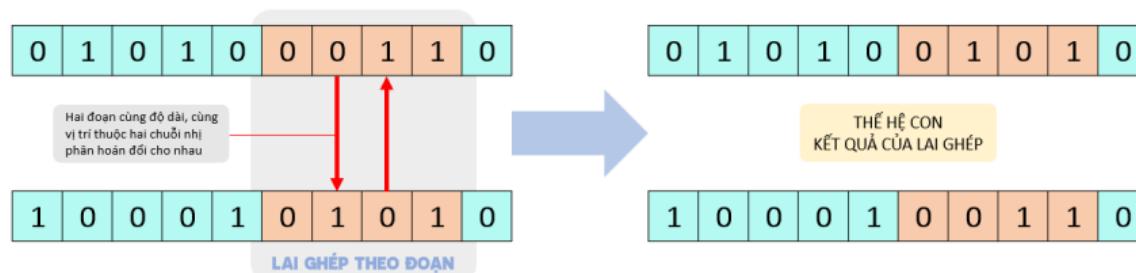
## Nguyên tắc chọn lọc

Các cá thể được chọn lọc để trở thành cha mẹ cho quá trình lai ghép tạo ra các cá thể con, dựa theo thể thức Roulette Wheel.

- Các cá thể con sau đó được huấn luyện trên tập dữ liệu để thu được giá trị thích nghi (độ chính xác).
- Bổ sung các cá thể con vào quần thể hiện tại và tiến hành chọn lọc ưu tú các cá thể tốt nhất cho quần thể tiếp theo.

# Toán tử lai ghép

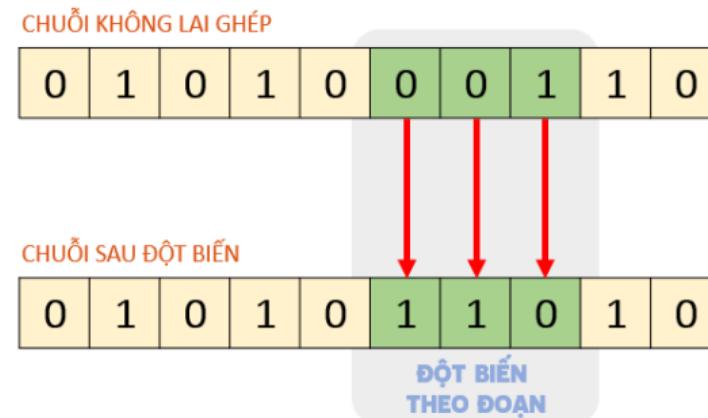
- Quá trình lai ghép thay đổi 2 cá thể cùng lúc.
- Thay vì xem xét từng bit một cách riêng lẻ, đơn vị cơ bản trong lai ghép là một đoạn (một số bit liên tiếp nhau). Tham số sử dụng là bán kính lai ghép  $r_C$ .
- Mỗi cặp đoạn được trao đổi với xác suất nhỏ  $p_C$ .



Hình 97: Minh họa phép lai ghép 2 điểm cắt với bán kính lai ghép 4

## Toán tử đột biến

- Với những cá thể trải qua lai ghép thì thực hiện đột biến.
- Quá trình này đảo bit của chuỗi với một xác suất nhỏ  $p_M$  và tác động lên một đoạn có độ dài phụ thuộc bán kính đột biến  $r_M$ .



Hình 98: Minh họa phép đột biến với bán kính đột biến 3

## Thuật toán 5: Thuật toán Di truyền

**Đầu vào:** Tập dữ liệu tham chiếu  $\mathcal{D}$ , số lượng thế hệ  $T$ , số lượng cá thể trong mỗi thế hệ  $N$ , xác suất đột biến và lai ghép là  $p_M$  và  $p_C$ , bán kính đột biến và lai ghép là  $r_M$  và  $r_C$ .

1. **Khởi tạo:** Bắt đầu với một tập các cá thể ngẫu nhiên và tính toán độ chính xác nhận diện của từng cá thể.
2. Trong mỗi vòng lặp  $t = 1, 2, \dots, T$  thực hiện các thao tác:
  - i **Chọn lọc:** Chọn cha mẹ tham gia lai ghép theo thể thức *Roulette Wheel*.
  - ii **Lai ghép:** Với cặp cha mẹ chọn được, thực hiện lai ghép với xác suất  $p_C$  và tham số  $r_C$ .
  - iii **Đột biến:** Với cá thể sau lai ghép, thực hiện đột biến với xác suất  $p_M$  và tham số  $r_M$ .
  - iv **Đánh giá:** Tính toán độ chính xác nhận dạng cho từng cá thể mới.
  - v **Đấu tranh:** Bổ sung thế hệ con vào quần thể và chọn ra những cá thể thích nghi tốt nhất cho thế hệ tiếp theo.

**Đầu ra:** Một tập hợp các cá thể trong thế hệ cuối cùng với độ chính xác trên  $\mathcal{D}$ .

## Thiết kế cho MNIST

Thuật toán Di truyền yêu cầu tài nguyên tính toán tương đối lớn. Để đơn giản, ta chỉ xem xét trường hợp hạn chế của nó:

- Có tất cả 10 thê hệ, mỗi thê hệ có 7 cá thể.
- Chọn tham số  $r_C = 4, r_M = 2$  để bảo toàn kiến trúc địa phương của mạng và không thay đổi mạng quá nhiều.
- Các xác suất  $p_C = 0.7, p_M = 0.2$  để sinh ra đa dạng kiến trúc cho quần thể.

Kiến trúc mạng được thiết kế:

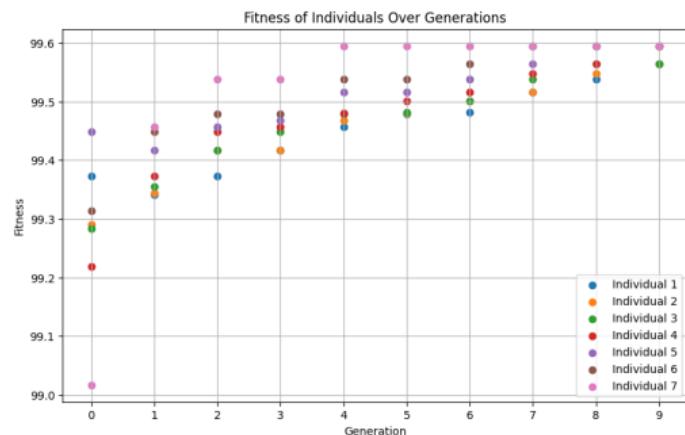
- Chỉ gồm đúng 1 giai đoạn, sau đó là gộp và kết nối đầy đủ.
- Có tất cả 7 nút trong mỗi giai đoạn, mã hóa bởi chuỗi nhị phân dài 21.
- Chỉ cập nhật tham số cho mạng bằng 5 vòng lặp với trình tối ưu ADAM,<sup>148</sup> tốc độ học mặc định 0.01.

---

<sup>148</sup>Adam: A Method for Stochastic Optimization

# Tìm kiếm kiến trúc MNIST

- Không khẳng định được kết quả cuối cùng là tối ưu toàn cục.
- Đa dạng trong quần thể bị giảm đi do áp dụng chọn lọc theo thứ hạng, một cá thể có khả năng xuất hiện nhiều lần gây ảnh hưởng đến quá trình di truyền.



Hình 99: Độ thích nghi của cá thể qua các thế hệ.

## Thiết kế cho CIFAR-10

Trường hợp được xét hạn chế với các điều kiện:

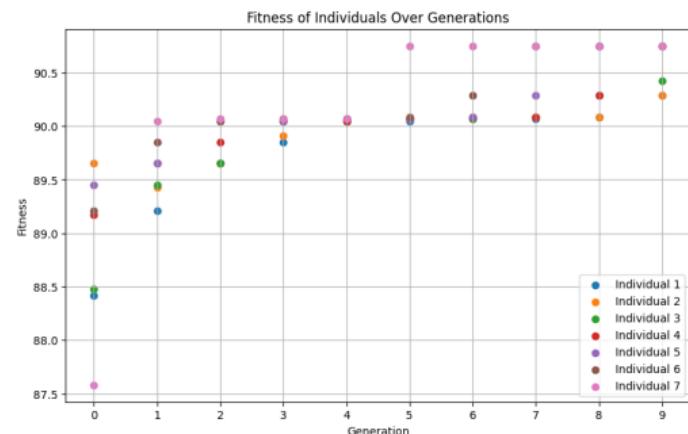
- Có tất cả 10 thế hệ, mỗi thế hệ có 7 cá thể.
- $r_C = 4, r_M = 2, p_C = 0.7, p_M = 0.2$  được chọn.

Kiến trúc mạng được thiết kế:

- Chỉ gồm 2 giai đoạn, kết thúc mỗi giai đoạn là gộp, kết thúc mạng là kết nối đầy đủ.
- Có 5 nút trong mỗi giai đoạn, mã hóa toàn mạng bởi chuỗi nhị phân dài 20.
- Cập nhật tham số cho mạng bằng 5 vòng lặp.

# Tìm kiếm kiến trúc CIFAR-10

- Thuật toán đạt được hiệu quả nhất định.
- Tình trạng giảm không gian tìm kiếm và mêt mát đa dạng nhiễm sắc thể.



Hình 100: Độ thích nghi của cá thể qua các thế hệ.

## AE: Giải thuật Tiên hóa lão hóa

Tiên hóa là một phương pháp đơn giản nhưng hiệu quả để khám phá ra các kiến trúc chất lượng cao, nhưng tồn tại các khuyết điểm như:

- **Tập trung quá mức vào các giải pháp tối ưu địa phương:** Các thuật toán này có xu hướng mắc kẹt ở các giải pháp tốt nhất địa phương mà không khám phá đủ rộng các phần khác của không gian tìm kiếm.
- **Đòi hỏi nhiều tài nguyên tính toán:** Do sự khám phá không gian lớn, các thuật toán này thường cần nhiều thời gian và tài nguyên máy tính để tìm ra giải pháp tối ưu.
- **Khó khăn trong việc điều chỉnh các tham số:** Cần phải cân nhắc kỹ lưỡng việc thiết lập các tham số như tỷ lệ đột biến, kích thước quần thể,...

## Điểm mới của thuật toán

Thuật toán **Tiên hóa lão hóa**<sup>149</sup> (Aging Evolution) cải tiến so với thuật toán tiên hóa thông thường ở các điểm sau:

- **Ưu tiên các giải pháp mới:** Bằng cách loại bỏ các giải pháp cũ hơn trong quần thể, thuật toán khuyến khích sự đa dạng và ngăn chặn sự tập trung quá mức vào các giải pháp tối ưu địa phương.
- **Khám phá hiệu quả hơn:** Sự lão hóa giúp đảm bảo rằng không gian tìm kiếm được khám phá một cách cân bằng hơn, không chỉ tập trung vào những gì đã biết.
- **Đáp ứng nhanh với môi trường đổi thay:** Do các thành viên mới liên tục được thêm vào, thuật toán có khả năng thích ứng tốt hơn với những thay đổi trong môi trường hoặc các yêu cầu mới.

---

<sup>149</sup>Regularized Evolution for Image Classifier Architecture Search

## Thuật toán 6: Thuật toán Tiền hóa lão hóa

1. Quần thể  $P \leftarrow$  hàng đợi rỗng. Lịch sử  $H \leftarrow \emptyset$ .
2. Khởi tạo  $P \leftarrow N$  kiến trúc ngẫu nhiên và đánh giá, thêm vào  $H$ .
3. Vòng lặp  $i = 1, 2, \dots, C$  thực hiện:
  - i Lấy mẫu ngẫu nhiên  $S$  cá thể trong  $P$ .
  - ii Chọn cha mẹ là cá thể tốt nhất trong đó.
  - iii Đột biến cha mẹ tạo ra cá thể con.
  - iv Đánh giá cá thể con.
  - v Thêm cá thể con vào  $H$  và *bên phải*  $P$ .
  - vi Loại bỏ cá thể cũ nhất *bên trái*  $P$ .

## GEA: Giải thuật Định hướng tiến hóa

- Cách tiếp cận đáng tin cậy nhất để có thông tin về không gian tìm kiếm trong khi tìm kiếm là đào tạo đầy đủ các kiến trúc được tạo ra và tối ưu hóa việc tìm kiếm dựa trên những cái hiệu suất nhất.
- Tuy nhiên, điều này **rất tốn kém**, và kết quả phụ thuộc rất nhiều vào các lược đồ đào tạo và cài đặt khởi tạo.
- Do đó, các ước lượng proxy không tốn kém (zero-cost proxies) trình bày một giải pháp hấp dẫn, nơi thông kê được rút ra từ các kiến trúc được tạo ra để ghi điểm cho chúng ở giai đoạn khởi tạo, do đó không cần huấn luyện.
- Các phương pháp này hiệu quả về thời gian và có khả năng thực hiện các mối tương quan tốt giữa điểm số và độ chính xác tương ứng khi các kiến trúc được huấn luyện.

## Khát quát thuật toán

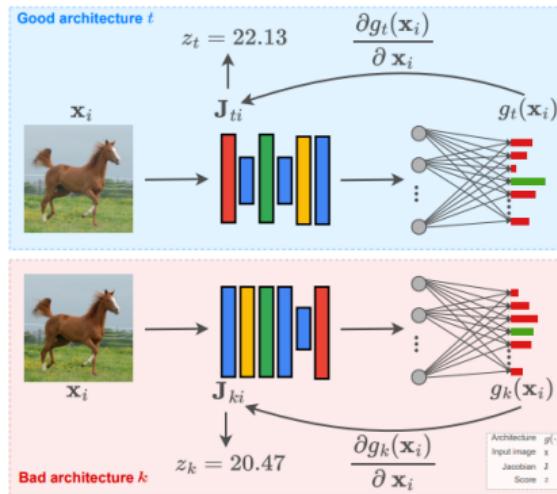
Nhóm tác giả đề xuất một phương pháp NAS tận dụng ước lượng proxy không tốn kém để hướng dẫn tìm kiếm một cách hiệu quả.

- Bằng cách sử dụng một chiến lược tiến hóa nơi các toán tử có thể bị đột biến và các kiến trúc mới hơn được ưu tiên, GEA buộc phải khai thác các kiến trúc hiệu suất nhất và khám phá không gian tìm kiếm bằng cách thực hiện các đột biến.
- Hơn nữa, các kiến trúc đều được đánh giá ở giai đoạn khởi tạo bằng một ước lượng proxy không tốn kém và chỉ có kiến trúc có điểm số cao nhất được huấn luyện và giữ lại cho thế hệ tiếp theo.

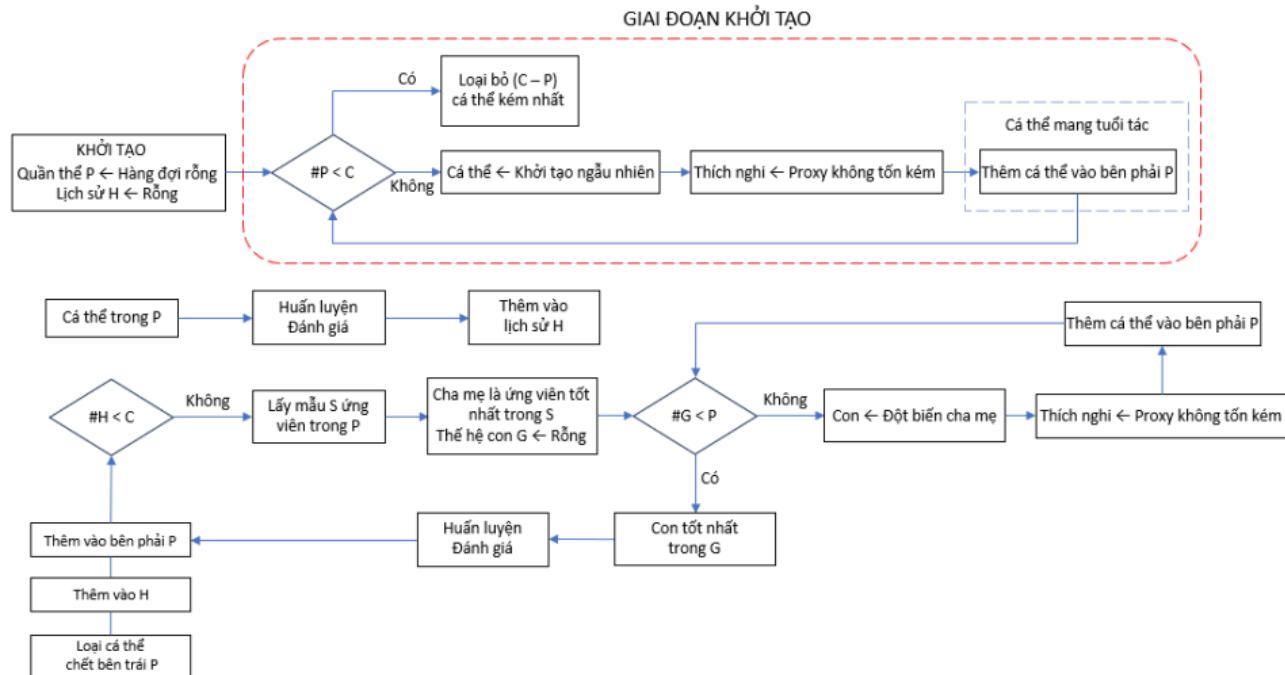
Làm như vậy, thuật toán **Định hướng tiến hóa**<sup>150</sup> (Guided Evolutionary Algorithm - GEA) có khả năng liên tục rút ra kiến thức về không gian tìm kiếm mà không làm ảnh hưởng đến việc tìm kiếm, dẫn đến kết quả hàng đầu trong không gian tìm kiếm NAS-Bench-101, NAS-Bench-201 và TransNAS-Bench101.

---

<sup>150</sup> Guided Evolutionary Neural Architecture Search With Efficient Performance Estimation

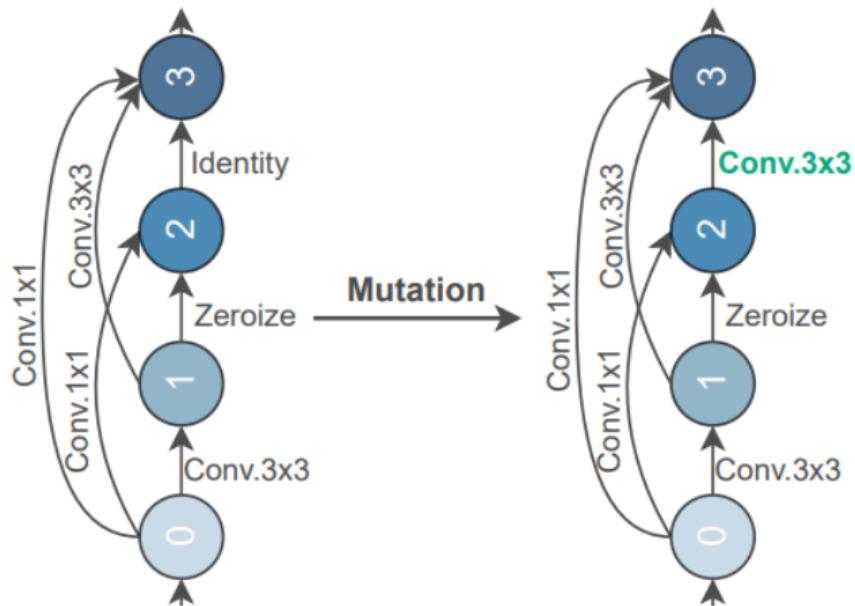


**Hình 101:** Đánh giá 2 kiến trúc sử dụng cùng đầu vào. Các kiến trúc được tạo ra ở mỗi thế hệ được xếp hạng dựa trên một điểm số tương quan với hiệu suất cuối cùng của chúng, điều này quyết định kiến trúc nào được chọn để trở thành một phần của quần thể.

**Hình 102:** Lưu đồ Thuật toán Định hướng Tiền hóa

## Thuật toán 7: Thuật toán Định hướng tiến hóa

1. Quần thể  $P \leftarrow$  hàng đợi rỗng. Lịch sử  $H \leftarrow \emptyset$ .
2. Lấy ngẫu nhiên đồng đều  $C$  kiến trúc.
3. Ghi điểm cho các kiến trúc bằng proxy không tốn kém.
4. Chọn  $P \leftarrow N$  kiến trúc đạt điểm cao nhất.
5. Huấn luyện các kiến trúc thu được độ thích nghi  $f$ .
6. Vòng lặp  $i = 1, 2, \dots, T$  thực hiện:
  - i Lấy mẫu ngẫu nhiên  $S$  cá thể từ quần thể.
  - ii Cha mẹ  $\leftarrow$  cá thể có độ thích nghi cao nhất.
  - iii  $P$  cá thể con  $\leftarrow$  đột biến cha mẹ.
  - iv Ghi điểm  $P$  cá thể con bằng proxy không tốn kém.
  - v Chọn cá thể con đạt điểm cao nhất.
  - vi Huấn luyện cá thể con.
  - vii Thêm cá thể con vào  $H$  và bên phải  $P$ .
  - viii Loại bỏ cá thể cũ nhất bên trái  $P$ .



Hình 103: Minh họa đột biến một toán tử trong một khối sử dụng NAS-Bench-201

- Về bản chất, các giá trị  $N$  cao hơn đại diện cho một mức độ **khám phá** cao hơn của không gian tìm kiếm.
- Việc tạo ra và đánh giá  $N$  kiến trúc tăng cường phương pháp tìm kiếm để tìm hướng tốt nhất cho sự tiến hóa của cha mẹ qua không gian tìm kiếm.
- Điều này cho phép phương pháp được hướng dẫn qua một không gian phức tạp mà không làm ảnh hưởng đến thời gian cần thiết để thực hiện sự tiến hóa hoặc độ phức tạp của phương pháp tìm kiếm.
- Trong khi các giá trị  $S$  cao hơn đại diện cho **khai thác** cao hơn bằng cách tăng xác suất các kiến trúc tốt nhất trong quần thể được chọn làm cha mẹ cho thế hệ tiếp theo

## Ma trận hiệp phương sai Jacobi

Mục tiêu của việc đánh giá các kiến trúc ngay từ giai đoạn khởi tạo là cung cấp thông tin về không gian tìm kiếm mà không phải chịu chi phí cao của việc đào tạo thực tế, do đó cho phép hướng dẫn tìm kiếm đến cài đặt tối ưu.

- Để làm điều này, chúng ta sử dụng ước lượng proxy không tốn kém dựa trên Ma trận hiệp phương sai Jacobi.
- Điều này cho phép nhanh chóng đánh giá xem một kiến trúc có tốt không mà không cần đào tạo, do đó cho phép lựa chọn một kiến trúc được tạo ra để thêm vào quần thể với sự tự tin cao hơn rằng việc tìm kiếm đang được hướng dẫn một cách chính xác.

## Công thức tính toán

Giả sử có một ánh xạ tuyến tính,  $w_i = g(x_i)$ , mà đầu vào  $x_i \in \mathbb{R}^D$ , thông qua mạng  $g(x_i)$ , với  $x_i$  đại diện cho một hình ảnh thuộc một lô  $X$ , và  $D$  là kích thước đầu vào. Sau đó, Jacobi của ánh xạ tuyến tính có thể được tính toán bằng công thức:

$$J_i = \frac{\partial g(x_i)}{\partial x_i}$$

Điều này cho phép chúng ta đánh giá hành vi kiến trúc cho các hình ảnh khác nhau bằng cách tính toán  $J_i$  cho các điểm dữ liệu khác nhau  $g(x_i)$ , của một lô  $X$  duy nhất với  $i = 1, 2, \dots, N$ :

$$J = \begin{bmatrix} \frac{\partial g(x_1)}{\partial x_1} & \frac{\partial g(x_2)}{\partial x_2} & \cdots & \frac{\partial g(x_N)}{\partial x_N} \end{bmatrix}^\top$$

- $J$  chứa thông tin về đầu ra của kiến trúc đối với đầu vào cho nhiều hình ảnh.
- Chúng ta có thể chia đầu vào thành các lớp và đánh giá cách một kiến trúc mô hình hóa các chức năng phức tạp tại giai đoạn khởi tạo và tác động của nó đối với hình ảnh thuộc cùng một lớp.
- Để làm điều này, chúng ta chia  $J$  thành nhiều tập hợp, nơi mỗi tập hợp  $M_k$ , chứa tất cả Jacobi thuộc về cùng một lớp  $k$ .
- Sau đó, chúng ta có thể tính toán ma trận tương quan cho mỗi lớp, kí hiệu  $\Sigma_{M_k}$ , sử dụng các tập hợp đã thu được  $M_k$  với  $k = 1, 2, \dots, K$ .

Các ma trận tương quan cá nhân cung cấp thông tin về cách một kiến trúc đơn lẻ xử lý hình ảnh cho mỗi lớp. Tuy nhiên, các ma trận tương quan khác nhau có thể cho kết quả với kích thước khác nhau, do số lượng hình ảnh mỗi lớp khác nhau. Để có thể so sánh các ma trận tương quan khác nhau, chúng được đánh giá riêng lẻ:

$$E_k = \begin{cases} \sum_{i=1}^N \sum_{j=1}^N \ln (|(\Sigma M_k)_{i,j}| + t), & \text{nếu } K \leq \tau \\ \frac{1}{\sqrt{\#\Sigma M_k}} \sum_{i=1}^N \sum_{j=1}^N \ln (|(\Sigma M_k)_{i,j}| + t), & \text{ngược lại} \end{cases}$$

Trong đó  $t = 1 \times 10^{-5}$ ,  $K$  là số lớp trong lô  $X$  và  $\#(\cdot)$  là số phần tử.

Cuối cùng, một kiến trúc được tính điểm dựa trên các đánh giá cá nhân của các ma trận tương quan bằng cách sử dụng công thức sau:

$$z = \begin{cases} \sum_{w=1}^K |E_w|, & \text{nếu } K \leq \tau \\ \frac{1}{\#E} \sum_{i=1}^K \sum_{j=i+1}^K |E_i - E_j|, & \text{ngược lại} \end{cases}$$

Trong đó  $E$  chứa tất cả các điểm số của các ma trận tương quan. Điểm số cuối cùng phụ thuộc vào số lượng lớp có trong  $X$ , vì các bộ dữ liệu với số lượng lớp cao hơn thường có nhiều nhiễu hơn, điều này được giảm bớt bằng cách thực hiện một sự khác biệt chuẩn hóa theo cặp (normalized pair-wise difference). Định nghĩa theo thực nghiệm  $\tau = 100$ , dựa trên không gian tìm kiếm và các bộ dữ liệu được sử dụng.

Sau đó, chúng ta có thể sử dụng  $z$  để xếp hạng các kiến trúc được tạo ra, cung cấp một cơ chế hiệu quả để phân biệt giữa kiến trúc xấu và tốt, từ đó cho phép hướng dẫn tìm kiếm về phía các cài đặt tốt hơn mà không làm tăng chi phí tìm kiếm.

# Tính ma trận Jacobi

```
1 def get_batch_jacobian(net, x):
2     net.zero_grad()
3
4     x.requires_grad_(True)
5
6     _, y = net(x)
7
8     y.backward(torch.ones_like(y))
9     jacob = x.grad.detach()
10
11    return jacob
```

# Tính điểm số

```
1 def eval_score_perclass(jacob, labels=None, n_classes=10):
2     k = 1e-5
3     #n_classes = len(np.unique(labels))
4     per_class={}
5     for i, label in enumerate(labels):
6         if label in per_class:
7             per_class[label] = np.vstack((per_class[label], jacob[i]))
8         else:
9             per_class[label] = jacob[i]
10
11    ind_corr_matrix_score = {}
12    for c in per_class.keys():
13        s = 0
14        try:
15            corrs = np.corrcoef(per_class[c])
16
17            s = np.sum(np.log(abs(corrs)+k))#/len(corrs)
18            if n_classes > 100:
19                s /= len(corrs)
20        except: # defensive programming
21            continue
22
23    ind_corr_matrix_score[c] = s
```

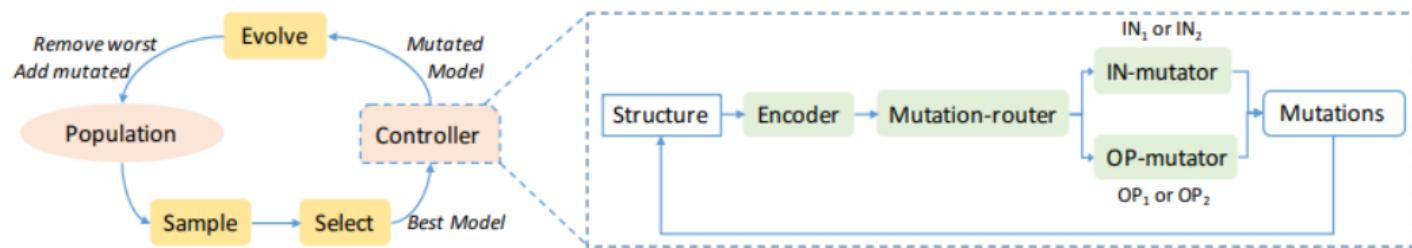
```
1 # per class-corr matrix A and B
2 score = 0
3 ind_corr_matrix_score_keys = ind_corr_matrix_score.keys()
4 if n_classes <= 100:
5
6     for c in ind_corr_matrix_score_keys:
7         # B)
8         score += np.absolute(ind_corr_matrix_score[c])
9     else:
10        for c in ind_corr_matrix_score_keys:
11            # A)
12            for cj in ind_corr_matrix_score_keys:
13                score += np.absolute(ind_corr_matrix_score[c]-ind_corr_matrix_score[cj])
14
15        # should divide by number of classes seen
16        score /= len(ind_corr_matrix_score_keys)
17
18 return score
```

## RENAS: Giải thuật Tiến hóa tăng cường

- Khi chỉ sử dụng EA cho NAS, có xu hướng tiến hoá các kiến trúc mạng sao cho vẫn bảo đảm sự đa dạng hoá quần thể. Tuy nhiên, thuật toán bị phụ thuộc quá nhiều vào sự đột biến ngẫu nhiên nên một số trường hợp, độ hiệu quả của EA không được đảm bảo.
- Khi chỉ sử dụng RL cho NAS, phụ thuộc nhiều vào các siêu tham số để đảm bảo tính ổn định. Nhưng khi quyết định kiến trúc mạng cho từng lớp, bộ điều khiển RL cần phải thử hàng chục hành động để có thể có được “phần thưởng”. Chính điều này có thể làm quá trình tìm kiếm thiếu hiệu quả.

# Tích hợp RL vào mô hình EA

Ý tưởng của thuật toán **Tiến hóa tăng cường**<sup>151</sup> (Reinforced Evolutionary - RENAS) chính là thiết kế một bộ điều khiển đột biến tăng cường nhằm học tạo ra con đường phù hợp cho việc tiến hóa quần thể, từ đó quần thể có thể tiến hóa tới một phiên bản tốt trong ít vòng lặp hơn.

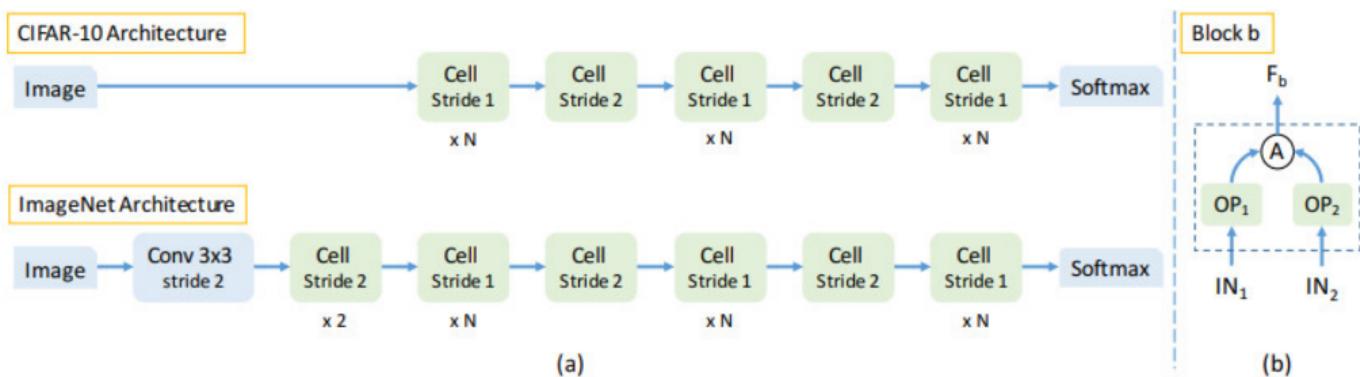


Hình 104: Mô hình thuật toán Tiến hóa tăng cường và cấu trúc bộ điều khiển đột biến tăng cường.

<sup>151</sup>RENAS: Reinforced Evolutionary Neural Architecture Search

# Không gian tìm kiếm

Thay vì thiết kế toàn bộ mạng cùng một lúc, chúng ta sẽ học từng kiến trúc cho tế bào (cell). Trong phần này, ta sẽ chia kiến trúc thành các tế bào và các khối.



Hình 105: Mô hình minh họa cho CIFAR-10 và ImageNet.

## Chi tiết mạng - Khối

1. Mỗi **khối** (block) sẽ bao gồm 2 đầu vào và 1 đầu ra, với các đầu vào  $\{i_1, i_2\}$  là các bản đồ đặc trưng (feature map), áp dụng 2 phép toán tương ứng với 2 bản đồ đặc trưng là  $\{o_1, o_2\}$  và kết hợp tạo thành đầu ra  $O$  qua phép cộng tương ứng phần tử (element-wise addition)  $A$ .
2. Mỗi **khối** sẽ được chỉ định là một xâu có độ dài 4,  $\{i_1, i_2, o_1, o_2\}$ , trong đó:
  - $\{i_1, i_2\}$  chọn từ  $\{O_1^c, O_2^c, O_3^c, \dots, O_{b-1}^c, O_B^{c-1}, O_B^{c-2}\}$ , với  $\{O_1^c, O_2^c, O_3^c, \dots, O_{b-1}^c\}$  là đầu ra của các khối trước trong tế bào hiện tại và  $\{O_B^{c-1}, O_B^{c-2}\}$  là đầu ra của 2 tế bào ngay trước tế bào hiện tại.
  - $\{o_1, o_2\}$  chọn từ set 6 toán tử sau:  $3 \times 3$  depth-wise separable convolution,  $5 \times 5$  depthwise-separable convolution,  $7 \times 7$  depth-wise separable convolution,  $3 \times 3$  avg pooling,  $3 \times 3$  max pooling, identity.

## Chi tiết mạng - Tế bào

1. Mỗi **tế bào** là đồ thị có hướng không chu trình chứa  $\#B$  khồi. Giả sử đầu vào là bản đồ đặc trưng  $h \times w \times f$ , với  $h, w$  là độ cao, độ rộng và  $f$  là số kênh thì:
  - Tế bào với bước nhảy 2 có đầu ra kích thước  $\frac{h}{2} \times \frac{w}{2} \times 2f$ .
  - Tế bào với bước nhảy 1 thì đầu ra kích thước không đổi.
2. Mỗi **tế bào** được xác định bằng 5 $\#B$  token, với 4 $\#B$  là số biến trong quá trình tìm kiếm.
3. Mỗi **mạng** xác định bởi ba yếu tố: cấu trúc tế bào, số lượng tế bào ( $\#N$ ), và số bộ lọc trong lớp đầu tiên ( $\#F$ ).

## Thuật toán 8.1: Thuật toán Tiền hóa tăng cường

**Đầu vào:** Số khôi cho mỗi tế bào  $\#B$ , số vòng lặp tối đa  $\#E$ , số bộ lọc trong lớp đầu tiên  $\#F$ , số tế bào trong mô hình  $\#N$ , kích thước quần thể  $\#P$ , kích thước mẫu  $\#S$ , tập huấn luyện  $D_{\text{train}}$ , tập xác thực  $D_{\text{val}}$ .

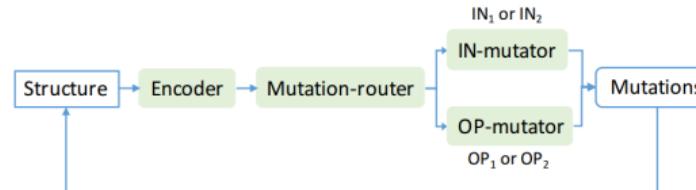
1.  $P^{(0)} \leftarrow \text{initialize}(\#F, \#N, \#P)$ .
2. Vòng lặp  $i = 1 : \#E$  thực hiện:
  - i  $S^{(i)} \leftarrow \text{sample}(P^{(i-1)}, \#S)$
  - ii  $B, W \leftarrow \text{select}(S^{(i)})$
  - iii  $C, w^B \leftarrow \text{reinforced-mutate}(B)$
  - iv  $w^C \leftarrow \text{finetune}(C, w^B, D_{\text{train}})$
  - v  $f_C \leftarrow \text{evaluate}(C, w^C, D_{\text{val}})$
  - vi  $P^{(i)} \leftarrow \text{push-pop}(P^{(i-1)}, C, f_C, W)$

**Đầu ra:** Quần thể các mô hình  $P$ .

## Đột biến tăng cường

**Đột biến tăng cường** (reinforced mutation) được thực hiện với bộ điều khiển đột biến để học sự ảnh hưởng bởi các biến đổi nhỏ và thực hiện đột biến. Bộ điều khiển nhận vào một chuỗi có độ dài  $5\#B$  đại diện cho kiến trúc tế bào được cung cấp, cụ thể nó bao gồm 4 phần:

1. Mã hóa (Encoder - *Enc*): theo sau một lớp nhúng để học ảnh hưởng của mỗi phần trong tế bào.
2. Cầu dẫn đột biến (Mutation-router - *Mut-rt*): để chọn một từ  $\{i_1, i_2, o_1, o_2\}$  của khối.
3. Đột biến đầu vào (IN-mutator - *IN-mut*): để thay đổi đầu vào của nút bằng một đầu vào mới  $i_{new}$ .
4. Đột biến toán tử (OP-mutator - *OP-mut*): để thay đổi toán tử của nút bằng một toán tử mới  $o_{new}$ .



Hình 106: Mô hình của bộ điều khiển.

## Encoder

- Đối với khối  $b$  trong  $Enc$ , các trạng thái ẩn của nó là  $\{H_{i_1}^b, H_{i_2}^b, H_{o_1}^b, H_{o_2}^b\}$ , với  $H_{o_1}^b$  biểu thị ảnh hưởng của  $o_1$  trong  $b$  lên toàn bộ mạng. Mỗi bước,  $Enc$  tạo ra  $5\#B$  trạng thái ẩn.
- Khởi tạo 2 trạng thái ban đầu,  $H^{c-1}, H^{c-2}$  là thông tin của 2 tế bào trước đó.
- Đối với khối  $b$ , bộ điều khiển thực hiện 2 quyết định theo trình tự. Ban đầu, dựa vào  $\{H_{i_1}^b, H_{i_2}^b, H_{o_1}^b, H_{o_2}^b\}$ ,  $Mut\text{-}rt$  quyết định sửa đổi 1 trong  $i_1, i_2, o_1, o_2$  ở khối  $b$  thông qua softmax. Nếu  $i_1$  hoặc  $i_2$  được chọn, thì  $IN\text{-}mut$  sẽ được kích hoạt để chọn một trong  $\{O_1^c, \dots, O_{b-1}^c, O_B^{c-1}, O_B^{c-2}\}$ . Ngược lại,  $OP\text{-}mut$  sẽ chọn một toán tử mới.
- Quá trình này được lặp lại  $B$  lần để sửa đổi một kiến trúc đã cho.

## Mutation-router

- Phần *Mut-rt* được thiết kế để tìm ra thành phần nào trong mỗi khối cần được sửa đổi.
- Đối với mỗi khối, đầu vào của *Mut-rt* là một tập con của đầu ra của *Enc* là  $\{H_{i_1}^b, H_{i_2}^b, H_{o_1}^b, H_{o_2}^b\}$  và đầu ra của nó là một trong các ID để đột biến:  $i_1, i_2, o_1, o_2$ .
- Áp dụng một lớp kết nối đầy đủ cho mỗi trạng thái ẩn và sử dụng softmax để tính toán xác suất sửa đổi cho mỗi thành phần  $\{P_{i_1}^b, P_{i_2}^b, P_{o_1}^b, P_{o_2}^b\}$  và lấy mẫu một trong  $i_1, i_2, o_1, o_2$  với những xác suất này.

# IN-mutator và OP-mutator

## 1. IN-mutator

- *IN-mut* chọn đầu vào mới cho nút, nếu  $ID \in (i_1, i_2)$ . Đầu vào gồm trạng thái ẩn  $H_{ID}^b$  của ID, trạng thái ẩn của tất cả đầu ra của các khối trước  $[H_A^1, \dots, H_A^{b-1}]$ , và trạng thái ẩn của 2 tế bào trước  $H^{c-1}, H^{c-2}$ .
- Nối  $[H_A^1, \dots, H_A^{b-1}, H^{c-1}, H^{c-2}]$  với  $H_{ID}^b$  và áp dụng một lớp kết nối đầy đủ cho chúng. Sử dụng softmax để tính toán xác suất thay thế đầu vào gốc bằng mỗi đầu vào thay thế và sau đó xác định  $i_{new}$  bằng cách chọn từ  $1, \dots, b-1, c-1, c-2$  với những xác suất này.

## 2. OP-mutator

- *OP-mut* xuất ra một toán tử mới  $o_{new}$  phụ thuộc vào đầu vào  $H_{ID}^b$ .
- Quá trình tương tự như *Mut-rt*.

## Thuật toán 8.2: Đột biến bở bộ điều khiển

**Đầu vào:** Số khồi mô tê bào  $\#B$ , một dãy  $a$  chứa  $4\#B$  số đặc trưng cho một tê bào.

1.  $H^{c-1}, H^{c-2} \leftarrow \text{Enc.begin}()$
2.  $H^1, H^2, \dots, H^B \leftarrow \text{Enc}(H^{c-1}, a)$
3. Vòng lặp  $b = 1 : \#B$  thực hiện:
  - i  $H_{i_1}^b, H_{i_2}^b, H_{o_1}^b, H_{o_2}^b, H_A^b \leftarrow H^b$
  - ii  $ID \leftarrow \text{Mut-rt}([H_{i_1}^b, H_{i_2}^b, H_{o_1}^b, H_{o_2}^b])$
  - iii Nếu  $ID \in \{i_1, i_2\}$  thì:  
 $i_{\text{new}} \leftarrow \text{IN-mut}(H_{ID}^b, [H_A^1, \dots, H_A^{b-1}, H^{c-1}, H^{c-2}])$   
 $m^{(b)} \leftarrow (ID, i_{\text{new}})$
  - iv Ngược lại thì:  
 $o_{\text{new}} \leftarrow \text{OP-mut}(H_{ID}^b)$   
 $m^{(b)} \leftarrow (ID, o_{\text{new}})$

**Đầu ra:** Một dãy hành động biến  $m$ .

## NSGA-II: Giải thuật Di truyền sắp xếp không trội

Bài toán tối ưu đa mục tiêu tổng quát có thể phát biểu dưới dạng:

$$\begin{aligned} & \min_{\mathbf{x} \in \Omega} F(\mathbf{x}) \\ \text{với } & g_i(\mathbf{x}) \leq 0, \forall i = \overline{1, m} \\ \text{và } & h_j(\mathbf{x}) = 0, \forall j = \overline{1, n} \end{aligned}$$

Trong đó:

- $\mathbf{x} = [x_1, x_2, \dots, x_d]$  là biến vector quyết định  $d$  chiều.
- $F(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_k(\mathbf{x})]$  gồm  $k$  mục tiêu ứng với  $k$  hàm số cần cực tiểu hóa.

Trong nhiều trường hợp, việc tìm lời giải cực tiểu toàn cục duy nhất cho tất cả mục tiêu là khó khăn hoặc thậm chí không thể có. Do đó, ta không sử dụng khái niệm tối ưu thông thường.

## Tính trội Pareto

Xét bài toán cực tiểu đa mục tiêu, lời giải  $\mathbf{x}^{(1)}$  được gọi là **trội hơn** so với lời giải  $\mathbf{x}^{(2)}$ , kí hiệu  $\mathbf{x}^{(1)} \preceq \mathbf{x}^{(2)}$ , nếu và chỉ nếu:

$$\begin{aligned}\forall i \in \{1, 2, \dots, k\} : \quad f_i(\mathbf{x}^{(1)}) &\leq f_i(\mathbf{x}^{(2)}) \\ \exists j \in \{1, 2, \dots, k\} : \quad f_j(\mathbf{x}^{(1)}) &< f_j(\mathbf{x}^{(2)})\end{aligned}$$

Nghĩa là  $\mathbf{x}^{(1)}$  không tệ hơn  $\mathbf{x}^{(2)}$  ở tất cả các hàm mục tiêu và có ít nhất một hàm mục tiêu mà  $\mathbf{x}^{(1)}$  tốt hơn hẳn  $\mathbf{x}^{(2)}$ .

## Tập tối ưu Pareto

Với bài toán cực tiểu đa mục tiêu  $F(\mathbf{x})$ , tập tối ưu Pareto  $\mathcal{P}^*$  được định nghĩa là:

$$\mathcal{P}^* = \{\mathbf{x} \in \Omega | \nexists \mathbf{x}' \in \Omega : F(\mathbf{x}') \preceq F(\mathbf{x})\}$$

Nói cách khác, nghiệm tối ưu của bài toán tối ưu đa mục tiêu là một tập các lời giải đôi một không trội hơn nhau và cũng không có lời giải nào khác trội hơn một trong các lời giải này.

Biên Pareto  $\mathcal{PF}^*$  được định nghĩa là:

$$\mathcal{PF}^* = \{F(\mathbf{x}) | \mathbf{x} \in \mathcal{P}^*\}$$

## Giải thuật tiến hóa cho bài toán tối ưu đa mục tiêu

- Với MOEA, quần thể tại thế hệ thứ  $t$  có tập tối ưu Pareto  $\mathcal{P}_{\text{current}}(t)$ .
- Ta sử dụng *quần thể phụ* lưu trữ các tập tối ưu Pareto tìm được từ thông tin của  $t$  thế hệ trước đó, các lời giải không trội hơn lẫn nhau trong  $\mathcal{P}_{\text{known}}(t) = \mathcal{P}_{\text{current}}(t) \cup \mathcal{P}_{\text{known}}(t - 1)$ .
- Tập tối ưu Pareto chính xác  $\mathcal{P}_{\text{true}}$  thường không thể tìm được chính xác.

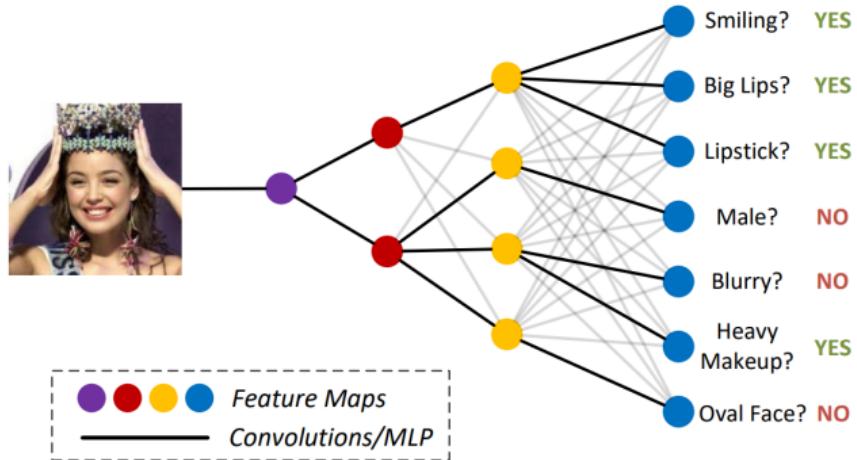
Mục tiêu của MOEA là trả về  $\mathcal{P}_{\text{known}} \subseteq \mathcal{P}_{\text{true}}$  hoặc xấp xỉ  $\mathcal{P}_{\text{true}}$  bởi  $\mathcal{P}_{\text{known}}$ .

## Thuật toán 9: Thuật toán Di truyền sắp xếp không trội 2

1. Khởi tạo quần thể  $\mathbb{P}$  gồm  $N$  cá thể.
2. Tính toán giá trị các hàm mục tiêu với mỗi cá thể.
3. Sắp xếp các cá thể thành các biên Pareto theo hạng.
4. Tạo quần thể con  $\mathbb{C}$ :
  - i Chọn cha mẹ theo thể thức đấu đôi.
  - ii Lai ghép và đột biến.
5. Vòng lặp thế hệ  $i = 1, 2, \dots, g$  với quần thể  $\mathbb{P} \cup \mathbb{C}$ :
  - i Tính hạng theo biên Pareto.
  - ii Trên mỗi biên Pareto tính độ đồng đúc từng cá thể.
  - iii Lấy ra đủ  $N$  cá thể tốt nhất dựa trên hạng và độ đồng đúc để cập nhật quần thể  $\mathbb{P}$ .
  - iv Tạo quần thể con  $\mathbb{C}$ .
  - v Bổ sung  $\mathbb{C}$  vào  $\mathbb{P}$ .

## Học đa thuộc tính sâu

- Học đa thuộc tính sâu (deep multi-attribute learning) là một lĩnh vực trong học máy và trí tuệ nhân tạo mà nó tập trung vào việc phát triển mô hình có khả năng học và hiểu được mối quan hệ phức tạp giữa các thuộc tính khác nhau, đồng thời cung cấp dự đoán chính xác cho mỗi thuộc tính. Mỗi thuộc tính có thể đại diện cho một khía cạnh cụ thể của đối tượng mà mô hình cần phải hiểu và dự đoán.
- Điều này có thể áp dụng trong nhiều lĩnh vực, từ xử lý ảnh và video đến xử lý ngôn ngữ tự nhiên, nơi mỗi thuộc tính có thể tương ứng với các khía cạnh như đối tượng, ngữ cảnh, tình cảm, và nhiều thuộc tính khác.



Hình 107: Minh họa học đa thuộc tính phân tích đa dạng đặc trưng của ảnh.

## GNAS: Giải thuật Tham lam

- Một vấn đề quan trọng trong học sâu đa thuộc tính là làm thế nào để khám phá các mối tương quan giữa các thuộc tính.
- Các phương pháp học sâu đa thuộc tính thông thường sẽ thiết kế thủ công cấu trúc mạng dựa trên nhiệm vụ cụ thể, dẫn đến sự thiếu linh hoạt trong các tình huống phức tạp.
- Phương pháp tìm kiếm kiến trúc mạng tham lam (GNAS) sẽ tự động tìm kiếm kiến trúc tối ưu dạng cây cho việc học đa thuộc tính. GNAS sẽ chia công việc tối ưu kiến trúc thành việc tối ưu hóa từng nốt riêng lẻ.
- Bằng cách liên tục cập nhật các kiến trúc con, toàn bộ kiến trúc dạng cây sẽ hội tụ với các lớp dưới cùng được chia sẻ trên các thuộc tính có liên quan và các lớp trên mã hóa nhiều hơn các tính năng cụ thể của thuộc tính.

## Một số ưu điểm của GNAS

- Với thuật toán tham lam, GNAS làm giảm số lượng kiến trúc ứng viên, giảm từ độ phức tạp theo hàm mũ xuống độ phức tạp tuyến tính.
- GNAS có thể đẩy nhanh đáng kể việc huấn luyện lan truyền ngược của các kiến trúc ứng viên bằng cách kết hợp cơ chế chia sẻ trọng số trên các kiến trúc ứng viên khác nhau.
- GNAS có thể sử dụng để tìm kiếm kiến trúc mạng có dạng cây tùy ý. Không gian tìm kiếm lớn của GNAS đảm bảo hiệu suất của kiến trúc được tìm kiếm.

## Phát biểu bài toán

Mục tiêu là tìm ra kiến trúc mạng nơ-ron dạng cây tối ưu  $\hat{G}$  cho reward  $R$  lớn nhất:

$$\hat{G} = \arg \max_G R(G) = \arg \max_G \frac{1}{N} \sum_{n=1}^N r_n(G)$$

$R$  là độ chính xác trung bình trong việc dự đoán các thuộc tính trên tập xác thực, còn  $r_n$  là độ chính xác dự đoán của thuộc tính thứ  $n$  trên tập xác thực và  $N$  là số thuộc tính.

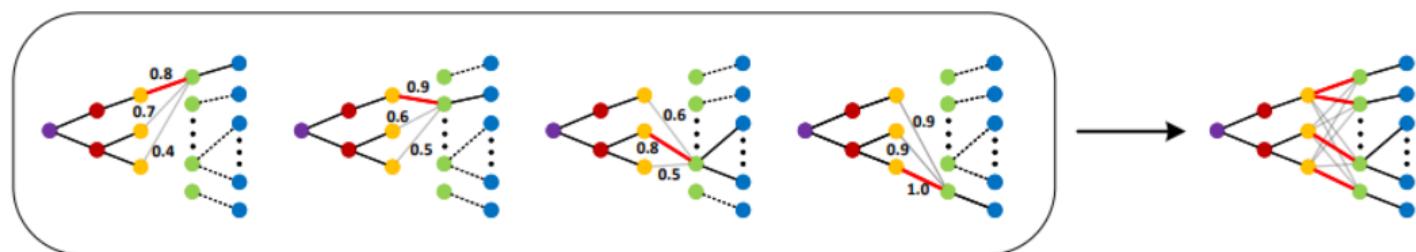
$G$  có cấu trúc dạng cây, có  $M$  lớp. Trong mỗi lớp  $l$ , có  $B_l$  khôi, trong đó mỗi khôi bao gồm một số lượng bản đồ tính năng cố định.  $B_1 = 1$  vì lớp đầu tiên là hình ảnh đầu vào và  $B_M = N$  vì lớp cuối cùng là  $N$  đầu ra của dự đoán thuộc tính.

Để thuận tiện, ta sử dụng một tập hợp các ma trận kề cận nhị phân  $A$  để biểu thị cấu trúc mạng của mạng nơ-ron  $G$ .  $A_{i,j}^{(l)} = 1$  biểu thị rằng có một kết nối - được cố định là convolutions hoặc perceptron đa lớp (Multilayer Perceptron - MLP) tùy theo nhu cầu - giữa khối thứ  $i$  của lớp  $l$  và khối thứ  $j$  của lớp  $l + 1$ , còn nếu không có kết nối thì  $A_{i,j}^{(l)} = 0$ . Khi đó bài toán trở thành

$$\hat{A} = \arg \max_A R(A), \quad \text{với điều kiện: } \sum_{i=1}^{B_l} A_{i,j}^{(l)} = 1, 1 \leq j \leq B_{l+1}$$

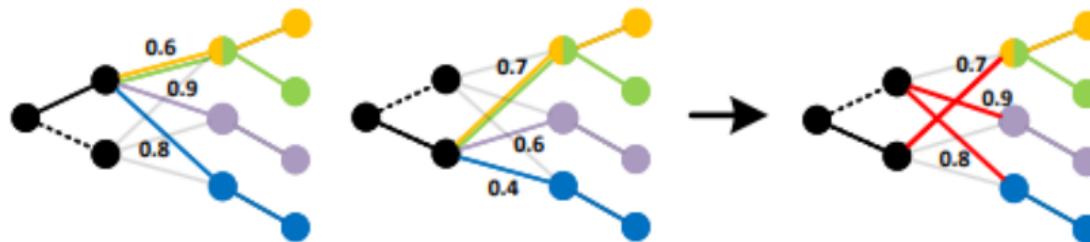
## Tối ưu hóa nội lớp

- Bước đầu tiên của GNAS là tối ưu hóa nội lớp (Intra-Layer Optimization). Bước này thực hiện rất đơn giản do chỉ phải đánh giá kết nối giữa  $B_l$  khối của lớp  $l$  với khối ở lớp thứ  $l + 1$ .
- Để tối ưu hóa các kết nối trong một lớp ta dựa vào một giả định tham lam sau: “Kiến trúc nội lớp tối ưu được tạo bởi các kết nối tối ưu đối với từng thuộc tính riêng lẻ”.



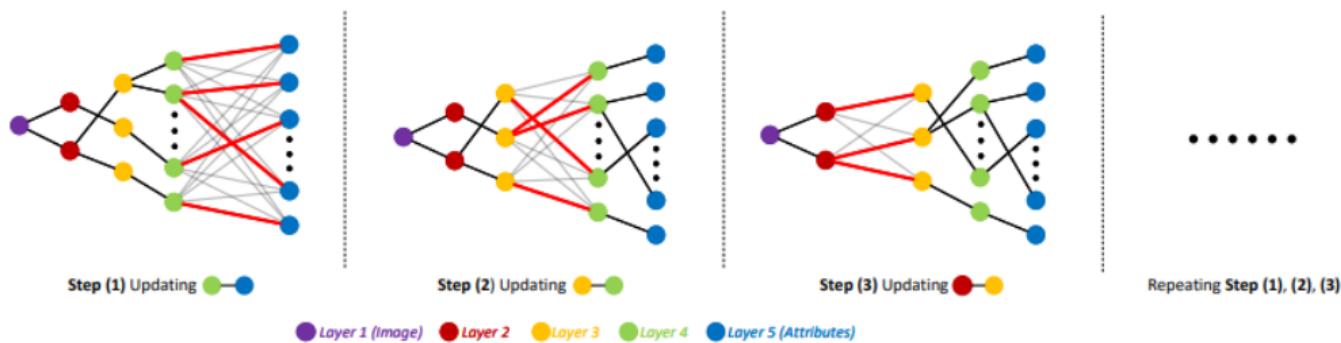
## Tìm kiếm nội lớp tăng tốc

- Người ta nhận thấy có thể đồng thời đánh giá reward  $R$  giữa một khối ở lớp thứ  $l$  với tất cả các khối ở lớp thứ  $l + 1$ .



## Cập nhật theo từng lớp

- Bước này dựa trên một giả định tham lam.



## Thuật toán 10: Thuật toán Tham lam

**Đầu vào:** Tập huấn luyện  $D_{\text{train}}$ , tập xác thực  $D_{\text{valid}}$ , số lượng lớp  $M$ , số lượng khối  $B$ .

1. Khởi tạo kiến trúc  $A$  và trọng số  $W$  bất kỳ.
2. Vòng lặp  $l = \overline{M - 1, 1}$  và  $b = \overline{1, B_l}$  thực hiện:
  - i Tìm kiếm nội lớp tăng tốc.
  - ii Cập nhật theo từng lớp.

## PSOCNN: Giải thuật Tối ưu bầy đàn

Tối ưu bầy đàn (Particle Swarm Optimization - PSO) là một thuật toán lấy ý tưởng từ tự nhiên, và có thể được dùng để tìm kiếm kiến trúc mạng nơ-ron tối ưu, tương tự như giải thuật di truyền.

- Thuật toán PSO mới đề xuất có thể tìm kiếm kiến trúc mạng tối ưu bằng việc sử dụng các cá thể có thể thay đổi "chiều dài" không giới hạn.
- Toán tử “khác biệt” mới cho phép 2 cá thể có sự khác biệt về số lớp và tham số kết hợp với nhau, và tốc độ (velocity) của cá thể được cập nhật mà không sử dụng sơ đồ mã hoá có giá trị thực.
- Toán tử “tốc độ” mới, được nghĩ ra để sửa đổi kiến trúc CNN đã có, giống với cá thể tốt nhất trong đòn hoặc cấu hình tốt nhất của bản thân. Nhằm giảm chiều PSO.

## Tối ưu hoá bầy đàn

PSO lấy ý tưởng từ kiểu bay của một đàn chim. Trong PSO, 1 lời giải gọi là cá thể, tập hợp các lời giải gọi là đàn.

Ý tưởng chính là mỗi cá thể chỉ mang thông tin về tốc độ hiện tại của chúng. Gồm  $pBest$  - cấu hình tốt nhất của bản thân,  $gBest$  - giá trị tốt nhất trong đàn. Sau mỗi vòng lặp, mỗi cá thể điều chỉnh tốc độ sao cho càng gần  $pBest$  và  $gBest$  cùng thời điểm.

Tốc độ  $v$  của cá thể, sẽ cập nhật theo công thức:

$$v_{i,j}(t+1) = w \times v_{i,j}(t) + c_p \times r_p \times (pBest_{i,j} - x_{i,j}(t)) + c_g \times r_g \times (gBest_j - x_{i,j}(t))$$

Trong đó:

- $v_{i,j}$  là tốc độ của cá thể  $i$  ở chiều  $j$ .
- $x$  là vị trí cá thể hiện tại.
- $w$  là hằng số chi phí thay đổi tốc độ.
- $c_p, c_g$  là hằng số được biết trước.
- $r_p, r_g$  là số ngẫu nhiên trong  $[0, 1]$ .

Vị trí của cá thể cũng được cập nhật như sau:

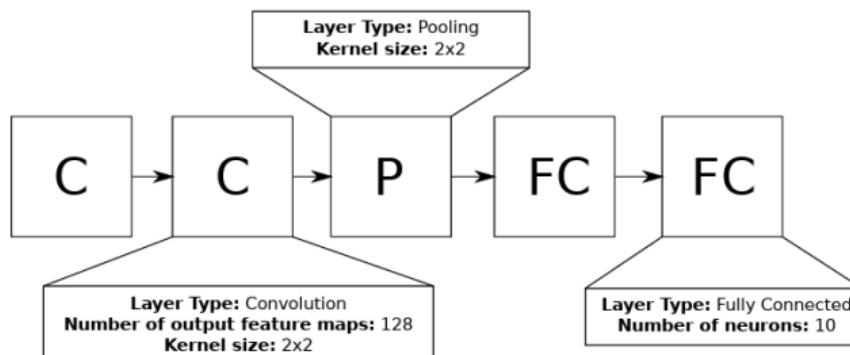
$$x_{i,j}(t+1) = x_{i,j}(t) + v_{i,j}(t+1)$$

## Đại diện cho mạng nơ-ron tích chập

Đại diện (Representation) là một trong những khía cạnh quan trọng nhất trong thiết kế thuật toán liên quan đến các cấu trúc phức tạp, như kiến trúc CNN.

Ví dụ: Có 3 loại lớp: Tích chập, Gộp, Kết nối đầy đủ.

- Đại diện cho lớp Tích chập là kích thước bộ lọc và số lượng kết quả bản đồ đặc trưng.
- Đại diện cho lớp Gộp là kích thước bộ lọc.
- Đại diện cho lớp Kết nối đầy đủ là số lượng nơ-ron.



Hình 108: Đại diện cho kiến trúc CNN.

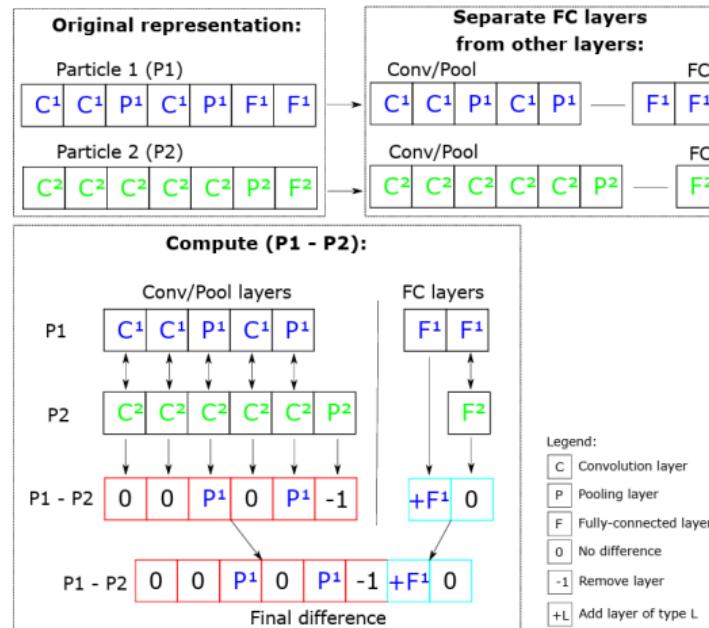
## Khởi tạo đàn

Khởi tạo đàn là bước đầu tiên trong đề xuất psoCNN.

- Tạo ra  $N$  cá thể với kiến trúc CNN ngẫu nhiên.
- Mỗi cá thể có số lớp ngẫu nhiên từ 3 đến  $l_{max}$  (số lớp tối đa).
- Sao cho lớp đầu, lớp cuối luôn là lớp Tích chập và lớp Kết nối đầy đủ.
- Và sau khi thêm lớp Kết nối đầy đủ vào kiến trúc thì mỗi lớp sau đó đều là lớp Kết nối đầy đủ.

Hàm thích nghi được thực hiện bởi hàm ComputeLoss(). Việc tự đánh giá sẽ hoàn thành dựa trên sự kết hợp các hàm loss của mỗi cá thể, ở đây là cross-entropy loss.

# Đo lường khác biệt giữa 2 cá thể.



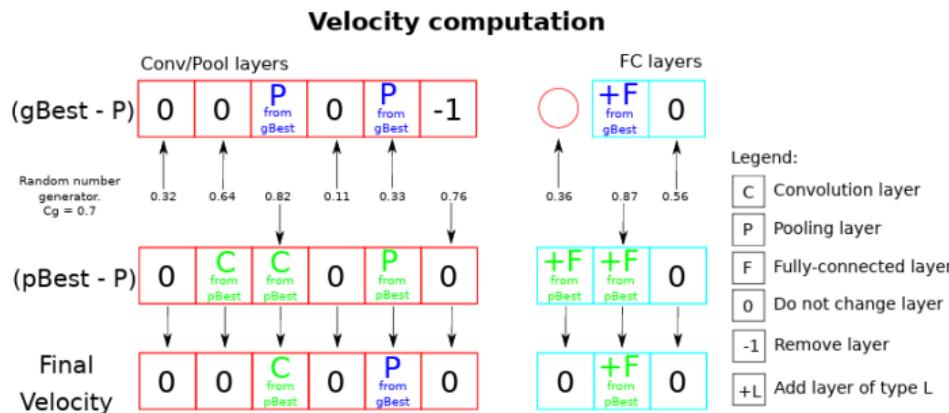
Hình 109: Minh họa toán tử khác biệt giữa hai kiến trúc.

# Tính toán tốc độ

Để tính toán tốc độ của cá thể  $P$  tới tốt nhất của đàn ( $gBest$ ) hay tốt nhất của bản thân ( $pBest$ ), cần tính 2 hàm khác biệt:  $(gBest - P)$  và  $(pBest - P)$ .

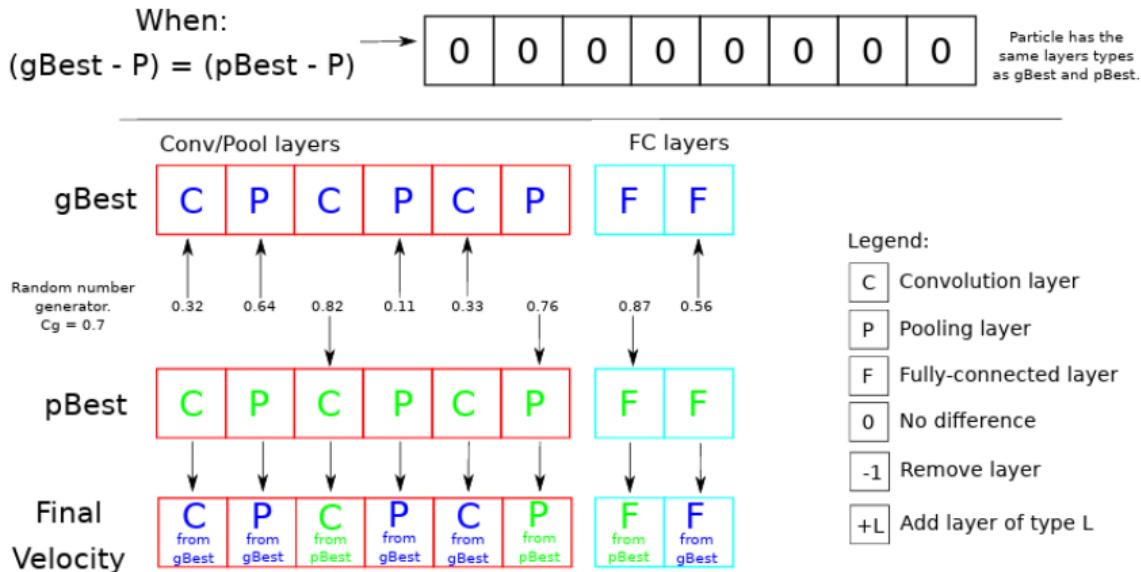
Tốc độ cuối được tính dựa vào số ngẫu nhiên  $r$  trong  $[0, 1]$

- Nếu  $r \leq c_g$ , ta sẽ chọn lớp đó từ  $(gBest - P)$ .
- Ngược lại, ta chọn lớp đó từ  $(pBest - P)$ .



Hình 110: Toán tử tốc độ giữa các kiến trúc.

## Velocity computation (special case)



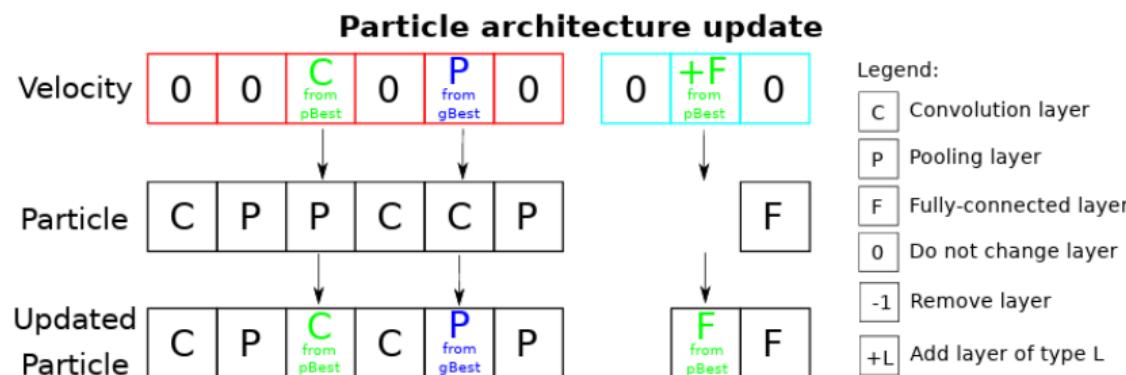
Hình 111: Trường hợp đặc biệt của toán tử tốc độ.

## Cập nhật cá thể

Lớp được thêm hay loại bỏ từ kiến trúc cá thể dựa vào tốc độ của chúng. Tuy nhiên, thuật toán cần phải theo dõi được số lượng lớp Gộp.

Phụ thuộc vào kích thước đầu vào huấn luyện, sẽ có hữu hạn lớp Gộp được cho phép.

Nếu sau khi cập nhật, số lượng lớp Gộp vượt quá số lượng cho phép, lớp Gộp trong cá thể sẽ bị loại bỏ từ cuối về đầu.



## Bài toán tối ưu hai cấp

Bài toán tối ưu hai cấp (Bilevel Optimization Problem) tổng quát có dạng:

$$\min_{x \in X \subseteq \mathbb{R}^{d_1}} \ell(x) := f(x, y^*(x))$$

$$\text{với } y^*(x) \in \arg \min_{y \in \mathbb{R}^{d_2}} g(x, y)$$

Trong đó  $d_1, d_2 \in \mathbb{N}$ ,  $X$  là tập đóng, lồi và  $f, g : X \times \mathbb{R}^{d_2} \rightarrow \mathbb{R}$  là các hàm số khả vi liên tục theo từng biến  $x, y$ .

- $\min_y g(x, y)$  là bài toán trong (cấp dưới)
- $\min_x \ell(x)$  là bài toán ngoài (cấp trên)

Ngay cả khi hàm số  $\ell(x)$  và  $g(x, y)$  lồi chặt theo các biến thì bài toán cũng rất khó giải quyết.

## Lời giải bài toán

Để giải bài toán theo phương pháp gradient, với mỗi  $x^{\text{cur}} \in \mathbb{R}^{d_1}$ , ta thực hiện 2 vòng lặp để:

1. Giải bài toán trong  $y^*(x^{\text{cur}}) = \arg \min g(x^{\text{cur}}, y)$  và sau đó
2. Tính toán gradient  $\nabla \ell(x^{\text{cur}})$  dựa trên lời giải  $y^*(x^{\text{cur}})$

Giải bài toán BiO, ta tập trung tìm cặp nghiệm  $(x^*, y^*)$  thỏa mãn điều kiện dừng bậc một:

$$\nabla_y g(x^*, y^*) = 0 \text{ (i)} \quad \text{và} \quad \langle \nabla \ell(x^*), x - x^* \rangle \geq 0, \forall x \in X \text{ (ii)}$$

- Nếu cho trước  $x^*$ , nghiệm  $y^*$  thỏa (i) có thể tìm được bằng cách cập nhật:

$$y \leftarrow y - \beta \nabla_y g(x^*, y)$$

- Mặt khác, nếu cho trước  $y^*(x)$ , nghiệm  $x^*$  thỏa (ii) cập nhật theo cách sau:

$$x \leftarrow x - \alpha \nabla \ell(x)$$

Theo quy tắc vi phân toàn phần hàm nhiều biến, ta có:

$$\nabla \ell(x) = \nabla_x f(x, y^*(x)) + \nabla_x y^*(x)^\top \nabla_y f(x, y^*(x))$$

Mặt khác,  $y^*(x)$  thỏa mãn  $\nabla_y g(x, y^*(x)) = 0$ . Lấy vi phân hàm ẩn:

$$\nabla_{xy}^2 g(x, y^*(x)) + \nabla_x y^*(x)^\top \nabla_{yy}^2 g(x, y^*(x)) = 0$$

Do đó:

$$\nabla \ell(x) = \nabla_x f(x, y^*(x)) - \nabla_{xy}^2 g(x, y^*(x)) [\nabla_{yy}^2 g(x, y^*(x))]^{-1} \nabla_y f(x, y^*(x))$$

Thay  $y^*(x)$  bởi  $y \in \mathbb{R}^{d_2}$ . Ta định nghĩa:

$$\bar{\nabla}_x f(x, y) := \nabla_x f(x, y) - \nabla_{xy}^2 g(x, y) [\nabla_{yy}^2 g(x, y)]^{-1} \nabla_y f(x, y)$$

## Giả mã giải thuật

### Thuật toán 10: TTSA<sup>152</sup>

1. Bắt đầu với  $(x_0, y_0) \in X \times \mathbb{R}^{d_2}$  và dãy bước nhảy  $\{\alpha_k, \beta_k\}_{k \geq 0}$ .
2. Trong vòng lặp thứ  $k$ , ta cập nhật:

$$\begin{aligned}y_{k+1} &= y_k - \beta_k \nabla_y g(x_k, y_k) \\x_{k+1} &= x_k - \alpha_k \bar{\nabla}_x f(x_k, y_{k+1})\end{aligned}$$

<sup>152</sup>A Two-Timescale Stochastic Algorithm Framework for Bilevel Optimization: Complexity Analysis and Application to Actor-Critic

## Biểu diễn kiến trúc mạng

Biểu diễn mạng bởi các tế bào tính toán (computation cell), sắp xếp các tế bào tạo nên kiến trúc mạng cuối cùng.

- Mỗi tế bào là một đồ thị có hướng không chu trình, gồm dãy  $N$  nút có thứ tự.
- Nối giữa hai nút  $x_i$  và  $x_j$  ( $i < j$ ) bởi một cạnh có hướng  $(i, j)$  thể hiện toán tử  $o_{ij}(\cdot)$ .
- Một nút sẽ được tính toán bằng tổng tất cả các nút trước nó sau khi tác động toán tử.

$$x_j = \sum_{i < j} o_{ij}(x_i)$$

- Có những toán tử *không*, thể hiện không kết nối.
- Đầu ra của cả tế bào do sự xếp chồng tất cả các nút, hay là  $\text{concat}(x_1, x_2, \dots, x_N)$ .

## Nối lồng ràng buộc liên tục

Xét một nút  $x_i$  nào đó. Gọi  $\mathcal{O}$  là tập các toán tử ứng viên, chẳng hạn: tích chập, gộp tối đại, và cả *toán tử không*.

Để không gian tìm kiếm trở nên liên tục, ta giảm sự lựa chọn theo loại của một toán tử cụ thể thành một softmax trên tất cả các toán tử có thể:

$$\bar{o}_{ij}(x_i) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_{ij}(o))}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{ij}(o'))} o(x_i)$$

Trong đó, mỗi toán tử ứng với cạnh  $(i, j)$  được tham số hóa bởi vector  $\alpha_{ij} \in \mathbb{R}^{|\mathcal{O}|}$ .

Việc tìm kiếm kiến trúc mạng trở thành việc học các biến mã hóa liên tục  $\alpha = \{\alpha_{ij}\}$ . Kết thúc tìm kiếm, ta thay toán tử hỗn hợp  $\bar{o}_{ij}$  bởi toán tử nào tựa nó nhất hay là:

$$o_{ij} = \arg \max_{o \in \mathcal{O}} \alpha_{ij}(o)$$

## Mục tiêu tối ưu

Ta sẽ học đồng thời kiến trúc  $\alpha$  và trọng số  $w$  trong mọi toán tử và cực tiểu hóa măt măt trên tập đánh giá (validation set) bằng gradient descent.

Gọi  $\mathcal{L}_{\text{train}}$  và  $\mathcal{L}_{\text{validation}}$  tương ứng là măt măt trên tập huấn luyện và tập đánh giá. Cả hai đều phụ thuộc kiến trúc  $\alpha$  và trọng số  $w$ .

Nhiệm vụ trở thành Bài toán tối ưu hai cấp độ:

$$\min_{\alpha} \mathcal{L}_{\text{validation}}(w^*(\alpha), \alpha)$$
$$\text{với } w^*(\alpha) = \arg \min_w \mathcal{L}_{\text{train}}(w, \alpha)$$

Trong đó  $\alpha$  là biến cấp trên và  $w$  là biến cấp dưới.

## Xấp xỉ gradient kiến trúc

Theo Thuật toán TTSA, ta thực hiện cập nhật theo dạng:

$$\begin{aligned} w_{k+1} &= w_k - B_k \nabla \mathcal{L}_{\text{train}}(w_k, \alpha_k) \\ \alpha_{k+1} &= \alpha_k - A_k \bar{\nabla}_\alpha \mathcal{L}_{\text{validation}}(w_{k+1}, \alpha_k) \end{aligned}$$

Tính toán chính xác gradient kiến trúc có thể trở nên không khả thi do bài toán tối ưu cấp dưới tồn kém. Với  $w' = w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha)$ , lấy gradient toàn phần ta có:

$$\bar{\nabla}_\alpha \mathcal{L}_{\text{validation}}(w', \alpha) = \underbrace{\nabla_\alpha \mathcal{L}_{\text{validation}}(w', \alpha)}_{\text{thành phần cấp 1}} - \xi \underbrace{\nabla_{\alpha w}^2 \mathcal{L}_{\text{train}}(w, \alpha)^\top \nabla_{w'} \mathcal{L}_{\text{validation}}(w', \alpha)}_{\text{thành phần cấp 2}} \quad (\text{i})$$

Trong đó ta sử dụng tốc độ học  $\xi \geq 0$  chung cho cả hai bài toán tối ưu. Không cần giải chính xác bài toán cấp dưới hay huấn luyện đến khi nó hội tụ. Nếu  $\xi = 0$ , ta có thể xấp xỉ (i) bởi thành phần gradient cấp 1  $\nabla_\alpha \mathcal{L}_{\text{validation}}(w, \alpha)$ , tuy nhanh hơn nhưng kém hiệu quả hơn.

Thành phần gradient cấp 2 chứa phép toán nhân ma trận có khôi lượng tính toán lớn. Do đó, ta sẽ tìm cách xấp xỉ đại lượng này.

Với số  $\varepsilon$  đủ nhỏ, chọn  $\varepsilon = \frac{0.01}{\|\nabla_{w'} \mathcal{L}_{\text{validation}}(w', \alpha)\|_2}$  và đặt  $w^\pm = w \pm \varepsilon \mathcal{L}_{\text{validation}}(w', \alpha)$ . Khi đó:

$$\nabla_{\alpha w}^2 \mathcal{L}_{\text{train}}(w, \alpha)^\top \nabla_{w'} \mathcal{L}_{\text{validation}}(w', \alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{\text{train}}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{\text{train}}(w^-, \alpha)}{2\varepsilon}$$

## Thuật toán 11: Tìm kiếm kiến trúc mạng khả vi<sup>153</sup>

1. Tạo toán tử hỗn hợp  $\bar{\alpha}_{ij}$  tham số hóa bởi  $\alpha_{ij}$  cho các cạnh  $(i, j)$ .

2. **while** chưa đạt điều kiện dừng **do**

2.1 Cập nhật kiến trúc  $\alpha$ :

$$\alpha := \alpha - \nabla_\alpha \mathcal{L}_{\text{validation}}(w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha), \alpha)$$

2.2 Cập nhật trọng số  $w$ :

$$w := w - \nabla_w \mathcal{L}_{\text{train}}(w, \alpha)$$

3. Huấn luyện kiến trúc  $\alpha$  học được.

<sup>153</sup>DARTS: Differentiable Architecture Search

# Định nghĩa các khối phép toán

## 1. Phép toán không

```
1 class Zero(nn.Module):
2
3     def __init__(self, stride):
4         super(Zero, self).__init__()
5         self.stride = stride
6
7     def forward(self, x):
8         if self.stride == 1:
9             return x.mul(0.)
10        return x[:, :, ::self.stride, ::self.stride].mul(0.)
```

Mục đích: Trả về tensor toàn số 0 có cùng hoặc đã giảm kích thước dựa trên bước nhảy.

## 2. Phép toán đồng nhất

```
1 class Identity(nn.Module):  
2  
3     def __init__(self):  
4         super(Identity, self).__init__()  
5  
6     def forward(self, x):  
7         return x
```

Mục đích: Trả về chính đầu vào.

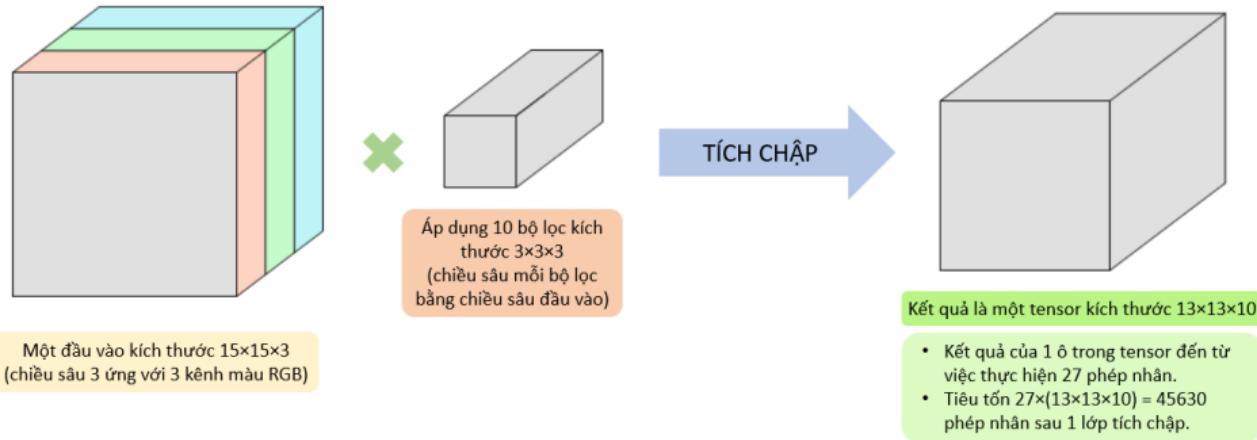
### 3. Phép toán điều chỉnh kích thước kenh

- Lấy cảm hứng từ kiến trúc NASNet và AmoebaNet.
- Sử dụng 2 phép toán tích chập, có bước nhảy 2, kích thước chỉ là  $1 \times 1$ .
- Mỗi phép tạo ra một tensor có độ sâu  $C_{\text{out}}/2$ , sau đó nối chúng lại.
- Kết quả là tạo ra tensor có độ sâu (số kênh) đúng bằng  $C_{\text{out}}$  (nếu  $C_{\text{out}}$  chẵn).

```
1 class FactorizedReduce(nn.Module):
2
3     def __init__(self, C_in, C_out, affine=True):
4         super(FactorizedReduce, self).__init__()
5         assert C_out % 2 == 0
6         self.relu = nn.ReLU(inplace=False)
7         self.conv_1 = nn.Conv2d(C_in, C_out // 2, 1, stride=2, padding=0, bias=False)
8         self.conv_2 = nn.Conv2d(C_in, C_out // 2, 1, stride=2, padding=0, bias=False)
9         self.bn = nn.BatchNorm2d(C_out, affine=affine)
10
11    def forward(self, x):
12        x = self.relu(x)
13        out = torch.cat([self.conv_1(x), self.conv_2(x[:, :, 1:, 1:])], dim=1)
14        out = self.bn(out)
15        return out
```

## 4. Phép toán tích chập phân tách (Separable Convolution)

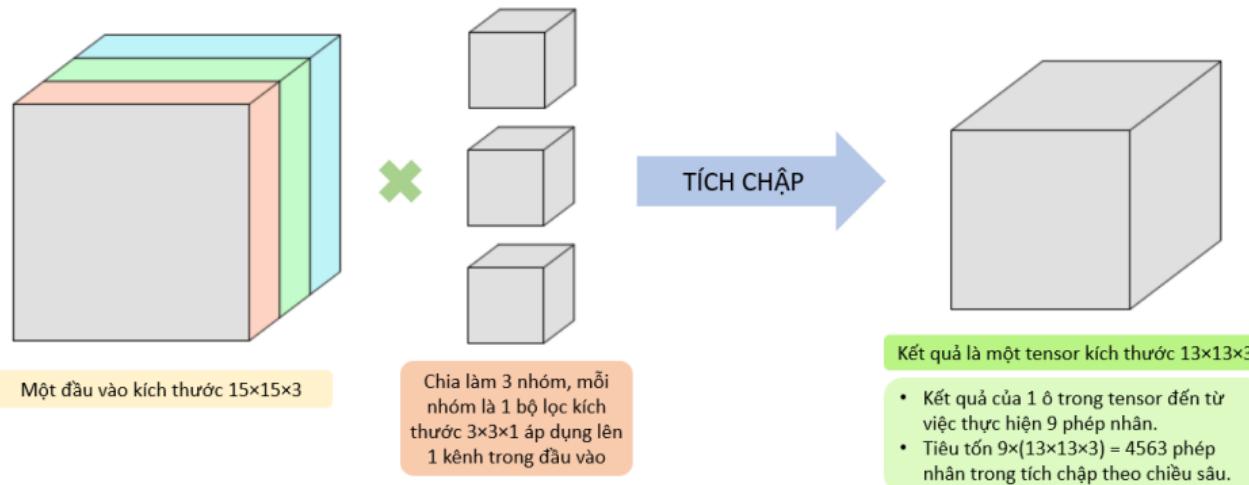
- Phiên bản tối ưu của tích chập truyền thống.
- Giúp giảm đáng kể số lượng phép nhân và tham số.
- Gồm 2 phép tích chập liên tiếp theo thứ tự: Depthwise, Pointwise.



Hình 112: Minh họa khái lượng tính toán của tích chập thông thường

## 4.1. Phép toán tích chập theo chiều sâu (Depthwise Convolution)

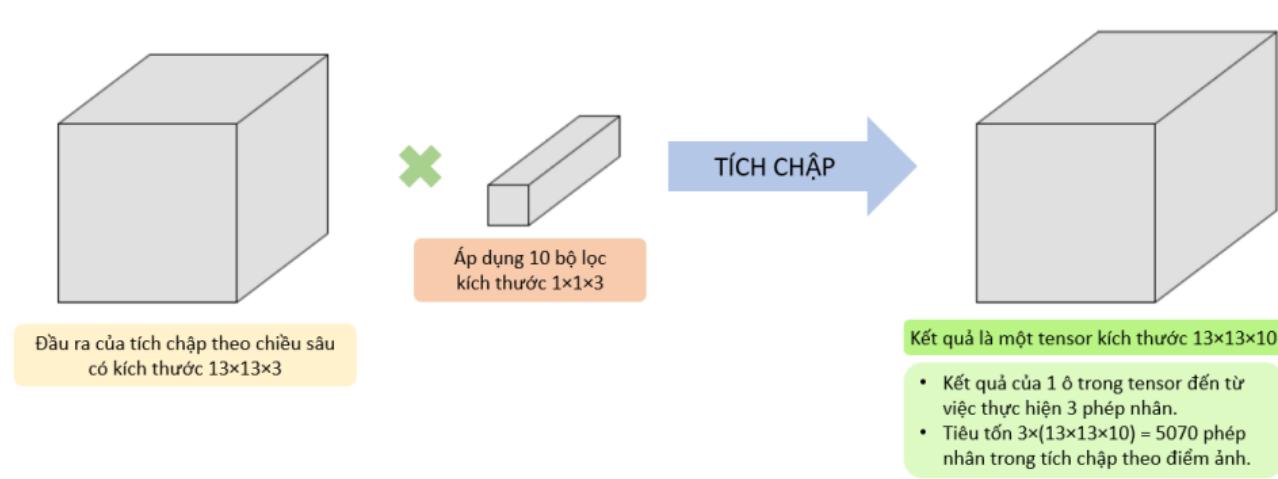
- Chia các bộ lọc làm  $C_{in}$  nhóm.
- Mỗi nhóm có một bộ lọc và thực hiện tích chập tương ứng lên một kênh.
- Đầu ra là một tensor bảo toàn số kênh.



Hình 113: Minh họa khối lượng tính toán của tích chập theo chiều sâu

## 4.2. Phép toán tích chập theo điểm ảnh (Pointwise Convolution)

- Áp dụng các bộ lọc kích thước  $1 \times 1$ , chiều sâu đúng bằng tensor áp dụng và số lượng đúng bằng chiều sâu muôn đạt được.
- Hai giai đoạn tích chập này có tổng khối lượng tính toán ít hơn rất nhiều so với tích chập truyền thống.



Hình 114: Minh họa khối lượng tính toán của tích chập theo điểm ảnh

```
1 class SepConv(nn.Module):
2
3     def __init__(self, C_in, C_out, kernel_size, stride, padding, affine=True):
4         super(SepConv, self).__init__()
5         self.op = nn.Sequential(
6             nn.ReLU(inplace=False),
7             nn.Conv2d(C_in, C_in, kernel_size=kernel_size, stride=stride, padding=padding,
8                     groups=C_in, bias=False),
9             nn.Conv2d(C_in, C_in, kernel_size=1, padding=0, bias=False),
10            nn.BatchNorm2d(C_in, affine=affine),
11            nn.ReLU(inplace=False),
12            nn.Conv2d(C_in, C_in, kernel_size=kernel_size, stride=1, padding=padding, groups=
13                     C_in, bias=False),
14            nn.Conv2d(C_in, C_out, kernel_size=1, padding=0, bias=False),
15            nn.BatchNorm2d(C_out, affine=affine),
16        )
17
18     def forward(self, x):
19         return self.op(x)
```

## 5. Phép toán tích chập mở rộng<sup>154</sup>

Với  $F : \mathbb{Z}^2 \rightarrow \mathbb{R}$  là tensor và  $k : [-r, r]^2 \cap \mathbb{Z}^2 \rightarrow \mathbb{R}$  là kernel kích thước  $(2r + 1) \times (2r + 1)$ . Phép toán tích chập  $*$  giữa  $F$  và  $k$  được định nghĩa là:

$$(F * k)(\mathbf{p}) = \sum_{\mathbf{s} + \mathbf{t} = \mathbf{p}} F(\mathbf{s})k(\mathbf{t})$$

Ta mở rộng khái niệm này, phép toán tích chập với độ mở rộng  $\ell \in \mathbb{N}$ , kí hiệu  $*_\ell$  cho bởi:

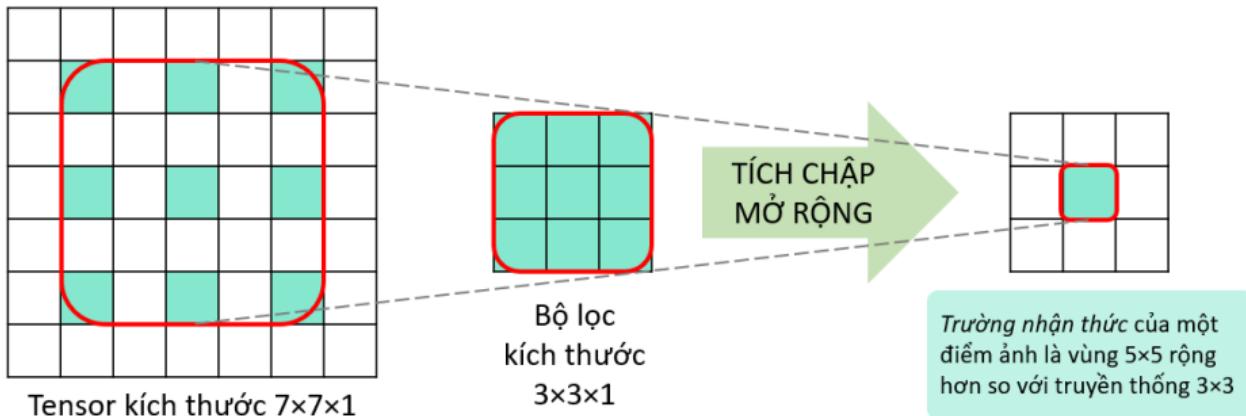
$$(F *_\ell k)(\mathbf{p}) = \sum_{\mathbf{s} + \ell\mathbf{t} = \mathbf{p}} F(\mathbf{s})k(\mathbf{t})$$

---

<sup>154</sup>Multi-Scale Context Aggregation by Dilated Convolutions

So với tích chập truyền thông, tích chập mở rộng có những ưu điểm hơn hẳn:

- Tăng trường nhận thức mà không tăng số lượng tham số, tăng hiệu suất tính toán.
- Giảm mất mát thông tin theo không gian so với phép gộp, duy trì trạng thái dữ liệu.
- Không làm mất độ phân giải của hình ảnh đầu ra.



**Hình 115:** Minh họa tích chập mở rộng có độ mở rộng  $\ell = 2$

```
1 class DilConv(nn.Module):
2
3     def __init__(self, C_in, C_out, kernel_size, stride, padding, dilation, affine=True):
4         super(DilConv, self).__init__()
5         self.op = nn.Sequential(
6             nn.ReLU(inplace=False),
7             nn.Conv2d(C_in, C_in, kernel_size=kernel_size, stride=stride, padding=padding,
8                     dilation=dilation, groups=C_in, bias=False),
9             nn.Conv2d(C_in, C_out, kernel_size=1, padding=0, bias=False),
10            nn.BatchNorm2d(C_out, affine=affine),
11        )
12
13     def forward(self, x):
14         return self.op(x)
```

## Định nghĩa toán tử ứng viên

Tập các toán tử ứng viên bao gồm:

- none: Phép toán không.
- avg\_pool\_3x3: Gộp trung bình.
- max\_pool\_3x3: Gộp tối đại.
- skip\_connection: Đồng nhất hoặc Điều chỉnh kênh để phù hợp với đầu vào.
- sep\_conv\_3x3, sep\_conv\_5x5, sep\_conv\_7x7
- dil\_conv\_3x3, dil\_conv\_5x5
- conv\_7x1\_1x7: Tương đương áp dụng bộ lọc  $7 \times 7$ .

## Hạn chế của DARTS

Với thuật toán DARTS trước đó, nhận thấy:

- Một tế bào (cell) được tạo nên từ các nút và các kết nối, mỗi kết nối là toán tử. Các toán tử gán trọng số kiến trúc, được học trong quá trình tìm kiếm.
- DARTS tìm kiếm kiến trúc trên mạng nông sau đó đánh giá trên mạng sâu hơn.

Như vậy, DARTS dẫn đến vấn đề *chênh lệch độ sâu*, nghĩa là giai đoạn tìm kiếm tìm ra một số toán tử hoạt động tốt trong kiến trúc nông, nhưng giai đoạn đánh giá thực sự cần những toán tử khác phù hợp hơn với một kiến trúc sâu.

## Đề xuất của P-DARTS<sup>155</sup>

Khắc phục các hạn chế của DARTS, giải thuật tiên bộ hơn được đề xuất:

- **Tăng dần độ sâu:** Chia quá trình tìm kiếm thành nhiều giai đoạn và tăng dần độ sâu của mạng ở cuối mỗi giai đoạn.
- **Không gian tìm kiếm xấp xỉ** (*search space approximation*): Khi độ sâu tăng lên, P-DARTS giảm số lượng ứng viên (toán tử) dựa trên điểm số của họ trong các giai đoạn tìm kiếm trước đó. Mục tiêu là giảm thiểu chi phí tính toán và tăng tốc độ tìm kiếm.
- **Không gian tìm kiếm hiệu chỉnh** (*search space regularization*): Một vấn đề khác, thiếu ổn định, xuất hiện khi tìm kiếm trên một kiến trúc sâu. Thuật toán có thể bị lệch nặng về kết nối tắt (skip connect) vì nó thường dẫn đến sự giảm lỗi nhanh chóng trong quá trình tối ưu hóa.
  - i Sử dụng dropout cấp độ toán tử để giảm chi phí của kết nối tắt.
  - ii Kiểm soát sự xuất hiện của kết nối tắt trong quá trình đánh giá.

---

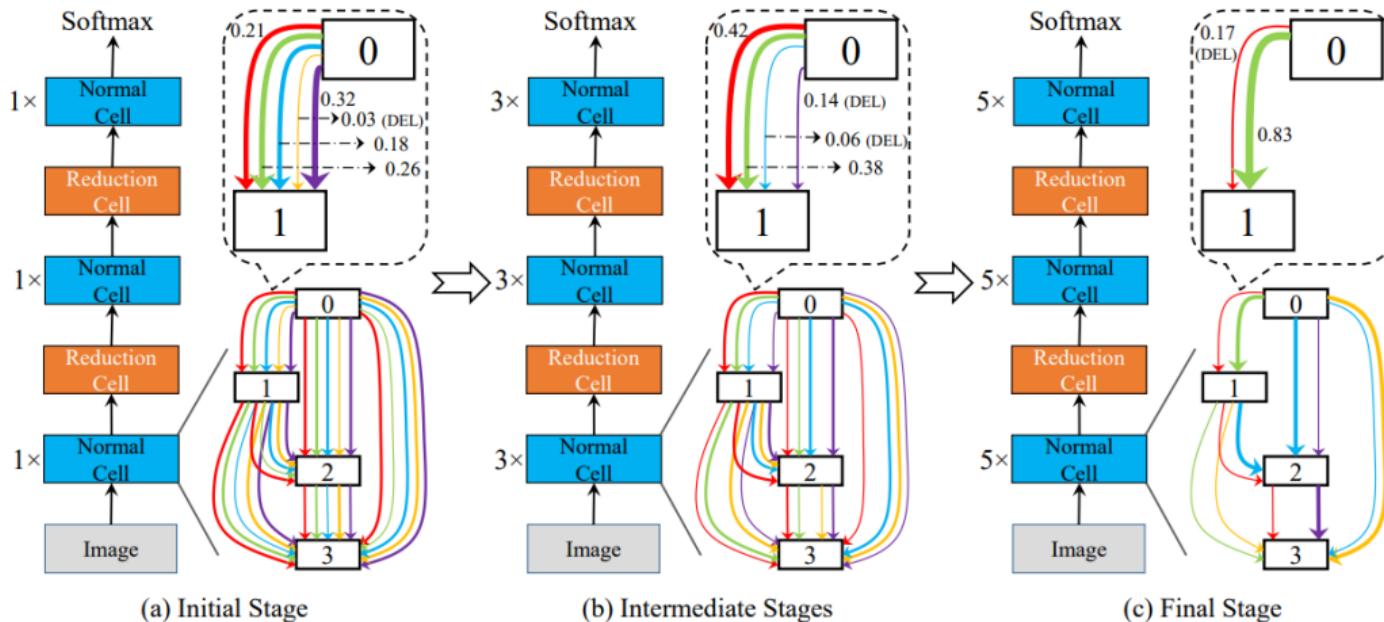
<sup>155</sup>Progressive Differentiable Architecture Search: Bridging the Depth Gap between Search and Evaluation

## Không gian tìm kiếm xấp xỉ

Quá trình tìm kiếm được chia thành nhiều giai đoạn, bao gồm một giai đoạn ban đầu, một hoặc vài giai đoạn trung gian và một giai đoạn cuối.

Đối với mỗi giai đoạn,  $\mathcal{S}_k$ , mạng bao gồm  $\mathcal{L}_k$  khối (cells) và tập toán tử có kích thước  $\mathcal{O}_k$  hay là  $|\mathcal{O}_{ij}^k| = \mathcal{O}_k$ . Tức mọi cạnh đều có tập toán tử gồm  $\mathcal{O}_k$  ứng viên.

- Ở giai đoạn đầu tiên, mạng tìm kiếm nông nhưng không gian toán tử lớn.
- Sau giai đoạn  $\mathcal{S}_{k-1}$ , tham số kiến trúc  $\alpha_{k-1}$  được học và thứ hạng các toán tử trên mỗi cạnh được xếp hạng theo  $\alpha_{k-1}$ . Khi đó, ta tăng số khối hay  $\mathcal{L}_k > \mathcal{L}_{k-1}$  và giảm số toán tử hay  $\mathcal{O}_k < \mathcal{O}_{k-1}$ .
- Quá trình tăng chiều sâu kiến trúc tiếp tục cho đến khi nó đủ gần với chiều sâu được sử dụng để đánh giá. Sau giai đoạn tìm kiếm cuối cùng, xác định các khối theo các tham số kiến trúc đã học.



Hình 116: Minh họa các giai đoạn tìm kiếm: tăng khối và giảm toán tử

## Không gian tìm kiếm hiệu chỉnh

Ở bắt đầu mỗi giai đoạn,  $S_k$ , huấn luyện lại kiến trúc (lúc này đã sửa đổi) từ đầu, nghĩa là tất cả trọng số được khởi tạo, vì một vài toán tử ứng viên đã bị loại bỏ.

- Tuy nhiên, việc huấn luyện một mạng sâu hơn khó khăn hơn so với việc huấn luyện một mạng nông.
- Bằng thực nghiệm, nhận thấy thông tin thích chảy qua các kết nối tắt thay vì tích chập hoặc gộp, có lẽ do kết nối tắt thường dẫn đến sự giảm gradient nhanh chóng.

Do đó, quá trình tìm kiếm có xu hướng tạo ra các kiến trúc với nhiều toán tử kết nối tắt, giới hạn số lượng tham số có thể học và do đó tạo ra hiệu suất không thỏa đáng ở giai đoạn đánh giá. Đây là một dạng quá khớp (overfitting).

Giải quyết hạn chế bằng hiệu chỉnh không gian tìm kiếm, gồm hai phần:

- i Chèn dropout ở cấp độ toán tử sau mỗi kết nối tắt nhằm cắt đứt một phần con đường truyền trực tiếp qua kết nối tắt, tạo điều kiện cho giải thuật khám phá các toán tử khác. Tuy nhiên, nếu liên tục chặn các kết nối tắt thì thuật toán sẽ loại bỏ bằng cách gán trọng số thấp cho chúng, điều này gây hại cho hiệu suất cuối cùng. Do đó, tỉ lệ dropout phải giảm dần theo thời gian.
- ii Tinh chỉnh kiến trúc, đơn giản là kiểm soát số lượng kết nối tắt được giữ lại, sau giai đoạn tìm kiếm cuối cùng, để trở thành một hằng số  $M$ .

Quá trình này lặp đi lặp lại:

- i Bắt đầu với việc xây dựng một cấu trúc khôi (cell topology) bằng cách sử dụng thuật toán tiêu chuẩn mô tả bởi DARTS.
- ii Kiểm tra xem số lượng kết nối tắt có đúng bằng  $M$  hay không.
- iii Nếu số lượng kết nối tắt không chính xác, tìm  $M$  kết nối tắt có trọng số lớn nhất và đặt trọng số của những kết nối còn lại về 0.
- iv Sau đó, xây dựng lại cấu trúc khôi với các tham số kiến trúc đã được sửa đổi.

Việc điều chỉnh trọng số và xây dựng lại cấu trúc tế bào có thể tạo ra các kết nối tắt mới không mong muốn. Do đó, phải lặp lại quá trình này cho đến khi đạt được số lượng kết nối tắt mong muốn.

## Đề xuất của PC-DARTS<sup>156</sup>

DARTS cần rất nhiều bộ nhớ và tài nguyên tính toán vì cố gắng huấn luyện một mạng lớn (gọi là siêu mạng) và đồng thời tìm kiếm kiến trúc tối ưu.

Để đạt được hiệu quả tìm kiếm nhanh chóng, giải thuật PC-DARTS được đề xuất:

- Thay vì tìm kiếm toàn bộ siêu mạng, ta chỉ lấy mẫu và tìm kiếm trên một phần nhỏ của nó. Điều này giúp giảm bớt sự dư thừa khi khám phá và tiết kiệm tài nguyên.
- Chọn ra một tập con các kênh trong siêu mạng để tìm kiếm, trong khi bỏ qua những kênh khác. Tuy nhiên, việc lựa chọn một phần của siêu mạng có thể tạo ra một số vấn đề. Để giải quyết vấn đề này, sử dụng *chuẩn hóa cạnh* (edge normalization).

---

<sup>156</sup>PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search

## Kết nối một phần kẽm

Với DARTS, trong mỗi tê bào có  $|\mathcal{O}|$  toán tử và đầu ra tương ứng được lưu tại mỗi nút, khiến cho việc sử dụng bộ nhớ tăng lên theo tỷ lệ của số lượng toán tử  $|\mathcal{O}|$ . Để phù hợp với GPU, người ta phải giảm kích thước lô (batch) trong quá trình tìm kiếm, làm chậm tốc độ và có thể làm giảm ổn định cũng như tính chính xác của việc tìm kiếm.

Xét kết nối từ  $x_i$  đến  $x_j$ . Ta xác định một lớp phủ  $S_{ij}$  gán 1 cho các kẽm được chọn và 0 cho các kẽm còn lại. Các kẽm được chọn được gửi vào tính toán hỗn hợp của  $|\mathcal{O}|$  toán tử, trong khi các kẽm còn lại bỏ qua các toán tử này, tức là:

$$\bar{o}_{ij}^{\text{PC}} = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_{ij}(o))}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{ij}(o'))} o(S_{ij}x_i) + (1 - S_{ij})x_i$$

Tỷ lệ của các kẽm được chọn là  $\frac{1}{K}$ . Bằng cách thay đổi  $K$ , chúng ta có thể đánh đổi giữa độ chính xác của tìm kiếm kiến trúc ( $K$  nhỏ) và hiệu quả ( $K$  lớn).

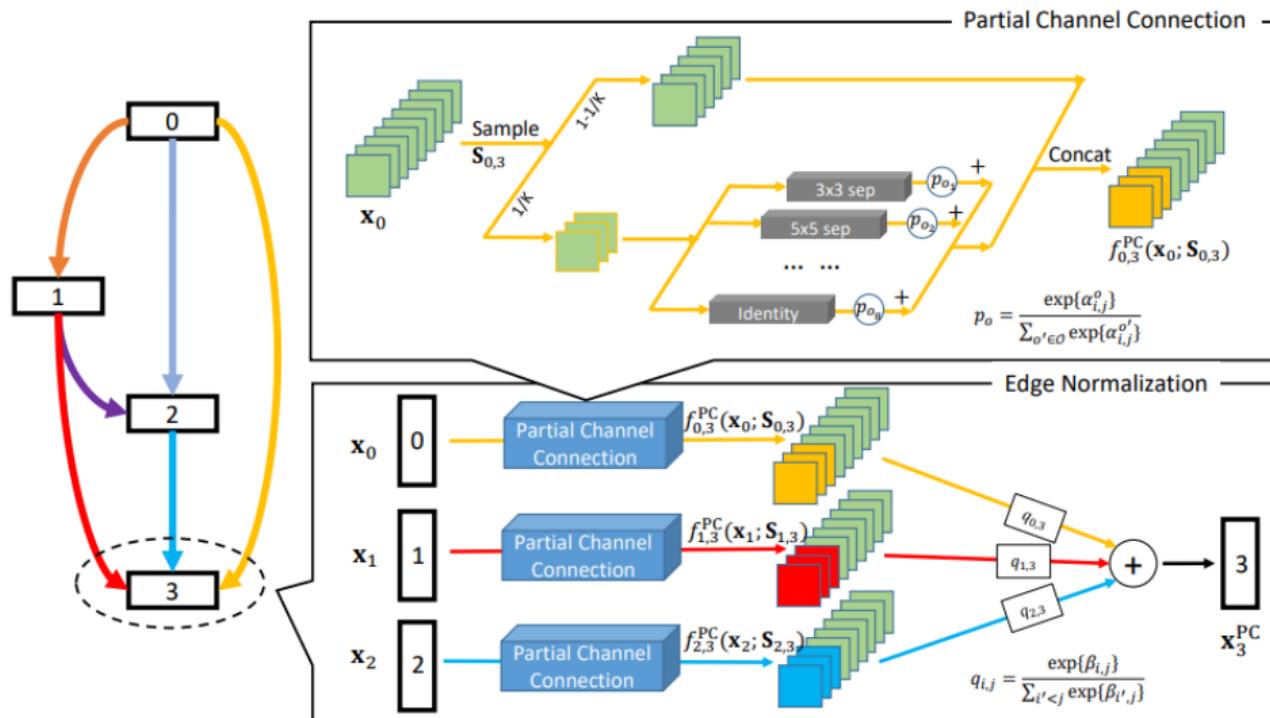
## Hiệu quả tìm kiếm

Bộ nhớ phụ của việc tính toán  $\bar{o}_{ij}^{\text{PC}}$  giảm đi  $K$  lần, do đó có thể sử dụng kích thước lô (batch) lớn hơn, tăng tính ổn định cho việc tìm kiếm.

Xem xét tác động của việc lấy mẫu kênh trong tìm kiếm:

- **Tích cực:** Ít thiên vị hơn trong việc chọn toán tử. Trong thực tế, các toán tử không sử dụng trọng số thường được ưu tiên ở giai đoạn đầu của quá trình tìm kiếm vì chúng không cần phải học trọng số và cung cấp đầu ra ổn định hơn. Ngược lại, các toán tử sử dụng trọng số cần thời gian để tối ưu hóa trọng số của chúng, làm cho đầu ra trở nên không ổn định.
- **Tiêu cực:** Trong một tế bào (cell), mỗi nút đầu ra cần phải chọn hai nút đầu vào từ tiền nhiệm của nó, điều này là dựa trên DARTS gốc. Những tham số kiến trúc này được tối ưu hóa thông qua việc lấy mẫu ngẫu nhiên các kênh. Nhưng vì sự ngẫu nhiên này, kết nối tối ưu có thể không ổn định theo thời gian.

Khắc phục hạn chế đó, ta sử dụng *chuẩn hóa cạnh*.



Hình 117: Minh họa giải thuật PC-DARTS

## Chuẩn hóa cạnh

Theo DARTS, xét một khối, mỗi đầu ra  $x_j$  cần lấy đầu vào từ 2 trong số các nút trước nó  $\{x_0, x_1, \dots, x_{j-1}\}$  được đánh trọng số bởi  $\max_{o \in \mathcal{O}} \alpha_{1,j}(o), \max_{o \in \mathcal{O}} \alpha_{2,j}(o), \dots, \max_{o \in \mathcal{O}} \alpha_{j-1,j}(o)$ .

Để giảm sự không ổn định khi lấy mẫu kênh ngẫu nhiên, ta chuẩn hóa cạnh bằng cách đánh trọng số  $\beta_{ij}$ . Khi đó:

$$x_j^{\text{PC}} = \sum_{i < j} \frac{\beta_{ij}}{\sum_{i' < j} \beta_{i'j}} \bar{o}_{ij}(x_i)$$

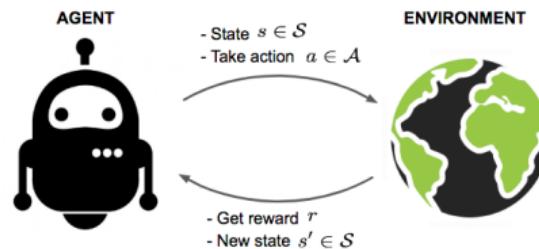
Kết thúc quá trình tìm kiếm, kết nối trên cạnh  $(i, j)$  được xác định bởi cả  $\{\alpha_{ij}(o)\}$  và  $\{\beta_{ij}\}$ . Bằng cách nhân hệ số:

$$\frac{\beta_{ij}}{\sum_{i' < j} \beta_{i'j}} \times \frac{\exp(\alpha_{ij}(o))}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{ij}(o'))}$$

## RL: Học tăng cường

Xét một **tác tử** (agent) trong một **môi trường** (environment) không biết trước và tác tử này có thể nhận được một số phần thưởng bằng cách tương tác với môi trường.

- Tác tử cần phải thực hiện các hành động để tối đa hóa phần thưởng tích lũy.
- Mục tiêu của Học tăng cường (RL) là học một chiến lược tốt cho tác tử từ các thử nghiệm và phản hồi đơn giản.
- Với chiến lược tối ưu, tác nhân có khả năng thích ứng chủ động với môi trường để tối đa hóa phần thưởng trong tương lai.



Hình 118: Minh họa tác tử và môi trường

# Khái niệm cơ bản

Tác tử đang hoạt động trong một môi trường.

- Cách môi trường phản ứng với các hành động nhất định được định nghĩa bởi một **mô hình** (model) mà chúng ta có thể biết hoặc không.
- Tác nhân có thể ở trong một trong nhiều **trạng thái** (stage,  $s \in \mathcal{S}$ ) của môi trường, và chọn thực hiện một trong nhiều hành động (action,  $a \in \mathcal{A}$ ) để chuyển từ trạng thái này sang trạng thái khác.
- Trạng thái mà tác nhân sẽ đến được quyết định bởi xác suất chuyển tiếp giữa các trạng thái ( $P$ ).
- Một khi hành động được thực hiện, môi trường cung cấp một phần thưởng (reward,  $r \in \mathcal{R}$ ) như phản hồi.

Mô hình định nghĩa hàm phần thưởng và xác suất chuyển tiếp. Chúng ta có thể biết hoặc không biết cách mô hình hoạt động và điều này phân biệt hai trường hợp:

- Biết mô hình: lập kế hoạch với thông tin hoàn hảo; làm RL dựa trên mô hình. Khi chúng ta hoàn toàn biết môi trường, chúng ta có thể tìm ra giải pháp tối ưu bằng cách Quy hoạch động (Dynamic Programming - DP).
- Không biết mô hình: học với thông tin không đầy đủ; làm RL không dựa trên mô hình hoặc cố gắng học mô hình một cách rõ ràng như một phần của thuật toán.

Tiếp theo, chính sách  $\pi(s)$  (policy) của tác nhân cung cấp hướng dẫn về hành động tối ưu để thực hiện trong một trạng thái nhất định với **mục tiêu tối đa hóa tổng số phần thưởng**.

- Mỗi trạng thái được liên kết với một hàm **giá trị** (value)  $V(s)$  dự đoán số lượng phần thưởng trong tương lai mà chúng ta có thể nhận được trong trạng thái này bằng cách hành động theo chính sách tương ứng.
- Nói cách khác, hàm giá trị định lượng mức độ tốt của một trạng thái. Cả chính sách và hàm giá trị là những gì chúng ta cố gắng học trong học tăng cường.

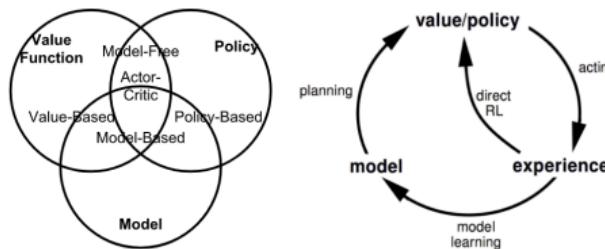
Sự tương tác của tác tử với môi trường thể hiện bởi một chuỗi hành động và phần thưởng được quan sát theo thời gian,  $t = 1, 2, \dots, T$ . Trong quá trình này, tác nhân tích luỹ kiến thức về môi trường, học chính sách tối ưu và đưa ra quyết định về hành động tiếp theo để học chính sách tốt nhất một cách hiệu quả.

Trạng thái, hành động, phần thưởng tại mốc thời gian  $t$  là  $S_t, A_t, R_t$ . Khi đó, chuỗi tương tác được mô tả hoàn toàn bằng một tập hợp các tập (episode, còn được gọi là “thử nghiệm” hoặc “quỹ đạo”) và chuỗi kết thúc tại trạng thái cuối cùng  $S_T$ .

$$S_1, A_1, R_2, S_2, A_2, \dots, A_{T-1}, R_T, S_T$$

## Thuật ngữ khác

- **Model-based:** Phụ thuộc vào mô hình của môi trường; hoặc mô hình đã biết hoặc thuật toán học nó một cách rõ ràng.
- **Model-free:** Không phụ thuộc vào mô hình trong quá trình học.
- **On-policy:** Sử dụng kết quả xác định (deterministic) hoặc mẫu từ chính sách mục tiêu để đào tạo thuật toán.
- **Off-policy:** Đào tạo trên một phân phối của các chuyển đổi (transition) hoặc tập (episode) được tạo ra bởi một chính sách hành vi khác với chính sách mục tiêu.



**Hình 119:** Tóm tắt về các phương pháp trong RL dựa vào việc chúng ta muốn mô hình hóa giá trị, chính sách hoặc môi trường.

## Mô hình: Chuyển đổi và Phần thưởng

Mô hình là một bộ mô tả về môi trường. Với mô hình, chúng ta có thể học hoặc suy luận về cách môi trường sẽ tương tác và cung cấp phản hồi cho tác nhân. Mô hình có 2 phần chính, hàm xác suất chuyển đổi  $P$  và hàm phần thưởng  $R$ .

Hãy nói khi chúng ta ở trong trạng thái  $s$ , chúng ta quyết định thực hiện hành động  $a$  để đến trạng thái tiếp theo  $s'$  và nhận phần thưởng  $r$ . Điều này được gọi là một bước **chuyển đổi** (transition), được biểu diễn bằng một bộ số  $(s, a, s', r)$ .

Hàm chuyển đổi  $P$  ghi lại xác suất chuyển từ trạng thái  $s$  sang  $s'$  sau khi thực hiện hành động  $a$  và nhận phần thưởng  $r$ . Chúng ta sử dụng  $\mathbb{P}$  để biểu thị “xác suất”.

$$P(s', r | s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a]$$

Hàm chuyển đổi trạng thái có thể được định nghĩa:

$$P_{ss'}^a = P(s'|s, a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} P(s', r | s, a)$$

Hàm phần thưởng  $R$  dự đoán phần thưởng tiếp theo được kích hoạt bởi một hành động:

$$R(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} P(s', r | s, a)$$

## Chính sách

Chính sách, dưới dạng hàm hành động của tác nhân, cho biết chúng ta nên thực hiện hành động nào trong trạng thái  $s$ . Nó là một ánh xạ từ trạng thái  $s$  sang hành động  $a$  và có thể là xác định hoặc ngẫu nhiên:

- Xác định:  $\pi(s) = a$ .
- Ngẫu nhiên:  $\pi(a|s) = \mathbb{P}_\pi[A = a|S = s]$ .

## Hàm định giá

Hàm định giá đo lường sự tốt lành của một trạng thái hoặc sự đáng khen ngợi của một trạng thái hoặc một hành động thông qua dự đoán về phần thưởng tương lai. Phần thưởng tương lai, còn được gọi là **hoàn trả** (return), là tổng của các phần thưởng đã giảm giá tiền về phía trước. Hãy tính toán hoàn trả  $G$  từ thời điểm  $t$ :

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{+\infty} \gamma^k R_{t+k+1}$$

Hệ số giảm giá  $\gamma \in [0, 1]$  làm giảm giá trị phần thưởng trong tương lai, bởi vì:

- Phần thưởng trong tương lai có thể có độ không chắc chắn cao hơn; ví dụ, thị trường chứng khoán.
- Phần thưởng trong tương lai không cung cấp lợi ích ngay lập tức; ví dụ, con người có thể thích có vui vẻ hôm nay hơn là sau 5 năm.
- Việc giảm giá tạo ra sự tiện lợi về mặt toán học; ví dụ, chúng ta không cần theo dõi vĩnh viễn các bước tương lai để tính hoàn trả.
- Chúng ta không cần phải lo lắng về các vòng lặp vô hạn trong biểu đồ chuyển trạng thái.

Giá trị của trạng thái  $s$  (state-value) là kỳ vọng của hoàn trả nếu chúng ta ở trong trạng thái này vào thời điểm  $t$ ,  $S_t = s$ :

$$V_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

Tương tự, chúng ta định nghĩa giá trị của hành động (action-value, hay Q-value) của một cặp trạng thái - hành động như sau:

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Ngoài ra, vì chúng ta tuân theo chính sách mục tiêu  $\pi$ , chúng ta có thể sử dụng phân phối xác suất qua các hành động có thể và giá trị  $Q$  để khôi phục giá trị trạng thái:

$$V_\pi(s) = \sum_{a \in \mathcal{A}} Q_\pi(s, a) \pi(a|s)$$

Sự khác biệt giữa giá trị hành động và giá trị trạng thái là hàm ưu điểm hành động (A-value):

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$$

## Giá trị tối ưu

Hàm giá trị tối ưu tạo ra hoàn trả tối đa:

$$V_*(s) = \max_{\pi} V_{\pi}(s)$$

$$Q_*(s, a) = \max_{\pi} Q_{\pi}(s, a)$$

Chính sách tối ưu đạt được các hàm giá trị tối ưu:

$$\pi_* = \arg \max_{\pi} V_{\pi}(s)$$

$$\pi_* = \arg \max_{\pi} Q_{\pi}(s, a)$$

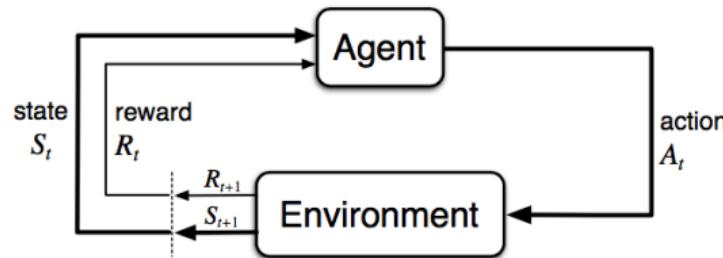
Hay là  $V_{\pi_*} = V_*(s)$  và  $Q_{\pi_*}(s, a) = Q_*(s, a)$ .

## Quá trình quyết định Markov

Hầu hết tất cả các vấn đề trong Học tăng cường có thể được trình bày dưới dạng Quá trình quyết định Markov (Markov Decision Processes - MDPs). Tất cả các trạng thái trong MDP đều có tính chất “Markov”, đề cập đến việc tương lai chỉ phụ thuộc vào trạng thái hiện tại, không phụ thuộc vào lịch sử:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, \dots, S_t]$$

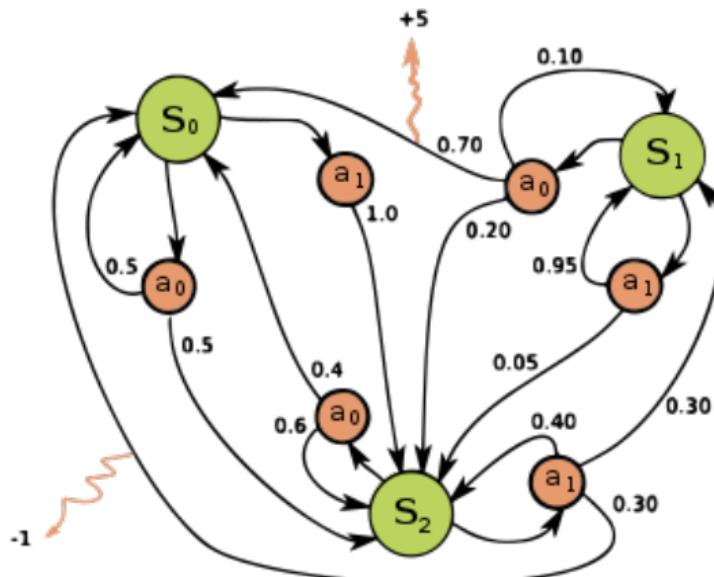
Hoặc nói cách khác, tương lai và quá khứ độc lập có điều kiện dựa vào hiện tại, vì trạng thái hiện tại bao gồm tất cả các thông kê chúng ta cần để quyết định về tương lai.



Hình 120: Tương tác của tác tử - môi trường dưới dạng quá trình quyết định Markov

Một quá trình quyết định Markov gồm 5 thành phần  $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$  với các ý nghĩa tương tự như trong Học tăng cường.

- $\mathcal{S}$  là tập các trạng thái.
- $\mathcal{A}$  là tập các hành động.
- $P$  là hàm xác suất chuyển đổi.
- $R$  là hàm phần thưởng.
- $\gamma$  là hệ số giảm giá cho phần thưởng trong tương lai.



Hình 121: Minh họa quá trình quyết định Markov

## Các phương pháp tiếp cận chung

Các cách tiếp cận chính và các thuật toán cổ điển để giải các bài toán RL bao gồm:

- Quy hoạch động (Dynamic Programming).
- Phương pháp Monte - Carlo (Monte - Carlo Methods).
- Học tập khác biệt theo thời gian (Temporal Difference Learning).
- Chính sách gradient (Policy Gradient).
- Chiến lược tiến hóa (Evolution Strategies).

Trong phần này, tập trung vào Phương pháp Monte - Carlo và Chính sách gradient.

## Phương pháp Monte - Carlo

Dầu tiên, nhắc lại  $V(s) = \mathbb{E}[G_t | S_t = s]$ . Phương pháp Monte - Carlo (MC) sử dụng một ý tưởng đơn giản: Nó học từ các tập (episode) tương tác trực tiếp mà không cần mô hình hóa sự thay đổi của môi trường và tính trung bình hoàn trả quan sát được như một xấp xỉ của kỳ vọng hoàn trả.

Để tính toán hoàn trả thực nghiệm  $G_t$ , các phương pháp MC cần học từ các tập hoàn chỉnh  $S_1, A_1, R_1, \dots, S_T$  để tính toán  $G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$  và tất cả các tập phải kết thúc.

Trung bình hoàn trả thực nghiệm cho trạng thái  $s$  là:

$$V(s) = \frac{\sum_{t=1}^T \mathbb{1}[S_t = s] G_t}{\sum_{t=1}^T \mathbb{1}[S_t = s]}$$

Trong đó  $\mathbb{1}[\cdot]$  là một hàm chỉ số nhị phân. Chúng ta có thể đếm sự ghé thăm của trạng thái  $s$  mỗi lần để có thể có nhiều lần ghé thăm trạng thái trong một tập (every-visit), hoặc chỉ đếm lần đầu tiên chúng ta gặp trạng thái trong một tập (first-visit).

Cách xấp xỉ này có thể dễ dàng mở rộng sang các hàm giá trị hành động bằng cách đếm cặp  $(s, a)$ .

$$Q(s, a) = \frac{\sum_{t=1}^T \mathbb{1}[S_t = s, A_t = a] G_t}{\sum_{t=1}^T \mathbb{1}[S_t = s, A_t = a]}$$

Để học chính sách tối ưu bằng MC, chúng ta lặp lại nó bằng cách:

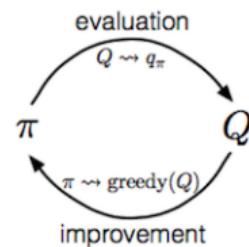
1. Cải thiện chính sách một cách tham lam với hàm giá trị hiện tại:

$$\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$$

2. Tạo ra một tập mới với chính sách mới  $\pi$ .

3. Ước tính  $Q$  bằng cách sử dụng tập mới:

$$Q_\pi(s, a) = \frac{\sum_{t=1}^T \mathbb{1}[S_t = s, A_t = a] \left( \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \right)}{\sum_{t=1}^T \mathbb{1}[S_t = s, A_t = a]}$$



Hình 122: Quy trình phương pháp Monte - Carlo

## Chính sách gradient

Hầu hết phương pháp đều nhắm mục tiêu học hàm giá trị trạng thái/hành động và sau đó chọn các hành động tương ứng. Các phương pháp Policy Gradient thay vào đó, học chính sách trực tiếp thông qua một hàm có tham số liên quan đến  $\theta$ , là  $\pi(a|s; \theta)$ .

Định nghĩa hàm phần thưởng  $\mathcal{J}(\theta)$  (ngược của hàm mất mát) là kỳ vọng của hoàn trả và huấn luyện thuật toán với mục tiêu là cực đại hàm phần thưởng.

Trong không gian rời rạc, với  $S_1$  là trạng thái ban đầu:

$$\mathcal{J}(\theta) = V_{\pi_\theta}(S_1) = \mathbb{E}_{\pi_\theta}(V_1)$$

Hoặc trong không gian liên tục:

$$\mathcal{J}(\theta) = \sum_{s \in \mathcal{S}} d_{\pi_\theta}(s) V_{\pi_\theta}(s) = \sum_{s \in \mathcal{S}} d_{\pi_\theta}(s) \left( \sum_{a \in \mathcal{A}} \pi(a|s; \theta) Q_\pi(s, a) \right)$$

Trong đó  $d_{\pi_\theta}(s)$  là phân phối cố định (stationary distribution) của xích Markov cho  $\pi_\theta$ .

# Định lý Chính sách gradient

Sử dụng thuật toán tăng gradient, chúng ta có thể tìm ra  $\theta$  tốt nhất để tạo ra hoàn trả cao nhất. Điều này dễ dàng hiểu rằng các phương pháp dựa trên chính sách hữu ích hơn trong không gian liên tục, bởi vì có một số vô hạn các hành động và/hoặc trạng thái để ước tính giá trị trong không gian liên tục và do đó phương pháp dựa trên giá trị đòi hỏi tính toán đắt đỏ hơn rất nhiều.

## Định lý: Chính sách gradient

Thừa nhận kết quả về gradient của hàm phần thưởng cho bởi công thức sau:

$$\nabla \mathcal{J}(\theta) = \mathbb{E}_{\pi_\theta} [\nabla \ln \pi(a|s; \theta) Q_\pi(s, a)]$$

# REINFORCE: Chính sách Monte - Carlo gradient

REINFORCE, hay Chính sách Monte - Carlo gradient (Monte - Carlo policy gradient), dựa vào  $Q_\pi(s, a)$ , một giá trị của hoàn trả ước lượng bằng phương pháp MC từ các mẫu của tập để cập nhật tham số chính sách  $\theta$ .

## Thuật toán 10: Chính sách Monte - Carlo gradient

1. Khởi tạo  $\theta$  một cách ngẫu nhiên.
2. Tạo một tập  $S_1, A_1, R_2, \dots, S_T$ .
3. Với  $t = 1, 2, \dots, T$  thực hiện:
  - 3.1 Ước lượng giá trị hoàn trả  $G_t$  kể từ thời điểm  $t$ .
  - 3.2 Cập nhật  $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla \ln \pi(A_t | S_t; \theta)$ .

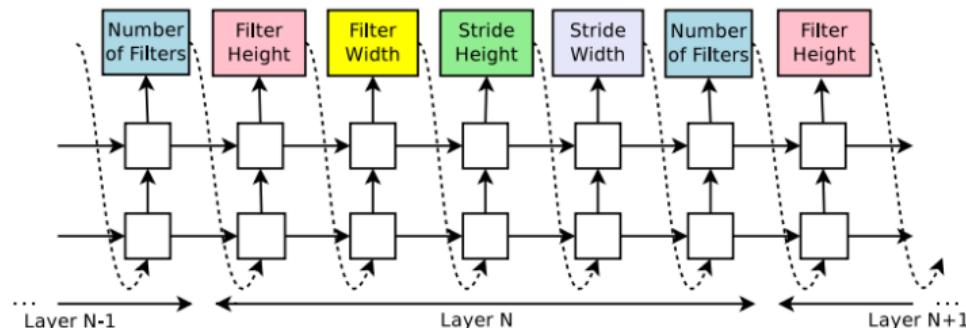
## Tạo mô tả mô hình

Đề xuất một phương pháp đơn giản để sử dụng mạng hồi quy tạo ra kiến trúc tích chập:

- Cách mạng hồi quy có thể được đào tạo với một chính sách gradient để tối đa hóa độ chính xác kỳ vọng của các kiến trúc được lấy mẫu.
- Một số cải tiến trong phương pháp nhằm tạo ra các kết nối tắt (skip connection) để tăng độ phức tạp cho mô hình và sử dụng phương pháp máy chủ tham số (parameter server) để tăng tốc độ huấn luyện cho mô hình.

Trong Tìm kiếm kiến trúc mạng nơ-ron, chúng ta sử dụng một bộ điều khiển (controller) để tạo ra các siêu tham số kiến trúc của mạng. Để linh hoạt, bộ điều khiển được triển khai như một mạng nơ-ron hồi quy.

Giả sử muốn dự đoán các mạng nơ-ron truyền thẳng chỉ với các lớp tích chập, chúng ta có thể sử dụng bộ điều khiển để tạo ra các siêu tham số của chúng dưới dạng chuỗi các token:



**Hình 123:** Cách mạng nơ-ron hồi quy điều khiển làm mẫu về một mạng tích chập đơn giản. Nó dự đoán chiều cao bộ lọc, chiều rộng bộ lọc, chiều cao bước nhảy, chiều rộng bước nhảy, và số lượng bộ lọc cho một lớp và lặp lại. Mỗi dự đoán được thực hiện bởi một phân loại softmax và sau đó được đưa vào bước thời gian tiếp theo làm đầu vào.

- Quá trình tạo ra một kiến trúc sẽ dừng lại nếu số lượng lớp vượt quá một giá trị nhất định. Ngưỡng số lượng lớp này tăng lên theo thời gian trong quá trình huấn luyện, theo một lịch trình cụ thể.
- Sau khi bộ điều khiển RNN tạo ra kiến trúc CNN, mạng nơ-ron được xây dựng và huấn luyện dựa trên kiến trúc đó. Khi hội tụ, độ chính xác của mạng được đánh giá trên một tập dữ liệu kiểm tra độc lập và ghi chép lại.
- Tham số của bộ điều khiển RNN,  $\theta_c$ , được tối ưu hóa để tối đa hóa độ chính xác kiểm tra kỳ vọng của các kiến trúc được đề xuất.
- Một phương pháp gradient chính sách được sử dụng để cập nhật tham số  $\theta_c$ , nhằm mục đích tạo ra các kiến trúc tốt hơn qua thời gian.

# Huấn luyện với REINFORCE

Danh sách các token mà bộ điều khiển dự đoán có thể được xem như là một loạt các hành động từ  $a_{1:T}$  để thiết kế một kiến trúc cho mạng con. Khi hội tụ, mạng con này sẽ đạt được một độ chính xác  $R$  trên một tập dữ liệu được giữ riêng biệt. Chúng ta có thể sử dụng độ chính xác  $R$  này như là tín hiệu thưởng và sử dụng học tăng cường để huấn luyện bộ điều khiển.

Cụ thể hơn, để tìm kiến trúc tối ưu, chúng ta yêu cầu bộ điều khiển của mình tối đa hóa phần thưởng kỳ vọng của nó, được biểu diễn bởi  $J(\theta_c)$ :

$$J(\theta_c) = \mathbb{E}_{P(a_{1:T}; \theta_c)}[R]$$

Vì tín hiệu thưởng  $R$  không khả vi nên cần sử dụng một phương pháp gradient chính sách để lặp lại việc cập nhật  $\theta_c$ . Trong công trình này, chúng ta sử dụng quy tắc REINFORCE<sup>157</sup>:

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^T \mathbb{E}_{P(a_{1:T}; \theta_c)} [\nabla_{\theta_c} \ln P(a_t | a_{(t-1):1}; \theta_c) R]$$

Một ước lượng thực nghiệm của đại lượng trên là:

$$\frac{1}{m} \sum_{k=1}^m \sum_{t=1}^T \nabla_{\theta_c} \ln P(a_t | a_{(t-1):1}; \theta_c) R_k$$

Trong đó  $m$  là số lượng kiến trúc khác nhau mà bộ điều khiển lấy mẫu trong một lô và  $T$  là số lượng siêu tham số mà bộ điều khiển phải dự đoán để thiết kế một kiến trúc mạng nơ-ron. Độ chính xác kiểm tra mà kiến trúc mạng nơ-ron thứ  $k$  đạt được sau khi được huấn luyện trên tập dữ liệu huấn luyện là  $R_k$ .

---

<sup>157</sup>Simple statistical gradient-following algorithms for connectionist reinforcement learning

Cập nhật trên đây là một ước tính không thiên lệch (unbiased) cho gradient, nhưng có phương sai rất lớn. Để giảm phương sai của ước tính này, chúng ta sử dụng một hàm cơ sở  $b$  (baseline function):

$$\frac{1}{m} \sum_{k=1}^m \sum_{t=1}^T \nabla_{\theta_c} \ln P(a_t | a_{(t-1):1}; \theta_c) (R_k - b)$$

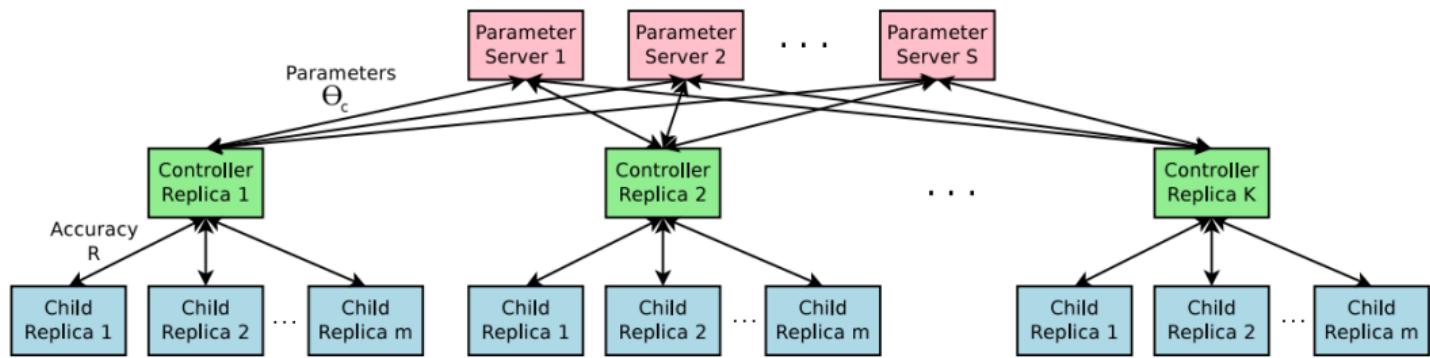
Miễn là hàm cơ sở  $b$  không phụ thuộc vào hành động hiện tại, thì đây vẫn là một ước tính gradient không thiên lệch. Trong công trình này, hàm cơ sở  $b$  được chọn là một trung bình trượt theo hình mũ (exponential moving average) của độ chính xác của các kiến trúc trước đó.

- Hàm cơ sở giúp giảm phương sai của ước lượng gradient bằng cách cung cấp một điểm tham chiếu ổn định để so sánh phần thưởng (hoặc độ chính xác) nhận được từ mỗi hành động. Làm cho quá trình học được ổn định hơn, giúp thuật toán hội tụ nhanh hơn.
- Mặc dù giảm phương sai, nhưng hàm cơ sở không làm thay đổi kỳ vọng của ước lượng gradient, đảm bảo rằng ước lượng vẫn không thiên lệch.
- Hàm cơ sở có thể giúp thuật toán tập trung vào lợi ích dài hạn bằng cách loại bỏ nhiều ngẫu nhiên và sự biến động do các phần thưởng cực đoan trong ngắn hạn.

## Kỹ thuật Song song hóa và Cập nhật bất đồng bộ

Trong kỹ thuật này, mỗi cập nhật gradient đối với các tham số điều khiển  $\theta_c$  tương đương với việc đào tạo một mạng con riêng lẻ đến khi hội tụ. Do việc đào tạo một mạng con có thể mất hàng giờ, chúng ta sử dụng **phương pháp đào tạo phân tán** và **cập nhật tham số bất đồng bộ** để tăng tốc quá trình học của bộ điều khiển.

- Sử dụng một cơ chế máy chủ tham số nơi chúng tôi có một máy chủ tham số của  $S$  phân đoạn (shards), lưu trữ các tham số chia sẻ cho  $K$  bản sao điều khiển (controller replica).
- Mỗi bản sao điều khiển lấy mẫu  $m$  kiến trúc con khác nhau để huấn luyện song song.
- Bộ điều khiển sau đó thu thập gradient dựa trên kết quả về độ chính xác từ minibatch của  $m$  kiến trúc khi hội tụ và gửi chúng đến máy chủ tham số để cập nhật trọng số  $\theta_c$  cho tất cả bản sao điều khiển.
- Trong triển khai này, hội tụ của mỗi mạng con được đạt đến khi đào tạo vượt qua một số lượng vòng lặp nhất định.



Hình 124: Tóm tắt cơ chế song song hóa

## Tăng độ phức tạp kiến trúc bằng kết nối tắt và loại lớp khác

Phần trước, không gian tìm kiếm không có các kết nối tắt (skip connection), hay các lớp nhánh (branching layer) được sử dụng trong các kiến trúc hiện đại như GoogLeNet và Residual Net.

Phần này, chúng ta giới thiệu một phương pháp cho phép bộ điều khiển đề xuất các kết nối tắt hoặc lớp nhánh, từ đó mở rộng không gian tìm kiếm.

Để cho phép bộ điều khiển dự đoán những kết nối này, chúng ta sử dụng **Cơ chế chú ý kiểu chọn lựa tập hợp**<sup>158</sup> (set-selection type attention) dựa trên **Cơ chế chú ý**<sup>159,160</sup> (attention mechanism).

---

<sup>158</sup> Neural programmer: Inducing latent programs with gradient descent

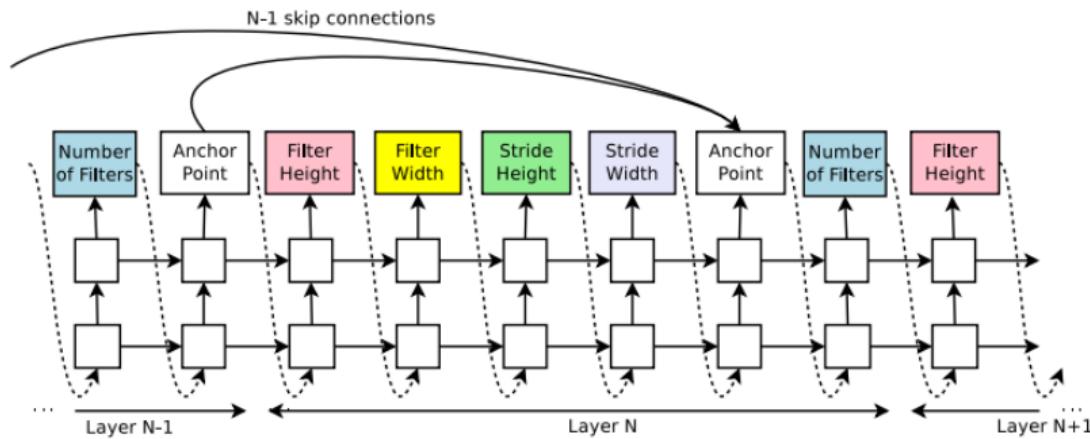
<sup>159</sup> Neural machine translation by jointly learning to align and translate

<sup>160</sup> Pointer networks

Tại tầng  $N$ , chúng tôi thêm một điểm neo có  $N - 1$  sigmoid dựa trên nội dung (content-based sigmoid) để chỉ ra các tầng trước đó cần được kết nối. Mỗi sigmoid là một hàm số của trạng thái ẩn hiện tại của bộ điều khiển và các trạng thái ẩn trước đó của  $N - 1$  điểm neo:

$$\mathbb{P}[\text{lớp } j \text{ là một đầu vào của lớp } i] = \text{sigmoid} (v^\top \tanh(W_{prev} \cdot h_j + W_{curr} \cdot h_i))$$

Trong đó,  $h_j$  đại diện cho trạng thái ẩn của bộ điều khiển tại điểm neo cho lớp thứ  $j$  với  $j = \overline{0, N - 1}$ . Sau đó chúng ta lấy mẫu từ các hàm sigmoid này để quyết định lớp nào trước đó sẽ được sử dụng làm đầu vào cho lớp hiện tại. Các ma trận  $W_{prev}, W_{curr}, v$  là các tham số có thể học được.



**Hình 125:** Cách bộ điều khiển sử dụng các kết nối tắt để quyết định lớp nào nó muốn nhận làm đầu vào cho lớp hiện tại

Trong khuôn khổ này, nếu một lớp có nhiều lớp đầu vào thì tất cả lớp đầu vào đó sẽ được nối liền với nhau theo chiều sâu. Các kết nối tắt có thể gây ra “sự cố biên dịch” nơi một lớp không tương thích với lớp khác, hoặc một lớp có thể không có đầu vào hoặc đầu ra nào. Để giải quyết những vấn đề này, chúng ta sử dụng 3 kỹ thuật đơn giản:

- Nếu một lớp không được kết nối với bất kỳ lớp đầu vào nào thì hình ảnh sẽ được sử dụng làm lớp đầu vào.
- Tại lớp cuối cùng chúng tôi lấy tất cả đầu ra của lớp chưa được kết nối và nối chúng lại trước khi gửi trạng thái ẩn cuối cùng này đến bộ phân loại.
- Nếu các lớp đầu vào được nối liền với nhau có kích thước khác nhau, chúng ta sẽ chèn thêm các số không vào các lớp nhỏ hơn để tất cả lớp nối liền có cùng kích thước.

Phần trước, chúng ta không dự đoán tốc độ học và giả định rằng các kiến trúc chỉ bao gồm các lớp tích chập, điều này khá hạn chế. Có thể thêm tốc độ học vào như một dự đoán.

Ngoài ra, cũng có thể dự đoán các lớp: gộp (pooling), chuẩn hóa tương phản địa phương<sup>161,162</sup> (local contrast normalization), và chuẩn hóa theo lô<sup>163</sup> (batchnorm). Để có thể thêm nhiều loại lớp hơn, cần thêm một bước vào bộ điều khiển RNN để dự đoán loại lớp, sau đó là các siêu tham số liên quan đến nó.

---

<sup>161</sup>What is the best multi-stage architecture for object recognition?

<sup>162</sup>Imagenet classification with deep convolutional neural networks

<sup>163</sup>Batch normalization: Accelerating deep network training by reducing internal covariate shift

## Tạo kiến trúc tế bào hồi quy

Chúng ta sẽ chỉnh sửa phương pháp trên để tạo ra các tế bào hồi quy (recurrent cell). Tại mỗi bước thời gian  $t$ , bộ điều khiển cần tìm một biểu diễn cho  $h_t$  mà lấy  $x_t$  và  $h_{t-1}$  làm đầu vào. Cách đơn giản nhất để có  $h_t$  là:

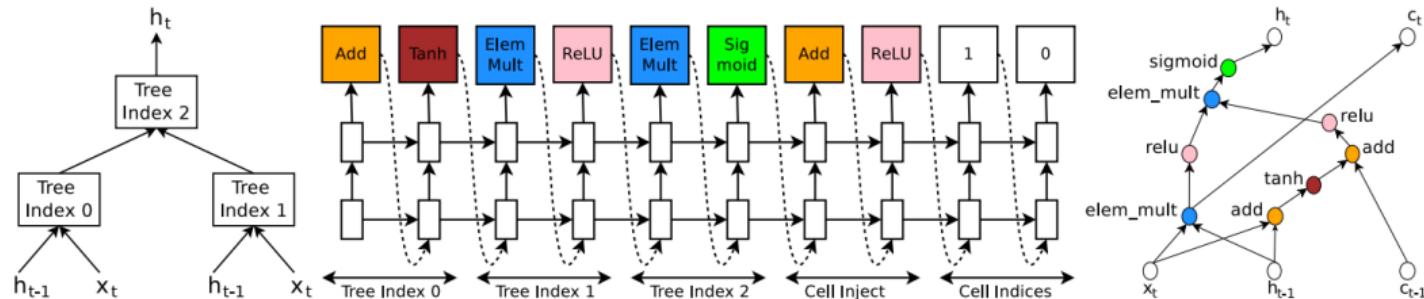
$$h_t = \tanh(W_1 \cdot x_t + W_2 \cdot h_{t-1})$$

Đây là công thức của một tế bào hồi quy cơ bản. Một công thức phức tạp hơn là tế bào hồi quy trí nhớ ngắn dài hạn (LSTM) được sử dụng rộng rãi.

Các phép toán cho tế bào RNN cơ bản và tế bào LSTM có thể được khái quát hóa như một cây của các bước thực hiện mà lấy  $x_t$  và  $h_{t-1}$  làm đầu vào và tạo ra  $h_t$  là đầu ra cuối cùng.

- Bộ điều khiển RNN cần đánh dấu mỗi nút trong cây với một toán tử kết hợp (cộng, nhân từng phần tử,...) và một hàm kích hoạt (tanh, sigmoid,...) để kết hợp 2 đầu vào và tạo ra 1 đầu ra.
- 2 đầu ra sau đó được cung cấp làm đầu vào cho nút tiếp theo trong cây.
- Để cho phép bộ điều khiển RNN lựa chọn các toán tử và hàm, ta đánh số các nút trong cây theo một thứ tự nhất định để bộ điều khiển RNN có thể ghé thăm mỗi nút một cách tuần tự và gắn nhãn các siêu tham số cần thiết.

- Lấy cảm hứng từ việc tạo ra tế bào LSTM, chúng ta cũng cần biến  $c_{t-1}$  và  $c_t$  để đại diện cho các trạng thái bộ nhớ.
- Để kết hợp các biến này, ta cần bộ điều khiển RNN để dự đoán nút nào trong cây sẽ kết nối 2 biến này lại với nhau.
- Những dự đoán này có thể được thực hiện trong 2 khối (block) cuối cùng của bộ điều khiển RNN.



**Hình 126:** Ví dụ về tế bào hồi quy được xây dựng từ cây có 2 nút lá (hệ cơ sở 2) và 1 nút nội bộ

- Bên trái: cây định nghĩa các bước tính toán cần được dự đoán bởi bộ điều khiển.
- Ở giữa: một tập hợp ví dụ các dự đoán được thực hiện bởi bộ điều khiển cho mỗi bước tính toán trong cây.
- Bên phải: đồ thị tính toán của tế bào hồi quy được xây dựng từ các dự đoán ví dụ của bộ điều khiển.