



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



Giới thiệu về học sâu

Nội dung

- Nhắc lại về học máy
- Giới thiệu khái niệm Học sâu
- Mạng nơ ron nhân tạo
- Giải thuật lan truyền ngược
- Tổng kết

Nhắc lại về học máy

Học máy có giám sát

Functions \mathcal{F}

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

Training data

$$\{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}\}$$

LEARNING



find $\hat{f} \in \mathcal{F}$
s.t. $y_i \approx \hat{f}(x_i)$



Learning machine

PREDICTION

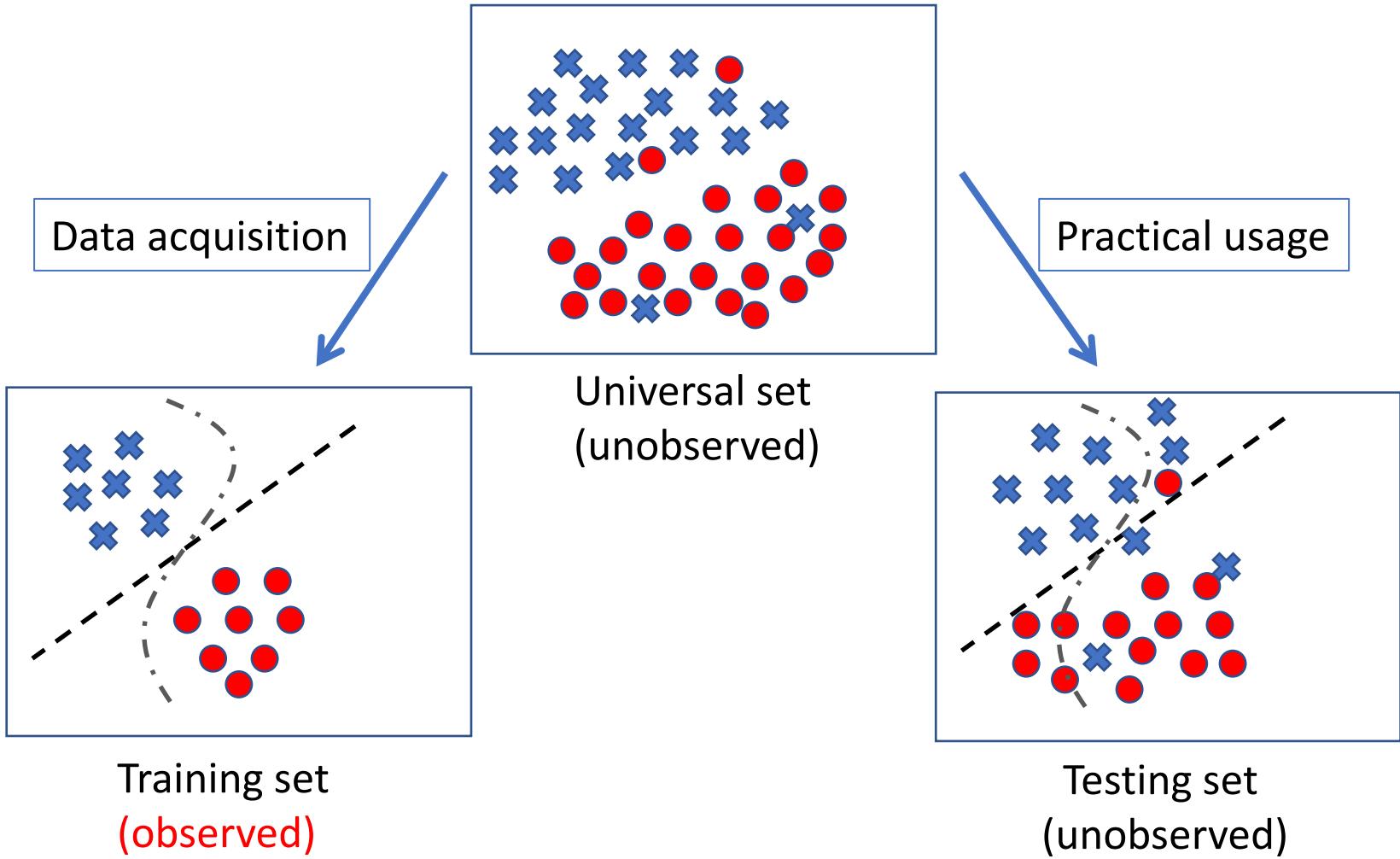
$y = \hat{f}(x)$



New data

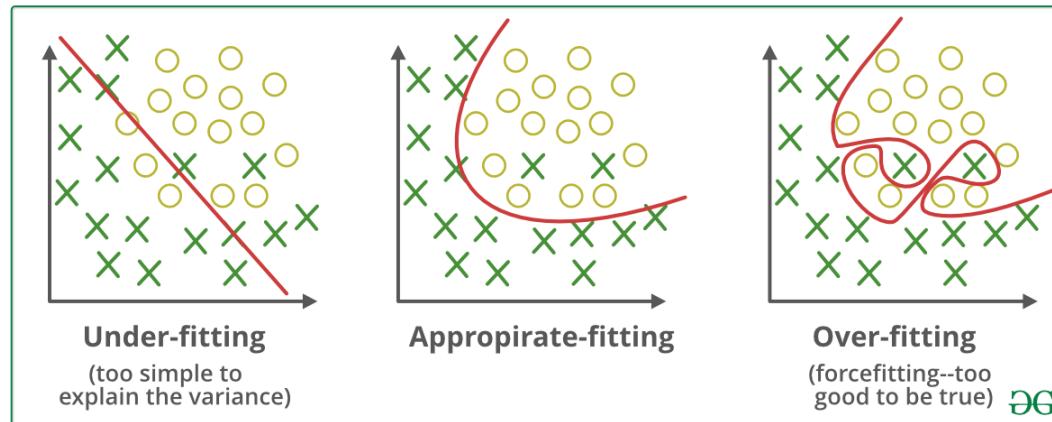
x

Tập huấn luyện và tập kiểm tra



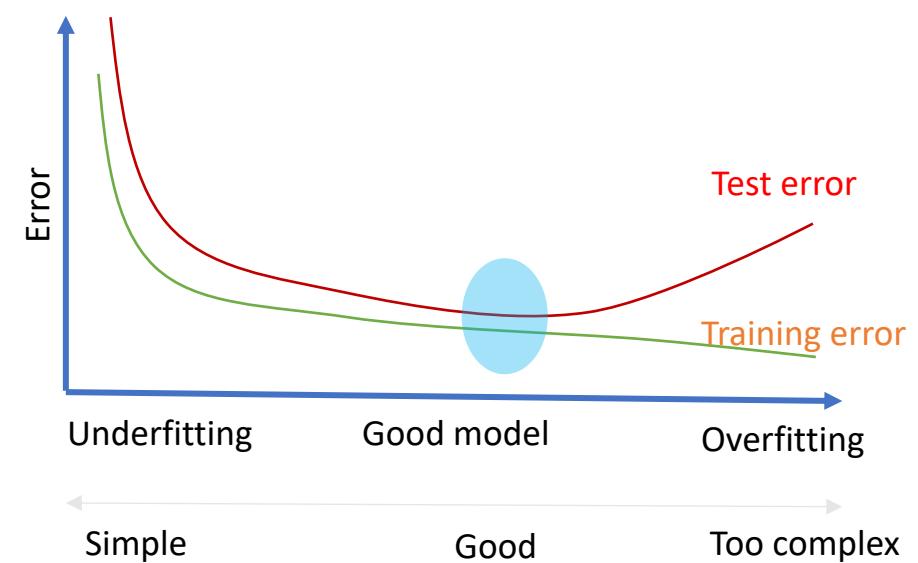
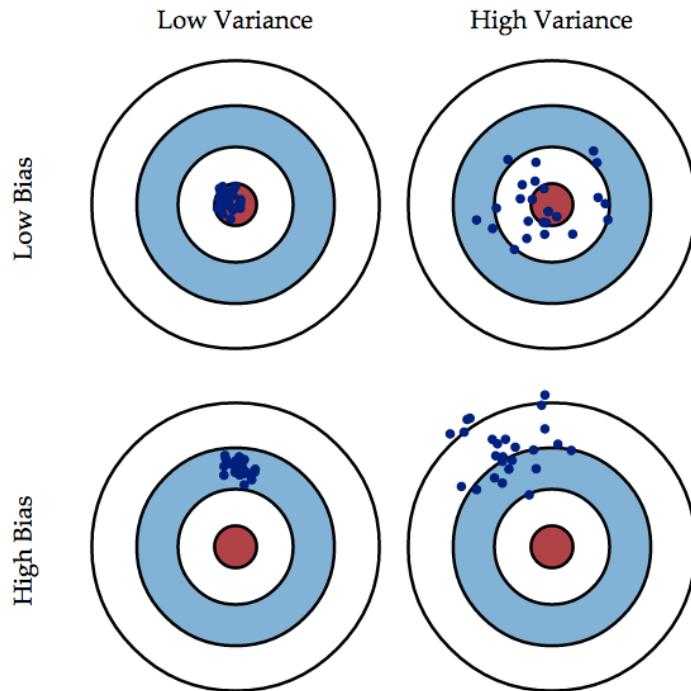
Hiện tượng overfit và underfit

- **Underfitting:** mô hình quá “đơn giản” để biểu diễn các tính chất của dữ liệu
 - Bias cao và variance thấp
 - Sai số cao trên tập huấn luyện và tập kiểm tra
- **Overfitting:** mô hình quá “phức tạp” dẫn tới học cả nhiễu trong dữ liệu
 - Bias thấp và variance cao
 - Sai số thấp trên tập huấn luyện và sai số cao trên tập kiểm tra



Bias và variance

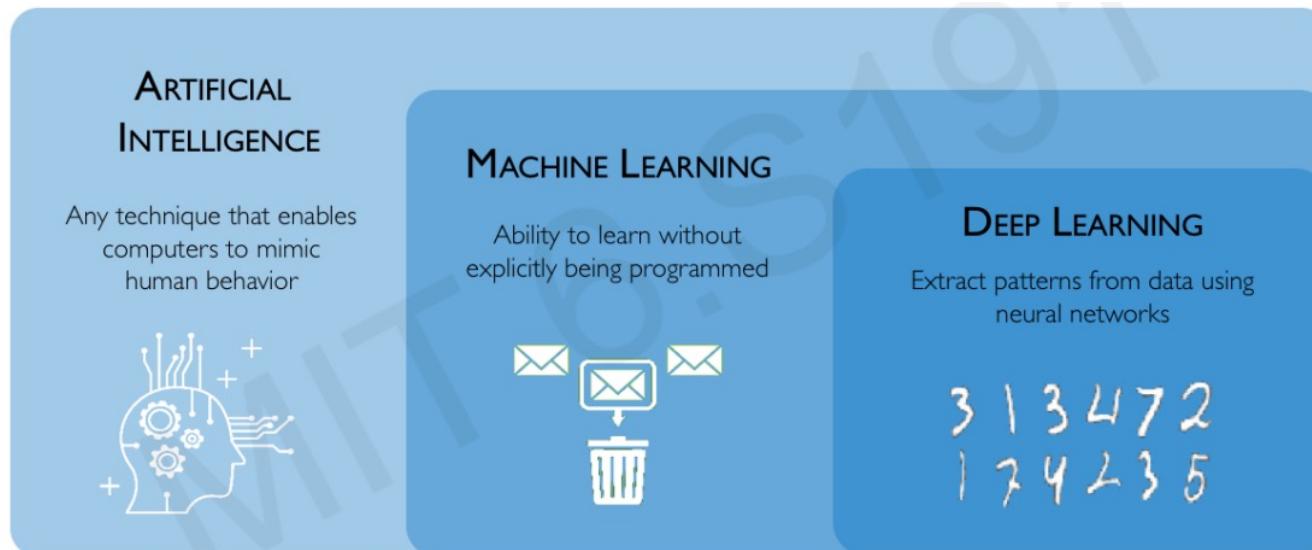
- Bias: Sự chênh lệch giữa giá trị trung bình dự đoán của các mô hình được huấn luyện trên các tập dữ liệu khác nhau so với giá trị thực tế của dữ liệu.
- Variance: Độ phân tán của dự đoán của các mô hình được huấn luyện trên các tập dữ liệu khác nhau



Giới thiệu khái niệm Học sâu

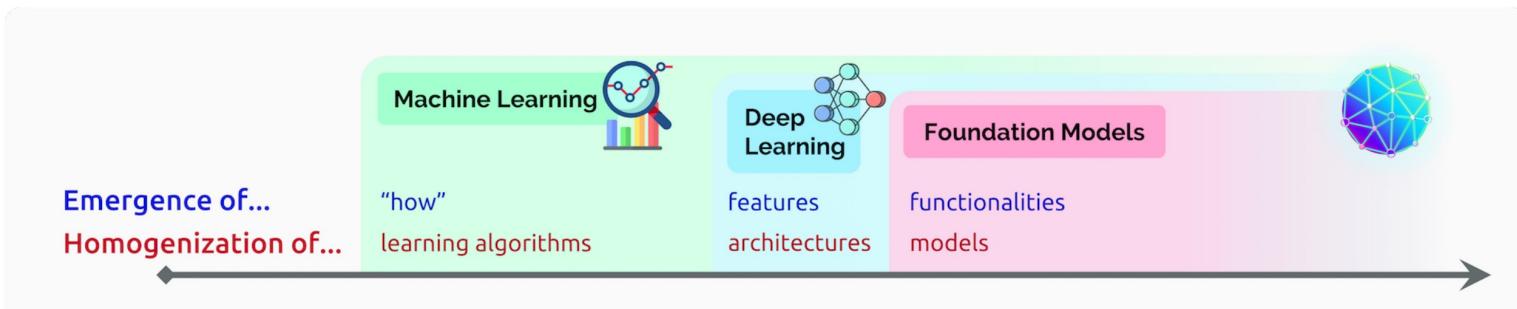
Học sâu và học máy

- Học máy (ML - Machine Learning) là một lĩnh vực nghiên cứu của Trí tuệ nhân tạo (Artificial Intelligence)
- Học sâu (Deep learning) là một nhánh của học máy
 - Là phương pháp học máy **sử dụng mạng nơ-ron nhân tạo** để trích xuất đặc trưng tự động từ dữ liệu

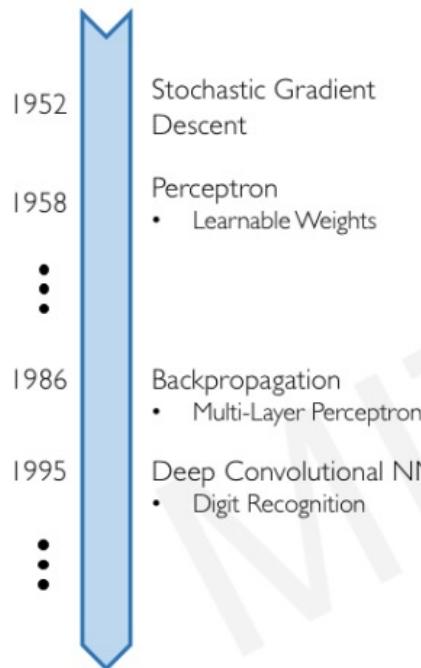


Mô hình nền tảng

- Lập trình truyền thống: Input + Program = Output
- Học máy: Input + Output = Program
 - Mỗi bài toán riêng cần tập dữ liệu riêng, phương pháp học máy riêng, thuật toán học riêng, trích xuất đặc trưng riêng
 - Học sâu: Mỗi bài toán cần tập dữ liệu riêng, phương pháp học máy chung (mạng nơ-ron) nhưng kiến trúc riêng, thuật toán học chung
 - **Mô hình nền tảng:** dữ liệu chung, mô hình chung, thuật toán chung → Đồng nhất hóa AI



Tại sao giờ mới bùng nổ học sâu?



Neural Networks date back decades, so why the resurgence?

I. Big Data

- Larger Datasets
- Easier Collection & Storage

IMGENET



2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable

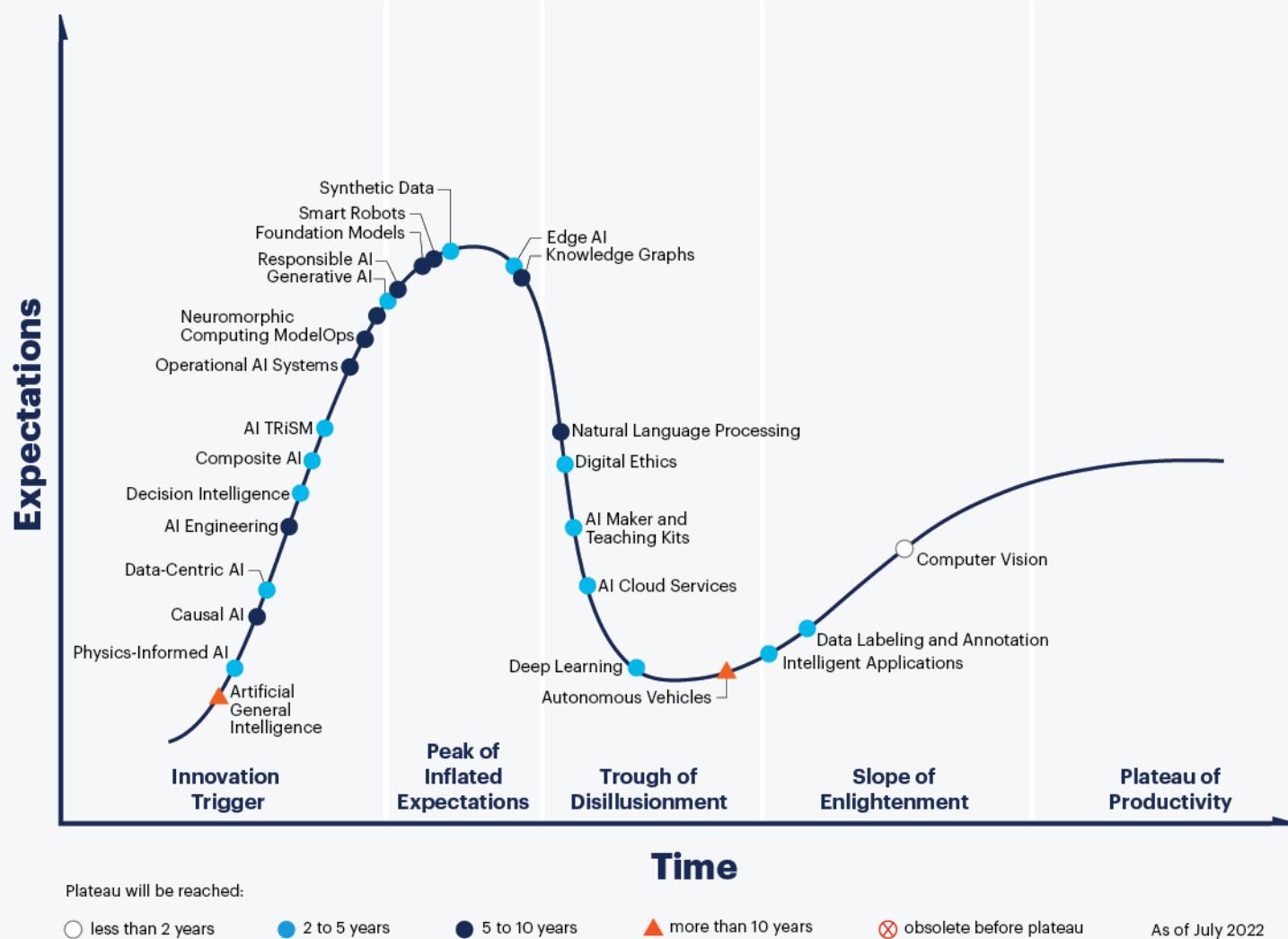


3. Software

- Improved Techniques
- New Models
- Toolboxes



Hype Cycle for Artificial Intelligence, 2022



gartner.com

Source: Gartner
© 2022 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner and Hype Cycle are registered trademarks of Gartner, Inc. and its affiliates in the U.S. 1957302

Hype Cycle for Artificial Intelligence, 2023



gartner.com

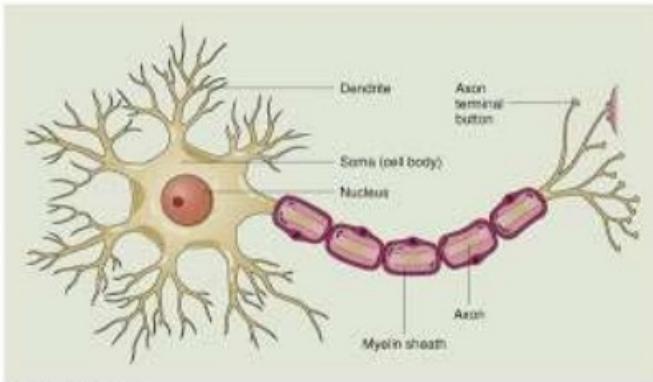
Source: Gartner
© 2023 Gartner, Inc. and/or its affiliates. All rights reserved. 2079794

Gartner®

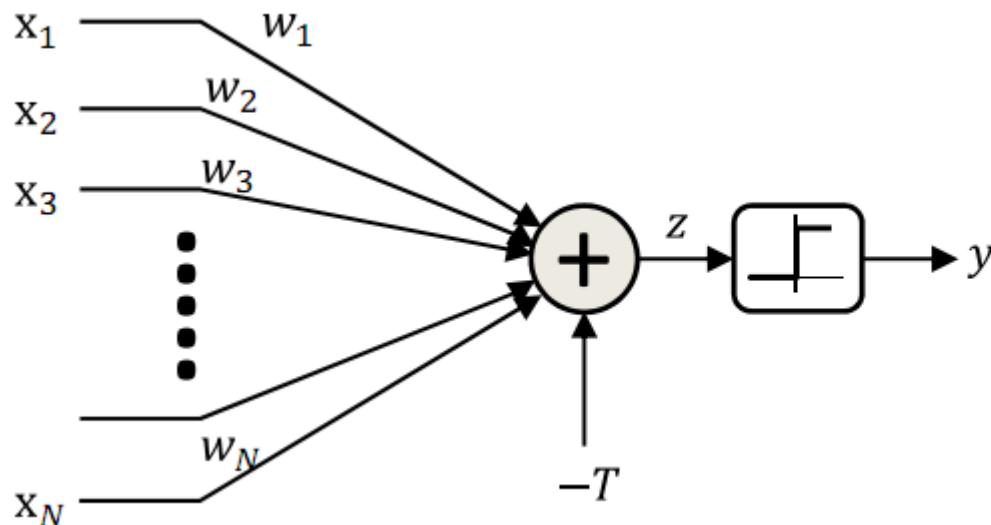
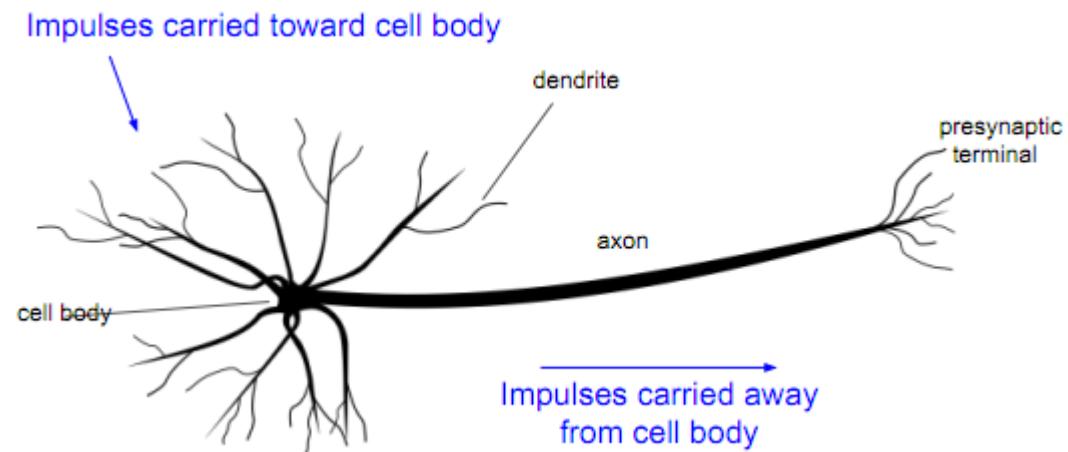
Mạng nơ ron nhân tạo

Mạng nơ-ron và bộ não

- Mạng nơ-ron mô phỏng cấu trúc kết nối của não người



Perceptron

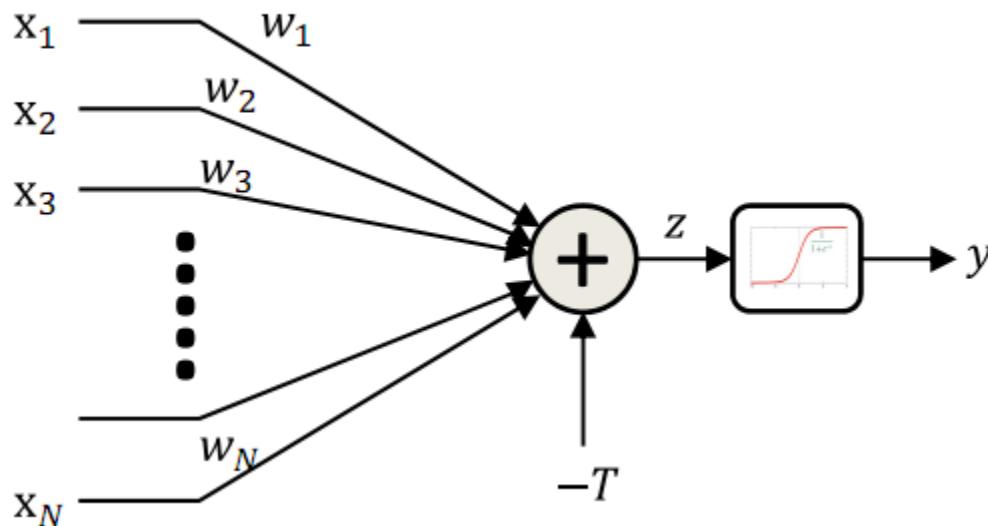


$$z = \sum_i w_i x_i - T$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{else} \end{cases}$$

- Bắn xung “fire” nếu tổng có trọng số của các đầu vào với “bias” T không âm

Perceptron mềm (logistic)

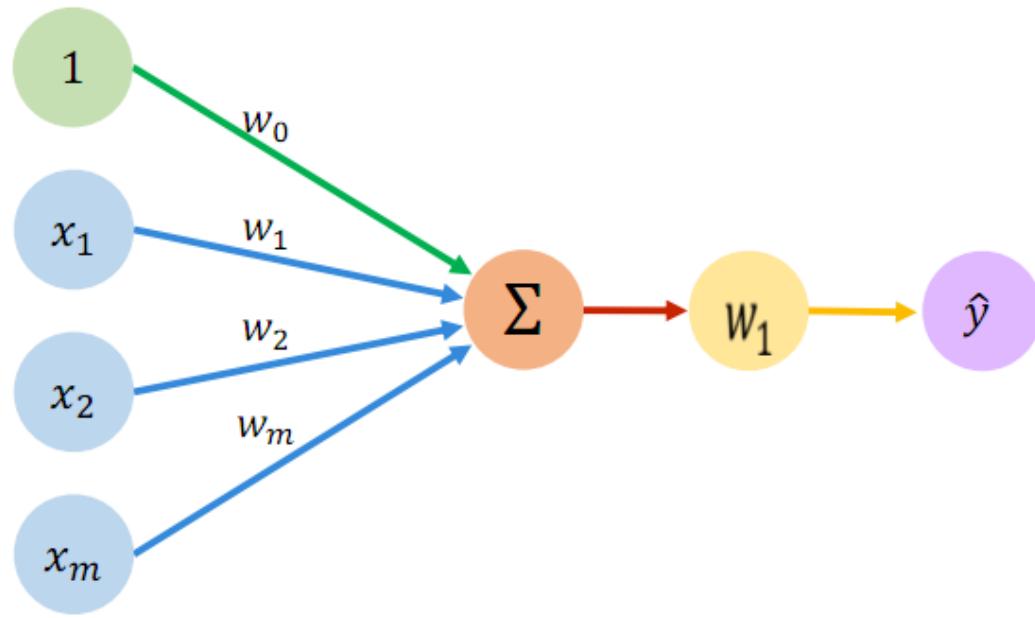


$$z = \sum_i w_i x_i - T$$

$$y = \frac{1}{1 + \exp(-z)}$$

- Sử dụng một hàm khả vi thay cho hàm xung
- Hàm kích hoạt sigmoid được dùng để xấp xỉ hàm xung
- Hàm kích hoạt là hàm tác động lên tổng có trọng số của các dữ liệu vào

Perceptron mềm (logistic)



Linear combination of inputs \downarrow

Output \downarrow

$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$

Non-linear activation function \uparrow

Bias \uparrow

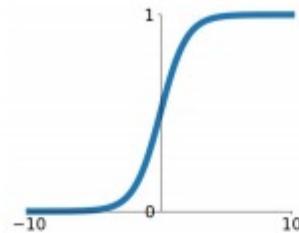
Diagram illustrating the mathematical formula for the output of a logistic perceptron. The output \hat{y} is calculated as the non-linear activation function g applied to the linear combination of inputs. The linear combination includes a bias term w_0 and the weighted sum of inputs $x_i w_i$.

Inputs Weights Sum Non-Linearity Output

Một số hàm kích hoạt thường gặp

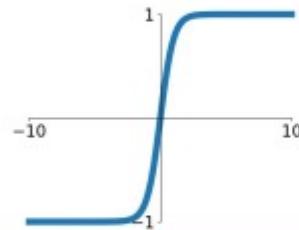
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



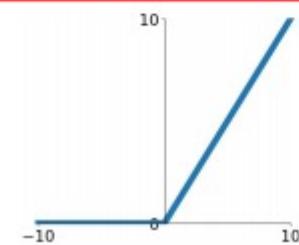
tanh

$$\tanh(x)$$



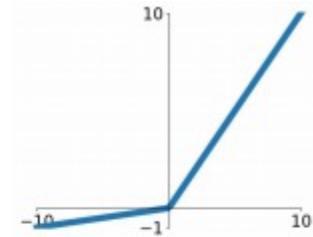
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

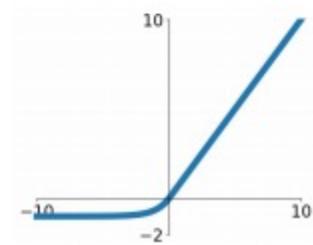


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

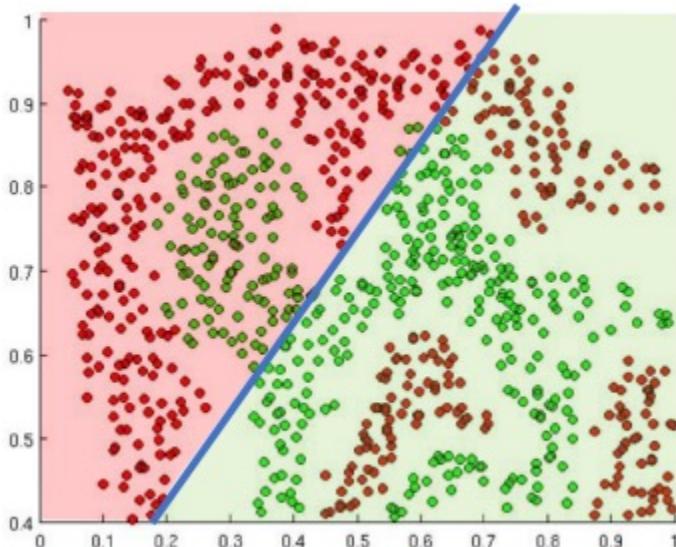
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



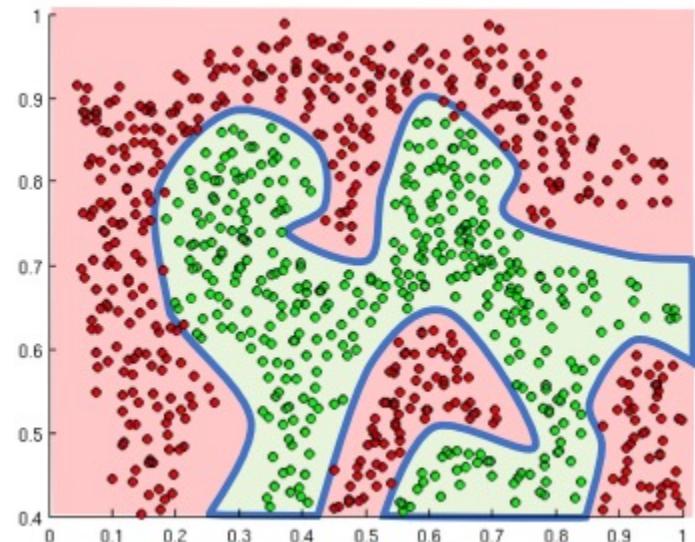
- ReLU là lựa chọn mặc định tốt cho nhiều bài toán
- Hiện nay xu hướng dùng một số hàm kích hoạt hiện đại hơn như ReLU6, swish, mish

Tâm quan trọng của hàm kích hoạt

- Mục đích sử dụng hàm kích hoạt là đưa các lớp phi tuyến vào mạng nơ-ron

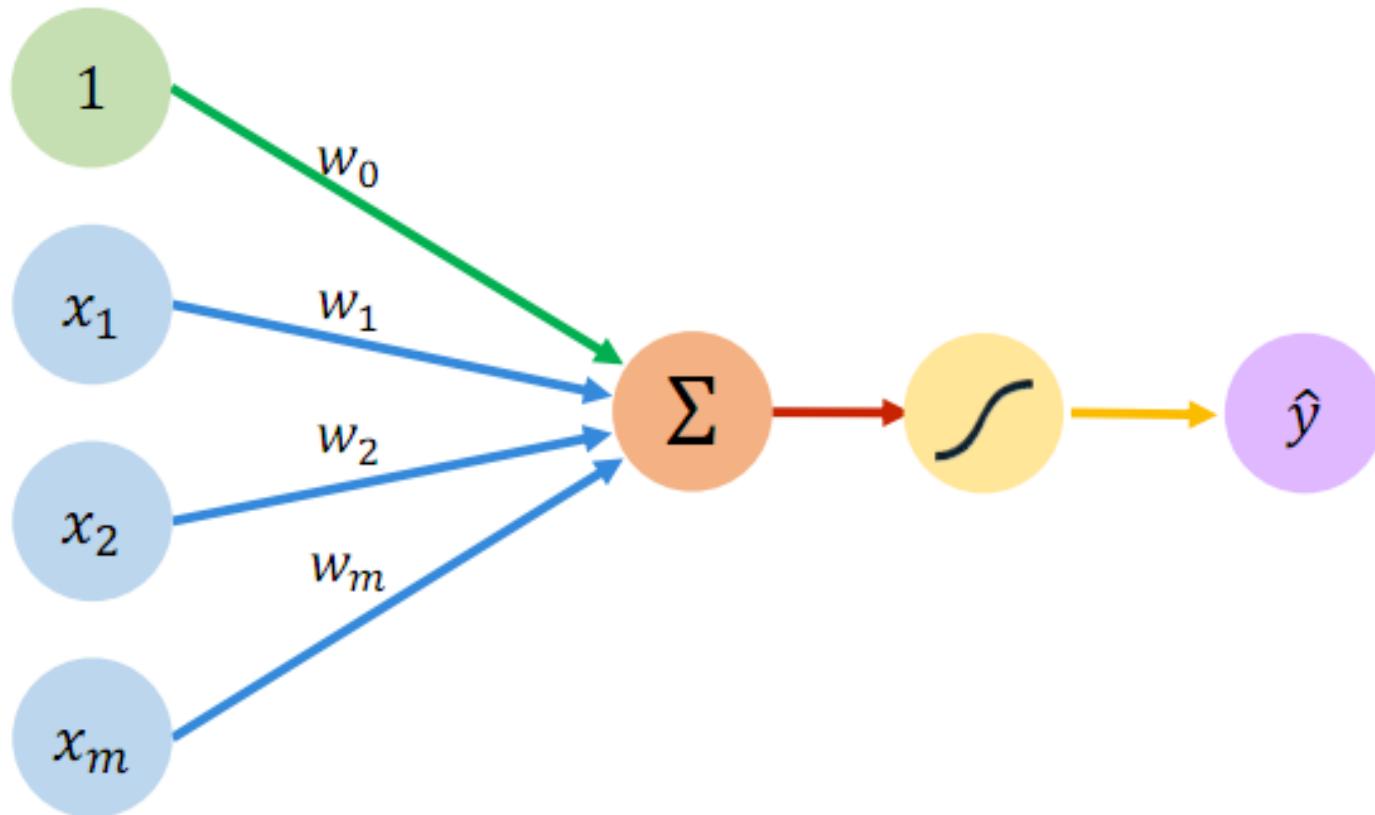


Hàm kích hoạt tuyến tính
luôn sinh ra đường phân
cách tuyến tính bất kể
mạng có lớn cỡ nào



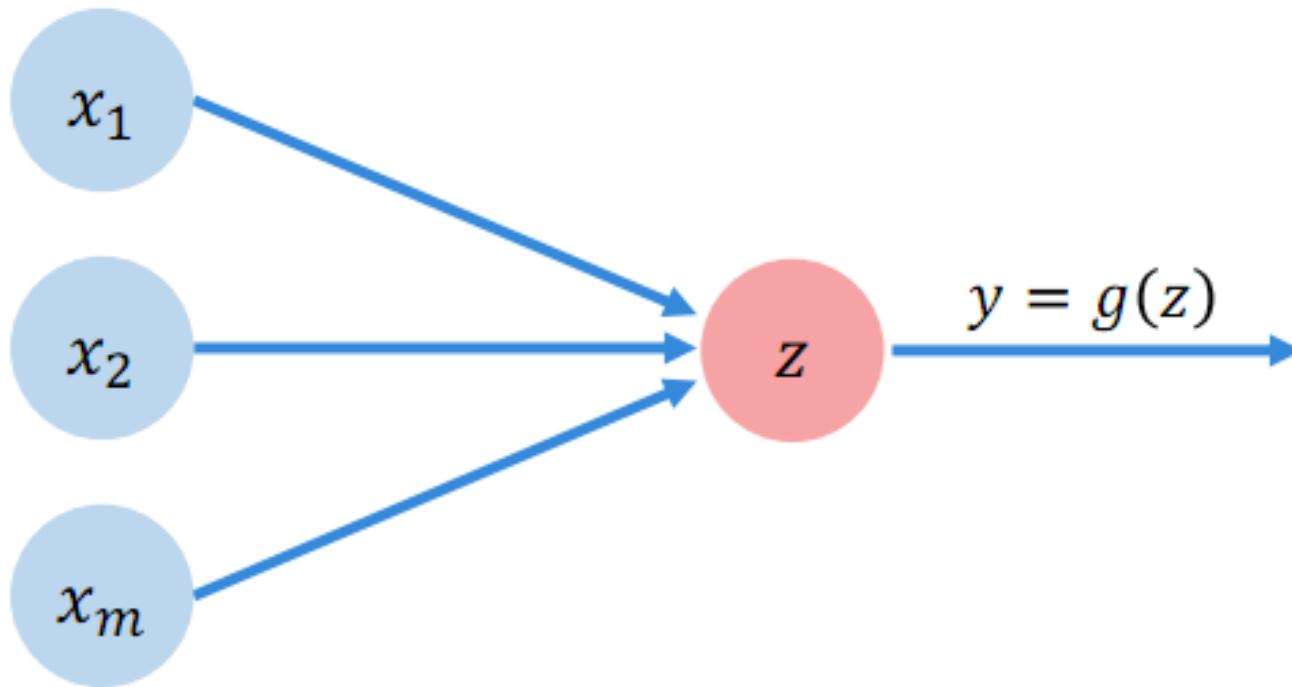
Các lớp phi tuyến cho
phép chúng ta xấp xỉ các
hàm phức tạp

Perceptron đơn giản hóa



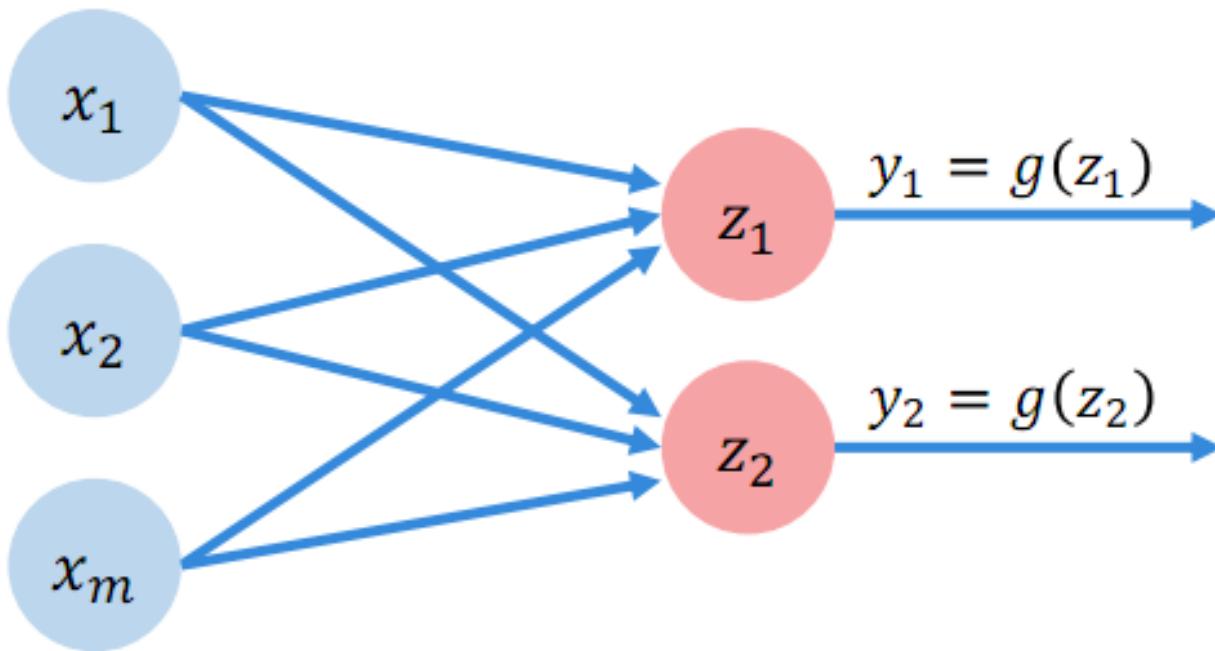
Inputs	Weights	Sum	Non-Linearity	Output
--------	---------	-----	---------------	--------

Perceptron đơn giản hóa



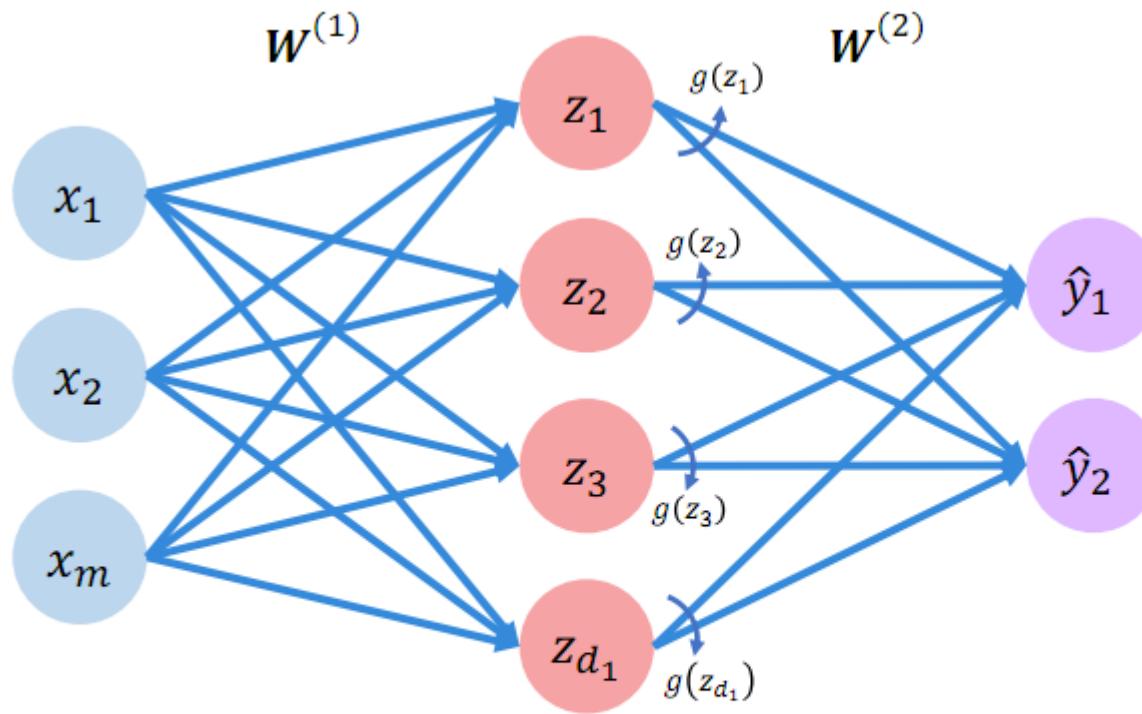
$$z = w_0 + \sum_{j=1}^m x_j w_j$$

Perceptron nhiều đầu ra



$$z_i = w_{0,i} + \sum_{j=1}^m x_j w_{j,i}$$

Mạng nơ-ron một lớp ẩn



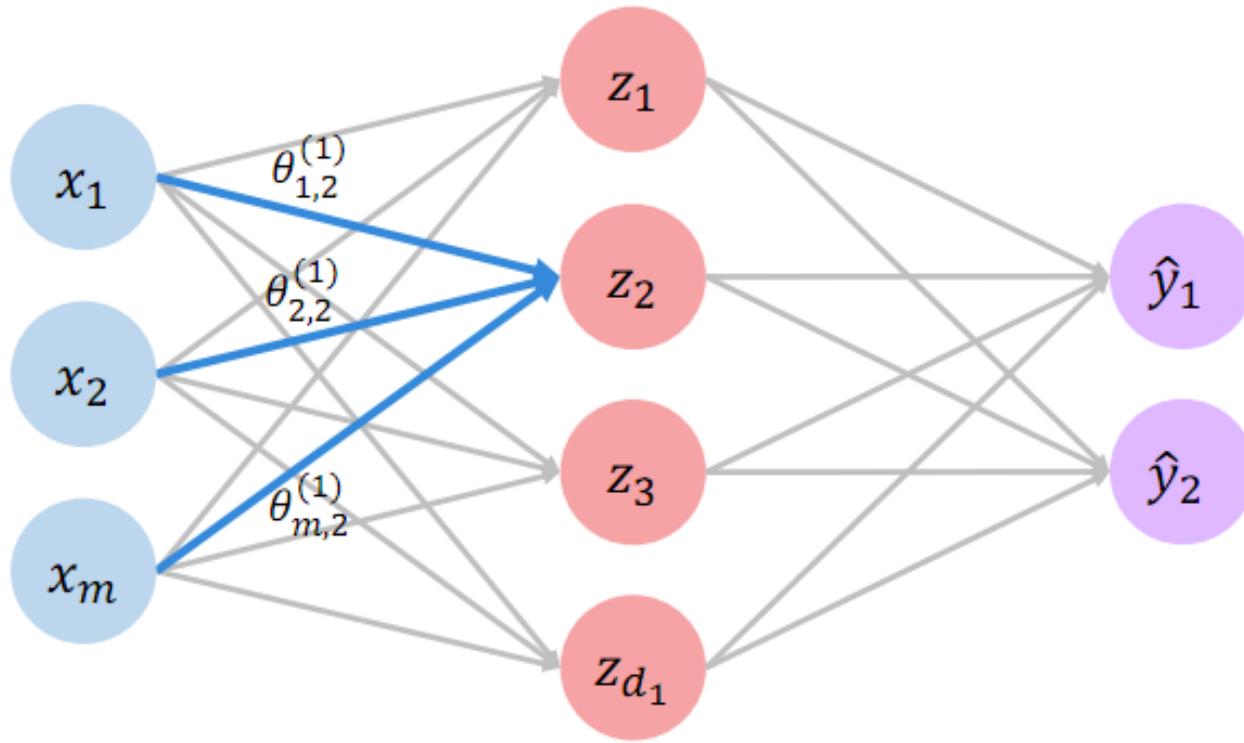
Inputs

Hidden

Final Output

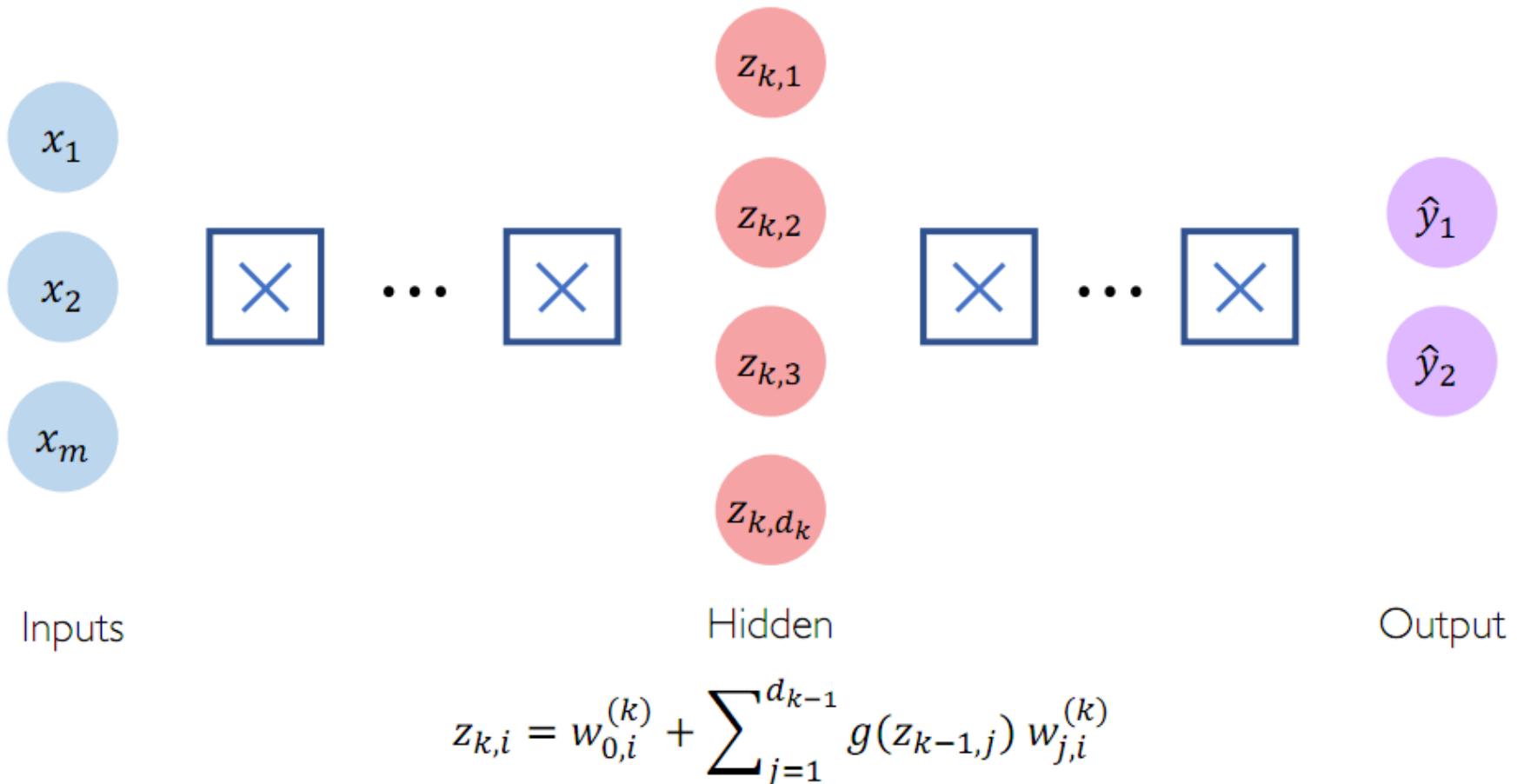
$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

Mạng nơ-ron một lớp ẩn



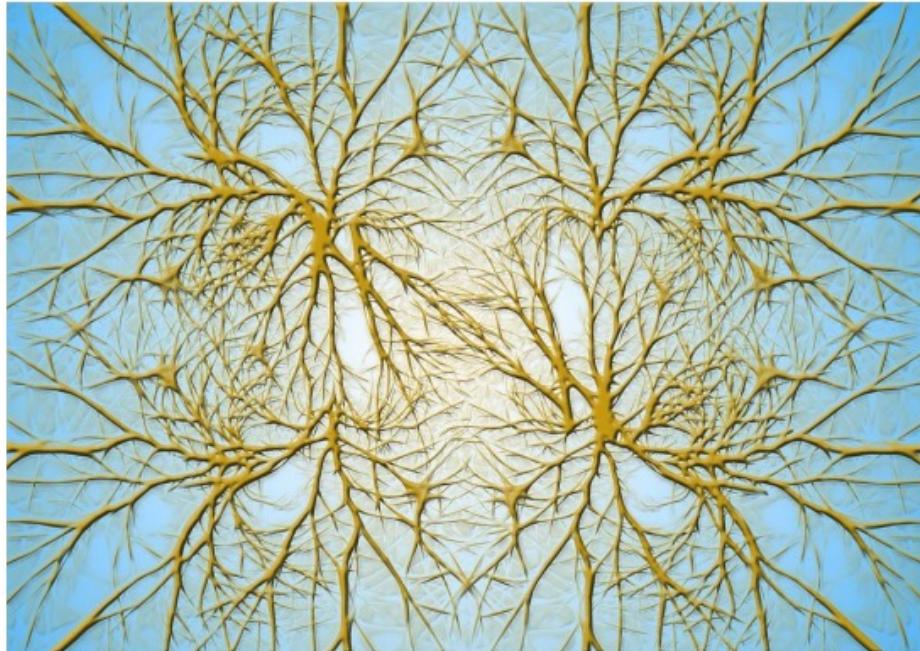
$$\begin{aligned} z_2 &= w_{0,2}^{(1)} + \sum_{j=1}^m x_j w_{j,2}^{(1)} \\ &= w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)} \end{aligned}$$

Mạng nơ-ron nhiều lớp

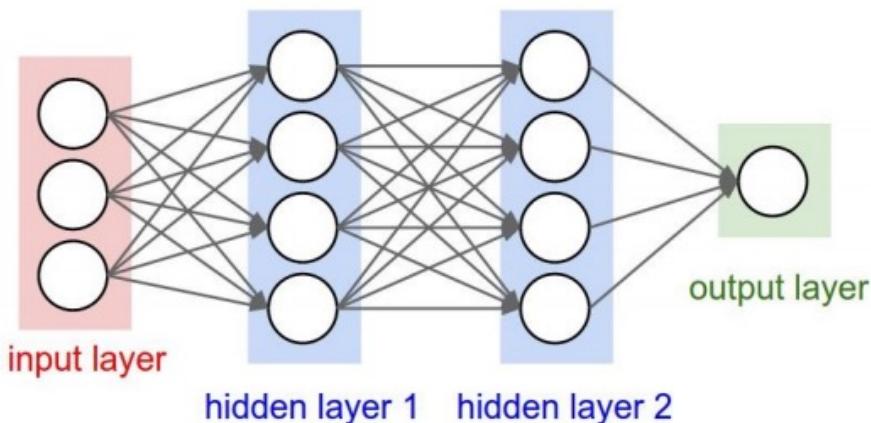


Mạng nơ-ron và bộ não

Nơ-ron sinh học:
Kết nối phức tạp



Mạng nơ-ron nhân tạo:
Các nơ-ron tổ chức
thành các lớp (layers)
để tăng hiệu quả tính
tổán nhờ song song hóa



Định lý xấp xỉ tổng quát

- Theorem (Universal Function Approximators). Một mạng nơ-ron từ hai lớp trở lên với số lượng nơ-ron đủ lớn có thể xấp xỉ bất kỳ hàm liên tục nào với độ chính xác tùy ý



$$n_1(x) = \text{Relu}(-5x - 7.7)$$

$$n_2(x) = \text{Relu}(-1.2x - 1.3)$$

$$n_3(x) = \text{Relu}(1.2x + 1)$$

$$n_4(x) = \text{Relu}(1.2x - .2)$$

$$n_5(x) = \text{Relu}(2x - 1.1)$$

$$n_6(x) = \text{Relu}(5x - 5)$$

$$\begin{aligned} Z(x) = & -n_1(x) - n_2(x) - n_3(x) \\ & + n_4(x) + n_5(x) + n_6(x) \end{aligned}$$

Universal Function Approximation Theorem*

Hornik theorem 1: Whenever the activation function is *bounded and nonconstant*, then, for any finite measure μ , standard multilayer feedforward networks can approximate any function in $L^p(\mu)$ (the space of all functions on R^k such that $\int_{R^k} |f(x)|^p d\mu(x) < \infty$) arbitrarily well, provided that sufficiently many hidden units are available.

Hornik theorem 2: Whenever the activation function is *continuous, bounded and non-constant*, then, for arbitrary compact subsets $X \subseteq R^k$, standard multilayer feedforward networks can approximate any continuous function on X arbitrarily well with respect to uniform distance, provided that sufficiently many hidden units are available.

- **In words:** Given any continuous function $f(x)$, if a 2-layer neural network has enough hidden units, then there is a choice of weights that allow it to closely approximate $f(x)$.

Cybenko (1989) "Approximations by superpositions of sigmoidal functions"

Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks"

Leshno and Schocken (1991) "Multilayer Feedforward Networks with Non-Polynomial Activation Functions Can Approximate Any Function"

Tại sao cần mạng nhiều lớp?

- Mạng nơ-ron nhiều lớp (thậm chí chỉ cần duy nhất một lớp ẩn!) là hàm xấp xỉ tổng quát
 - Mạng nơ-ron có thể biểu diễn hàm bất kỳ nếu nó đủ rộng (số nơ-ron trong một lớp đủ nhiều), đủ sâu (số lớp đủ lớn). Nếu muốn giảm độ sâu của mạng trong nhiều trường hợp sẽ phải bù lại bằng cách tăng chiều rộng lên lũy thừa lần!
 - Mạng nơ-ron một lớp ẩn có thể cần tới số lượng nơ-ron cao gấp lũy thừa lần so với một mạng nhiều tầng
 - Mạng nhiều lớp cần số lượng nơ-ron ít hơn rất nhiều so với các mạng nông (shallow networks) để cùng biểu diễn một hàm số giống nhau
- Mạng nhiều lớp giá trị hơn

Cực tiểu hóa hàm mục tiêu

- Tìm trọng số của mạng để hàm mục tiêu đạt giá trị cực tiểu

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$



Remember:

$$\mathbf{W} = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \dots\}$$

Cực tiểu hóa hàm mục tiêu



<https://www.youtube.com/watch?v=0jaN49pIXYw>

Cực tiểu hóa hàm mục tiêu

- Thuật toán Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$

```
 weights = tf.random_normal(shape, stddev=sigma)
```

2. Loop until convergence:

3. Compute gradient, $\frac{\partial J(W)}{\partial W}$

```
 grads = tf.gradients(ys=loss, xs=weights)
```

4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$

```
 weights_new = weights.assign(weights - lr * grads)
```

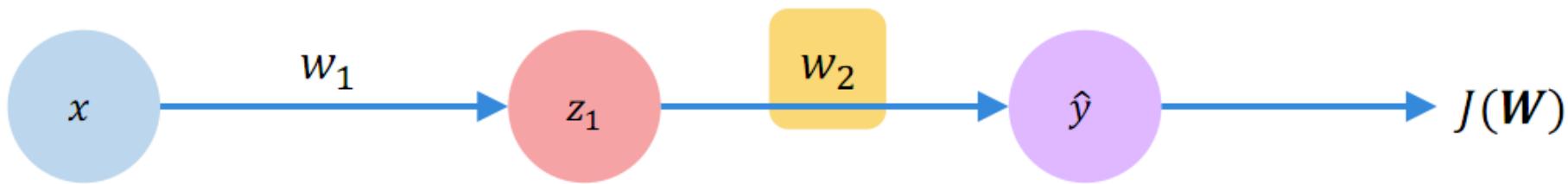
5. Return weights

Demo SGD: <https://www.youtube.com/watch?v=Q3pTEtSEvDI>

Giải thuật lan truyền ngược

Giải thuật lan truyền ngược

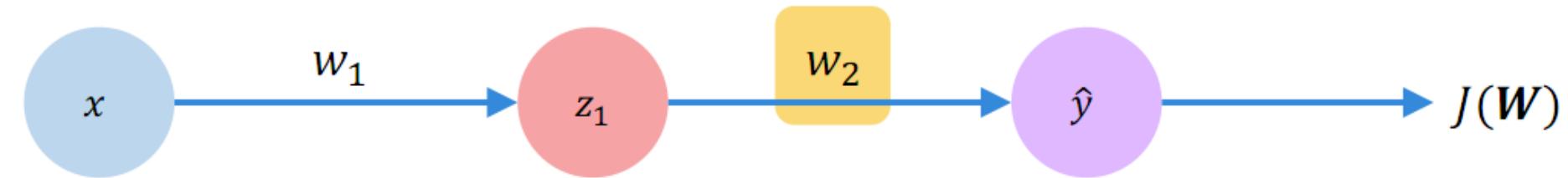
- Đánh giá sự thay đổi nhỏ ở một trọng số nào đó ảnh hưởng như thế nào tới hàm mục tiêu của mạng?



How does a small change in one weight (ex. w_2) affect the final loss $J(\mathbf{W})$?

Giải thuật lan truyền ngược

- Đánh giá sự thay đổi nhỏ ở một trọng số nào đó ảnh hưởng như thế nào tới hàm mục tiêu của mạng?

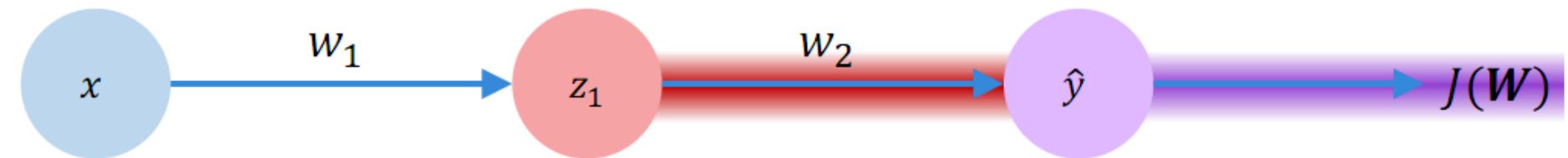


$$\frac{\partial J(\mathbf{W})}{\partial w_2} =$$

Let's use the chain rule!

Giải thuật lan truyền ngược

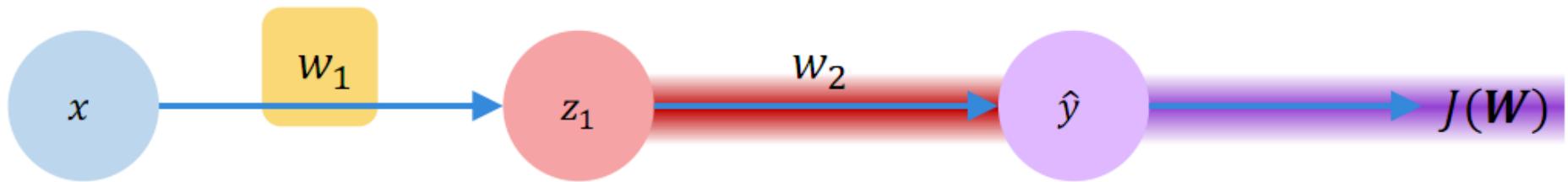
- Đánh giá sự thay đổi nhỏ ở một trọng số nào đó ảnh hưởng như thế nào tới hàm mục tiêu của mạng?



$$\frac{\partial J(\mathbf{W})}{\partial w_2} = \underbrace{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}}_{\text{purple bar}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}_{\text{red bar}}$$

Giải thuật lan truyền ngược

- Đánh giá sự thay đổi nhỏ ở một trọng số nào đó ảnh hưởng như thế nào tới hàm mục tiêu của mạng?



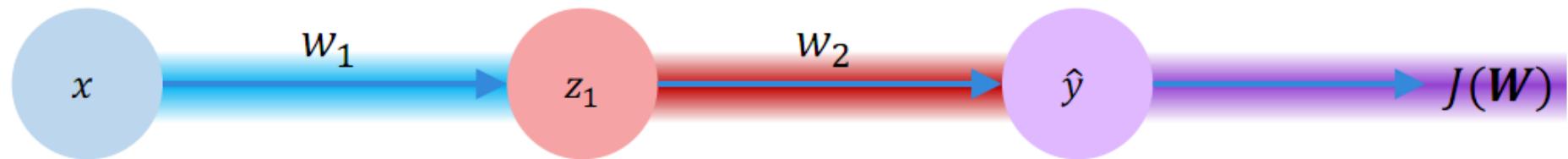
$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

Giải thuật lan truyền ngược

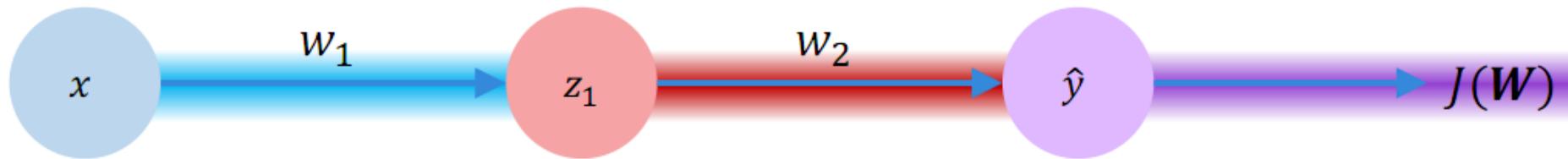
- Đánh giá sự thay đổi nhỏ ở một trọng số nào đó ảnh hưởng như thế nào tới hàm mục tiêu của mạng?



$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \underbrace{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}}_{\text{purple bar}} * \underbrace{\frac{\partial \hat{y}}{\partial z_1}}_{\text{red bar}} * \underbrace{\frac{\partial z_1}{\partial w_1}}_{\text{blue bar}}$$

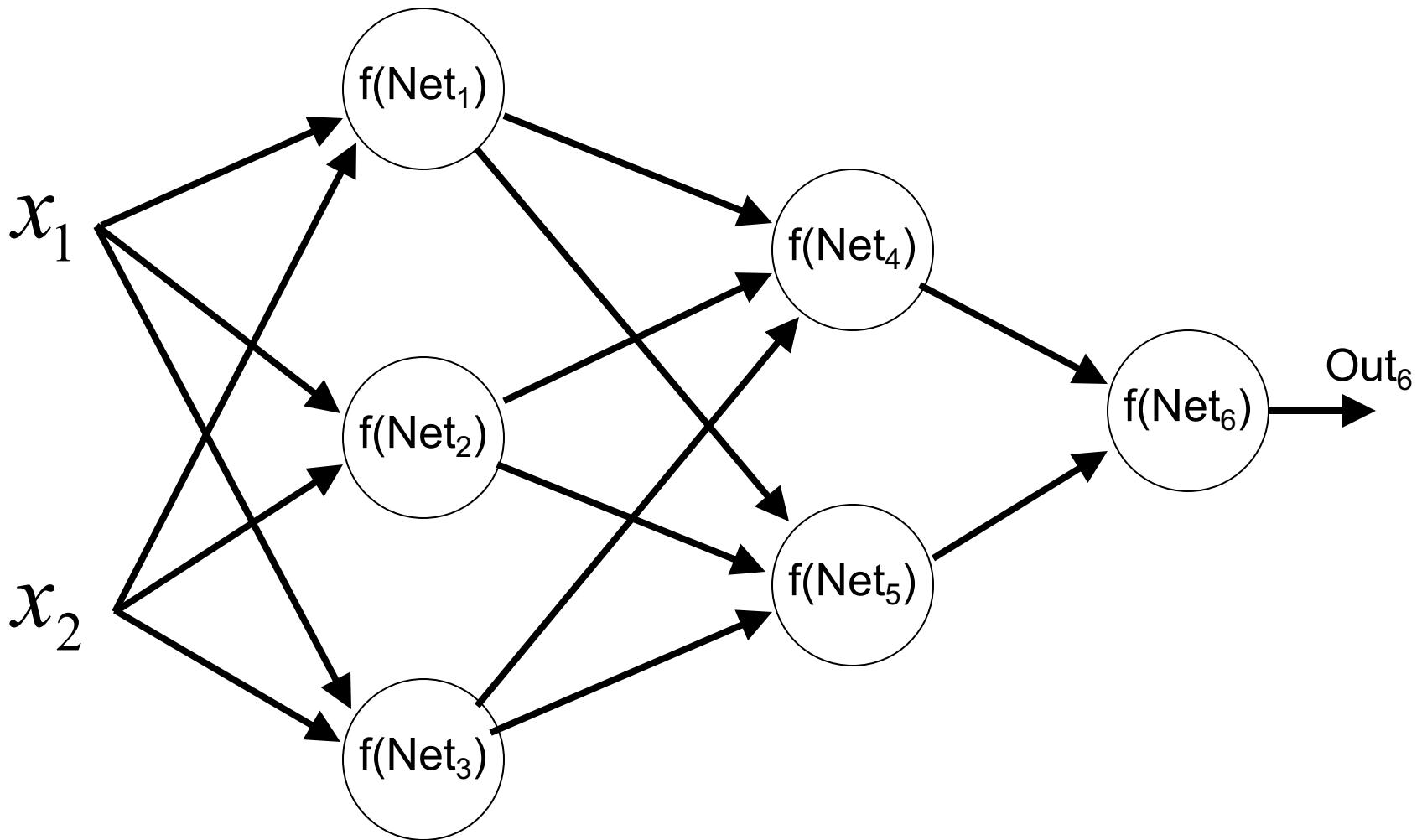
Giải thuật lan truyền ngược

- Lặp lại cách ước lượng này cho tất cả các trọng số trọng mạng dựa trên gradients đã tính ở các lớp trước

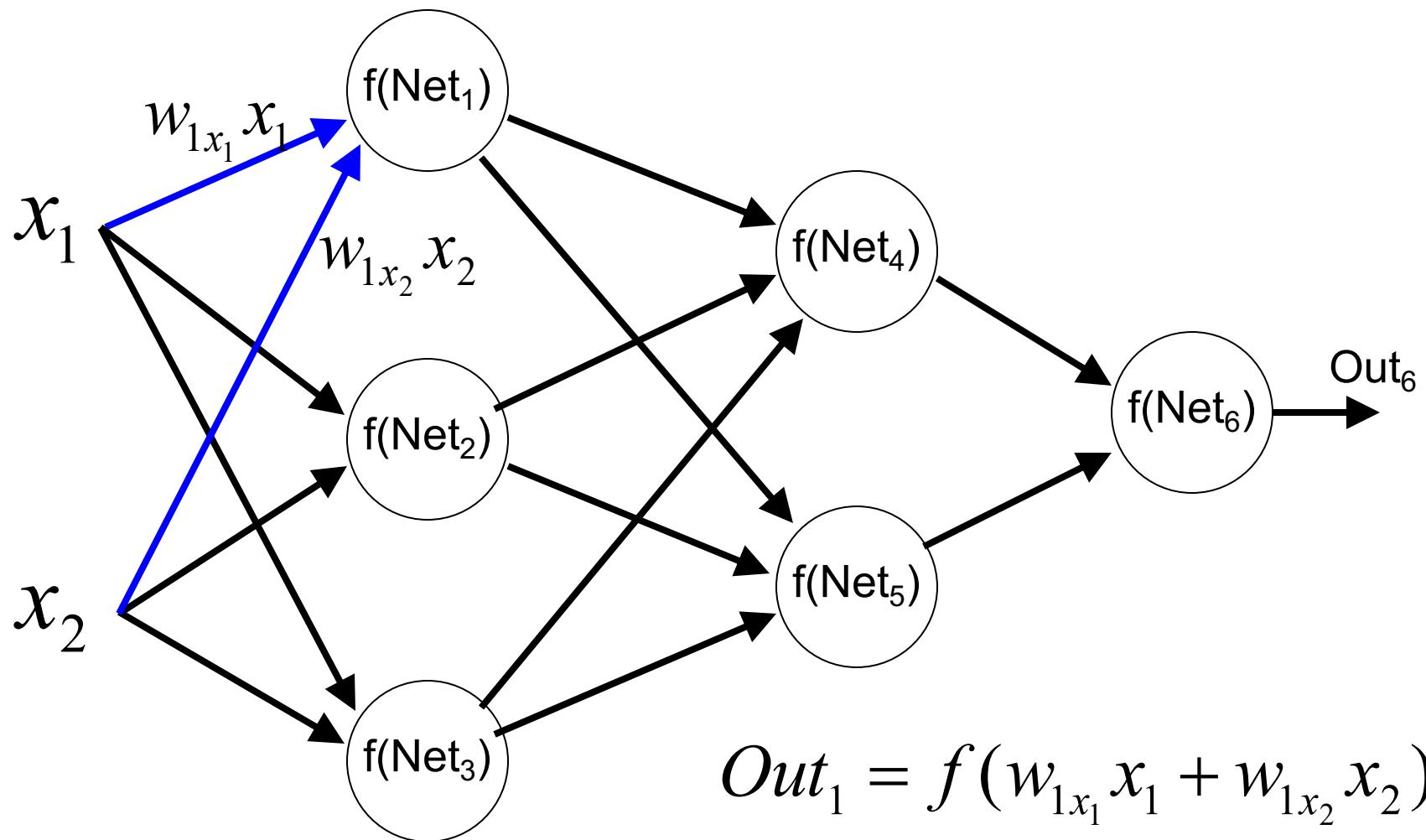


$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \underbrace{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}}_{\text{purple bar}} * \underbrace{\frac{\partial \hat{y}}{\partial z_1}}_{\text{red bar}} * \underbrace{\frac{\partial z_1}{\partial w_1}}_{\text{blue bar}}$$

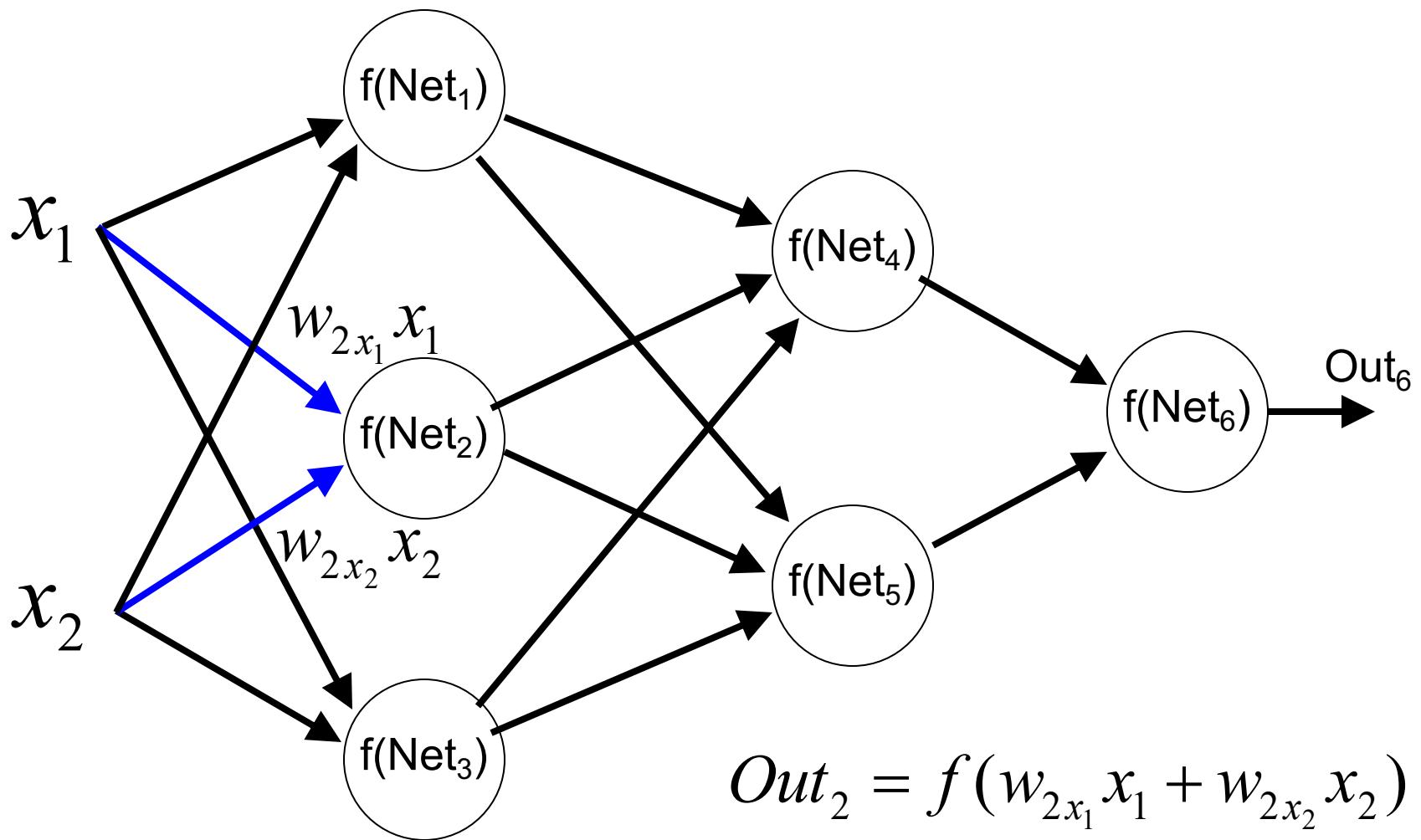
Giải thuật BP: Lan truyền tiến (1)



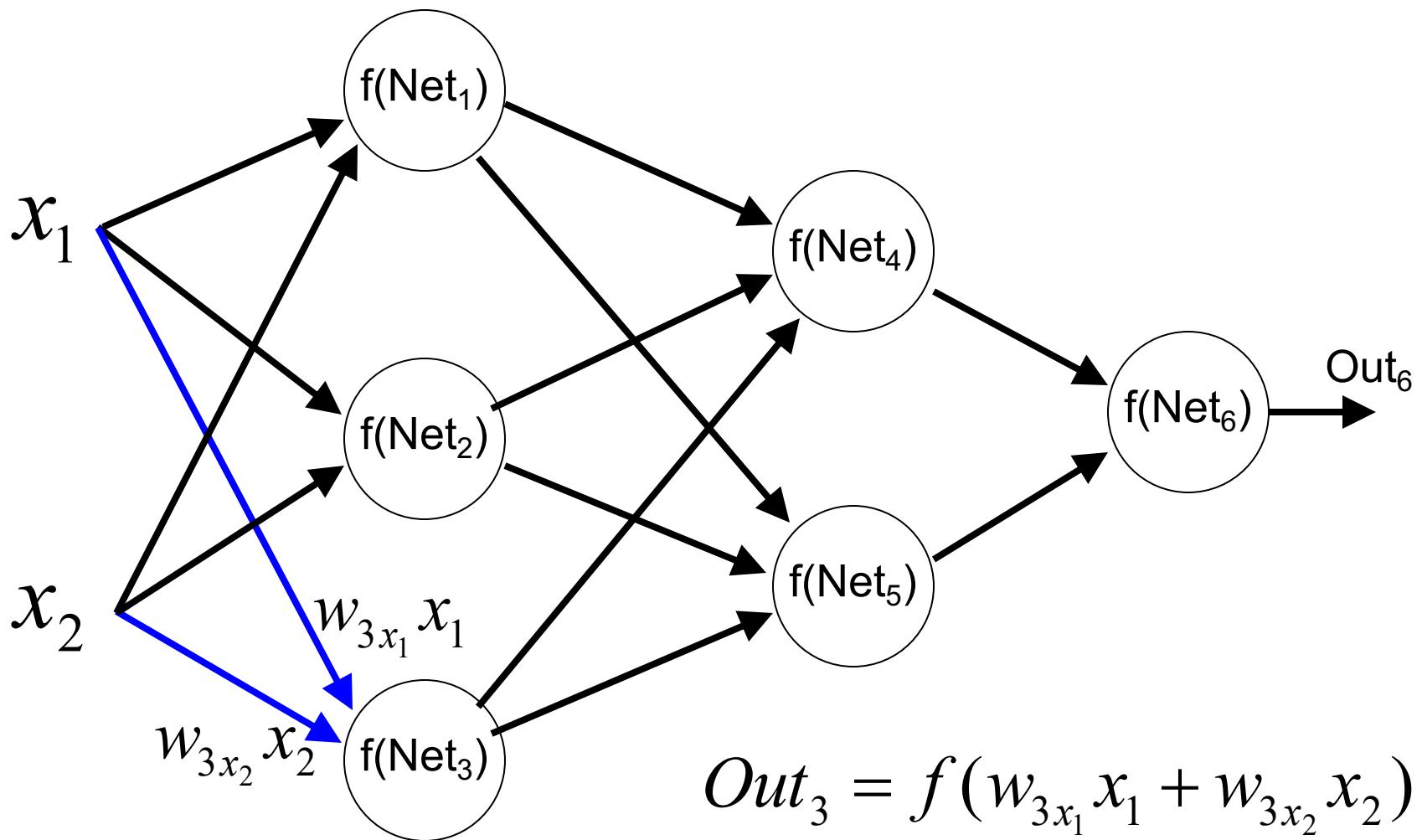
Giải thuật BP: Lan truyền tiến (2)



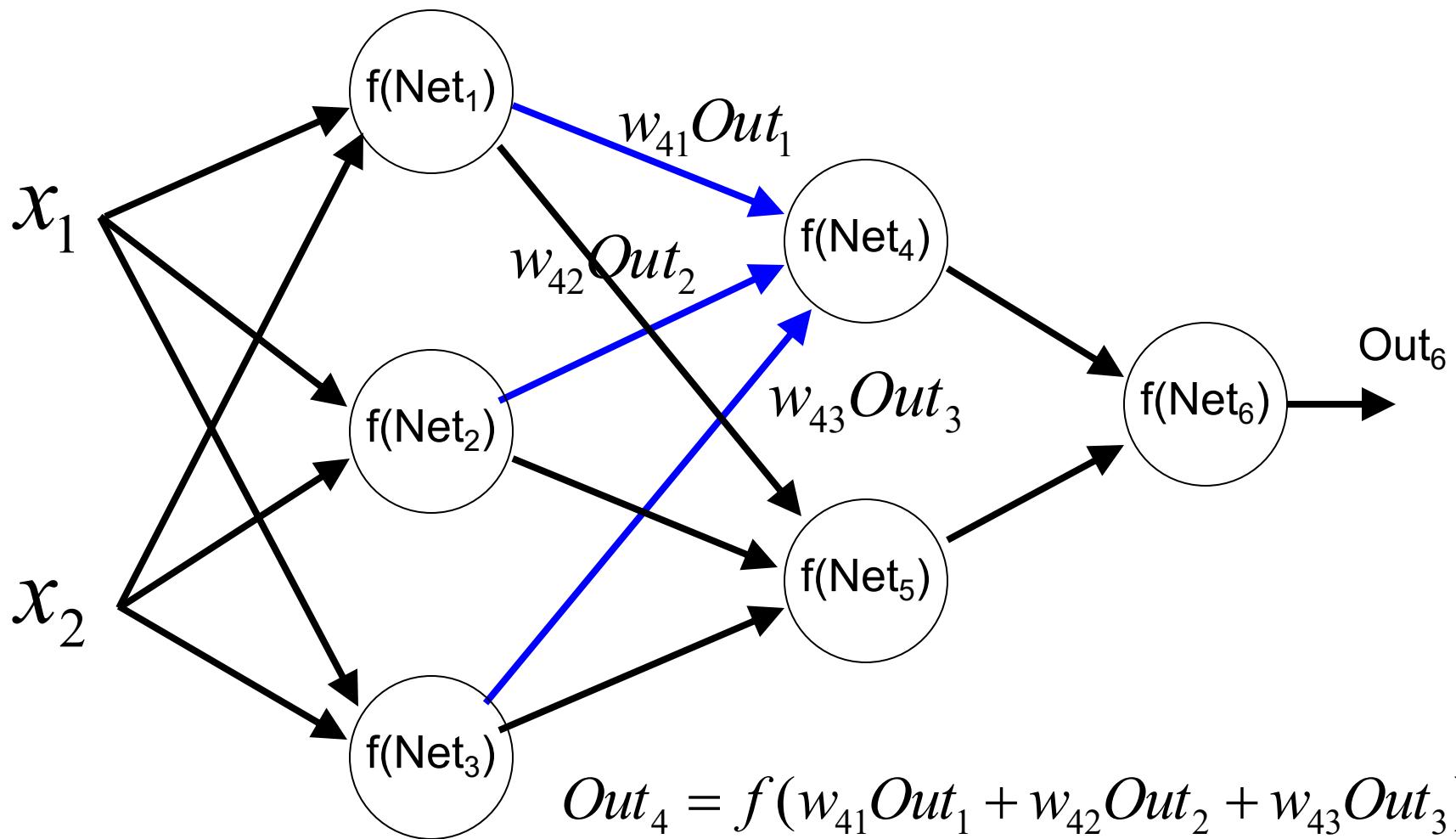
Giải thuật BP: Lan truyền tiến (3)



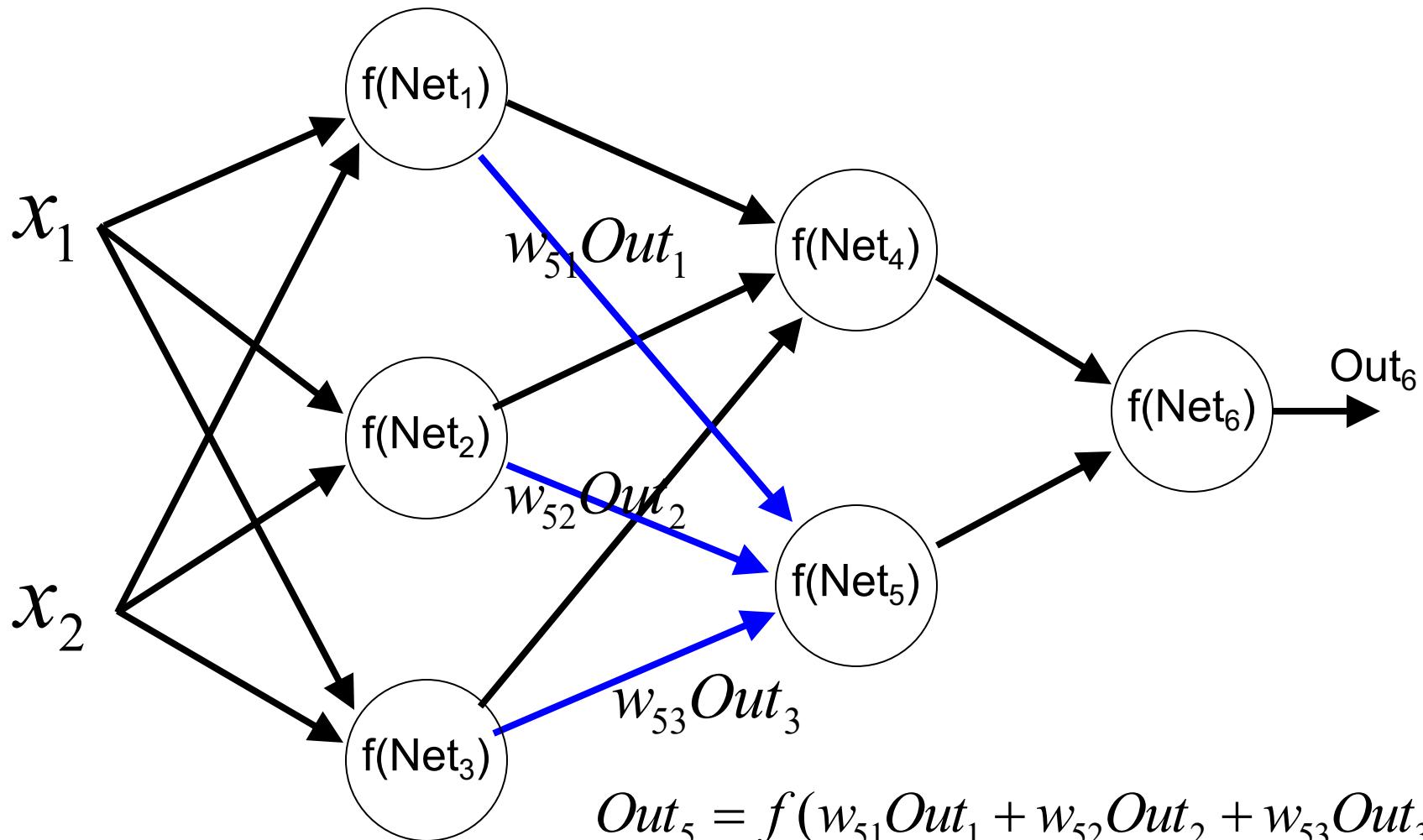
Giải thuật BP: Lan truyền tiến (4)



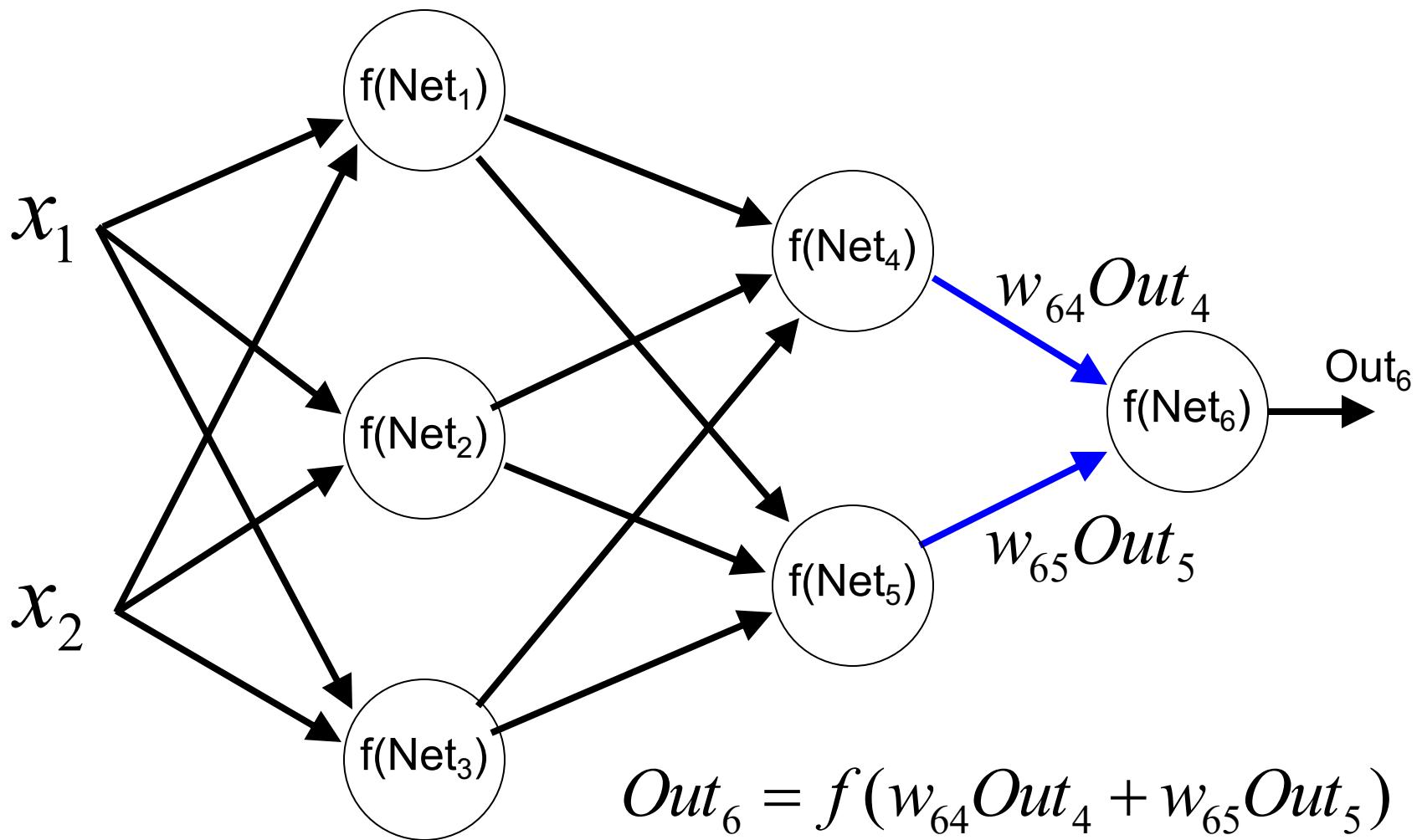
Giải thuật BP: Lan truyền tiến (5)



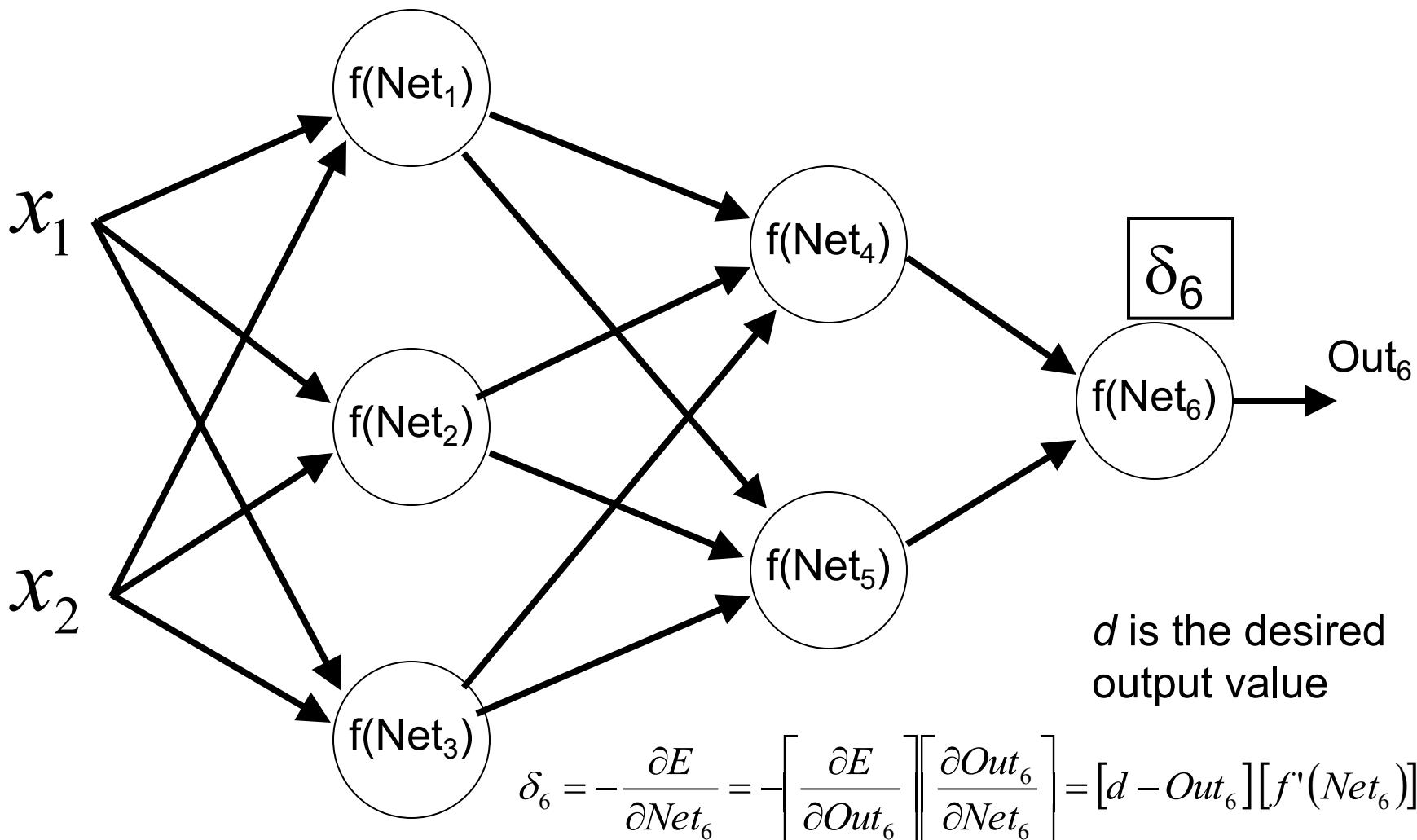
Giải thuật BP: Lan truyền tiến (6)



Giải thuật BP: Lan truyền tiến (7)

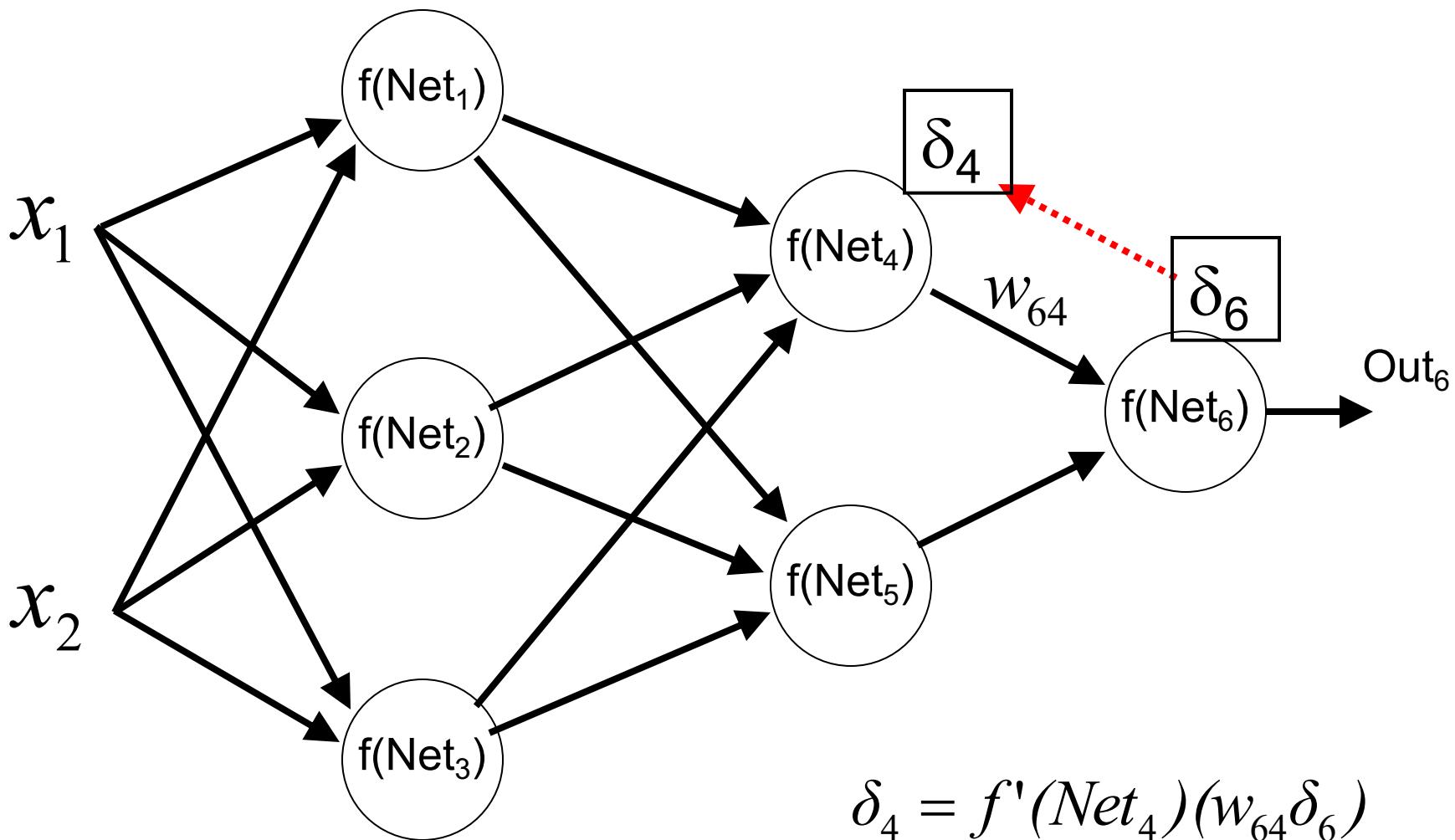


Giải thuật BP: Tính toán lỗi

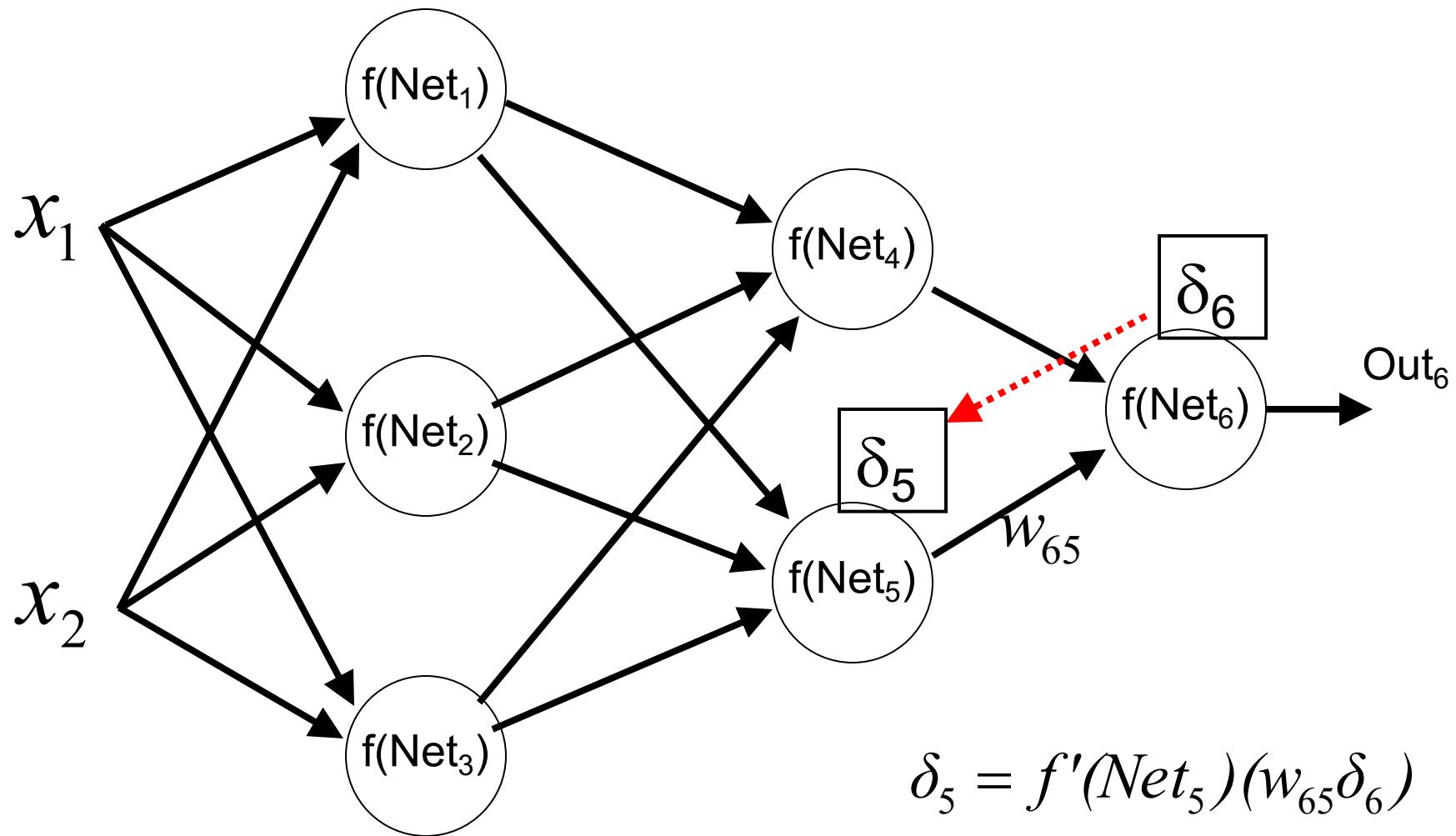


d is the desired output value

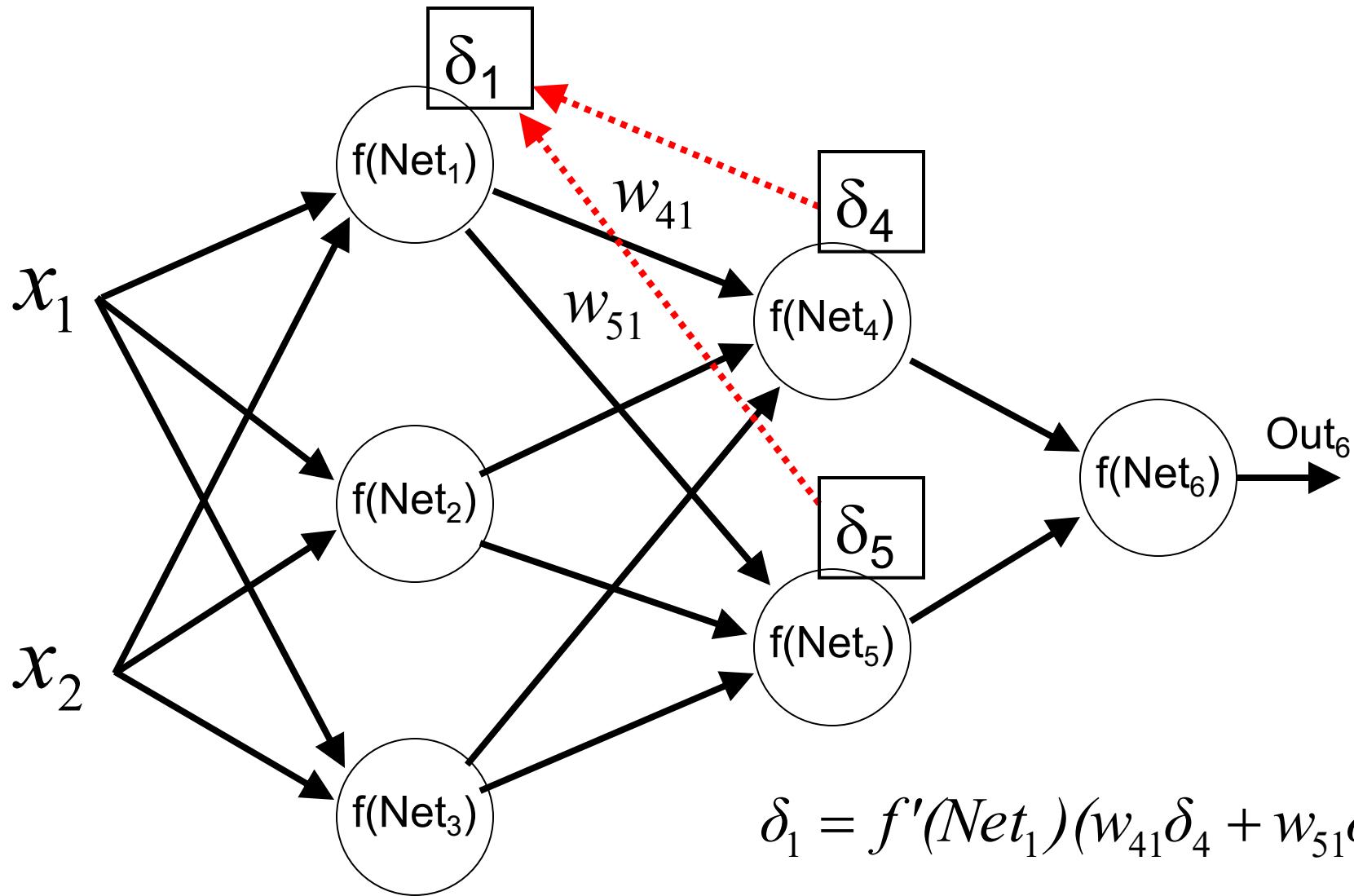
Giải thuật BP: Lan truyền ngược (1)



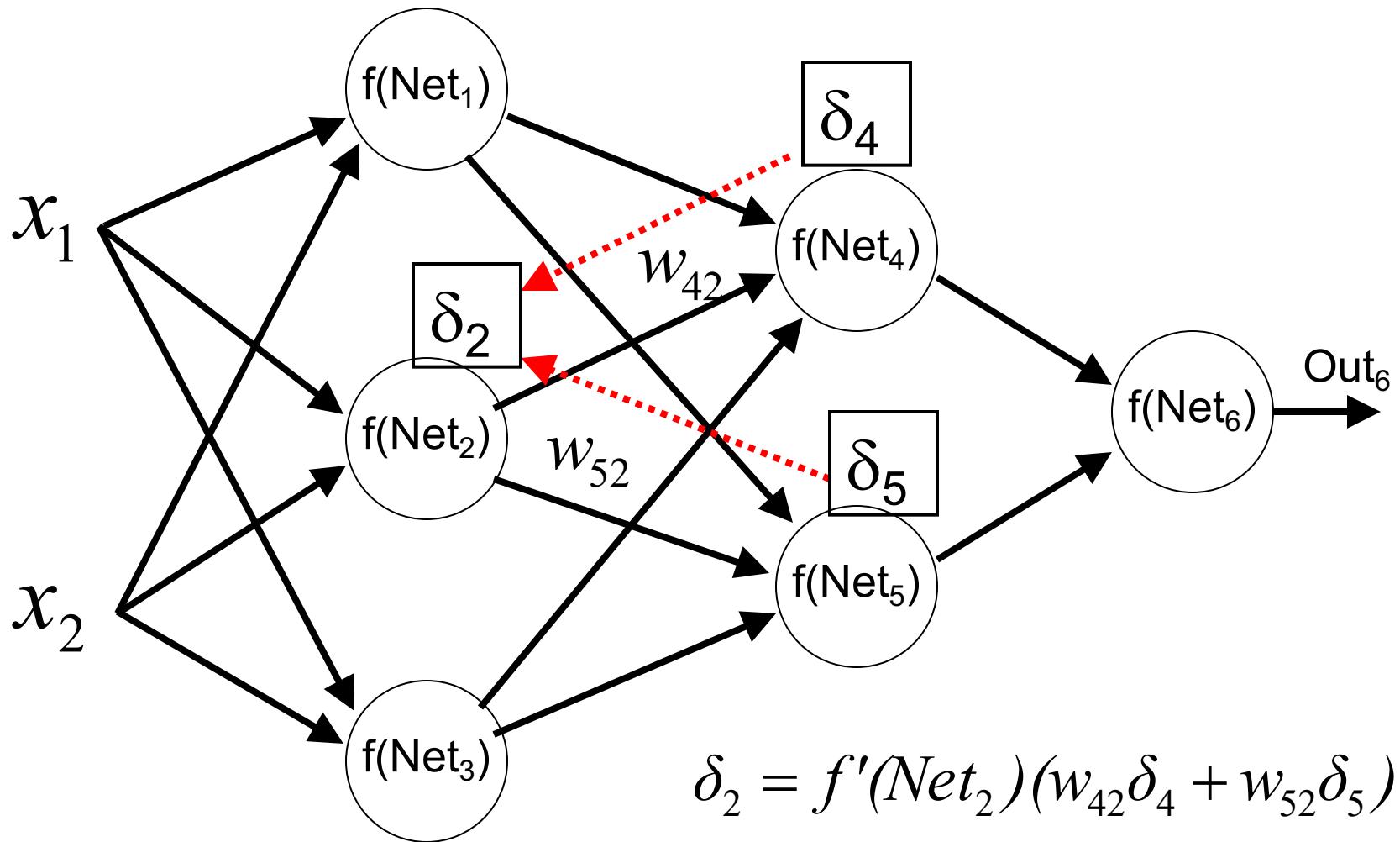
Giải thuật BP: Lan truyền ngược (2)



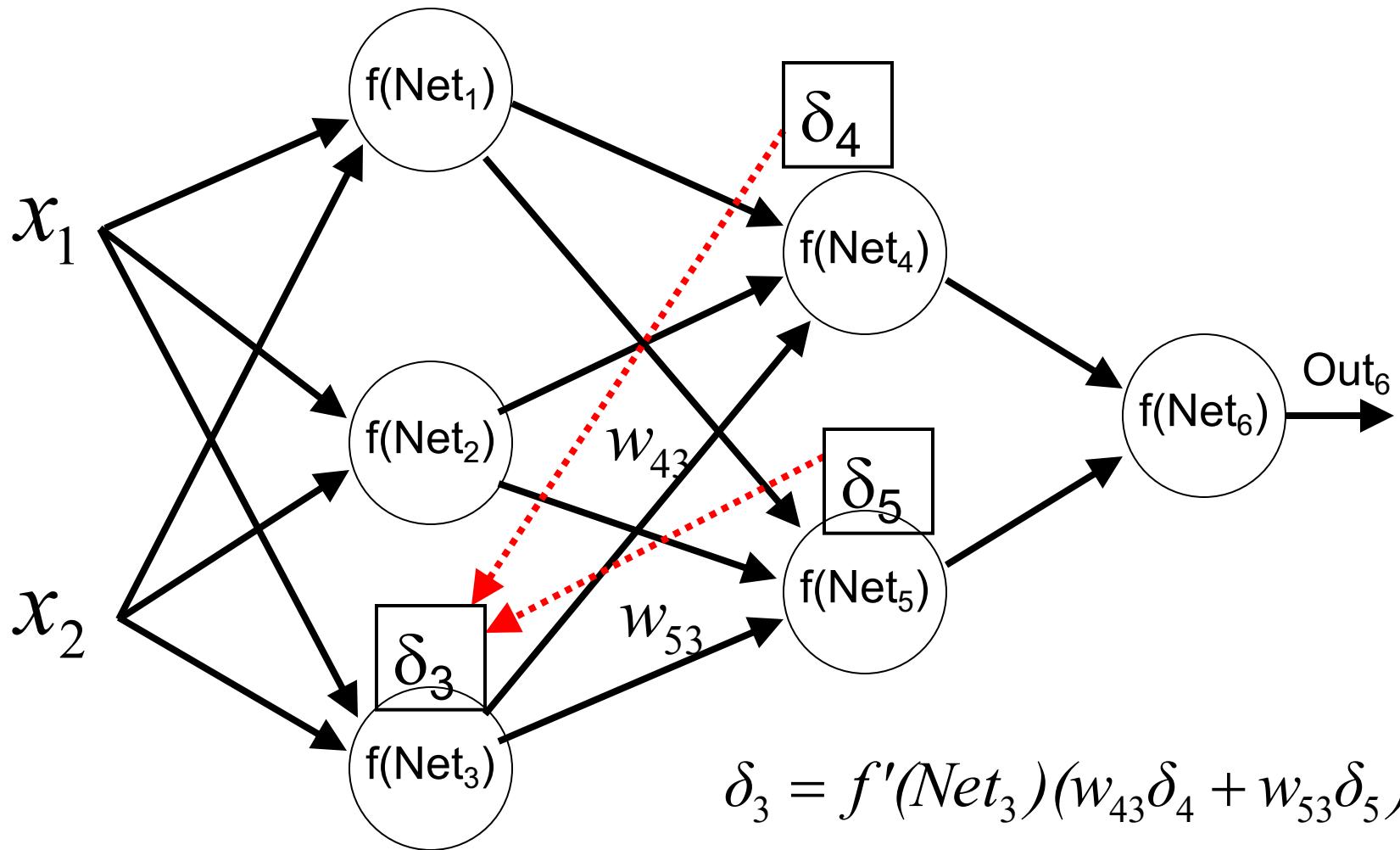
Giải thuật BP: Lan truyền ngược (3)



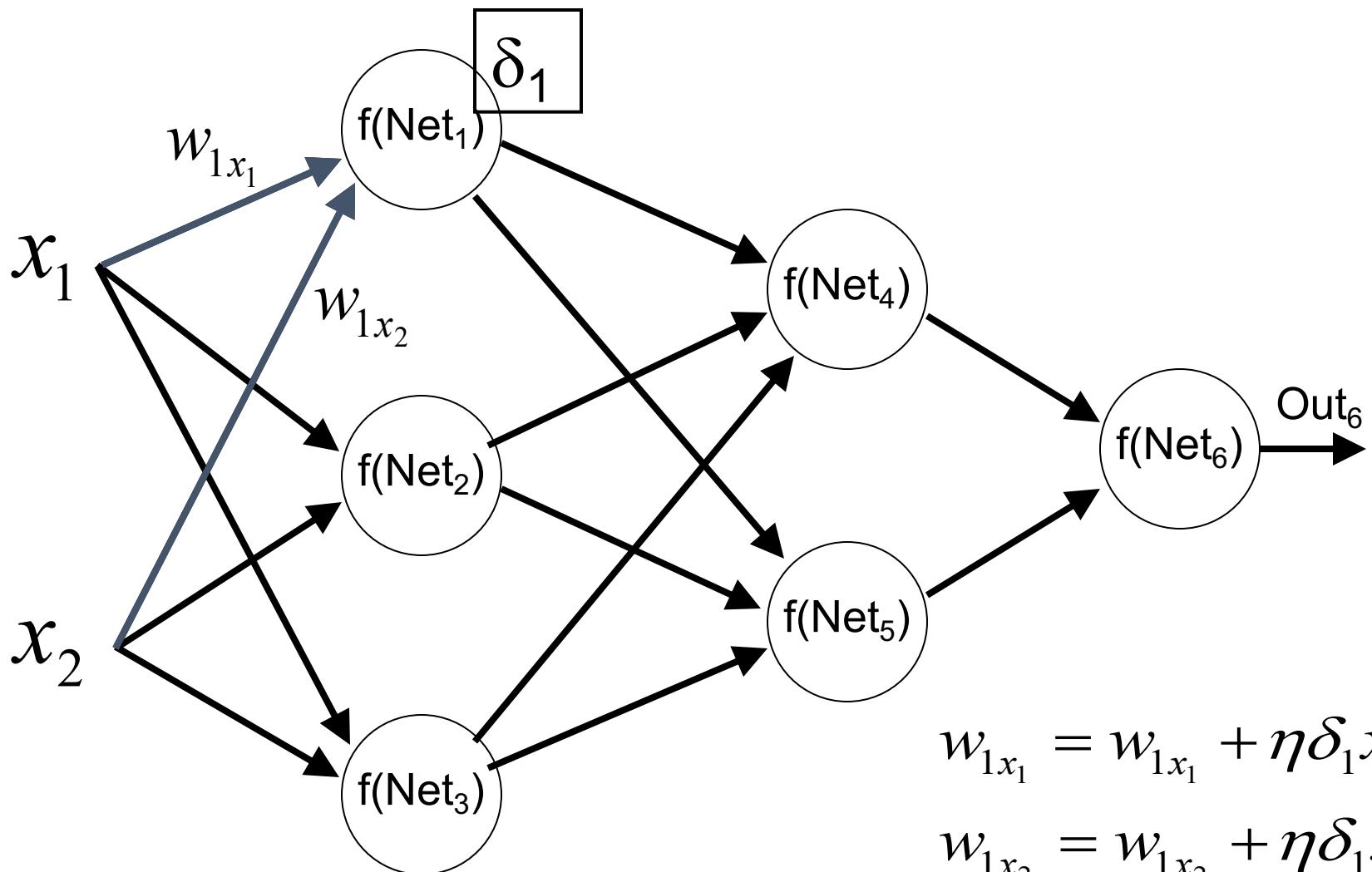
Giải thuật BP: Lan truyền ngược (4)



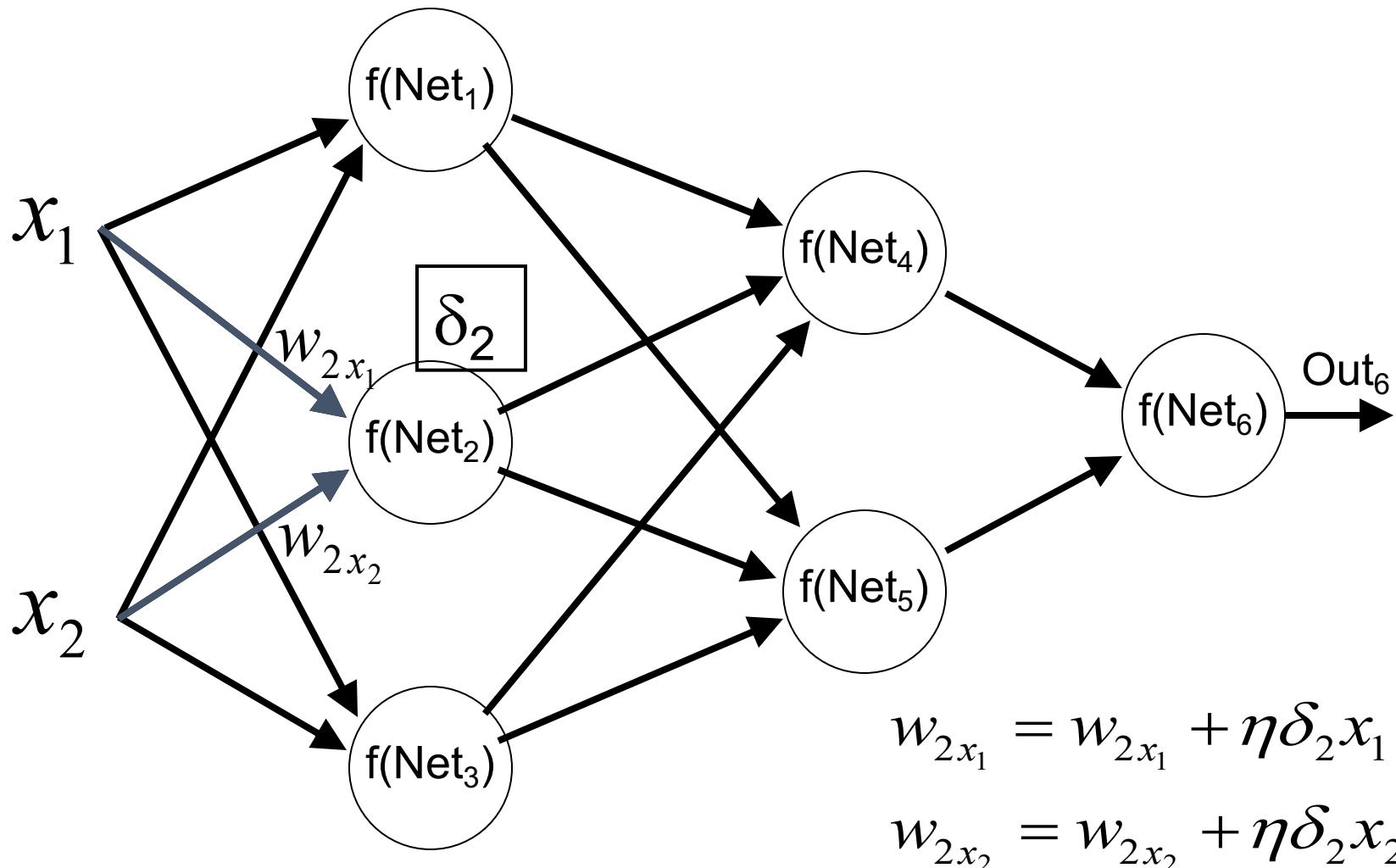
Giải thuật BP: Lan truyền ngược (5)



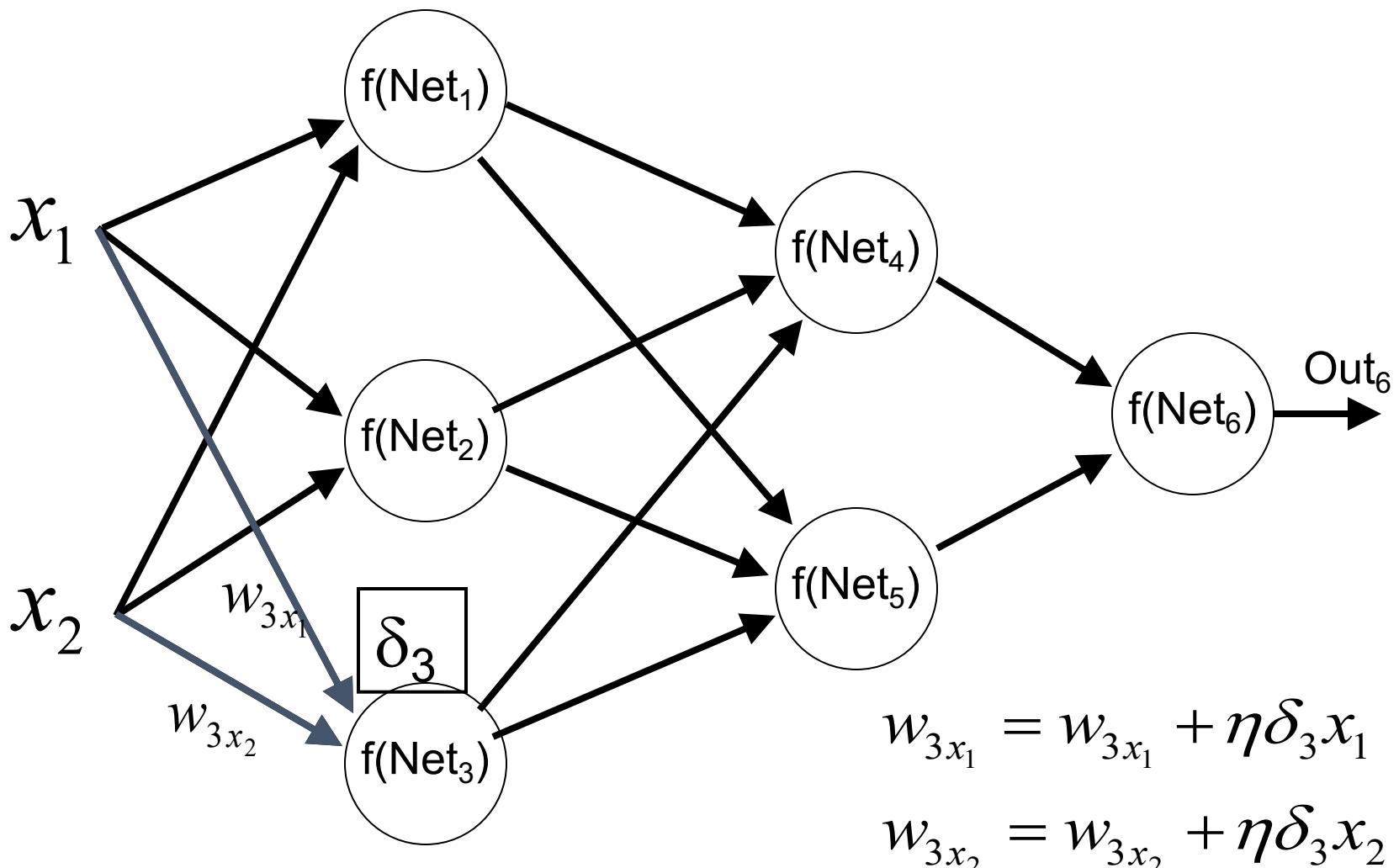
Giải thuật BP: Cập nhật trọng số (1)



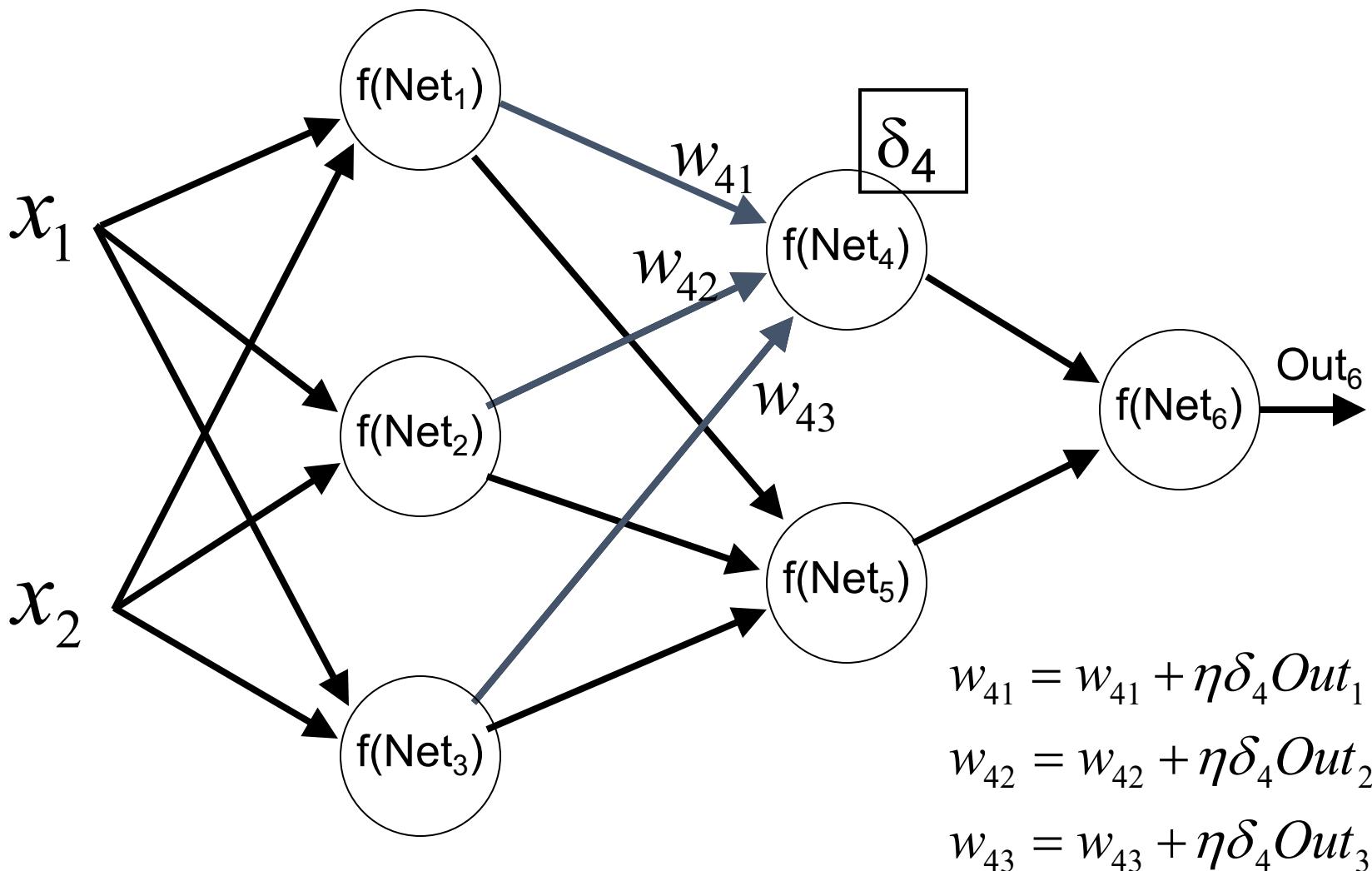
Giải thuật BP: Cập nhật trọng số (2)



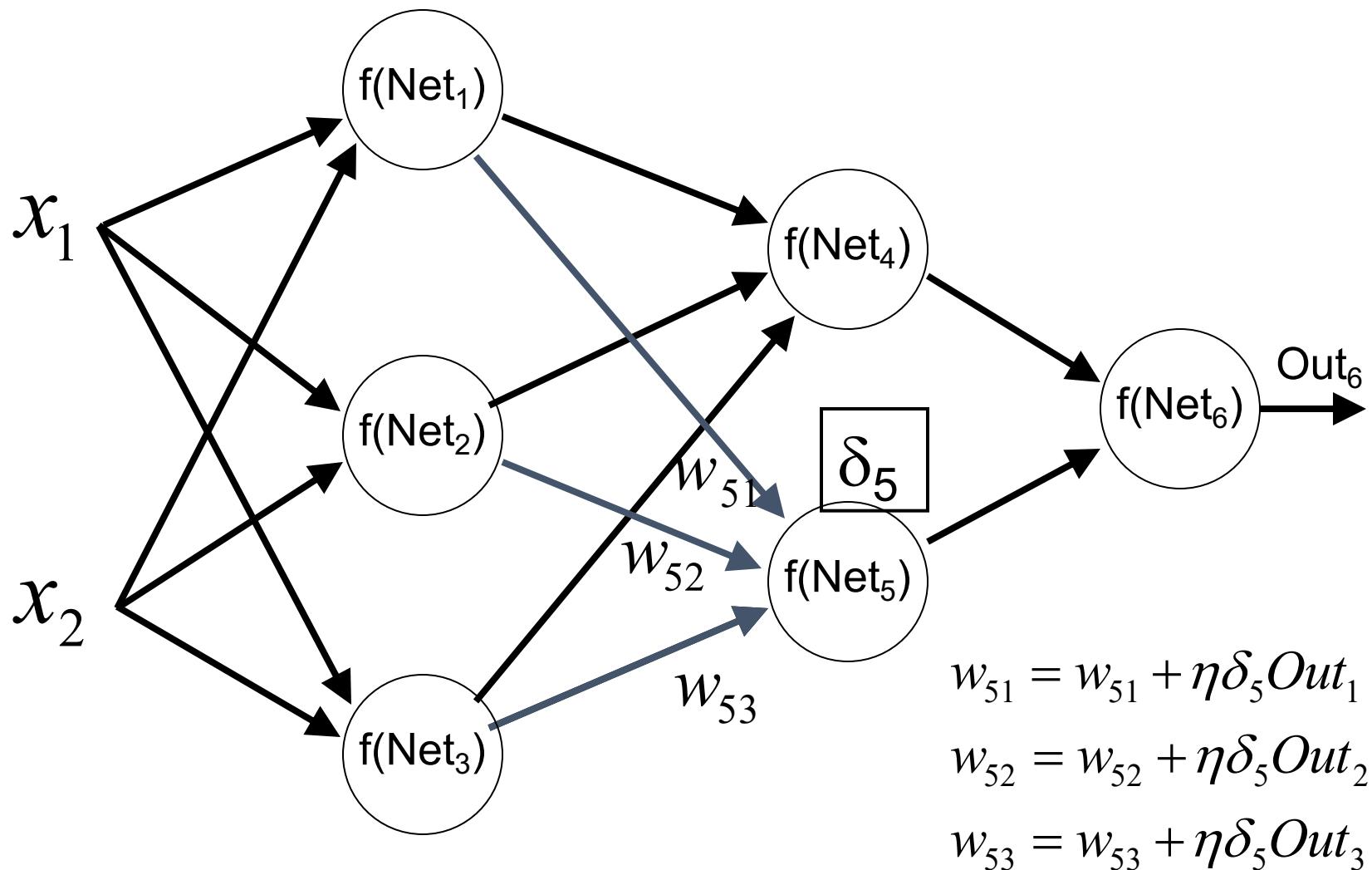
Giải thuật BP: Cập nhật trọng số (3)



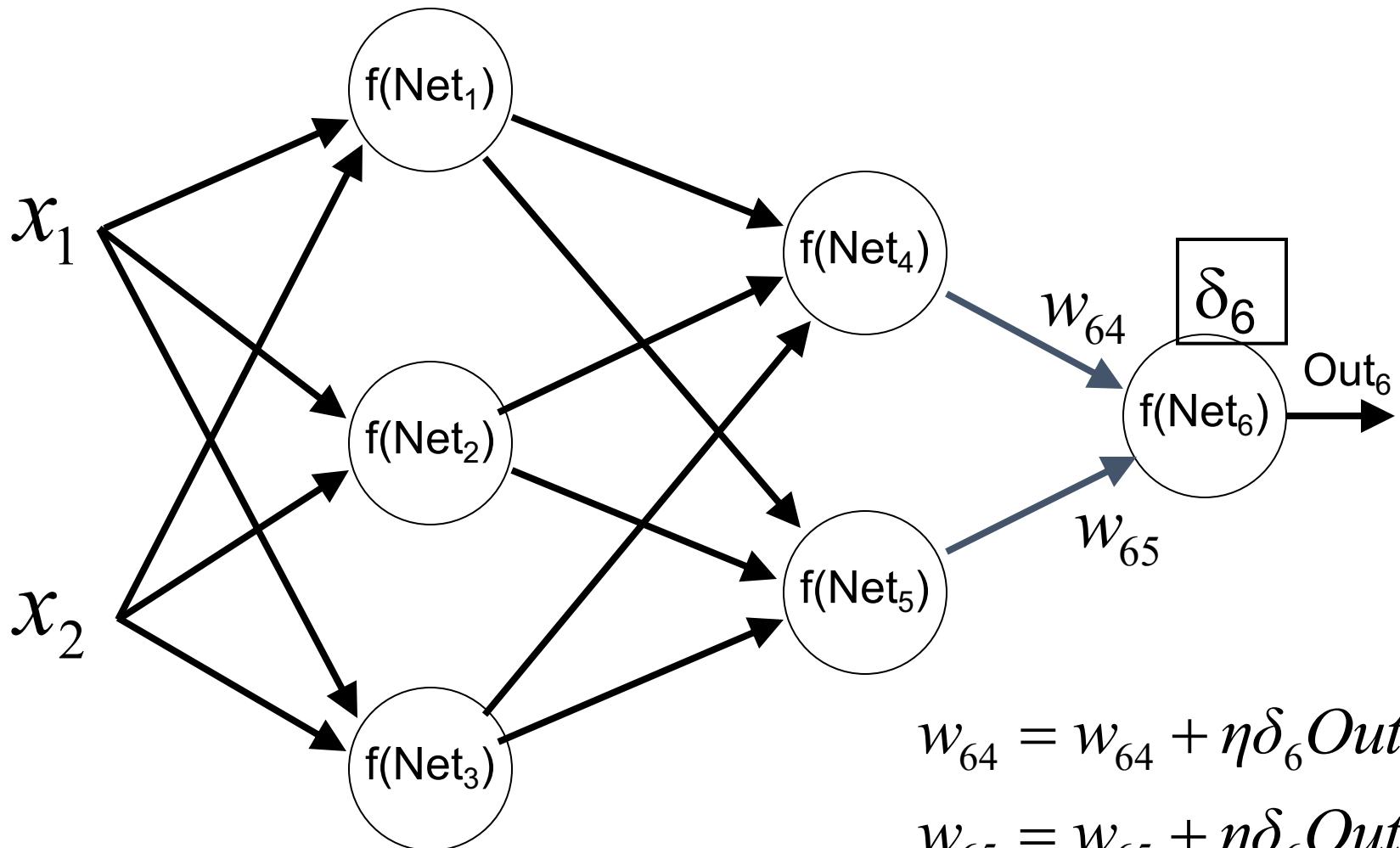
Giải thuật BP: Cập nhật trọng số (4)



Giải thuật BP: Cập nhật trọng số (5)



Giải thuật BP: Cập nhật trọng số (6)

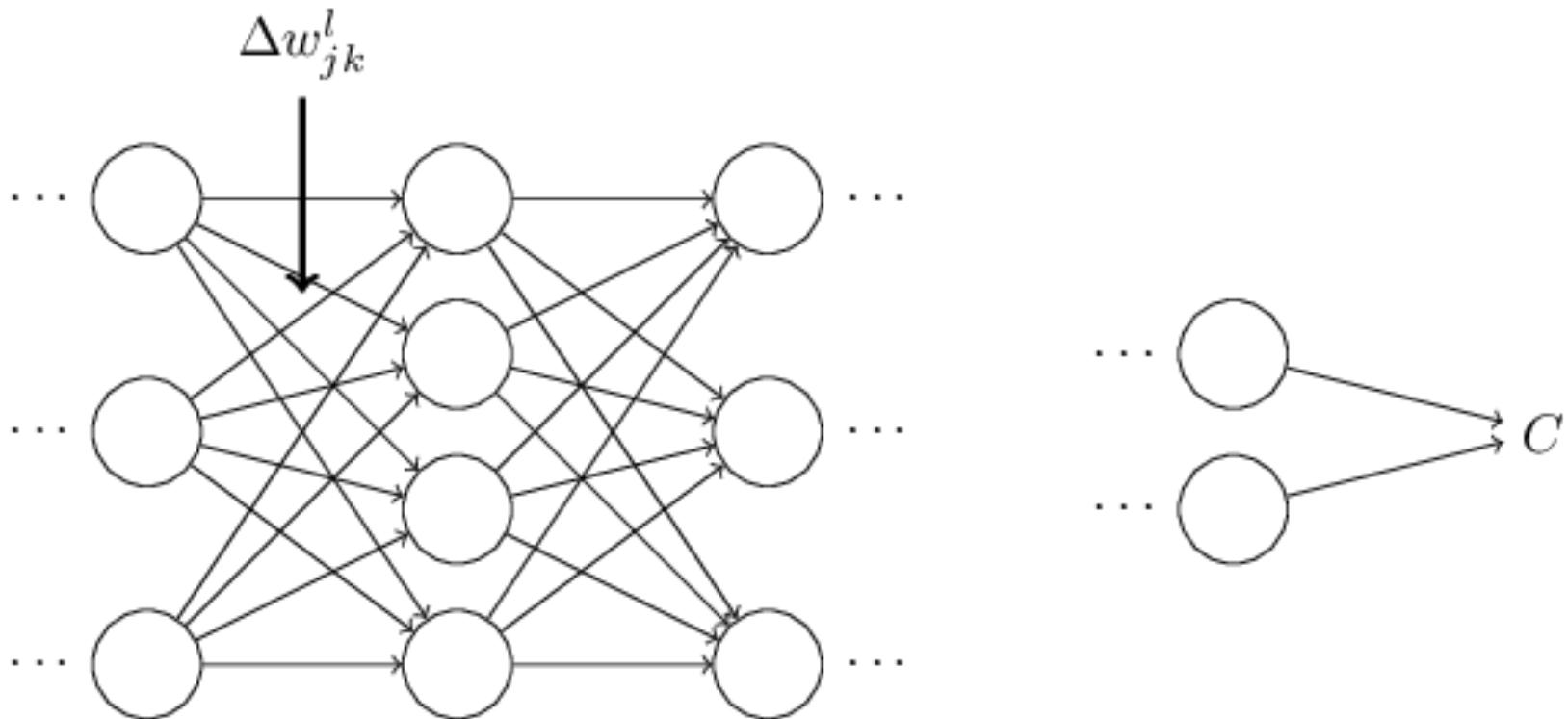


Giải thuật lan truyền ngược

1. **Input** x : Set the corresponding activation a^1 for the input layer.
2. **Feedforward**: For each $l = 2, 3, \dots, L$ compute
$$z^l = w^l a^{l-1} + b^l \text{ and } a^l = \sigma(z^l).$$
3. **Output error** δ^L : Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate the error**: For each $l = L - 1, L - 2, \dots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Output**: The gradient of the cost function is given by
$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \text{ and } \frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

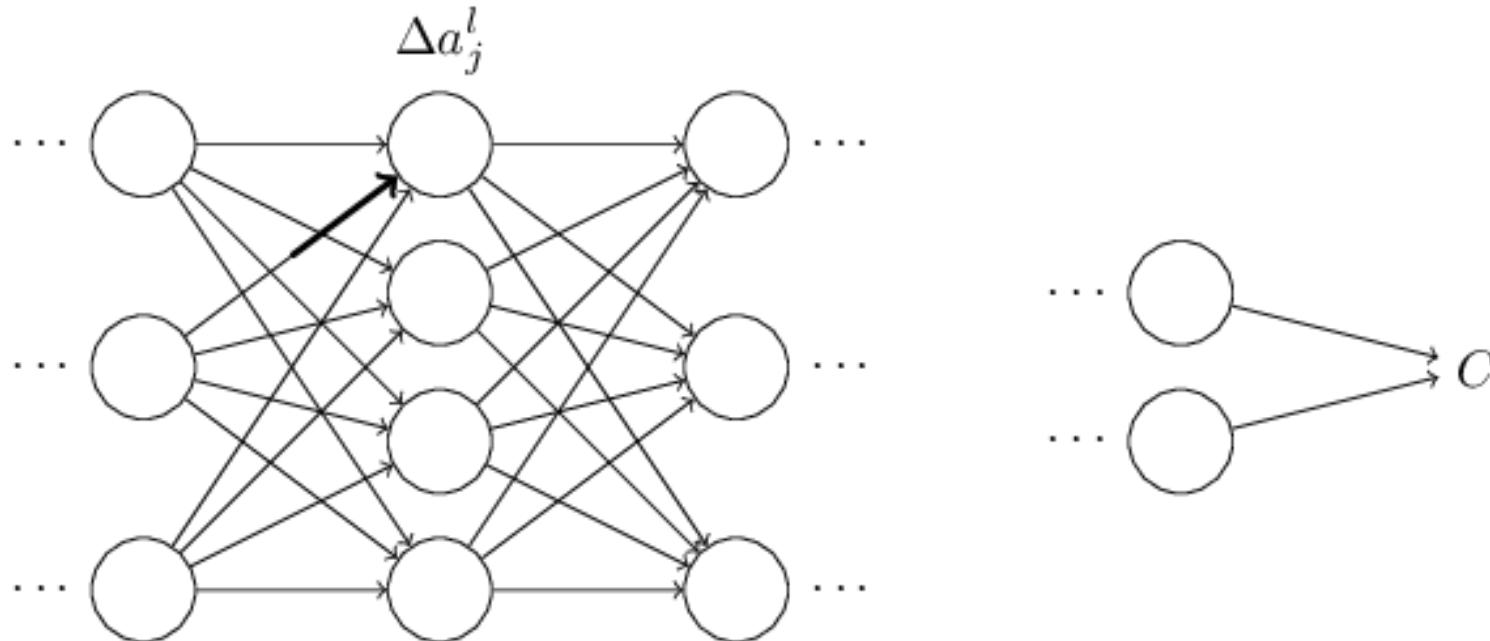
Giải thuật lan truyền ngược

- Giả sử có sự thay đổi nhỏ Δw_{jk}^l giá trị của trọng số w_{jk}^l ở lớp thứ l



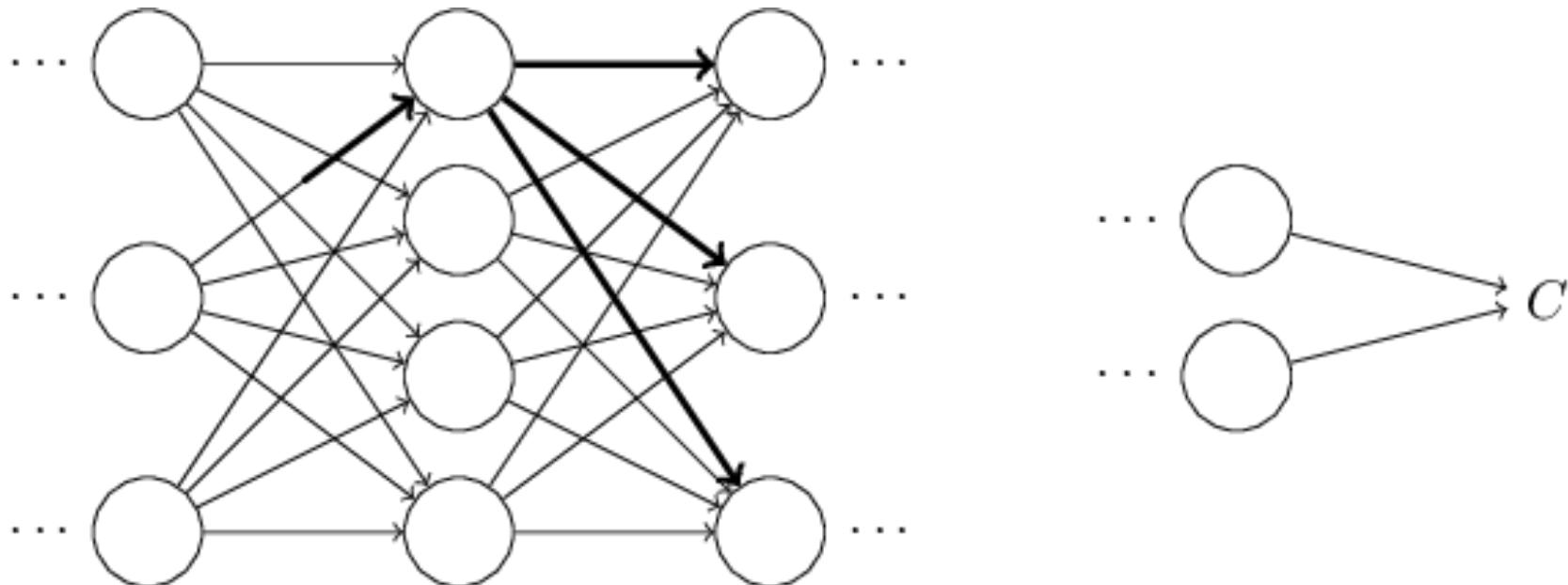
Giải thuật lan truyền ngược

- Sự thay đổi sẽ ảnh hưởng tới giá trị đầu ra của hàm kích hoạt nơ-ron tương ứng



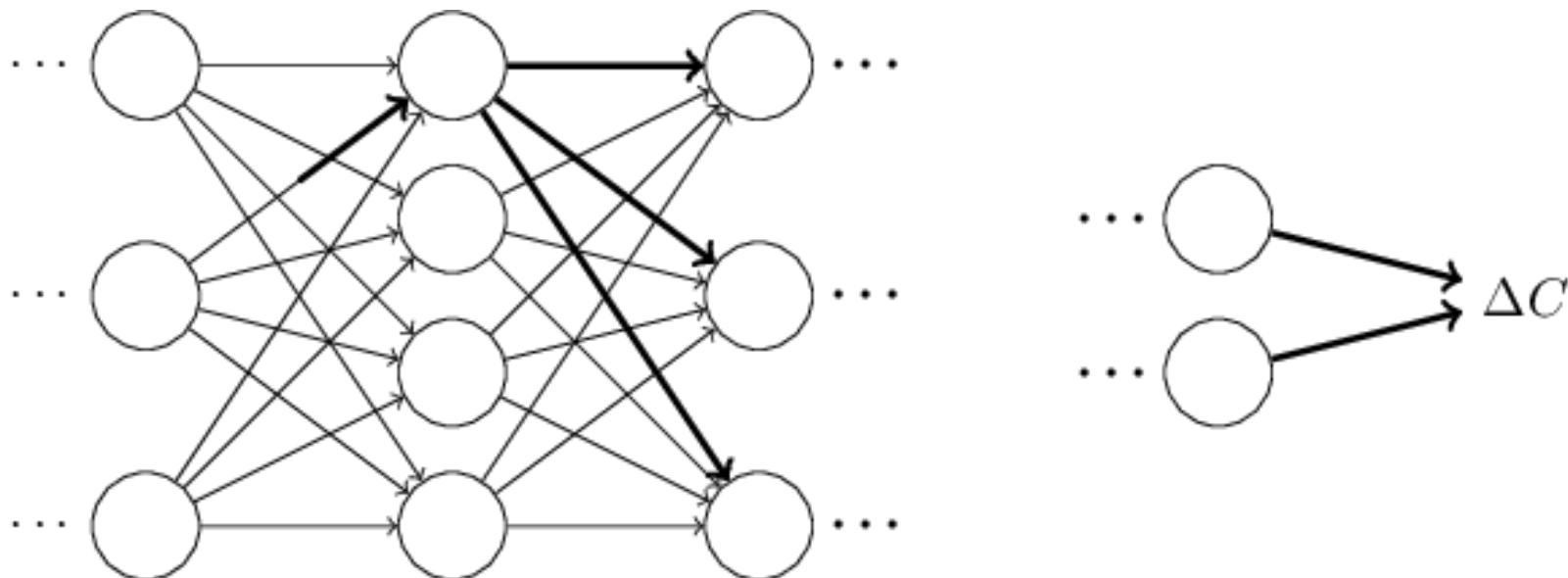
Giải thuật lan truyền ngược

- Và sau đó sẽ làm thay đổi giá trị đầu ra của tất cả các hàm kích hoạt ở các lớp phía sau



Giải thuật lan truyền ngược

- Sự thay đổi sẽ lan truyền tiếp tới các lớp sau nữa và cuối cùng sẽ ảnh hưởng tới hàm mục tiêu, gây ra một lượng thay đổi ΔC



$$\Delta C \approx \frac{\partial C}{\partial w_{jk}^l} \Delta w_{jk}^l$$

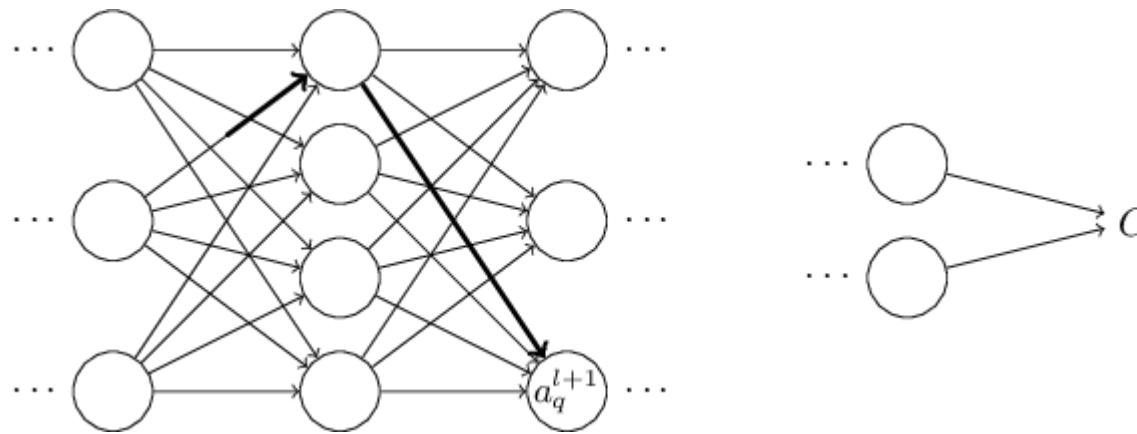
Giải thuật lan truyền ngược

- Như vậy có thể tính đạo hàm riêng của hàm mục tiêu đối với trọng số w_{jk}^l bằng cách theo dõi xem sự thay đổi của trọng số Δw_{jk}^l từng bước ảnh hưởng đến sự thay đổi của hàm mục tiêu ra sao
- Đầu tiên Δw_{jk}^l làm thay đổi hàm kích hoạt của neuron tương ứng một lượng Δa_j^l

$$\Delta a_j^l \approx \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

Giải thuật lan truyền ngược

- Sự thay đổi của hàm kích hoạt a_j^l tiếp tục ảnh hưởng tới các hàm kích hoạt ở lớp kế tiếp



$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \Delta a_j^l$$

$$\Delta a_q^{l+1} \approx \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

Giải thuật lan truyền ngược

- Sự thay đổi Δa_q^{l+1} tiếp tục ảnh hưởng các hàm kích hoạt phía sau và lan tới hàm mục tiêu.
- Ta có thể tưởng tượng ra một đường đi trong mạng từ w_{jk}^l tới hàm mục tiêu C , theo đó sự thay đổi Δw_{jk}^l sẽ dần dần ảnh hưởng tới các hàm kích hoạt trong đường đi và lan tới C . Giả sử đường đi chứa các hàm kích hoạt $a_j^l, a_q^{l+1}, \dots, a_n^{L-1}, a_m^L$ (L là số lớp của mạng). Khi đó ta có công thức:

$$\Delta C \approx \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

Giải thuật lan truyền ngược

- Sự thay đổi Δa_q^{l+1} tiếp tục ảnh hưởng các hàm kích hoạt phía sau và lan tới hàm mục tiêu.
- Ta có thể tưởng tượng ra một đường đi trong mạng từ w_{jk}^l tới hàm mục tiêu C , theo đó sự thay đổi Δw_{jk}^l sẽ dần dần ảnh hưởng tới các hàm kích hoạt trong đường đi và lan tới C . Giả sử đường đi chứa các hàm kích hoạt $a_j^l, a_q^{l+1}, \dots, a_n^{L-1}, a_m^L$ (L là số lớp của mạng). Khi đó ta có công thức:

$$\Delta C \approx \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

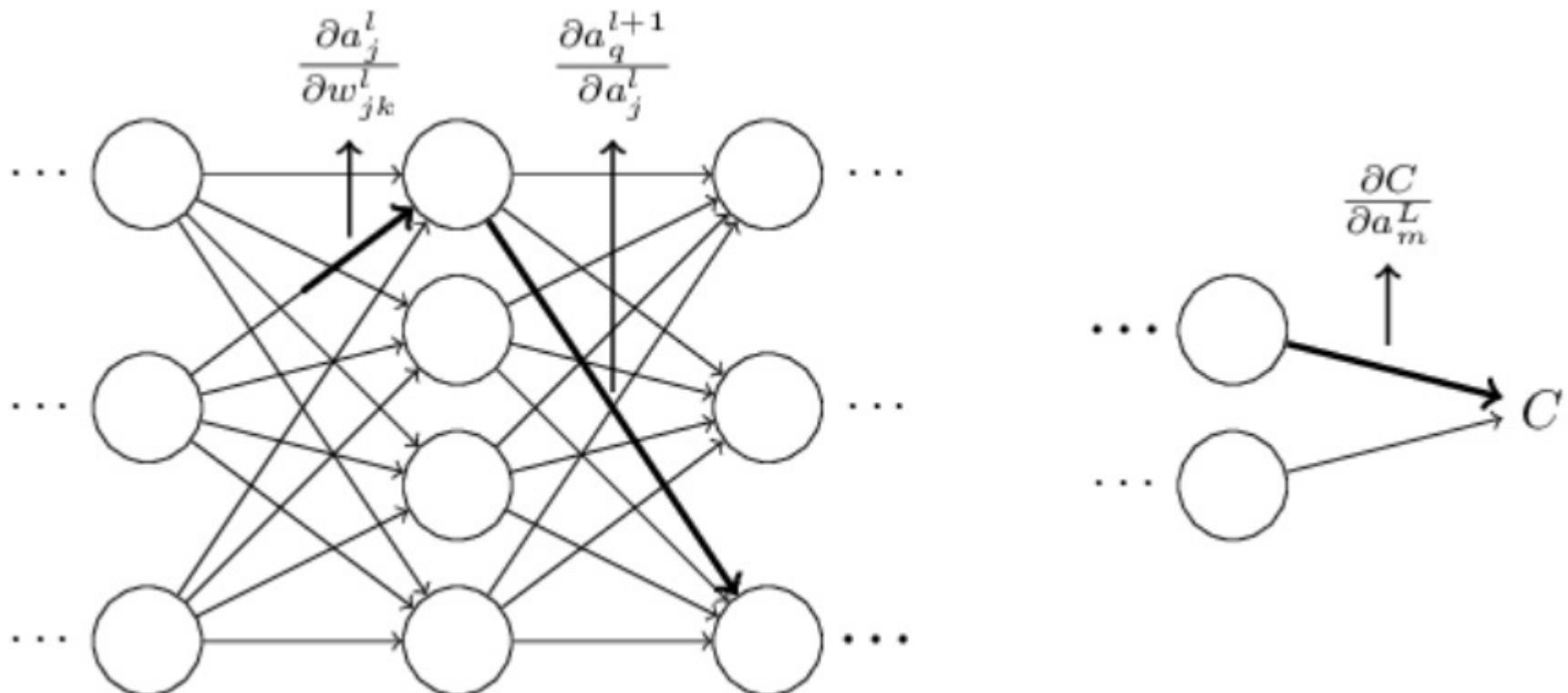
- Hiển nhiên có nhiều đường đi như vậy. Hàm mục tiêu sẽ bị thay đổi theo tất cả các đường đi:

$$\Delta C \approx \sum_{mnp\dots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \cdots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l} \Delta w_{jk}^l$$

Giải thuật lan truyền ngược

- Cuối cùng ta thu được công thức:

$$\frac{\partial C}{\partial w_{jk}^l} = \sum_{mnp\dots q} \frac{\partial C}{\partial a_m^L} \frac{\partial a_m^L}{\partial a_n^{L-1}} \frac{\partial a_n^{L-1}}{\partial a_p^{L-2}} \dots \frac{\partial a_q^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{jk}^l}$$



Thư viện mở

K Keras


Caffe2


TensorFlow

PYTORCH

Tài liệu tham khảo

- Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, 2 (4), 303-314
- Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks, 4(2), 251–257



25
YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

**Thank you for
your attention!**

