
2102470 Học máy

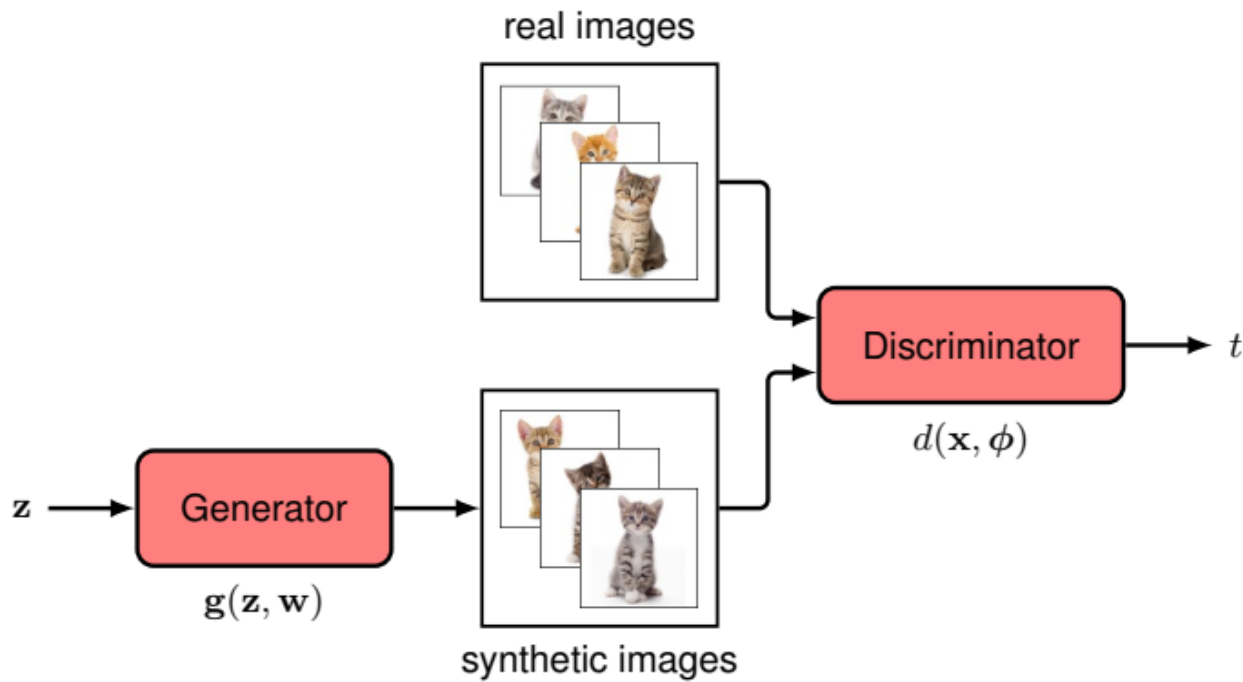
Bài giảng: 2.4 Neural Networks

Chương 2: Xấp xỉ và phân lớp

Ôn lại bài học trước

- Bạn có nhớ ? % ?





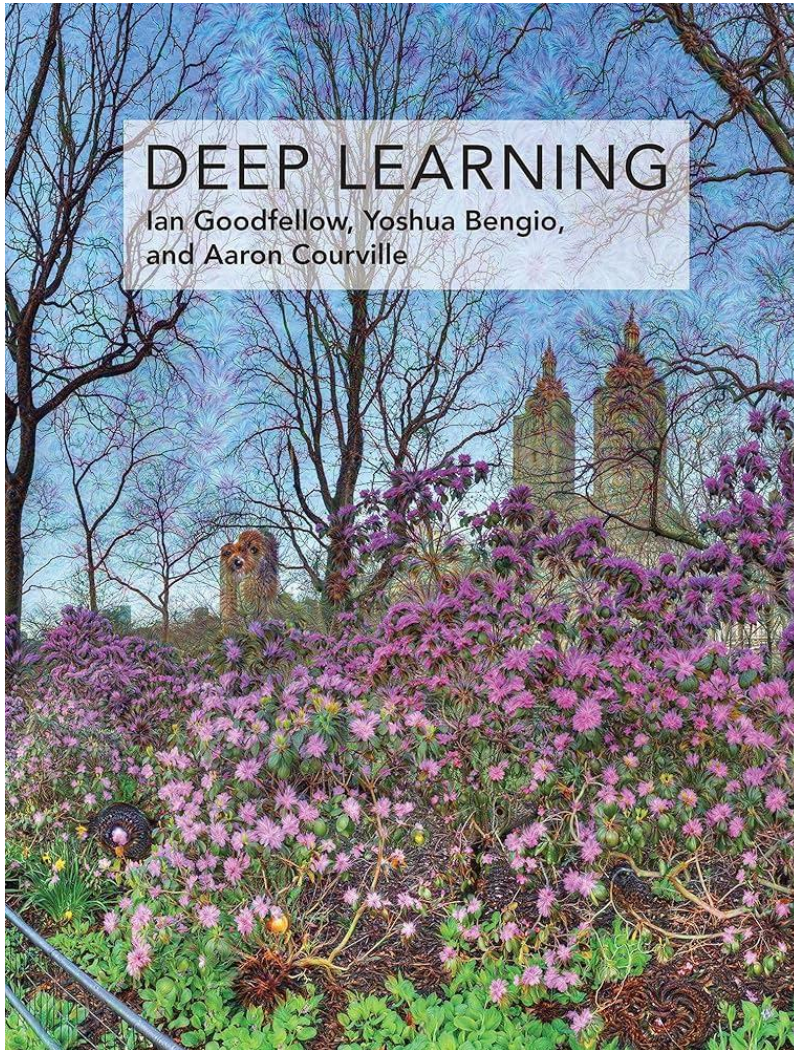
<https://www.bishopbook.com/>

<https://generated.photos/human-generator/66e258a8099bb6000d892905>

Nội dung chính

- 4.1 Giới thiệu về mạng neuron nhân tạo
- 4.2 Perceptron
- 4.3 Backpropagation
- 4.4 Các mạng neuron nhân tạo và ứng dụng

Deep Learning



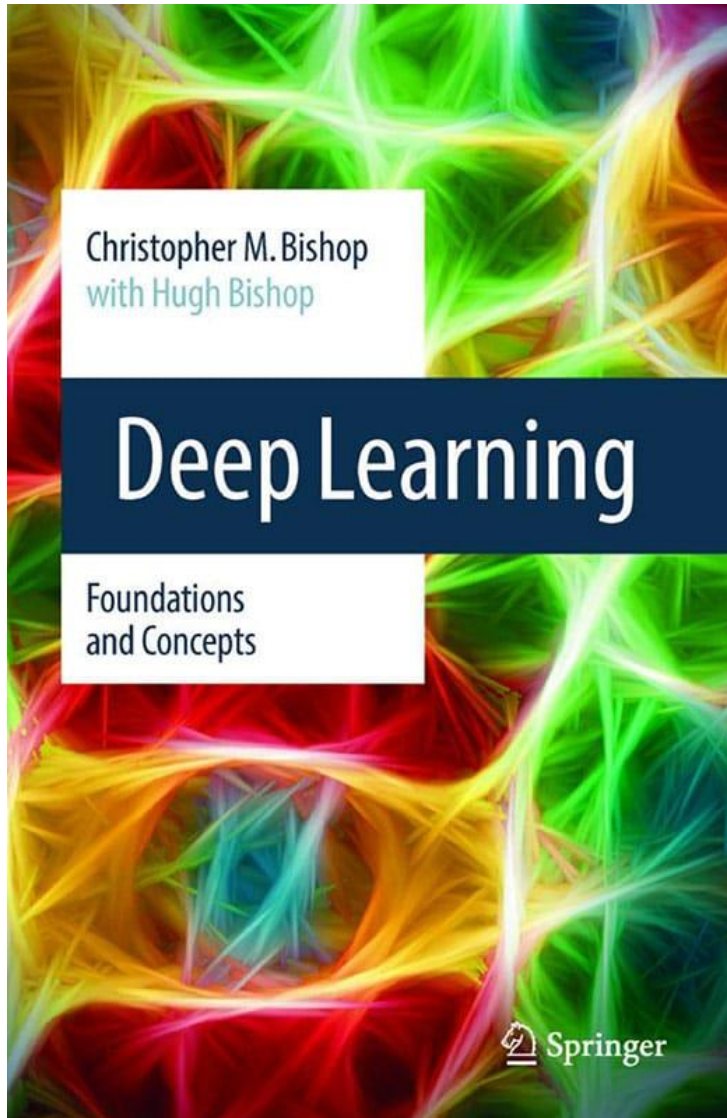
Deep Learning

An MIT Press book

Ian Goodfellow and Yoshua Bengio and Aaron Courville

<https://www.deeplearningbook.org/>

Deep Learning



Chris Bishop is a Technical Fellow at Microsoft and is the Director of Microsoft Research AI4Science.

He is a Fellow of Darwin College Cambridge, a Fellow of the Royal Academy of Engineering, and a Fellow of the Royal Society.



Hugh Bishop is an Applied Scientist at Wayve, a deep learning autonomous driving company in London where he designs and trains deep neural networks.

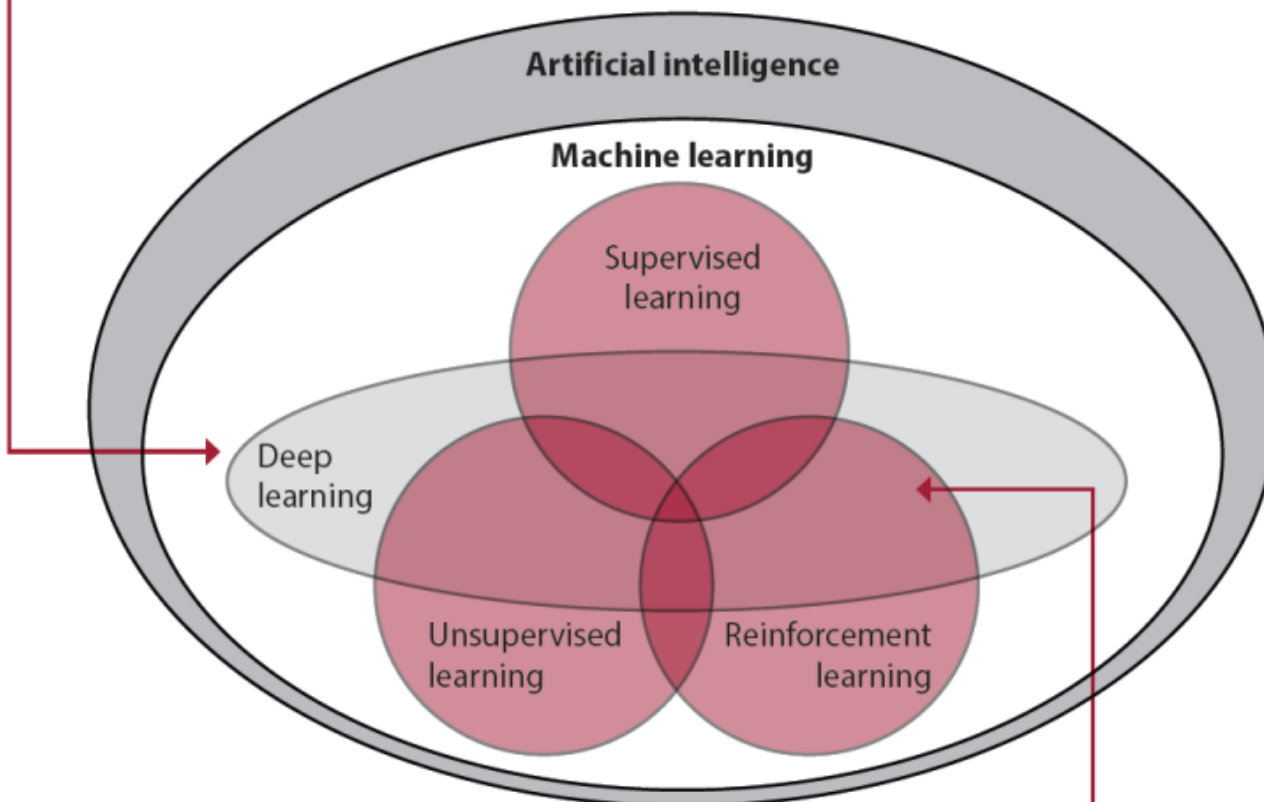
He completed his MPhil in Machine Learning and Machine Intelligence at Cambridge University.

<https://www.bishopbook.com/>

4.1 Giới thiệu

- Deep learning

(i) The important thing here is deep learning is a toolbox, and any advancement in the field of deep learning is felt in all of machine learning.

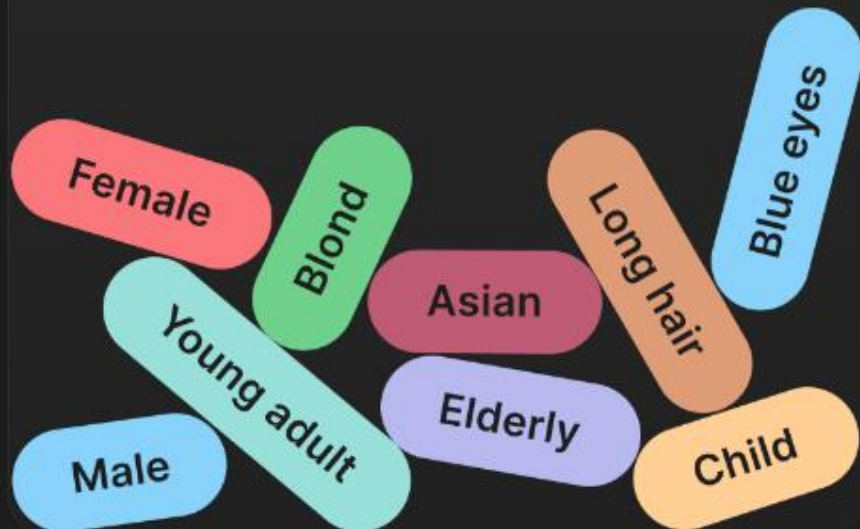


(a) Deep reinforcement learning is the intersection of reinforcement learning and deep learning.

Giới thiệu

Explore

Quickly find exactly what you are looking for by using filters in our Faces database or uploading a similar face to Anonymizer.



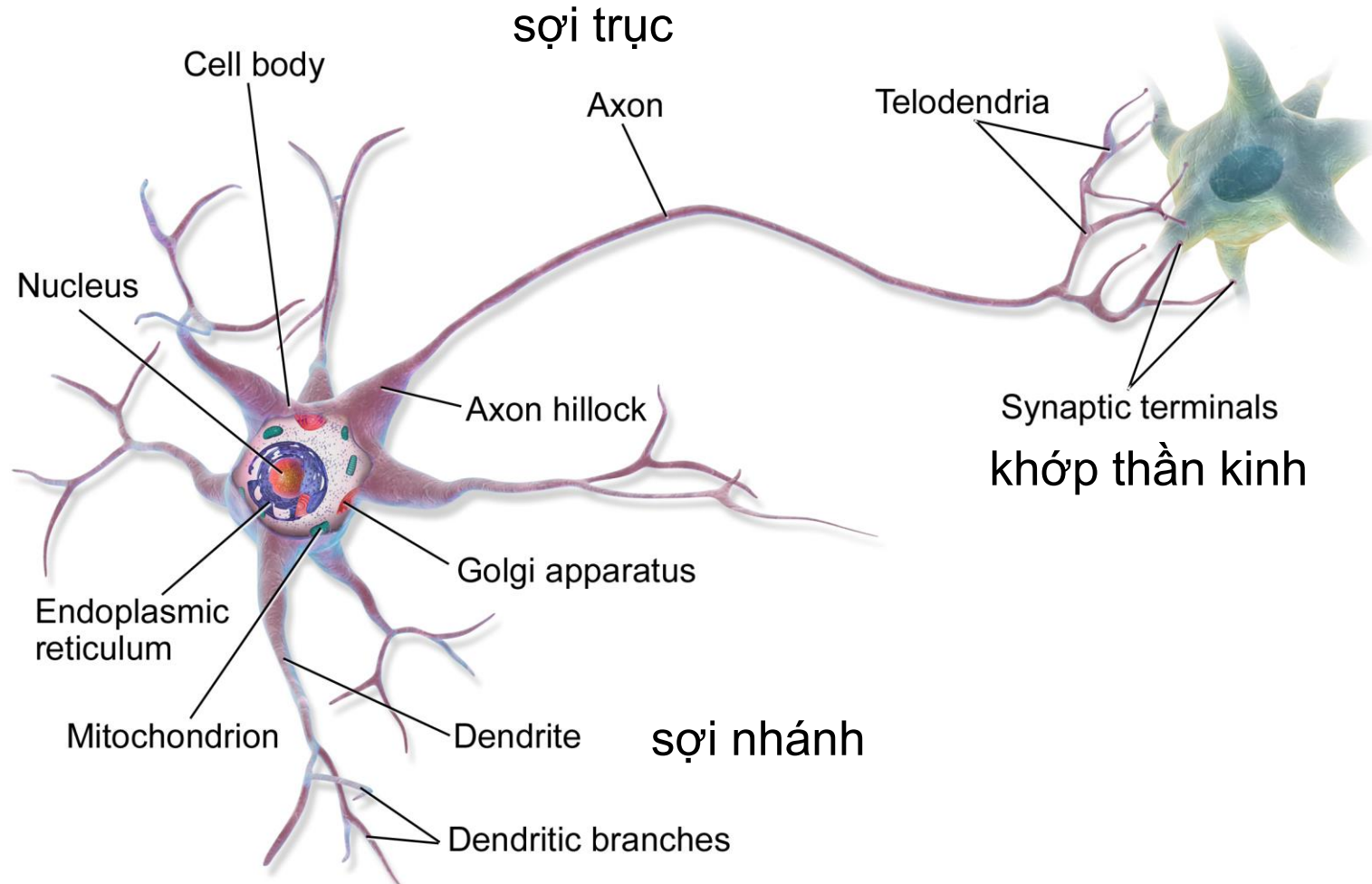
Generate

Create unique photo-realistic faces or full-body humans with your parameters. Or upload and modify your photos in Face Generator and Human Generator.



Biological neuron

Biological neuron => artificial neuron



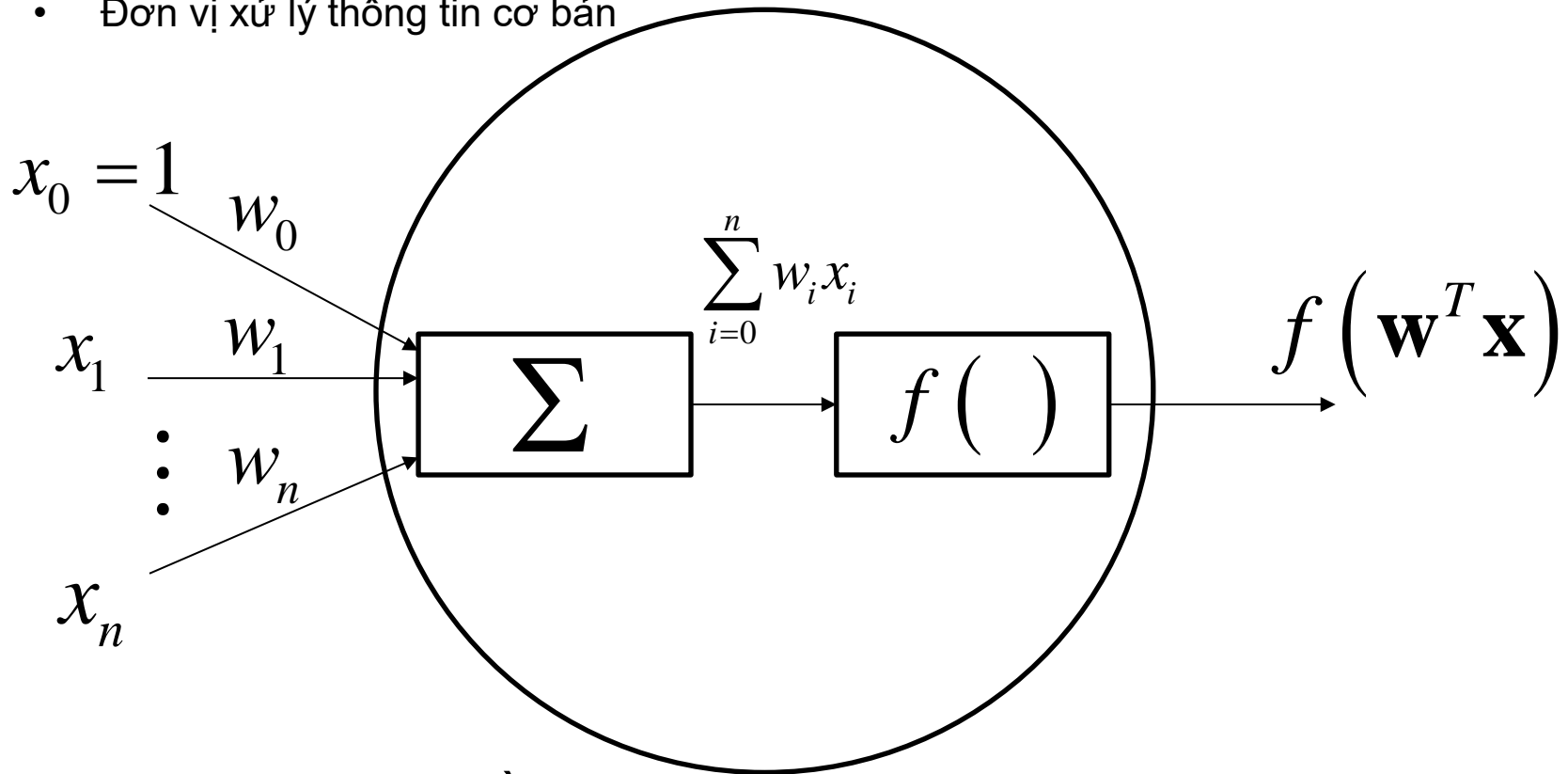
By BruceBlaus - Own work, CC BY 3.0,
<https://commons.wikimedia.org/w/index.php?curid=28761830>

Mạng neuron nhân tạo

- Artificial neural network (ANN)
 - Lấy cảm hứng từ cấu trúc và chức năng của mạng neuron sinh học trong não động vật
 - Xử lý thông tin: song song, phân tán
 - Mong muốn: khả năng học, xử lý thông tin giống con người

Neuron

- Đơn vị xử lý thông tin cơ bản



x_i : Các tín hiệu đầu vào

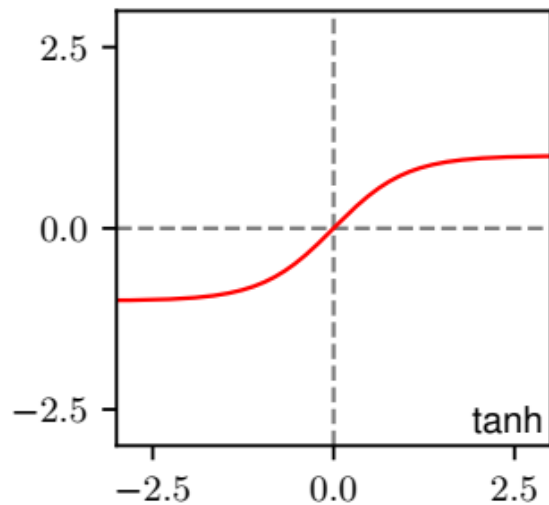
w_i : Các trọng số

$f(\)$: Activation/transfer function
hàm kích hoạt/truyền

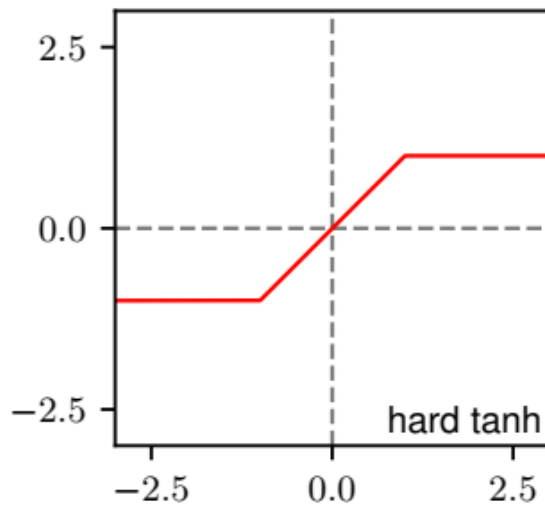
Hàm kích hoạt/truyền

- Đóng vai trò quan trọng
 - Xác định cách các đầu vào được chuyển đổi thành đầu ra
 - Ảnh hưởng đến quá trình học và hiệu suất của NN
 - Chú ý lựa chọn hàm kích hoạt phù hợp
 - Vấn đề cần giải quyết
 - Cấu trúc của NN

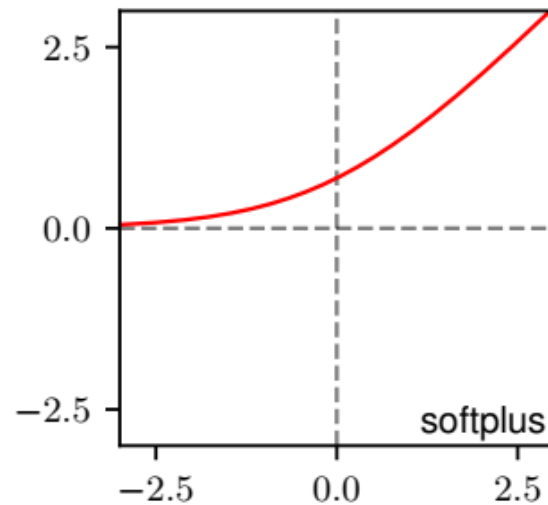
Hàm kích hoạt



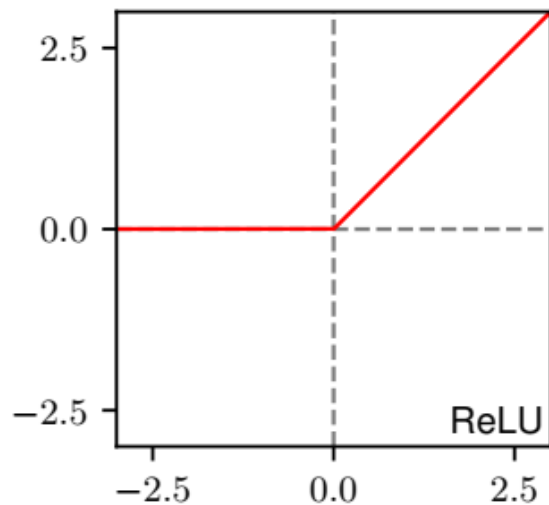
(a)



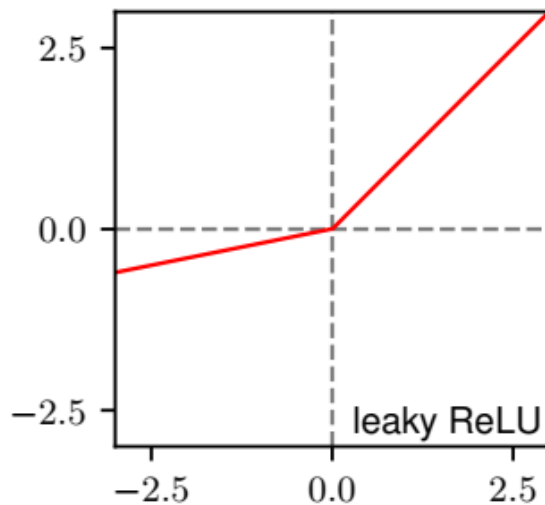
(b)



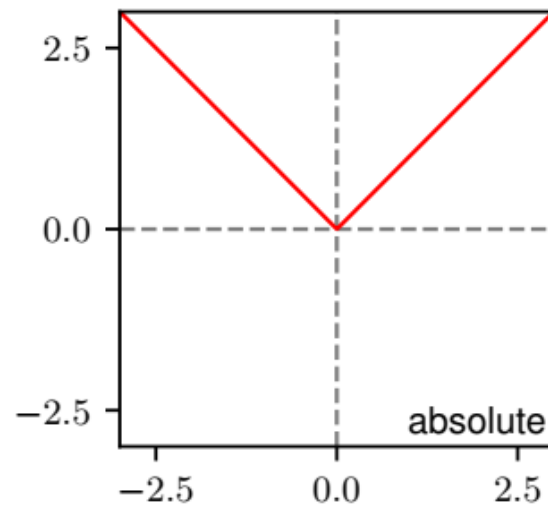
(c)



(d)



(e)



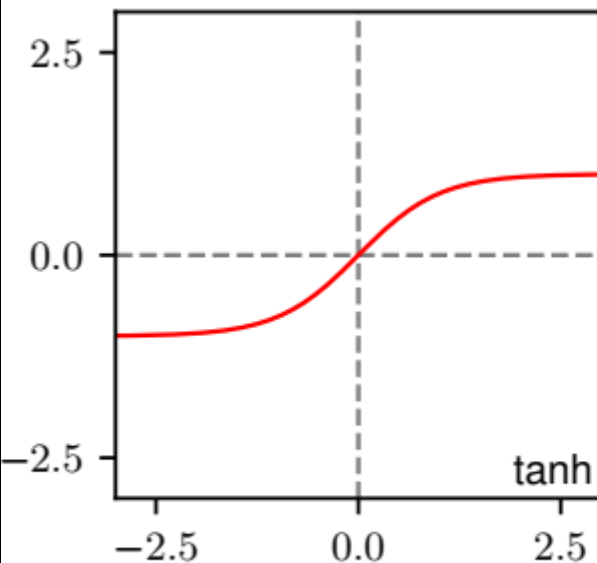
(f)

Các hàm kích hoạt

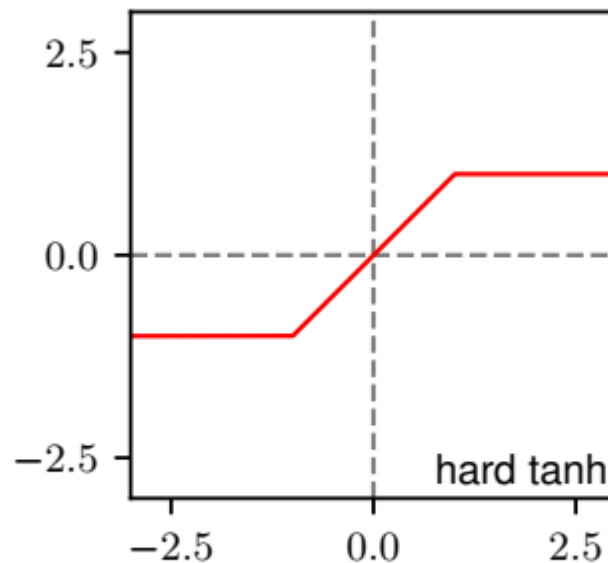
(a) tanh
$$h(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

(b) hard tanh
$$h(x) = \max(-1, \min(1, x))$$

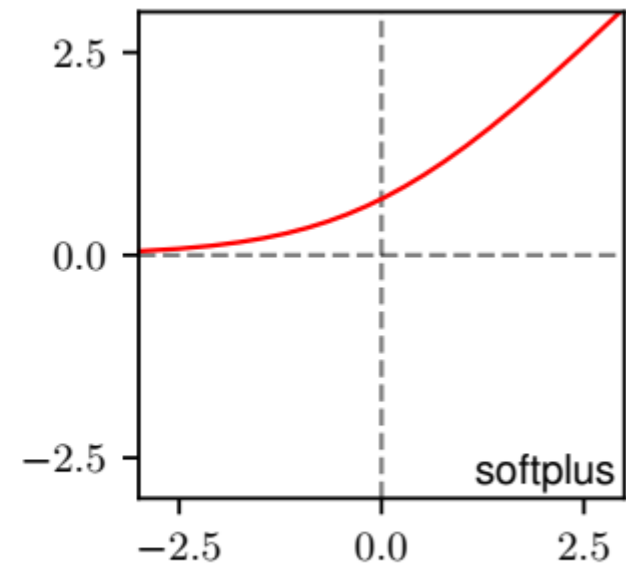
(c) softplus
$$h(x) = \ln(1 + \exp(x))$$



(a)



(b)



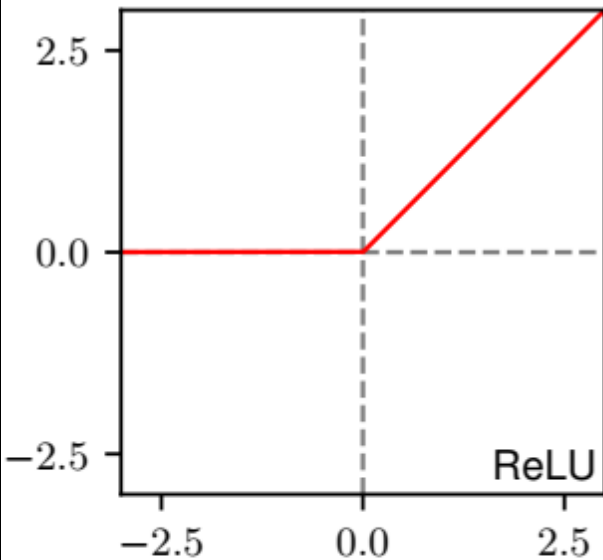
(c)

Các hàm kích hoạt

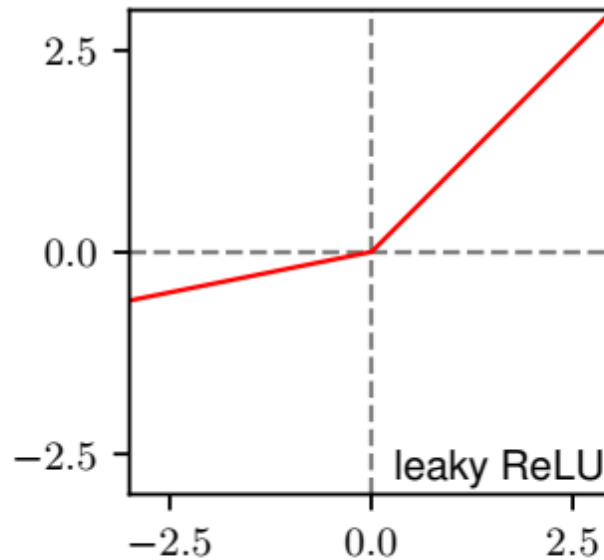
(d) **ReLU** $h(x) = \max(0, x)$

(e) **leaky ReLU** $h(x) = \max(0, x) + \alpha \min(0, x)$
 $0 < \alpha < 1$

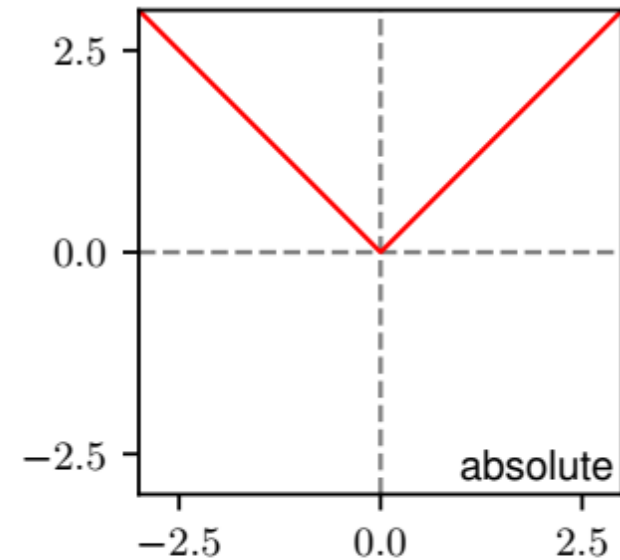
(f) **absolute** $h(x) = |x|$



(d)



(e)



(f)

Rectified linear unit (ReLU)

- Đây là một trong những hàm kích hoạt có hiệu suất tốt nhất và được sử dụng rộng rãi
 - Việc giới thiệu ReLU đã mang lại sự cải thiện lớn về hiệu quả huấn luyện so với các hàm kích hoạt sigmoid trước đây
 - Ít nhạy cảm hơn nhiều với việc khởi tạo ngẫu nhiên các trọng số.
 - Phù hợp cho việc thực hiện với độ chính xác thấp, ví dụ như 8 bit cố định so với 64 bit dấu phẩy động
- Một cách nghiêm ngặt, đạo hàm của hàm ReLU không được xác định khi $x = 0$
 - nhưng trên thực tế, điều này có thể được bỏ qua một cách an toàn

Cấu trúc NN

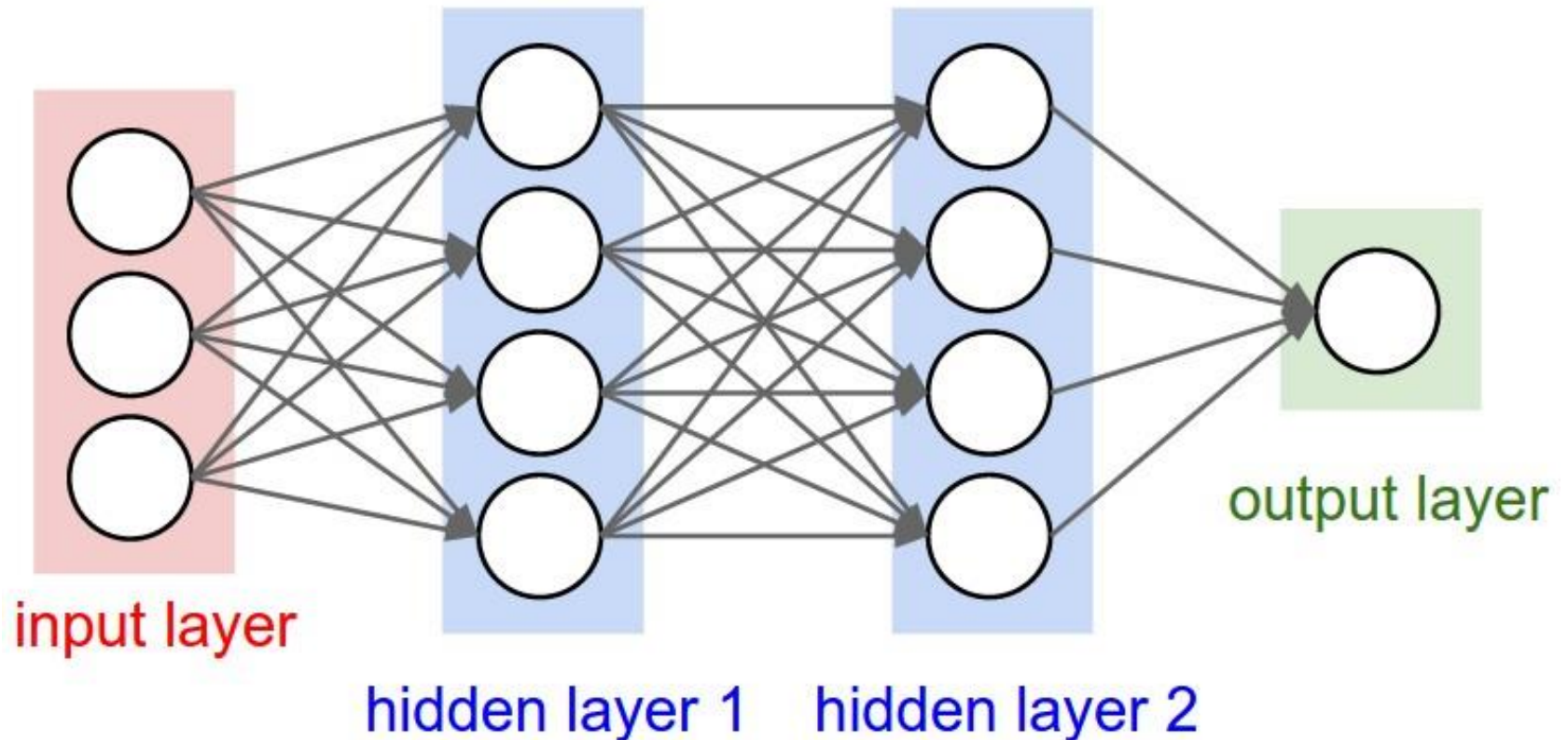
- Khi xây dựng một NN cần chú ý
 - Số lượng các tín hiệu đầu vào
 - Số lượng các tầng/lớp (layer)
 - Input layer, hidden layers, output layer
 - Số lượng neuron trong mỗi tầng
 - Số lượng các kết nối giữa các neuron
 - Cách thức các neuron liên kết với nhau
 - Trong một tầng
 - Giữa các tầng
 - Số lượng các tín hiệu đầu ra

Cấu trúc NN

- Fully connected: liên kết đầy đủ
- Feed-forward network: mạng lan truyền tiến
- Feedback network: mạng phản hồi
 - Lateral feedback: phản hồi bên
 - Recurrent network: mạng hồi quy
- Deep NN: ANN với nhiều hidden layers

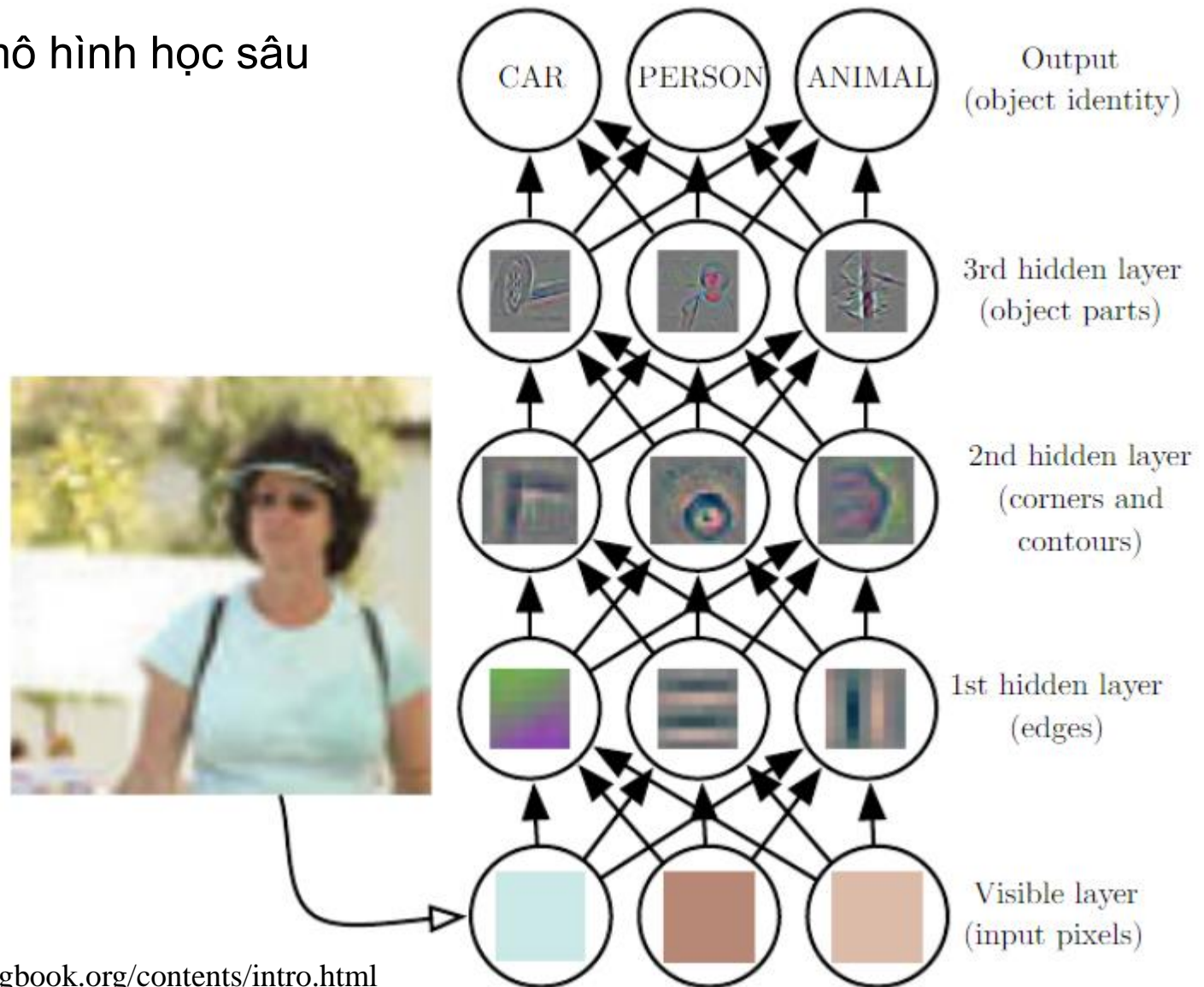
Ví dụ

- Feed-forward fully connected NN



Ví dụ

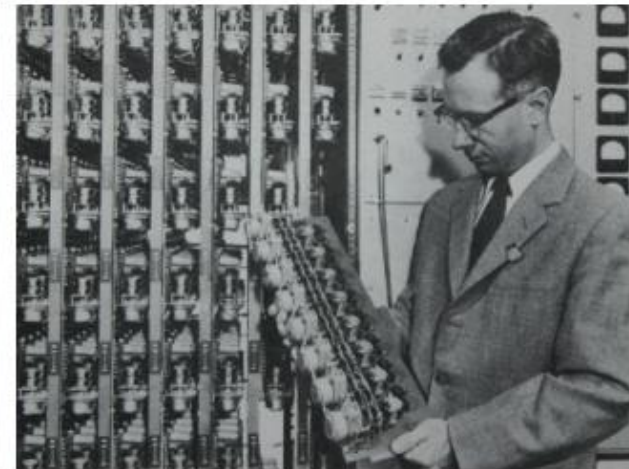
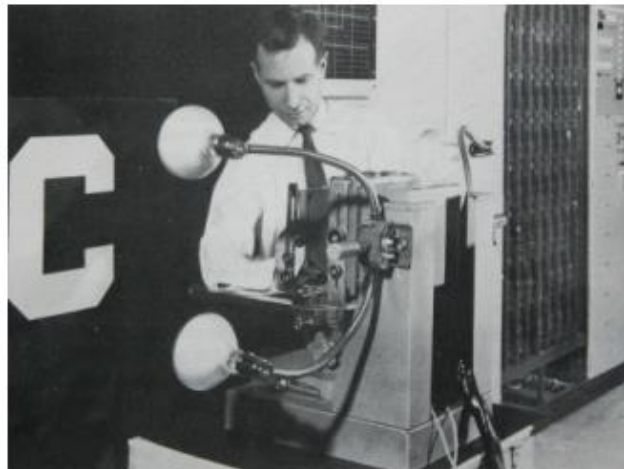
Minh họa một mô hình học sâu



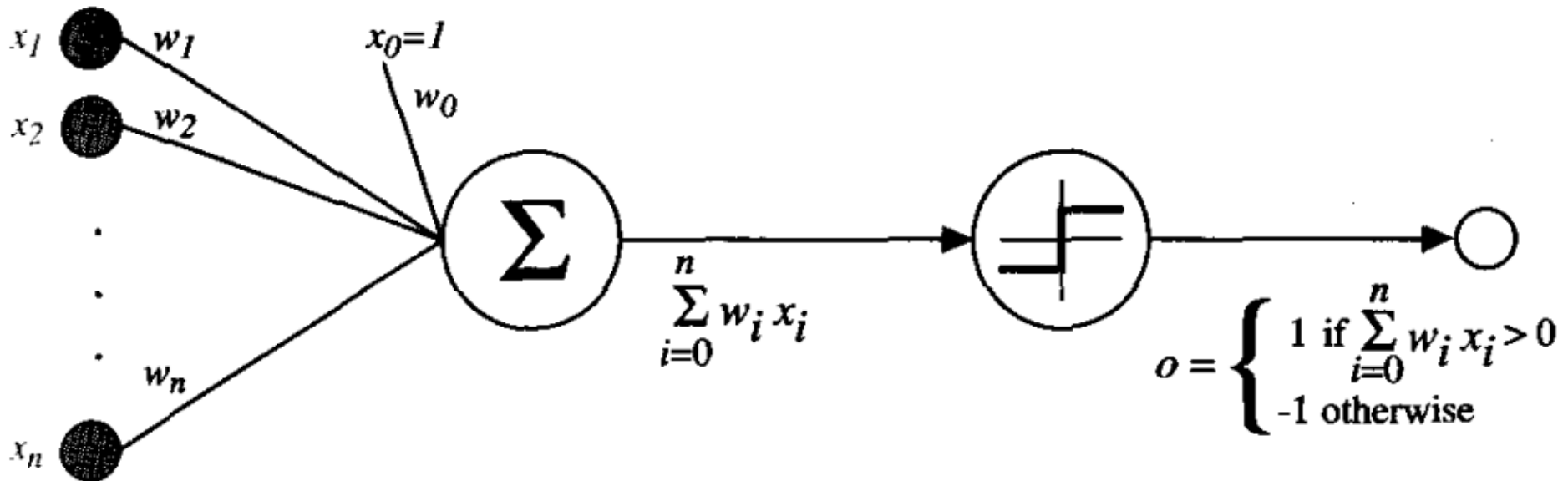
<https://www.deeplearningbook.org/contents/intro.html>

4.2 Perceptron

- Perceptron (McCulloch–Pitts neuron) phát minh vào năm 1943 bởi Warren McCulloch and Walter Pitts
- F. Rosenblatt mô tả chi tiết vào những năm 1950



Perceptron



$w_0 = b$:bias

$$o = \text{sgn}(\mathbf{w}^T \mathbf{x})$$

Bộ phân lớp nhị phân (tuyến tính)?

Có thể dùng hàm Heaviside làm hàm kích hoạt

$$H(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

$$\text{sgn}(x) := \begin{cases} -1 & x < 0 \\ 0 & x = 0 \\ 1 & x > 0 \end{cases}$$

Perceptron

- Huấn luyện ~ tìm vector trọng số \mathbf{w}
 - B1: Khởi tạo ngẫu nhiên vector trọng số
 - B2: Đưa mẫu huấn luyện vào đầu vào
 - B3: Tính giá trị tại đầu ra $o = \hat{y}$
 - B4: Cập nhật trọng số để giá trị tại đầu ra đúng với giá trị mong muốn/nhãn (y)

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(y - \hat{y})\mathbf{x}(n)$$

- B5: Lặp lại bước 3 và 4 cho tới khi không còn lỗi, hoặc số lần lặp đạt tới mức ngưỡng đặt trước

Perceptron

- Một perceptron có thể được dùng để biểu diễn các hàm boolean khác nhau như
 - AND, OR, NOT
 - NAND, NOR

Ôn lại

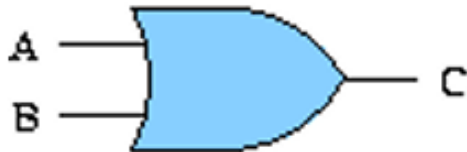
AND



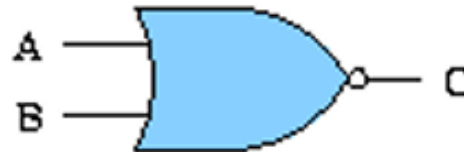
NAND



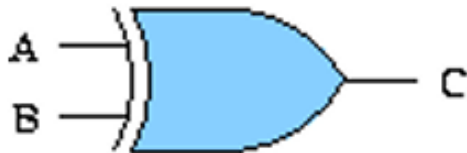
OR



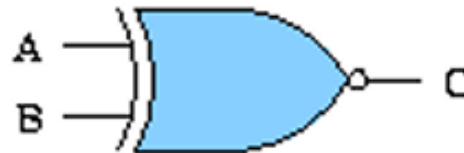
NOR



XOR



XNOR



A	B	AND	OR	XOR	NAND	NOR	XNOR
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	0
1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	1

{AND, OR, NOT}

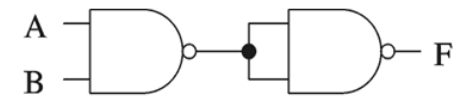
{AND, NOT}

{OR, NOT}

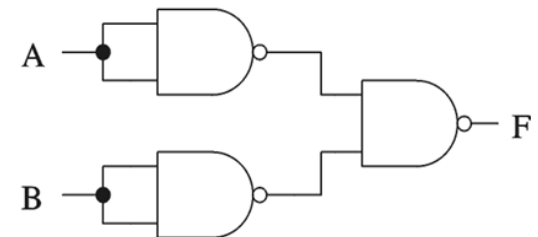
{NAND}

{NOR}

Ví dụ: Chỉ dùng NAND



AND gate



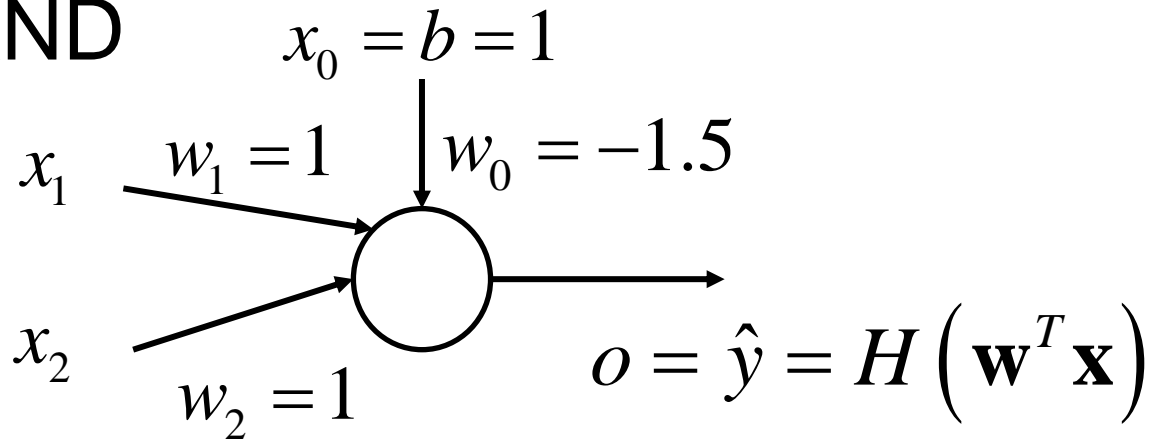
OR gate



NOT gate

Perceptron

- AND



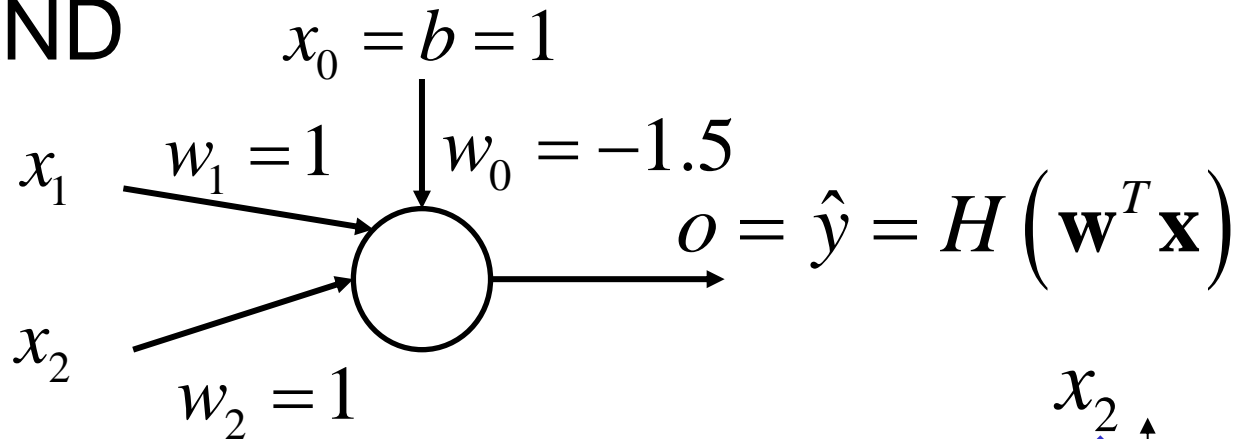
x_1	x_2	AND	$o = \hat{y}$
0	0		
0	1		
1	0		
1	1		

hàm kích hoạt: hàm Heaviside

$$H(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

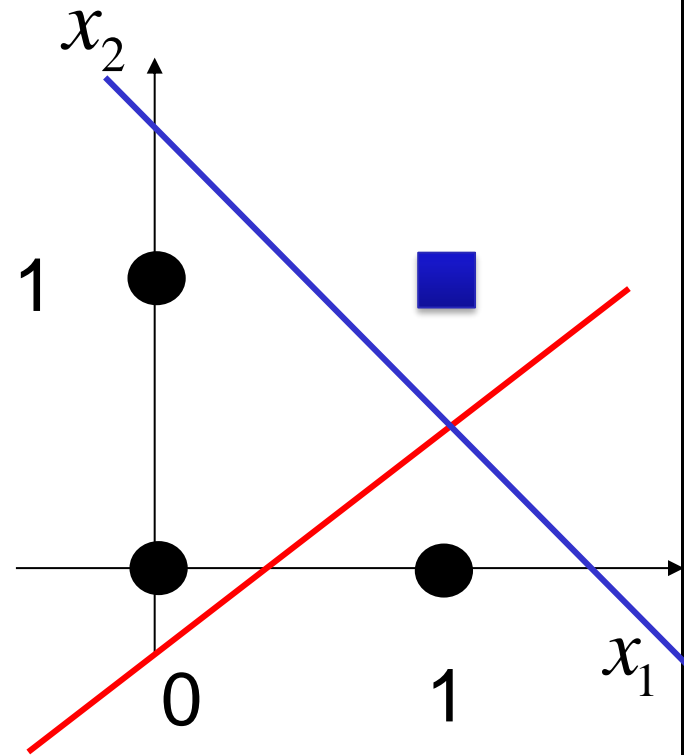
Perceptron

- AND



x_1	x_2	AND	$o = \hat{y}$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

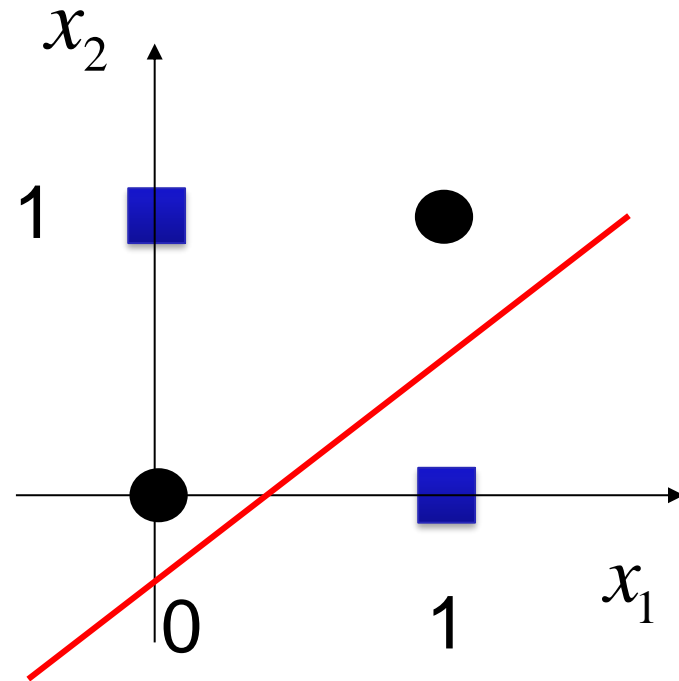
$$o = \hat{y} = H(-1.5 + x_1 + x_2)$$



Perceptron

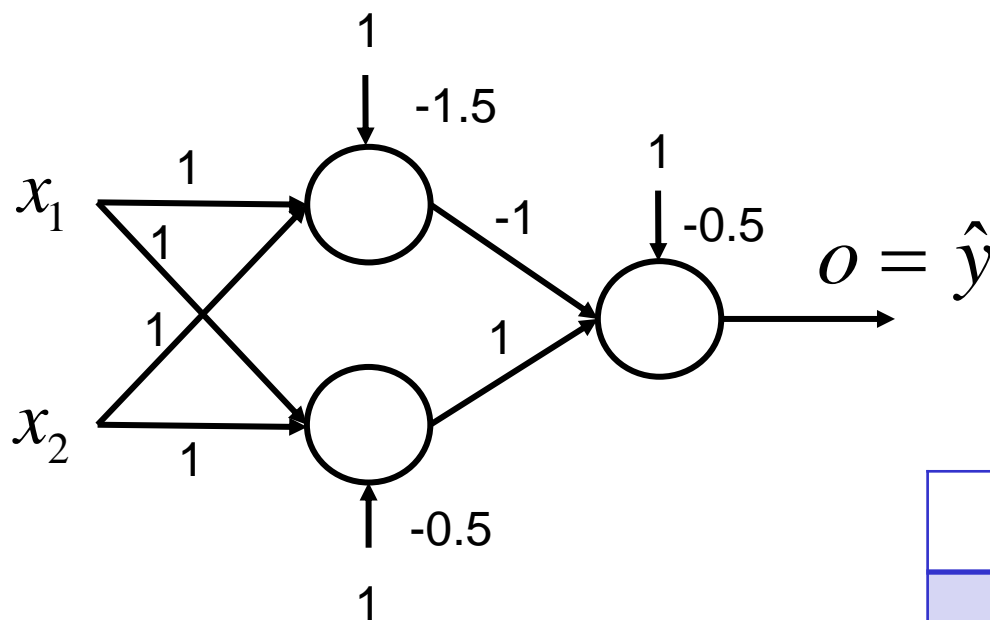
- Có thể thực hiện được hàm XOR với perceptron 1 lớp?

x_1	x_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0



Multilayer perceptron (MLP)

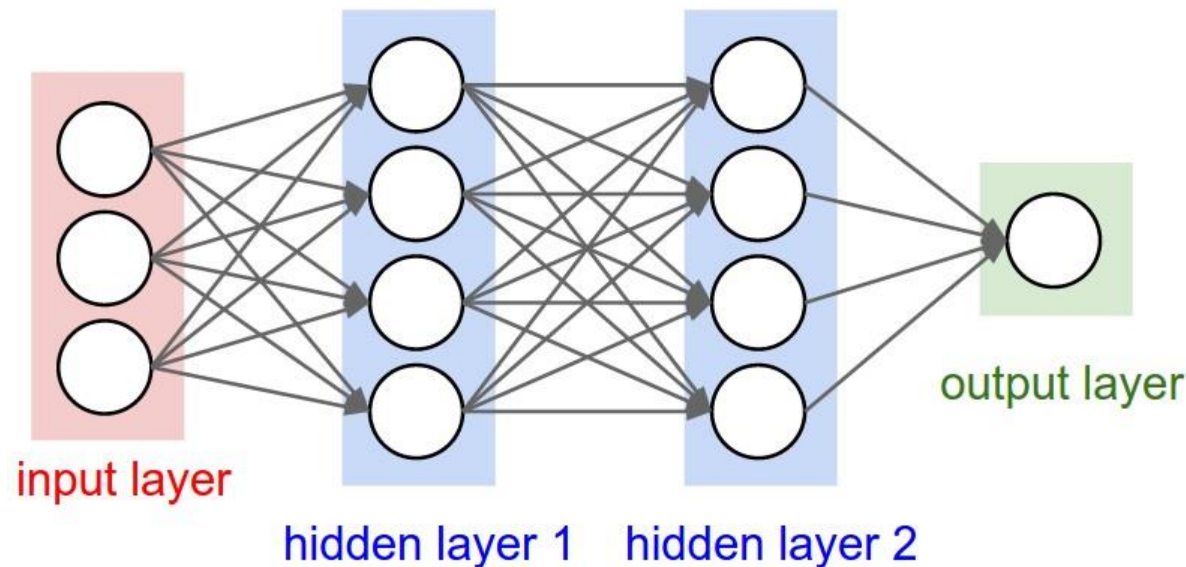
- Hàm XOR không thể được biểu diễn bằng perceptron một lớp => perceptron nhiều lớp



x_1	x_2	XOR	$o = \hat{y}$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Mạng NN nhiều lớp

- Thiết kế các NN để thực hiện nhiều tác vụ phức tạp?
 - Cấu trúc
 - Tham số



Mạng neuron 2 lớp

Lớp 1

$a_j^{(1)}$: pre-activation

$$a_j^{(1)} = w_{j0}^{(1)} + \sum_{i=1}^D w_{ji}^{(1)} x_i$$

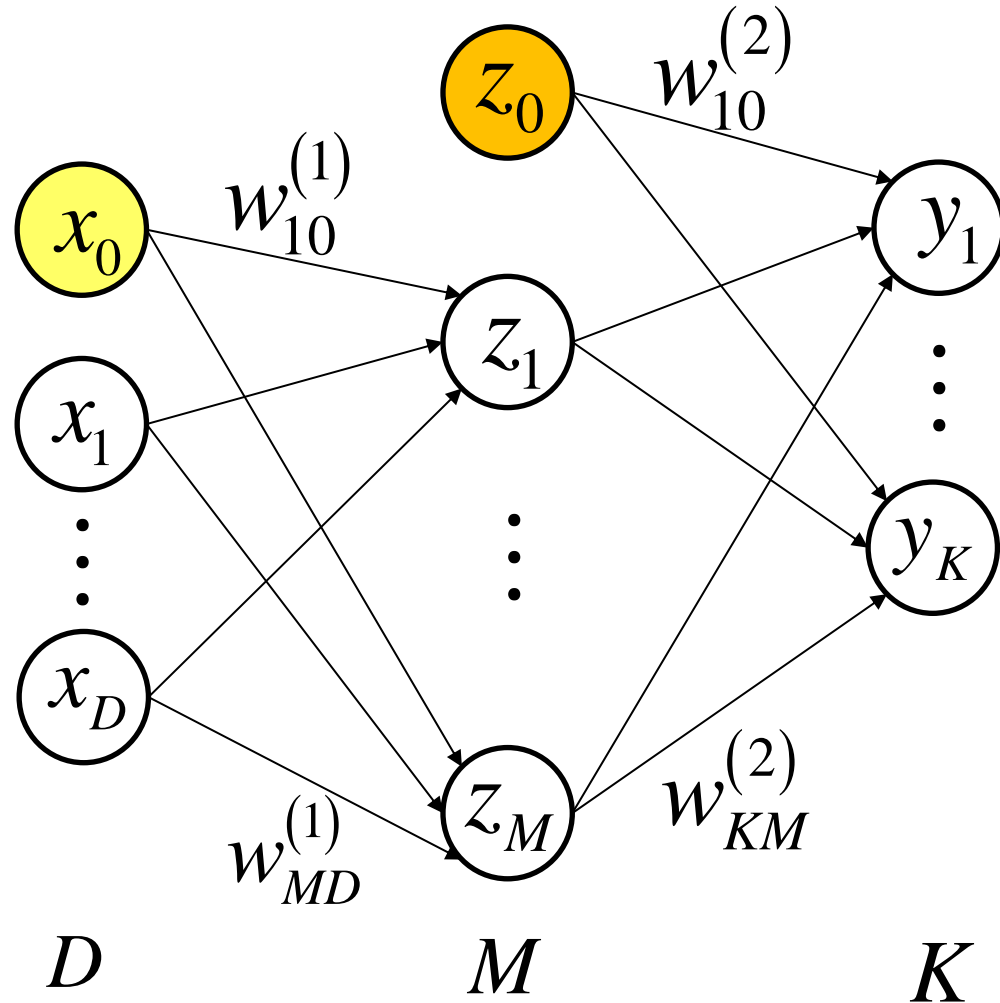
$$z_j^{(1)} = h(a_j^{(1)})$$

$$j = 1, \dots, M$$

$w_{ji}^{(1)}$: trọng số

$w_{j0}^{(1)}$: bias

$h(\)$: hàm kích hoạt phi tuyến



Mạng neuron 2 lớp

Lớp 2

$a_k^{(2)}$: pre-activation

$$a_k^{(2)} = w_{k0}^{(2)} + \sum_{j=1}^M w_{kj}^{(2)} z_j^{(1)}$$

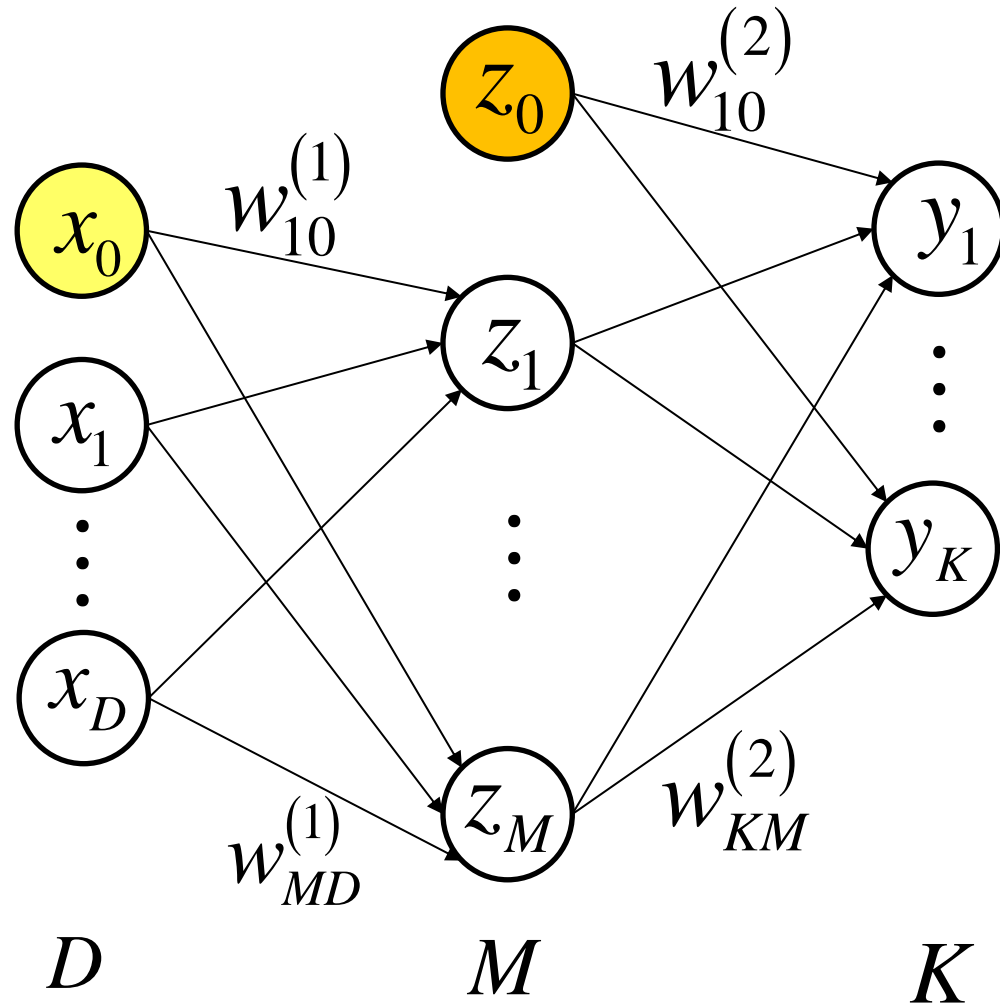
$$y_k^{(2)} = f(a_k^{(2)})$$

$$k = 1, \dots, K$$

$w_{kj}^{(2)}$: trọng số

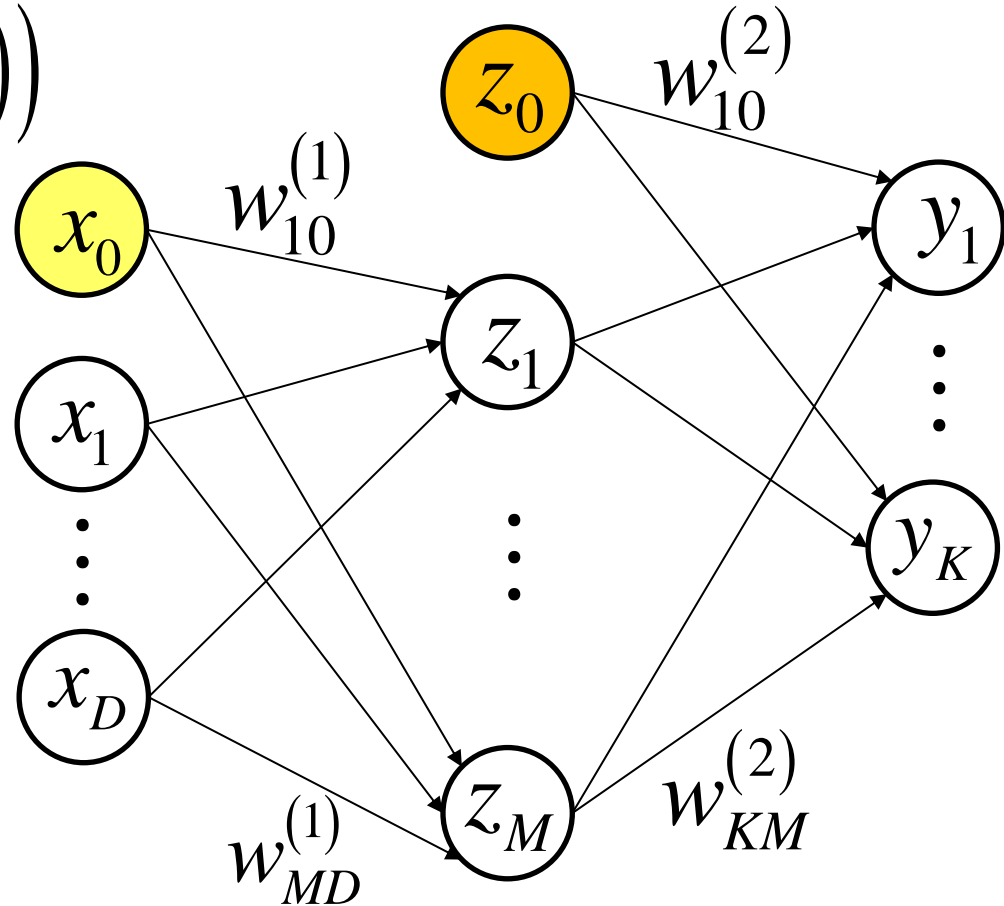
$w_{k0}^{(2)}$: bias

$f(\)$: hàm kích hoạt phi tuyến



Mạng neuron 2 lớp

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = f\left(\mathbf{W}^{(2)} h\left(\mathbf{W}^{(1)} \mathbf{x}\right)\right)$$



$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^M w_{kj}^{(2)} z_j^{(1)}\right)$$

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^M w_{kj}^{(2)} h\left(\sum_{i=0}^D w_{ji}^{(1)} x_i\right)\right)$$

Mạng neuron 2 lớp

- Universal approximation

$$f(x) = x^2$$

?

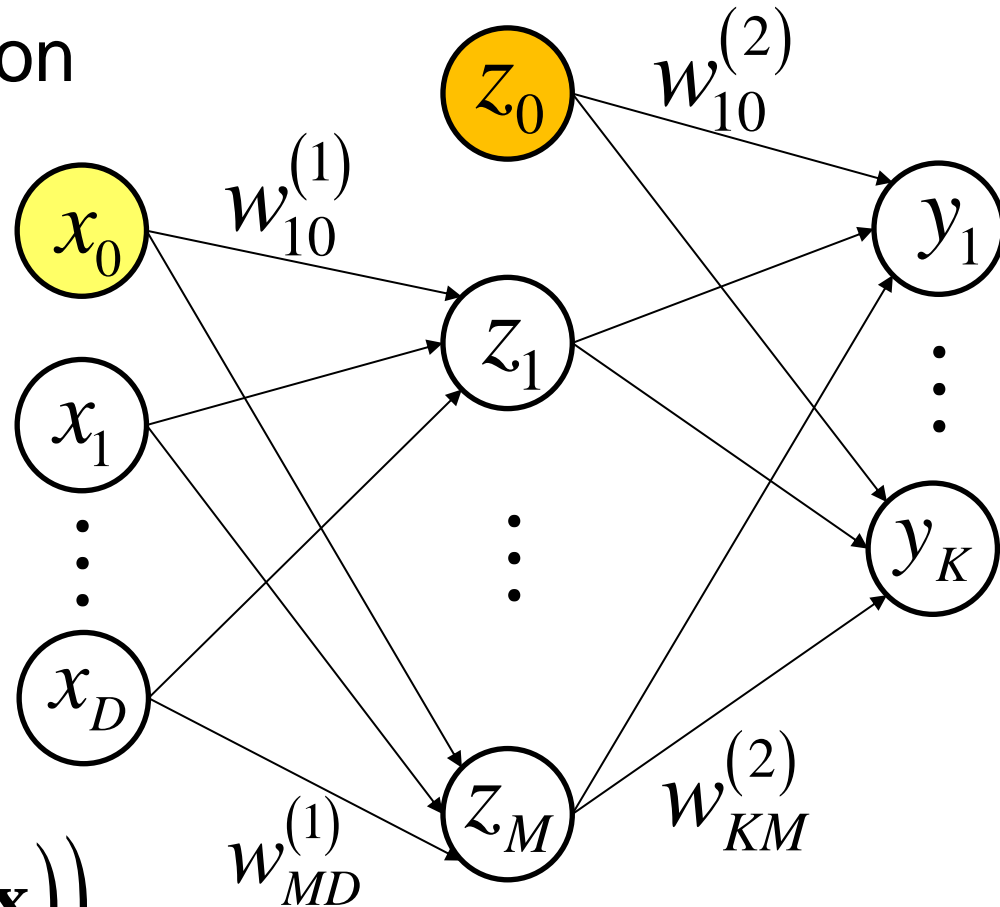
$$f(x) = \sin(x)$$

$$f(x) = |x|$$

$$f(x) = H(x)$$

$$\mathbf{y}(\mathbf{x}, \mathbf{w}) = f\left(\mathbf{W}^{(2)} h\left(\mathbf{W}^{(1)} \mathbf{x}\right)\right)$$

$$y_k(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=0}^M w_{kj}^{(2)} h\left(\sum_{i=0}^D w_{ji}^{(1)} x_i\right)\right)$$



Universal approximation

- Các tính chất xấp xỉ của mạng truyền thẳng hai lớp đã được nghiên cứu rộng rãi vào những năm 1980
 - Các định lý cho thấy rằng, đối với nhiều hàm kích hoạt, các mạng có thể xấp xỉ bất kỳ hàm nào được xác định trên một tập hợp con liên tục của \mathbb{R}^D với độ chính xác tùy ý
- Kết quả tương tự cũng đúng đối với các hàm từ bất kỳ không gian rời rạc hữu hạn chiều nào đến bất kỳ không gian nào khác
- Do đó, các mạng neuron có thể được coi là các bộ xấp xỉ phổ quát (universal approximators)

Universal approximation

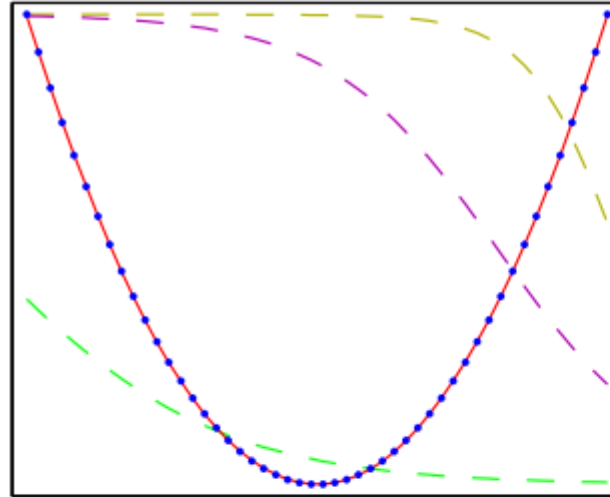
- Chú ý:
 - Chỉ cho chúng ta biết rằng tồn tại một mạng có thể biểu diễn hàm mà ta quan tâm
 - không chỉ ra việc liệu một thuật toán học có thể tìm thấy một mạng như vậy hay không
 - không bao giờ có thể tìm thấy một thuật toán học máy thực sự phổ quát (“no free lunch theorem”)
 - Trong ứng dụng thực tế, có thể có nhiều ưu điểm khi sử dụng các mạng có nhiều hơn hai lớp, có thể học các biểu diễn nội bộ phân cấp => học sâu (deep learning)

Ví dụ

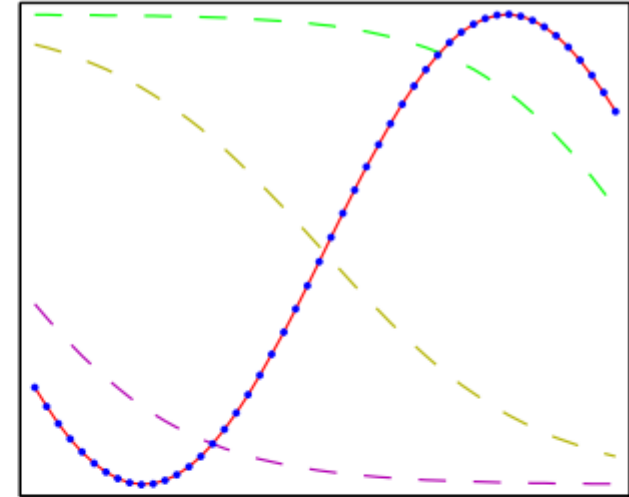
Sử dụng NN với 2 lớp,
3 hidden units

Hàm kích hoạt: tanh, và
tuyến tính (cho đầu ra)

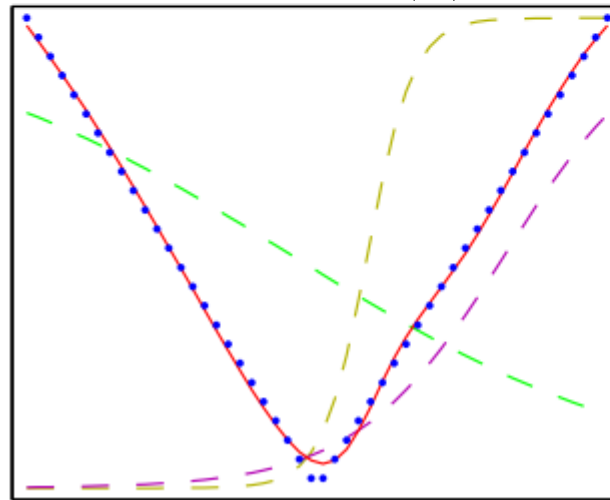
Tập dữ liệu huấn luyện:
50 mẫu



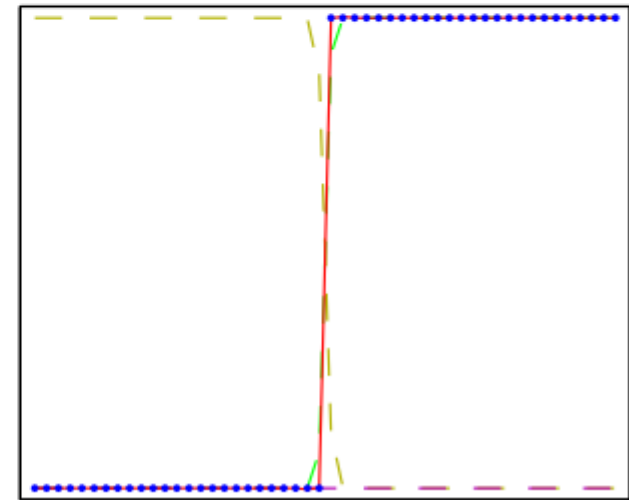
(a) $f(x) = x^2$



(b) $f(x) = \sin(x)$



(c) $f(x) = |x|$



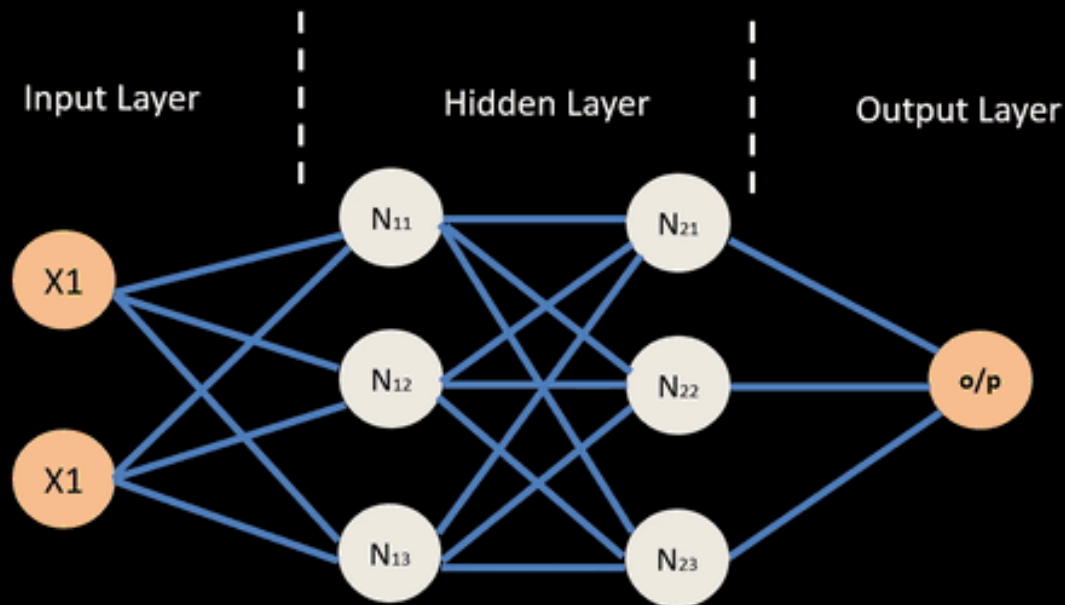
(d) $f(x) = H(x)$

4.3 Backpropagation

- Lan truyền ngược
 - Giải thuật dùng để tìm các trọng số của NN dựa trên Gradient Descent

Backpropagation

Neural Network – Backpropagation

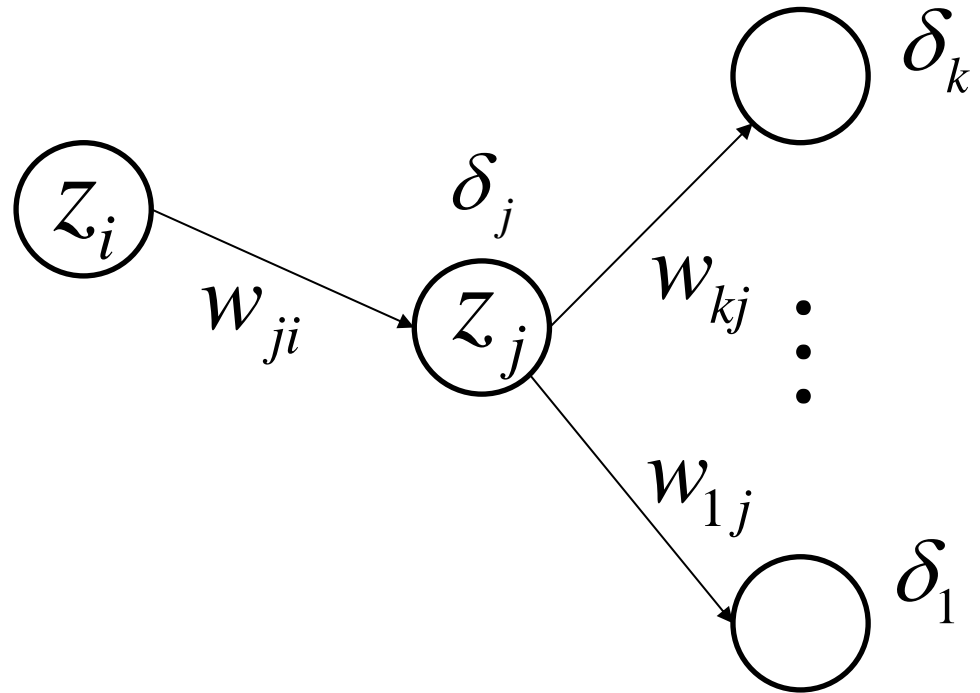


Backpropagation

forward propagation

$$a_j = \sum_i w_{ji} z_i$$

$$z_j = h(a_j)$$



Backpropagation

Chú ý:

- đạo hàm của hàm lỗi

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

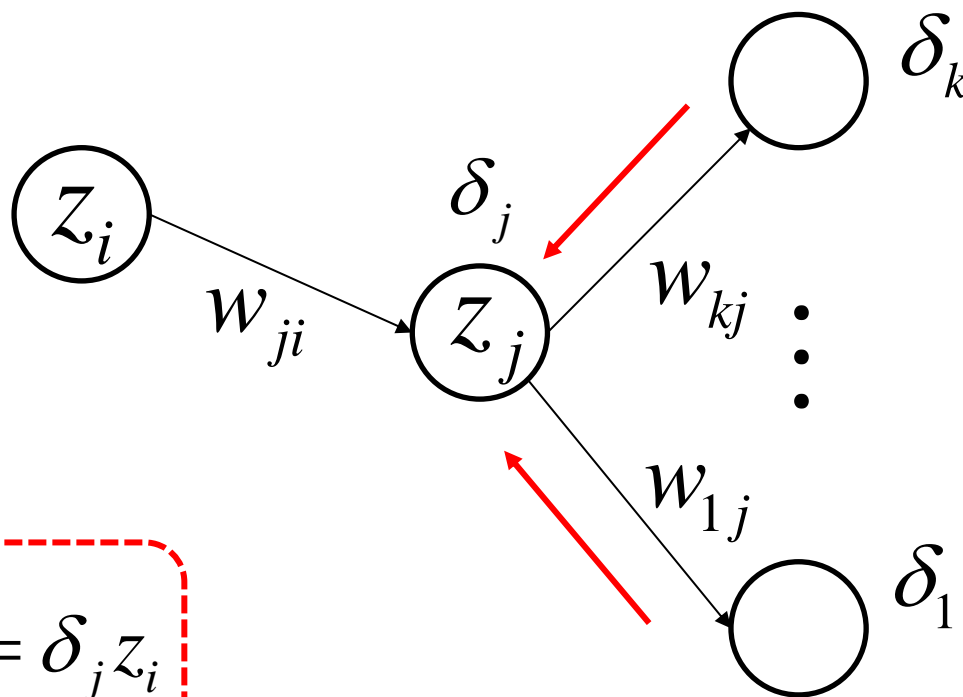
- errors $\delta_j := \frac{\partial E_n}{\partial a_j}$

Do $\frac{\partial a_j}{\partial w_{ji}} = z_i$ ta có $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$

$$\delta_j := \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

Lan truyền ngược

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$



Thuật toán

- Đầu vào, đầu ra

Input: Input vector \mathbf{x}_n

Network parameters \mathbf{w}

Error function $E_n(\mathbf{w})$ for input x_n

Activation function $h(a)$

Output: Error function derivatives $\{\partial E_n / \partial w_{ji}\}$

Thuật toán

```
// Forward propagation
for  $j \in$  all hidden and output units do
     $a_j \leftarrow \sum_i w_{ji} z_i$  //  $\{z_i\}$  includes inputs  $\{x_i\}$ 
     $z_j \leftarrow h(a_j)$  // activation function
end for

// Error evaluation
for  $k \in$  all output units do
     $\delta_k \leftarrow \frac{\partial E_n}{\partial a_k}$  // compute errors
end for

// Backward propagation, in reverse order
for  $j \in$  all hidden units do
     $\delta_j \leftarrow h'(a_j) \sum_k w_{kj} \delta_k$  // recursive backward evaluation
     $\frac{\partial E_n}{\partial w_{ji}} \leftarrow \delta_j z_i$  // evaluate derivatives
end for

return  $\left\{ \frac{\partial E_n}{\partial w_{ji}} \right\}$ 
```

Giải thuật lan truyền ngược

- Sử dụng trong việc huấn luyện mạng NN để tìm ra các thông số của mạng giúp tối thiểu hóa lỗi tổng thể của mạng trên tập huấn luyện
 - Lan truyền tiến: các mẫu huấn luyện đầu vào được “lan truyền” qua các tầng đến tầng đầu ra
 - Lan truyền ngược lỗi:
 - Tính toán giá trị lỗi dựa trên giá trị đầu ra mong đợi
 - Từ tầng đầu ra, giá trị lỗi được lan truyền ngược qua mạng NN (từ tầng đầu, qua các tầng ẩn, đến tầng đầu vào) => thông qua việc tính toán giá trị gradient cục bộ của mỗi neuron

Automatic Differentiation (AD)

- Kỹ thuật tính toán được sử dụng để tính toán hiệu quả và chính xác các đạo hàm của hàm số
- Có hai cách tiếp cận
 - Forward-Mode AD
 - Phù hợp cho các hàm có ít đầu vào và nhiều đầu ra
 - Reverse-Mode AD
 - Phù hợp cho các hàm có nhiều đầu vào và ít đầu ra

Automatic Differentiation (AD)

$$f(x_1, x_2) = x_1 x_2 + \exp(x_1 x_2) - \sin(x_2)$$



$$\frac{\partial f}{\partial x_1} = ?$$

Forward-Mode Automatic Differentiation

$$f(x_1, x_2) = x_1 x_2 + \exp(x_1 x_2) - \sin(x_2)$$

$$\frac{\partial f}{\partial x_1} = ?$$

$$v_1 = x_1$$

$$v_2 = x_2$$

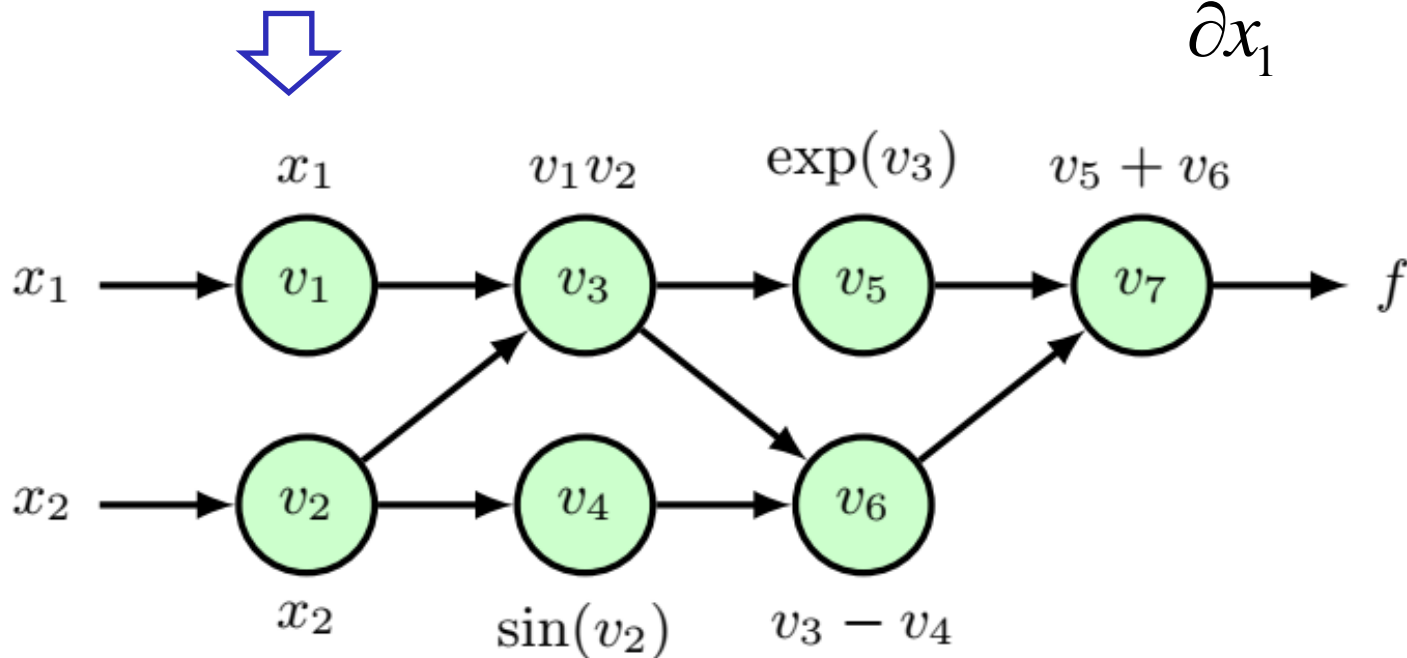
$$v_3 = v_1 v_2$$

$$v_4 = \sin(v_2)$$

$$v_5 = \exp(v_3)$$

$$v_6 = v_3 - v_4$$

$$v_7 = v_5 + v_6$$



$$\dot{v}_i = \frac{\partial v_i}{\partial x_1} = \sum_{j \in \text{pa}(i)} \frac{\partial v_j}{\partial x_1} \frac{\partial v_i}{\partial v_j} = \sum_{j \in \text{pa}(i)} \dot{v}_j \frac{\partial v_i}{\partial v_j}$$

$\text{pa}(i)$: tập các nút cha của nút i

Automatic Differentiation (AD)

$$f(x_1, x_2) = x_1 x_2 + \exp(x_1 x_2) - \sin(x_2)$$

$$v_1 = x_1 \quad \dot{v}_1 = 1$$

$$v_2 = x_2 \quad \dot{v}_2 = 0$$

$$v_3 = v_1 v_2 \quad \dot{v}_3 = v_1 \dot{v}_2 + \dot{v}_1 v_2$$

$$v_4 = \sin(v_2) \quad \dot{v}_4 = \dot{v}_2 \cos(v_2)$$

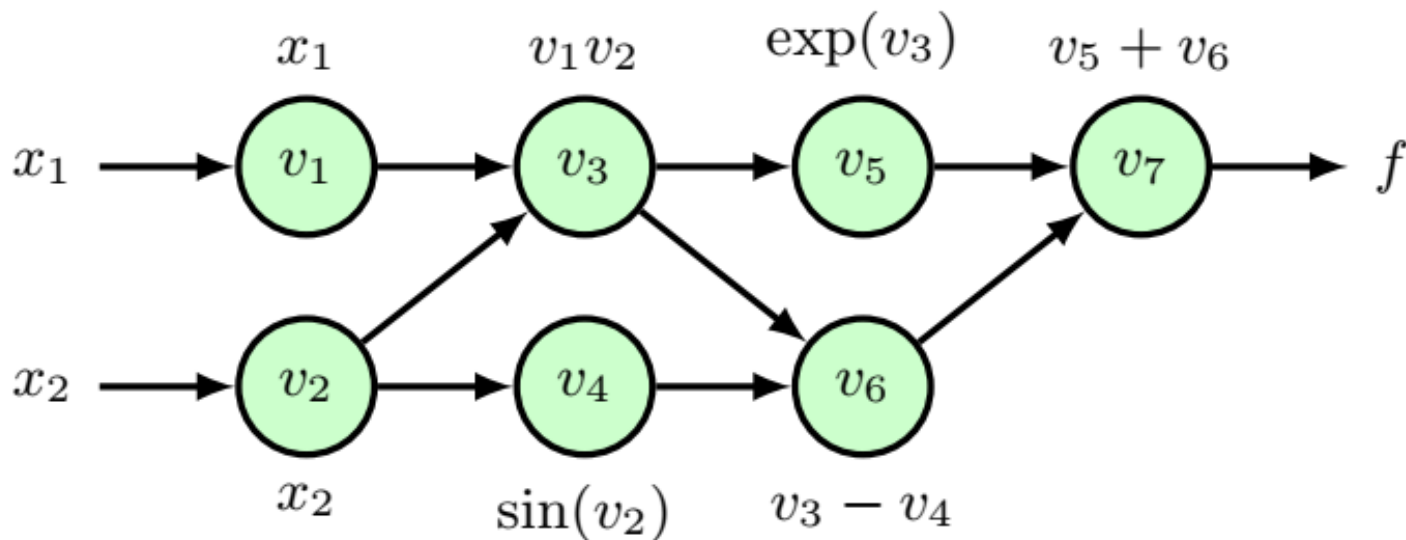
$$v_5 = \exp(v_3) \quad \dot{v}_5 = \dot{v}_3 \exp(v_3)$$

$$v_6 = v_3 - v_4 \quad \dot{v}_6 = \dot{v}_3 - \dot{v}_4$$

$$v_7 = v_5 + v_6 \quad \dot{v}_7 = \dot{v}_5 + \dot{v}_6$$

$$\frac{\partial f}{\partial x_1} = ?$$

Reverse-Mode Automatic Differentiation



$$\bar{v}_i = \frac{\partial f}{\partial v_i} = \sum_{j \in \text{ch}(i)} \frac{\partial f}{\partial v_j} \frac{\partial v_j}{\partial v_i} = \sum_{j \in \text{ch}(i)} \bar{v}_j \frac{\partial v_j}{\partial v_i}$$

$\text{ch}(i)$: tập các nút con của node i

Reverse-Mode Automatic Differentiation

$$f(x_1, x_2) = x_1 x_2 + \exp(x_1 x_2) - \sin(x_2)$$

$$v_1 = x_1 \quad \bar{v}_7 = 1$$

$$v_2 = x_2 \quad \bar{v}_6 = \bar{v}_7$$

$$v_3 = v_1 v_2 \quad \bar{v}_5 = \bar{v}_7$$

$$v_4 = \sin(v_2) \quad \bar{v}_4 = -\bar{v}_6$$

$$v_5 = \exp(v_3) \quad \bar{v}_3 = \bar{v}_5 v_5 + \bar{v}_6$$

$$v_6 = v_3 - v_4 \quad \bar{v}_2 = \bar{v}_2 v_1 + \bar{v}_4 \cos(v_2)$$

$$v_7 = v_5 + v_6 \quad \bar{v}_1 = \bar{v}_3 v_2$$

$$\frac{\partial f}{\partial x_1} = ?$$

Huấn luyện các Deep Neural Network

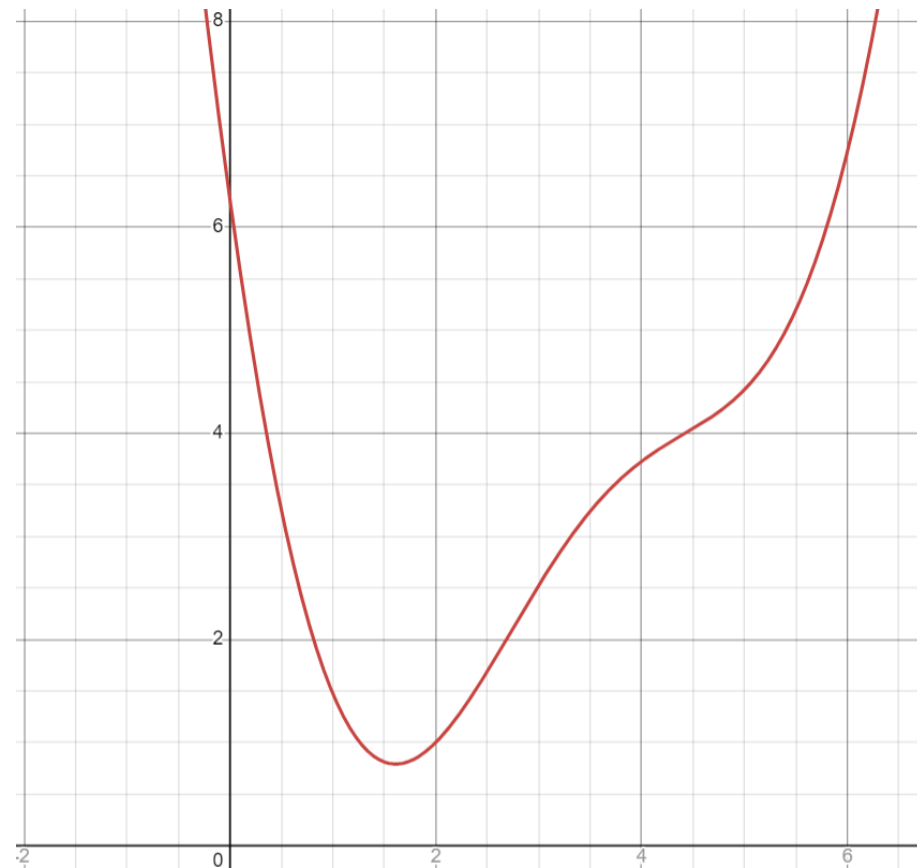
- Sinh viên tham khảo thêm các vấn đề quan tâm liên quan tới huấn luyện DNN, ví dụ như:
 - Vanishing/exploding gradients problems
 - Reusing pretrained layers
 - Faster optimization: momentum, Nesterov accelerated gradient, AdaGrad, RMSProp, Adam, AdaMax, Nadam, AdamW ...
 - Avoiding overfitting through regularization: dropout, max-norm regularization...

Ví dụ

- Ví dụ: Thiết kế mạng neuron để có thể xấp xỉ một hàm phi tuyến

$$f(x) = (x-3)^2 + 3\sin(x-2)$$

$$0 \leq x \leq 6$$



Ví dụ

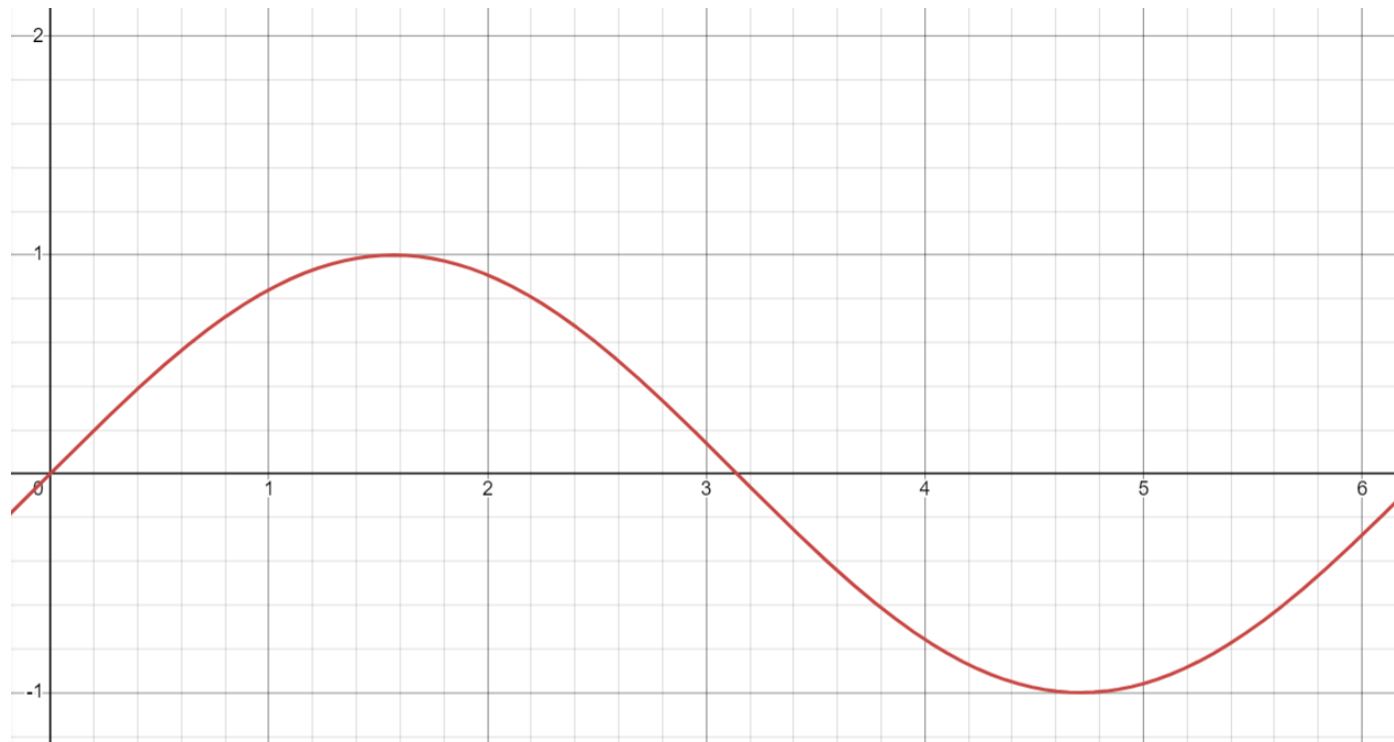
- Ví dụ: Thiết kế mạng neuron để có thể xấp xỉ một hàm phi tuyến

$$f(x) = \sin(x)$$



$$0 \leq x \leq 6$$

$$0 \leq x \leq 1$$



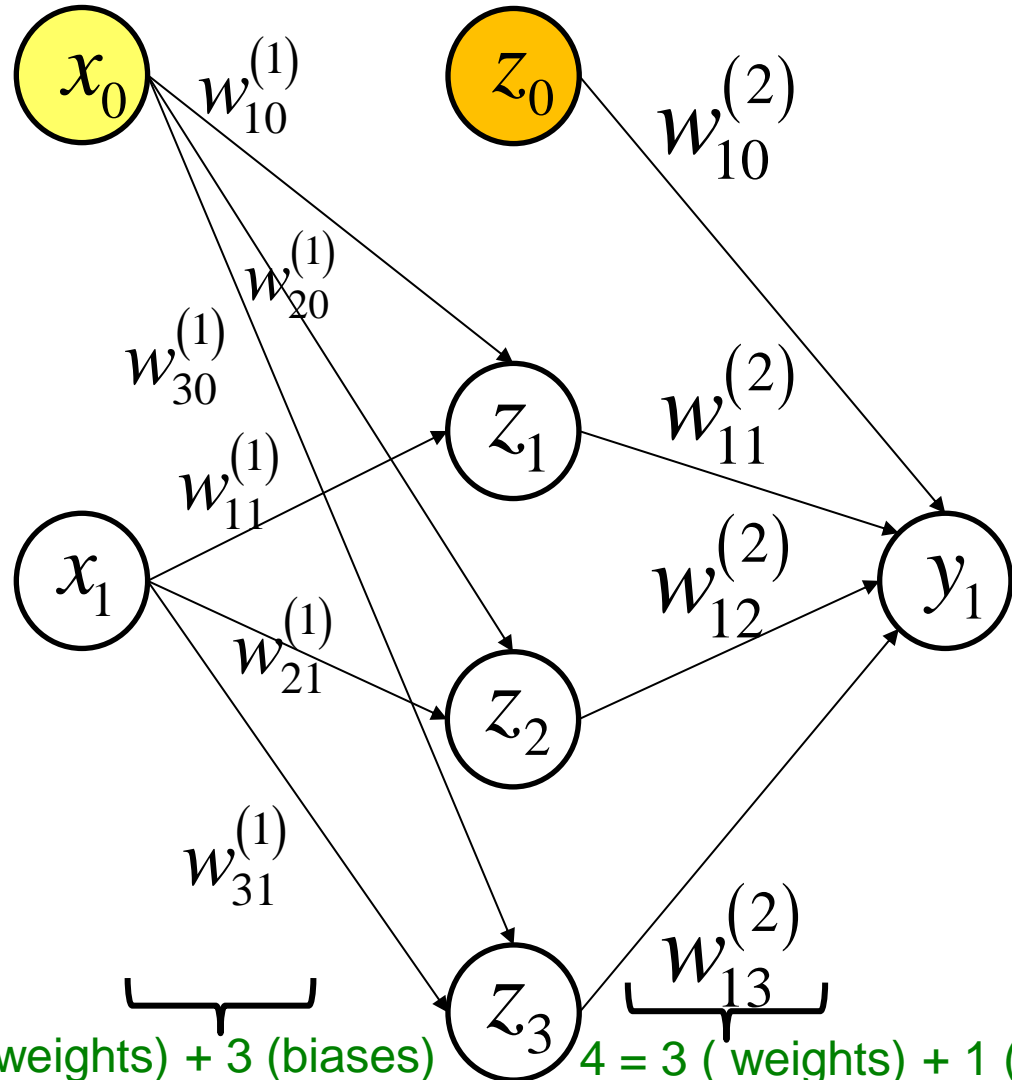
Ví dụ

Thiết kế một mạng đơn giản

$$D = 1$$

$$M = 3$$

$$K = 1$$



$$x_0 = 1$$

$$z_0 = 1$$

$$6 = 3 \text{ (weights)} + 3 \text{ (biases)}$$

$$4 = 3 \text{ (weights)} + 1 \text{ (bias)}$$

Ví dụ

Lớp 1:

$a_j^{(1)}$: pre-activation

$$a_j^{(1)} = w_{j0}^{(1)} + \sum_{i=1}^{D=1} w_{ji}^{(1)} x_i$$

$$z_j^{(1)} = h(a_j^{(1)})$$

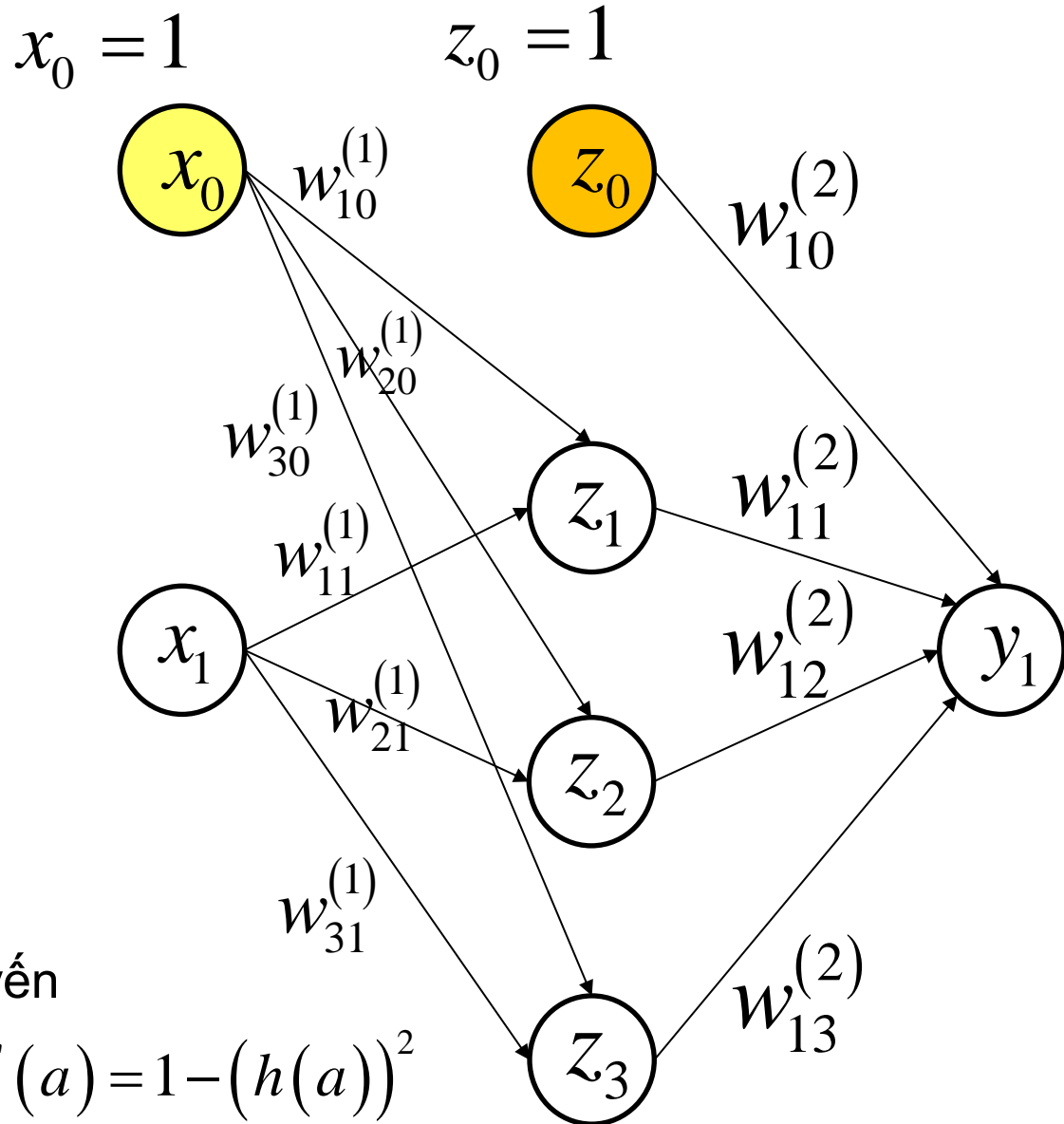
$$j = 1, \dots, M = 3$$

$w_{ji}^{(1)}$: trọng số

$w_{j0}^{(1)}$: bias

$h(\)$: hàm kích hoạt phi tuyến

$$h(a) = \tanh(a) \text{ do đó } h'(a) = 1 - (h(a))^2$$



Ví dụ

Lớp 2:

$$x_0 = 1$$

$$z_0 = 1$$

$a_k^{(2)}$: pre-activation

$$a_k^{(2)} = w_{k0}^{(2)} + \sum_{j=1}^{M=3} w_{kj}^{(2)} z_j^{(1)}$$

$$y_k^{(2)} = f(a_k^{(2)})$$

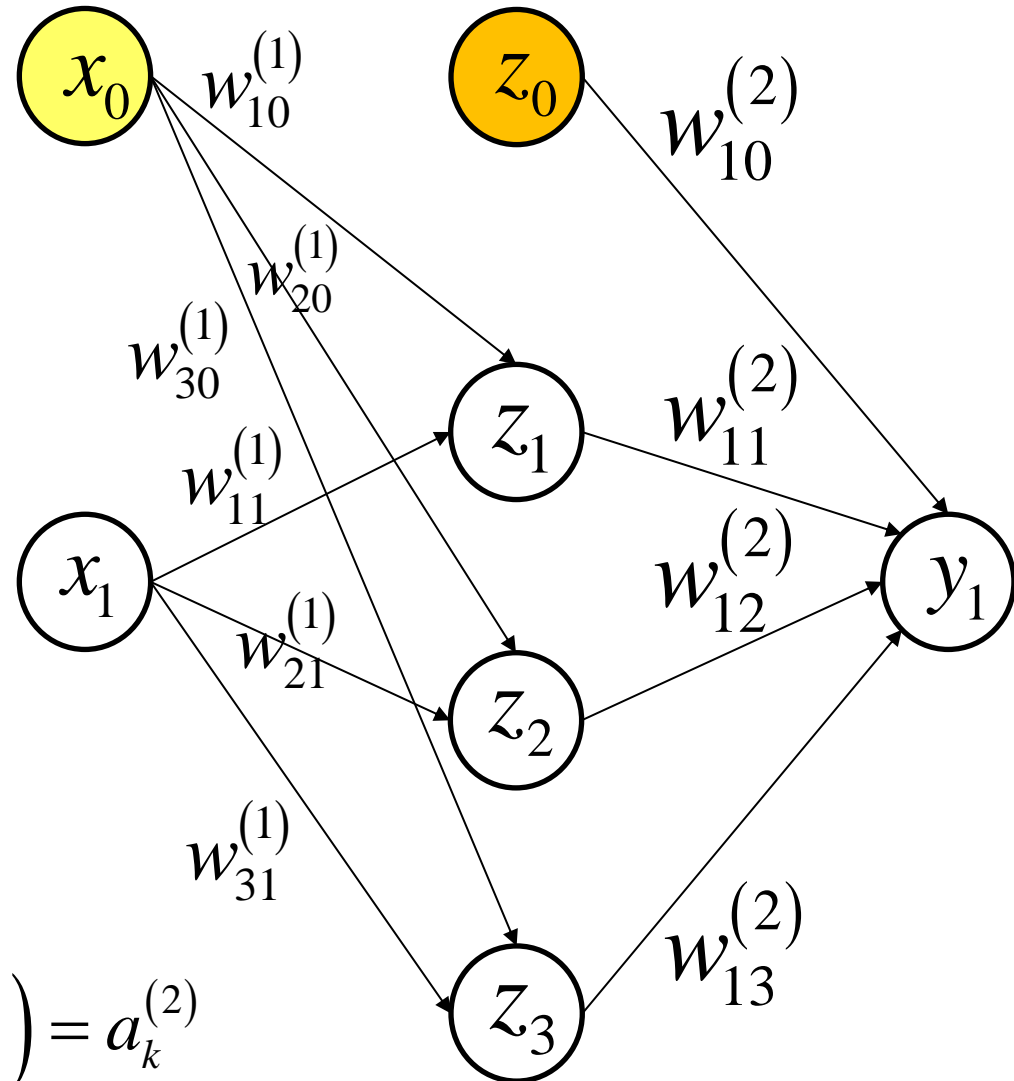
$$k = 1, \dots, K = 1$$

$w_{kj}^{(2)}$: trọng số

$w_{k0}^{(2)}$: bias

$f(\cdot)$: hàm kích hoạt tuyến tính

$$f(a) = a \text{ do đó } y_k^{(2)} = f(a_k^{(2)}) = a_k^{(2)}$$



Ví dụ

Đối với 1 điểm dữ liệu $n \in \{(\mathbf{x}, \mathbf{t})\}$
lỗi được tính bởi

\mathbf{t} : nhãn (target)

$$E_n = \frac{1}{2} \sum_{k=1}^{K=1} \left(y_k^{(2)} - t_k \right)^2 = \frac{1}{2} \left(y_1^{(2)} - t_1 \right)^2$$

Ví dụ

- B1: Khởi tạo giá trị các trọng số và bias

$$x_0 = 1$$

$$6 = 3 \text{ (weights)} + 3 \text{ (biases)}$$

$$w_{11}^{(1)}$$

$$w_{10}^{(1)}$$

$$w_{21}^{(1)}$$

$$w_{20}^{(1)}$$

$$w_{31}^{(1)}$$

$$w_{30}^{(1)}$$

$$z_0 = 1$$

$$4 = 3 \text{ (weights)} + 1 \text{ (bias)}$$

$$w_{11}^{(2)}$$

$$w_{10}^{(2)}$$

$$w_{12}^{(2)}$$

$$w_{13}^{(2)}$$

Ví dụ

- B2: Lan truyền tiến

$$a_j^{(1)} = w_{j0}^{(1)} + \sum_{i=1}^{D=1} w_{ji}^{(1)} x_i = \sum_{i=0}^{D=1} w_{ji}^{(1)} x_i \quad x_0 = 1$$

$$z_j^{(1)} = \tanh(a_j^{(1)}) \quad z_0^{(1)} = 1$$

$$j = 1, \dots, M = 3$$

Đầu ra: $k = 1$

$$y_k^{(2)} = w_{k0}^{(2)} + \sum_{j=1}^{M=3} w_{kj}^{(2)} z_j^{(1)} = \sum_{j=0}^{M=3} w_{kj}^{(2)} z_j^{(1)}$$

Ví dụ

- B3: Tính lỗi ở phía đầu ra

$$\delta_k^{(2)} = y_k^{(2)} - t_k \quad k = 1$$

Vậy ta chỉ có: $\delta_1^{(2)} = y_1^{(2)} - t_1$

Ví dụ

- B4: Lan truyền ngược

$$\delta_j^{(1)} = h' \left(a_j^{(1)} \right) \sum_k w_{kj}^{(2)} \delta_k^{(2)} \quad \begin{matrix} j = 1, \dots, M = 3 \\ k = 1 \end{matrix}$$

hay

$$\delta_j^{(1)} = \left(1 - \left(z_j^{(1)} \right)^2 \right) w_{1j}^{(2)} \delta_1^{(2)} \quad j = 1, \dots, 3$$

Do đó tính được các đạo hàm

6 = 3 (weights) + 3 (biases)

$$\frac{\partial E_n}{\partial w_{ji}^{(1)}} = \delta_j^{(1)} x_i$$

$$j = 1, \dots, 3 \quad i = 0, 1$$

4 = 3 (weights) + 1 (bias)

$$\frac{\partial E_n}{\partial w_{kj}^{(2)}} = \delta_k^{(2)} z_j^{(1)}$$

$$j = 0, \dots, 3 \quad k = 1$$

Ví dụ

- B5: Cập nhật các thông số của mạng (trọng số, bias)

$$6 = 3 \text{ (weights)} + 3 \text{ (biases)}$$

$$4 = 3 \text{ (weights)} + 1 \text{ (bias)}$$

$$w_{ji}^{(1)} \leftarrow w_{ji}^{(1)} - \eta \frac{\partial E_n}{\partial w_{ji}^{(1)}}$$

$$j = 1, \dots, 3 \quad i = 0, 1$$

$$w_{kj}^{(2)} \leftarrow w_{kj}^{(2)} - \eta \frac{\partial E_n}{\partial w_{kj}^{(2)}}$$

$$j = 0, \dots, 3 \quad k = 1$$

Ví dụ

- Có thể dễ dàng xây dựng và huấn luyện mạng neuron với Keras và TensorFlow

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 3)	6
dense_1 (Dense)	(None, 1)	4

$$6 = 3 \text{ (weights)} + 3 \text{ (biases)}$$

$$4 = 3 \text{ (weights)} + 1 \text{ (bias)}$$

Layer 0:

Weights: $\begin{bmatrix} -0.77810127 & -1.4763303 & 1.2871214 \end{bmatrix}$

Biases: $\begin{bmatrix} 0.40375996 & 4.596628 & -0.6489421 \end{bmatrix}$

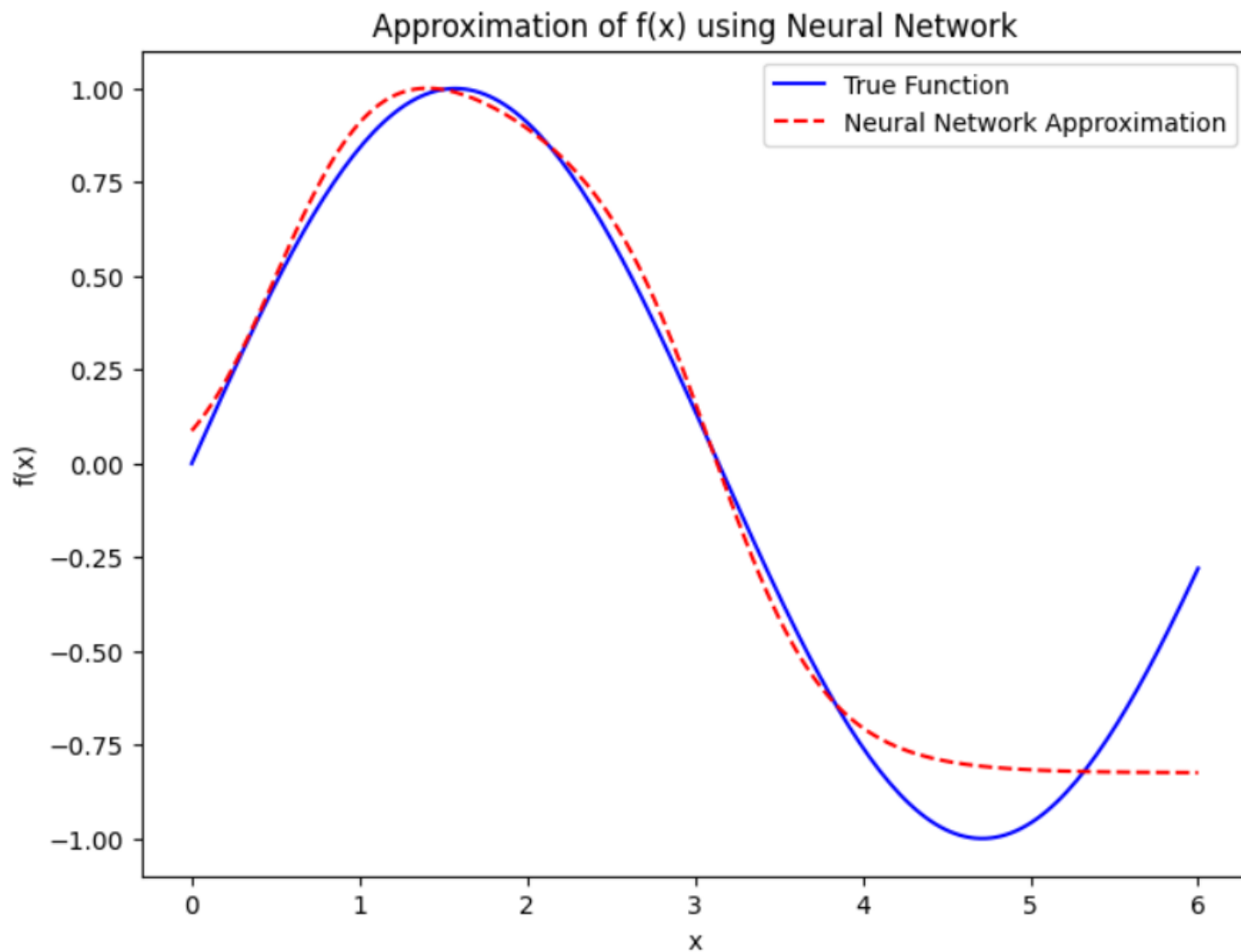
Layer 1:

Weights: $\begin{bmatrix} 1.2153453 \\ 0.8098653 \\ 1.520071 \end{bmatrix}$

Biases: $\begin{bmatrix} -0.3201559 \end{bmatrix}$

Ví dụ

- Nhận xét kết quả



4.4 Các mạng neuron nhân tạo và ứng dụng

- Sinh viên lựa chọn một lĩnh vực quan tâm, tìm hiểu, tổng hợp, trình bày và thảo luận
 - Deep computer vision using Convolutional Neural Networks (CNNs)
 - Processing sequences using Recurrent Neural Networks (RNNs)
 - Natural Language Processing (NLP) with RNNs and attention
 - Generative Adversarial Networks (GANs)
 - Autoencoders
 - Large Language Model (LLM)
 - Diffusion Models
 - ...

4.4 Các mạng neuron nhân tạo và ứng dụng

- Sinh viên lựa chọn một ứng dụng cụ thể cần quan tâm, tìm hiểu, tổng hợp, trình bày và thảo luận
 - IoTs
 - Image Generation and Editing
 - Text-to-Image Synthesis
 - Data Augmentation
 - Anomaly Detection
 - Video Generation and Animation
 - Speech Synthesis
 - Drug Discovery and Molecular Design
 - ...

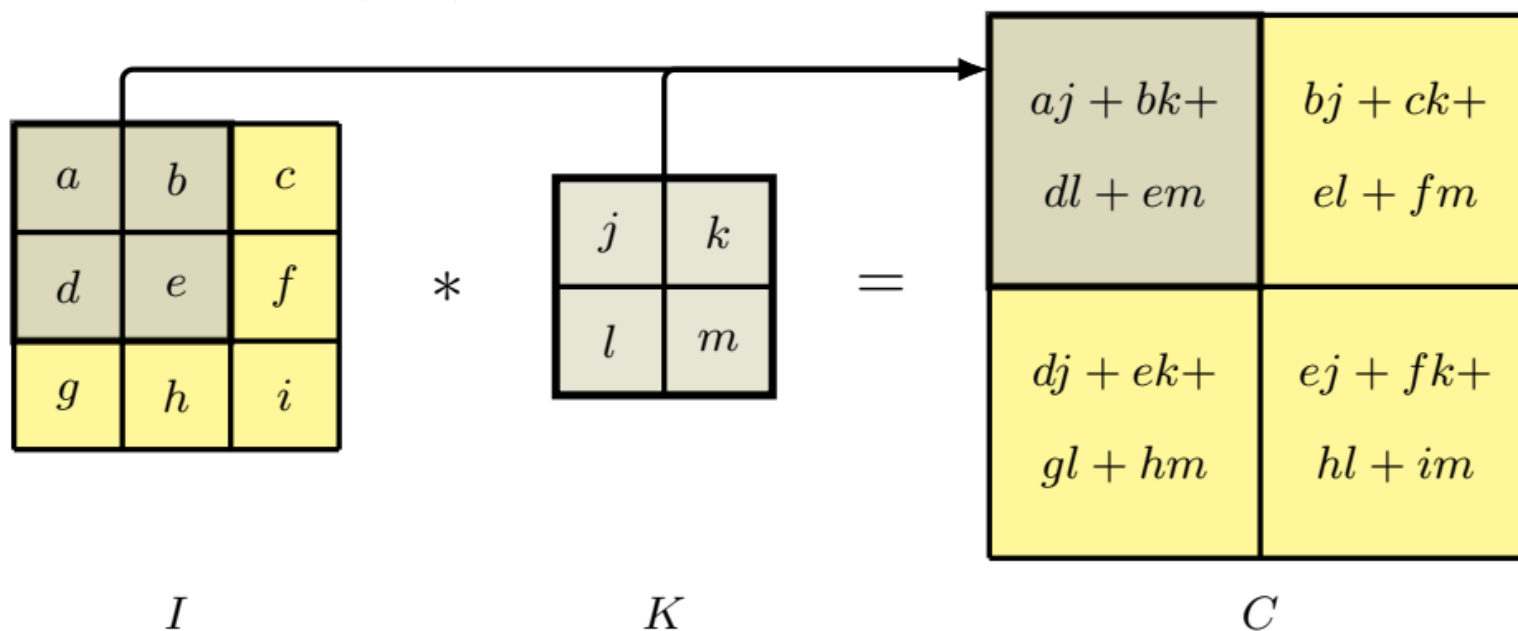
Ví dụ: Convolutional neural network

- Computer vision
 - Convolution

$$C(j, k) = \sum_l \sum_m I(j+l, k+m) K(l, m)$$

I : Ảnh $I(j, k)$: cường độ điểm ảnh

K : Bộ lọc $K(l, m)$: giá trị điểm ảnh



Ví dụ: Convolutional neural network

- Computer vision
 - Phát hiện cạnh bằng bộ lọc tích chập



Phát hiện các cạnh dọc

-1	0	1
-1	0	1
-1	0	1

Bộ lọc 3x3



Phát hiện các cạnh ngang

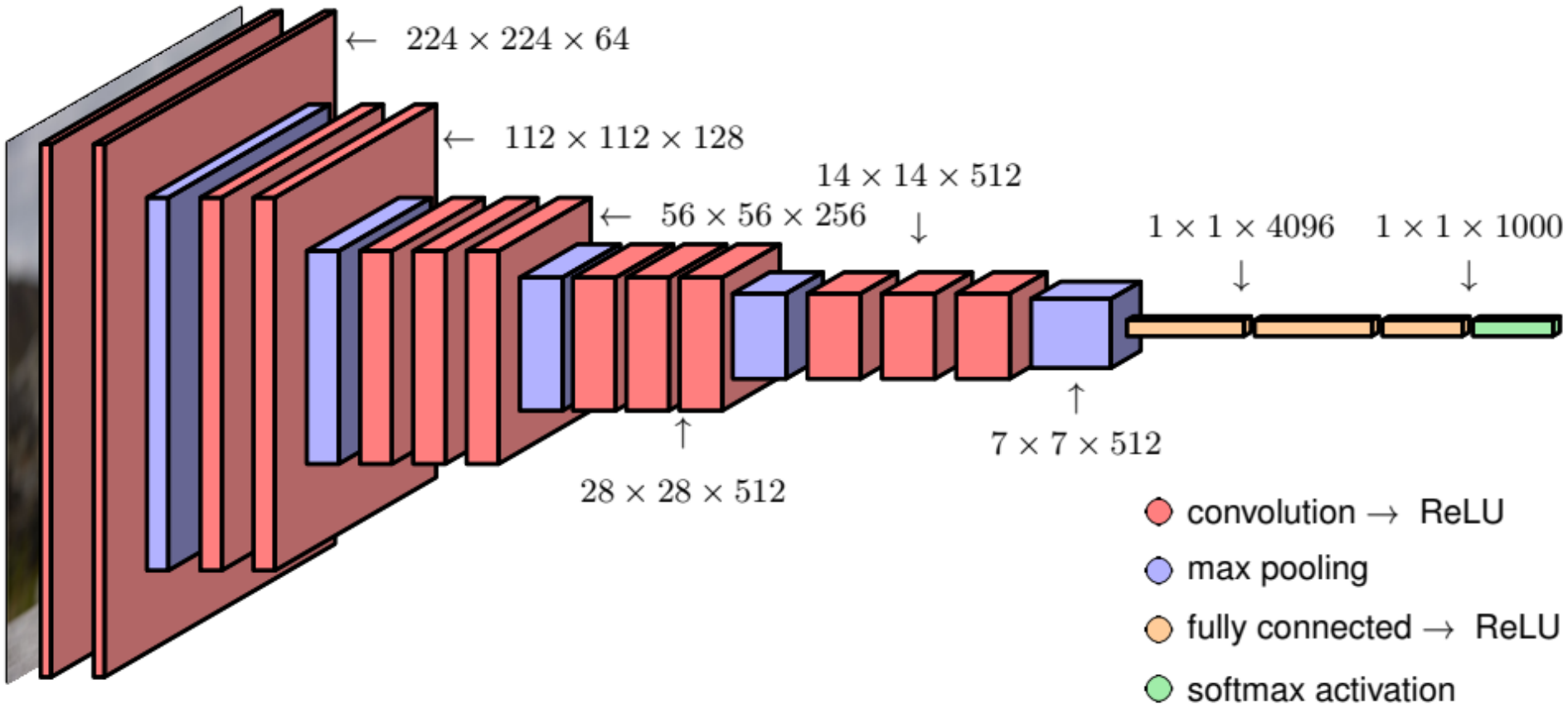
-1	-1	-1
0	0	0
1	1	1

Ví dụ: Convolutional neural network

- Kiến trúc của mô hình VGG-16

input image

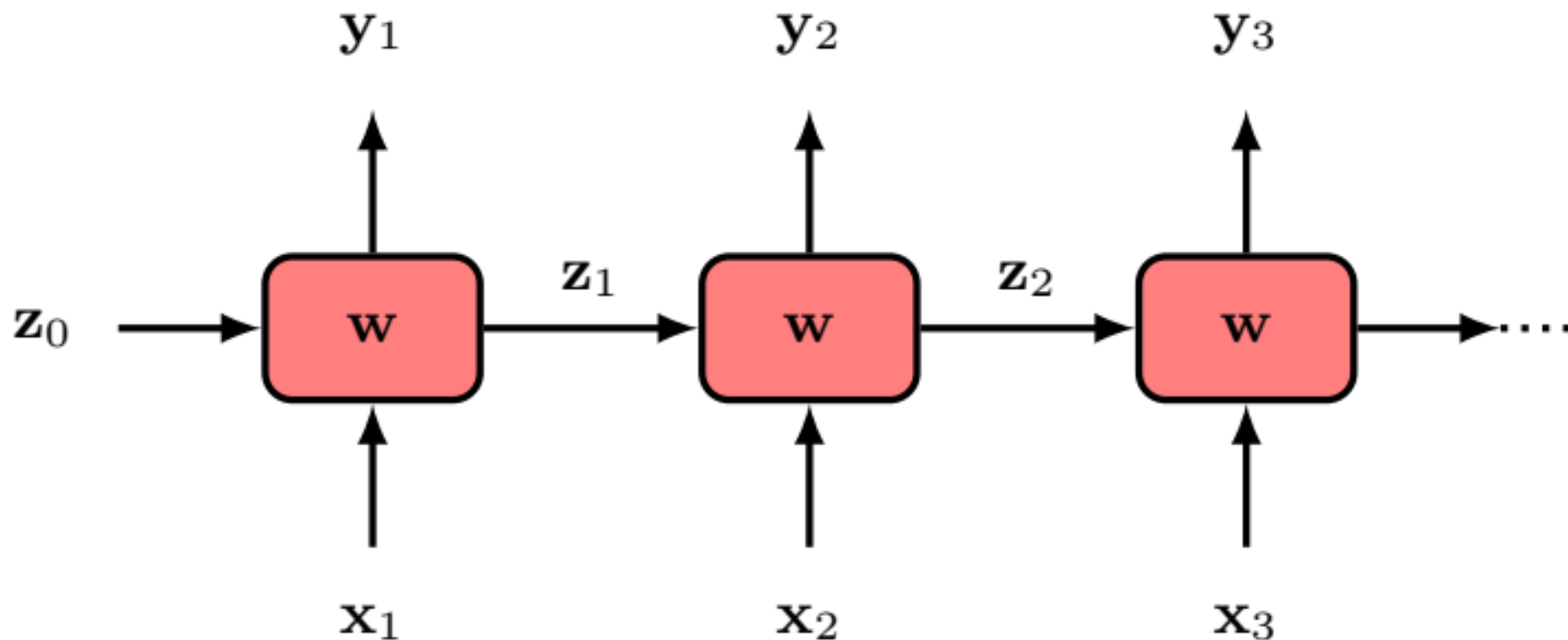
$224 \times 224 \times 3$



<https://www.bishopbook.com/>

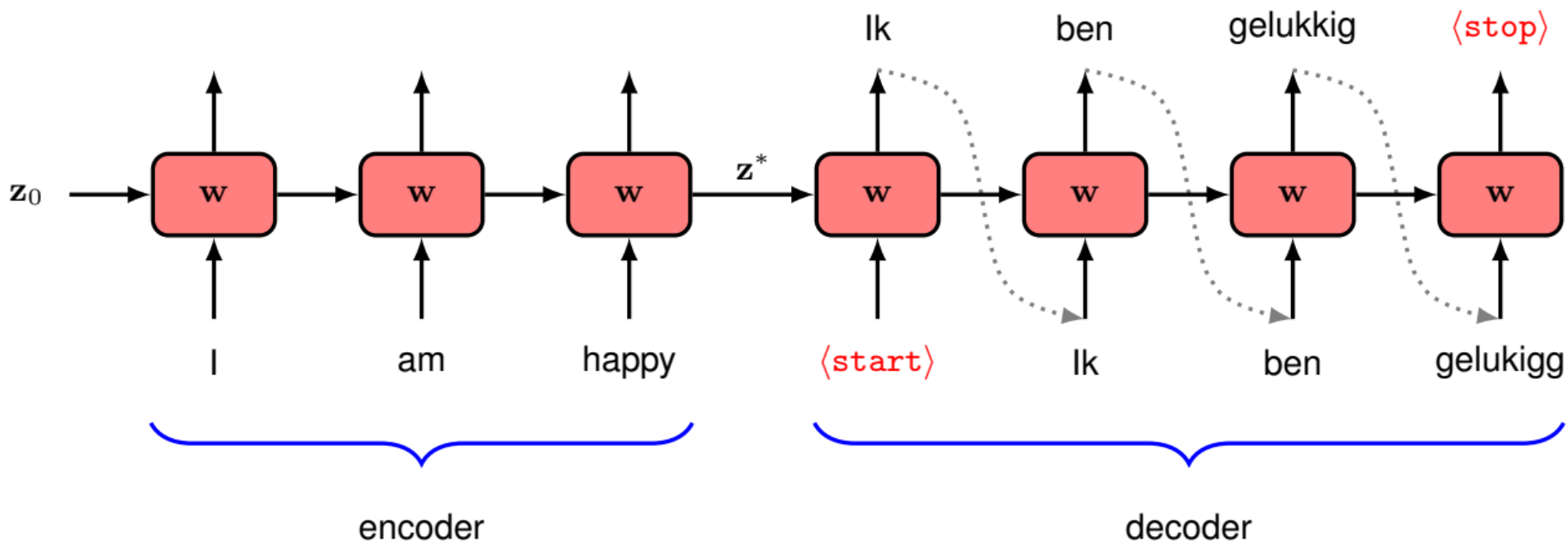
Ví dụ: Recurrent neural network

- RNN



Ví dụ: Recurrent neural network

- RNN để dịch từ ngôn ngữ tiếng Anh sang tiếng Hà Lan





18+

CÁM

The Sisters · 2024 · 122 phút

★ 6 ^{2.9K}
đánh giá

Bống Bống Bang Bang, lên ăn máu thịt, máu thịt nhà ta.

Nội dung

Phim điện ảnh CáM - Dị bản kinh dị đẫm máu chuyển thể từ cổ tích Tấm Cám

Ngày chiếu	Thể loại	Quốc gia
20/09/2024	Kinh dị	Việt Nam

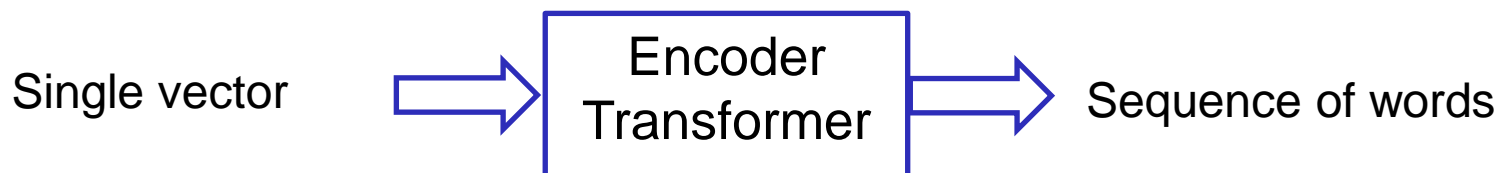
<https://www.momo.vn/cinema/the-sisters-23868>



<https://www.google.com/url?sa=i&url=https%3A%2F%2Fcellphones.com.vn%2Fforum%2Fphim-cam&psig=AOvVaw1bk0PJppYSIJVqFJiTY3HV&ust=1727236669317000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCOjxLvY2ogDFQAAAAAdAAAAABAg>

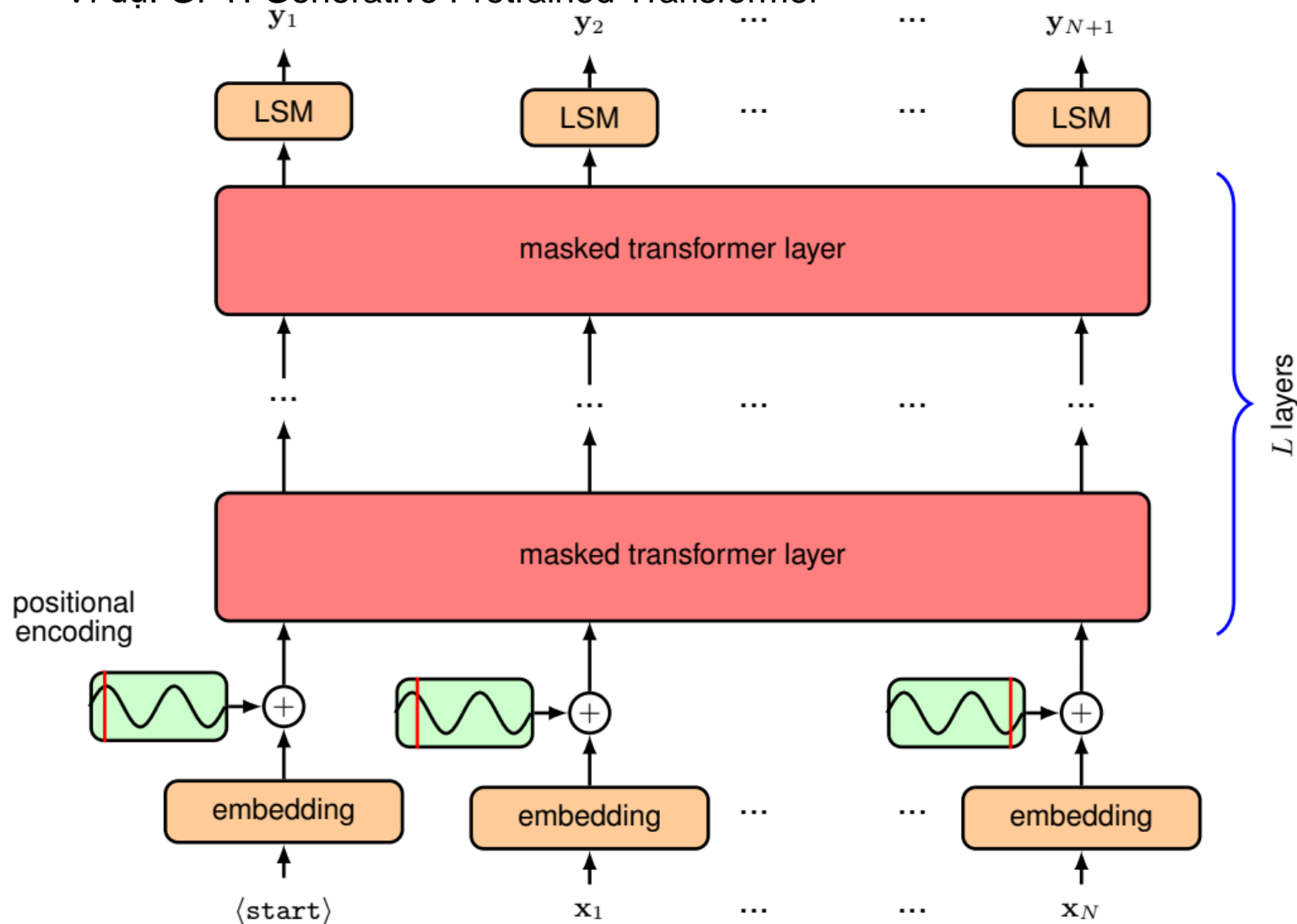
Ví dụ: Transformer Language Models

- Massive neural networks được biết tới như Large Language Models (LLMs)



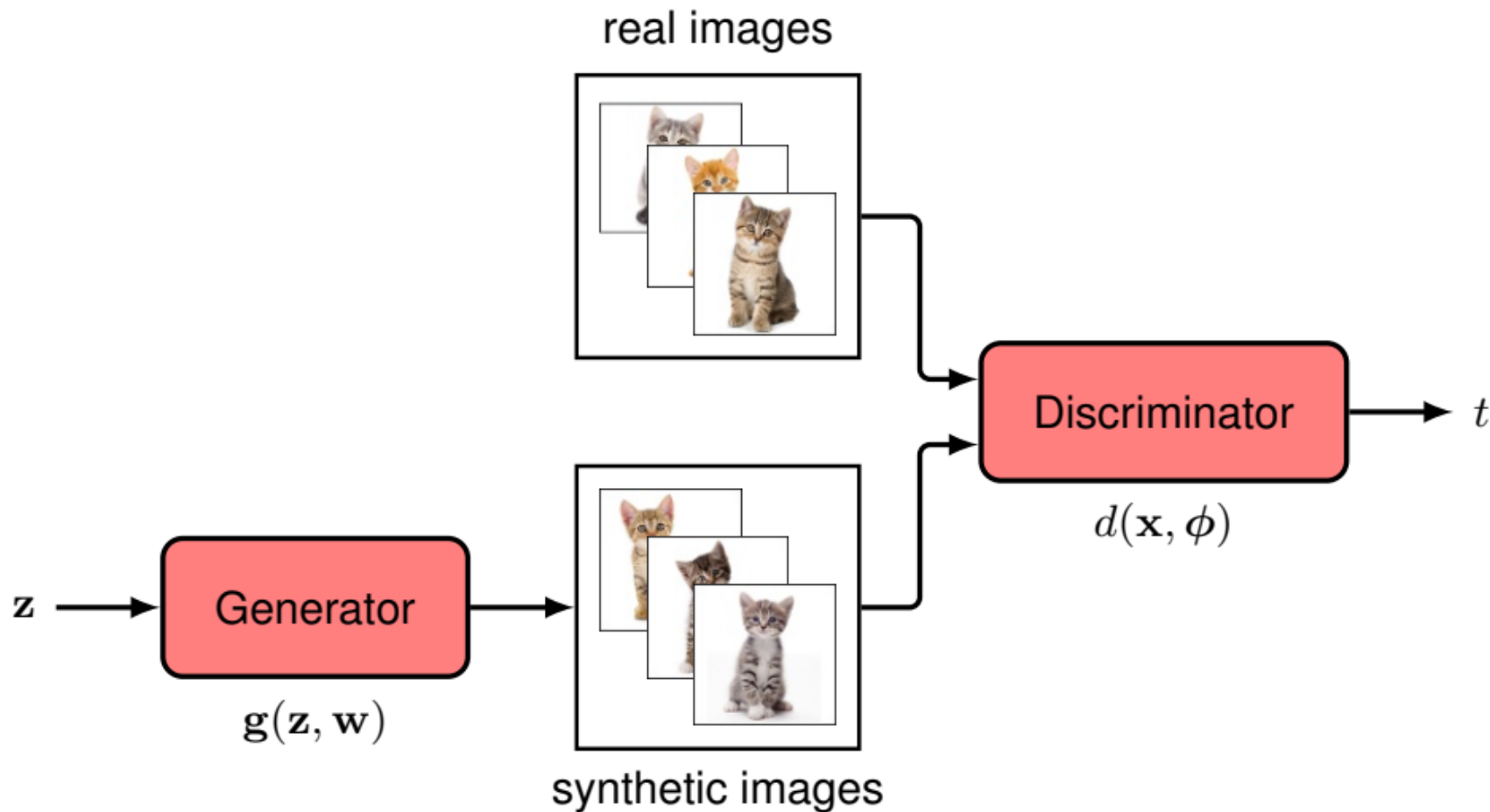
Ví dụ: Decoder Transformer

- Được dùng như các “generative model”
 - Ví dụ: GPT: Generative Pretrained Transformer



Ví dụ: Generative Adversarial Networks

- GAN



Ví dụ

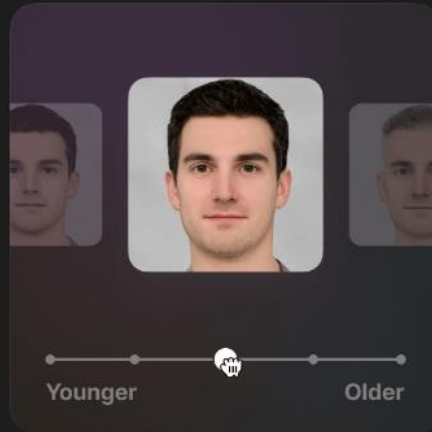
Unique, worry-free model photos

Enhance your creative works with photos generated completely by AI. Search our gallery of high-quality diverse photos or create unique models by your parameters in real time



Faces

2,676,245 pre-generated diverse faces



Face Generator

Create unique faces by your parameters



Humans

100,000 super realistic full-body images




Human Generator New

Create unique full-body humans by your parameters

<https://generated.photos/>

Ví dụ

 **HUMAN GENERATOR** New ▾


Description

Pose

Face upload

Hot

Description

 Randomize

reset

Enter prompt...

adult ▾ | ✕

female ▾ | ✕

vietnamese ▾ | ✕

shirt ▾ | ✕

shorts ▾ | ✕

sneakers ▾ | ✕

Negative prompt ⓘ

Enter negative prompt...

Gender

Female

clear

☒ Female

☐ Male

☐ Non-binary

Age

Adult

clear

Skin tone

Not set

▾

Generate variation

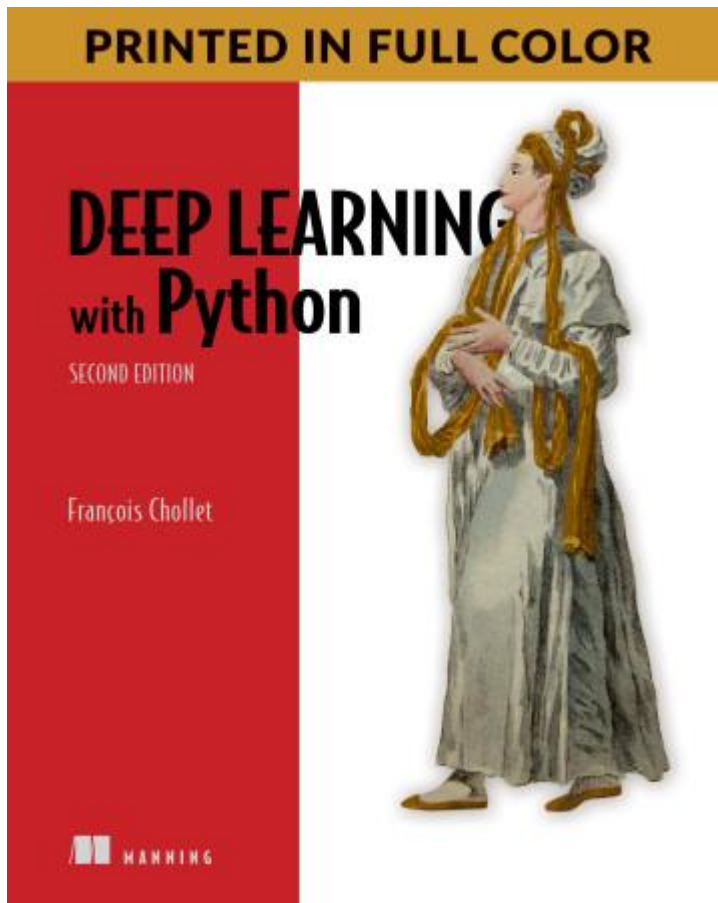
↵ Enter



<https://generated.photos/human-generator/66e25623098b8100117e41e9>

Keras & TensorFlow

- Keras: TensorFlow's high-level deep learning API
 - Xây dựng, huấn luyện, đánh giá các kiểu NNs



<https://www.manning.com/books/deep-learning-with-python>

Keras & TensorFlow

- Ví dụ: https://github.com/ageron/handson-ml3/blob/main/10_neural_nets_with_keras.ipynb

```
tf.keras.backend.clear_session()
tf.random.set_seed(42)

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=[28, 28]),
    tf.keras.layers.Dense(300, activation="relu"),
    tf.keras.layers.Dense(100, activation="relu"),
    tf.keras.layers.Dense(10, activation="softmax")
])
```

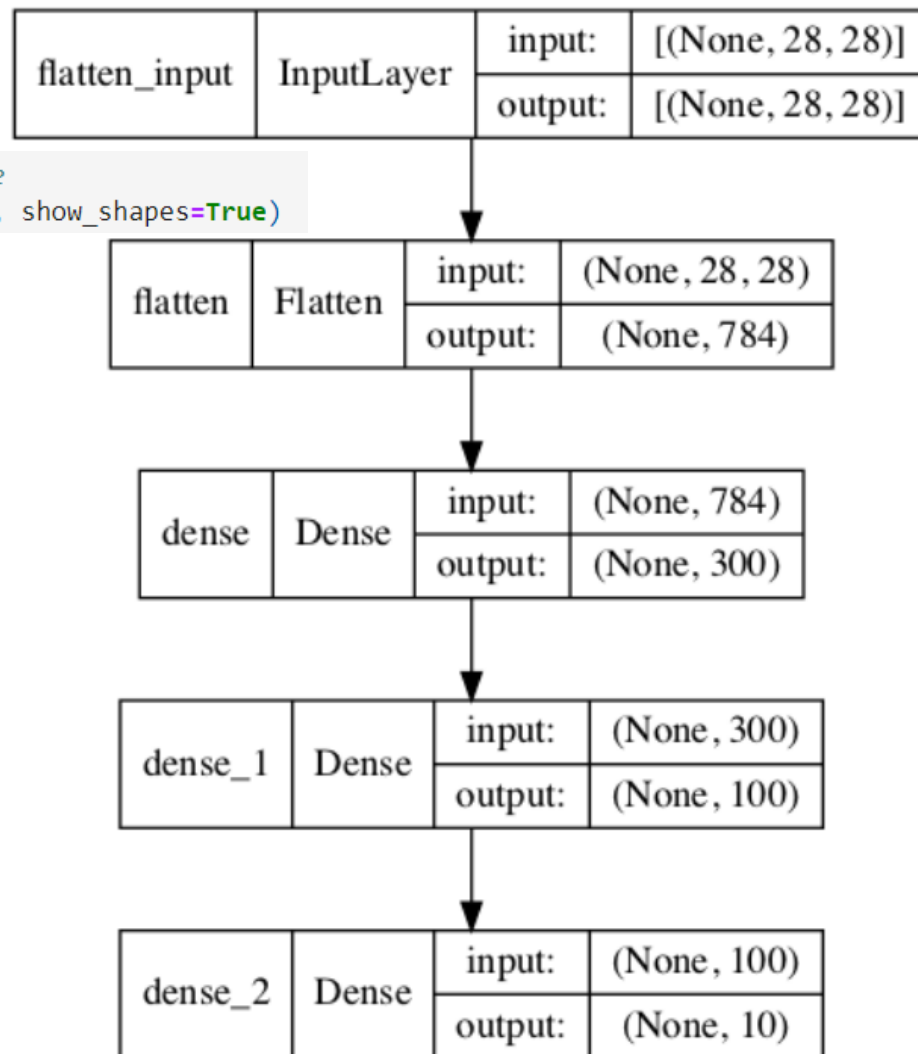
```
model.summary()
```

Layer (type)	Output Shape	Param #
=====		
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 300)	235500
dense_1 (Dense)	(None, 100)	30100
dense_2 (Dense)	(None, 10)	1010
=====		
Total params: 266,610		
Trainable params: 266,610		
Non-trainable params: 0		

Keras & TensorFlow

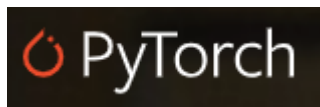
- Ví dụ: https://github.com/ageron/handson-ml3/blob/main/10_neural_nets_with_keras.ipynb

```
# extra code - another way to display the model's architecture  
tf.keras.utils.plot_model(model, "my_fashion_mnist_model.png", show_shapes=True)
```



Pytorch

- PyTorch: an open-source deep learning framework
 - Facebook's AI Research (FAIR) lab
 - Được sử dụng rộng rãi để xây dựng và huấn luyện mạng neuron và thực hiện nhiều tác vụ khác nhau trong học máy và học sâu
 - dynamic computation graph



PyTorch Edge

The AI landscape is quickly evolving, with AI models being deployed beyond server to edge devices such as mobile phones, wearables, AR/VR/MR and embedded devices. PyTorch Edge extends PyTorch's research-to-production stack to these edge devices and paves the way for building innovative, privacy-aware experiences with superior productivity, portability, and performance, optimized for these diverse hardware platforms.

Introducing ExecuTorch

To advance our PyTorch Edge offering, we developed **ExecuTorch**, our new runtime for edge devices. ExecuTorch facilitates PyTorch inference on edge devices while supporting portability across hardware platforms with lower runtime and framework tax. ExecuTorch was developed collaboratively between industry leaders including Meta, Arm, Apple, and Qualcomm.

With ExecuTorch, we've renewed our commitment to on-device AI. This extends our ecosystem in a much more "in the spirit of PyTorch" way, with productivity, hackability, and extensibility as critical components. We look forward to supporting edge and embedded applications with low latency, strong privacy, and innovation on the edge.

<https://pytorch.org/>

Hoạt động sau buổi học

- Tìm hiểu thêm về các artificial neural network và ứng dụng
- Làm bài tập về nhà

Chuẩn bị cho buổi học tiếp theo

- Tìm hiểu về học không giám sát

Tài liệu tham khảo

- <https://www.deeplearningbook.org/>
- <https://www.manning.com/books/deep-learning-with-python>