# Bài 4:
# Huấn luyện mạng nơ-ron
*(Phần 1)*

# Nội dung

1. Hàm kích hoạt
2. Tiền xử lý dữ liệu
3. Khởi tạo trọng số
4. Các kỹ thuật chuẩn hóa

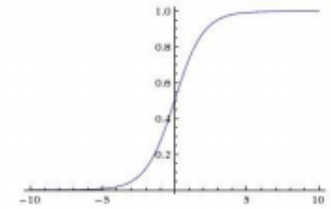
# Hàm kích hoạt

# Hàm kích hoạt

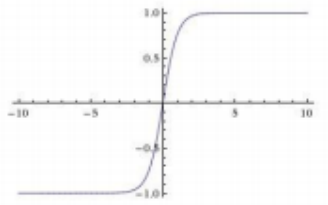
## Activation Functions

**Sigmoid**

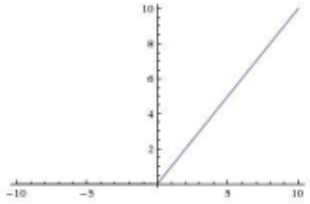
$$\sigma(x) = 1/(1+e^{-x})$$



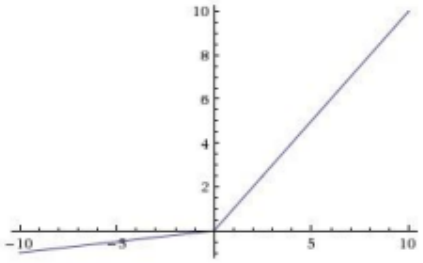
**tanh** tanh(x)



**ReLU** max(0,x)



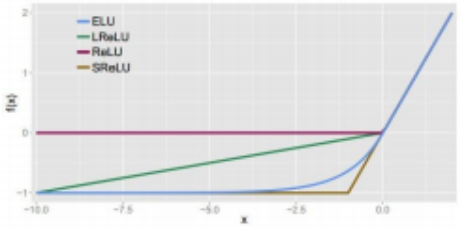
**Leaky ReLU**
max(0.1x, x)



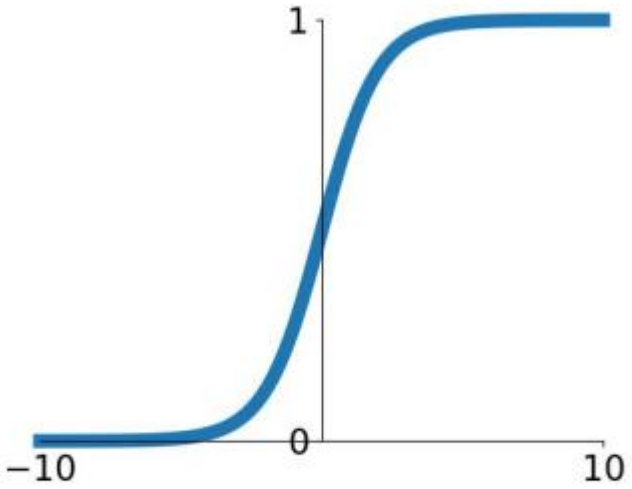
**Maxout** $\max(w_1^T x + b_1, w_2^T x + b_2)$

**ELU**

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$



# Hàm kích hoạt



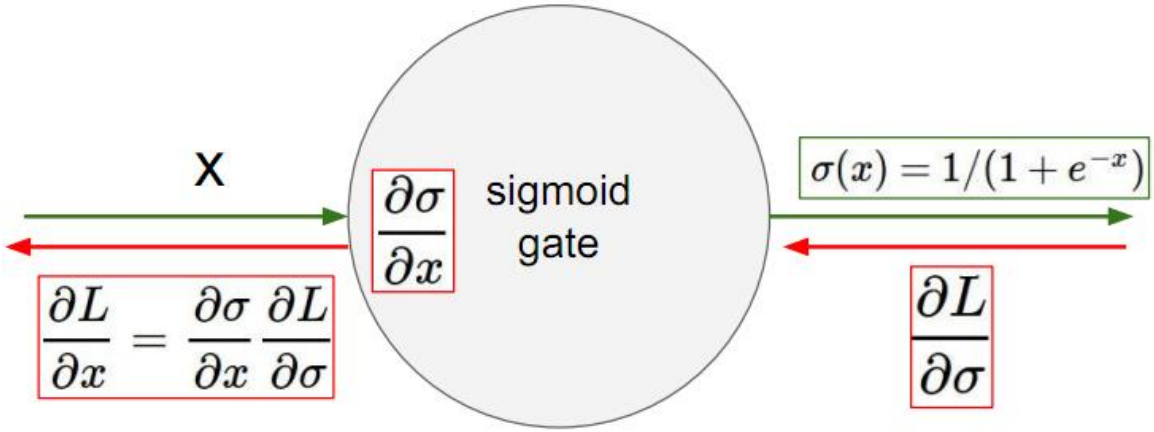

**Sigmoid**

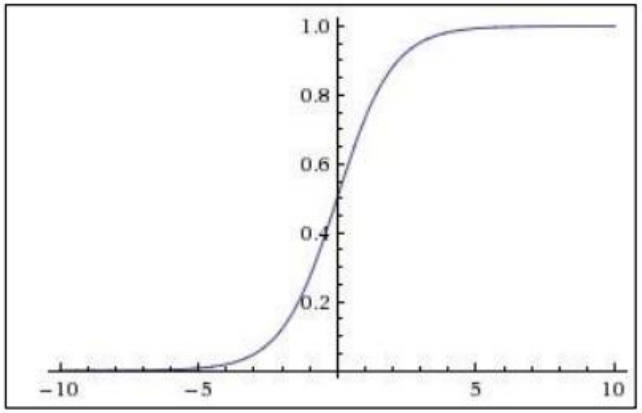
$$\sigma(x) = 1/(1 + e^{-x})$$

- Nhận giá trị trong khoảng [0,1]
- Được dùng phổ biến trong lịch sử mạng nơ-ron do chúng mô phỏng tốt tỉ lệ bắn xung (firing rate) của nơ-ron
- Có 3 nhược điểm:

- Nơ-ron bão hòa triệt tiêu gradient

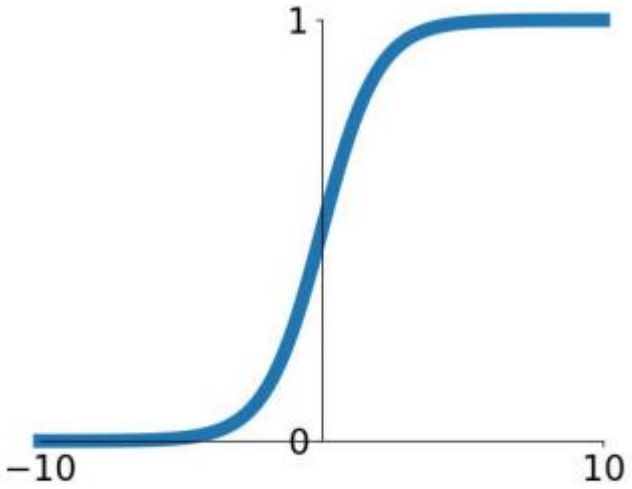
# Hàm kích hoạt




- Điều gì sẽ xảy ra khi x = -10?
- Điều gì sẽ xảy ra khi x = 0?
- Điều gì sẽ xảy ra khi x = 10?

# Hàm kích hoạt




**Sigmoid**

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

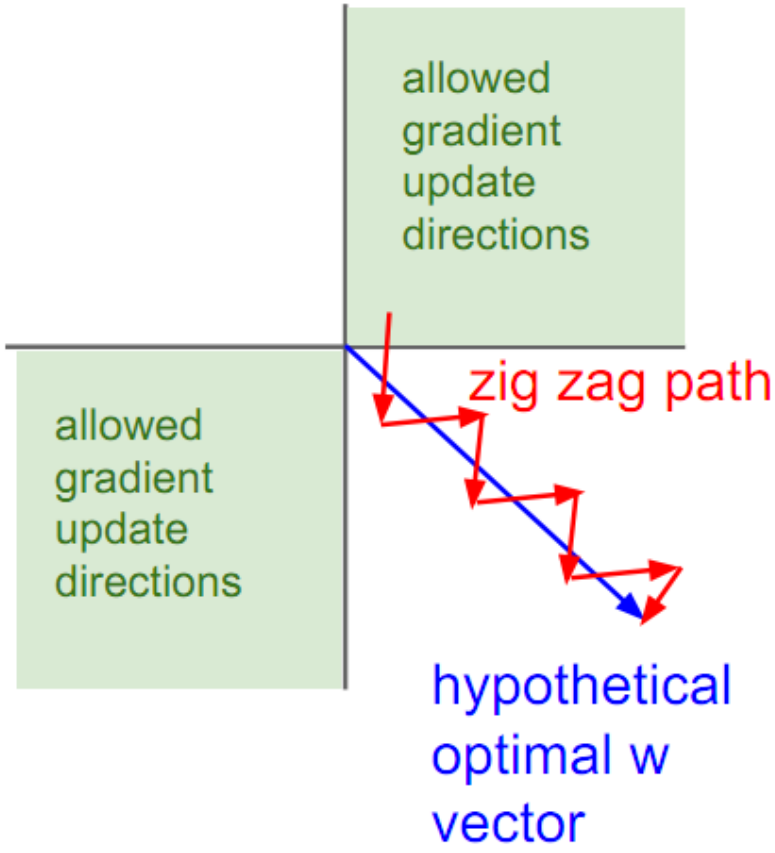
- Nhận giá trị trong khoảng [0,1]
- Được dùng phổ biến trong lịch sử mạng nơ-ron do chúng mô phỏng tốt tỉ lệ bắn xung (firing rate) của nơ-ron
- Có 3 nhược điểm:

- Nơ-ron bão hòa triệt tiêu gradient
- Trung bình đầu ra khác 0

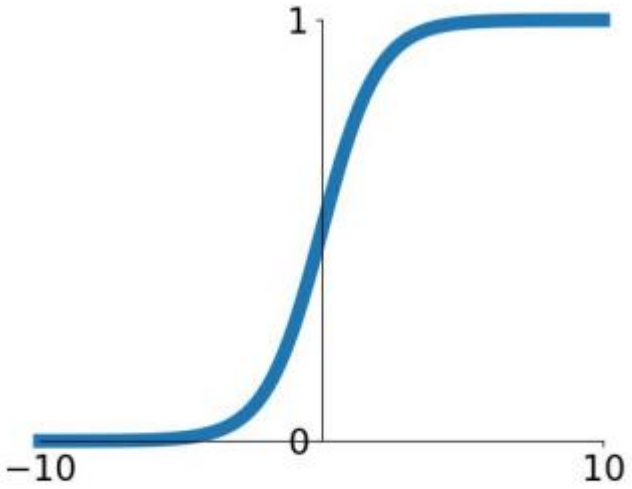
# Hàm kích hoạt

$$f\left(\sum_i w_i x_i + b\right)$$

- Điều gì xảy ra nếu tất cả đầu vào $x_i$ của nơ-ron đều dương?
- Khi đó gradient của hàm mục tiêu đối với **w** sẽ ra sao?
- Tất cả các phần tử của **w** đều cùng dấu với f'(**w**), tức là cùng âm hoặc cùng dương
- Khi đó gradient chỉ có thể hướng theo một số chiều nhất định trong không gian tìm kiếm



# Hàm kích hoạt



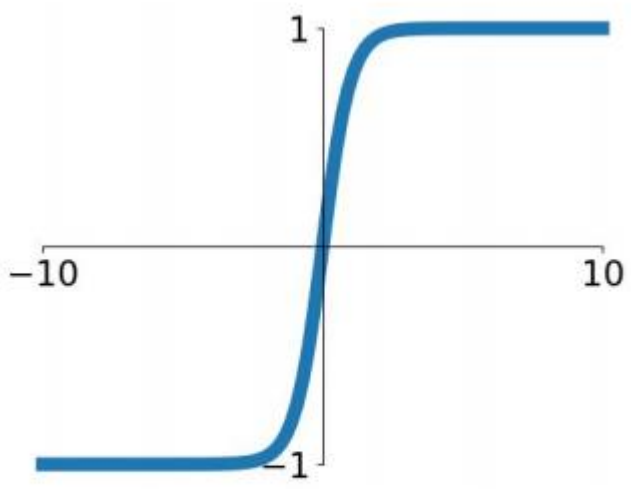

**Sigmoid**

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

- Nhận giá trị trong khoảng [0,1]
- Được dùng phổ biến trong lịch sử mạng nơ-ron do chúng mô phỏng tốt tỉ lệ bắn xung (firing rate) của nơ-ron
- Có 3 nhược điểm:

- Nơ-ron bão hòa triệt tiêu gradient
- Trung bình đầu ra khác 0
- Tính toán hàm mũ exp() tốn kém

# Hàm kích hoạt

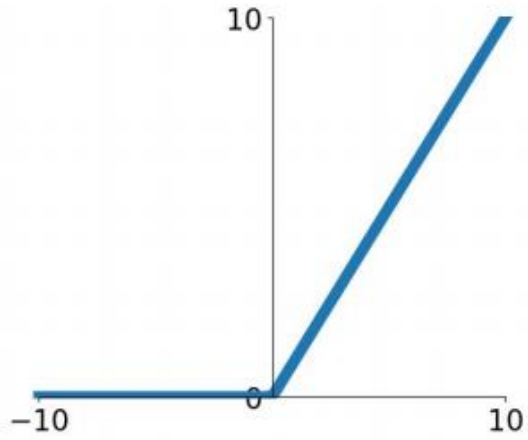



**tanh(x)**

$$\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- Nhận giá trị trong khoảng [-1,1]
- Trung bình đầu ra bằng 0
- Vẫn bị hiện tượng bão hòa, triệt tiêu gradient

# Hàm kích hoạt



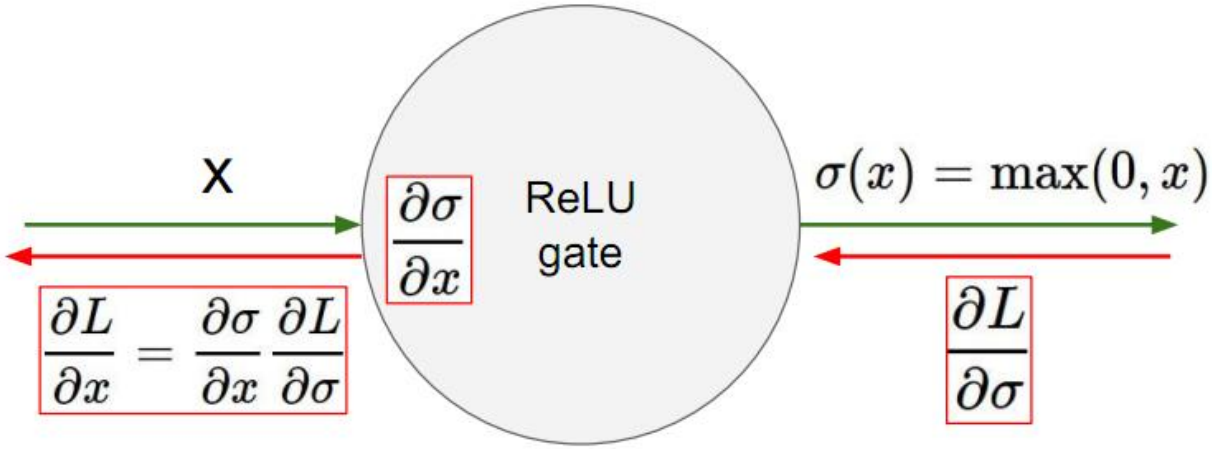

**ReLU**
(Rectified Linear Unit)

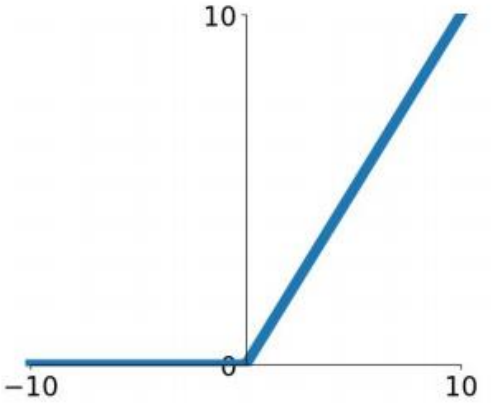
$$f(x) = \max(0,x)$$

- Không bị bão hòa trong vùng dương
- Tính toán hiệu quả
- Trong thực tế hội tụ nhanh hơn sigmoid/tanh (khoảng 6 lần)

- Đầu ra trung bình khác 0
- Và một vấn đề nữa…

# Hàm kích hoạt



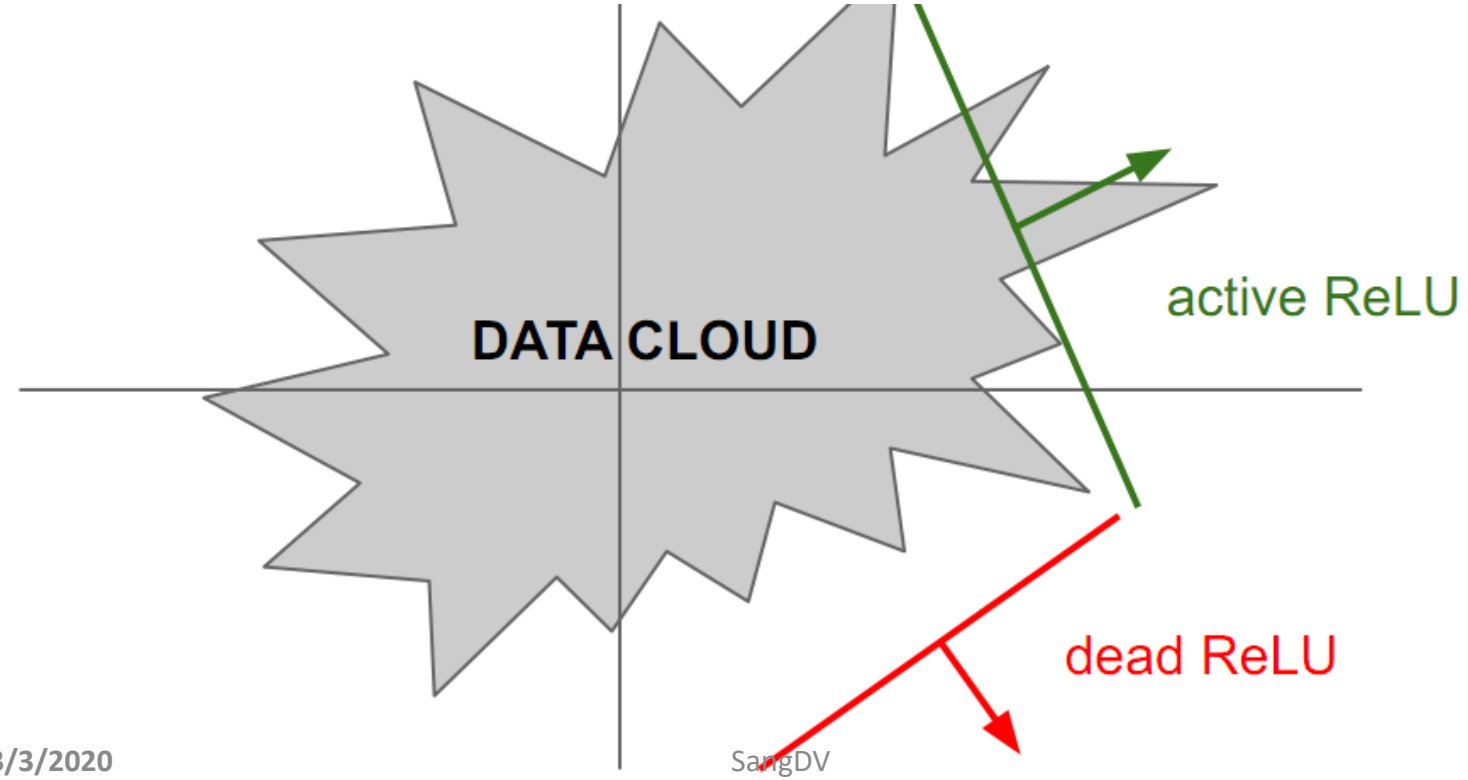

- Điều gì sẽ xảy ra khi x = -10?
- Điều gì sẽ xảy ra khi x = 0?
- Điều gì sẽ xảy ra khi x = 10?

# Hàm kích hoạt

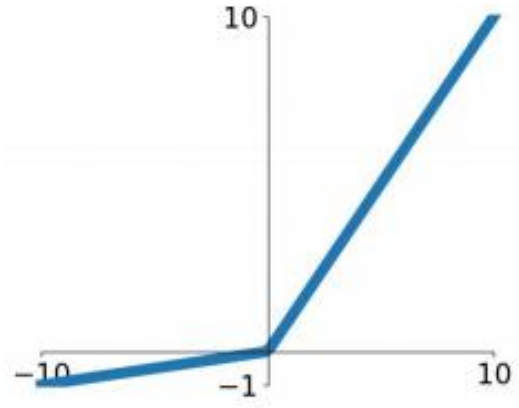
- ReLU bị "văng" ra khỏi tập dữ liệu dẫn tới đầu ra luôn âm và không bao giờ được cập nhật trọng số nữa

  ➔ ReLU chết

- Thường khởi tạo nơ-ron ReLU với bias dương bé (ví dụ 0.01)



# Hàm kích hoạt

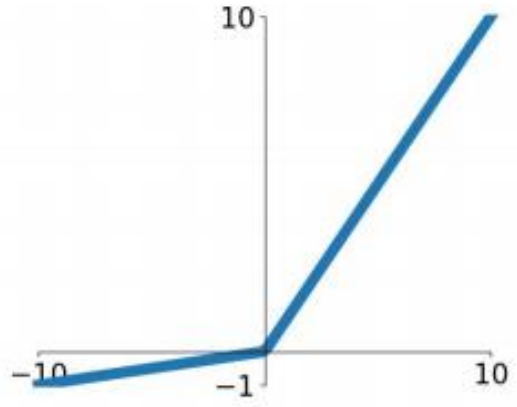



- Không bị bão hòa trong vùng dương
- Tính toán hiệu quả
- Trong thực tế hội tụ nhanh hơn sigmoid/tanh (khoảng 6 lần)
- Không bao giờ “chết”

**Leaky ReLU**

$$f(x) = \max(0.01x, x)$$

# Hàm kích hoạt




- Không bị bão hòa trong vùng dương
- Tính toán hiệu quả
- Trong thực tế hội tụ nhanh hơn sigmoid/tanh (khoảng 6 lần)
- Không bao giờ “chết”

**Leaky ReLU**

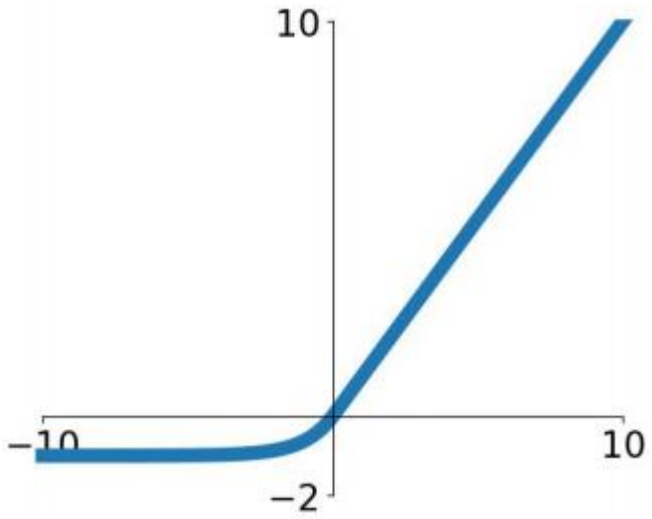
$$f(x) = \max(0.01x, x)$$

**Parametric Rectifier (PReLU)**

$$f(x) = \max(\alpha x, x)$$

backprop into \alpha (parameter)

# Hàm kích hoạt




- Có tất cả ưu điểm của ReLU
- Trung bình đầu ra gần 0 hơn
- Không “chết”

- Tính toán lâu do có hàm exp()

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\left(\exp(x) - 1\right) & \text{if } x \leq 0 \end{cases}$$

# Hàm kích hoạt Maxout

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

- Tổng quát hóa của ReLU và Leaky ReLU
- Tính toán tuyến tính
- Không bão hòa
- Không chết

- Gấp đôi số tham số mỗi nơ-ron

# Hàm kích hoạt

- **Trong thực tế:**

- Thường dùng ReLU. Cẩn thận với tốc độ học để tránh ReLU bị chết.
- Có thể thử Leaky ReLU / Maxout / ELU
- Có thể thử tanh nhưng không kỳ vọng nhiều
- Không dùng sigmoid

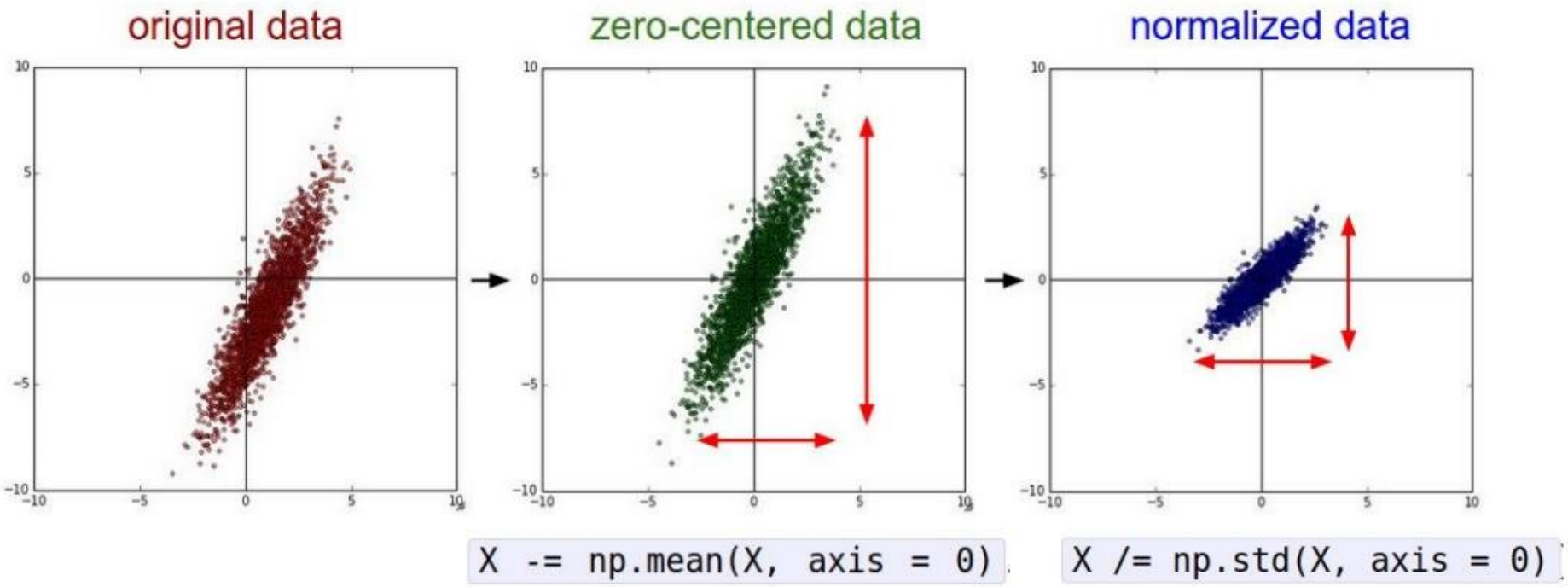
- Gần đây xuất hiện một số hàm kích hoạt mới:

- ReLU6 = min(6, ReLU(x))
- Swish
- Mish

# Tiền xử lý dữ liệu

# Tiền xử lý dữ liệu

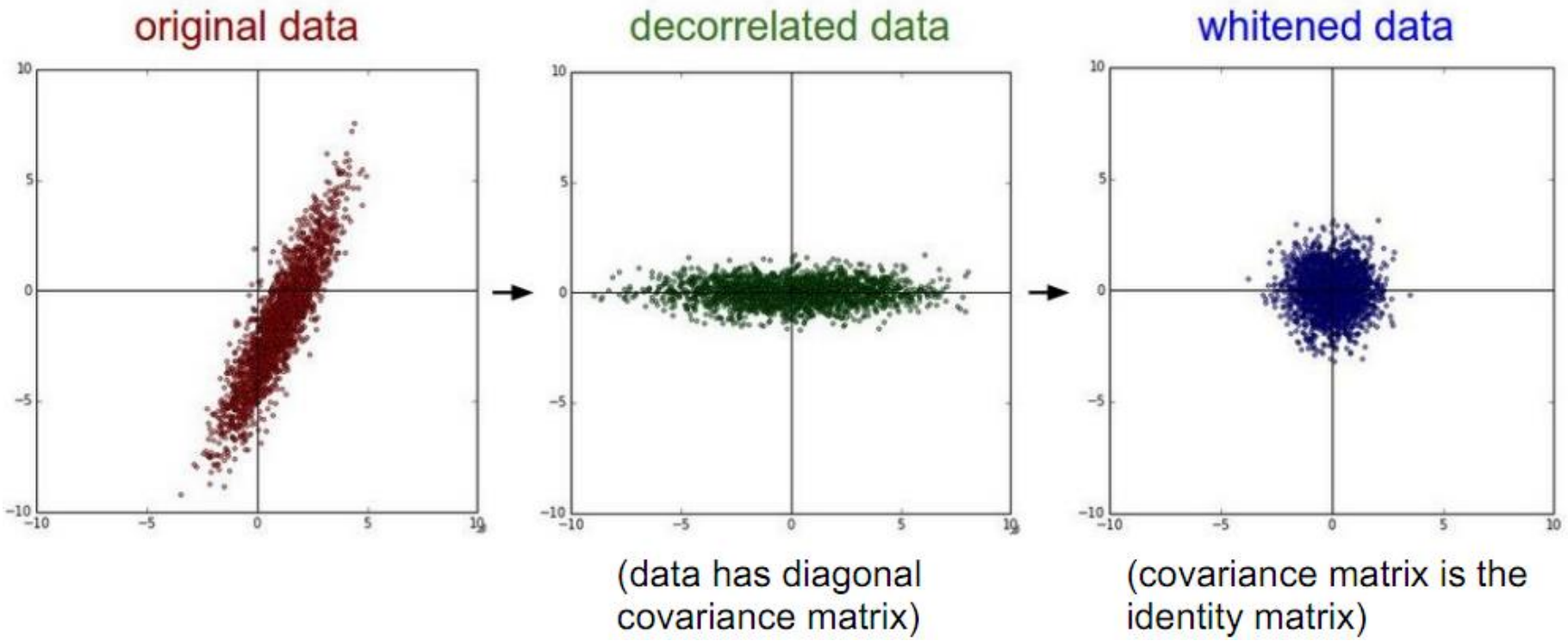
- Biến đổi phân phối dữ liệu về kỳ vọng bằng 0: trừ tất cả mẫu dữ liệu cho mẫu trung bình
- Biến đổi phân phối dữ liệu về độ lệch chuẩn đơn vị




Giả sử X [NxD] là ma trận dữ liệu, mỗi mẫu dữ liệu là một dòng

# Tiền xử lý dữ liệu

- Trong thực tế có thể sử dụng PCA hoặc Whitening dữ liệu



# Tiền xử lý dữ liệu

- Ví dụ với bộ CIFAR10 với các ảnh kích thước 32x32x3

- Subtract the mean image (e.g. AlexNet)
  (mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet)
  (mean along each channel = 3 numbers)
- Subtract per-channel mean and
  Divide by per-channel std (e.g. ResNet)
  (mean along each channel = 3 numbers)

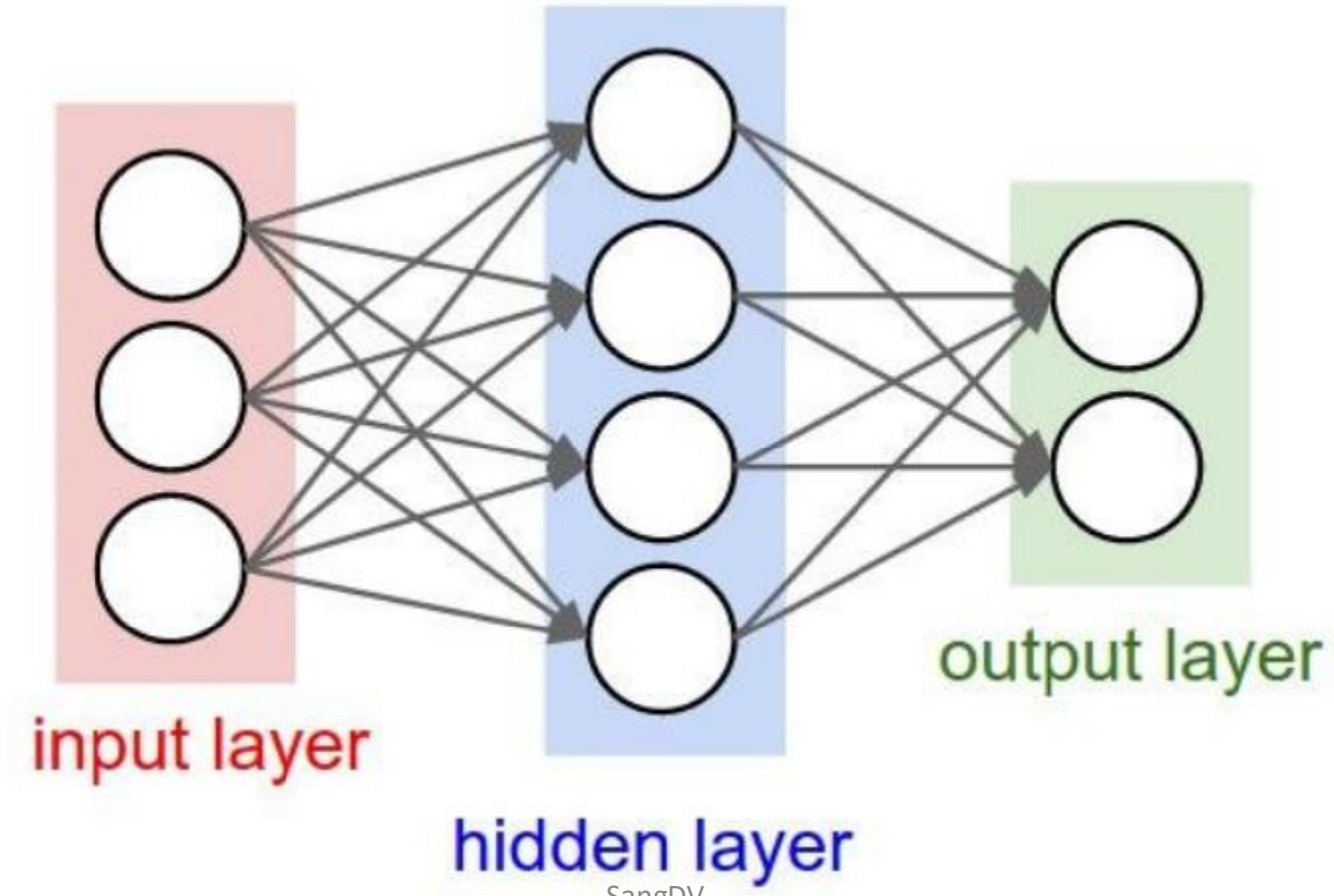
- Thường ít sử dụng PCA hoặc whitening

# Khởi tạo trọng số

# Khởi tạo trọng số

- Điều gì xảy ra nếu khởi tạo tất cả các trọng số bằng 0?

➔Không có ý nghĩa do tất cả các nơ-ron đều học và xử lý giống hệt nhau



# Khởi tạo trọng số

- Ý tưởng thứ nhất: Khởi tạo ngẫu nhiên các giá trị nhỏ (Ví dụ theo phân bố chuẩn với kỳ vọng 0, độ lệch chuẩn 0.01)

```
W = 0.01 * np.random.randn(Din, Dout)
```

Làm việc ổn với các mạng nơ-ron nhỏ, nhưng có vấn đề với các mạng nơ-ron sâu hơn.

# Khởi tạo trọng số

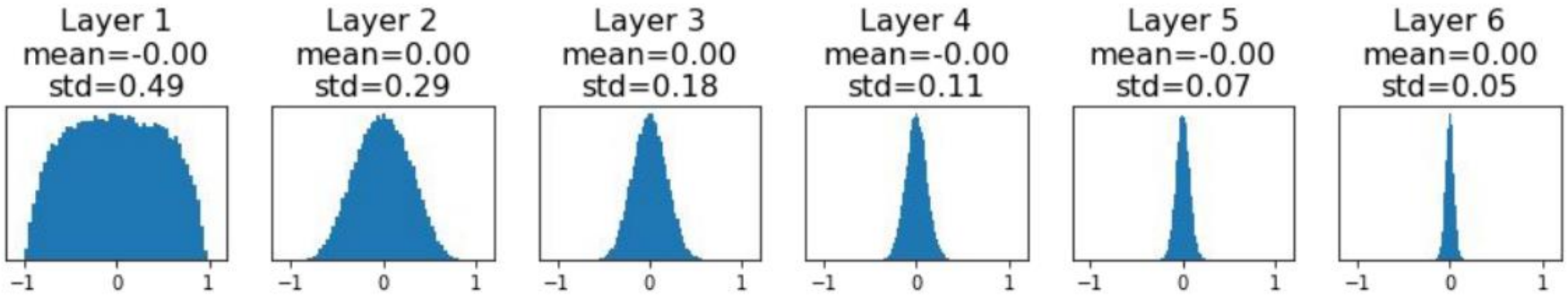
```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Forward pass for a 6-layer net with hidden size 4096

All activations tend to zero for deeper network layers

**Q**: What do the gradients dL/dW look like?

**A**: All zero, no learning =(




- Gradient dL/dW = 0

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

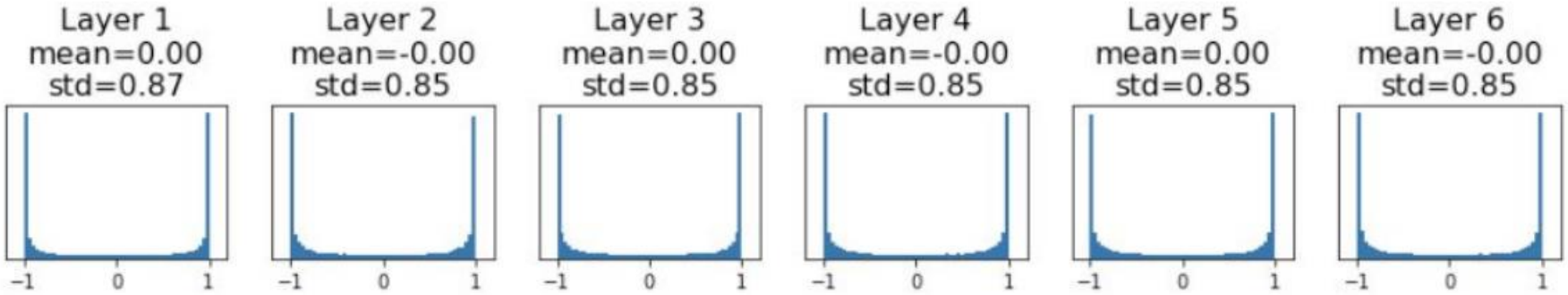
# Khởi tạo trọng số

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = 0.05 * np.random.randn(Din, Dout)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Increase std of initial weights from 0.01 to 0.05

All activations saturate

**Q**: What do the gradients look like?




- Gradient dL/dW = 0

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

# Khởi tạo trọng số: Khởi tạo Xavier

- Giả sử $x$ và $w$ là iid, độc lập nhau và trung bình bằng 0
- Tính toán theo chiều tiến forward:

$$var(y) = var(w_1x_1 + w_2x_2 + ... + w_{Nin}x_{Nin} + b)$$

$$var(w_ix_i) = E(x_i)^2 var(w_i) + E(w_i)^2 var(x_i) + var(w_i)var(x_i)$$

$$var(y) = N_{in} * var(w_i) * var(x_i)$$

$$N_{in} * var(w_i) = 1$$

$$var(w_i) = 1/ N_{in}$$

- Tương tự với luồng tín hiệu gradient backward:

$$var(w_i) = 1/ N_{out}$$

- Trung bình:

$$var(w_i) = 2/ (N_{in} + N_{out})$$

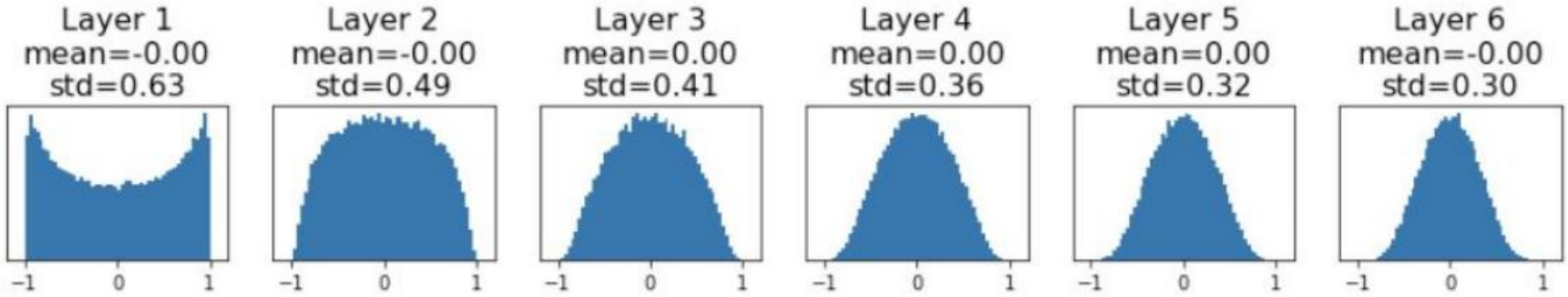
# Khởi tạo trọng số

- Khởi tạo Xavier

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

"Xavier" initialization:
std = 1/sqrt(Din)

"Just right": Activations are nicely scaled for all layers!



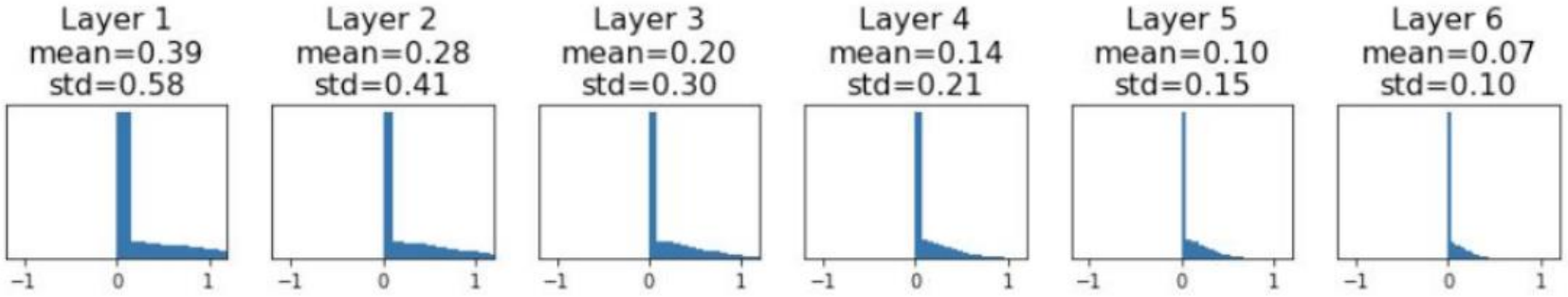
# Khởi tạo trọng số

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) / np.sqrt(Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

Change from tanh to ReLU

Xavier assumes zero centered activation function

Activations collapse to zero again, no learning =(




$$var(y) = var(w_1x_1 + w_2x_2 + ... + w_{Nin}x_{Nin} + b)$$

$$var(y) = N_{in} /2 * var(w_i) * var(x_i)$$

$$N_{in} /2 * var(w_i) = 1$$

$$var(w_i) = 2/ N_{in}$$

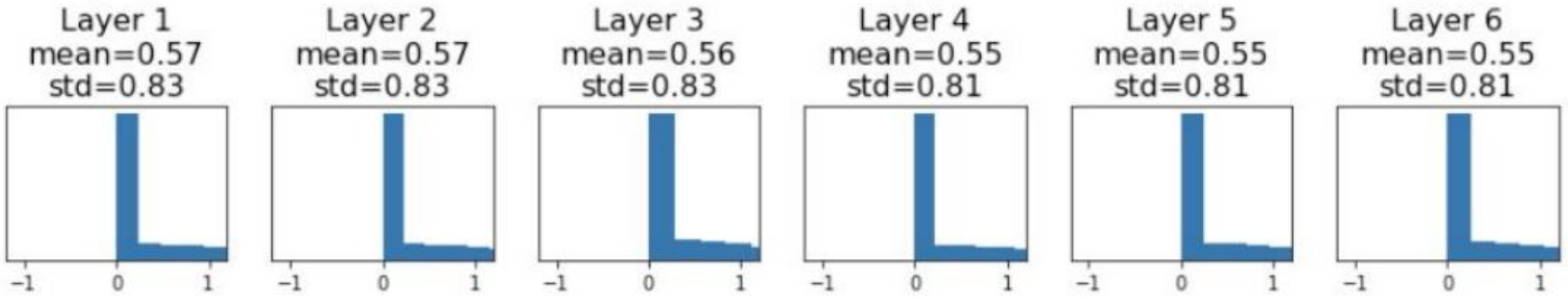
# Khởi tạo trọng số

- He / MSRA Initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for Din, Dout in zip(dims[:-1], dims[1:]):
    W = np.random.randn(Din, Dout) * np.sqrt(2/Din)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

ReLU correction: std = sqrt(2 / Din)

"Just right": Activations are nicely scaled for all layers!



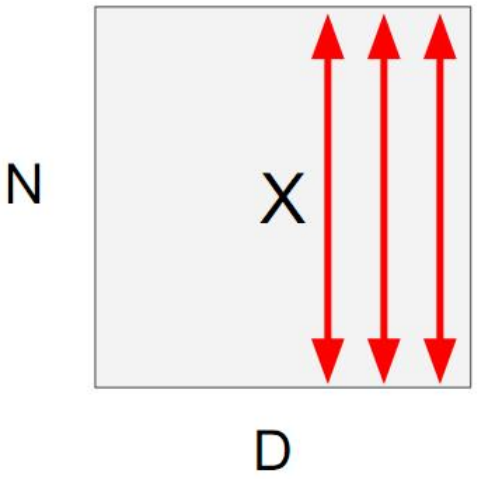
# Các kỹ thuật chuẩn hóa

# Batch Normalization

- Muốn hàm kích hoạt có phân bố đầu ra với trung bình bằng 0 và độ lệch chuẩn đơn vị? Hãy biến đổi theo ý tưởng đó!

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

**Input:** $x : N \times D$




$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$ Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$ Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ Normalized x, Shape is N x D

# Batch Normalization

- Ràng buộc kỳ vọng bằng 0 và độ lệch chuẩn đơn vị là quá chặt! Có thể khiến mô hình bị underfitting.

➔ Nới lỏng cho mô hình, tạo lối thoát cho mô hình nếu nó không muốn bị ràng buộc.

**Input:** $x : N \times D$

**Learnable scale and shift parameters:**

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$, $\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Output, Shape is N x D

# Batch Normalization

- Không thể tính kỳ vọng và phương sai theo lô dữ liệu (batch) lúc suy diễn

**Input:** $x : N \times D$

**Learnable scale and shift parameters:**

$$\gamma, \beta : D$$

Learning $\gamma = \sigma$ ,
$\beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$ Per-channel mean, shape is D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$ Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ Output, Shape is N x D

# Batch Normalization

- Lúc suy diễn, BN đơn giản là phép biến đổi tuyến tính. Có thể áp dụng phía sau lớp FC hoặc conv

**Input:** $x : N \times D$

**Learnable scale and shift parameters:**

$\gamma, \beta : D$

During testing batchnorm becomes a linear operator!
Can be fused with the previous fully-connected or conv layer

$\mu_j =$ (Running) average of values seen during training — Per-channel mean, shape is D

$\sigma_j^2 =$ (Running) average of values seen during training — Per-channel var, shape is D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$ — Normalized x, Shape is N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$ — Output, Shape is N x D

# Batch Normalization

Batch Normalization for **fully-connected** networks

$x: N \times D$

Normalize ↓

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

$x: N \times C \times H \times W$

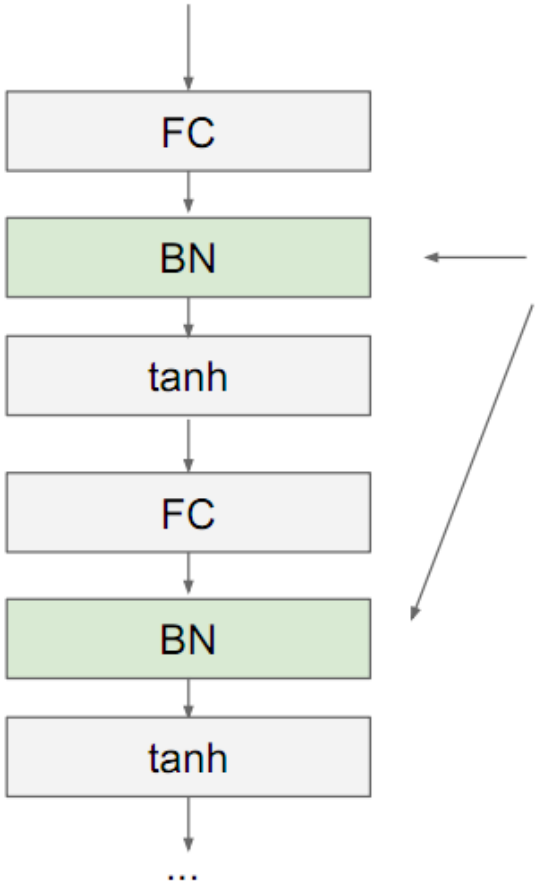
Normalize ↓ ↓ ↓

$\mu, \sigma: 1 \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$y = \gamma(x-\mu)/\sigma+\beta$

# Batch Normalization




Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

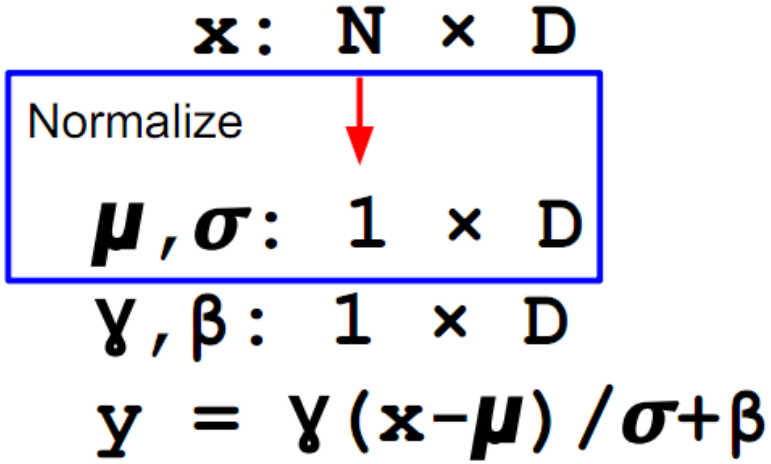
$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

# Ưu điểm của BN

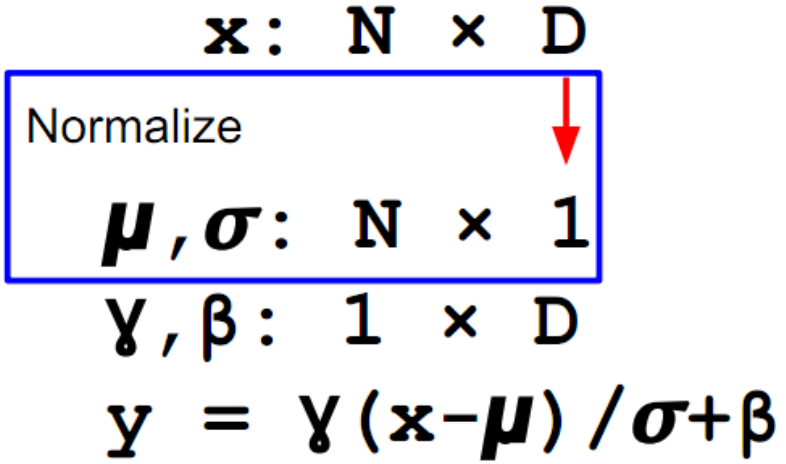
- Dễ dàng hơn khi huấn luyện các mạng sâu!
- Cải thiện luồng gradient
- Cho phép huấn luyện với tốc độ học cao hơn, hội tụ nhanh hơn
- Mạng ổn định hơn, đỡ phụ thuộc hơn với khởi tạo trọng số
- Một kiểu ràng buộc khi huấn luyện
- Khi suy diễn không cần tính toán thêm, đơn giản là biến đổi tuyến tính
- Khi huấn luyện và khi suy diễn làm việc khác nhau: đây là nguồn gốc gây ra nhiều lỗi!

# Chuẩn hóa theo lớp

**Batch Normalization** for fully-connected networks

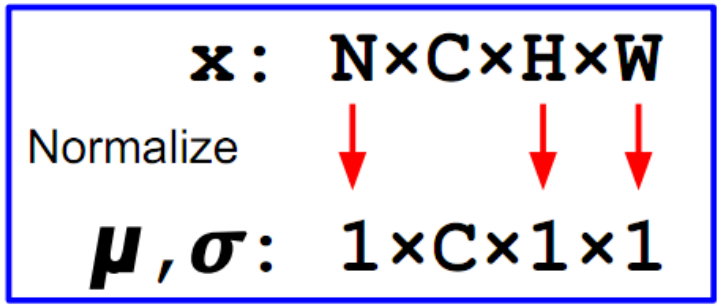



**Layer Normalization** for fully-connected networks
Same behavior at train and test!
Can be used in recurrent networks

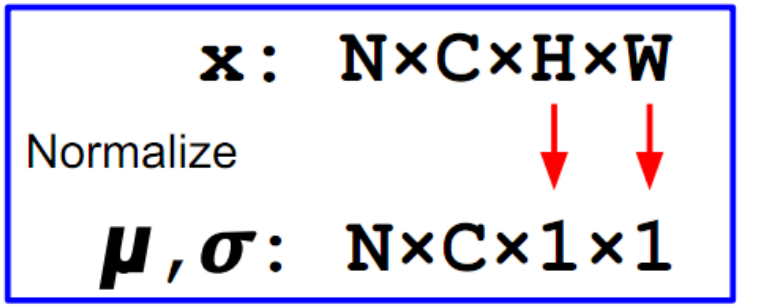


# Chuẩn hóa theo mẫu

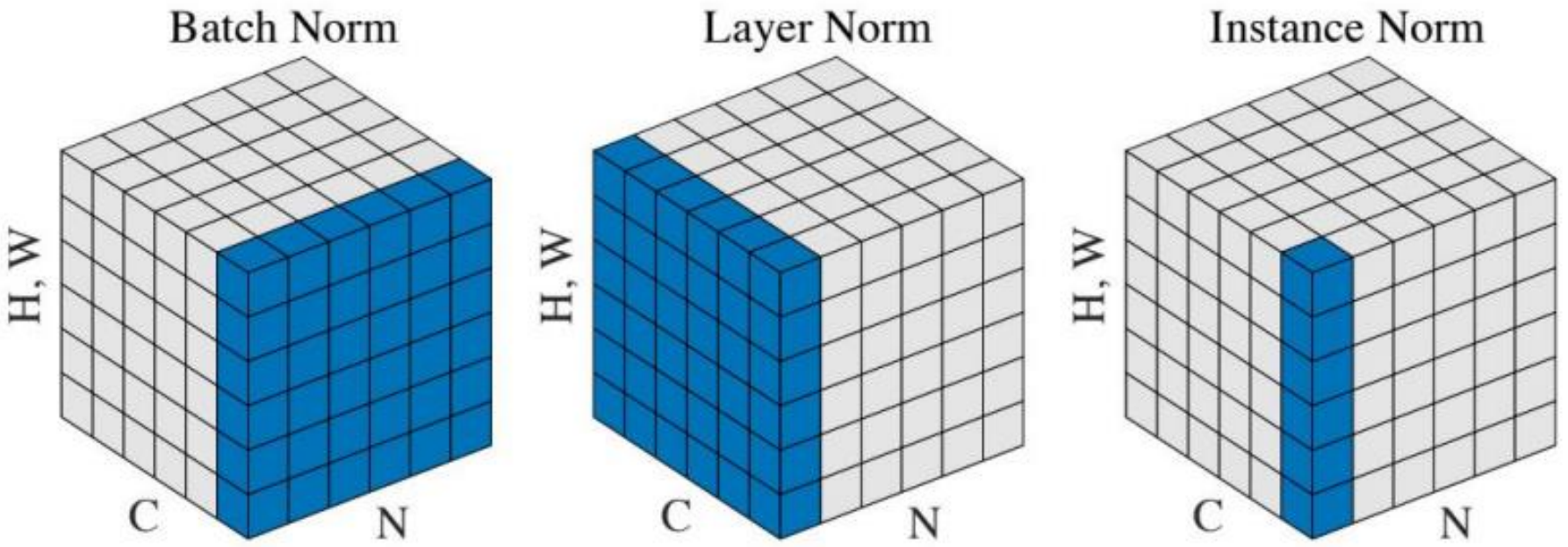
**Batch Normalization** for convolutional networks



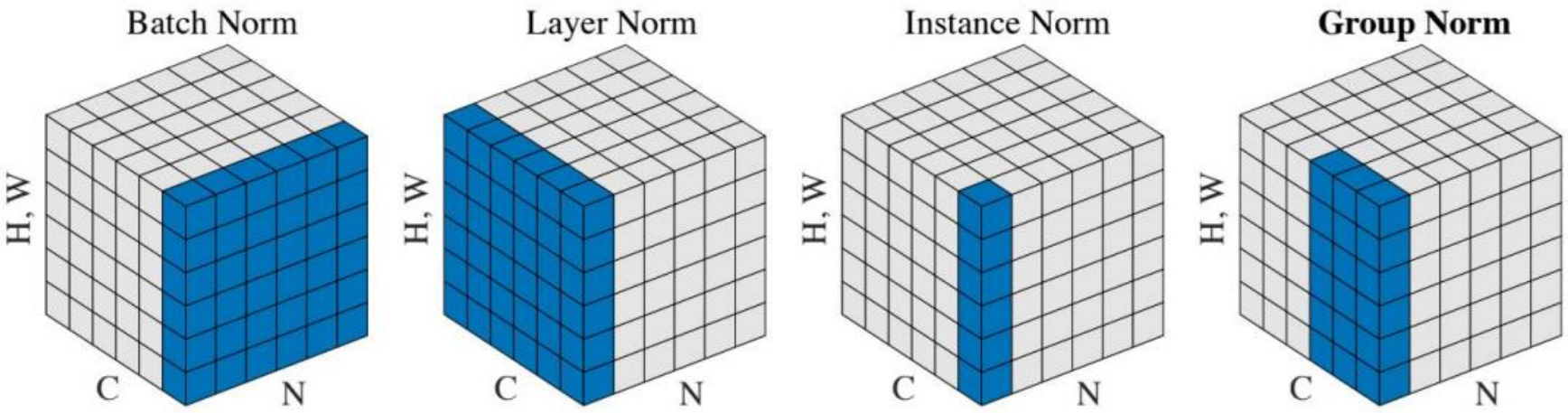

**Instance Normalization** for convolutional networks
Same behavior at train / test!



# So sánh các phương pháp chuẩn hóa



# Chuẩn hóa theo nhóm



# Tài liệu tham khảo

1. Bài giảng biên soạn dựa trên khóa cs231n của Stanford, bài giảng số 7:

http://cs231n.stanford.edu

2. Khởi tạo Xavier: https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/