



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



# Huấn luyện mạng nơ-ron (Phần 1)

Hà Nội, 8/2021

# Nội dung

1. Hàm kích hoạt
2. Tiền xử lý dữ liệu
3. Khởi tạo trọng số
4. Các kỹ thuật chống học quá khớp
5. Làm giàu dữ liệu

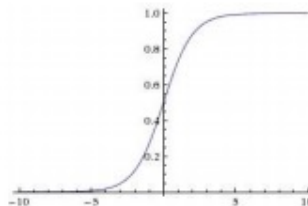
# Hàm kích hoạt

# Hàm kích hoạt

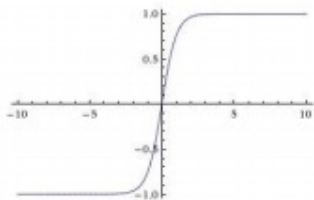
## Activation Functions

### Sigmoid

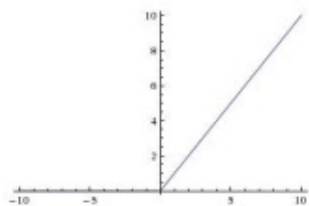
$$\sigma(x) = 1/(1 + e^{-x})$$



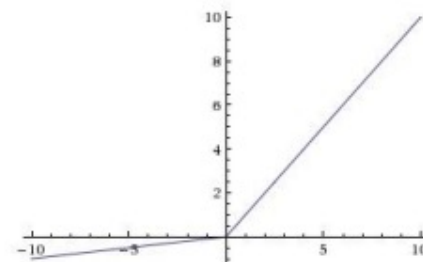
### tanh tanh(x)



### ReLU max(0,x)



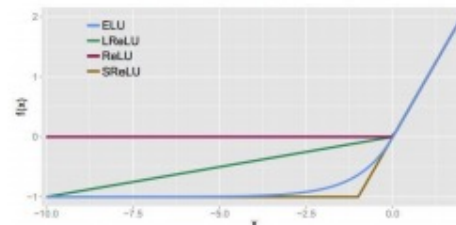
### Leaky ReLU $\max(0.1x, x)$



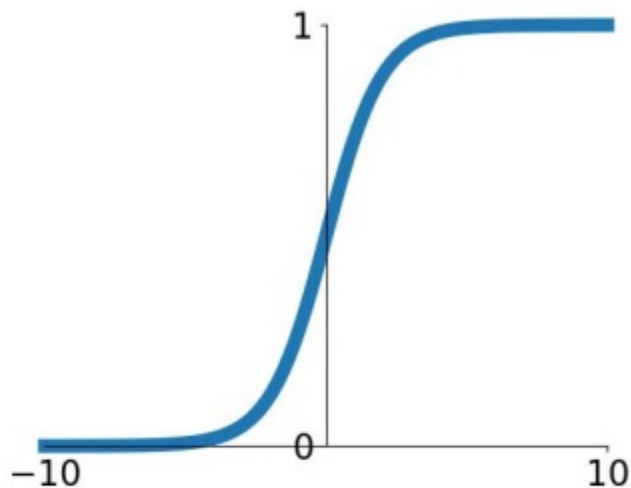
### Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

### ELU

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



# Hàm kích hoạt

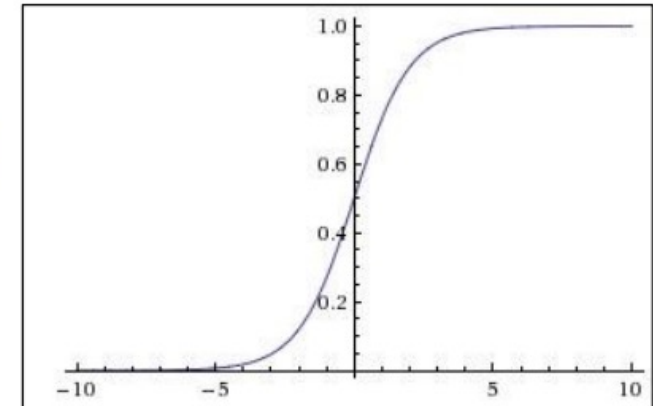
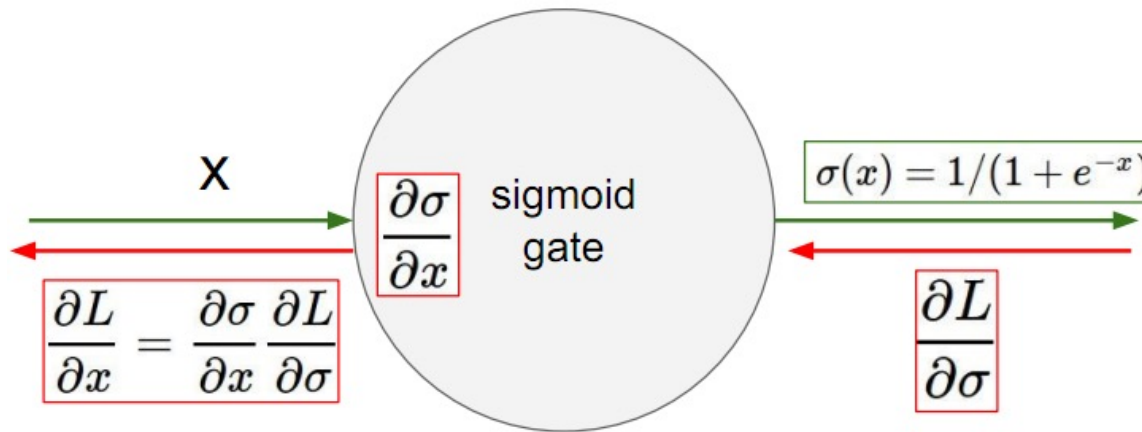


**Sigmoid**

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

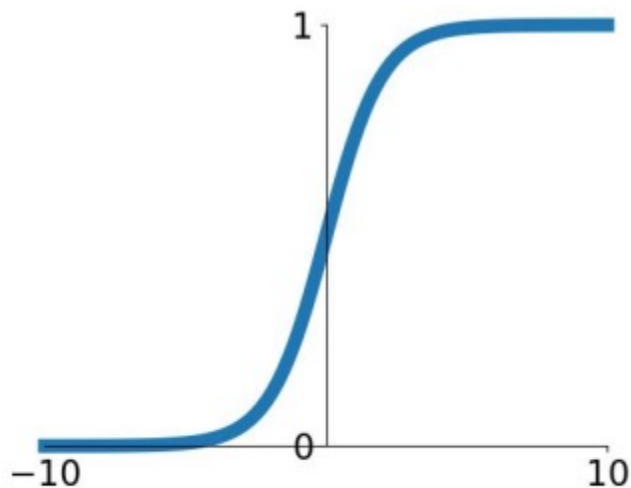
- Nhận giá trị trong khoảng  $[0,1]$
- Được dùng phổ biến trong lịch sử mạng nơ-ron do chúng mô phỏng tốt tỉ lệ bắn xung (firing rate) của nơ-ron
- Có 3 nhược điểm:
  - Nơ-ron bão hòa triệt tiêu gradient

# 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**

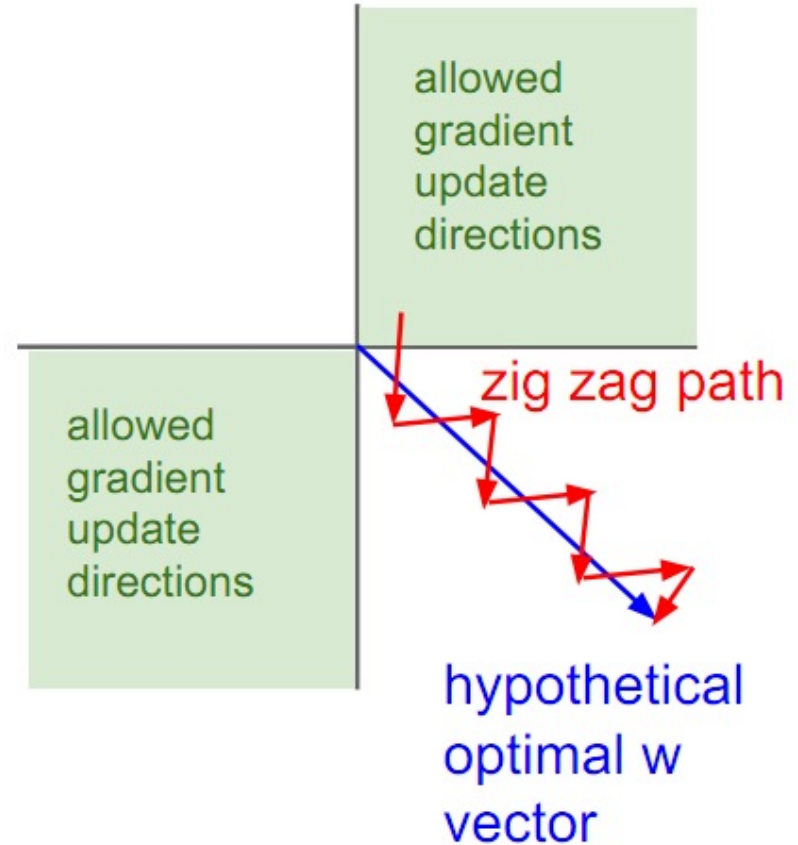
$$\sigma(x) = 1 / (1 + e^{-x})$$

- 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

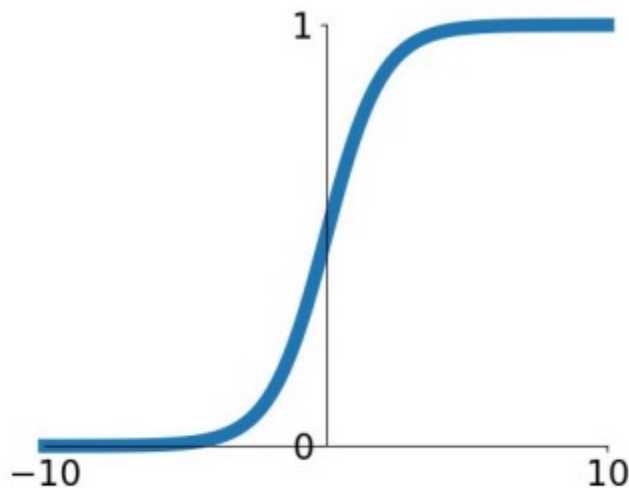
$$f\left(\sum_i w_i x_i + b\right)$$

- 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  $\mathbf{w}$  sẽ ra sao?
- Tất cả các phần tử của  $\mathbf{w}$  đều cùng dấu với  $f'(\mathbf{w})$ , 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

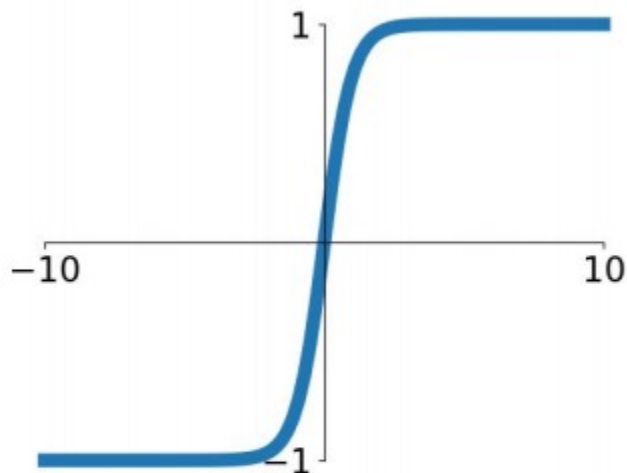


**Sigmoid**

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

- 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

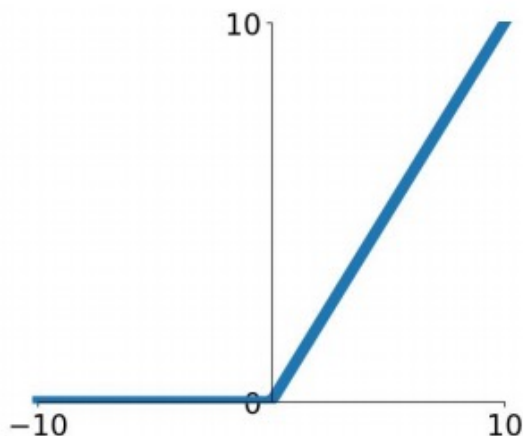


- Nhận giá trị trong khoảng  $[-1, 1]$
- Trung bình đầu ra bằng 0
- Vẫn bị hiện tượng bão hòa, triệt tiêu gradient

**$\tanh(x)$**

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

# Hàm kích hoạt



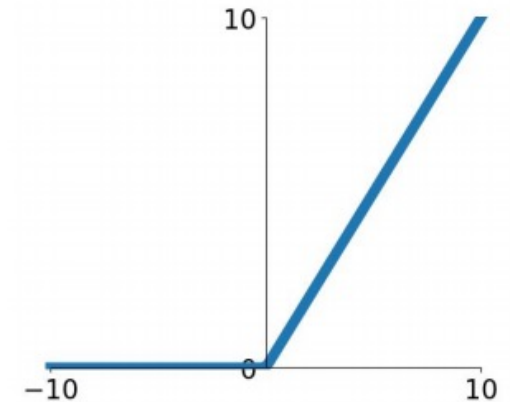
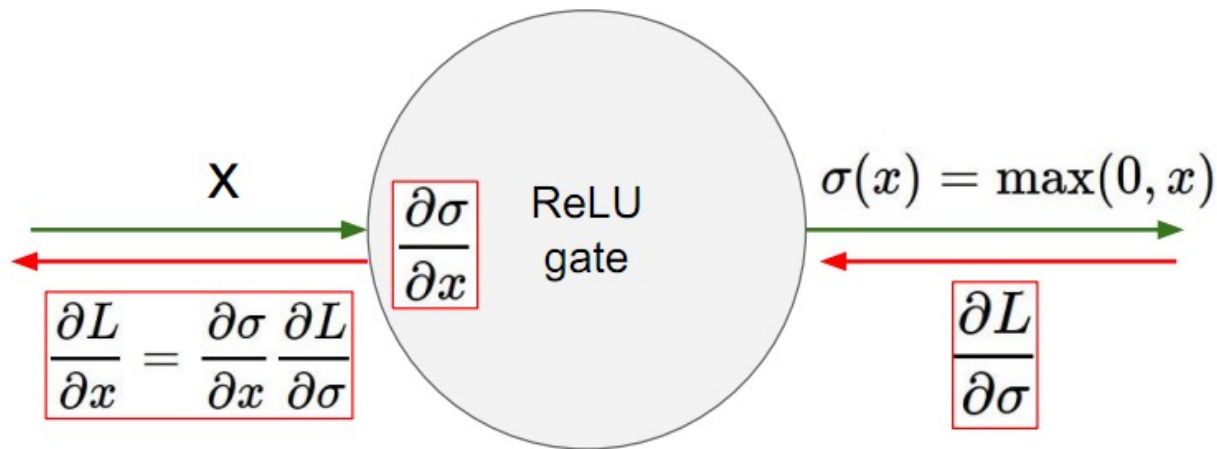
- Không bị bão hòa trong vùng dương
- Tính toán hiệu quả
- Trong thực tế hội tụ nhanh hơn sigmoid/tanh (khoảng 6 lần)

**ReLU**  
(Rectified Linear Unit)

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

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

# Hàm kích hoạt



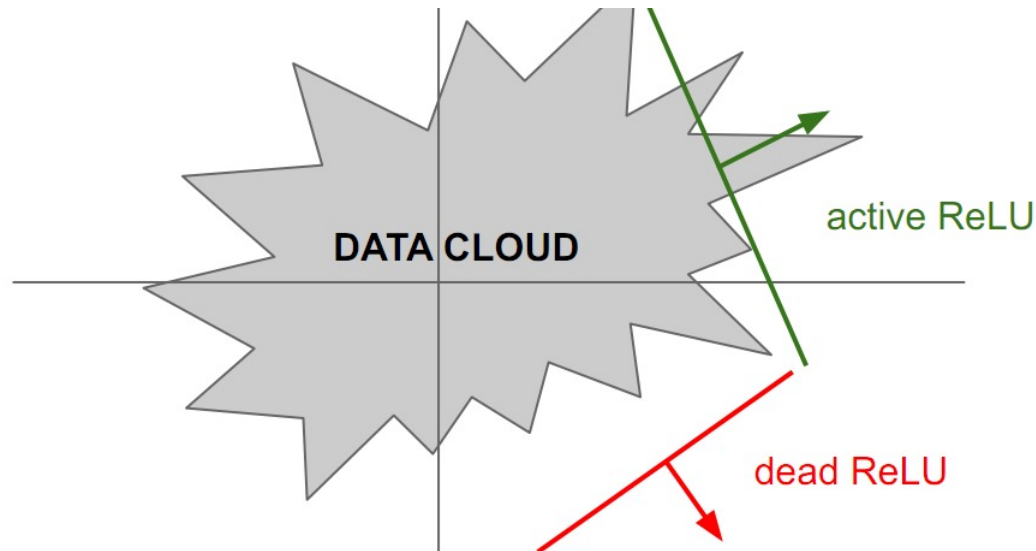
- Điều gì sẽ xảy ra khi  $x = -10$ ?
- Điều gì sẽ xảy ra khi  $x = 0$ ?
- Điều gì sẽ xảy ra khi  $x = 10$ ?

# Hàm kích hoạt

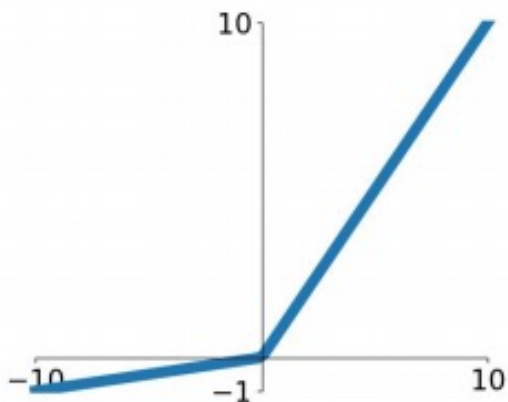
- ReLU bị “văng” ra khỏi tập dữ liệu dẫn tới đầu ra luôn âm và không bao giờ được cập nhật trọng số nữa

→ ReLU chết

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



# Hàm kích hoạt

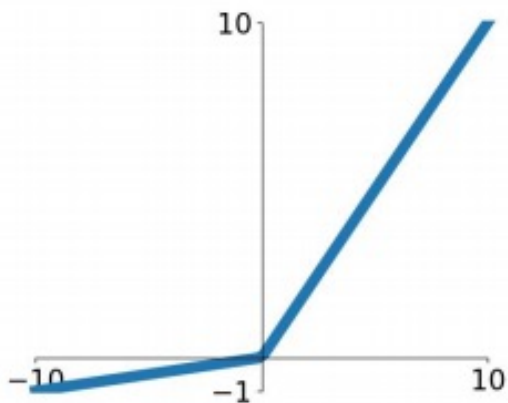


## Leaky ReLU

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

- 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”

# 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

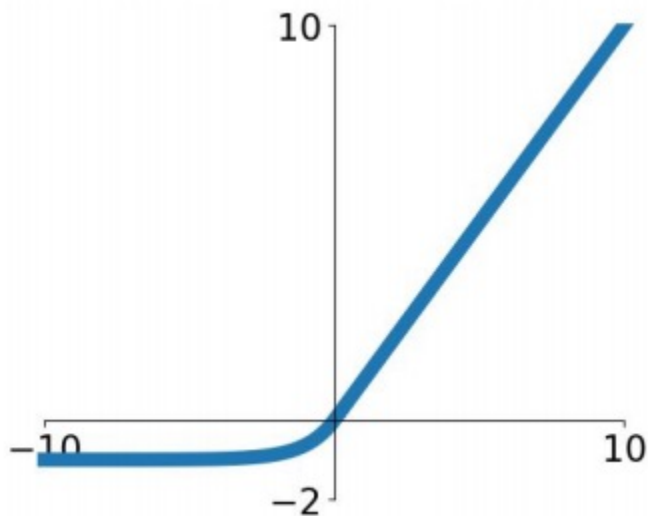
$$f(x) = \max(0.01x, x)$$

## Parametric Rectifier (PReLU)

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

backprop into  $\alpha$   
(parameter)

# Hàm kích hoạt ELU



- Có tất cả ưu điểm của ReLU
- Trung bình đầu ra gần 0 hơn
- Không “chết”
- Tính toán lâu do có hàm  $\exp()$

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



# 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

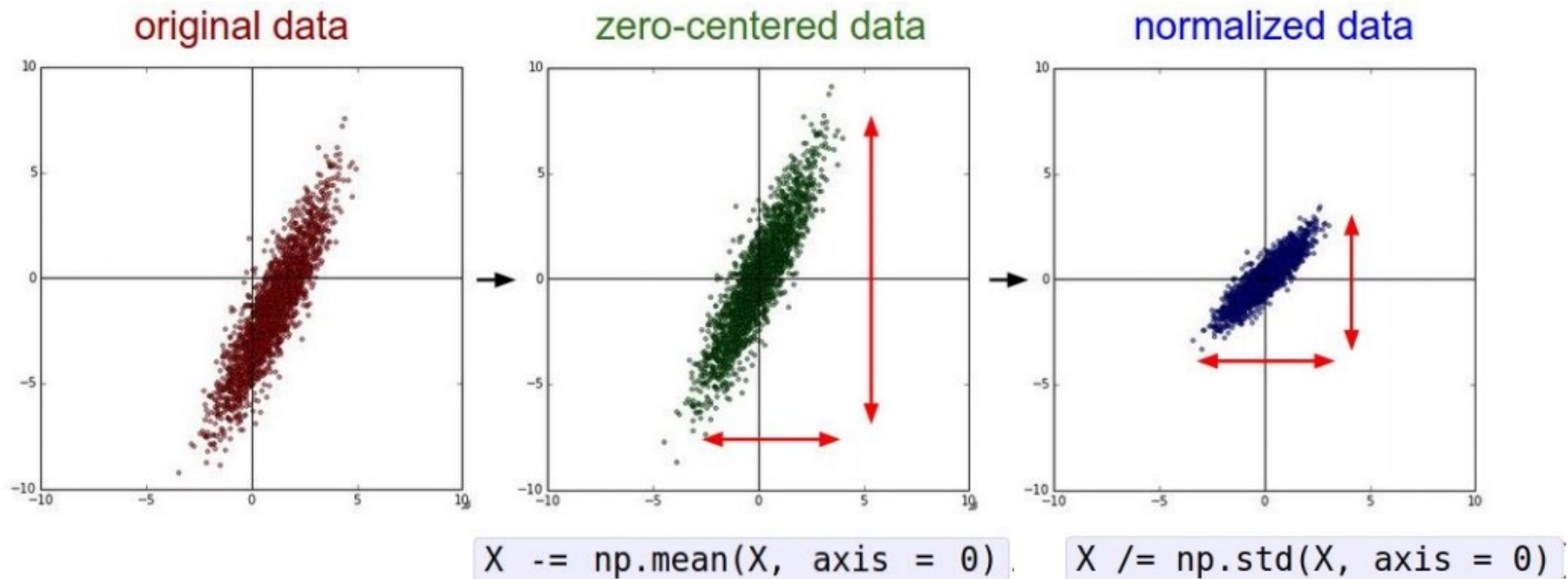
# Hàm kích hoạt

- **Trong thực tế:**
  - Thường dùng ReLU. Cần 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 =  $\min(6, \text{ReLU}(x))$
  - Swish
  - Mish

# Tiền xử lý dữ liệu

# Tiền xử lý dữ liệu

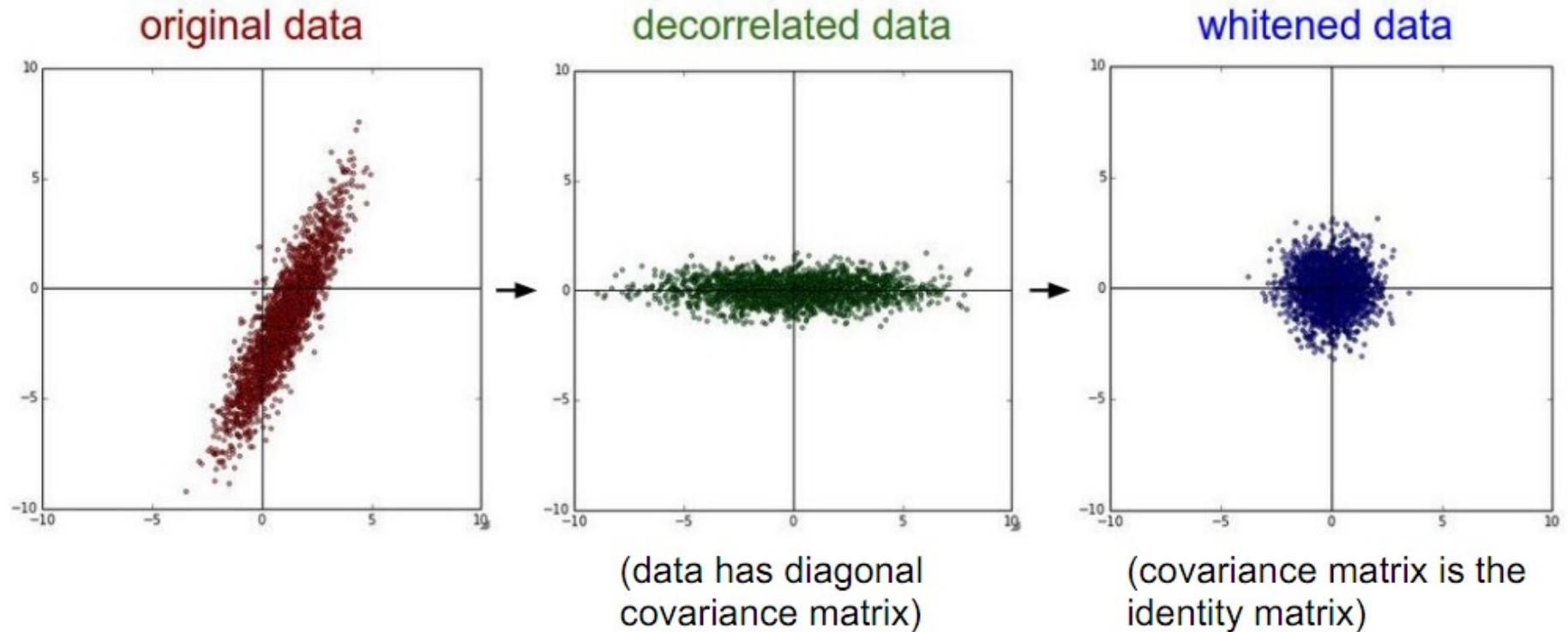
- Biến đổi phân phối dữ liệu về kỳ vọng bằng 0: trừ tất cả mẫu dữ liệu cho mẫu trung bình
- Biến đổi phân phối dữ liệu về độ lệch chuẩn đơn vị



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



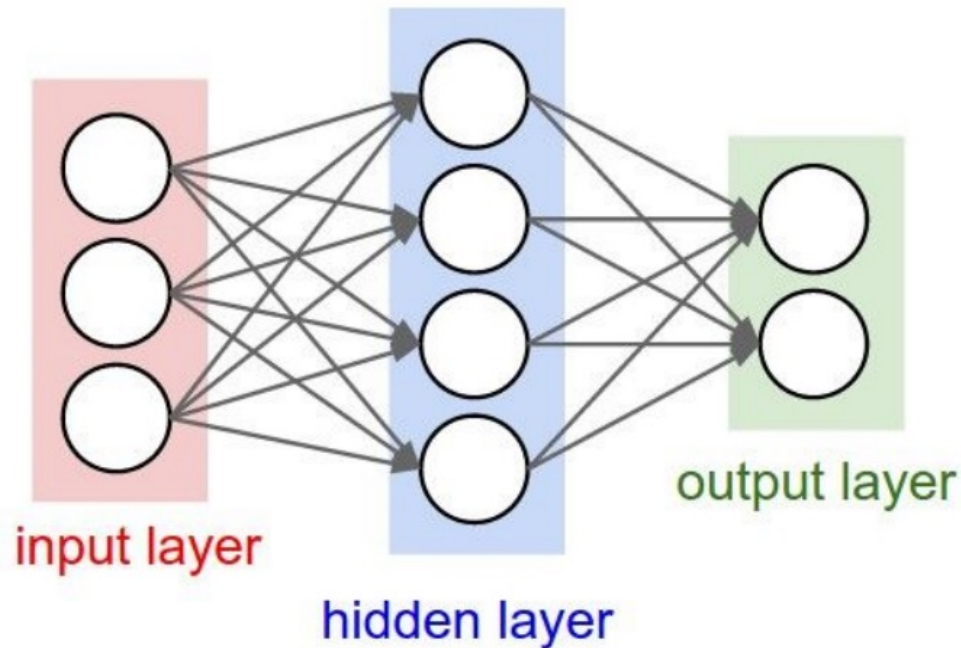
# Tiền xử lý dữ liệu

- Ví dụ với bộ CIFAR10 với các ảnh kích thước
  - 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.

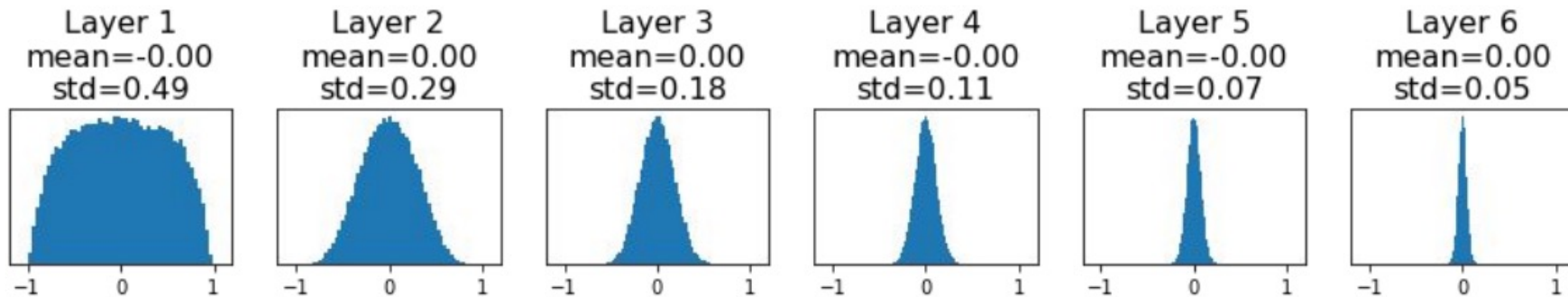
# Khởi tạo trọng số

```
dims = [4096] * 7      Forward pass for a 6-layer  
                        net with hidden size 4096  
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)
```

All activations tend to zero for deeper network layers

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

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



- Gradient  $dL/dW = 0$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

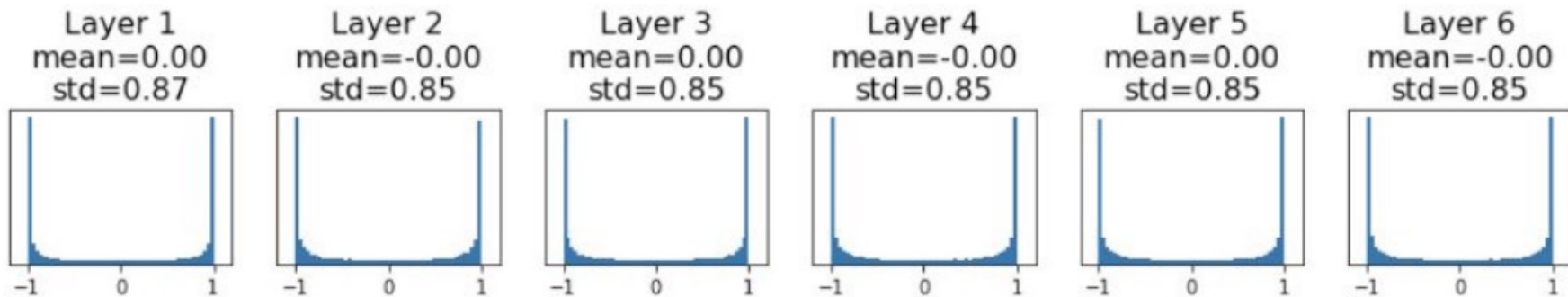
$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

# Khởi tạo trọng số

```
dims = [4096] * 7    Increase std of initial
hs = []              weights from 0.01 to 0.05
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)
```

All activations saturate

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



- Gradient  $dL/dW = 0$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

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

# Khởi tạo trọng số Xavier

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

$$\text{var}(y) = \text{var}(w_1x_1 + w_2x_2 + \dots + w_{N_{in}}x_{N_{in}} + b)$$

$$\text{var}(w_ix_i) = E(x_i)^2 \text{var}(w_i) + E(w_i)^2 \text{var}(x_i) + \text{var}(w_i) \text{var}(x_i)$$

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

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

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

- Tương tự với luồng tín hiệu gradient backward:

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

- Trung bình:

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

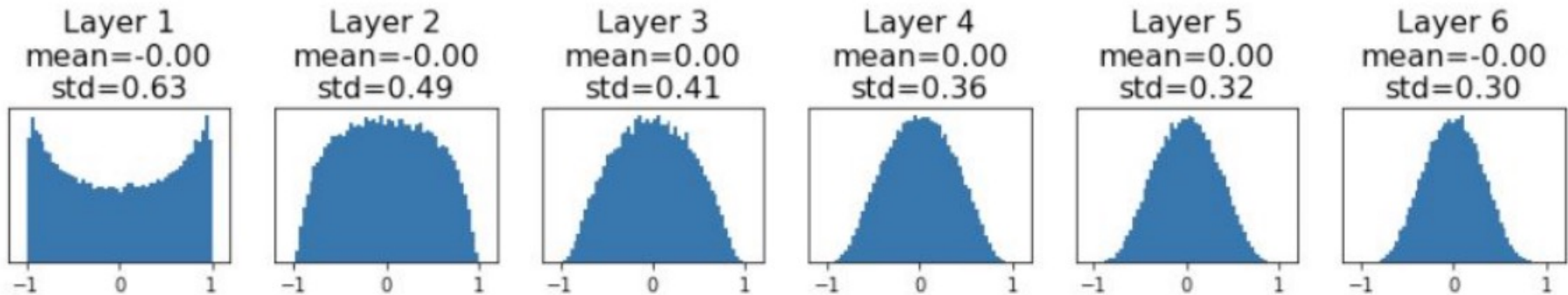
# Khởi tạo trọng số Xavier

```
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:

$\text{std} = 1/\sqrt{D_{\text{in}}}$

“Just right”: Activations are nicely scaled for all layers!

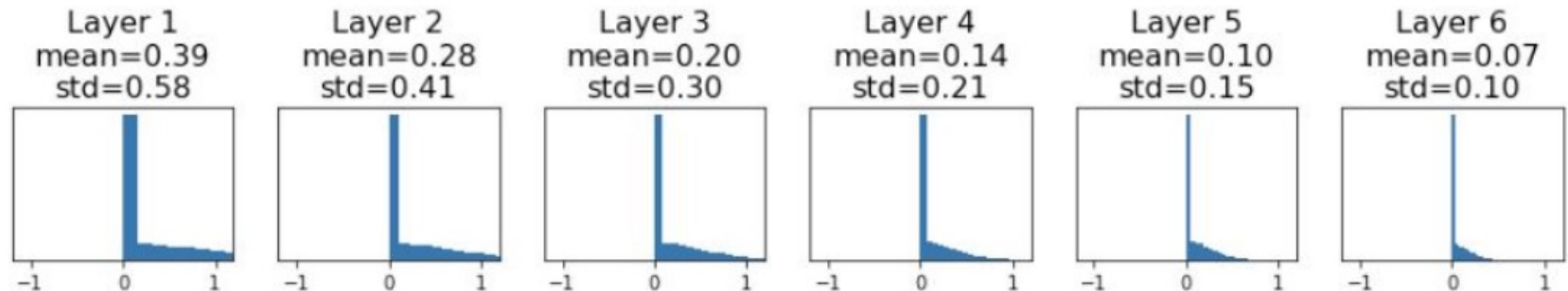


# Khởi tạo trọng số: He initialization

```
dims = [4096] * 7      Change from tanh to ReLU
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)
```

Xavier assumes zero centered activation function

Activations collapse to zero again, no learning =(



$$\text{var}(y) = \text{var}(w_1x_1 + w_2x_2 + \dots + w_{N_{in}}x_{N_{in}} + b)$$

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

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

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



# Khởi tạo trọng số: He initialization

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

$$\mathbf{y}_{l+1} = \mathbf{W}_l \mathbf{x}_l + \mathbf{b}_l$$

$$y_{l+1}^i = w_{l1}^i x_{l1} + w_{l2}^i x_{l2} + \dots + w_{lN_l}^i x_{lN_l} + b_l, i = 1, \dots, N_{l+1}$$
$$\text{var}(y_l) = N_l \times \text{var}(w_l x_l) \quad (1)$$

$$\begin{aligned} \text{var}(w_l x_l) &= (E(x_l))^2 \text{var}(w_l) + (E(w_l))^2 \text{var}(x_l) + \\ &\quad \text{var}(w_l) \text{var}(x_l) = \text{var}(w_l) \left[ \text{var}(x_l) + (E(x_l))^2 \right] \\ &= \text{var}(w_l) E(x_l^2) \end{aligned} \quad (2)$$

# Khởi tạo trọng số: He initialization

- Do  $w_l$  phân bố đối xứng quanh 0, ta có thể chứng minh  $y_l$  cũng phân bố đối xứng quanh 0:

$$\begin{aligned} & Pr(w_l x_l > 0) \\ &= Pr((w_l > 0 \ \&\& \ x_l > 0) \ || \ (w_l < 0 \ \&\& \ x_l < 0)) \\ &= Pr(w_l > 0 \ \&\& \ x_l > 0) + Pr(w_l < 0 \ \&\& \ x_l < 0) \\ &= Pr(w_l > 0) Pr(x_l > 0) + Pr(w_l < 0) Pr(x_l < 0) \\ &= \frac{1}{2} Pr(x_l > 0) + \frac{1}{2} Pr(x_l < 0) = \frac{1}{2} \end{aligned}$$

$$E(w_l x_l) = E(w_l)E(x_l) = 0$$



# Khởi tạo trọng số: He initialization

$$\begin{aligned} E(x_l^2) &= E(\max(0, y_{l-1})^2) = \frac{1}{2} E(y_{l-1}^2) \\ &= \frac{1}{2} (E(y_{l-1}^2) - E(y_{l-1})^2) = \frac{1}{2} \text{var}(y_{l-1}) \end{aligned} \quad (3)$$

Từ (1), (2) và (3) ta có:

$$\text{var}(y_l) = \frac{1}{2} N_l \text{var}(w_l) \text{var}(y_{l-1})$$

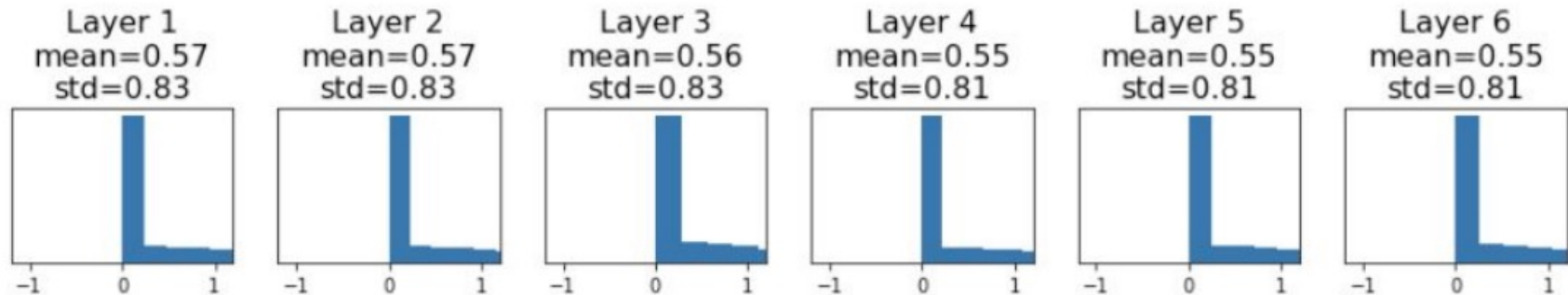
Để phân bố  $y_l$  ổn định thì:  $\text{var}(w_l) = \frac{2}{N_l}$

# Khởi tạo trọng số: He initialization

- 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)
```

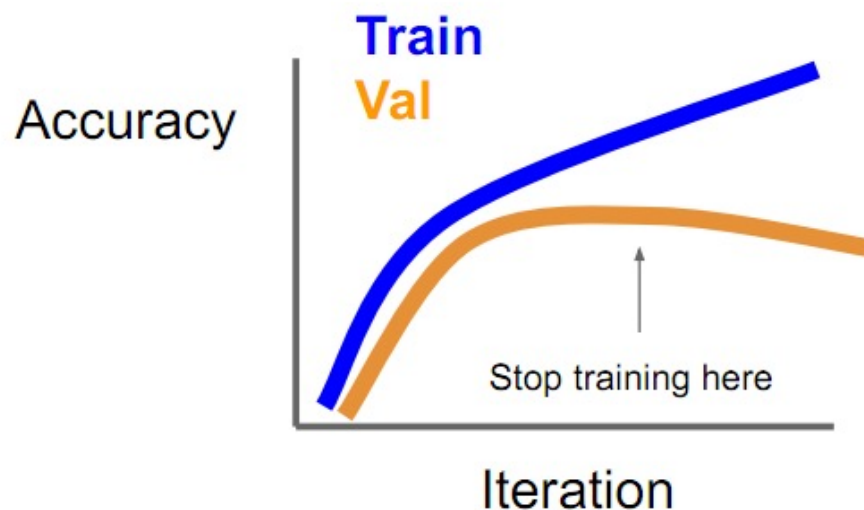
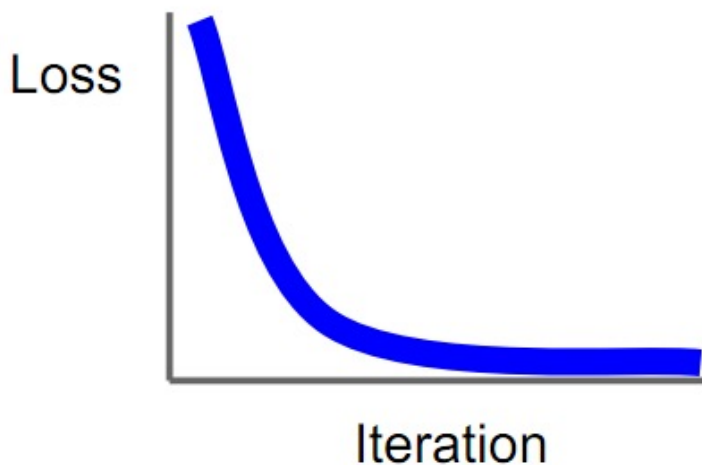
“Just right”: Activations are nicely scaled for all layers!



# Một số kỹ thuật chống overfitting

# Dừng sớm

- Dừng huấn luyện khi độ chính xác trên tập val bắt đầu giảm



# Điều khiển quá trình huấn luyện

$$L = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

Một số ràng buộc hay sử dụng:

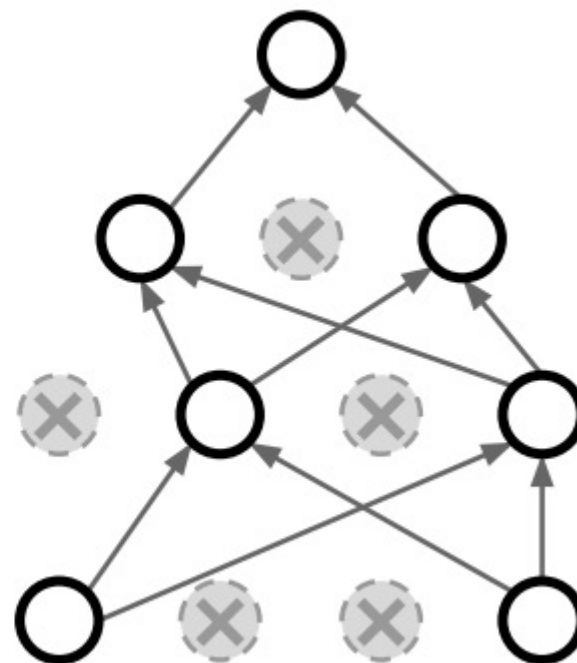
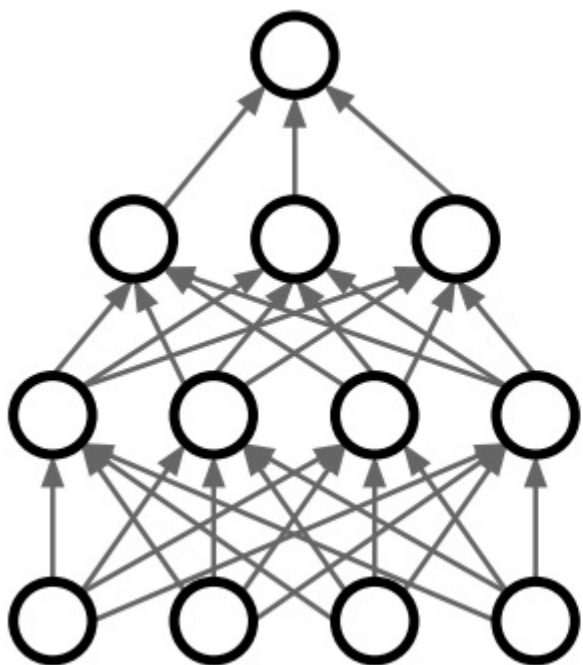
**L2 regularization**  $R(W) = \sum_k \sum_l W_{k,l}^2$  (Weight decay)

L1 regularization  $R(W) = \sum_k \sum_l |W_{k,l}|$

Elastic net (L1 + L2)  $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

# Dropout

- Trong quá trình tính toán tiến (forward pass), ngẫu nhiên thiết lập đầu ra một số nơ-ron về 0.
- Xác suất drop thường là 0.5



# Dropout

- Ví dụ quá trình tính toán tiến của một mạng nơ-ron 3 lớp sử dụng dropout

```
p = 0.5 # probability of keeping a unit active. higher = less dropout

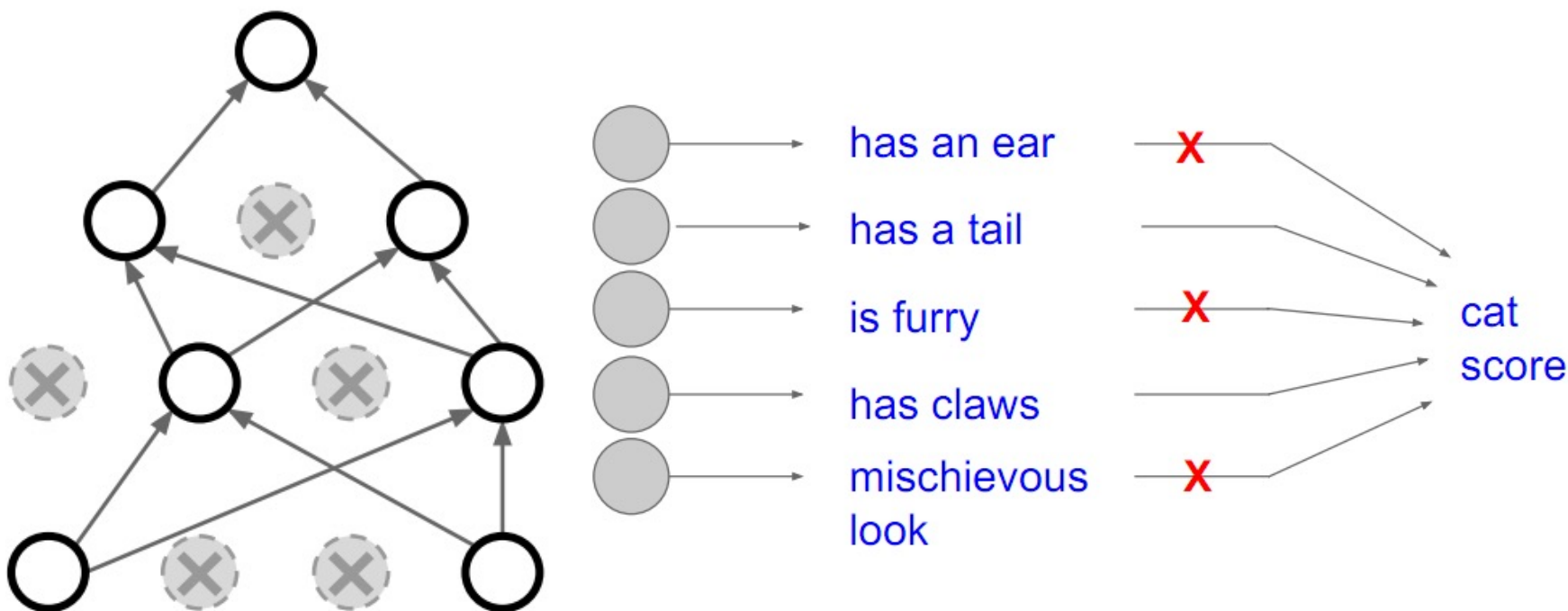
def train_step(X):
    """ X contains the data """

    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

# Tác dụng dropout

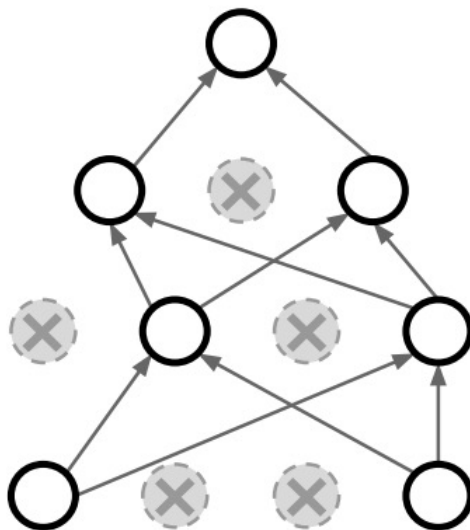
- Ép mạng nơ-ron phải học biểu diễn dư thừa (redundant representation)





# Tác dụng dropout

- Dropout khi huấn luyện có thể diễn giải như huấn luyện đồng thời nhiều mô hình khác nhau
- Mỗi kiểu drop nơ-ron tương ứng với một mô hình
- Một lớp kết nối đầy đủ với 4096 nơ-ron sẽ có  $2^{4096} \sim 10^{1233}$  phương án drop
- ... chỉ có cỡ  $10^{82}$  nguyên tử trong toàn bộ vũ trụ!



# Lúc suy diễn

- Dropout làm kết quả đầu ra ngẫu nhiên
- 

$$\boxed{y} = f_W(\boxed{x}, \boxed{z})$$

Output (label)      Input (image)      Random mask

- Cần phải lấy trung bình tất cả các kết quả

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

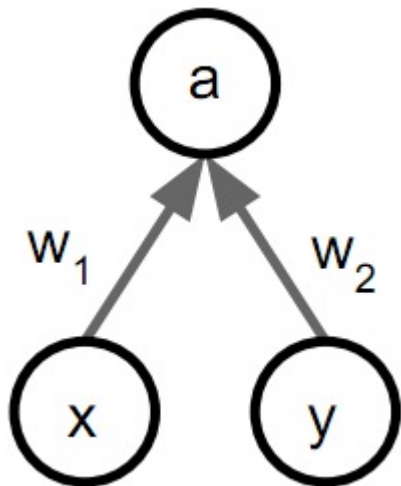
- Nhưng tích phân này là không thể...

# Lúc suy diễn

- Xấp xỉ tích phân

$$y = f(x) = E_z[f(x, z)] = \int p(z)f(x, z)dz$$

- Ví dụ xét một nơ-ron



- Lúc suy diễn:  $E[a] = w_1x + w_2y$
- Lúc huấn luyện:

$$\begin{aligned} E[a] &= \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) \\ &\quad + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) \\ &= \frac{1}{2}(w_1x + w_2y) \end{aligned}$$

# Lúc suy diễn

- Lúc suy diễn tất cả nơ-ron đều hoạt động. Vì vậy phải scale đầu ra của mỗi nơ-ron:

Đầu ra khi suy diễn = kỳ vọng đầu ra khi huấn luyện

→ Nhân với tỉ lệ **keeping rate**

```
def predict(X):  
    # ensembled forward pass  
    H1 = np.maximum(0, np.dot(W1, X) + b1) * p # NOTE: scale the activations  
    H2 = np.maximum(0, np.dot(W2, H1) + b2) * p # NOTE: scale the activations  
    out = np.dot(W3, H2) + b3
```

# Làm giàu dữ liệu

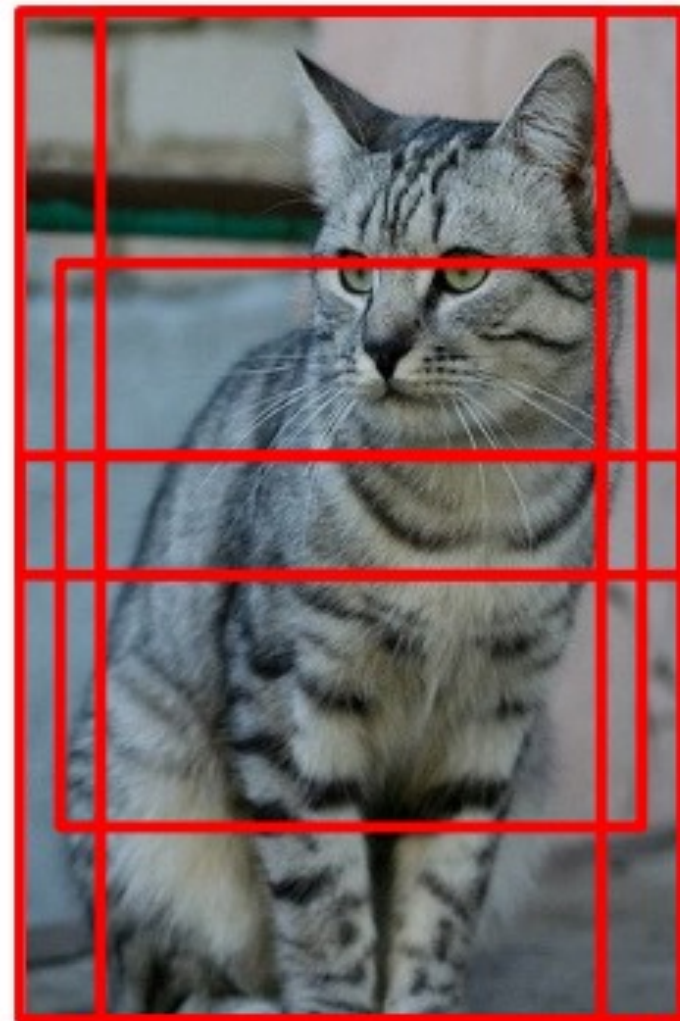
# Data Augmentation

# Flip ngang



# Crop ngẫu nhiên và scale ảnh

- Ví dụ ResNet:
  1. Chọn ngẫu nhiên  $L$  trong khoảng  $[256, 480]$
  2. Resize ảnh để chiều nhỏ nhất bằng  $L$
  3. Crop ngẫu nhiên vùng kích thước  $224 \times 224$





# Thay đổi màu sắc





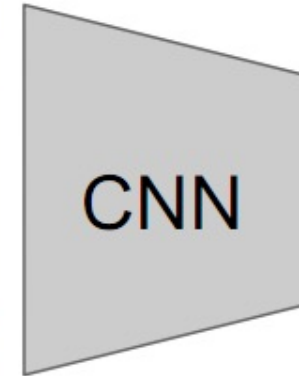
# Các phép biến đổi khác...

- Tịnh tiến
- Xoay ảnh
- stretching
- shearing
- lens distortions...

# Mixup



Randomly blend the pixels  
of pairs of training images,  
e.g. 40% cat, 60% dog



Target label:  
cat: 0.4  
dog: 0.6

# Một số thư viện

## 1. Alumentations

<https://github.com/alumentations-team/alumentations>

## 2. Imgaug

<https://github.com/aleju/imgaug>

## 3. Augmentor

<https://github.com/mdbloice/Augmentor>

# Tài liệu tham khảo

1. Bài giảng biên soạn dựa trên khóa cs231n của Stanford, bài giảng số 7-8:

<http://cs231n.stanford.edu>

2. Khởi tạo Xavier:

<https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/>



25 YEARS ANNIVERSARY  
**SOICT**

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

**Thank you  
for your  
attentions!**



[soict.hust.edu.vn/](http://soict.hust.edu.vn/)



[fb.com/groups/soict](https://fb.com/groups/soict)

