

25
SOICT

YEARS ANNIVERSARY

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

Bài 4:

Huấn luyện mạng nơ-ron

(Phần 1)

Nội dung

- Hàm kích hoạt
- Tiền xử lý dữ liệu
- Khởi tạo trọng số
- Các kỹ thuật chuẩn hóa

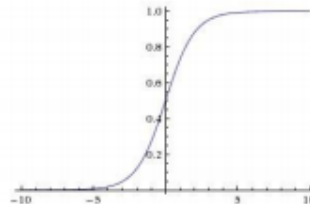
Hàm kích hoạt

Hàm kích hoạt

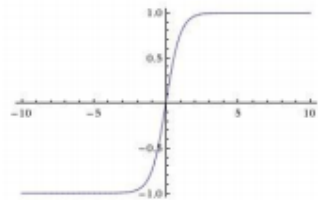
Activation Functions

Sigmoid

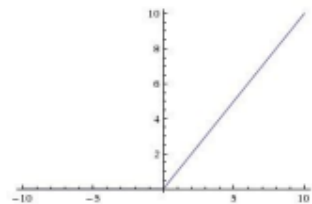
$$\sigma(x) = 1/(1 + e^{-x})$$



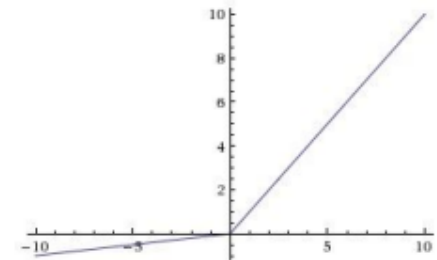
tanh tanh(x)



ReLU max(0,x)



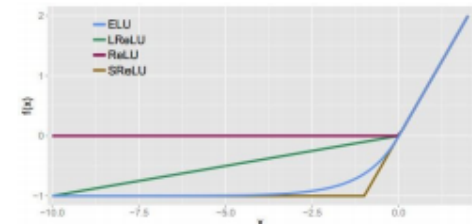
Leaky ReLU $\max(0.1x, x)$



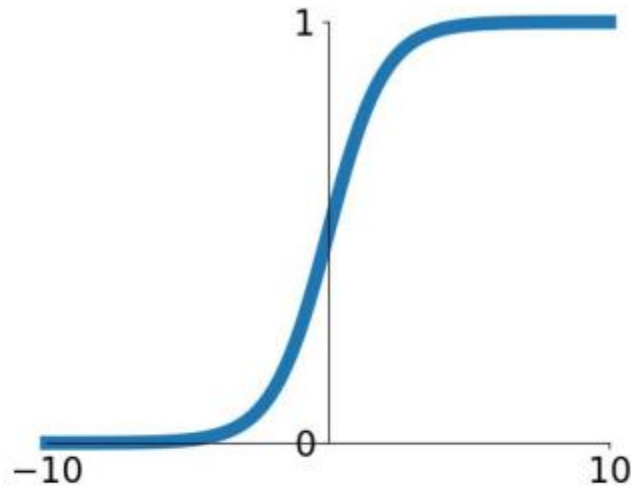
Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

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



Hàm kích hoạt

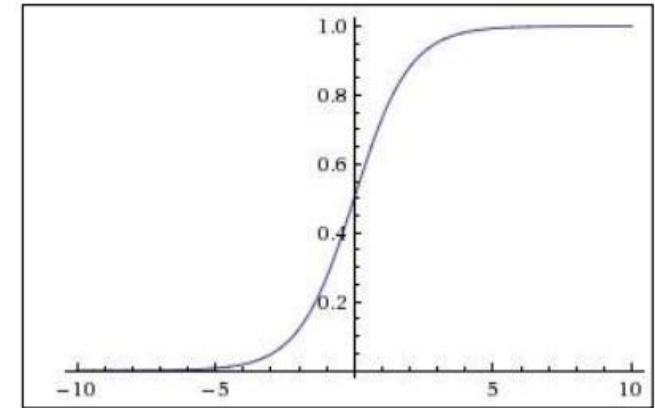
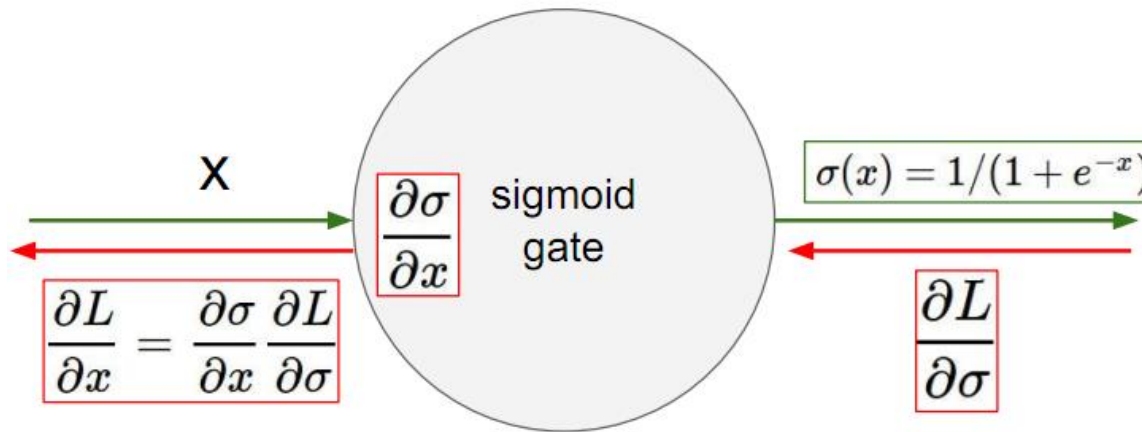


Sigmoid

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

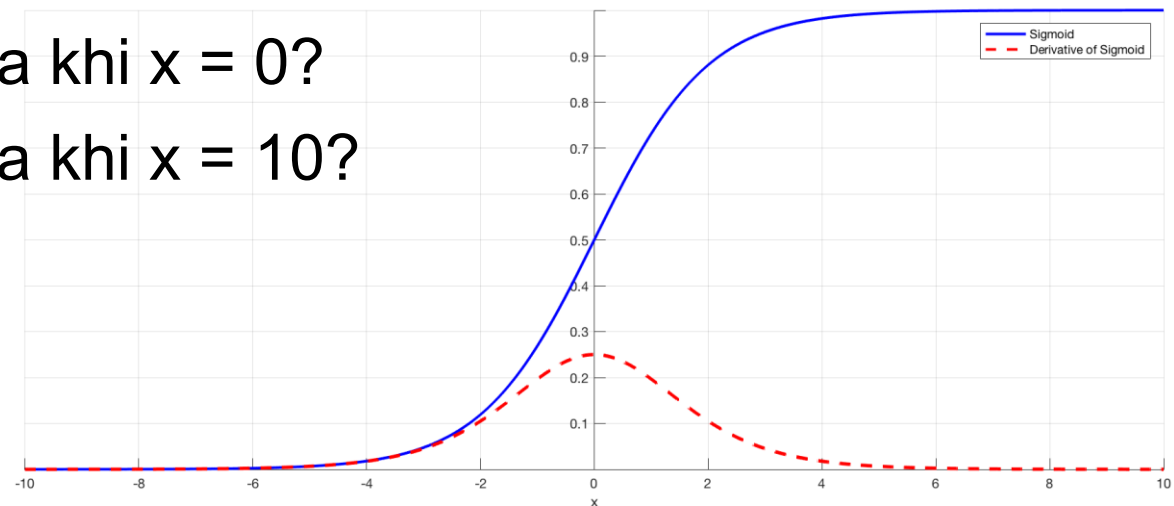
- Nhận giá trị trong khoảng $[0, 1]$
- Được dùng phổ biến trong lịch sử mạng nơ-ron do chúng mô phỏng tốt tỉ lệ bắn xung (firing rate) của nơ-ron
- Có 3 nhược điểm:
 - Nơ-ron bão hòa triệt tiêu gradient

Hàm kích hoạt

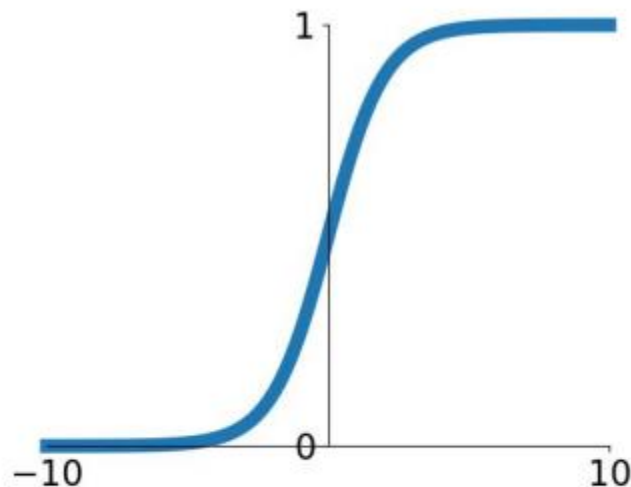


- Điều gì sẽ xảy ra khi $x = -10$?
- Điều gì sẽ xảy ra khi $x = 0$?
- Điều gì sẽ xảy ra khi $x = 10$?

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



Hàm kích hoạt



Sigmoid

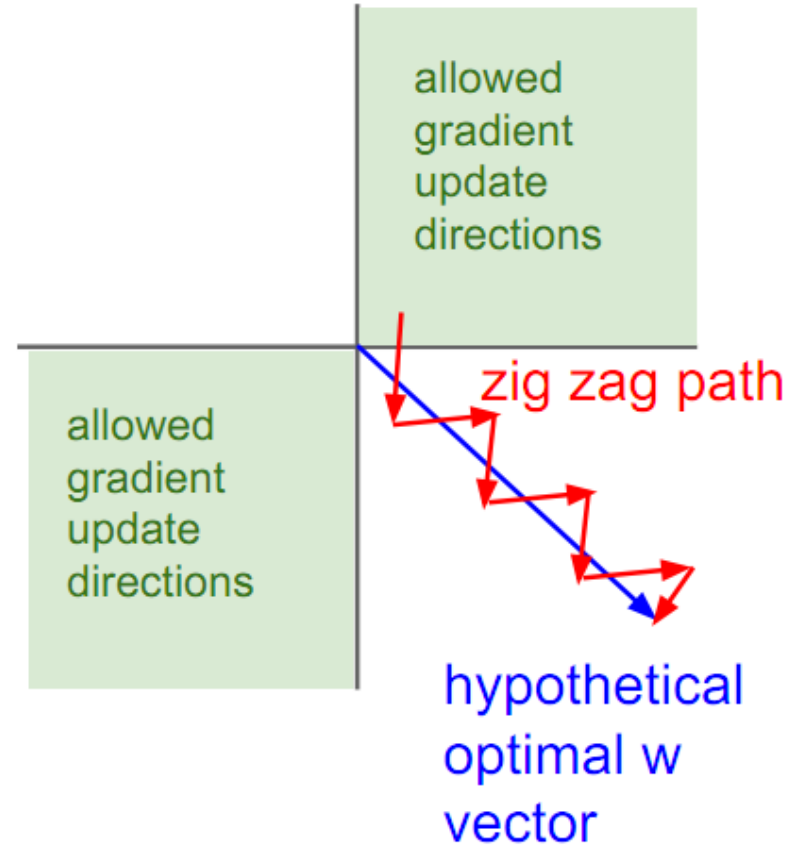
$$\sigma(x) = 1 / (1 + e^{-x})$$

- 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

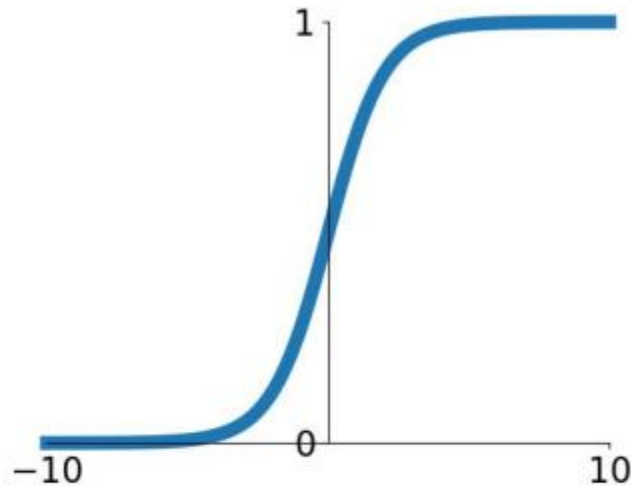
Hàm kích hoạt

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

- 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

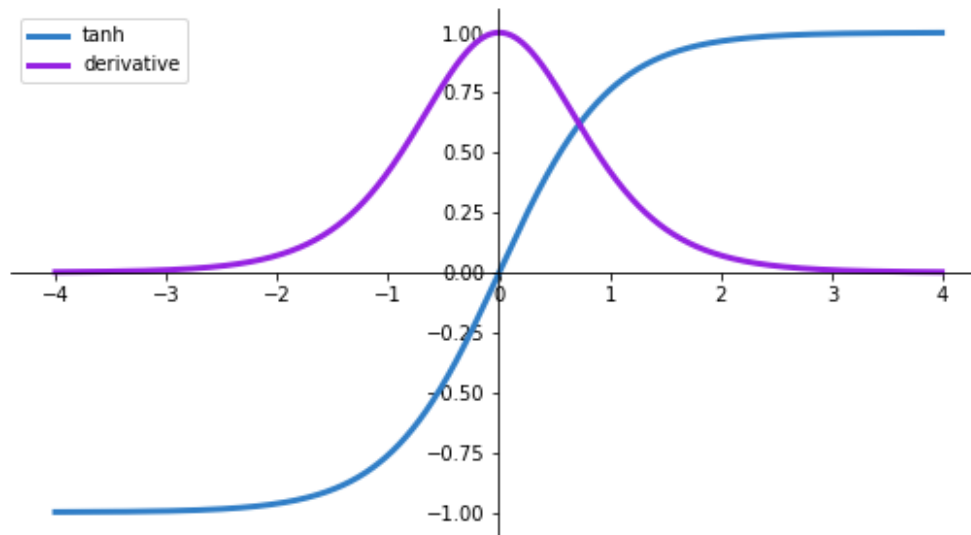
$$\sigma(x) = 1 / (1 + e^{-x})$$

- 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

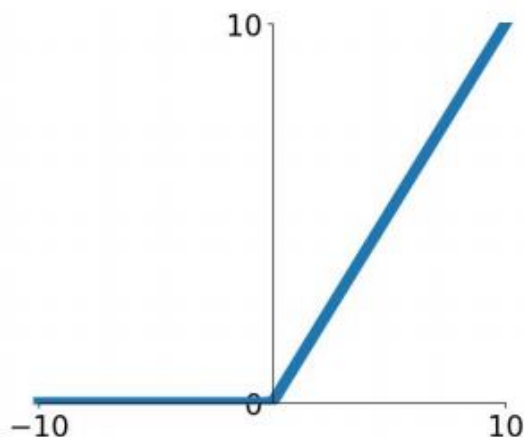
Hàm kích hoạt

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

- 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

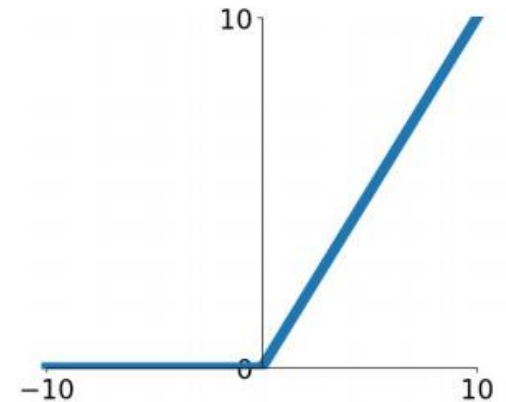
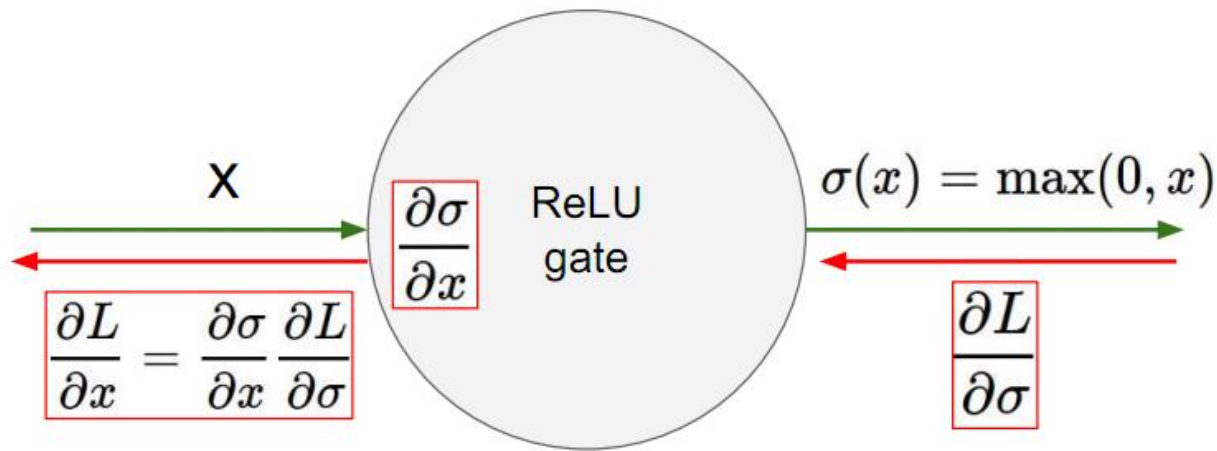


ReLU
(Rectified Linear Unit)

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

- 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...

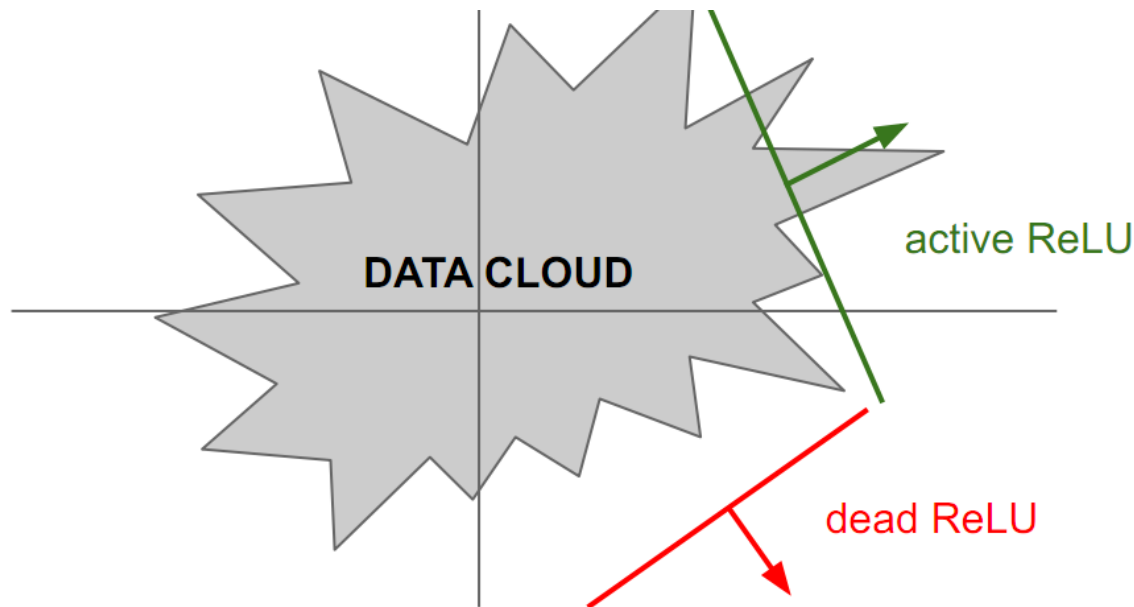
Hàm kích hoạt



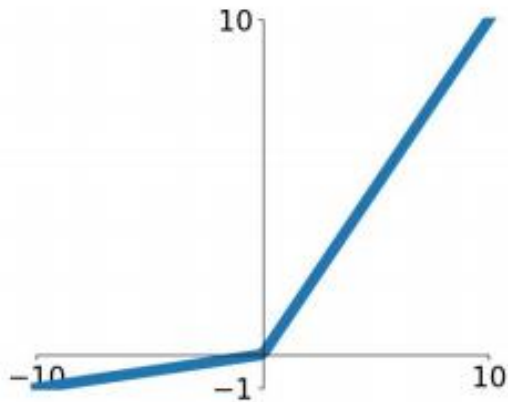
- Điều gì sẽ xảy ra khi $x = -10$?
- Điều gì sẽ xảy ra khi $x = 0$?
- Điều gì sẽ xảy ra khi $x = 10$?

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



Hàm kích hoạt

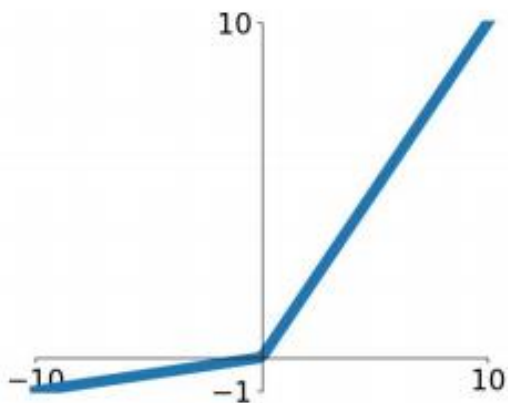


Leaky ReLU

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

- 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

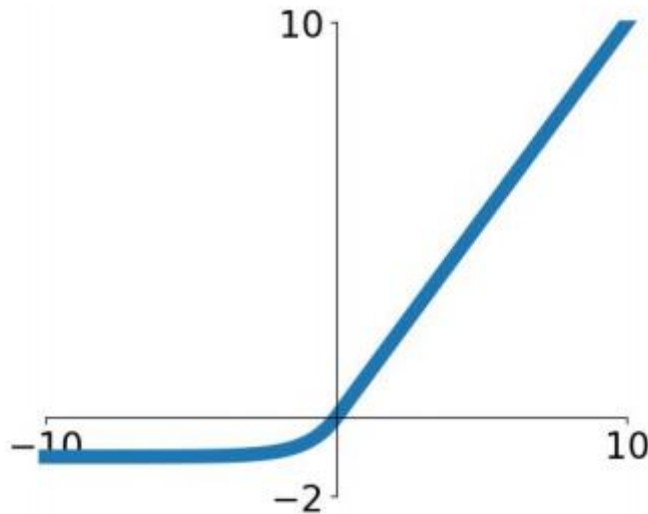
$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

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

backprop into α
(parameter)

Hàm kích hoạt ELU



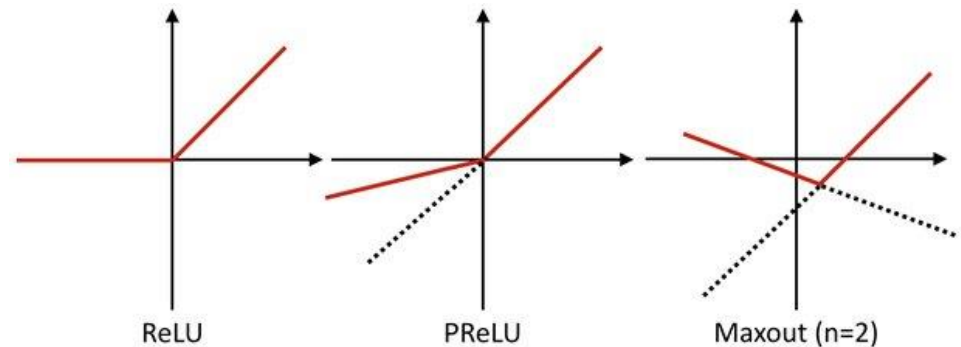
- Có tất cả ưu điểm của ReLU
- Trung bình đầu ra gần 0 hơn
- Không “chết”
- Tính toán lâu do có hàm $\exp()$

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

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

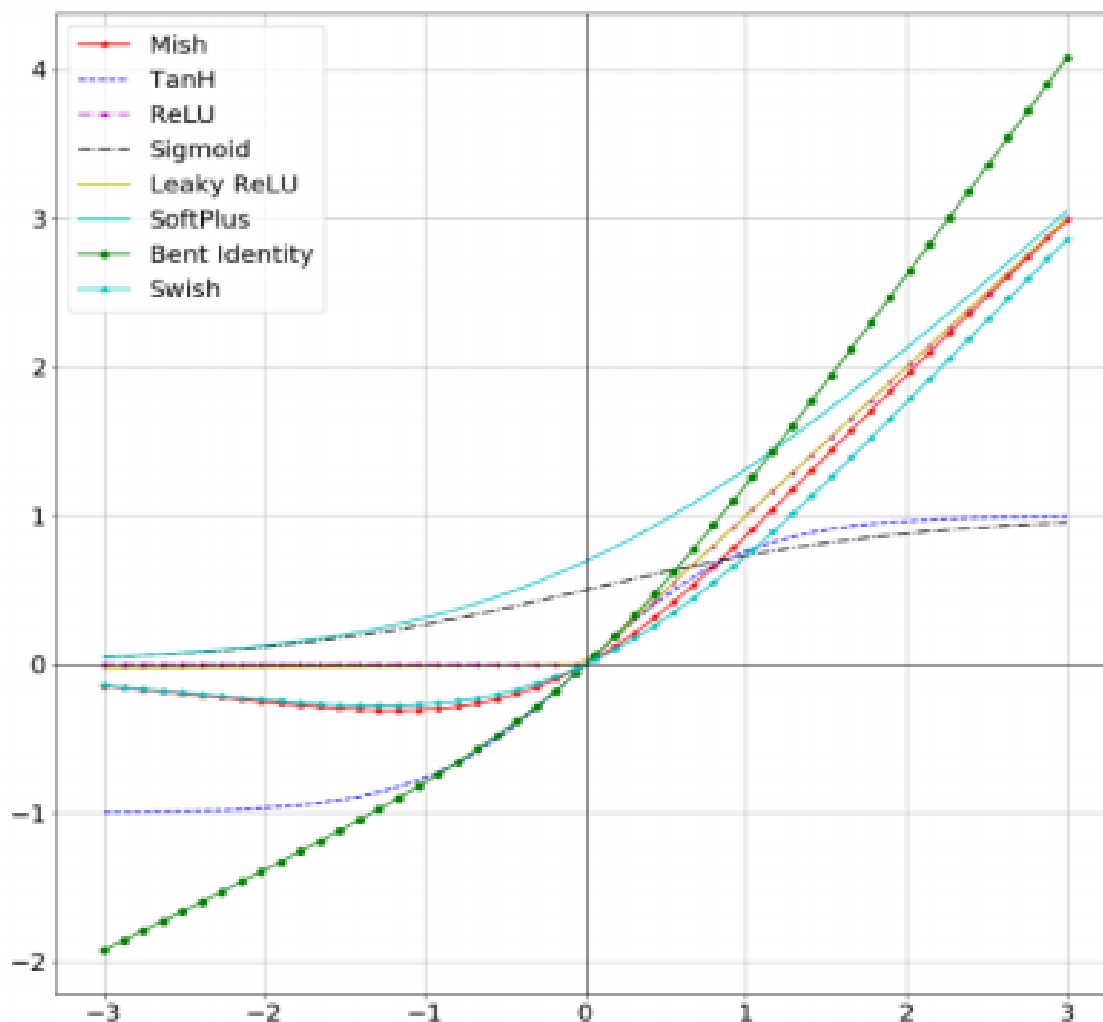
- **Trong thực tế:**

- 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

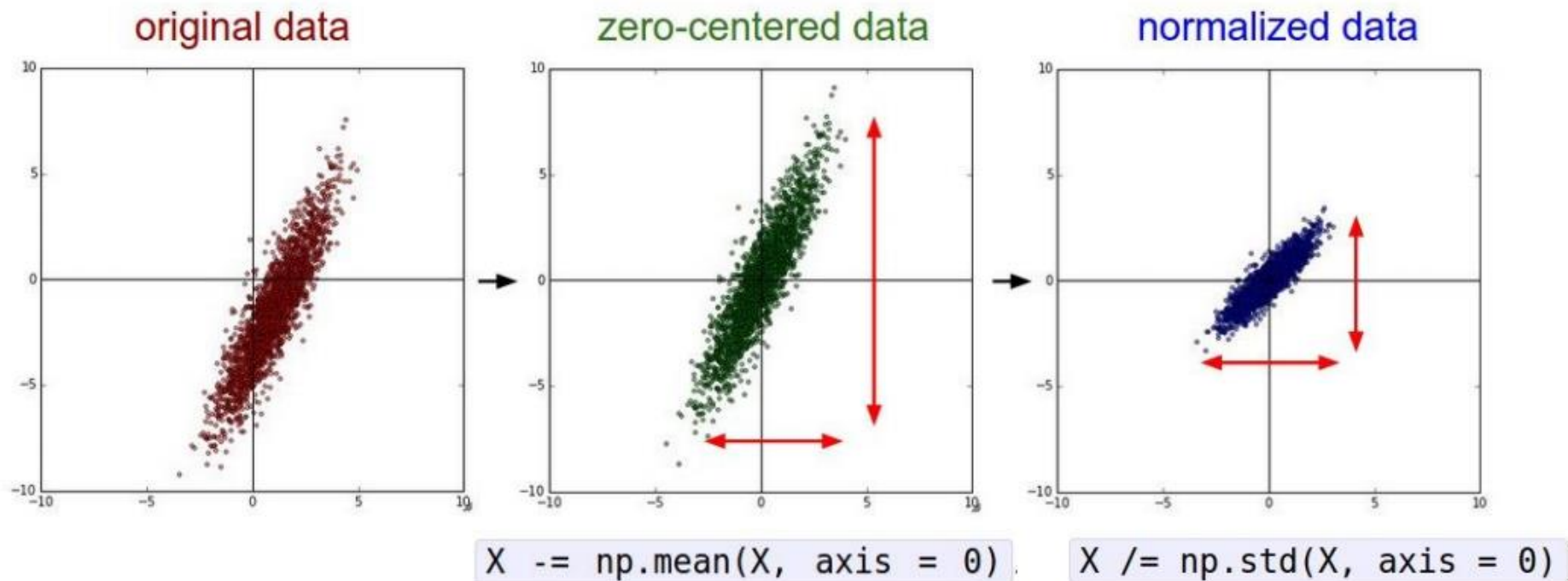
Hàm kích hoạt



Tiền xử lý dữ liệu

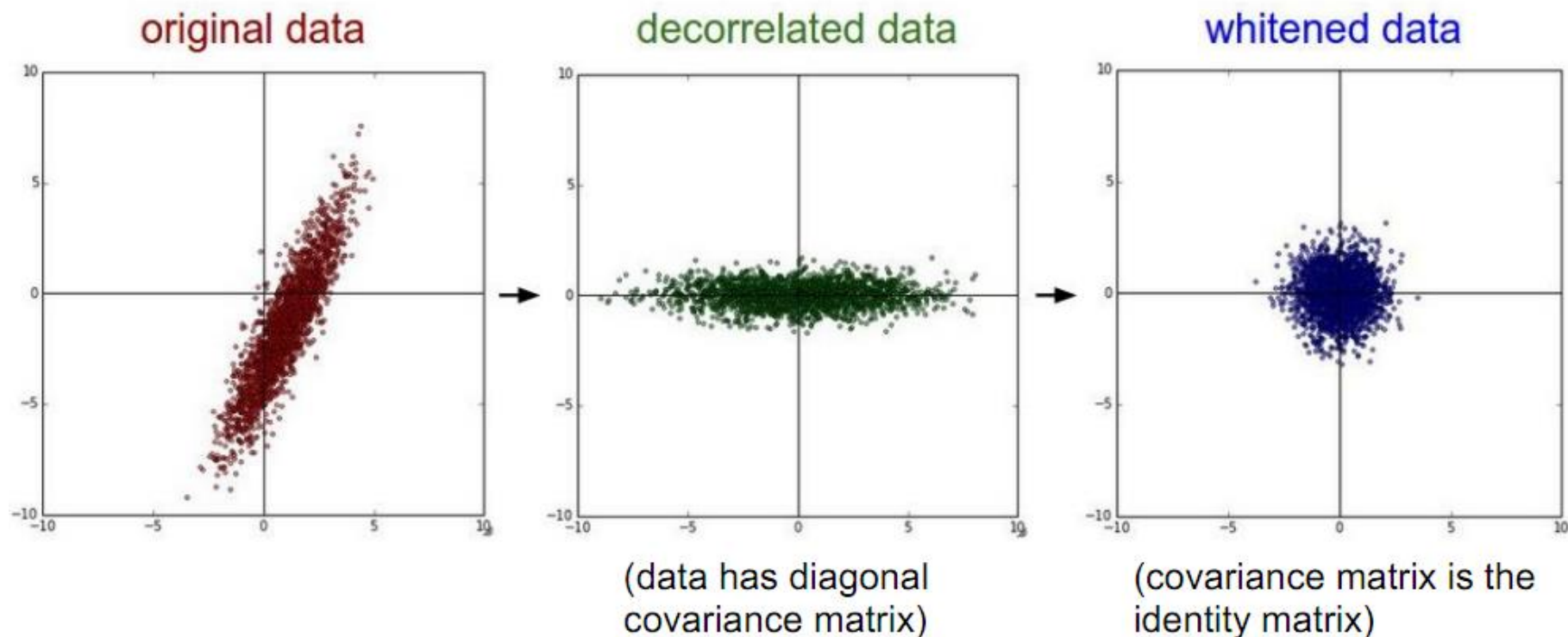
Tiền xử lý dữ liệu

- 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



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

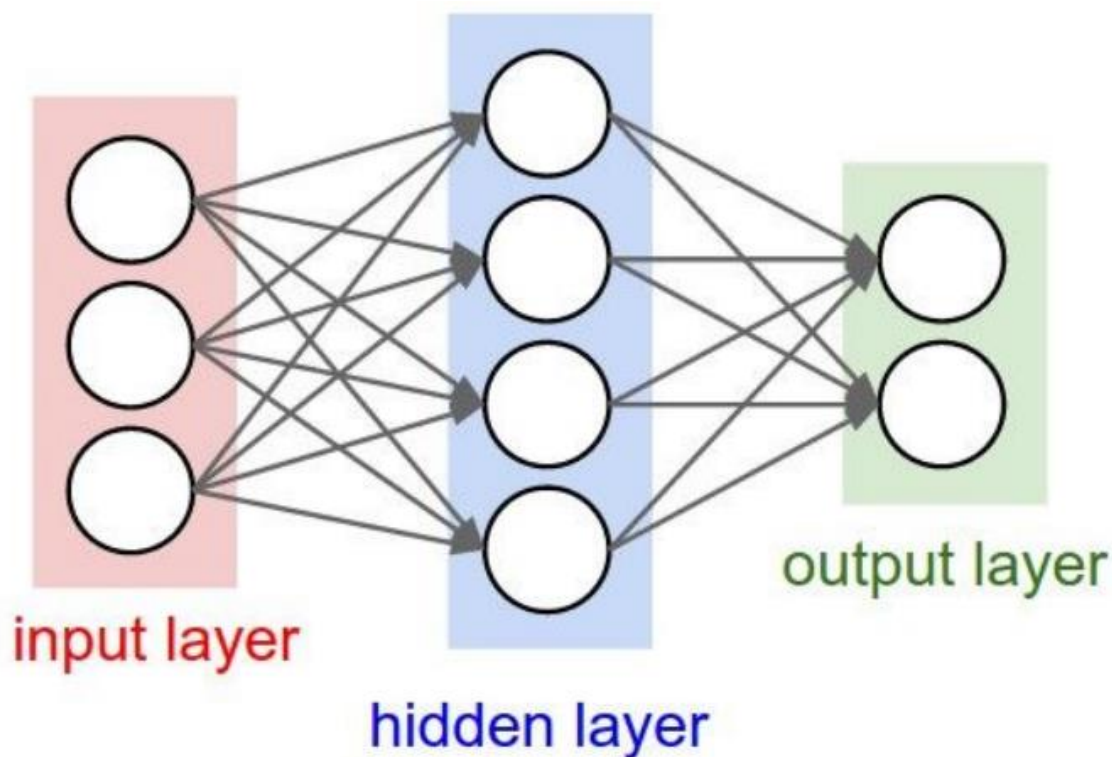
Demo

- <https://github.com/trungtv/Preprocessing-for-deep-learning>

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

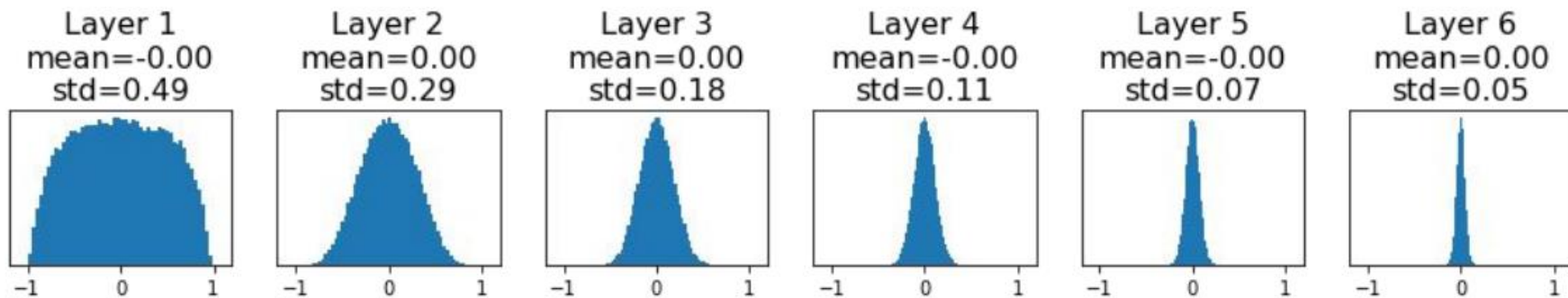
```

dims = [4096] * 7      Forward pass for a 6-layer
hs = []               net with hidden size 4096
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 $dL/dW = 0$

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

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

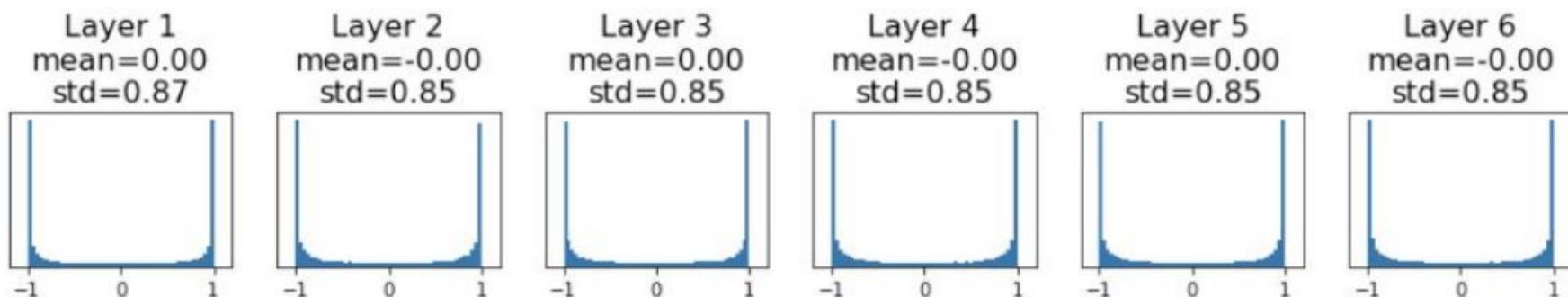
Khởi tạo trọng số

```

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: Khởi tạo 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:

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

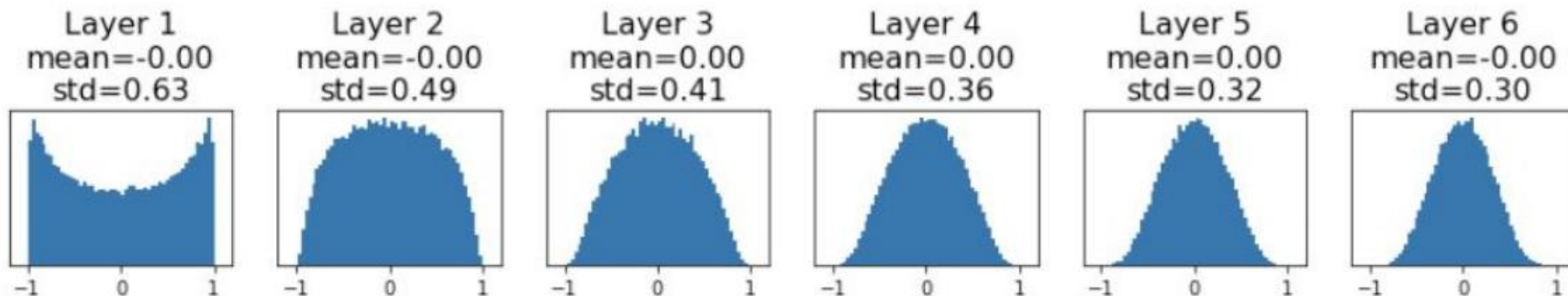
Khởi tạo trọng số

- 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 = $1/\sqrt{D_{in}}$

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

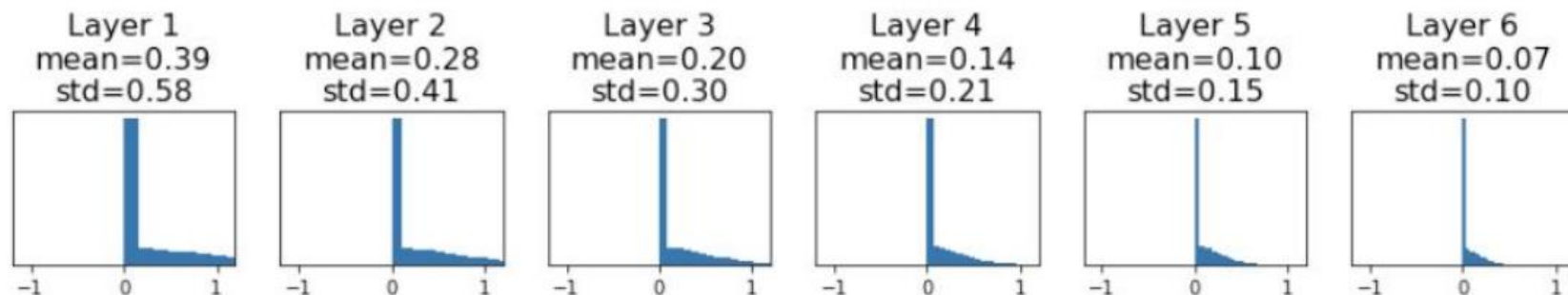


Khởi tạo trọng số

```
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 =(



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

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

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

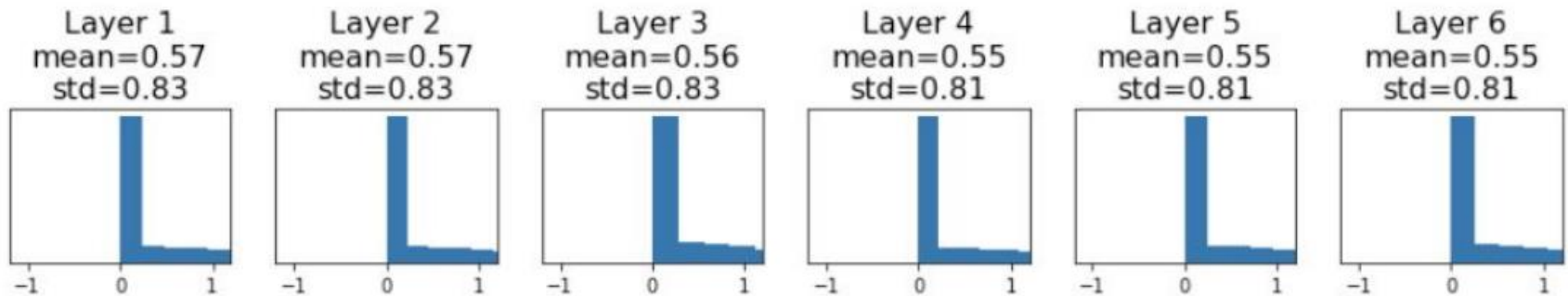
$$\text{var}(w_i) = 2 / N_{in}$$

Khởi tạo trọng số

- He / 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!



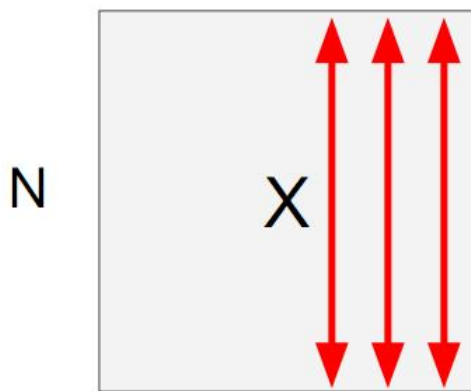
Các kỹ thuật chuẩn hóa

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!

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

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

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

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.

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

Input: $x : N \times D$

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

Learnable scale and shift parameters:

$$\gamma, \beta : D$$

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

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

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

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

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

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

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: $x : N \times D$

$\mu_j =$ (Running) average of values seen during training

Per-channel mean, shape is D

Learnable scale and shift parameters:

$\gamma, \beta : D$

$\sigma_j^2 =$ (Running) average of values seen during training

Per-channel var, shape is D

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

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

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

$$\mathbf{x}: \mathbf{N} \times \mathbf{D}$$

Normalize



$$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times \mathbf{D}$$

$$\boldsymbol{\gamma}, \boldsymbol{\beta}: 1 \times \mathbf{D}$$

$$\mathbf{y} = \boldsymbol{\gamma}(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta}$$

Batch Normalization for
convolutional networks
(Spatial Batchnorm, BatchNorm2D)

$$\mathbf{x}: \mathbf{N} \times \mathbf{C} \times \mathbf{H} \times \mathbf{W}$$

Normalize

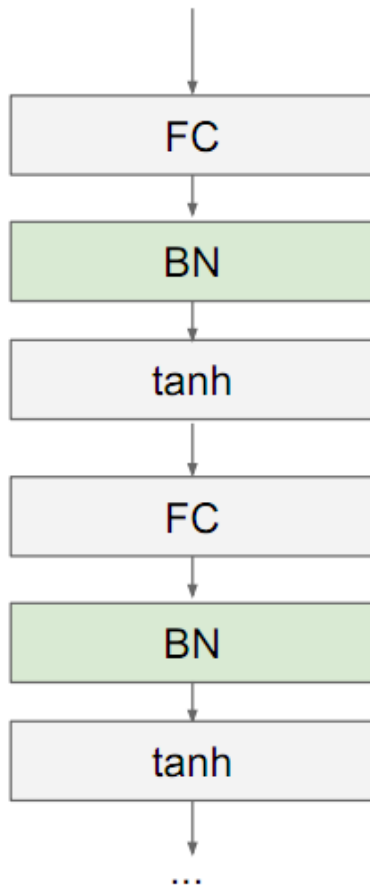


$$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times \mathbf{C} \times 1 \times 1$$

$$\boldsymbol{\gamma}, \boldsymbol{\beta}: 1 \times \mathbf{C} \times 1 \times 1$$

$$\mathbf{y} = \boldsymbol{\gamma}(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta}$$

Batch Normalization



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

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

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 hiệu chỉnh 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

$$\mathbf{x}: \mathbf{N} \times \mathbf{D}$$

Normalize



$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \mathbf{1} \times \mathbf{D}$$

$$\boldsymbol{\gamma}, \boldsymbol{\beta}: \mathbf{1} \times \mathbf{D}$$

$$\mathbf{y} = \boldsymbol{\gamma}(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta}$$

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

$$\mathbf{x}: \mathbf{N} \times \mathbf{D}$$

Normalize



$$\boldsymbol{\mu}, \boldsymbol{\sigma}: \mathbf{N} \times \mathbf{1}$$

$$\boldsymbol{\gamma}, \boldsymbol{\beta}: \mathbf{1} \times \mathbf{D}$$

$$\mathbf{y} = \boldsymbol{\gamma}(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta}$$

Chuẩn hóa theo mẫu

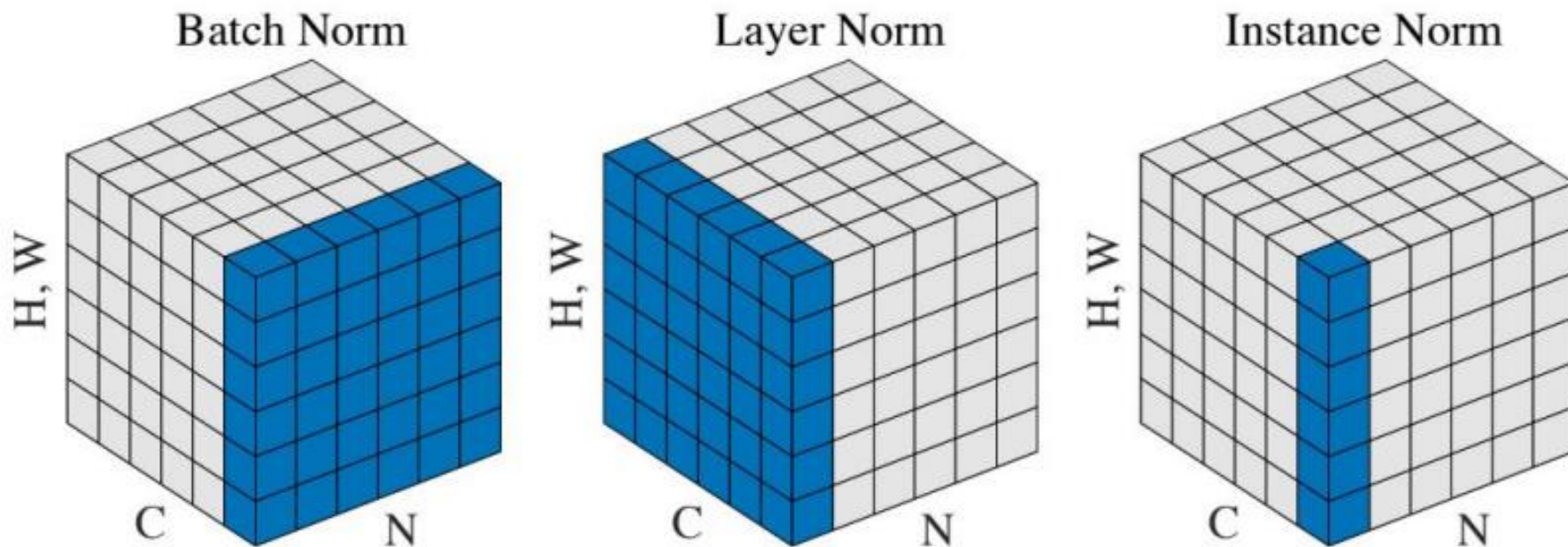
Batch Normalization for
convolutional networks

$$\begin{array}{l} \mathbf{x}: N \times C \times H \times W \\ \text{Normalize} \quad \downarrow \quad \downarrow \quad \downarrow \\ \boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times C \times 1 \times 1 \\ \boldsymbol{\gamma}, \boldsymbol{\beta}: 1 \times C \times 1 \times 1 \\ \mathbf{y} = \boldsymbol{\gamma}(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta} \end{array}$$

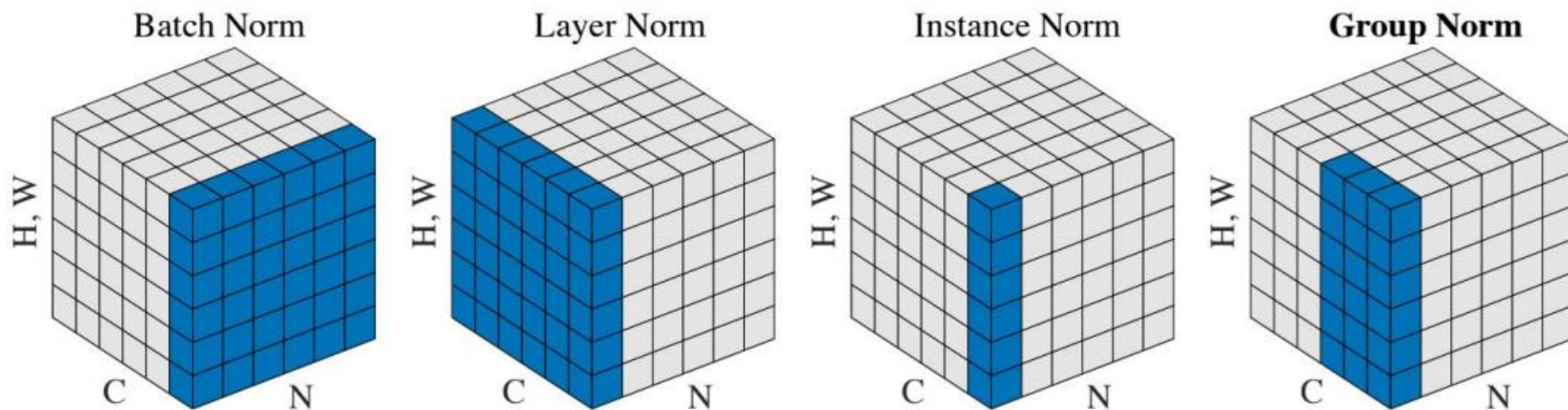
Instance Normalization for
convolutional networks
Same behavior at train / test!

$$\begin{array}{l} \mathbf{x}: N \times C \times H \times W \\ \text{Normalize} \quad \quad \quad \downarrow \quad \downarrow \\ \boldsymbol{\mu}, \boldsymbol{\sigma}: N \times C \times 1 \times 1 \\ \boldsymbol{\gamma}, \boldsymbol{\beta}: 1 \times C \times 1 \times 1 \\ \mathbf{y} = \boldsymbol{\gamma}(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \boldsymbol{\beta} \end{array}$$

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/>



25 YEARS ANNIVERSARY
SOICT

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

Chân thành
cảm ơn!!!



soict.hust.edu.vn/



fb.com/groups/soict

