
Neural Network from Scratch

Hãy bắt đầu từ những điều đơn giản nhất

cuong@techmaster.vn

Có nhất thiết phải code lại mạng neural network?



Lợi ích khi tự code thuật toán Deep Learning

- Căn bản vững chắc sẽ dễ hiểu các khái niệm phức tạp
- Hiểu để tuning các hyper parameters tốt hơn
- Viết công thức toán học nhìn có vẻ nguy hiểm hơn (thực ra là cần đối với sinh viên sẽ học Master và PhD)

Mô hình từ cấu trúc sinh học bộ não con người

Thiết kế phỏng sinh học



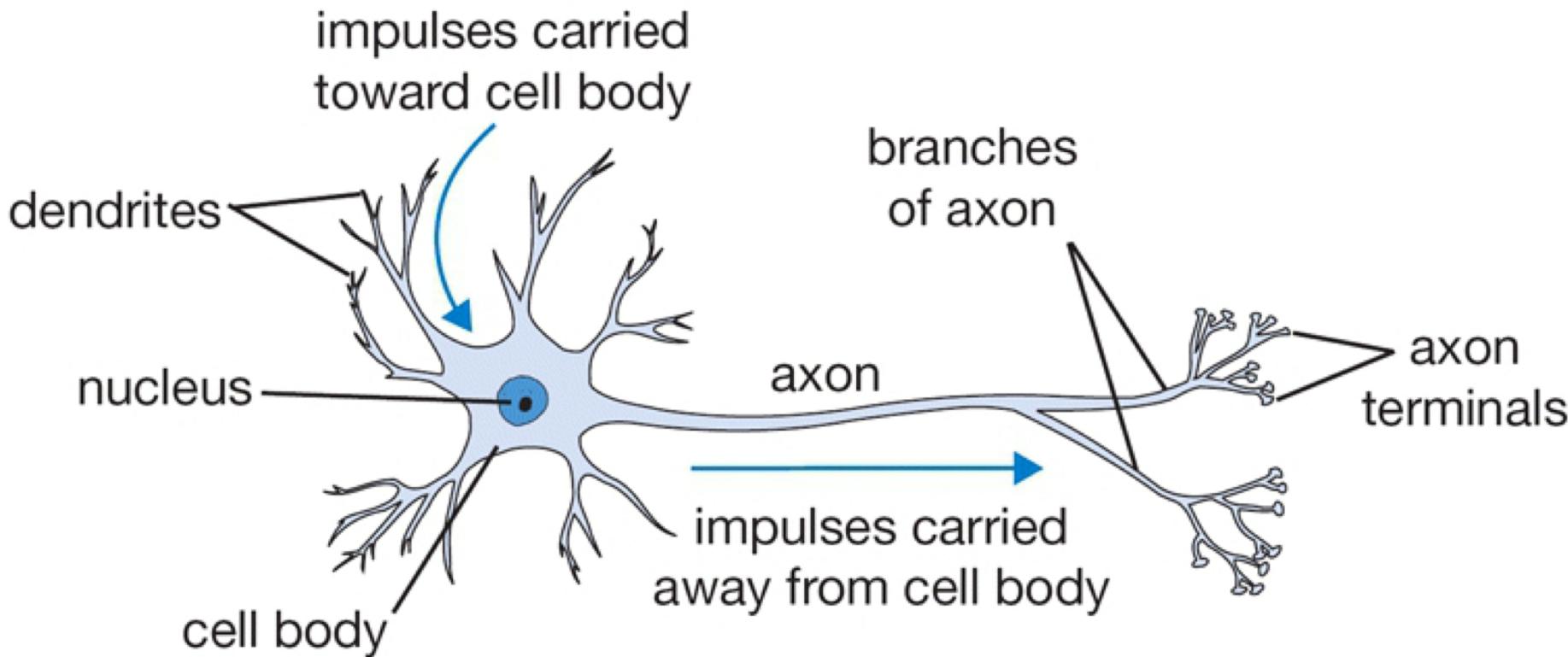


Chim bói cá



Mũi tàu cao tốc Nhật Bản phỏng theo
mũi chim bói cá để giảm tiếng ồn âm
thanh

Mô hình tế bào thần kinh trong bài toán mô phỏng tín hiệu, machine learning



Tế bào thần kinh

Đại số tuyến tính

Trước khi đi xa hãy chuẩn bị tốt hành lý !



Toán trong neural network

- Đạo hàm
 - http://tutorial.math.lamar.edu/Extras/CheatSheets_Tables.aspx
- Vi phân riêng phần
 - <https://www.youtube.com/watch?v=ly4S0oi3Yz8>
- Cách lấy vi phân riêng phần đối với vector
 - <http://cs231n.stanford.edu/vecDerivs.pdf>

Tính chất đạo hàm

If $f(x)$ and $g(x)$ are differentiable functions (the derivative exists), c and n are any real numbers,

$$1. \ (c f)' = c f'(x)$$

$$2. \ (f \pm g)' = f'(x) \pm g'(x)$$

$$3. \ (f g)' = f' g + f g' - \textbf{Product Rule}$$

$$4. \ \left(\frac{f}{g} \right)' = \frac{f' g - f g'}{g^2} - \textbf{Quotient Rule}$$

$$5. \ \frac{d}{dx}(c) = 0$$

$$6. \ \frac{d}{dx}(x^n) = n x^{n-1} - \textbf{Power Rule}$$

$$7. \ \frac{d}{dx}(f(g(x))) = f'(g(x))g'(x)$$

This is the **Chain Rule**

Một số đạo hàm thường gặp

$$\frac{d}{dx}(x) = 1$$

$$\frac{d}{dx}(\sin x) = \cos x$$

$$\frac{d}{dx}(\cos x) = -\sin x$$

$$\frac{d}{dx}(\tan x) = \sec^2 x$$

$$\frac{d}{dx}(\sec x) = \sec x \tan x$$

$$\frac{d}{dx}(\csc x) = -\csc x \cot x$$

$$\frac{d}{dx}(\cot x) = -\csc^2 x$$

$$\frac{d}{dx}(\sin^{-1} x) = \frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}(\cos^{-1} x) = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}(\tan^{-1} x) = \frac{1}{1+x^2}$$

$$\frac{d}{dx}(a^x) = a^x \ln(a)$$

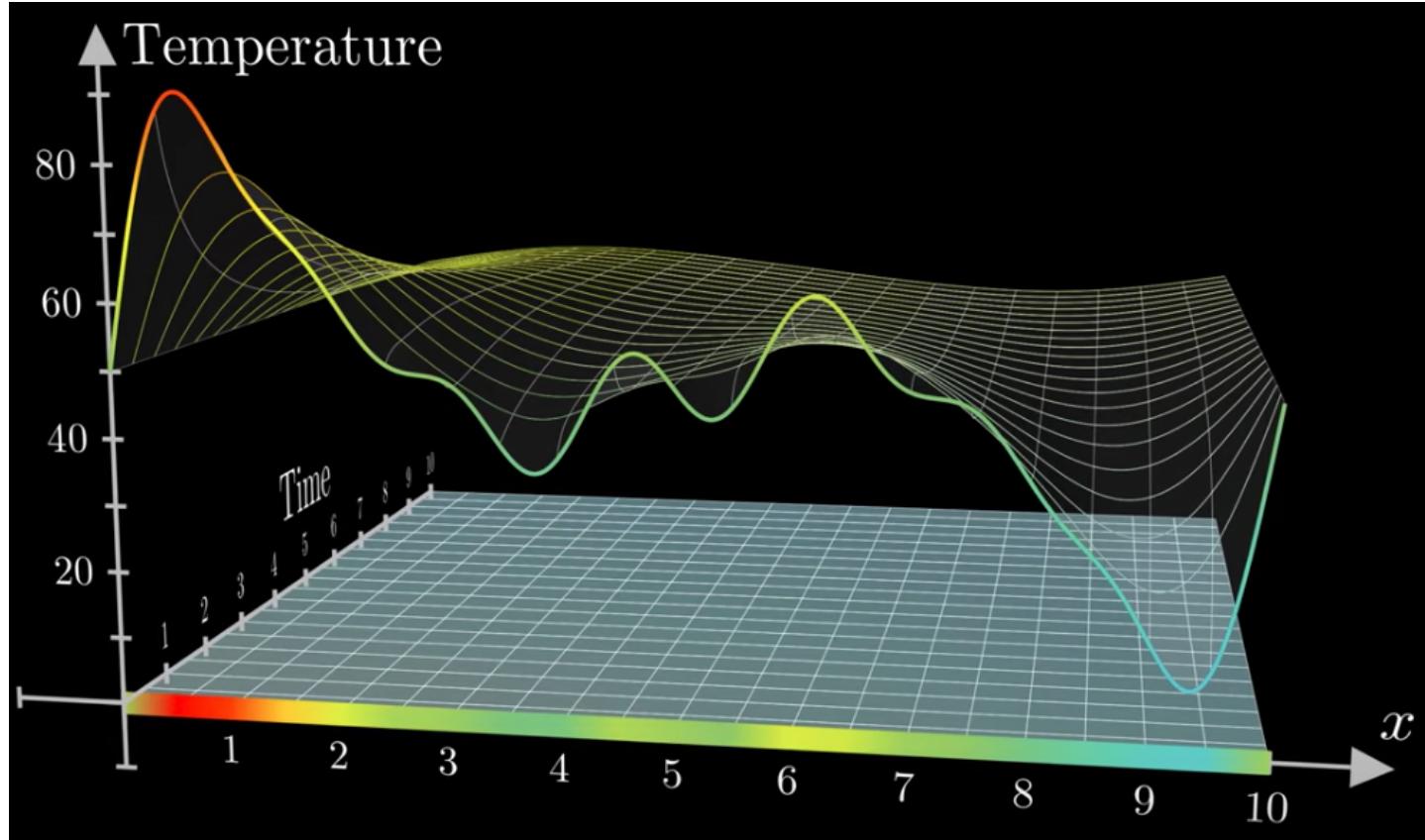
$$\frac{d}{dx}(\mathbf{e}^x) = \mathbf{e}^x$$

$$\frac{d}{dx}(\ln(x)) = \frac{1}{x}, \quad x > 0$$

$$\frac{d}{dx}(\ln|x|) = \frac{1}{x}, \quad x \neq 0$$

$$\frac{d}{dx}(\log_a(x)) = \frac{1}{x \ln a}, \quad x > 0$$

Ví dụ đạo hàm biến thiên nhiệt độ theo thời gian và chiều dài
<https://www.youtube.com/watch?v=ly4S0oi3Yz8>



Giả thiết ban đầu

Vector cột y độ dài C là tích của ma trận W có C hàng và D cột với vector cột x độ dài D

$$\vec{y} = W\vec{x}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Partial Derivative – vi phân riêng phần

Cần tính vi phân riêng phần từng thành phần \vec{y} đối với từng thành phần \vec{x}

$$\frac{\partial y_1}{\partial x_1}$$

$$\frac{\partial y_2}{\partial x_1}$$

$$y_2 = \sum_{j=1}^D w_{2j} x_j$$

$$\frac{\partial y_1}{\partial x_2}$$

$$\frac{\partial y_2}{\partial x_2}$$

$$y_2 = w_{21} x_1 + w_{22} x_2 + w_{23} x_3$$

$$\frac{\partial y_1}{\partial x_3}$$

$$\frac{\partial y_2}{\partial x_3}$$

Vi phân riêng phần từng thành phần y đối với từng thành phần x

$$\frac{\partial y_2}{\partial x_3} = \frac{\partial}{\partial x_3} [w_{21}x_1 + w_{22}x_2 + w_{23}x_3]$$

Coi như không đổi

$$\frac{\partial y_2}{\partial x_3} = 0 + 0 + \frac{\partial}{\partial x_3} [w_{23}x_3]$$

w₂₃

$$\frac{\partial y_2}{\partial x_3} =$$

$$\frac{\partial y_i}{\partial x_j} = w_{ij}$$

Công thức tổng quát

Tính vi phân riêng phần vector \vec{y} đối với vector \vec{x}

$$\vec{y} = W\vec{x}$$

$$\frac{\partial \vec{y}}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \frac{\partial y_1}{\partial x_3} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \frac{\partial y_2}{\partial x_3} \\ \frac{\partial y_3}{\partial x_1} & \frac{\partial y_3}{\partial x_2} & \frac{\partial y_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = W$$

Jacobian Matrix

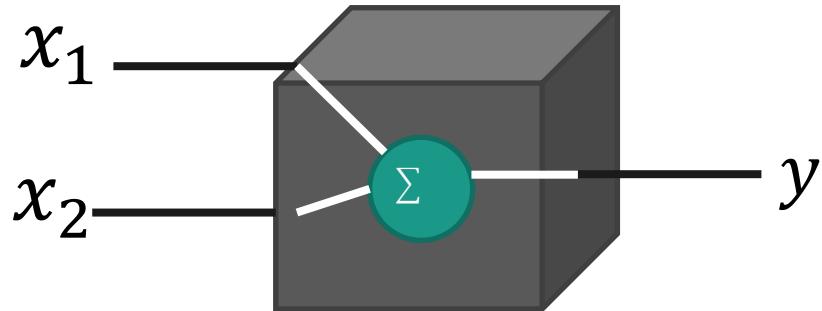
chỉ là tên gọi thôi, đừng bận tâm quá

Perceptron

Perceptron mô phỏng một tế bào thần kinh. Trong phần này chúng ta chỉ cần dùng duy nhất một perceptron để giả lập được hàm logic AND, OR và XOR

Nhiệm vụ

Hãy xây dựng một máy nhận 2 tín hiệu đầu vào và xuất tín hiệu đầu ra như mong muốn



AND

| x_1 | x_2 | Y Output |
|-------|-------|----------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

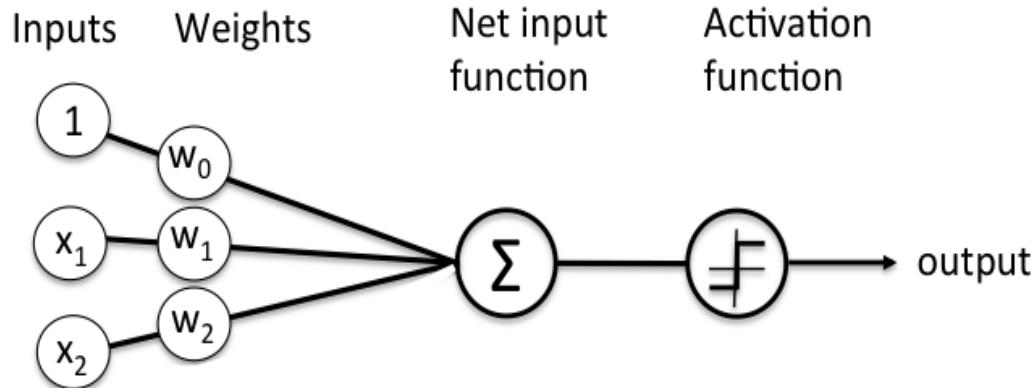
OR

| x_1 | x_2 | Y Output |
|-------|-------|----------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

XOR

| x_1 | x_2 | Y Output |
|-------|-------|----------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Mô phỏng logic AND



$$output = Activation(w_0 + x_1w_1 + x_2w_2)$$

$$\hat{y} = \sigma(w_0 + x_1w_1 + x_2w_2)$$

\hat{y} gọi là output hypothesis

Hãy tìm w_0, w_1, w_2 để output mô phỏng đúng hàm logic AND!

| X1 | X2 | Output |
|----|----|--------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

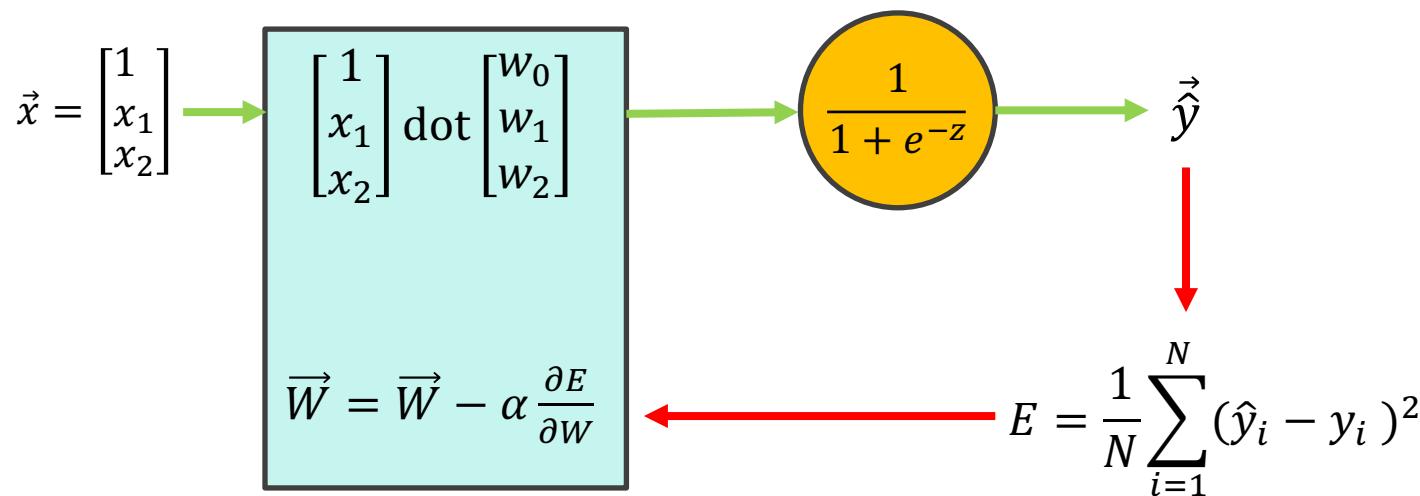
Một số quy ước

1. Tập dữ liệu gồm mảng N cặp input, output (\vec{x}_i, \vec{y}_i) . \vec{x}_i là input, \vec{y}_i là output. Gọi tập này là $X = \{(\vec{x}_1, \vec{y}_1), \dots, (\vec{x}_N, \vec{y}_N)\}$. Tập dữ liệu huấn luyện
2. Θ là tập các tham số w_{ij} cần được điều chỉnh lại sau mỗi lần huấn luyện.
 - o w_{ij} nối node i vào node j
 - o w_{0j} là bias tạo chênh lệch
3. Error function, $E(X, \theta)$ là sai khác giữa đầu ra mong đợi \vec{y}_i so với $\hat{\vec{y}}_i$

Huấn luyện 1 perceptron

Là một quá trình lặp lại 2 bước qua nhiều epochs

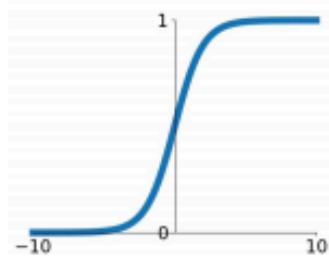
- Feed Forward → để từ đầu vào \vec{x} đi qua perceptrons tính ra \hat{y} output hypothesis
- Back propagation ← để tính hàm sai lệch, loss function → vi phân từng phần của loss function với từng



Activation Function

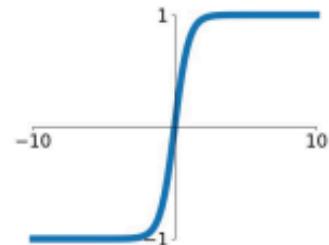
Sigmoid

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



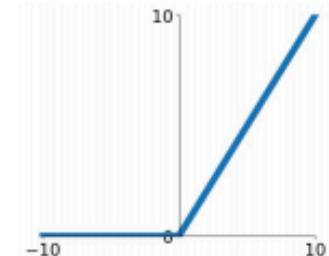
tanh

$$\tanh(x)$$



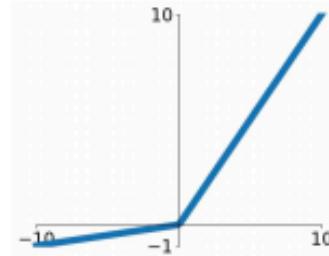
ReLU

$$\max(0, x)$$



Leaky ReLU

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

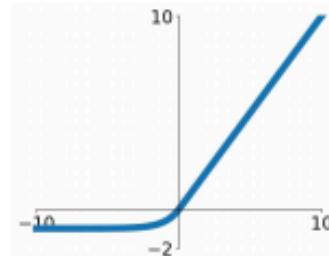


Maxout

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

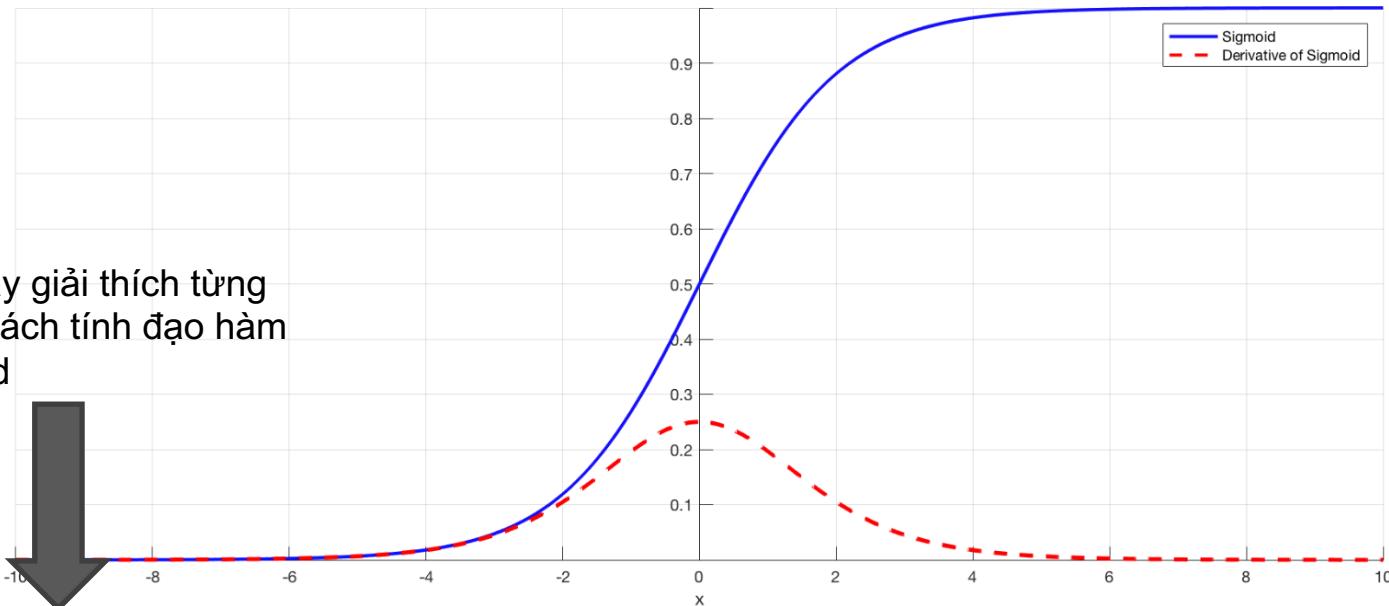
ELU

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



Vi phân hàm sigmoid

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



Link này giải thích từng
bước cách tính đạo hàm
sigmoid

<https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e>

Đặc điểm Sigmoid function

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

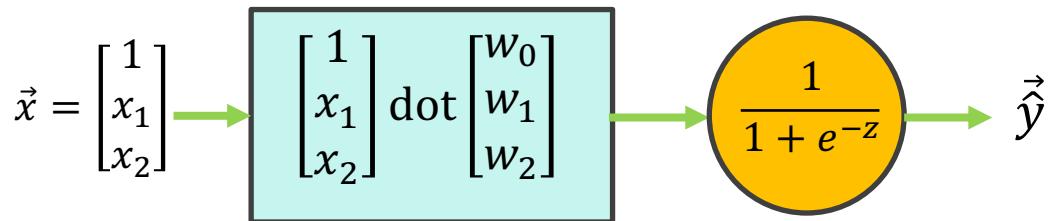
- Giá trị đầu ra luôn trong khoảng [0...1]
- Đạo hàm rất dễ tính toán $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ thuận tiện cho tính toán gradient descent

Output Hypothesis $\vec{\hat{y}}$

$$\vec{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}, \vec{W} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}, \vec{z} = \vec{W} \cdot \vec{x}$$

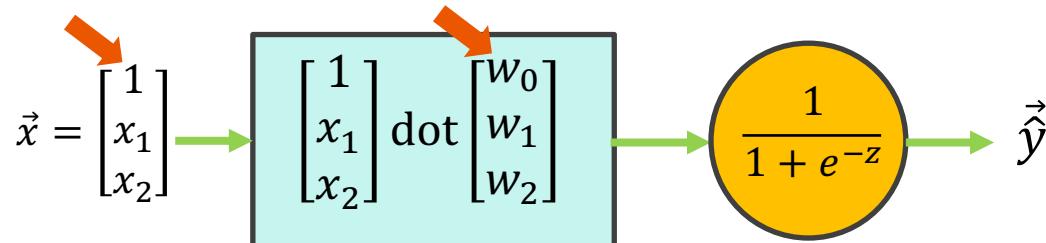
$\vec{\hat{y}} = \frac{1}{1+e^{-\vec{z}}}$ là vector, trong trường hợp tổng quát có nhiều hơn 1 đầu ra

$X = \{(\vec{x}_1, \vec{y}_1), \dots, (\vec{x}_N, \vec{y}_N)\}$ với N mẫu huấn luyện sẽ có N output $\{\vec{\hat{y}}_1, \vec{\hat{y}}_2 \dots \vec{\hat{y}}_N\}$



Bias term

- Nếu $\vec{x} = 0 \rightarrow \vec{z} = \vec{W} \cdot \vec{x} = 0$
- Ngoại trừ sigmoid function, các hàm activation đều trả về 0 nếu đầu vào là 0, $f(\vec{z} = 0) = 0$
- Để có đầu ra phù hợp với dữ liệu huấn luyện cần bổ xung thêm bias term, $b = w_0$ cho một đầu vào quy ước x_0 luôn $= 1$



Loss Function

$$E(X, W) = \frac{1}{N} \left(\frac{1}{2} \sum_{i=1}^N (\hat{y}_i - y_i)^2 \right)$$

- Q: Tại sao lại chia $1/N$?
A: Vì N mẫu dữ liệu huấn luyện, hàm Loss tính cho một epochs đưa N mẫu qua perceptron và tính tổng sai lệch của mỗi mẫu chia trung bình
- Q: Tại sao lại có term $1/2$?
A: Để khi đạo hàm phần bình phương sai lệnh, sẽ triệt tiêu 2 trên tử số
- Q: Còn có các loss function kiểu khác không?
A: Có !

Mục tiêu huấn luyện là Minize Loss Function

- Loss Function $E(X, W)$ là hàm của dữ liệu huấn luyện \vec{X} và weights tại perceptron \vec{W}
- \vec{X} là dữ liệu huấn luyện không thể điều chỉnh
- Vậy chỉ có thể điều chỉnh \vec{W} để $E(X, W)$ nhỏ nhất. Chúng ta dùng phương pháp gradient descent

$$\vec{W}^{t+1} = \vec{W}^t - learning_rate \frac{\partial E(\vec{X}, \vec{W}^t)}{\partial W}$$

 Cái này là vi phần từ phần của E
đối với từng biến W, học ở mấy
trang đầu slide rồi

Partial derivative E với từng biến w_k

$$E(X, W) = \frac{1}{N} \left(\frac{1}{2} \sum_{i=1}^N (\hat{y}_i - y_i)^2 \right)$$

$$E(X, W) = \frac{1}{N} \sum_{i=1}^n E_i \quad \text{with } E_i = \frac{(\hat{y}_i - y_i)^2}{2}$$

Loss function cho mỗi mẫu huấn luyện (x_i, y_i)

$$\frac{\partial E_i}{\partial w_k} = (\hat{y}_i - y_i) \frac{\partial \hat{y}_i}{\partial w_k} \quad \text{with } \hat{y}_i = \frac{1}{1 + e^{-z}} \text{ with } z = Wx$$

Power rule

$$\frac{\partial E_i}{\partial w_k} = (\hat{y}_i - y_i) \hat{y}_i (1 - \hat{y}_i) \frac{\partial z}{\partial w_k}$$

derivative of sigmoid

$$\frac{\partial E_i}{\partial w_k} = (\hat{y}_i - y_i) \hat{y}_i (1 - \hat{y}_i) x_k$$


Chính xác phải viết là $\vec{z} = \vec{W}\vec{x}$ nhưng để đỡ rối mắt
mình bỏ ký hiệu vector nhé

i là chỉ số thứ tự mẫu dữ liệu huấn luyện

k là chỉ số các weight trong perceptron

$$\frac{\partial E_i}{\partial w_k} = (\hat{y}_i - y_i) \hat{y}_i (1 - \hat{y}_i) x_k$$

$$\frac{\partial E_i}{\partial w_0} = (\hat{y}_i - y_i) \hat{y}_i (1 - \hat{y}_i)$$

Do $x_0 = 1$ để bỏ xung bias term

$$\frac{\partial E_i}{\partial w_1} = (\hat{y}_i - y_i) \hat{y}_i (1 - \hat{y}_i) x_1$$

$$\frac{\partial E_i}{\partial w_2} = (\hat{y}_i - y_i) \hat{y}_i (1 - \hat{y}_i) x_2$$

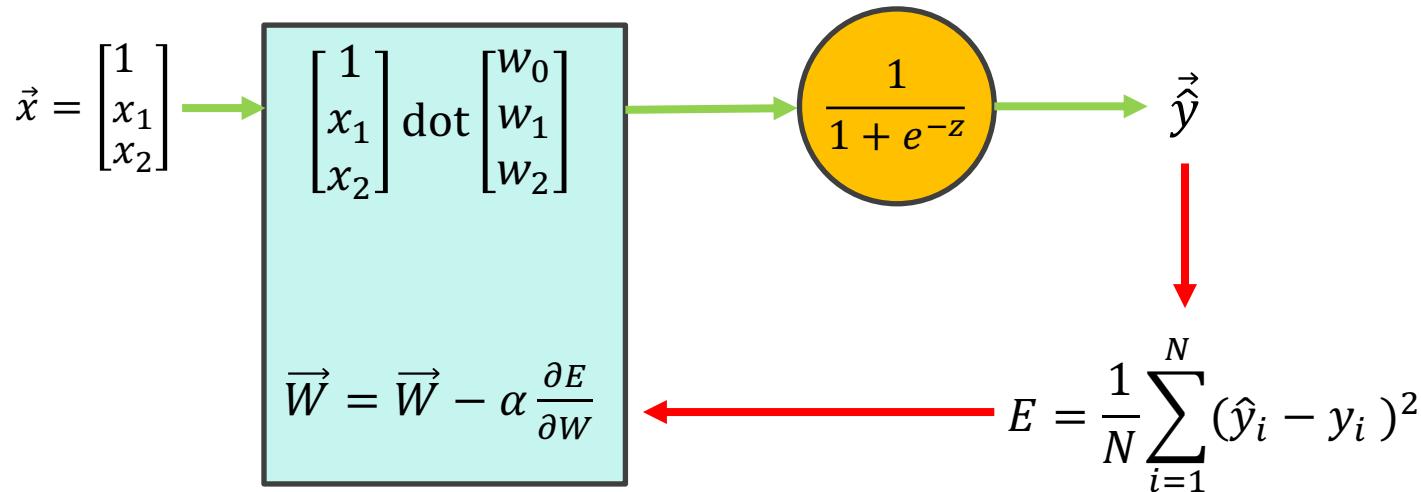
| x_1 | x_2 | Y Output |
|-------|-------|----------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Cập nhật lại Weight

$$w_0 = w_0 - \alpha(\hat{y}_i - y_i)\hat{y}_i(1 - \hat{y}_i)$$

$$w_1 = w_1 - \alpha(\hat{y}_i - y_i)\hat{y}_i(1 - \hat{y}_i)x_1 \quad \alpha \text{ là learning rate}$$

$$w_2 = w_2 - \alpha(\hat{y}_i - y_i)\hat{y}_i(1 - \hat{y}_i)x_2$$



Code thôi !



Mã nguồn 1 perceptron mô phỏng logic AND

```
import matplotlib.pyplot as plt
import numpy as np

def sigmoid(z):
    return 1.0 / (1 + np.exp(-z))

def sigmoid_derivative(z):
    return z * (1.0 - z)
```

```

class Perceptron:
    def __init__(self, X, Y):
        self.N = X.shape[0] # Lấy ra số mẫu number of training samples
        # Thêm bias term vào cột số 0.
        self.X = np.hstack((np.ones([X.shape[0], 1]), X))
        self.Y = Y
        self.weights = np.random.rand(3, 1)
        self.output = np.zeros(self.Y.shape)
        self.mean_square_error_log = []

    def feed_forward(self):
        self.output = sigmoid(np.dot(self.X, self.weights))

    def back_propagation(self):
        output_diff = (self.output - self.Y) # Sai khác giữa real output và desired
output

        # Log lại Mean Square Error mỗi lần huấn luyện
        self.mean_square_error_log.append(np.sum(output_diff ** 2, axis=0) / self.N)

        res = output_diff * sigmoid_derivative(self.output) ←  $(\hat{y}_i - y_i)\hat{y}_i(1 - \hat{y}_i)$ 
        d_weights = np.dot(self.X.T, res) / self.N ←  $\frac{\partial E_i}{\partial w_k} = (\hat{y}_i - y_i)\hat{y}_i(1 - \hat{y}_i)x_k$ 
        alpha = 1 # Learning rate
                    Bỏ qua hoặc chuyển
                    vào learning rate
        self.weights -= alpha * d_weights # Cập nhật lại weight w0, w1, w2

```

```

def main():
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]]) # shape 4x2
    Y = np.array([[0], [0], [0], [1]]) # shape 4x1
    print(Y.shape)

    perceptron = Perceptron(X, Y)
    for i in range(2000): # 2000 epochs
        perceptron.feed_forward()
        perceptron.back_propagation()

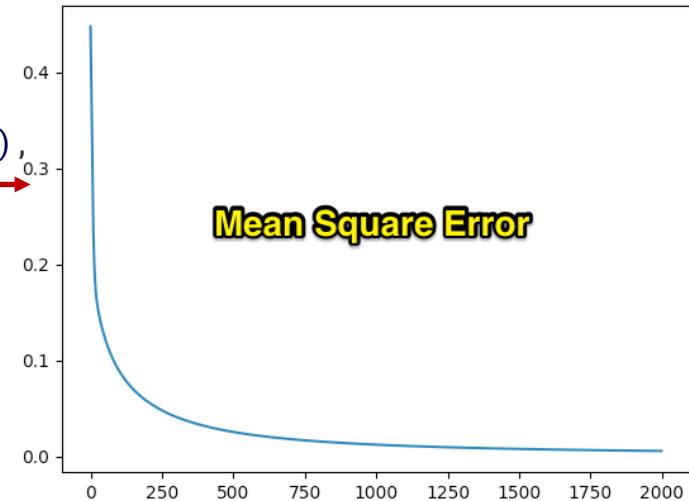
    # In đầu ra sau lần huấn luyện cuối
    np.set_printoptions(precision=3, suppress=True)
    print(perceptron.output)

    # Vẽ biểu đồ Mean Square Error
    plt.plot(np.arange(len(perceptron.mean_square_error_log)),
             perceptron.mean_square_error_log)
    plt.show()

if __name__ == '__main__':
    main()

```

| Thực tế | Kỳ vọng |
|---------|---------|
| [0.001] | [0] |
| [0.082] | [0] |
| [0.082] | [0] |
| [0.903] | [1] |



```
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
```

} Thêm vào đầu file

Thêm vào cuối hàm main



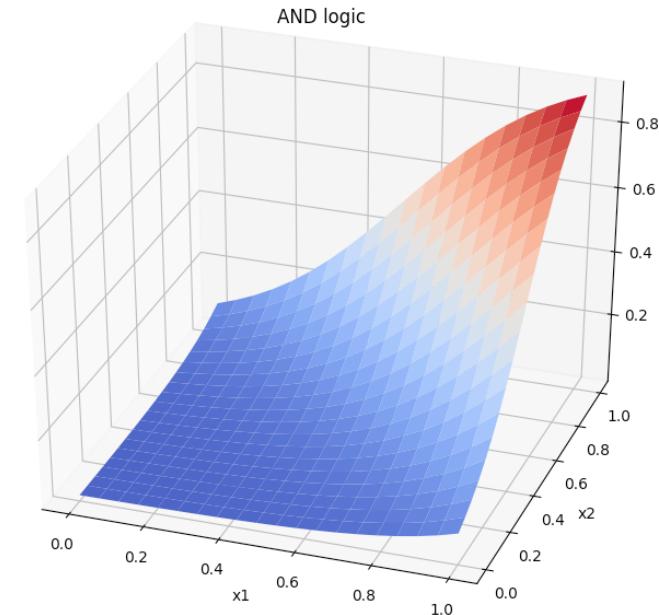
```
# ---- Predict and plot training to surface

K = 20
x = np.linspace(0, 1, K) # Sinh một dữ liệu theo 1 trục x
X1, X2 = np.meshgrid(x, x)

X = np.array([X1, X2]).T.reshape(-1, 2)
perceptron.predict(X)
Z = perceptron.output.reshape(-1, K)

fig = plt.figure()
ax = fig.gca(projection='3d')
ax.set_title("AND logic")
ax.set_xlabel('x1')
ax.set_ylabel('x2')
# Plot the surface.
surf = ax.plot_surface(X1, X2, Z, cmap=cm.coolwarm,
                       linewidth=0, antialiased=True)
plt.show()
```

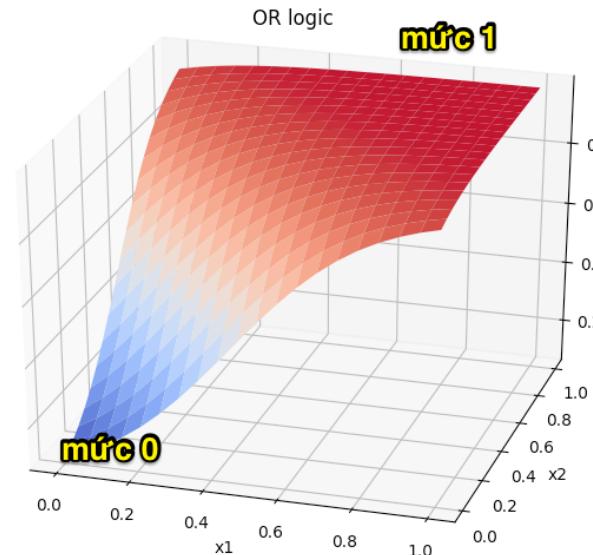
Thử vẽ biểu đồ đầu ra y với đầu vào
x1, x2 bất kỳ trong dải [0...1]



Huấn luyện hàm OR

```
def main():
    X = np.array([[0, 0],
                  [0, 1],
                  [1, 0],
                  [1, 1]])
    Y = np.array([[0],
                  [1],
                  [1],
                  [1]])
```

| x_1 | x_2 | Y Output |
|-------|-------|----------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

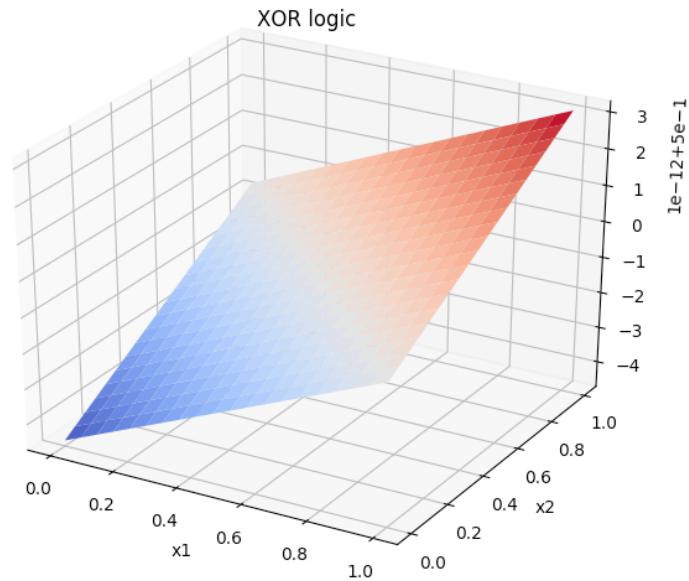


Huấn luyện hàm XOR nhưng failed!

| Kỳ vọng | | |
|---------|-------|----------|
| x_1 | x_2 | Y Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

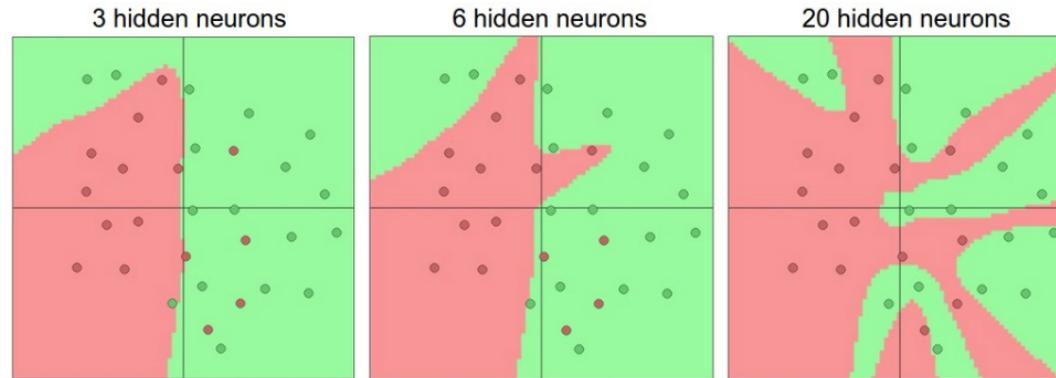
Thực tế rất khác

$[[0.5]$
 $[0.5]$
 $[0.5]$
 $[0.5]]$



Kết luận #1

Dùng duy nhất một perceptron có thể mô phỏng logic AND, OR tốt nhưng thất bại với XOR. Ở bài sau khi bổ xung thêm hidden layers với nhiều perceptron không những xử lý XOR tốt mà còn huấn luyện với mẫu dữ liệu rất phức tạp



Ảnh lấy ở <http://cs231n.github.io/neural-networks-1/>

Kết luận #2

Từ mô hình tối giản duy nhất một perceptron, không có hidden layer, chúng ta áp dụng kỹ thuật backpropagation (lan truyền ngược) để điều chỉnh lại weight

$$\vec{W}^{t+1} = \vec{W}^t - learning_rate \frac{\partial E(\vec{X}, \vec{W}^t)}{\partial W}$$

Khi đã nắm vững nguyên lý, việc lập trình mạng Neural Network hay các mạng phức tạp hơn sẽ yên tâm hơn (không có nghĩa là dễ hơn)

Tham khảo

- <https://brilliant.org/wiki/backpropagation/>
- <https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6>
- <https://ml-cheatsheet.readthedocs.io/en/latest/forwardpropagation.html>
- <http://cs231n.stanford.edu/vecDerivs.pdf>
- <https://www.youtube.com/watch?v=ly4S0oi3Yz8>
- http://tutorial.math.lamar.edu/Extras/CheatSheets_Tables.aspx
- <https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e>

Thời gian đọc tài liệu, code, làm slide 2 ngày x 8 tiếng = 16 tiếng