MACHINE LEARNING

Cao Văn Chung
cvanchung@hus.edu.vn

Informatics Dept., MIM, HUS, VNU Hanoi

Perceptron & Neural Networks

- Perceptron Learning Algorithm (PLA)
    - Binary classifications
    - Loss function
    - Gradient Descent (GD) methods
- Simple Neural Network
- Multilayers Neural Network
- GD method for Multilayers PLA
- Examples
    - XOR Bitwise Operator
    - 2D Points Set Classification

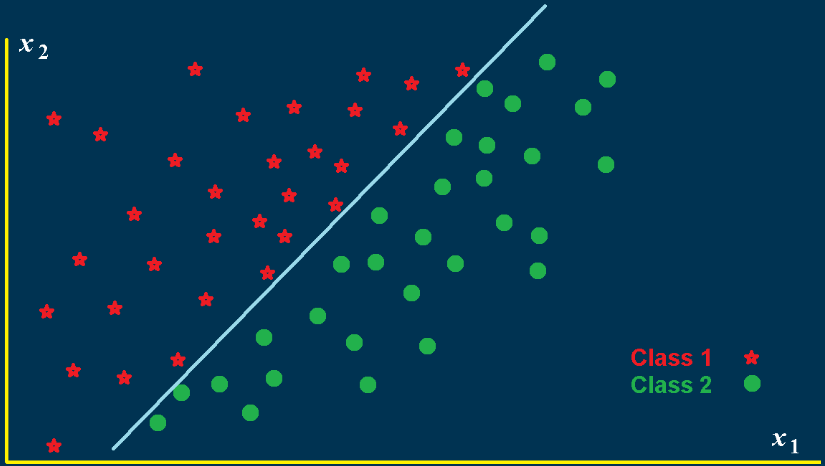
Perceptron Learning Algorithm (PLA)

- Trước hết, ta xét bài toán đơn giản: Phân loại tập dữ liệu vào 02 lớp:
- Biến quan sát: $x = (x_1, \dots x_d)^T \in X \subset \mathbb{R}^d$, biến dự báo $y \in \{-1, 1\}$.
- Giả thiết tập dữ liệu là tách được tuyến tính - *linearly separable* hoặc gần như tách được tuyến tính - *nearly linearly separable* (đặc điểm chung của các phương pháp phân loại tuyến tính), tức là tồn tại siêu phẳng (*hyperplan*)

$$b + w_1 x_1 + \dots w_d x_d = 0 \Leftrightarrow b + \sum_{j=1}^{d} w_j x_j = 0.$$

- Nếu không có tham số b, siêu phẳng bắt buộc phải đi qua gốc tọa độ. Tham số b còn gọi là bias.
- Bổ sung tọa độ $x_0 \equiv 1$ vào x và $w_0 = b$, có thể viết gọn phương trình siêu phẳng: $w^T \bar{x} = 0$.

Perceptron Learning Algorithm (PLA)



Hình: Nearly linearly separable set

Perceptron Learning Algorithm (PLA)

- Perceptron (PLA): Supervised Learning Algorithm - nền tảng của Neural Network.
- Giả sử tập dữ liệu huấn luyện $X = \{x_1, \ldots x_N\} \subset \mathbb{R}^{d \times N}$ và tập biến dự báo tương ứng $y = \{y_1, \ldots y_N\} \in \mathbb{R}^{N \times 1}$ với $y_i = 1$ nếu x_i thuộc Class 1 (đỏ); $y_i = -1$ nếu x_i thuộc Class 2 (lục).
- Thuật toán Perceptron: Đi tìm tham số $w \in \mathbb{R}^d$ để siêu phẳng

$$f_w(x) = b + \sum_{j=1}^{d} w_j x_j = b + w^T x = 0$$

tách tập dữ liệu X vào các class 1 và class 2. Siêu phẳng trên gọi là đường biên - (*boundary*).

Perceptron Learning Algorithm (PLA)

- Nhận xét: Những điểm nằm cùng một phía của boundary sẽ làm cho hàm số $f_w(x)$ có cùng dấu.
- Có thể giả thiết những điểm x thuộc Class 1 sẽ có $f_w(x) \geq 0$ (dấu dương); nếu thuộc Class 2 sẽ có $f_w(x) < 0$ (trường hợp ngược lại đổi dấu của w).
- Vậy có thể xác định class (nhãn) cho dữ liệu x theo cách

$$\text{label}(x) = y = 1 \quad \text{nếu} \quad f_w(x) \geq 0; \quad \text{label}(x) = y = -1 \quad \text{nếu} \quad f_w(x) < 0$$

hoặc đơn giản

$$\text{label}(x) = y = \text{sign}(b + w^T x)$$

ở đây đặt $\text{sign}(0) = 1$.

Hàm tổn thất (Loss function)

- Chú ý: Khoảng cách từ x đến boundary là

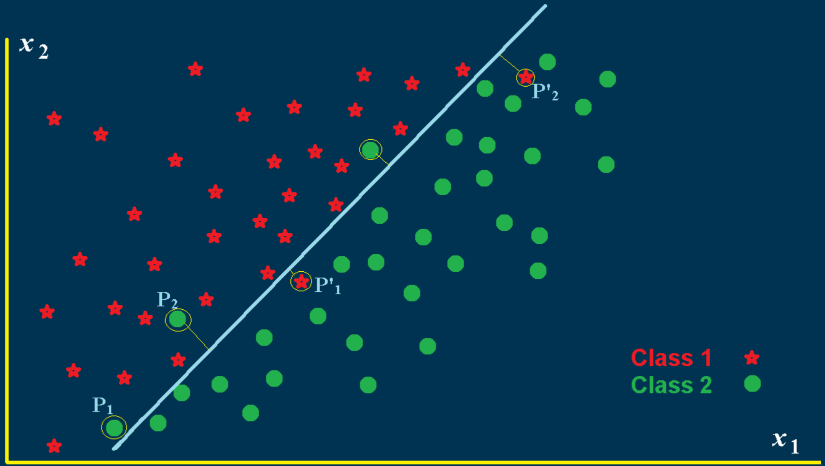
$$\operatorname{dist}(x) = \frac{|b + w_1 x_1 + \ldots + w_d x_d|}{\sqrt{w_1^2 + \ldots w_d^2}} = \frac{|f_w(x)|}{\|w\|_2}.$$

- Theo cách đặt của ta, nếu x được phân lớp đúng, tức là $y = \operatorname{sign}(f_w(x))$, hơn nữa nếu ta nhân w và b với cùng một hệ số thì boudary không đổi. Do đó luôn có thể giả thiết $\|w\|_2 = 1$ và khoảng cách trên trở thành

$$\operatorname{dist}(x) = \frac{|f_w(x)|}{\|w\|_2} = y f_w(x) = y\left(b + w^T x\right). \tag{1}$$

Dễ thấy tính toán với biểu thức này dễ dàng hơn với biểu thức nguyên bản.

Hàm tổn thất (Loss function)



Hình: Phân lớp lỗi và lượng phạt

Hàm tổn thất (Loss function)

- Xét các điểm nằm sai lớp (phân lớp lỗi - misclassified), ví dụ P_1, P_2, P_1', P_2' trong hình, lúc đó $y\text{sign}(b+\theta^T x) = -1$ nên từ (1) ta có

$$\text{dist}(x) = -yf_w(x) = -y\left(b + w^T x\right).$$

- Dựa vào đó, ta xây dựng hàm tổn thất như sau:
  - Nếu điểm dữ liệu x được phân lớp đúng, lượng tổn thất ứng với x là 0;
  - Nếu điểm dữ liệu x được phân sai lớp, ta lấy lượng tổn thất ứng với x là khoảng cách đến đường boundary $\text{dist}(x) = -y\left(b + w^T x\right)$.
- Giả sử bộ tham số ta đang dùng là $(b, w_0, \ldots w_d)^T$. Gọi tập các điểm bị phân lớp sai trong dữ liệu huấn luyện là $\mathcal{M}$. Ta có hàm tổn thất

$$J(\mathbf{w}, b) = \sum_{\mathbf{x}_i \in \mathcal{M}} \left[-y_i\left(b + \mathbf{w}^T \mathbf{x_i}\right)\right]. \tag{2}$$

Giải bài toán tối ưu

- Từ (2), để tìm đường biên tối ưu, ta cần cực tiểu hóa hàm tổn thất, tức là

$$J(\mathbf{w}, b) = \sum_{\mathbf{x}_i \in \mathcal{M}} \left[-y_i \left(b + \mathbf{w}^T \mathbf{x_i}\right)\right] \longrightarrow \min_{w,b} . \tag{3}$$

- Nếu chỉ xét $\mathbf{x}_i \in \mathcal{M}$, ta có thể tính đạo hàm của hàm số $J(\mathbf{w}, b)$. Cụ thể với $\mathbf{x}_i \in \mathcal{M}$, xét lượng mất mát

$$J(\mathbf{w}, b; x_i, y_i) = -y_i \left(b + \mathbf{w}^T \mathbf{x_i}\right)$$

có đạo hàm

$$\nabla_{w,b} J(\mathbf{w}, b; x_i, y_i) = -y_i \left(1, \mathbf{x_i}\right)^T = -y_i \left(1, x_{1,i}, \ldots x_{d,i}\right)^T = -y_i \bar{\mathbf{x}}_\mathbf{i}.$$

Giải bài toán tối ưu

- Vậy ta có thể sử dụng các phương pháp dạng Gradient Descent để giải (3).
- Ví dụ sử dụng **Mini-Batch Gradient Descent**:
  - Khởi tạo: $(\mathbf{w}, b) = (0, 0)$;
  - Giả sử bước hiện tại có xấp xỉ $(\mathbf{w}, b) =: \bar{w}$, tìm $\mathcal{M} = \{x_i | \text{sign}(b + w^T x_i) = -y_i\}$;
  - Cập nhật $(\mathbf{w}, b)$ cho bước tiếp theo:
  $$\bar{w} = \bar{w} - \alpha \sum_{\mathbf{x}_i \in \mathcal{M}} \nabla_{w,b} J(\mathbf{w}, b; x_i, y_i) = \bar{w} + \alpha \sum_{\mathbf{x}_i \in \mathcal{M}} y_i \bar{\mathbf{x}}_i,$$
  ở đây α là hệ số học.
  - Lặp lại cho đến khi đạt yêu cầu.

Giải bài toán tối ưu

- Ví dụ sử dụng **Stochastic Gradient Descent (SGD)**:
    - Khởi tạo: $(\mathbf{w}, b) = (0, 0)$;
    - Giả sử bước hiện tại có xấp xỉ $(\mathbf{w}, b) =: \bar{w}$, tìm $\mathcal{M} = \{x_i | \text{sign}(b + w^T x_i) = -y_i\}$;
        - Với mỗi $\mathbf{x}_i \in \mathcal{M}$, cập nhật $(\mathbf{w}, b)$ cho bước tiếp theo:

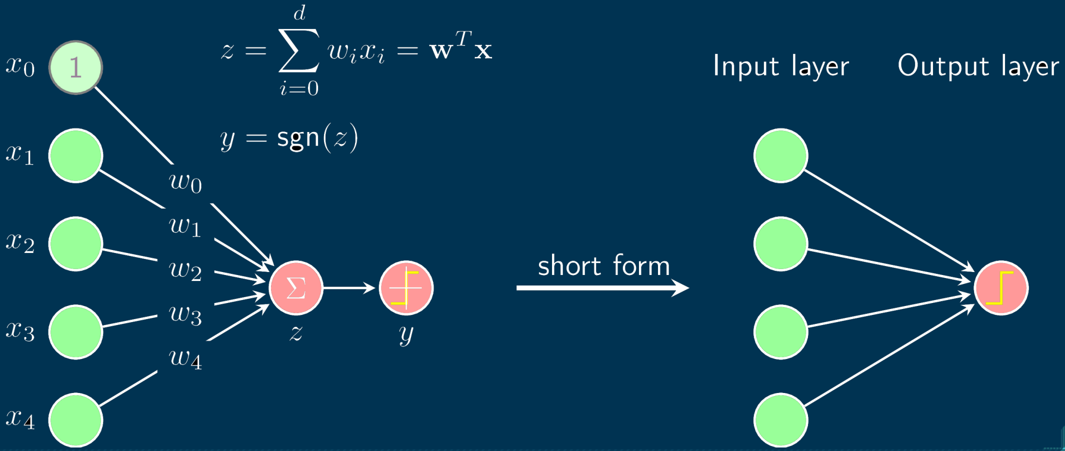
$$\bar{w} = \bar{w} - \alpha \nabla_{w,b} J(\mathbf{w}, b; x_i, y_i) = \bar{w} + y_i \bar{\mathbf{x}}_i,$$

          ở đây α là hệ số học.
        - Lần lượt thực hiện với mọi phần tử trong $\mathcal{M}$.
    - Lặp lại cho đến khi đạt yêu cầu.

Chú ý: *Việc chứng minh sự hội tụ của các phương pháp trên cho bài toán* (3) *dành cho người học.*

Neural Network

Sơ đồ vận hành của thuật toán Perceptron $y = \text{label}(x) = f_w(x) = \text{sign}(\bar{w}^T\bar{x})$ có thể mô tả như hình sau



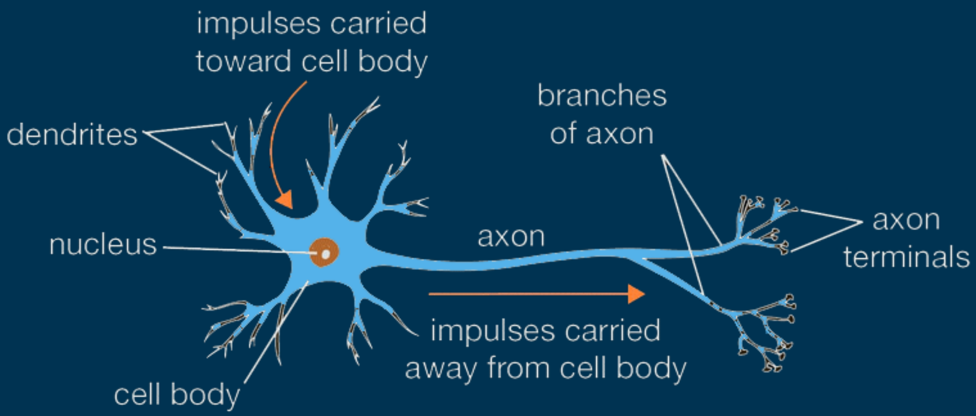
Neural Network

- Trong hình vẽ trên, ta ví dụ
  - Số chiều dữ liệu là 4 $X \subset \mathbb{R}^4$ và ta bổ sung thành phần $x_0 \equiv 1$ và w_0 - bias. Vậy số chiều tính toán là 5.
  - Tham số $w_0, w_1, \ldots, w_d$ gắn trên mũi tên từ x_i đến node biểu thị phép nhân:
    $$z = \sum_{i=0}^{d} w_i x_i = \mathbf{w}^T \mathbf{x}$$
  - Hàm số $y = \text{sign}(z)$ ở đây còn được gọi là *activation function*.
- Tập hợp các node xanh lục (gốc mũi tên) x_i gọi là tầng input (input layer); Tập hợp các giá trị của y gọi là tầng output (output layer).
- Mô tả dạng thu gọn chỉ gồm tầng input và output như ở hình bên phải.

Với cách mô tả như trên, ta thấy phương pháp Perception hoạt động tương tự một Neural Cell, tức là mạng Neural 01 tầng.

Neural Network

Hình: Thuật toán Perceptron gần giống với Neural Cell



Neural Network

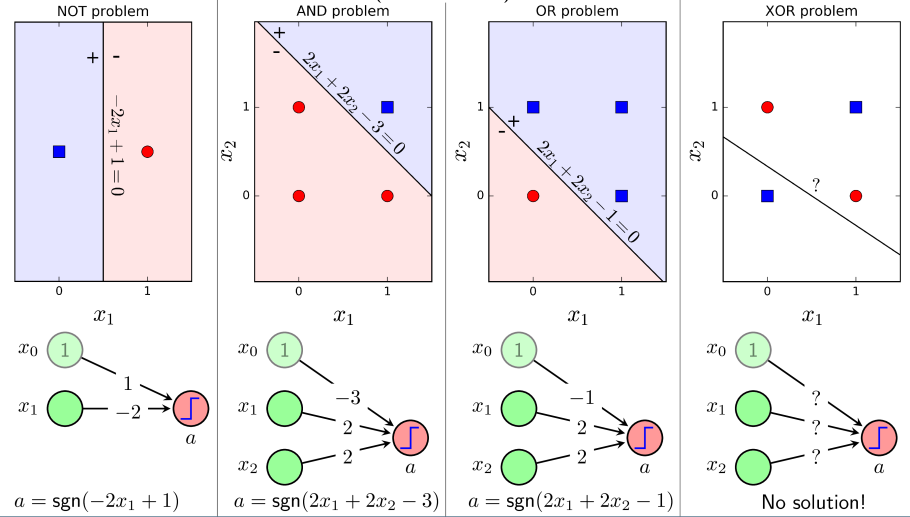
- Thuật toán Perception mô tả trên có dạng Neural Network 01 tầng (01 layer).
- Ở đây, giữa tầng input và tầng output có mối liên hệ tuyến tính:
  - Tồn tại siêu phẳng (tổ hợp tuyến tính) $b + w^T x = 0$ tách tập dữ liệu thành 2 nhóm.
- Trường hợp dữ liệu không tách được tuyến tính như trên, mạng Neural 01 tầng không mô tả được.
- Lúc đó cần bổ sung các tầng giữa tầng input và tầng output, để hai tầng này không liên hệ (tuyến tính) trực tiếp với nhau.

Ví dụ: Dùng Perceptron 01 layer để mô tả các phép toán logic đơn:

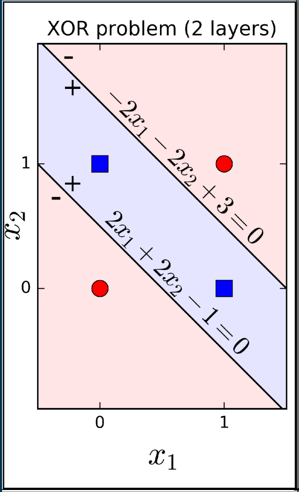
$$X \in \{0, 1\} \times \{0, 1\}; \qquad y \in \{0, 1\}.$$

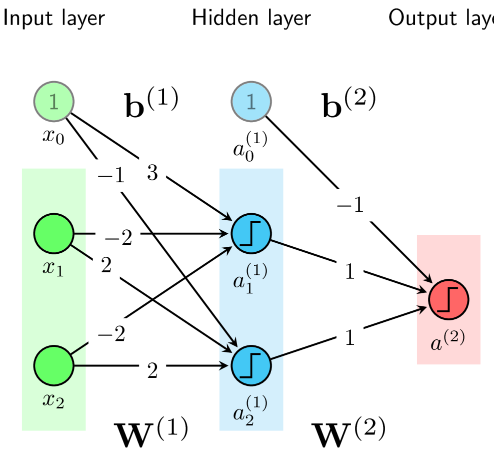
Multilayers Perceptron

(From Internet)



Multilayers Perceptron





(From Internet)

$$\mathbf{W}^{(1)} = \begin{bmatrix} -2 & 2 \\ -2 & 2 \end{bmatrix}; \quad \mathbf{b}^{(1)} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}; \quad \mathbf{b}^{(2)} = \begin{bmatrix} -1 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}; \mathbf{a}^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \end{bmatrix}$$

$$\mathbf{a}^{(1)} = f\left(\mathbf{W}^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}\right)$$

$$a^{(2)} = f\left(\mathbf{W}^{(2)T}\mathbf{a}^{(1)} + \mathbf{b}^{(2)}\right)$$

$$f(.) = \text{sgn}(.) \text{ (element-wise)}$$

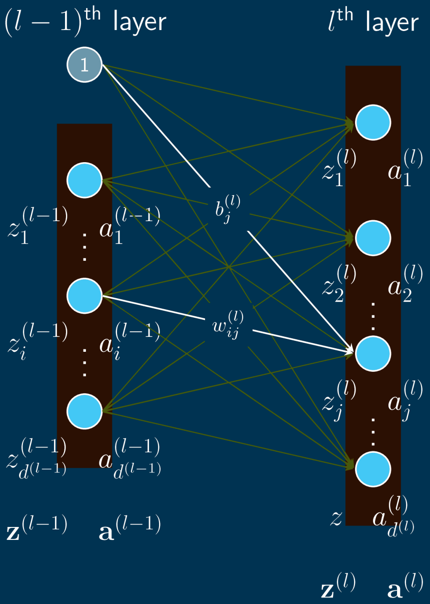
Notes: Layers

- Ngoài Input layers và Output layers, một Multi-layer Perceptron (MLP) có thể có nhiều layers ở giữa - gọi là các Hidden layers.
- Các Hidden layers theo thứ tự từ input layer đến output layer được đánh số thứ thự là Hidden layer 1, Hidden layer 2,...
- Khi đếm số layers của một MLP, ta không tính input layers. Số lượng layer trong một MLP thường được ký hiệu là L.
- Các Hidden layers tổng quát được ký hiệu là l^{th} layer.

Notes: Units

- Dữ liệu input, kết quả output và các node tính toán được gọi chung là các Units.
- Đầu vào của các hidden layer được ký hiệu bởi z; đầu ra của mỗi unit ký hiệu là a (activation, tức giá trị của mỗi unit sau khi ta áp dụng activation function lên z).
- Chú ý: Đầu ra tại mỗi Hidden layer không nhất thiết phải là 2, và không nhất thiết giống nhau.
- Đầu ra của unit thứ i trong layer thứ l được ký hiệu là $a_i^{(l)}$ và số unit trong layer thứ l (không tính bias) là $d^{(l)}$, ta có $a^{(l)} \in \mathbb{R}^{d^{(l)}}$.

Notes



$$\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$$

$$\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)} \times 1}$$

$$z_j^{(l)} = \mathbf{w}_j^{(l)T} \mathbf{a}^{(l-1)} + b_j^{(l)}$$

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)T} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$$

Notes: Weights & Biases

- Ta gọi tham số w trong Perceptron 01 layer là trọng số.
- Phần tử $w_{i,j}^{(l)}$ là tham số trong tổ hợp kết nối từ node thứ i của layer thứ $(l-1)$ tới node j của layer thứ (l).
- Coi input là layer thứ 0, các trọng số $w_{i,j}^{(l)}$ tạo thành ma trận $\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}$. Đây là ma trận trọng số $\mathbf{W}^{(l)}$ kết nối giữa layer thứ $(l-1)$ và layer thứ (l).
- Có L ma trận trọng số cho một MLP có L layers: $\mathbf{W}^{(l)}, \quad l = 1, 2, \ldots, L$.

Notes: Activation functions

- Mỗi output của một unit (trừ các input units) được tính dựa vào công thức:

$$a_i^{(l)} = f(\mathbf{w}_i^{(l)T}\mathbf{a}^{(l-1)} + b_i^{(l)}) = f(\mathbf{z}_i^{(l)}) \quad \text{với} \quad \mathbf{z}_i^{(l)} = \sum_{j=1}^{d^{(l-1)}} \mathbf{w}_{i,j}^{(l)}\mathbf{a}_j^{(l-1)} + b_i^{(l)}.$$

- Ở đây $f(\cdot)$ là một (nonlinear) activation function. Ở dạng vector

$$\mathbf{a}^{(l)} = f(\mathbf{W}^{(l)T}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}) \in \mathbb{R}^{d^{(l)}}.$$

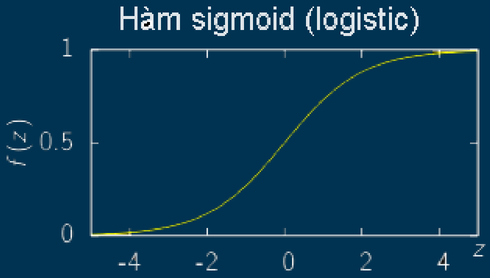
- Khi activation function $f(\cdot)$ được áp dụng cho một ma trận (hoặc vector), ta hiểu rằng nó được áp dụng kiểu *element-wise*, tức cho từng thành phần của ma trận đó

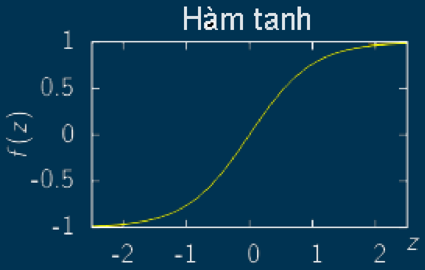
$$\forall A = (a_{ij}) \in \mathbb{R}^{m\times n}: \qquad f(A) = \Big(f(a_{ij})\Big)_{i=\overline{1,m};j=\overline{1,n}}.$$

Notes: Activation functions

Một số hàm activation:

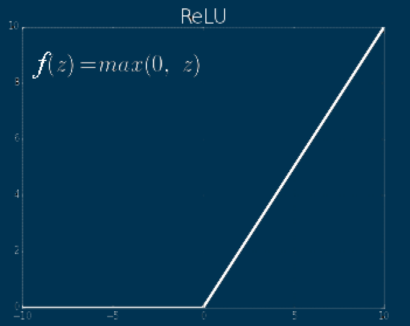
$$\text{sigmoid}(z) = \frac{1}{[1 + \exp(-z)]} \quad \text{và} \quad \tanh(z) = \frac{[\exp(z)] - \exp(-z)]}{[\exp(z)] + \exp(-z)]}$$





Notes: Activation functions

- *Nhược điểm của sigmoid*(z) *và tanh*(z): Với $z \gg 1$ thì $f'(z) \approx 0$.
- sign(z) không dùng được ở đây, vì $f'(z) \equiv 0$ với $z \neq 0$; $f'(0)$ không tồn tại.
- **ReLU (Rectified Linear Unit)** : $f(z) = \max\{0, z\}$.
  Đặt đạo hàm $f'(0) = 0$, lúc đó với $z > 0$, $f'(z) = 1$ và $z \leq 0$, $f'(z) = 0$.



Solving of $\mathbf{W}, \mathbf{b}$

- Một số dạng hàm tổn thất
  - Mean Square Error (MSE):

$$J_0(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y}) = \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{y}_n - \hat{\mathbf{y}}_n\|_2^2 = \frac{1}{N}\sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{a}_n^{(L)}\|_2^2$$

  - Cross-entropy (Xem thêm lý thuyết thông tin):

$$J_0(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y}) = -\sum_{n=1}^{N} (\mathbf{y}_n \log \mathbf{a}_n^{(L)} + (1 - \mathbf{y}_n) \log(1 - \mathbf{a}_n^{(L)})).$$

- $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N}$ - Các cặp dữ liệu huấn luyện; $\hat{\mathbf{y}}_n$ là đầu ra dự báo của $\mathbf{x}_n$ qua mô hình. Chú ý tại layer cuối (layer L^{th}) $\hat{\mathbf{y}}_n = \mathbf{a}_n^{(L)}$.

Solving of $\mathbf{W}, \mathbf{b}$

- Xét bài toán có hiệu chỉnh (Regularization)

$$J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y}) = J_0(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y}) + \frac{\lambda}{2} \sum_{l=1}^{L} \|\mathbf{W}^{(\mathbf{l})}\|_F^2$$

  $\lambda \geq 0$ - tham số hiệu chỉnh; $\|\cdot\|_F$ - chuẩn Frobenius: $\|(a_{ij})\|_F^2 = \sum_i \sum_j a_{ij}^2$.
- Giả sử $J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y})$ là một hàm tổn thất của bài toán
  - $\mathbf{W}, \mathbf{b}$: tập hợp tất cả ma trận trọng số giữa các layers và biases của mỗi layer;
  - $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N, \mathbf{Y} = \{\mathbf{y}_n\}_{n=1}^N$ là tập dữ liệu huấn luyện với mỗi cột tương ứng với một điểm dữ liệu.
- Để tìm các tham số $\mathbf{W}, \mathbf{b}$ cho mô hình, cần giải bài toán cực tiểu hóa

$$J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y}) \longrightarrow \min_{\mathbf{W}, \mathbf{b}}.$$

Solving of $\mathbf{W}, \mathbf{b}$

- Phương pháp đơn giản và phổ biến nhất để tối ưu MLP vẫn là Gradient Descent (GD).
- Nói chung, $J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y})$ không phải là hàm lồi, nên ta không thể chắc cực trị đạt được có phải là cực trị toàn cục.
- Tuy nhiên với xấp xỉ ban đầu được chọn hợp lý, phương pháp dạng Gradient Descent thường vẫn cho nghiệm (bộ tham số $\mathbf{W}, \mathbf{b}$) đủ tốt.

GD method for MLP

- Áp dụng phương pháp GD: Giả sử từ một xấp xỉ ban đầu nào đó, hiện tại ta đang có xấp xỉ $\mathbf{W}^{(l)} = \left(w_{i,j}^{(l)}\right)$; $\mathbf{b}^{(l)} = \left(b_i^{(l)}\right)$, $i = 1, \ldots d^{(l-1)}$; $j = 1, \ldots d^{(l)}$, $l = 1, \ldots L$.
- Tại mỗi bước lặp GD, ta cập nhật lại các trọng số $\mathbf{W}^{(l)}$ và bias $\mathbf{b}^{(l)}$ theo công thức

$$\begin{aligned} w_{i,j}^{(l)} &= w_{i,j}^{(l)} - \alpha \frac{\partial J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y})}{\partial w_{i,j}^{(l)}} = w_{i,j}^{(l)} - \alpha \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial w_{i,j}^{(l)}} \\ b_i^{(l)} &= b_i^{(l)} - \alpha \frac{\partial J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y})}{\partial b_i^{(l)}} = b_i^{(l)} - \alpha \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial b_i^{(l)}}. \end{aligned} \tag{4}$$

GD method for MLP

- Để thực hiện phương pháp, ta cần tính các đạo hàm

$$\frac{\partial}{\partial w_{i,j}^{(l)}} J(\mathbf{W}, \mathbf{b}) \quad \text{và} \quad \frac{\partial}{\partial b_i^{(l)}} J(\mathbf{W}, \mathbf{b}), \quad i = 1, \ldots d^{(l-1)}; j = 1, \ldots d^{(l)}$$

với $l = 1, \ldots L$.

- Nhắc lại cách đặt:

$$\mathbf{z}_i^{(l)} = \sum_{j=1}^{d^{(l-1)}} \mathbf{w}_{i,j}^{(l)} \mathbf{a}_j^{(l-1)} + b_i^{(l)} \quad \text{và} \quad a_i^{(l)} = f(\mathbf{z}_i^{(l)}).$$

$l = 1, \ldots L$; $i = 1, \ldots d^{(l)}$.

GD method for MLP

- Theo quy tắc lấy đạo hàm của hàm hợp

$$\frac{\partial J(\mathbf{W},\mathbf{b})}{\partial w_{i,j}^{(l)}} = \frac{\partial J(\mathbf{W},\mathbf{b})}{\partial \mathbf{z}_i^{(l)}} \frac{\partial \mathbf{z}_i^{(l)}}{\partial w_{i,j}^{(l)}}.$$

- Từ cách đặt ta có

$$\frac{\partial \mathbf{z}_i^{(l)}}{\partial w_{i,j}^{(l)}} = \mathbf{a}_j^{(l-1)}.$$

- Do đó ta chỉ cần phải tính

$$e_i^{(l)} := \frac{\partial J(\mathbf{W},\mathbf{b})}{\partial \mathbf{z}_i^{(l)}}, \quad i = 1,\dots d^{(l)}; l = 1,\dots L.$$

GD method for MLP - FeedForward

Để tính các $e_i^{(l)}$, với xấp xỉ $\mathbf{W}, \mathbf{b}$ hiện tại, chúng ta thực hiện qua 02 quá trình

- **FeedForward**: Quá trình tính các đầu ra theo chiều tiến, từ đầu vào $\{x_n\}$ - layer 0^{th} đến đầu ra $\hat{\mathbf{y}}$ - layer L^{th}

$$
\begin{aligned}
\mathbf{a}^{(0)} &= \mathbf{x}; \\
z_i^{(l)} &= \mathbf{w}_i^{(l)T}\mathbf{a}^{(l-1)} + b_i^{(l)}; \\
\mathbf{z}^{(l)} &= \mathbf{W}^{(l)T}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}; \\
\mathbf{a}^{(l)} &= f(\mathbf{z}^{(l)}), \quad l = 1, 2, \ldots, L; \\
\hat{\mathbf{y}} &= \mathbf{a}^{(L)}.
\end{aligned}
$$

Trong quá trình này, chúng ta lưu lại tất cả $\mathbf{a}^{(l)} = \left(a_j^{(l)}\right)_{j=1}^{d^{(l)}}$; $l = 1, 2, \ldots, L$.

Backpropagation

- **Backpropagation** - Quá trình tính ngược các đạo hàm từ layer L về layer 1. Chú ý: Hàm tổn thất cho N cặp dữ liệu huấn luyện được xây dựng dạng tổng theo các cặp (x_i, y_i). Do đó cách tính đạo hàm dễ dàng được suy ra từ việc tính cho 01 cặp dữ liệu. Vậy chúng ta chỉ cần xét một cặp dữ liệu input (x, y)
  - Tại output (layer L^{th}), lấy đạo hàm của J theo từng thành phần tham số

$$\frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial w_{ij}^{(L)}} = \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial z_j^{(L)}} . \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = e_j^{(L)} a_i^{(L-1)}; \quad \frac{\partial J}{\partial b_j^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} . \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = e_j^{(L)}.$$

Ở đây vì $z_j^{(L)} = \mathbf{w}_j^{(L)T} \mathbf{a}^{(L-1)} + b_j^{(L)}$ nên $\dfrac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = a_i^{(L-1)}$ và $\dfrac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = 1$.

  - Với các dạng đã nêu của J, thành phần $e_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}}$ tính được khá dễ dàng.

Backpropagation

Chú ý, ta đã có các xấp xỉ $w_{ij}^{(l)}$, $b_j^{(l)}$ và tất cả $a_j^{(l)}$ đã tính trong bước feedforward, cho $i = 1, \ldots, d^{(l-1)}; j = 1, \ldots, d^{(l)}$; $l = 1, 2, \ldots, L$. Do đó tính được tất cả $z_j^{(l)}$.

- Tại layer L^{th}

$$e_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \cdot \frac{\partial f(z_j^{(L)})}{\partial z_j^{(L)}} = \frac{\partial J}{\partial a_j^{(L)}} \cdot f'(z_j^{(L)}). \tag{5}$$

- Với $z_j^{(l)}$ đã biết, tính được $f'(z_j^{(l)})$ cho các hàm activation f, ví dụ

$$\text{ReLU:} \quad f'(z) = \begin{cases} 1 & | \quad z > 0 \\ 0 & | \quad z \leq 0 \end{cases};$$

$$\text{Sigmoid:} \quad f'(z) = f(z)\left[1 - f(z)\right]; \ \text{tanh:} \quad f'(z) = 1 - \tanh^2(z).$$

Backpropagation

- $\frac{\partial J}{\partial a_j^{(L)}}$: Các đại lượng này có thể được tính dễ dàng với các dạng J đã nêu. Ta không xét phần hiệu chỉnh $R(W)$ vì dễ dàng tính trực tiếp đạo hàm $\frac{\partial R(W)}{\partial w_{ij}^{(l)}}$.
  - Ví dụ với dạng hàm tổn thất MSE: $\frac{\partial J}{\partial a_j^{(L)}} = 2(a_j^{(L)} - y_j)$.
  - Ví dụ với dạng hàm tổn thất Cross-Entropy: $\frac{\partial J}{\partial a_j^{(L)}} = -\frac{y_j}{a_j^{(L)}} + \frac{1 - y_j}{1 - a_j^{(L)}}$.

    Chú ý ở đây $\log(z)$ áp dụng kiểu element-wise nếu đầu ra nhiều chiều.

Backpropagation

- Ở đây cần chú ý chỉ số j của $a_j^{(L)}$ chỉ thành phần tọa độ thứ j của đầu ra (trường hợp đầu ra nhiều chiều $d^{(L)} > 1$) chứ không phải chỉ số cặp dữ liệu huấn luyện (mà ta dùng chỉ số n).
- Ví dụ:
  - Các bài toán phân loại cho 01 thuộc tính ta đã xét trong phần trước, số chiều đầu ra là $d^{(L)} = 1$, ta chỉ có $a^{(L)}$ (không cần chỉ số j).
  - Nhiều mô hình Multitask learning cần phân loại $d^{(L)} > 1$ thuộc tính. Lúc đó ta có $a_j^{(L)}, j = 1, \ldots, d^{(L)}$. Ví dụ dùng mô hình ML để mô tả ảnh chân dung, và với một ảnh đầu vào x, mô hình cho kết luận dạng:

    Phụ nữ, da trắng, mắt xanh, tóc bạch kim, khoảng 30 tuổi.

    Ở đây đầu ra có 05 thuộc tính phân loại độc lập lẫn nhau, tức là $d^{(L)} = 5$.

Backpropagation

Chúng ta tiếp tục thuật toán lan truyền ngược tới các layer l: $1 < l < L$.

- Tại layer l^{th} ta cũng có

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} . \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} = e_j^{(l)} a_i^{(l-1)}; \quad \frac{\partial J}{\partial b_j^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} . \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = e_j^{(l)}. \tag{6}$$

- Chú ý các đại lượng $a_j^{(l)}$ đã được lưu từ bước FeedForward và do đó tính được $z_j^{(l)}$ với mọi $l = 1, \dots, L$; $e_j^{(L)}$ được tính như đã trình bày, ta chỉ cần tính các $e_j^{(l)}$, với $l < L$.
- Ta tìm công thức dạng truy hồi ngược của $e_j^{(l)}$ qua các $e_k^{(l+1)}$ và các $a_j^{(l)}, z_j^{(l)}$.

Backpropagation

- Ta có $e_j^{(l)} = \dfrac{\partial J}{\partial z_j^{(l)}} = \dfrac{\partial J}{\partial a_j^{(l)}} \cdot \dfrac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \dfrac{\partial J}{\partial a_j^{(l)}} \cdot f'(z_j^{(l)})$, với $f'(z_j^{(l)})$ đã tính được.
- Để có công thức truy hồi ngược, cần biểu diễn $\dfrac{\partial J}{\partial a_j^{(l)}}$ qua các $e_k^{(l+1)} = \dfrac{\partial J}{\partial z_k^{(l+1)}}$.
- Từ $z_k^{(l+1)} = \sum_{q=0}^{d^{(l+1)}} w_{kq}^{(l+1)} a_q^{(l)}$ và quy tắc lấy đạo hàm hàm hợp trong trường hợp hàm nhiều biến, với $j = 1, \ldots, d^{(l)}$, ta có

$$\begin{aligned}\frac{\partial J}{\partial a_j^{(l)}} &= \frac{\partial J\left(z_1^{(l+1)}(a_j^{(l)}), z_2^{(l+1)}(a_j^{(l)}), \ldots, z_{d^{(l+1)}}^{(l+1)}(a_j^{(l)})\right)}{\partial a_j^{(l)}} \\ &= \sum_{k=1}^{d^{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} = \sum_{k=1}^{d^{(l+1)}} e_k^{(l+1)} w_{jk}^{(l+1)}. \end{aligned} \tag{7}$$

Backpropagation

- Đến đây, ta có thể tính tất cả các $e_j^{(l)}$ với $1 < l \leq L-1$, $j = 1, \ldots, d^{(l)}$ với các $e_j^{(L)}$ tính từ (5).
- Áp dụng (6), ta tính được các đạo hàm $\dfrac{\partial J}{\partial w_{ij}^{(l)}}$ và $\dfrac{\partial J}{\partial b_j^{(l)}}$.
- Trong (7), nếu đặt $\mathbf{e}^{(l+1)} = [e_1^{(l+1)}, e_2^{(l+1)}, \ldots, e_{d^{(l+1)}}^{(l+1)}]^T \in \mathbb{R}^{d^{(l+1)} \times 1}$ và ký hiệu dòng thứ j của ma trận $\mathbf{W}^{(l+1)}$ là $\mathbf{w}_{j,:}^{(l+1)}$, ta có biểu thức dạng vector

$$e_j^{(l)} = \left(\mathbf{w}_{j,:}^{(l+1)} \mathbf{e}^{(l+1)}\right) f'(z_j^{(l)}).$$

Stochastic Gradient Descent (SGD)

Để tăng tốc độ tính toán và dễ song song hóa, ta sẽ tính theo các khối ma trận. Nhắc lại, đặt $\mathbf{e}^{(l+1)} = [e_1^{(l+1)}, e_2^{(l+1)}, \dots, e_{d^{(l+1)}}^{(l+1)}]^T \in \mathbb{R}^{d^{(l+1)} \times 1}$, ta có thuật toán SGD cho mô hình MLP:

Với **mỗi** dữ liệu quan sát $\mathbf{x}_n$

- **Feedforward**: Từ $\mathbf{a}^{(0)} = \mathbf{x}_n$, tính và lưu lại các $\mathbf{a}^{(l)}$, $l = 1, \dots L$.
- **Backpropagation**:
  - Với output layer L^{th}, tính: $\mathbf{e}^{(L)} = \dfrac{\partial J}{\partial \mathbf{z}^{(L)}}$.
  - Dựa vào đó để tính $\dfrac{\partial J}{\partial \mathbf{W}^{(L)}} = \mathbf{a}^{(L-1)}\mathbf{e}^{(L)T}$; $\dfrac{\partial J}{\partial \mathbf{b}^{(L)}} = \mathbf{e}^{(L)}$.
  - Với $l = L-1, \dots, 1$, tính: $\mathbf{e}^{(l)} = \left(\mathbf{W}^{(l+1)}\mathbf{e}^{(l+1)}\right) \odot f'(\mathbf{z}^{(l)})$

    trong đó $\odot$ là element-wise product hay Hadamard product, tức lấy từng thành phần của hai vector nhân với nhau.

Stochastic Gradient Descent (SGD)

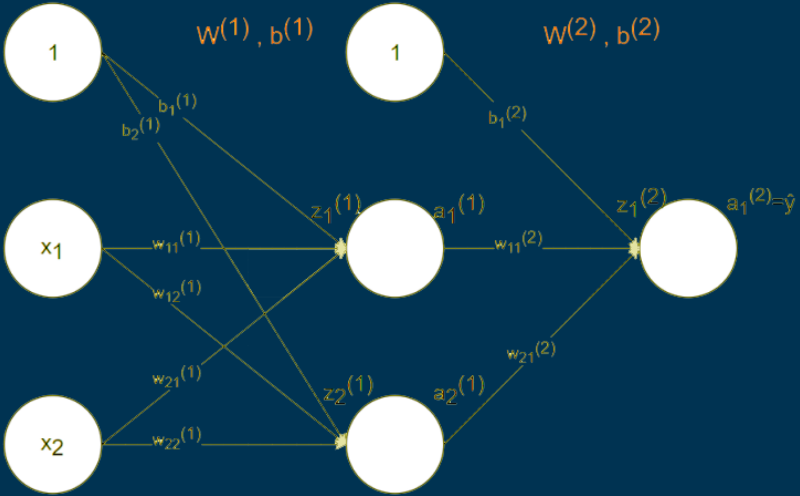
- **Backpropagation**:
  - ...
  - Tính đạo hàm cho ma trận trọng số và vector biases:

$$\frac{\partial J}{\partial \mathbf{W}^{(l)}} = \mathbf{a}^{(l-1)}\mathbf{e}^{(l)T}; \qquad \frac{\partial J}{\partial \mathbf{b}^{(l)}} = \mathbf{e}^{(l)}.$$

- Cập nhật lại trọng số theo công thức (4).

Lặp lại với các dữ liệu quan sát $\mathbf{x}_n$, $n = 1, \ldots, N$.

Eg.1. XOR Bitwise Operator



Eg.1. XOR Bitwise Operator

- Mô hình trên có 2 layer (số lượng layer của mô hình không tính input layer).
- Mô hình: 2-2-1, nghĩa là 2 node trong input layer, 1 hidden layer có 2 node và output layer có 1 node.
- Input layer và hidden layer luôn thêm node 1 để tính bias cho layer sau, nhưng không tính vào số lượng node trong layer.
- Ở mỗi node trong hidden layer và output layer đều thực hiện 2 bước: tính tổng linear và áp dụng activation function.
- Các hệ số $w_{ij}^{(l)}$ và bias $b_j^{(l)}$ tương ứng được ký hiệu như trong hình.

Eg.1. XOR Bitwise Operator

- Chọn ngẫu nhiên các $\mathbf{W}^{(l)}$ và đặt $\mathbf{b}^{(l)} \equiv 0$; $l = 1, 2$.
- **Feedforward**: Dạng ma trận

$$\begin{aligned}
\mathbf{Z}^{(1)} &= \mathbf{X} * \mathbf{W}^{(1)} + \mathbf{b}^{(1)};\\
\mathbf{A}^{(1)} &= f(\mathbf{Z}^{(1)});\\
\mathbf{Z}^{(2)} &= \mathbf{A}^{(1)} * \mathbf{W}^{(2)} + \mathbf{b}^{(2)};\\
\mathbf{A}^{(2)} &= f(\mathbf{Z}^{(2)}) = \text{sigmoid}(\mathbf{Z}^{(2)}).
\end{aligned}$$

Eg.1. XOR Bitwise Operator

- Sử dụng hàm kích hoạt sigmoid và hàm tổn thất Cross-Entropy (chú ý ở đây đầu ra chỉ có 1 chiều): $J = -\sum_{n=1}^{N}\left(y_n \log\left(a_n^{(2)}\right) + (1-y_n)\log\left(1-a_n^{(2)}\right)\right)$.
- Chỉ xét tại 01 cặp dữ liệu $(\mathbf{x}, y)$ (trường hợp sử dụng thuật toán Batch GD, ta chỉ cần lấy tổng). Tại layer output $(L=2)$: $\dfrac{\partial J}{\partial a^{(2)}} = -\left(\dfrac{y}{a^{(2)}} - \dfrac{(1-y)}{(1-a^{(2)})}\right)$.
- **Backpropagation**:

$$\frac{\partial J}{\partial b_1^{(2)}} = \frac{\partial J}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial b_1^{(2)}} = -\left(\frac{y}{a^{(2)}} - \frac{(1-y)}{(1-a^{(2)})}\right) a^{(2)}(1-a^{(2)}) = a^{(2)} - y;$$

$$\frac{\partial a^{(2)}}{\partial w_{11}^{(2)}} = a_1^{(1)} a^{(2)}(1-a^{(2)}) \Leftrightarrow \frac{\partial J}{\partial w_{11}^{(2)}} = \frac{\partial J}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial w_{11}^{(2)}} = a_1^{(1)}(a^{(2)} - y); \ldots$$

Eg.1. XOR Bitwise Operator

- **Backpropagation**:

$$\frac{\partial a^{(2)}}{\partial w_{21}^{(2)}} = a_2^{(1)} a^{(2)}(1 - a^{(2)}) \Leftrightarrow \frac{\partial J}{\partial w_{11}^{(2)}} = \frac{\partial J}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial w_{21}^{(2)}} = a_2^{(1)}(a^{(2)} - y);$$

$$\frac{\partial J}{\partial w_{11}^{(1)}} = x_1 a_1^{(1)}(1 - a_1^{(1)}) w_{11}^{(2)}(a^{(2)} - y);$$

$$\frac{\partial J}{\partial w_{12}^{(1)}} = x_1 a_2^{(1)}(1 - a_2^{(1)}) w_{11}^{(2)}(a^{(2)} - y);$$

$$\frac{\partial J}{\partial w_{21}^{(1)}} = x_1 a_1^{(1)}(1 - a_1^{(1)}) w_{21}^{(2)}(a^{(2)} - y);$$

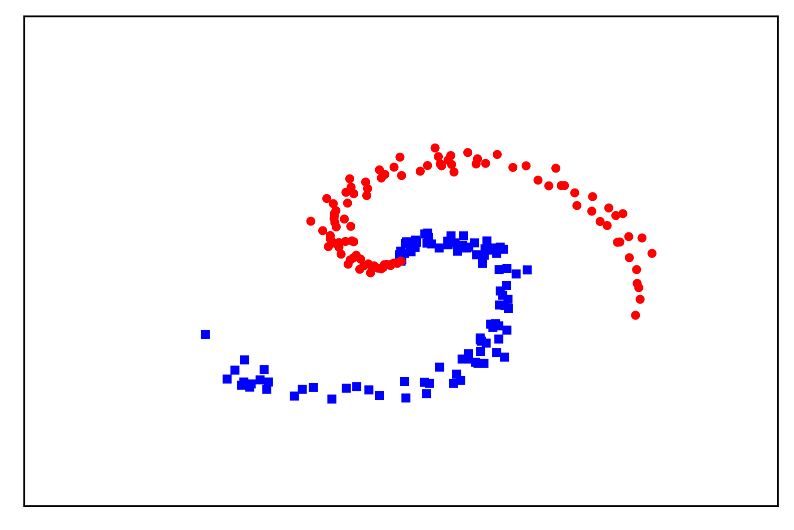
$$\frac{\partial J}{\partial w_{22}^{(1)}} = x_2 a_2^{(1)}(1 - a_2^{(1)}) w_{21}^{(2)}(a^{(2)} - y);$$

$$\frac{\partial J}{\partial w_{11}^{(1)}} = a_1^{(1)}(1 - a_1^{(1)}) w_{11}^{(2)}(a^{(2)} - y).$$

Eg.2. 2D Points Set Classification

- Ví dụ thứ hai lấy từ https://cs231n.github.io/neural-networks-case-study/.
- Xây dựng một dataset là các điểm 2D, phân vào C lớp không tách được tuyến tính.
- Nếu đã học phần hồi quy logistic nhiều lớp, chúng ta tạo $C = 3$ lớp và sử dụng softmax để phân loại. Trường hợp ngược lại ta đặt $C = 2$ và sử dụng perceptron.
- Phần code (python) tạo dữ liệu được gửi kèm cùng phần bài giảng.
- Người đọc hoàn thành phần chương trình cho mô hình mạng neural với $C = 2$ (bắt buộc) và phát triển với $C = 3$ (tùy chọn).

Eg.2. 2D Points Set Classification



Eg.2. 2D Points Set Classification

Code tạo dữ liệu.

```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import math
import numpy as np
import matplotlib.pyplot as plt

N = 100 # number of points per class
d0 = 2 # dimensionality
C = 2 # number of classes
X = np.zeros((d0, N*C)) # data matrix (each row = single example)
y = np.zeros(N*C, dtype='uint8') # class labels
```

Eg.2. 2D Points Set Classification

```
for j in range(C):
  ix = range(N*j,N*(j+1))
  r = np.linspace(0.0,1,N) # radius
  t = np.linspace(j*4,(j+1)*4,N) + np.random.randn(N)*0.2 # theta
  X[:,ix] = np.c_[r*np.sin(t), r*np.cos(t)].T
  y[ix] = j
```