MẠNG NORON NHÂN TẠO và GIẢI THUẬT DI TRUYỀN

Neural Network & Genetic Algorithm

Biên soạn: ThS.Phạm Đình Tài
pdtai@ntt.edu.vn
098573.39.39

CHƯƠNG 5

Một số phương pháp huấn luyện mạng

MỤC TIÊU

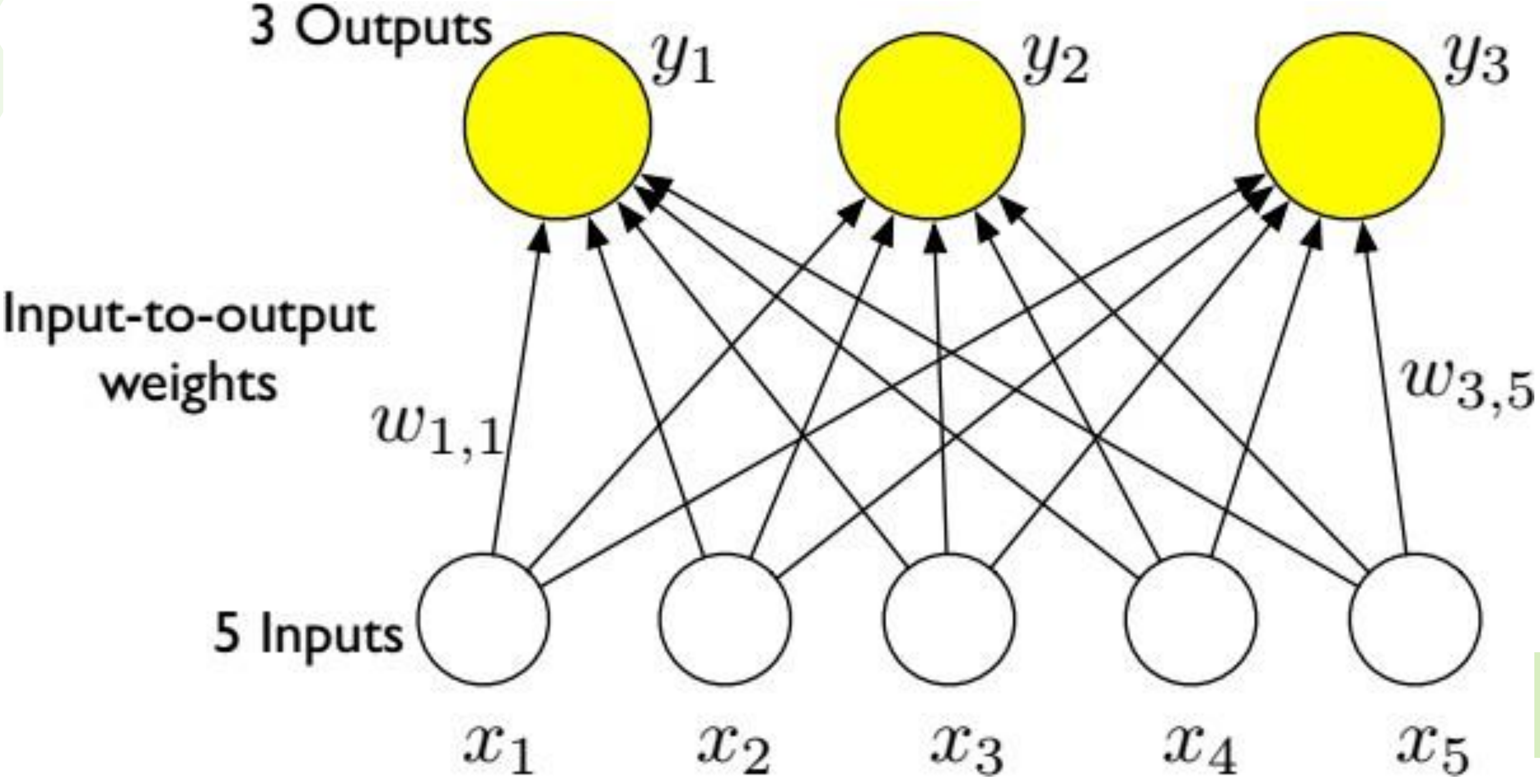
- ✓ **Dựa trên các giải thuật học để huấn luyện mạng neuron một lớp và nhiều lớp (MLP)**
- ✓ **Thực hành với một trong số các thuật toán trên**

Source: Steve Renals and Pavlos Andreadis, Machine Learning Practical — MLP

CÁC MẠNG CÓ MỘT LỚP

- **Mục tiêu:** Học phép ánh xạ một vector đầu vào x để đưa ra đầu ra y
- **Quá trình học**: tối ưu các tham số của hệ thống
- **Khái quát hóa**: tính độ chính xác của đầu ra với các mẫu chưa biết
- **Mạng một lớp**: sử dụng một lớp để ánh xạ đầu vào và ra

CÁC MẠNG CÓ MỘT LỚP



CÁC MẠNG CÓ MỘT LỚP

Input vector $\mathbf{x} = (x_1, x_1, \ldots, x_d)^T$

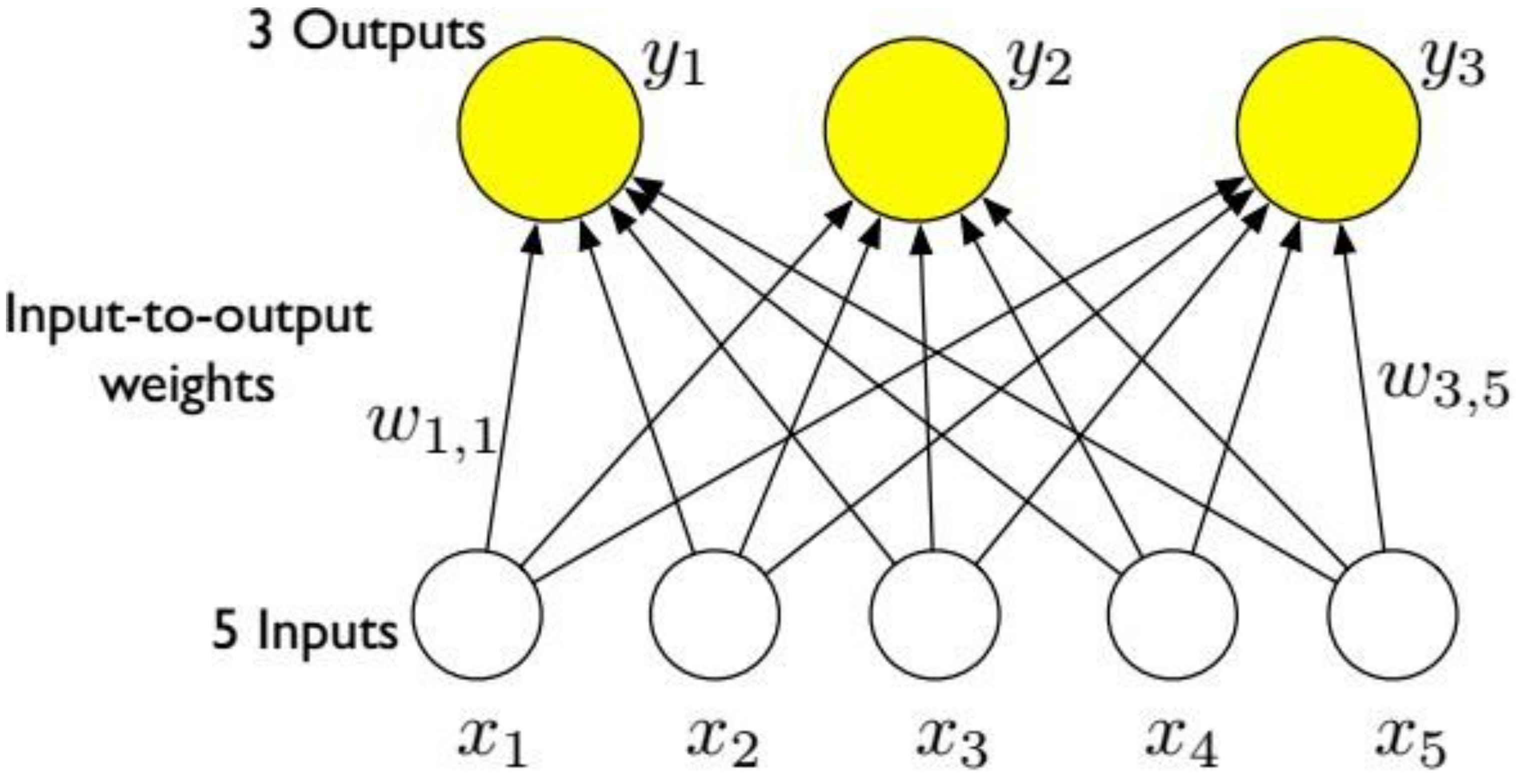
Output vector $\mathbf{y} = (y_1, \ldots, y_K)^T$

Weight matrix $\mathbf{W}$: w_{ki} is the weight from input x_i to output y_k

Bias b_k is the bias for output k

$$y_k = \sum_{i=1}^{d} w_{ki} x_i + b_k \quad ; \quad \mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

CÁC MẠNG CÓ MỘT LỚP



$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

$$y_k = \sum_{i=1}^{d} W_{ki} x_i + b_k$$

HUẤN LUYỆN MẠNG MỘT LỚP

Training set N input/output pairs $\{(\mathbf{x}^n, \mathbf{t}^n) : 1 \le n \le N\}$

Target vector $\mathbf{t}^n = (t_1^n, \ldots, t_K^n)^T$ – the target output for input $\mathbf{x}^n$

Output vector $\mathbf{y}^n = \mathbf{y}^n(\mathbf{x}^n; \mathbf{W}, \mathbf{b})$ – the output computed by the network for input $\mathbf{x}^n$

Trainable parameters weight matrix $\mathbf{W}$, bias vector $\mathbf{b}$

Training problem Set the values of the weight matrix $\mathbf{W}$ and bias vector $\mathbf{b}$ such that each input $\mathbf{x}^n$ is mapped to its target $\mathbf{t}^n$

Error function Define the training problem in terms of an error function E; training corresponds to setting the weights so as to minimise the error

Supervised learning There is a target output for each input

HÀM LỖI

- Error function should measure how far an output vector is from its target – e.g. (squared) Euclidean distance – *sum square error*.
  E^n is the error per example:

$$\boxed{E^n} = \frac{1}{2}||\mathbf{y}^n - \mathbf{t}^n||^2 = \frac{1}{2}\sum_{k=1}^{K}(y_k^n - t_k^n)^2$$

  E is the total error averaged over the training set:

$$\boxed{E} = \frac{1}{N}\sum_{n=1}^{N} E^n = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}||\mathbf{y}^n - \mathbf{t}^n||^2\right)$$

- Training process: set **W** and **b** to minimise E given the training set

KHÔNG GIAN TRỌNG SỐ VÀ ĐẠO HÀM

- **Weight space:** A $K \times d$ dimension space – each possible weight matrix corresponds to a point in weight space. $E(\mathbf{W})$ is the value of the error at a specific point in weight space (given the training data).
- **Gradient** of $E(\mathbf{W})$ given $\mathbf{W}$ is $\nabla_{\mathbf{W}} E$, the matrix of partial derivatives of E with respect to the elements of $\mathbf{W}$:
- **Gradient Descent Training:** adjust the weight matrix by moving a small direction down the gradient, which is the direction along which E decreases most rapidly.
  - update each weight w_{ki} by adding a factor $-\eta \cdot \partial E / \partial w_{ki}$
  - η is a small constant called the *step size* or *learning rate*.
- Adjust bias vector similarly

1. Initialise weights and biases with small random numbers
2. For each epoch (complete pass through the training data)
   1. Initialise total gradients: $\Delta w_{ki} = 0$, $\Delta b_k = 0$
   2. For each training example n:
      1. Compute the error E^n
      2. For all k, i: Compute the gradients $\partial E^n / \partial w_{ki}$, $\partial E^n / \partial b_k$
      3. Update the total gradients by accumulating the gradients for example n

$$\Delta w_{ki} \leftarrow \Delta w_{ki} + \frac{\partial E^n}{\partial w_{ki}} \quad \forall k, i$$

$$\Delta b_k \leftarrow \Delta b_k + \frac{\partial E^n}{\partial b_k} \quad \forall k$$

   3. Update weights:

$$\Delta w_{ki} \leftarrow \Delta w_{ki} / N; \qquad w_{ki} \leftarrow w_{ki} - \eta \Delta w_{ki} \quad \forall k, i$$

$$\Delta b_k \leftarrow \Delta b_k / N; \qquad b_k \leftarrow b_k - \eta \Delta b_k \quad \forall k$$

Terminate either after a fixed number of epochs, or when the error stops decreasing by more than a threshold.

GIẢI THUẬT HỌC

- Error function:

$$E = \frac{1}{N}\sum_{n=1}^{N} E^n \qquad E^n = \frac{1}{2}\sum_{k=1}^{K}(y_k^n - t_k^n)^2$$

- Gradients:

$$\boxed{\frac{\partial E^n}{\partial w_{rs}}} = \frac{\partial E^n}{\partial y_r}\frac{\partial y_r}{\partial w_{rs}} = \underbrace{(y_r^n - t_r^n)}_{\text{output error}}\ \underbrace{x_s^n}_{\text{input}}$$

$$\boxed{\frac{\partial E}{\partial w_{rs}}} = \frac{1}{N}\sum_{n=1}^{N}\frac{\partial E^n}{\partial w_{rs}} = \frac{1}{N}\sum_{n=1}^{N}(y_r^n - t_r^n)x_s^n$$

- Weight update

$$w_{rs} \leftarrow w_{rs} - \eta \cdot \frac{1}{N}\sum_{n=1}^{N}(y_r^n - t_r^n)x_s^n$$

STOCHASTIC GRADIENT DESCENT

- **Việc huấn luyện mạng sử dụng cả tập mẫu là rất chậm (batch gradient descent):**
  - Khi cập nhật ma trận trọng số ta phải sử dụng tất cả các điểm dữ liệu
  - Cách làm này hạn chế khi dữ liệu lớn (>1 tỷ người dùng facebook)
  - Việc tính toán đạo hàm của tất cả các điểm này trở nên cồng kềnh và không hiệu quả
  - Thuật toán này không hiệu quả với online learning
- **Giải pháp: Stochastic Gradient Descent**

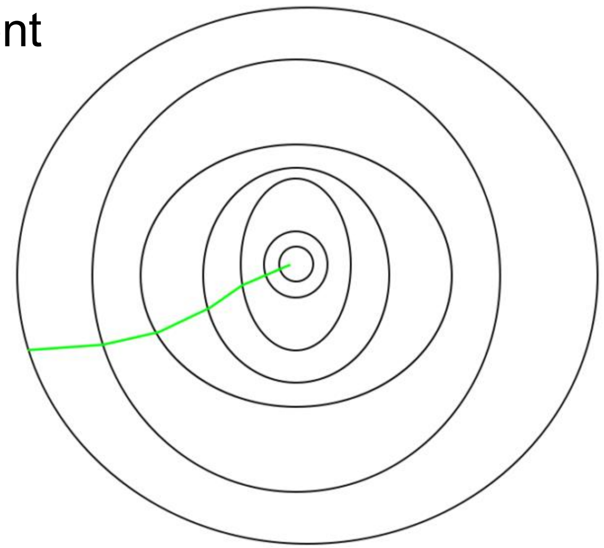
STOCHASTIC GRADIENT DESCENT

- Tại một thời điểm ta chỉ tính đạo hàm trên một mẫu và cập nhật ma trận trọng số
- Mỗi lần duyệt qua tất cả các điểm trên toàn bộ dữ liệu ta gọi là epoch
  - Với **GD**: mỗi lần cập nhật ma trận trọng số gọi là một ***epoch***
  - Với **SGD**: Mỗi epoch tương ứng với N lần cập nhật trọng số
- Việc cập nhật từng điểm có thể giảm tốc độ thực hiện 1 epoch
- Tuy nhiên SGD chỉ yêu cầu một lượng epoch nhỏ và thường phù hợp với bài toán có dữ liệu lớn *(deep learning)*

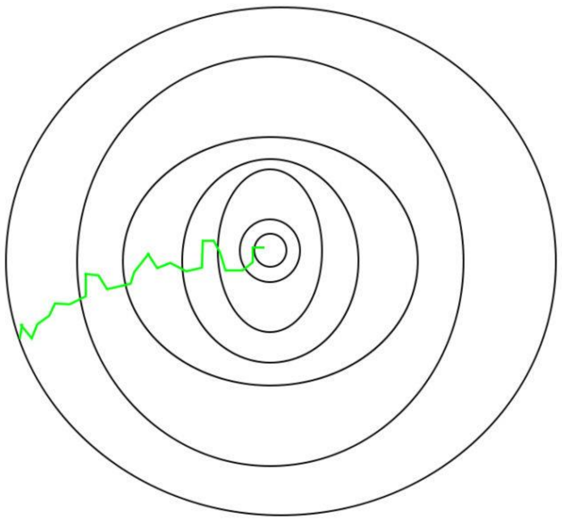
STOCHASTIC GRADIENT DESCENT

- Thứ tự chọn điểm dữ liệu: sau mỗi epoch ta cần trộn thứ tự của các điểm dữ liệu để đảm bảo tính ngẫu nhiên
- Giải thuật SGD hội tụ nhanh hơn GD

Đường dẫn được thực hiện bởi Batch Gradient Descent



Đường dẫn được thực hiện bởi Stochastic Gradient Descent



GIẢI THUẬT SGD

procedure SGDTRAINING($\mathbf{X}, \mathbf{T}, \mathbf{W}$)
    initialize $\mathbf{W}$ to small random numbers
    randomize order of training examples in $\mathbf{X}$
    while not converged **do**
        for $n \leftarrow 1, N$ **do**
            for $k \leftarrow 1, K$ **do**
                $y_k^n \leftarrow \sum_{i=1}^{d} w_{ki} x_i^n + b_k$
                $g_k^n \leftarrow y_k^n - t_k^n$
                for $i \leftarrow 1, d$ **do**
                    $w_{ki} \leftarrow w_{ki} - \eta \cdot g_k^n \cdot x_i^n$
                end for
                $b_k \leftarrow b_k - \eta \cdot g_k^n$
            end for
        end for
    end while
end procedure

N: Số mẫu học
K: Số neuron của mạng
d: Số chiều của vector đầu vào

Xem video minh họa
https://www.youtube.com/watch?v=vMh0zPT0tLI

MINIBATCHES

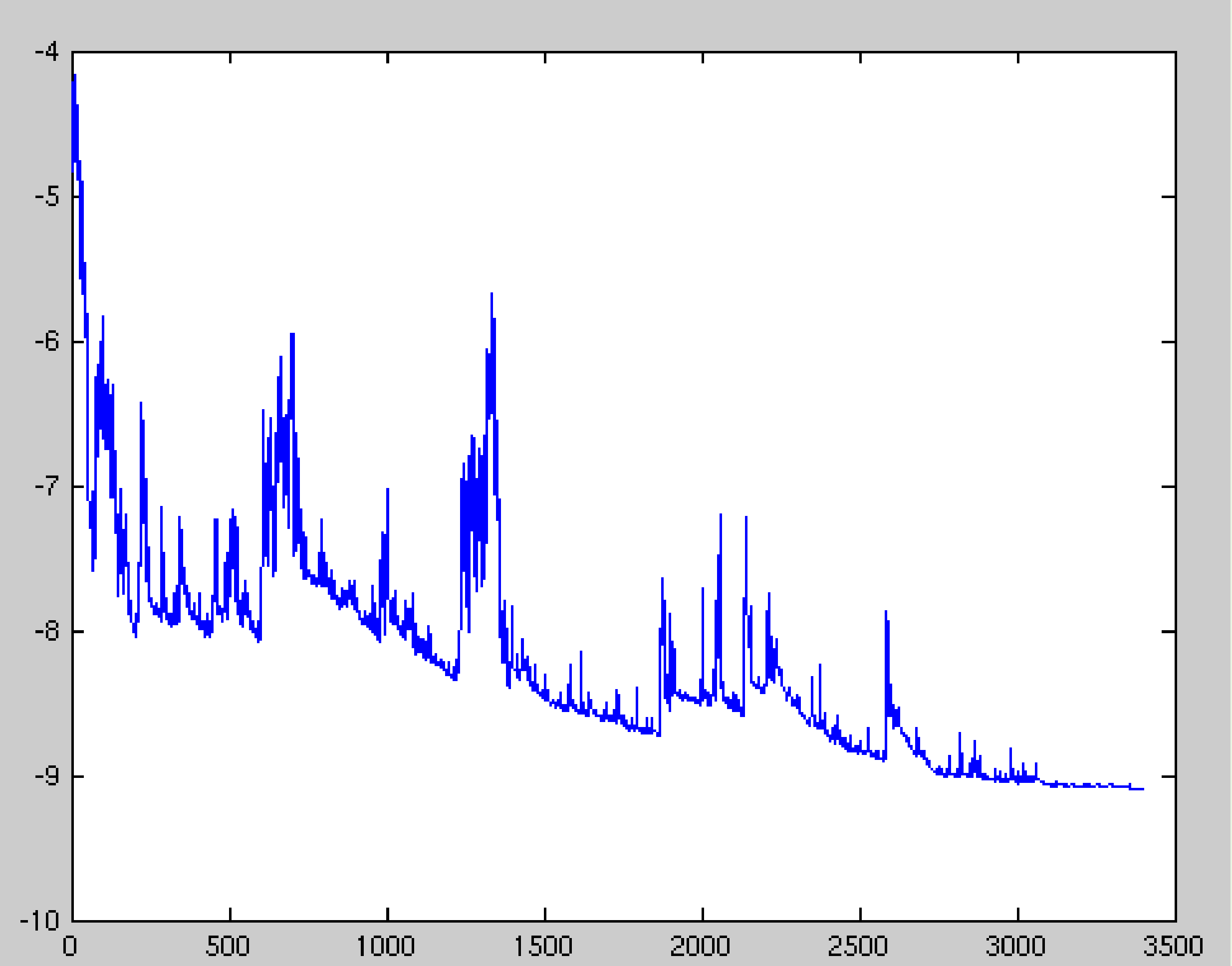
- **Minibatch**: mini-batch sử dụng một số lượng n>1 (nhưng n << N là tổng số dữ liệu)
- **Mini-batch Gradient Descent:**
  - Bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu
  - Sau đó, chia toàn bộ dữ liệu thành các mini-batch, mỗi mini-batch có n điểm dữ liệu (trừ mini-batch cuối có thể có ít hơn nếu N không chia hết cho n)
  - Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật.
- **Mini-batch GD:**
  - Được sử dụng trong hầu hết các thuật toán Machine Learning, đặc biệt là trong *Deep Learning*.
  - Giá trị n thường được chọn là khoảng từ 50 đến 100.

MINIBATCHES

- **Minibatch**: mini-batch sử dụng một số lượng n>1 (nhưng n << N là tổng số dữ liệu)
- **Mini-batch Gradient Descent:**
  - Bắt đầu mỗi epoch bằng việc xáo trộn ngẫu nhiên dữ liệu
  - Sau đó, chia toàn bộ dữ liệu thành các mini-batch, mỗi mini-batch có n điểm dữ liệu (trừ mini-batch cuối có thể có ít hơn nếu N không chia hết cho n)
  - Mỗi lần cập nhật, thuật toán này lấy ra một mini-batch để tính toán đạo hàm rồi cập nhật.
- **Mini-batch GD:**
  - Được sử dụng trong hầu hết các thuật toán Machine Learning, đặc biệt là trong *Deep Learning*.
  - Giá trị n thường được chọn là khoảng từ 50 đến 100.

MINIBATCHES

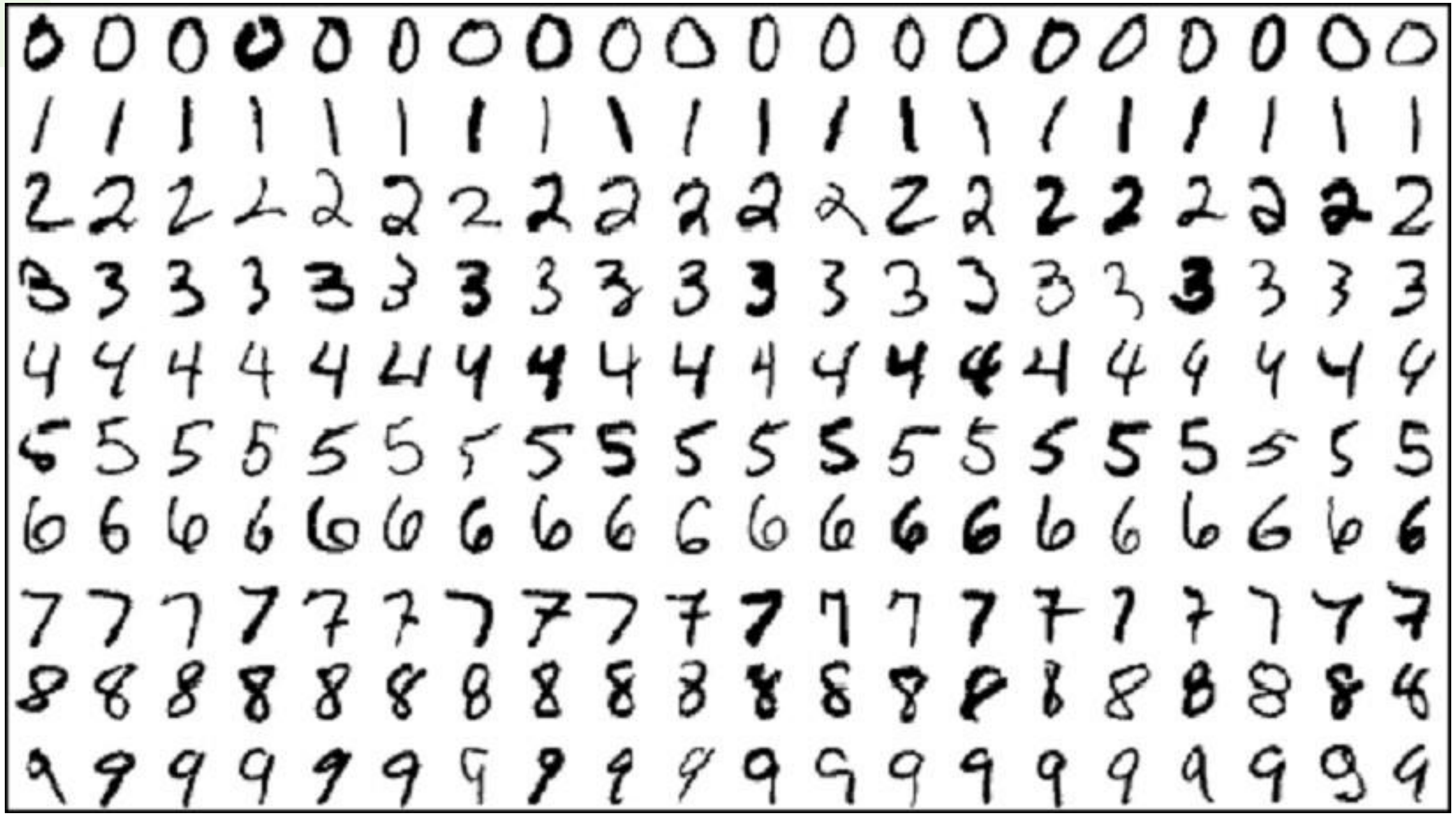
Ví dụ: Về giá trị của hàm mất mát mỗi khi cập nhật tham số θ của một bài toán khác phức tạp hơn.



Hàm mất mát *nhảy lên nhảy xuống* (fluctuate) sau mỗi lần cập nhật nhưng nhìn chung giảm dần và có xu hướng hội tụ về cuối. (Nguồn: Wikipedia).

PHÂN LỚP VÀ HỒI QUY

- **Bài toán:** Nhận dạng chữ số viết tay



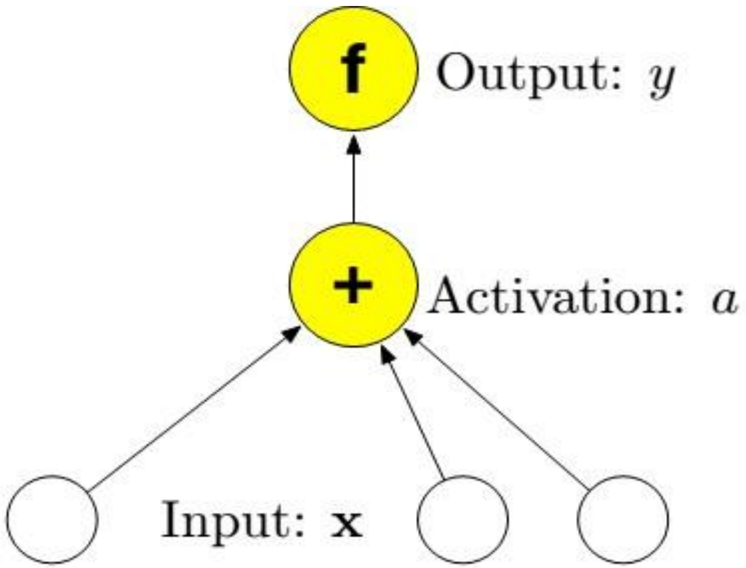
https://www.cs.toronto.edu/~graves/handwriting.html

MNIST dataset: http://yann.lecun.com/exdb/mnist/

GIỚI THIỆU

- **Regression (hồi quy):** Dự báo giá trị của đầu ra từ đầu vào cho trước
  - Dự báo lượng mưa trong ngày mai
  - Dự báo số lượng người dừng lại ở một cửa hàng
  - Dự báo số lần truy cập vào một trang web nào đó
- **Classification (phân lớp):** Dự báo lớp (loại) từ một đầu vào cho trước
  - Ngày mai có mưa không (yes or no)
  - Đầu ra của bộ dự báo:
    - **Binary**: 1 (yes) or 0 (no)
    - **Probability**: p or 1-p (for a 2-class problem)

BÀI TOÁN PHÂN LỚP

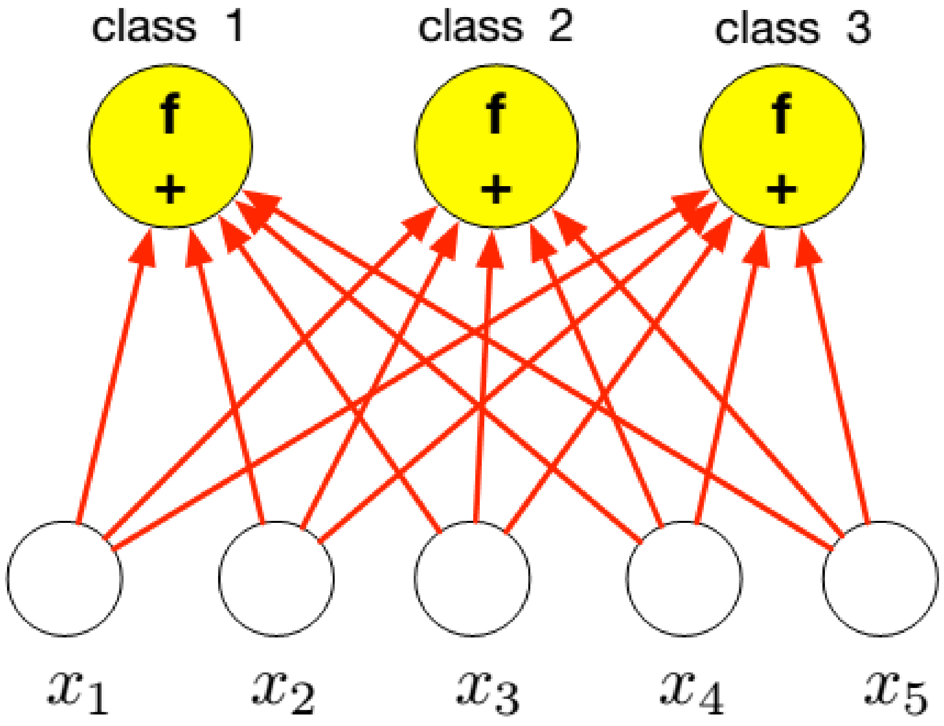


Single-layer network, binary/sigmoid output

Binary (step function): $f(a) = \begin{cases} 1 & \text{if } a \geq 0.5 \\ 0 & \text{if } a < 0.5 \end{cases}$

Probabilistic (sigmoid function): $\boxed{f(a) = \dfrac{1}{1 + \exp(-a)}}$

- **Nếu có bài toán phân loại đa lớp (K), sử dụng one-from-K output coding:**
  - Đầu ra của lớp đúng có giá trị = 1
  - Đầu ra của những lớp còn lại có giá trị = 0

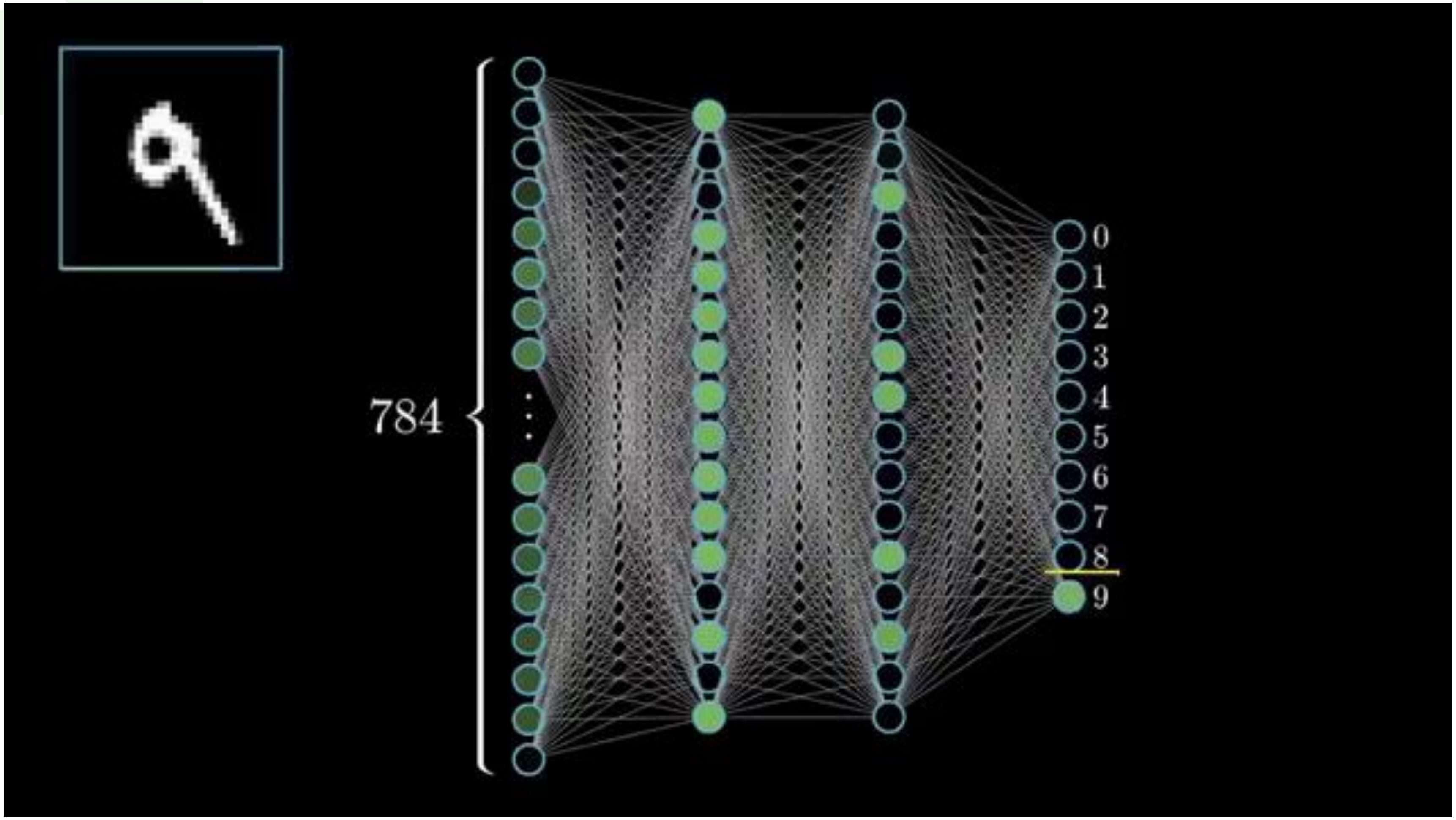


ONE HOT CODE

- **Nếu ta có K lớp, ta sẽ xây dựng K bộ phân lớp = {C1, C2, ..., CK}**
  - C1 : class 1 & not-class 1 (P1 and 1-P1)
  - C2 : class 2 & not-class 2 (P2 and 1-P2)
  - CK : class K & not-class K (PK and 1-PK)
- Như vậy kết luận cuối cùng: xác định class mà điểm rơi vào với xác suất cao nhất, cụ thể là

- Ký hiệu: $\underset{k \in \{1, K\}}{argmax}(P_k)$

MINH HỌA MẠNG PHÂN LOẠI ĐA LỚP



https://www.youtube.com/watch?v=aircAruvnKk

HÀM SOFTMAX

- Ta cần một hàm sao cho mỗi giá trị đầu vào x, a_i thể hiện xác suất mà nó rơi vào class thứ i, với điều kiện:

$$\sum_{k=1}^{K} P(k|x) = 1$$

- Ta thấy nếu $z_i = w^T{}_i x$ càng lớn thì xác suất rơi vào lớp thứ i càng cao, nghĩa là cần một hàm đồng biến.
- Ngoài ra, do z có thể nhận giá trị dương hoặc âm, vì thế để đảm bảo z dương và đồng biến ta cho $\exp(z^i)=e^z$

HÀM SOFTMAX

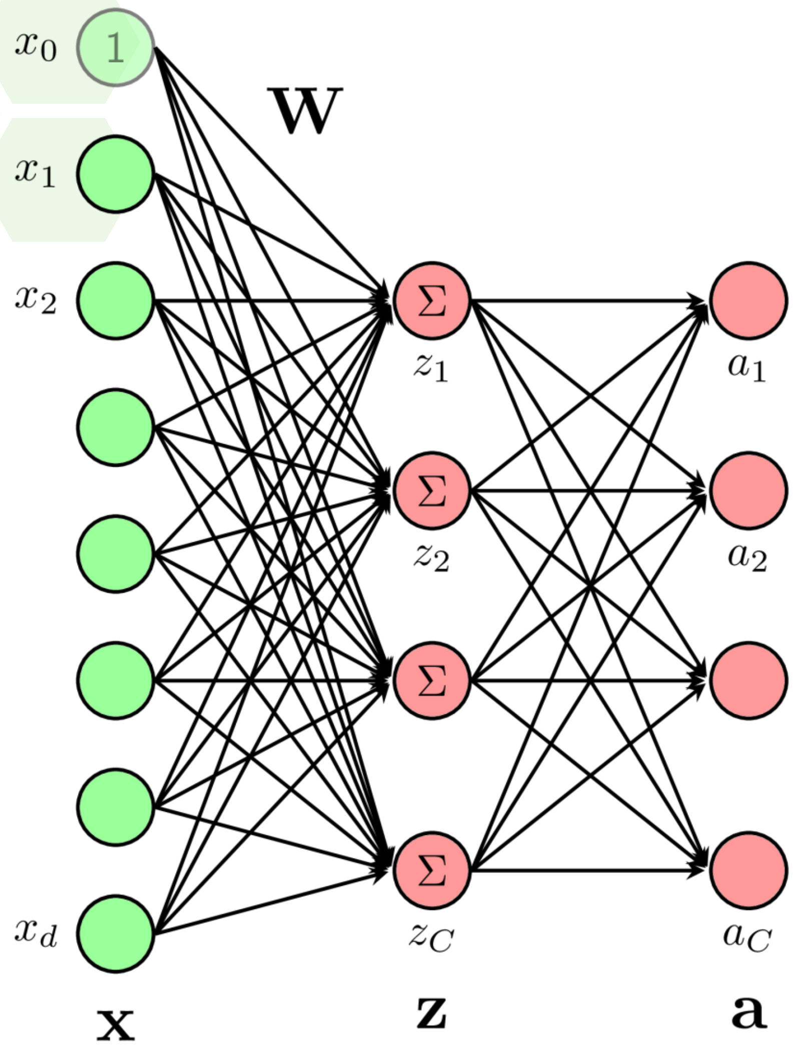
- Nếu ta sử dụng nhiều hàm sigmoid cho bài toán phân loại đa lớp, như vậy:

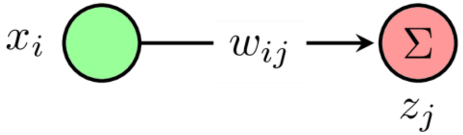
$$\sum_{k=1}^{K} P(k|x) \neq 1$$

- Nếu chúng ta muốn đầu ra của mạng như là xác suất để đầu ra là của một lớp, ta sử dụng một hàm kích hoạt sum-to-one: softmax

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)}, \quad \forall i = 1, 2, \ldots, C$$

HÀM SOFTMAX





w_{0j}: biases, don't forget!

d: data dimension

C: number of classes

$\mathbf{x} \in \mathbb{R}^{d+1}$

$\mathbf{W} \in \mathbb{R}^{(d+1)\times C}$

$z_i = \mathbf{w}_i^T \mathbf{x}$

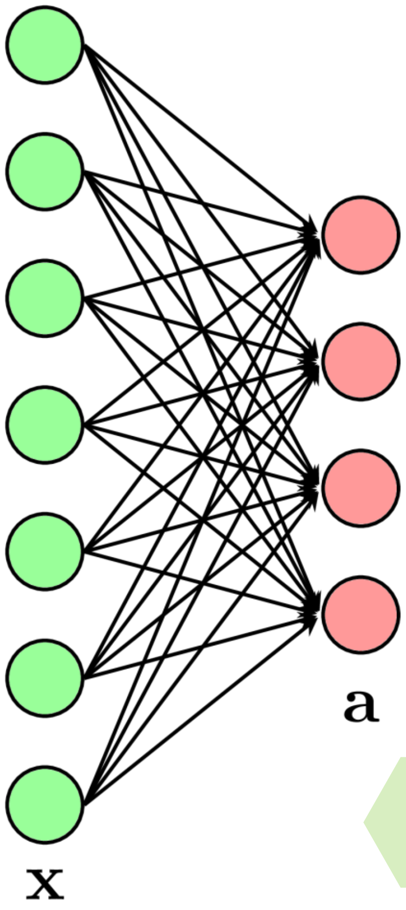
$\mathbf{z} = \mathbf{W}^T \mathbf{x} \in \mathbb{R}^C$

$\mathbf{a} = \text{softmax}(\mathbf{z}) \in \mathbb{R}^C$

$a_i > 0, \quad \sum_{i=1}^{C} a_i = 1$

short form $\longrightarrow$

$\boxed{\mathbf{z} = \text{softmax}(\mathbf{W}^T \mathbf{x})}$



PHIÊN BẢN ỔN ĐỊNH HƠN CỦA SOFTMAX

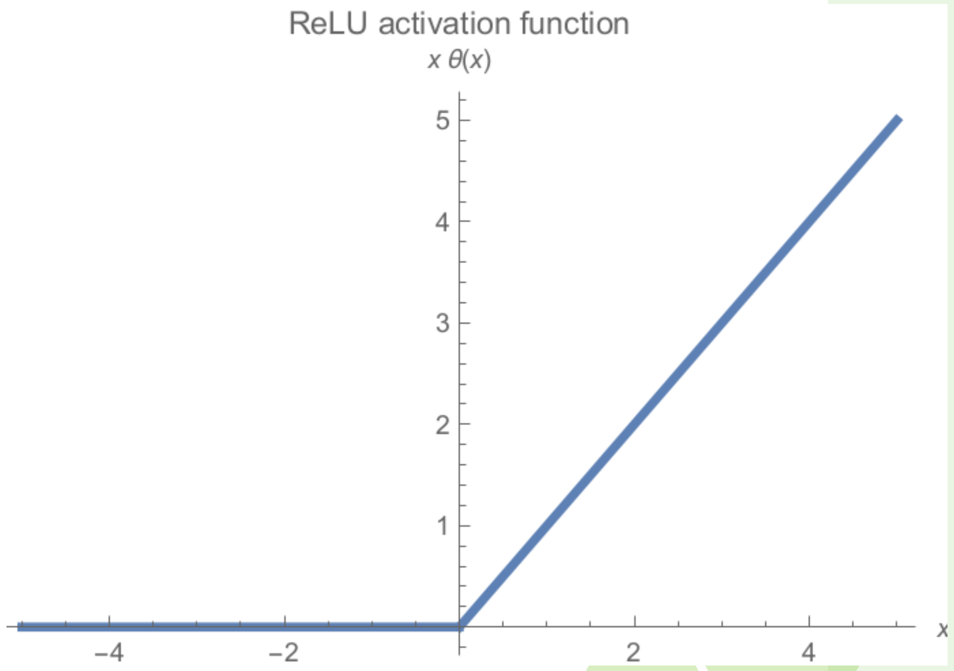
- Khi z_i quá lớn, việc tính $\exp(z_i)$ thường gặp hiện tượng tràn số. Ta có thể khắc phục như sau:

$$\frac{\exp(z_i)}{\sum_{j=1}^{C}\exp(z_j)} = \frac{\exp(-c)\exp(z_i)}{\exp(-c)\sum_{j=1}^{C}\exp(z_j)} = \frac{\exp(z_i - c)}{\sum_{j=1}^{C}\exp(z_j - c)}$$

- với c là một hằng số bất kỳ, thông thường $c = \max(z_i)$

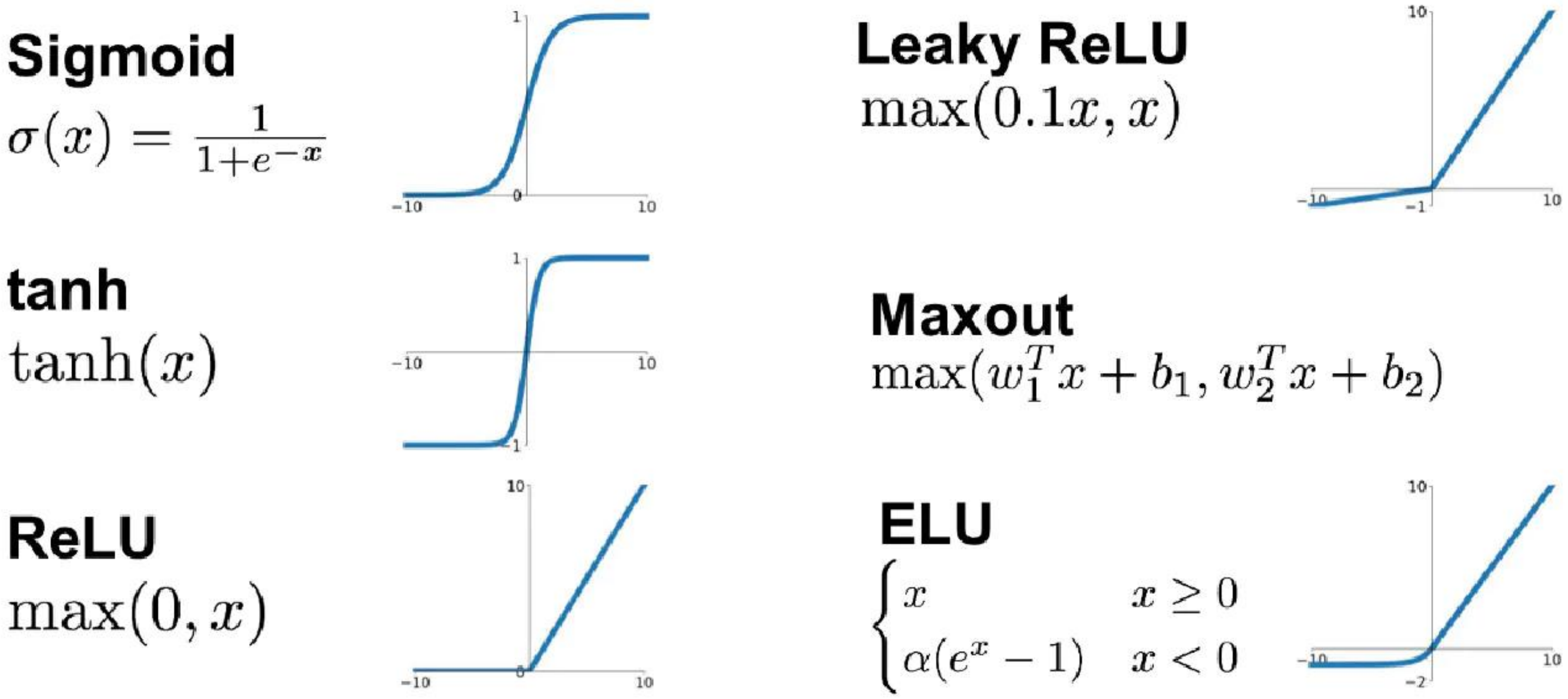
RELU (RECTIFIED LINEAR UNIT)

- **Là hàm sử dụng rộng rãi trong thời gian gần đây do tính đơn giản**
- **Dạng toán học: f(s) = max (0, s)**
- **Các ưu điểm chính:**
  - *Hội tụ nhanh hơn nhiều so với các hàm kích hoạt (tanh) do việc tính toán của hàm cũng như gradient rất nhanh (gradient = 1 nếu s > 0 và = 0 trong trường hợp còn lại*
  - *Mặc dù ReLU không có đạo hàm tại s=0 nhưng vẫn giả thiết là đạo hàm tại đây = 0*



TỔNG KẾT

- Giải thuật học mạng với Gradient Descent
- Giải thuật Stochastic Gradient Descent
- Minibatches, Softmax, ReLU



Thank you !