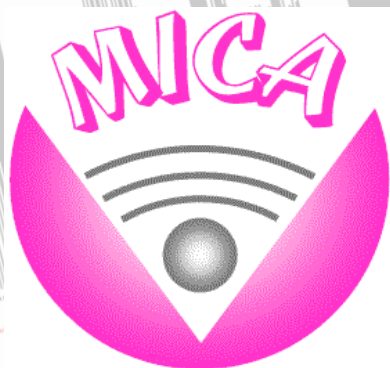




Mạng neuron và ứng dụng trong xử lý tín hiệu



Giảng viên
Trần Thị Thanh Hải

International Research Institute MICA
Multimedia, Information, Communication & Applications
UMI 2954

Hanoi University of Science and Technology
1 Dai Co Viet - Hanoi - Vietnam

Bài 6:

Một số phương pháp huấn luyện mạng



Mục tiêu

■ Bài 5:

- ◆ Đã học các giải thuật để tối ưu hóa hàm mục tiêu
- ◆ Đã đưa ra một số giải pháp để nâng cao chất lượng học

■ 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

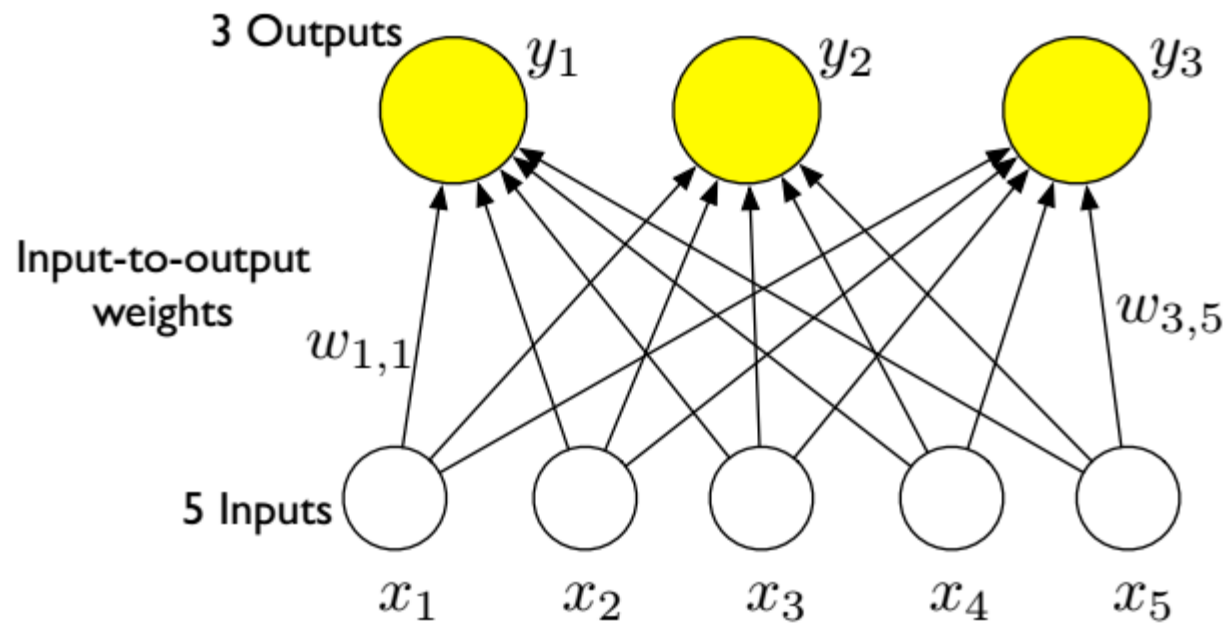
*Source: Steve Renals and Pavlos Andreadis,
Machine Learning Practical — MLP*

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



Mạng một lớp



Mạng một lớp

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

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

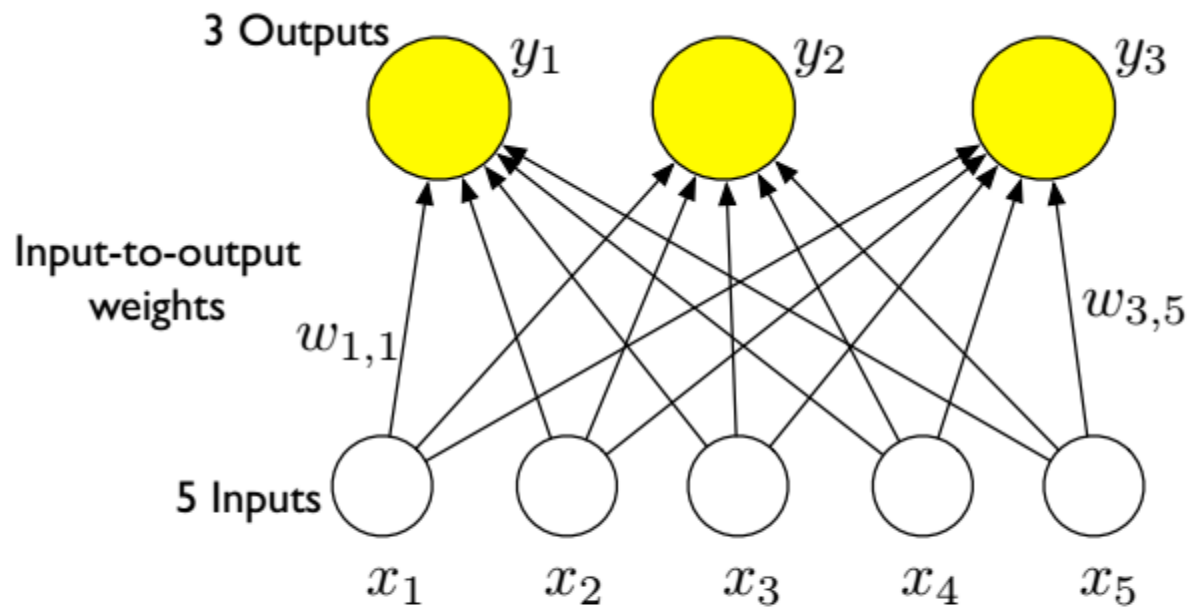
Weight matrix $\mathbf{W}$: w_{ki} is the weight from input x_i to output y_k

Bias b_k is the bias for output k

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



Mạng 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 $\{(\mathbf{x}^n, \mathbf{t}^n) : 1 \leq n \leq N\}$

Target vector $\mathbf{t}^n = (t_1^n, \dots, t_K^n)^T$ – 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:

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

E is the total error averaged over the training set:

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

- Training process: set $\mathbf{W}$ and $\mathbf{b}$ 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



Giải thuật học

- 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:

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

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

Giải thuật học

- Error function:

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

- 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**



Giải thuật SGD

```
1: procedure SGDTRAINING(X, T, W)
2:   initialize W to small random numbers
3:   randomize order of training examples in X
4:   while not converged do
5:     for  $n \leftarrow 1, N$  do
6:       for  $k \leftarrow 1, K$  do
7:          $y_k^n \leftarrow \sum_{i=1}^d w_{ki} x_i^n + b_k$ 
8:          $g_k^n \leftarrow y_k^n - t_k^n$ 
9:         for  $i \leftarrow 1, d$  do
10:           $w_{ki} \leftarrow w_{ki} - \eta \cdot g_k^n \cdot x_i^n$ 
11:        end for
12:         $b_k \leftarrow b_k - \eta \cdot g_k^n$ 
13:      end for
14:    end for
15:  end while
16: end procedure
```

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

Minibatches

- **Minibatch:** mini-batch sử dụng một số lượng $n > 1$ (nhưng $n \ll N$ 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.

Phân lớp và hồi quy

- Bài toán: nhận dạng chữ số viết tay



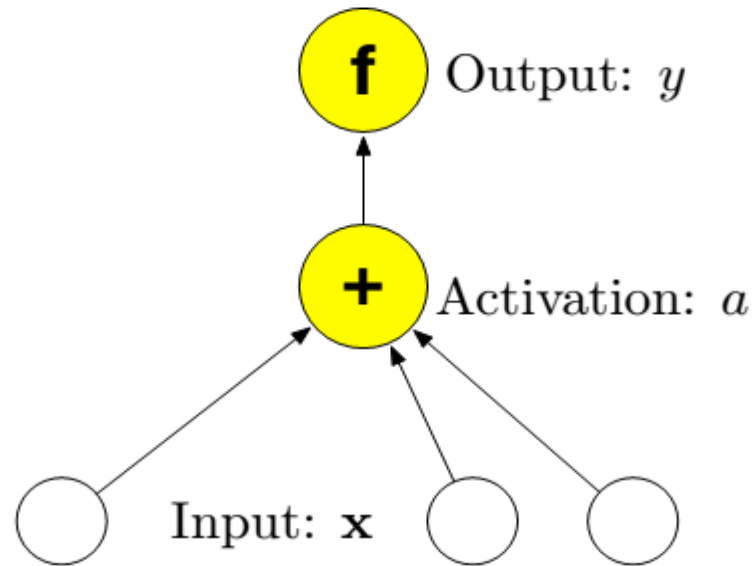
MNIST dataset:

<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

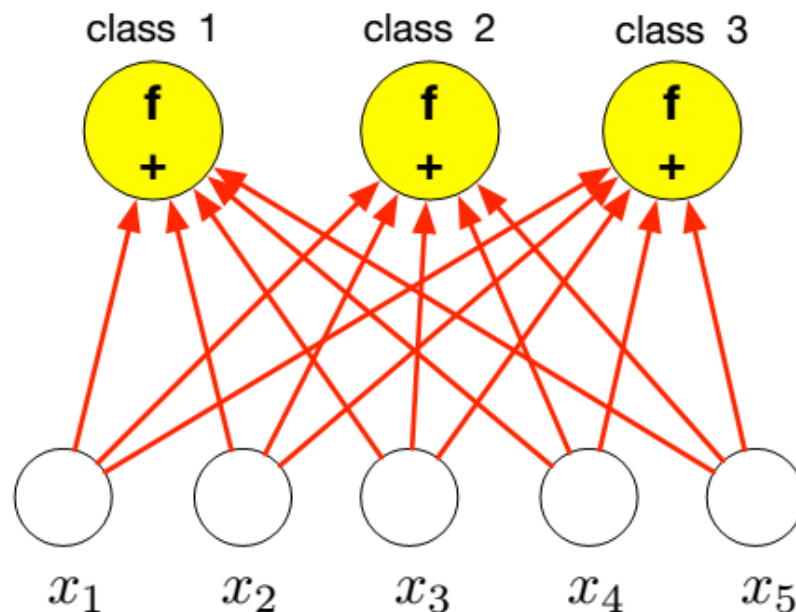
Binary (step function):
$$f(a) = \begin{cases} 1 & \text{if } a \geq 0.5 \\ 0 & \text{if } a < 0.5 \end{cases}$$

Probabilistic (sigmoid function):

$$f(a) = \frac{1}{1 + \exp(-a)}$$

Bài toán phân lớp

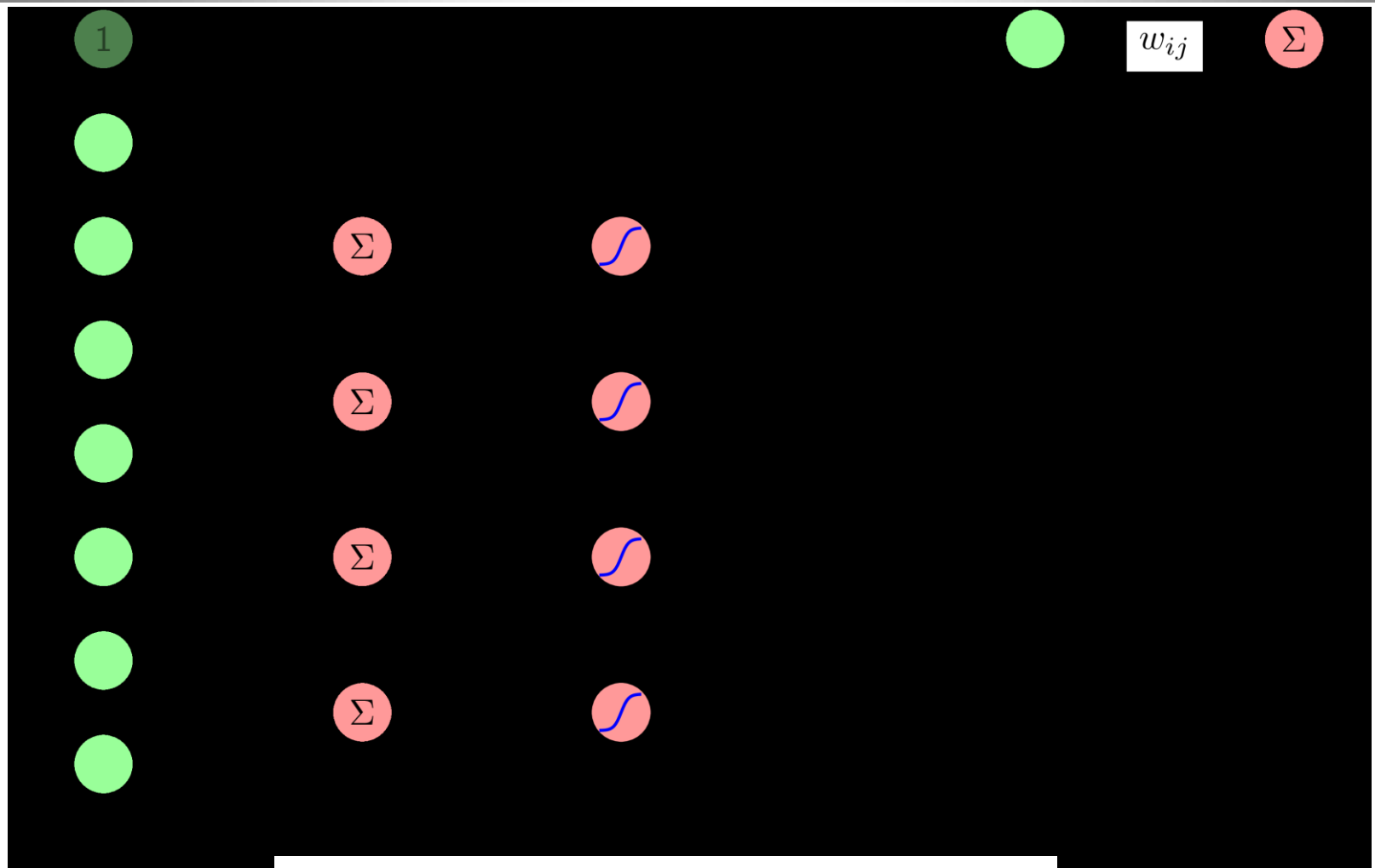
- Nếu có bài toán phân loại đa lớp (K), sử dụng one-from-K output coding:
 - ◆ Đầu ra của lớp đúng có giá trị = 1
 - ◆ Đầu ra của những lớp còn lại có giá trị = 0



One hot code

- Nếu ta có K lớp, ta sẽ xây dựng K bộ phân lớp $\{C_1, C_2, \dots, C_K\}$
 - ◆ C_1 : class 1 & not-class 1 (P_1 and $1-P_1$)
 - ◆ C_2 : class 2 & not-class 2 (P_2 and $1-P_2$)
 - ◆ C_K : class K & not-class K (P_K and $1-P_K$)
- Như vậy kết luận cuối cùng: xác định class mà điểm rơi vào với xác suất cao nhất, cụ thể là
 - $\underset{k \in \{1, K\}}{\operatorname{argmax}} (P_k)$

Minh họa mạng phân loại đa lớp



$$a_i = \text{sigmoid}(z_i) = \text{sigmoid}(\mathbf{w}_i^T \mathbf{x})$$

Hàm Softmax

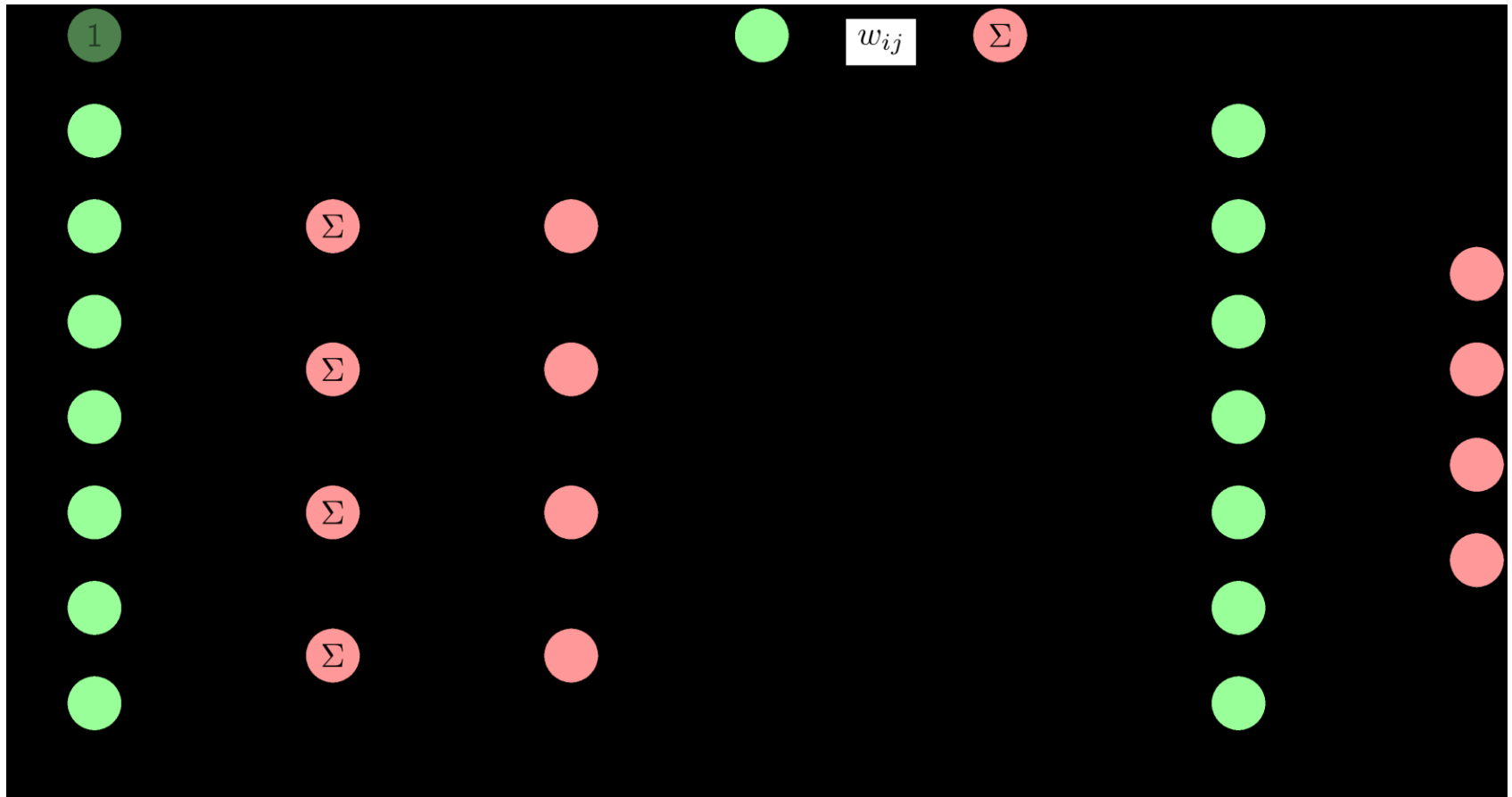
- Ta cần một hàm sao cho mỗi giá trị đầu vào x , a_i thể hiện xác suất mà nó rơi vào class thứ i với điều kiện:
$$\sum_{k=1}^K P(k|x) = 1$$
- Ta thấy nếu $z_i = w_i^T x$ càng lớn thì xác suất rơi vào lớp thứ i càng cao, nghĩa là cần một hàm đồng biến
- Ngoài ra, do z có thể nhận giá trị dương hoặc âm, vì thế để đảm bảo z dương và đồng biến ta cho $\exp(z^i) = e^{z_i}$

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

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

Hàm softmax



Phiên bản ổn định hơn của softmax

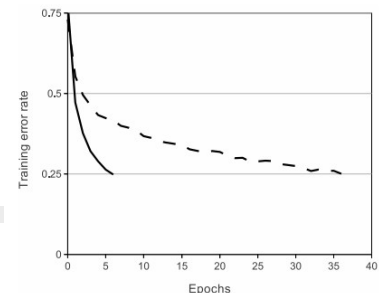
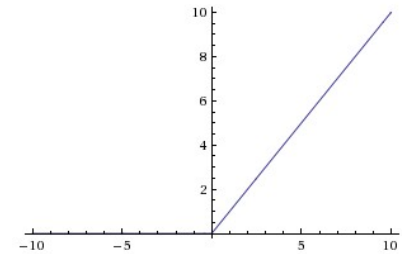
- Khi z_i quá lớn, việc tính $\exp(z_i)$ thường gặp hiện tượng tràn số. Ta có thể khắc phục như sau:

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

- 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



Tài liệu tham khảo

- Neural network design
- Slide môn học Machine Learning Pratical (MLP)
<https://www.inf.ed.ac.uk/teaching/courses/mlp/lectures.html>
- Website: <https://machinelearningcoban.com/>

