HCMUT EE MACHINE LEARNING & IOT LAB

Khóa hè 2025: Python và Machine Learning
Day 13

DEEP LEARNING: MULTILAYER PERCEPTRONS

Presentation By: Trần Huy

Ho Chi Minh City – 2025

Mục tiêu buổi học:

1. Hiểu được bản chất của mạng Neuron Network
2. Biết cách sử dụng hàm kích hoạt
3. Biết cách hoạt đông và cách học của một mạng MLP

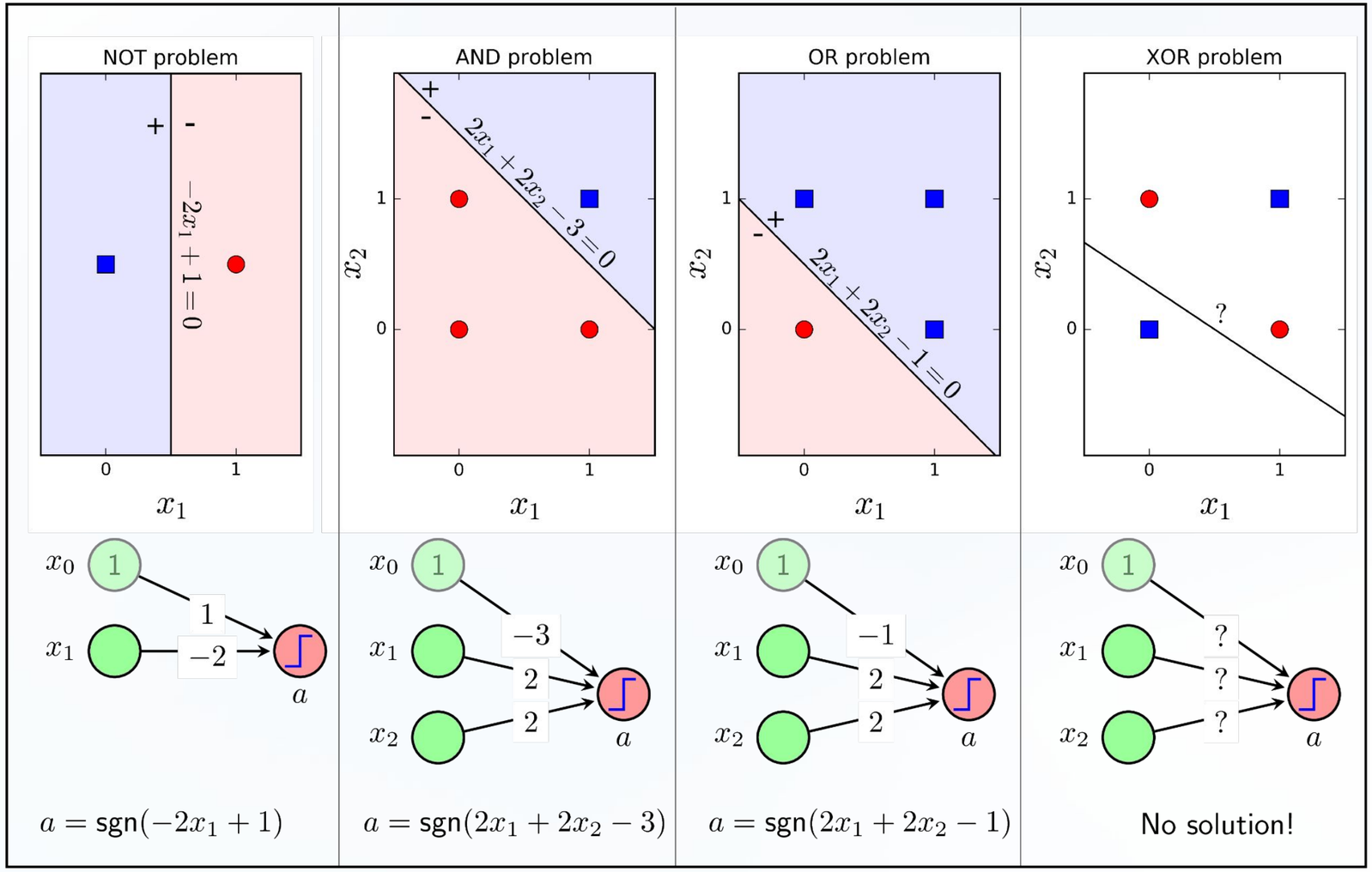
Table of Content

I. TỪ TUYẾN TÍNH ĐẾN PHI TUYẾN

I. Từ tuyến tính đến phi tuyến

Mô hình tuyến tính và phi tuyến

Các bài toán OR, AND, NOT có thể được biểu diễn bởi 1 PLA là một hàm tuyến tính.



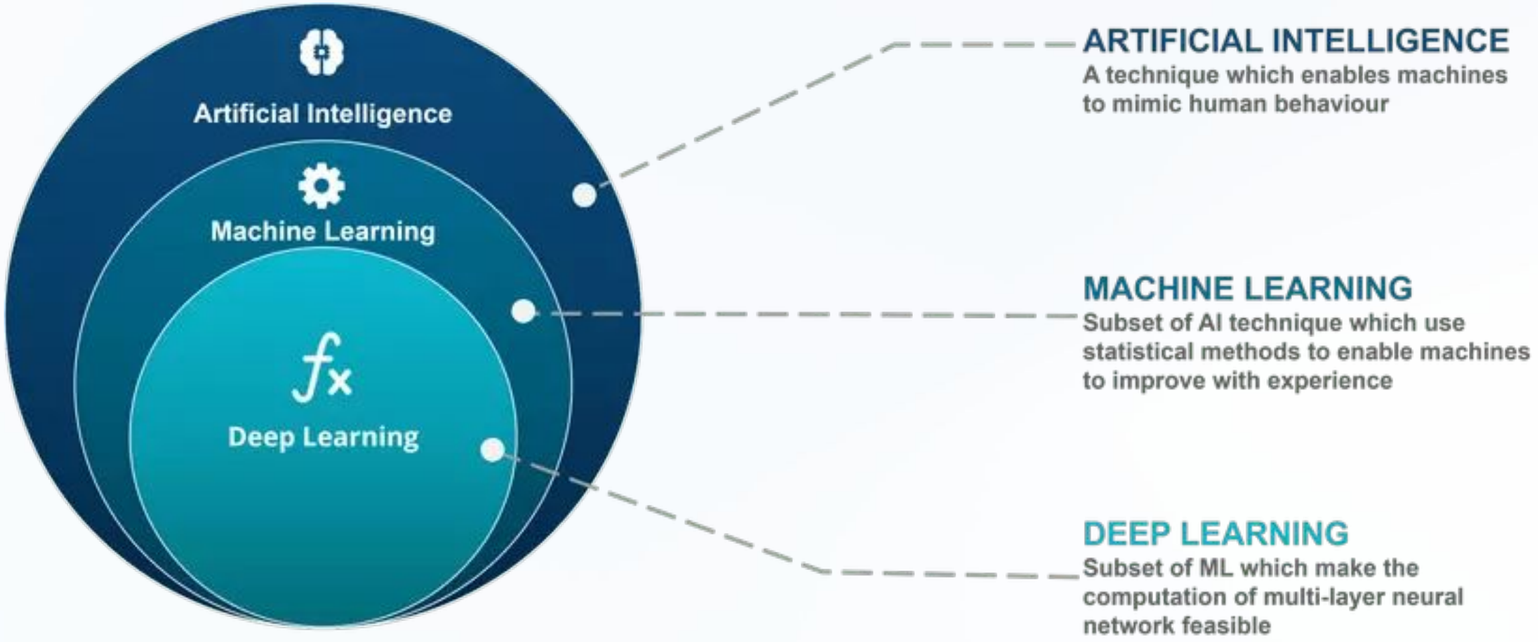
I. Từ tuyến tính đến phi tuyến

Machine Learning và Deep Learning

Deep Learning = Machine Learning + mô hình học đặc trưng sâu, thường là neural networks nhiều tầng (deep neural networks)

Đặc trưng:

- Nhiều lớp (layers): tự học feature
- Mô hình có thể học phi tuyến mạnh
- Tối ưu bằng gradient descent và backpropagation



II. Mạng Neuron sinh học và nhân tạo

II. Mạng Neuron sinh học và nhân tạo

Neuron sinh học và thần kinh con người

Neuron là tế bào thần kinh, đơn vị xử lý thông tin cơ bản trong não người.

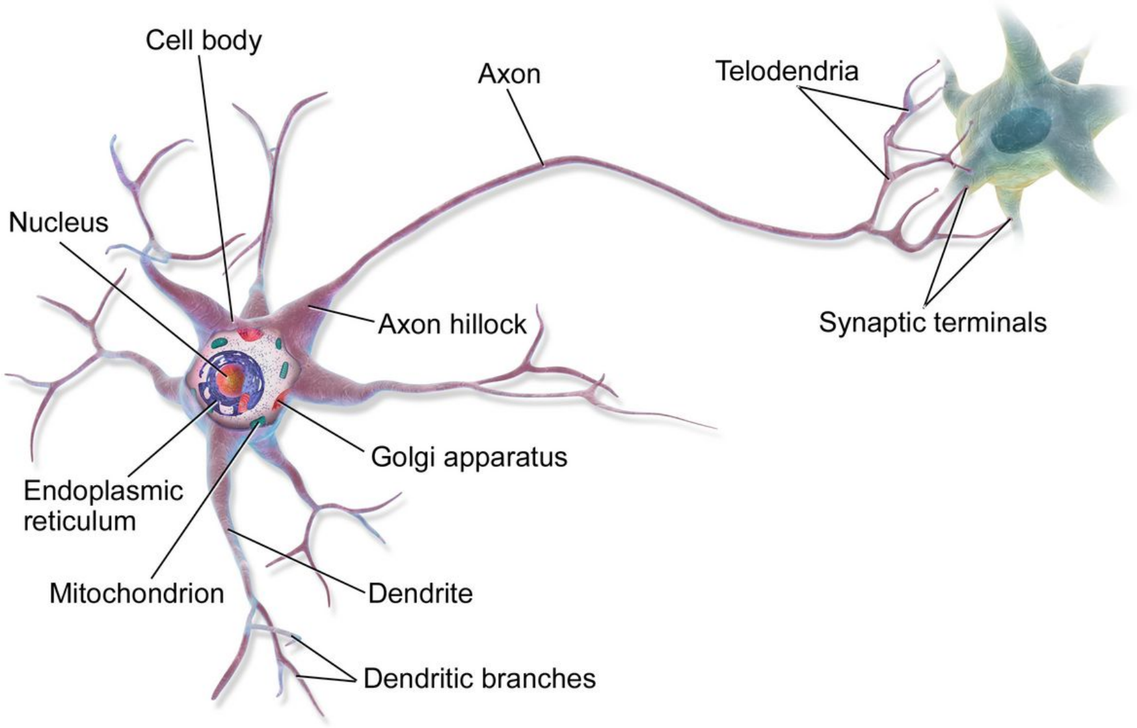
Mỗi neuron có 3 phần chính:

- Dendrites (nhánh vào) – nhận tín hiệu từ các neuron khác
- Cell body (soma) – xử lý thông tin
- Axon (nhánh ra) – truyền tín hiệu đi

Một neuron có thể kết nối với hàng ngàn neuron khác ⇒ mạng lưới siêu phức tạp.

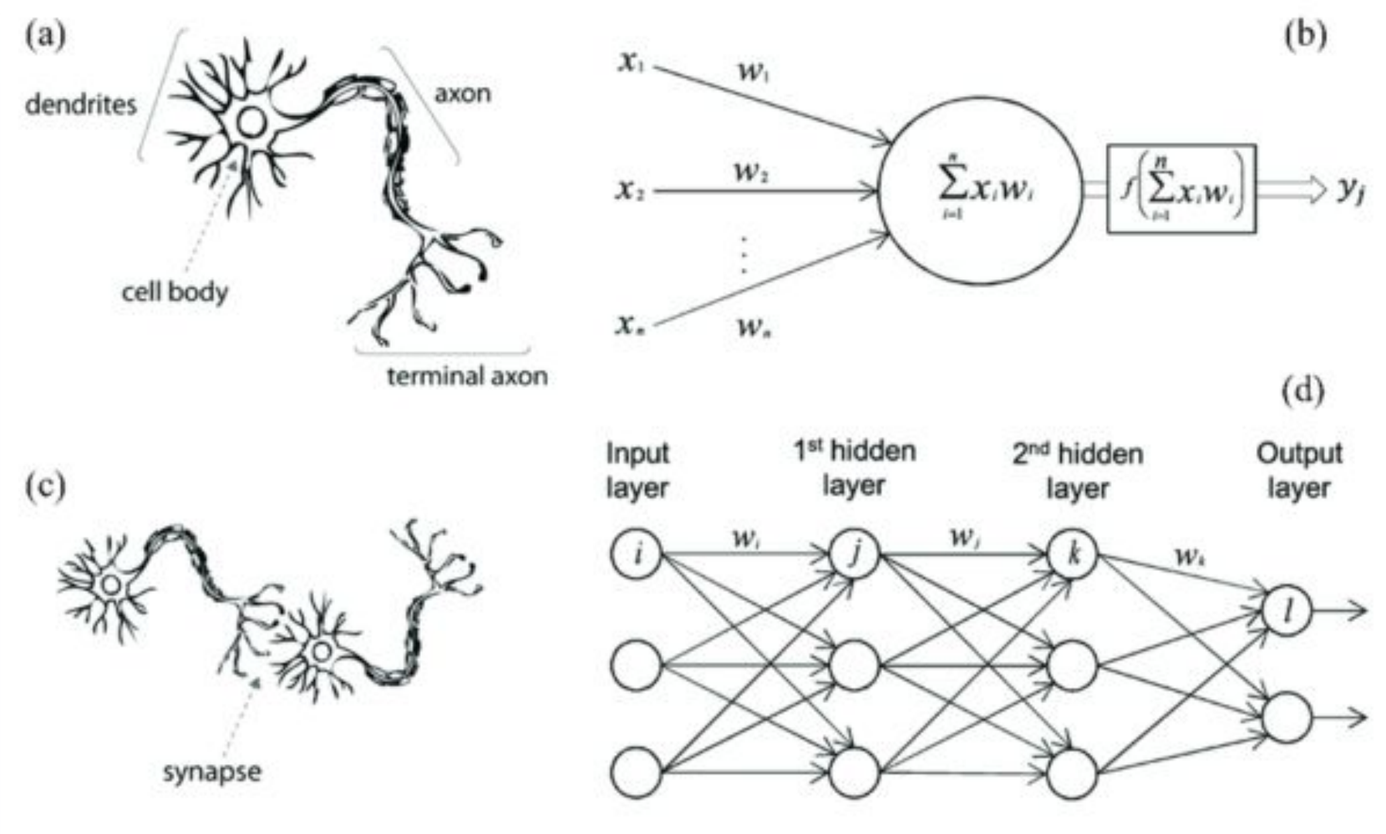
Mỗi tín hiệu đầu vào có cường độ (giống trọng số).

Nếu tổng tín hiệu vào đủ mạnh (vượt ngưỡng), neuron sẽ phát tín hiệu ra ⇒ gọi là kích hoạt (firing).



II. Mạng Neuron sinh học và nhân tạo

Mạng Neuron Nhân tạo



II. Mạng Neuron sinh học và nhân tạo

Mạng Neuron Nhân tạo

Năm	Sự kiện / Công trình	Ý nghĩa & Chi tiết	Nhân vật chính
1943	Mô hình neuron logic của McCulloch & Pitts	Mô tả hoạt động của neuron bằng hàm nhị phân. Neuron phát tín hiệu nếu tổng tín hiệu vào vượt ngưỡng.→ Mở đầu cho ý tưởng mô phỏng não bộ bằng logic toán học.	Warren McCulloch, Walter Pitts
1958	Perceptron – Neuron đơn giản biết học	Perceptron là mô hình học tuyến tính đơn giản có thể điều chỉnh trọng số dựa trên sai số. Ứng dụng đầu tiên trong nhận dạng hình ảnh cơ bản (văn bản).	Frank Rosenblatt
1969	Cuốn "Perceptrons" chỉ ra giới hạn của mạng đơn lớp	Minsky & Papert chứng minh Perceptron không thể giải bài toán phi tuyến như XOR. Gây mất niềm tin vào ANN trong cộng đồng AI.→ Giai đoạn "AI Winter" bắt đầu.	Marvin Minsky, Seymour Papert
1986	Thuật toán Backpropagation – học được mạng nhiều lớp (MLP)	Lan truyền ngược giúp cập nhật trọng số trong các lớp ẩn. Lần đầu tiên khả thi hóa việc huấn luyện mạng nhiều lớp.→ Mở ra thời kỳ đầu của Deep Learning hiện đại.	David Rumelhart, Geoffrey Hinton, Ronald Williams
1990s – 2000s	Mạng neuron phát triển chậm, bị lu mờ bởi các mô hình khác (SVM, RF, etc.)	Hạn chế phần cứng, dữ liệu chưa đủ lớn, thời gian huấn luyện dài → ANN ít được ứng dụng thực tế. Nhưng vẫn tồn tại trong nghiên cứu học sâu.	–
2012	Chiến thắng ImageNet của AlexNet – Cột mốc Deep Learning bùng nổ	Mạng CNN 8 lớp học trực tiếp từ ảnh gốc, vượt xa các mô hình cổ điển. Sử dụng GPU để huấn luyện. Dẫn đến cuộc cách mạng Deep Learning trong thị giác máy tính, NLP, v.v.	Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton
2015+	Bùng nổ các kiến trúc mới: ResNet, GAN, RNN, Transformer...	DL trở thành nền tảng của AI hiện đại. Các công ty công nghệ lớn đầu tư mạnh. Các kiến trúc sâu, rộng, và tối ưu được triển khai trên quy mô lớn.	Nhiều nhà nghiên cứu, tập thể cộng đồng AI

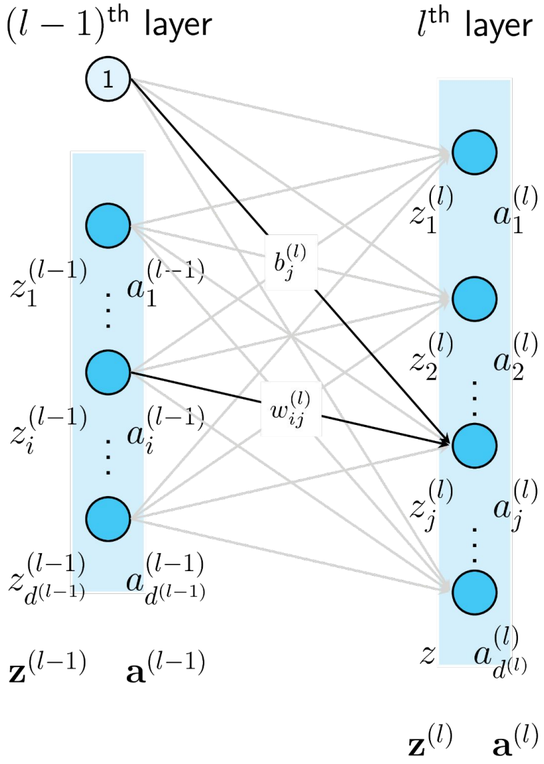
III.MultiLayer Perceptrons

III. MultiLayer Perceptrons

Các khái niệm cơ bản

Layers

Ngoài *Input layers* và *Output layers*, một Multi-layer Perceptron (MLP) có thể có nhiều *Hidden layers* ở giữa. Các *Hidden layers* theo thứ tự từ input layer đến output layer được đánh số thứ thự là *Hidden layer 1*, *Hidden layer 2*, ...



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

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

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

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

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

III. MultiLayer Perceptrons

Các khái niệm cơ bản

Có L ma trận trọng số cho một MLP có L layers. Các ma trận này được ký hiệu là $\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}, l = 1, 2, \ldots, L$ trong đó $\mathbf{W}^{(l)}$ thể hiện các *kết nối* từ layer thứ $l-1$ tới layer thứ l (nếu ta coi input layer là layer thứ 0). Cụ thể hơn, phần tử $w_{ij}^{(l)}$ thể hiện kết nối từ node thứ i của layer thứ $(l-1)$ tới node từ j của layer thứ (l). Các biases của layer thứ (l) được ký hiệu là $\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)}}$.

Giả sử chúng ta có một mạng nơ-ron đơn giản:

- **Đầu vào**: $x = [x_1, x_2]^T$, với $x_1 = 1, x_2 = 2$.
- **Tầng ẩn**: Có 2 đơn vị ẩn ($h = 2$), với ma trận trọng số:

$$\mathbf{W}_h = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.3 \\ 0.2 & 0.4 \end{bmatrix}$$

- **Hàm kích hoạt**: ReLU, $\sigma(z) = \max(z, 0)$, áp dụng từng phần tử.
- **Tầng đầu ra**: Có 1 đơn vị ($q = 1$), với vector trọng số:

$$\mathbf{W}_o = [w_{o1}, w_{o2}]^T = [0.6, 0.7]^T$$

- **Hàm mất mát**: Hàm mất mát bình phương, $l(o, y) = \frac{1}{2}(o - y)^2$, với nhãn thực tế $y = 1$.

III. MultiLayer Perceptrons

Lý thuyết MLP

MLP bao gồm nhiều tầng (layers) được xếp chồng lên nhau:

- Tầng đầu vào: Chứa các đặc trưng đầu vào (ví dụ, giá trị pixel của hình ảnh).
- Tầng ẩn: Một hoặc nhiều tầng xử lý dữ liệu thông qua các phép biến đổi affine và hàm kích hoạt phi tuyến.
- Tầng đầu ra: Tạo ra kết quả cuối cùng (ví dụ, xác suất phân loại).

Mỗi tầng được kết nối đầy đủ (fully connected), nghĩa là mọi đầu vào ảnh hưởng đến mọi đơn vị trong tầng ẩn, và mỗi đơn vị ẩn ảnh hưởng đến mọi đầu ra.

IV. Hàm kích hoạt

IV. Hàm kích hoạt

Tại sao cần có hàm kích hoạt?

Trong một MLP với một tầng ẩn, quá trình tính toán diễn ra như sau:

- **Biểu diễn ẩn**: Tầng đầu vào được biến đổi thông qua một phép biến đổi affine để tạo ra các **biểu diễn ẩn** (hidden representations):

$$H = XW_h + b_h$$

- **Đầu ra**: Các biểu diễn ẩn được biến đổi tiếp để tạo đầu ra:

$$O = HW_o + b_o$$

Nếu không có hàm kích hoạt phi tuyến, MLP chỉ đơn thuần là một chuỗi các phép biến đổi affine, có thể được gộp lại thành một phép biến đổi affine duy nhất:

$$O = (XW_h + b_h)W_o + b_o = X(W_hW_o) + (b_hW_o + b_o)$$

Điều này tương đương với một mô hình tuyến tính đơn tầng, không mang lại lợi ích gì thêm.

IV. Hàm kích hoạt

Hàm ReLU (Rectified Linear Unit)

Định nghĩa:

$$ReLU(x)=max(x,0)$$

Hàm ReLU giữ nguyên các giá trị dương và đặt các giá trị âm về 0. Đây là hàm kích hoạt đơn giản nhưng hiệu quả, được sử dụng rộng rãi nhờ tính dễ triển khai và hiệu suất tốt trong nhiều tác vụ.
Đặc điểm:

- Đồ thị: Hàm ReLU là một hàm tuyến tính phân đoạn (piecewise linear), với giá trị bằng 0 khi $x<0$ và bằng x khi $x\geq0$.
- Đạo hàm:
  - Nếu $x>0$, đạo hàm là 1.
  - Nếu $x<0$, đạo hàm là 0.
  - Tại $x=0$, hàm không khả vi, nhưng theo quy ước, đạo hàm được lấy là 0 (đạo hàm bên trái).
- Lợi ích: Đạo hàm của ReLU rất đơn giản (0 hoặc 1), giúp tối ưu hóa tốt hơn và tránh vấn đề vanishing gradients (đạo hàm biến mất) – một vấn đề phổ biến trong các mạng nơ-ron trước đây.

IV. Hàm kích hoạt

Hàm Sigmoid

- **Định nghĩa**:

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

Hàm sigmoid chuyển đổi đầu vào từ $(-\infty, \infty)$ thành khoảng $(0, 1)$, thường được gọi là hàm "nén" (squashing function).

- **Đặc điểm**:
  - **Đồ thị**: Hàm sigmoid có dạng chữ S, tiếp cận 0 khi $x \to -\infty$ và 1 khi $x \to \infty$. Khi x gần 0, nó gần giống một phép biến đổi tuyến tính.
  - **Ứng dụng**: Thường được sử dụng trong tầng đầu ra của các bài toán phân loại nhị phân để biểu diễn xác suất, hoặc trong các mạng nơ-ron hồi tiếp (recurrent neural networks) để kiểm soát luồng thông tin.
  - **Đạo hàm**:

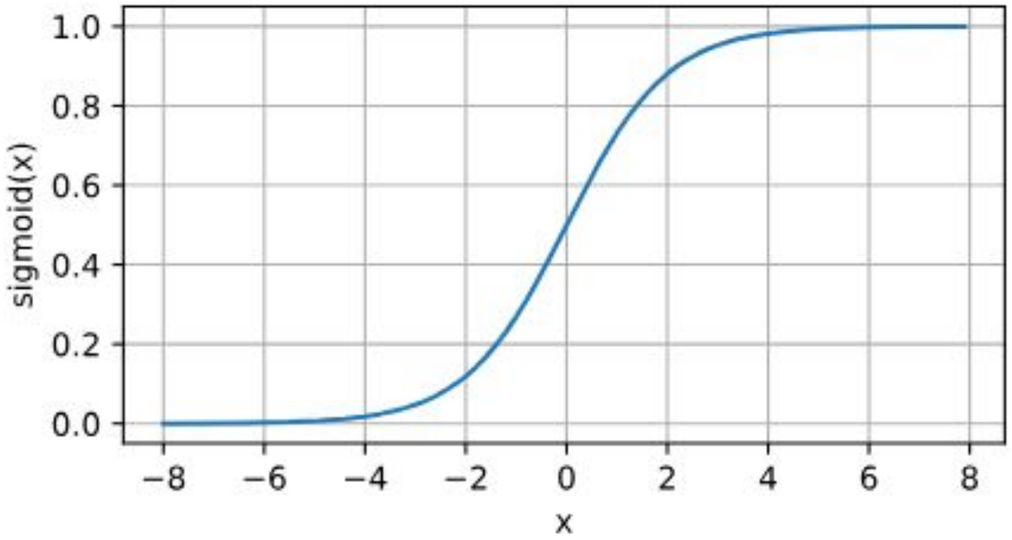
$$\frac{d}{dx}\text{sigmoid}(x) = \text{sigmoid}(x) \cdot (1 - \text{sigmoid}(x))$$

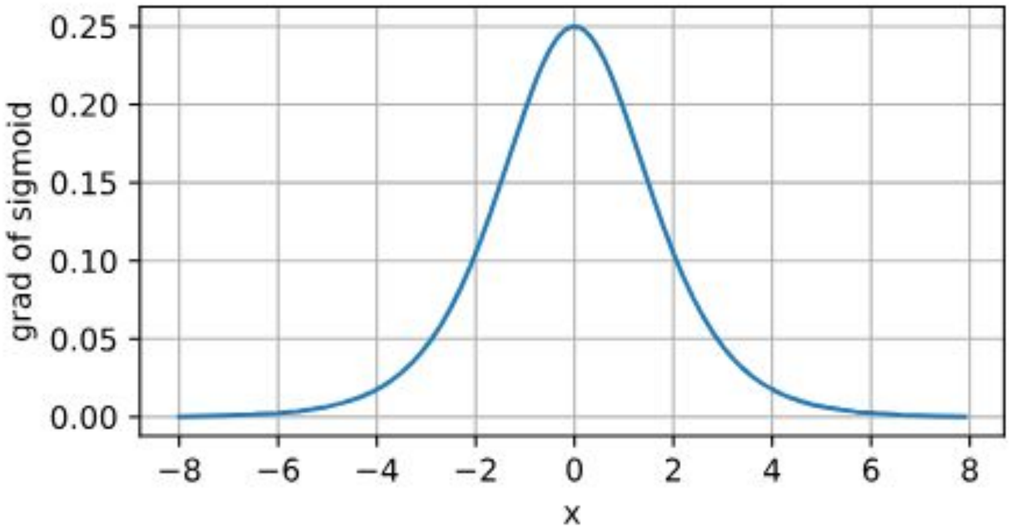
Đạo hàm đạt giá trị tối đa 0.25 tại $x = 0$, và tiến gần về 0 khi x lớn hoặc nhỏ, dẫn đến vấn đề **vanishing gradients** trong quá trình tối ưu hóa.

- **Hạn chế**: Do đạo hàm nhỏ ở các giá trị lớn hoặc nhỏ, sigmoid có thể gây ra hiện tượng "tắc nghẽn" trong quá trình huấn luyện, đặc biệt ở các mạng sâu.

IV. Hàm kích hoạt

Hàm Sigmoid





IV. Hàm kích hoạt

Hàm Tanh

- **Định nghĩa**:

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

Hàm tanh chuyển đổi đầu vào thành khoảng $(-1, 1)$, tương tự như sigmoid nhưng đối xứng quanh gốc tọa độ.

- **Đặc điểm**:
  - **Đồ thị**: Tương tự sigmoid, nhưng giá trị đầu ra nằm trong $(-1, 1)$. Khi x gần 0, hàm gần giống một phép biến đổi tuyến tính.
  - **Đạo hàm**:

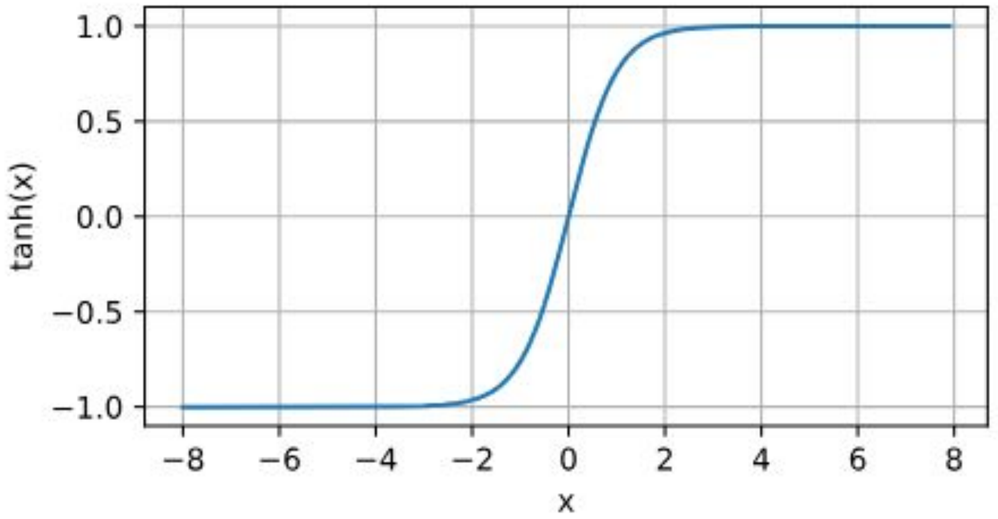
$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$

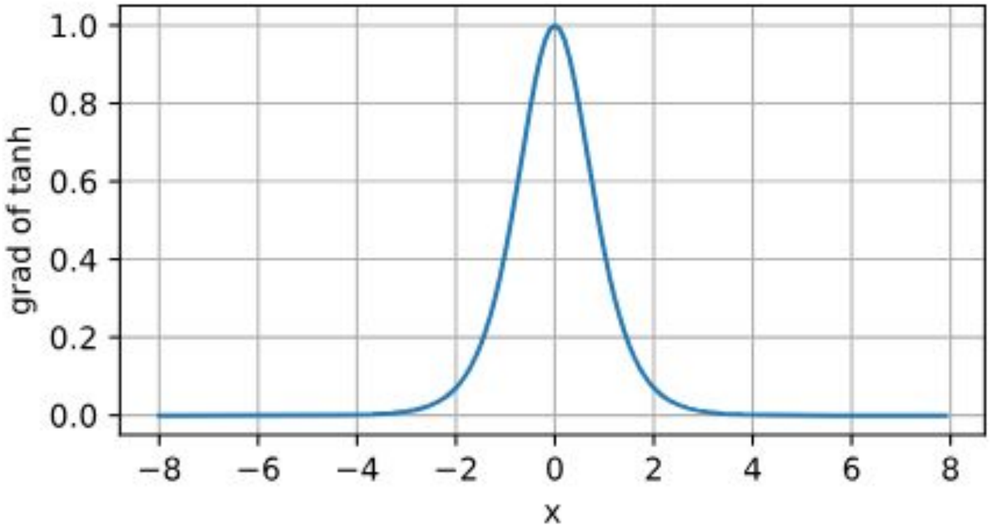
Đạo hàm đạt tối đa 1 tại $x = 0$, và tiến gần về 0 khi x lớn hoặc nhỏ, tương tự như sigmoid.

  - **Lợi ích**: Tanh thường hoạt động tốt hơn sigmoid trong các tầng ẩn vì đầu ra tập trung quanh 0, giúp cân bằng các giá trị trong quá trình huấn luyện.
  - **Hạn chế**: Vẫn gặp vấn đề vanishing gradients ở các giá trị lớn hoặc nhỏ.

IV. Hàm kích hoạt

Hàm Tanh





V. Huấn luyện mạng MLP

V. Huấn luyện mạng MLP

Tổng quan

- ***Lan truyền xuôi (Forward Propagation):*** Là quá trình tính toán và lưu trữ các biến trung gian (bao gồm đầu ra) của mạng nơ-ron, bắt đầu từ tầng đầu vào đến tầng đầu ra. Nó xác định đầu ra của mô hình dựa trên đầu vào và các tham số hiện tại (trọng số và bias).
- ***Lan truyền ngược (Backpropagation):*** Là phương pháp tính gradient của các tham số mạng nơ-ron (trọng số và bias) bằng cách đi ngược từ tầng đầu ra về tầng đầu vào, sử dụng quy tắc chuỗi (chain rule) trong giải tích. Gradient này được dùng để cập nhật các tham số trong quá trình tối ưu hóa (ví dụ, sử dụng gradient descent).
- ***Đồ thị tính toán (Computational Graphs):*** Là một cách biểu diễn trực quan các phép tính trong mạng nơ-ron, trong đó các nút biểu diễn các biến (như đầu vào, trọng số, đầu ra) và các cạnh biểu diễn các phép toán (như nhân ma trận, hàm kích hoạt). Đồ thị giúp hình dung luồng dữ liệu trong lan truyền xuôi và cách tính gradient trong lan truyền ngược.
- Tầm quan trọng của tự động hóa gradient (Automatic Differentiation): Trước đây, việc tính đạo hàm của các mô hình phức tạp phải thực hiện thủ công, rất tốn thời gian và dễ sai sót. Các framework học sâu hiện đại (như PyTorch, TensorFlow) tự động tính gradient, giúp đơn giản hóa việc triển khai các thuật toán học sâu.

V. Huấn luyện mạng MLP

Forward Propagation

1. **Tính biểu diễn ẩn trước kích hoạt**:

$$\mathbf{z} = \mathbf{W}_h^T \mathbf{x} + \mathbf{b}_h$$

Trong đó: - $\mathbf{x} \in R^d$: Vector đầu vào. - $\mathbf{W}_h \in R^{d \times h}$: Ma trận trọng số của tầng ẩn. - $\mathbf{b}_h \in R^h$: Vector bias của tầng ẩn. - $\mathbf{z} \in R^h$: Vector trung gian trước kích hoạt.

2. **Áp dụng hàm kích hoạt**:

$$\mathbf{h} = \sigma(\mathbf{z})$$

Trong đó σ là hàm kích hoạt (ví dụ: ReLU, $\sigma(z) = \max(0, z)$), được áp dụng từng phần tử.

3. **Tính đầu ra**:

$$\mathbf{o} = \mathbf{W}_o^T \mathbf{h} + \mathbf{b}_o$$

Trong đó: - $\mathbf{W}_o \in R^{h \times q}$: Ma trận trọng số của tầng đầu ra. - $\mathbf{b}_o \in R^q$: Vector bias của tầng đầu ra. - $\mathbf{o} \in R^q$: Vector đầu ra.

4. **Tính hàm mất mát**:

$$L = l(\mathbf{o}, y)$$

Trong đó l là hàm mất mát (ví dụ: $l(\mathbf{o}, y) = \frac{1}{2}\|\mathbf{o} - y\|^2$ cho bài toán hồi quy).

V. Huấn luyện mạng MLP

Backward Propagation

Mục tiêu là tính gradient của hàm mục tiêu $J = l(\mathbf{o}, y)$ đối với các tham số $\mathbf{W}_h$, $\mathbf{b}_h$, $\mathbf{W}_o$, và $\mathbf{b}_o$.

1. **Gradient của hàm mục tiêu đối với đầu ra o:**

$$\frac{\partial J}{\partial \mathbf{o}} = \frac{\partial l(\mathbf{o}, y)}{\partial \mathbf{o}}$$

Trong đó $\frac{\partial l}{\partial \mathbf{o}}$ phụ thuộc vào dạng của hàm mất mát (ví dụ: nếu $l(\mathbf{o}, y) = \frac{1}{2}\|\mathbf{o} - y\|^2$, thì $\frac{\partial l}{\partial \mathbf{o}} = \mathbf{o} - y$).

2. **Gradient đối với trọng số tầng đầu ra $\mathbf{W}_o$:**

$$\frac{\partial J}{\partial \mathbf{W}_o} = \frac{\partial J}{\partial \mathbf{o}} \cdot \frac{\partial \mathbf{o}}{\partial \mathbf{W}_o} = \frac{\partial l}{\partial \mathbf{o}} \mathbf{h}^T$$

Trong đó $\frac{\partial \mathbf{o}}{\partial \mathbf{W}_o} = \mathbf{h}$, vì $\mathbf{o} = \mathbf{W}_o^T \mathbf{h} + \mathbf{b}_o$.

3. **Gradient đối với bias tầng đầu ra $\mathbf{b}_o$:**

$$\frac{\partial J}{\partial \mathbf{b}_o} = \frac{\partial J}{\partial \mathbf{o}} \cdot \frac{\partial \mathbf{o}}{\partial \mathbf{b}_o} = \frac{\partial l}{\partial \mathbf{o}}$$

Vì $\frac{\partial \mathbf{o}}{\partial \mathbf{b}_o} = 1$.

4. **Gradient đối với đầu ra tầng ẩn h:**

$$\frac{\partial J}{\partial \mathbf{h}} = \frac{\partial J}{\partial \mathbf{o}} \cdot \frac{\partial \mathbf{o}}{\partial \mathbf{h}} = \frac{\partial l}{\partial \mathbf{o}} \mathbf{W}_o$$

Vì $\frac{\partial \mathbf{o}}{\partial \mathbf{h}} = \mathbf{W}_o$.

5. **Gradient đối với biến trung gian trước kích hoạt z:**

$$\frac{\partial J}{\partial \mathbf{z}} = \frac{\partial J}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{z}} = \frac{\partial J}{\partial \mathbf{h}} \odot \sigma'(\mathbf{z})$$

Trong đó $\odot$ là phép nhân từng phần tử, và $\sigma'(\mathbf{z})$ là đạo hàm của hàm kích hoạt σ.

6. **Gradient đối với trọng số tầng ẩn $\mathbf{W}_h$:**

$$\frac{\partial J}{\partial \mathbf{W}_h} = \frac{\partial J}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{W}_h} = \frac{\partial J}{\partial \mathbf{z}} \mathbf{x}^T$$

Vì $\frac{\partial \mathbf{z}}{\partial \mathbf{W}_h} = \mathbf{x}$.

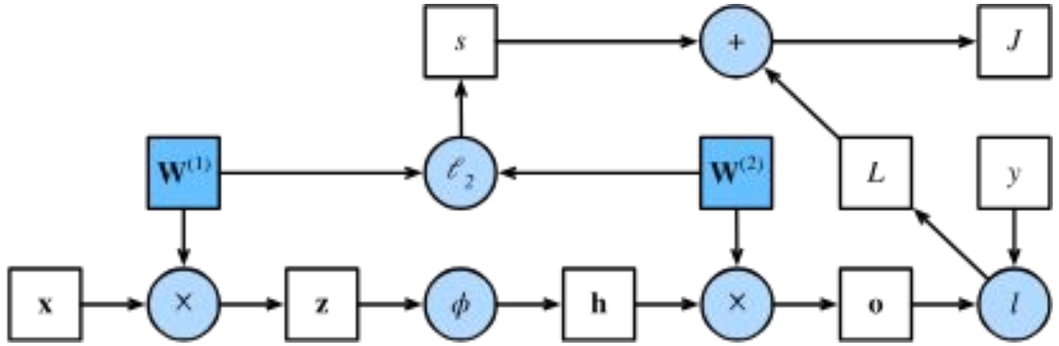
7. **Gradient đối với bias tầng ẩn $\mathbf{b}_h$:**

$$\frac{\partial J}{\partial \mathbf{b}_h} = \frac{\partial J}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{b}_h} = \frac{\partial J}{\partial \mathbf{z}}$$

Vì $\frac{\partial \mathbf{z}}{\partial \mathbf{b}_h} = 1$.

V. Huấn luyện mạng MLP

Backward Propagation



Đồ thị tính toán biểu diễn các bước trên một cách trực quan:

- Nút: Đại diện cho các biến (như x,h,o,L,s,J).
- Vòng tròn: Đại diện cho các phép toán (như nhân ma trận, hàm kích hoạt, hàm mất mát).
- Mũi tên: Chỉ hướng luồng dữ liệu từ đầu vào đến đầu ra (hướng từ trái sang phải và từ dưới lên trên).

Đồ thị này giúp hình dung cách các biến phụ thuộc lẫn nhau trong quá trình lan truyền xuôi.

V. Huấn luyện mạng MLP

Quá trình huấn luyện

- Trong lan truyền xuôi, các biến trung gian ($\mathbf{h}, \mathbf{o}, L, s$) được tính và lưu trữ dựa trên các tham số hiện tại ($\mathbf{W}_h, \mathbf{W}_o$).
- Trong lan truyền ngược, các gradient ($\frac{\partial J}{\partial \mathbf{W}_h}, \frac{\partial J}{\partial \mathbf{W}_o}$) được tính dựa trên các biến trung gian từ lan truyền xuôi.
- Quá trình huấn luyện lặp lại:
  1. **Khởi tạo tham số**: Các trọng số $\mathbf{W}_h, \mathbf{W}_o$ được khởi tạo ngẫu nhiên.
  2. **Lan truyền xuôi**: Tính toán đầu ra và hàm mục tiêu J.
  3. **Lan truyền ngược**: Tính gradient của J đối với các tham số.
  4. **Cập nhật tham số**: Sử dụng gradient descent (hoặc các biến thể) để cập nhật $\mathbf{W}_h, \mathbf{W}_o$.

VI. Numerical Issues và Khởi tạo

VI. Numerical Issues và Init

Các vấn đề trọng yếu

Các vấn đề tính toán số trong MLP phát sinh do đặc điểm của phép tính trên máy tính và cấu trúc của mạng nơ-ron. Dưới đây là các vấn đề chính:

- ***Gradient biến mất (Vanishing Gradient):***
  - Trong quá trình lan truyền ngược (backpropagation), gradient của hàm mất mát có thể trở nên rất nhỏ, đặc biệt ở các tầng sâu hoặc khi sử dụng hàm kích hoạt như sigmoid hoặc tanh. Điều này làm cho trọng số cập nhật chậm, dẫn đến hội tụ kém.
  - Nguyên nhân: Các gradient được nhân liên tiếp qua các tầng, và nếu giá trị nhỏ hơn 1, chúng có thể tiến gần về 0.
- ***Gradient bùng nổ (Exploding Gradient):***
  - Ngược lại, gradient có thể trở nên quá lớn, khiến trọng số thay đổi mạnh, làm mất ổn định quá trình huấn luyện.
  - Nguyên nhân: Các giá trị gradient lớn được tích lũy qua nhiều tầng, đặc biệt khi trọng số khởi tạo quá lớn hoặc ma trận trọng số có giá trị riêng (eigenvalue) lớn.
- Sai số làm tròn (Rounding Errors):
  - MLP thường xử lý số lượng lớn phép tính (ma trận, tích vô hướng), và sai số làm tròn trong biểu diễn số thực trên máy tính có thể tích lũy, đặc biệt trong các mạng sâu.
- Điều kiện không tốt (Ill-conditioning):
  - Ma trận Hessian (liên quan đến đạo hàm bậc hai của hàm mất mát) có thể có số điều kiện lớn, gây khó khăn trong việc tối ưu hóa bằng các phương pháp như gradient descent.
- Vấn đề về tính ổn định số:
  - Các hàm kích hoạt như ReLU có thể gây ra vấn đề “nơ-ron chết” (dead neurons) nếu nhiều nơ-ron có đầu ra bằng 0, làm giảm khả năng học của mạng

VI. Numerical Issues và Init

Các vấn đề trọng yếu

Khởi tạo trọng số và bias trong MLP đóng vai trò quan trọng để đảm bảo quá trình huấn luyện ổn định và hội tụ nhanh.
Các vấn đề số học như gradient biến mất hoặc bùng nổ thường liên quan trực tiếp đến cách khởi tạo.

Khởi tạo tốt giúp đảm bảo gradient có giá trị hợp lý, tránh các vấn đề vanishing/exploding gradient.
Nó ảnh hưởng đến tốc độ hội tụ và khả năng thoát khỏi các cực trị địa phương của hàm mất mát.
Một khởi tạo phù hợp giúp cân bằng giữa tính đa dạng (diversity) của các nơ-ron và sự ổn định trong huấn luyện.

- **Phương pháp khởi tạo phổ biến**:
  1. **Khởi tạo ngẫu nhiên đơn giản**:
     - Trọng số được khởi tạo ngẫu nhiên từ phân phối đều hoặc Gaussian với phương sai nhỏ (ví dụ: $\mathcal{N}(0, 0.01)$).
     - Nhược điểm: Có thể dẫn đến gradient biến mất hoặc bùng nổ nếu phương sai không được chọn cẩn thận.
  2. **Khởi tạo Xavier/Glorot**:
     - Dành cho hàm kích hoạt như sigmoid hoặc tanh.
     - Trọng số được lấy mẫu từ phân phối $\mathcal{N}(0, \sqrt{\frac{2}{n_{\text{in}}+n_{\text{out}}}})$, trong đó n_{in} và n_{out} là số nơ-ron vào và ra của tầng.
     - Mục tiêu: Giữ phương sai của đầu ra và gradient ổn định qua các tầng.
  3. **Khởi tạo He**:
     - Dành cho hàm kích hoạt ReLU hoặc các biến thể.
     - Trọng số được lấy mẫu từ $\mathcal{N}(0, \sqrt{\frac{2}{n_{\text{in}}}})$.
     - Giúp tránh vấn đề "nơ-ron chết" và đảm bảo gradient không biến mất quá nhanh.
  4. **Khởi tạo đồng nhất (Uniform Initialization)**:
     - Trọng số được lấy mẫu từ phân phối đều trong khoảng $[-\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}, \sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}]$ (cho Xavier) hoặc điều chỉnh cho He.
  5. **Bias initialization**:
     - Bias thường được khởi tạo bằng 0 hoặc một giá trị nhỏ để tránh làm lệch đầu ra ban đầu.

VI. Numerical Issues và Init

Giảm thiểu Numerical Issues thông qua Khởi tạo

Chọn phương pháp khởi tạo phù hợp:

- Sử dụng Xavier cho sigmoid/tanh, He cho ReLU để cân bằng phương sai.
- Tránh khởi tạo tất cả trọng số bằng cùng một giá trị (ví dụ: 0), vì điều này làm mất tính đa dạng của nơ-ron, khiến mạng không học được.

Kỹ thuật bổ sung:

- Batch Normalization: Chuẩn hóa đầu ra của mỗi tầng để giảm sự phụ thuộc vào khởi tạo và cải thiện ổn định số.
- Gradient Clipping: Giới hạn giá trị gradient để tránh bùng nổ.
- Sử dụng hàm kích hoạt phù hợp: ReLU hoặc các biến thể như Leaky ReLU giúp giảm vấn đề gradient biến mất so với sigmoid.

VII. Biến thể của Gradient Descent

VII. Biến thể của GD

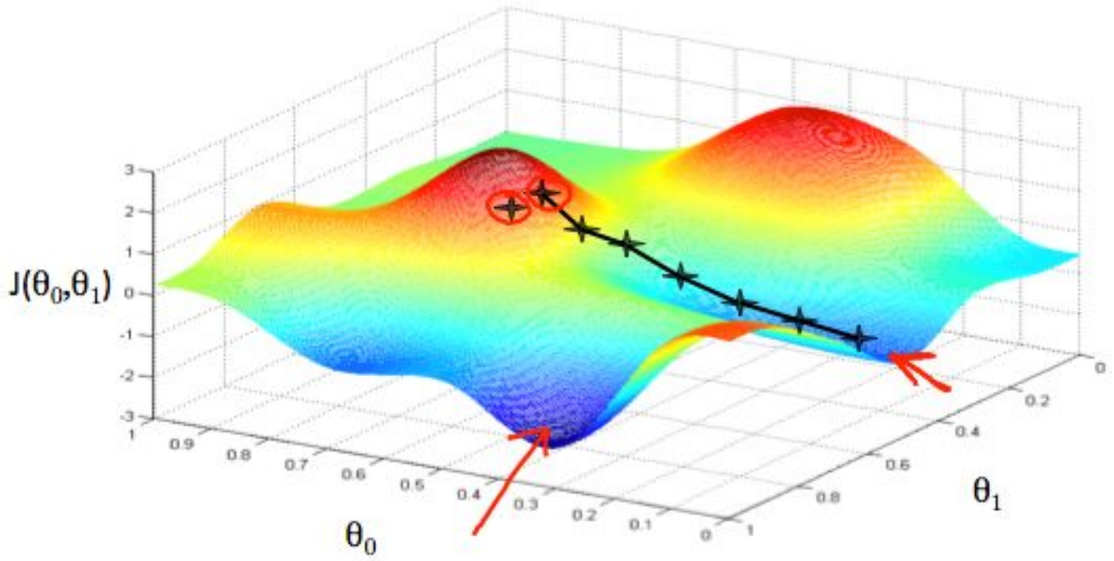
Vấn đề của GD

Gradient Descent (GD) là phương pháp tối ưu hóa cơ bản, cập nhật tham số theo công thức:

$$\theta = \theta - \eta \nabla L(\theta)$$

trong đó η là **tốc độ học (learning rate)**, $\nabla L(\theta)$ là gradient của hàm mất mát.

Tuy nhiên, Gradient Descent cơ bản có hạn chế (như tốc độ hội tụ chậm, dễ bị kẹt ở cực trị địa phương)



VII. Biến thể của GD

Vấn đề của GD

Stochastic Gradient Descent (SGD)

- **Cách hoạt động**: Tính gradient trên từng mẫu dữ liệu (hoặc mini-batch) và cập nhật tham số ngay lập tức.
- **Ưu điểm**:
  - Nhanh hơn GD vì chỉ sử dụng một phần dữ liệu mỗi lần cập nhật.
  - Tính ngẫu nhiên giúp thoát khỏi các cực trị địa phương.
- **Nhược điểm**: Gradient dao động mạnh, có thể gây bất ổn.
- **Ứng dụng**: Phổ biến trong huấn luyện MLP với mini-batch (thường 32–256 mẫu).

SGD with Momentum

- **Cách hoạt động**: Thêm thành phần **đà (momentum)** để tăng tốc gradient descent bằng cách tích lũy gradient từ các bước trước: $v_t = \beta v_{t-1} + (1 - \beta)\nabla L(\theta)\, \theta = \theta - \eta v_t$ trong đó β (thường 0.9) là hệ số đà.
- **Ưu điểm**:
  - Giảm dao động, tăng tốc hội tụ ở các vùng phẳng hoặc dốc.
  - Hữu ích khi hàm mất mát có dạng “thung lũng” hẹp.
- **Nhược điểm**: Cần điều chỉnh thêm tham số β.
- **Ứng dụng**: Hiệu quả trong MLP cho các bài toán phân loại phức tạp.

VII. Biến thể của GD

Vấn đề của GD

AdaGrad (Adaptive Gradient)

- **Cách hoạt động**: Điều chỉnh tốc độ học riêng cho từng tham số dựa trên tổng bình phương gradient trong quá khứ: $\theta = \theta - \frac{\eta}{\sqrt{G_t+\epsilon}}\nabla L(\theta)$ trong đó G_t là tổng bình phương gradient, ϵ là hằng số nhỏ để tránh chia cho 0.
- **Ưu điểm**: Tự động điều chỉnh tốc độ học, phù hợp với các tham số có gradient thưa.
- **Nhược điểm**: Tốc độ học giảm nhanh, có thể dừng hội tụ sớm.
- **Ứng dụng**: Ít được dùng trực tiếp trong MLP nhưng là nền tảng cho các phương pháp khác.

RMSProp (Root Mean Square Propagation)

- **Cách hoạt động**: Cải tiến AdaGrad bằng cách sử dụng trung bình động mũ (exponential moving average) của bình phương gradient: $E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)(\nabla L(\theta))^2\ \theta = \theta - \frac{\eta}{\sqrt{E[g^2]_t+\epsilon}}\nabla L(\theta)$ trong đó ρ (thường 0.9) là hệ số suy giảm.
- **Ưu điểm**: Ngăn tốc độ học giảm quá nhanh, cải thiện hội tụ.
- **Nhược điểm**: Vẫn cần điều chỉnh η và ρ.
- **Ứng dụng**: Phổ biến trong huấn luyện MLP, đặc biệt cho các bài toán có gradient không đồng đều.

VII. Biến thể của GD

Vấn đề của GD

Adam (Adaptive Moment Estimation)

- **Cách hoạt động**: Kết hợp momentum và RMSProp, sử dụng trung bình động mũ của gradient (momentum thứ nhất) và bình phương gradient (momentum thứ hai): $m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla L(\theta)\, v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla L(\theta))^2\, \theta = \theta - \eta \frac{m_t}{\sqrt{v_t} + \epsilon}$ trong đó $\beta_1 \approx 0.9$, $\beta_2 \approx 0.999$, $\epsilon \approx 10^{-8}$.
- **Ưu điểm**:
  - Hiệu quả, hội tụ nhanh trong nhiều bài toán.
  - Tự động điều chỉnh tốc độ học, ít cần điều chỉnh tham số.
- **Nhược điểm**: Có thể không tổng quát hóa tốt như SGD trong một số trường hợp.
- **Ứng dụng**: Là lựa chọn mặc định trong huấn luyện MLP, đặc biệt cho các bài toán phân loại hoặc hồi quy phức tạp.

VIII. Pretrained Model

VIII. Pretrained Model

Định nghĩa

Định nghĩa: Pretrained model là mô hình đã được huấn luyện trên tập dữ liệu lớn (ví dụ: ImageNet cho computer vision hoặc BERT cho NLP) để học các đặc trưng tổng quát (general features) như cạnh, màu sắc, cấu trúc văn bản, ngữ nghĩa, v.v.

Ví dụ:

- Computer Vision: VGG, ResNet, EfficientNet, Inception.
- NLP: BERT, GPT, RoBERTa, T5.
- Speech: Wav2Vec, DeepSpeech.

Ưu điểm:

- Tiết kiệm thời gian và tài nguyên so với huấn luyện từ đầu (training from scratch).
- Hiệu quả cao trên các bài toán có dữ liệu nhỏ, nhờ tận dụng kiến thức đã học (transfer learning).
- Dễ dàng áp dụng cho các tác vụ liên quan hoặc mở rộng.

VIII. Pretrained Model

Các cách sử dụng pretrained model

Cách sử dụng	Cách hoạt động	Khi nào sử dụng
a. Feature Extraction(Tính năng cố định)	- Dùng pretrained model như bộ trích xuất đặc trưng- Loại bỏ các tầng cuối (fully connected), giữ lại các tầng convolution/transformer- Dùng đặc trưng trích xuất để huấn luyện mô hình đơn giản hơn (SVM, Linear Classifier, vài tầng FC mới)	- Dữ liệu huấn luyện rất ít- Không đủ tài nguyên tính toán để fine-tune toàn bộ mô hình
b. Fine-tuning(Tinh chỉnh mô hình)	- Tiếp tục huấn luyện pretrained model trên dữ liệu mới- Có thể giữ nguyên (freeze) các tầng thấp và tinh chỉnh (unfreeze) các tầng cao- Thay tầng cuối để phù hợp với bài toán	- Có tập dữ liệu đủ lớn để tinh chỉnh mà không overfitting- Bài toán tương tự nhưng không giống hoàn toàn pretrained model
c. Transfer Learning(Chuyển giao học tập)	- Kết hợp feature extraction và fine-tuning- Điều chỉnh một phần hoặc toàn bộ mô hình- Dùng learning rate nhỏ để giữ lại đặc trưng đã học	- Bài toán liên quan đến nhiệm vụ gốc của pretrained model- Muốn tối ưu hiệu suất trên dữ liệu mới

VIII. Pretrained Model

Inference với Pretrained Model

Inference là quá trình sử dụng mô hình đã huấn luyện (pretrained hoặc fine-tuned) để dự đoán trên dữ liệu mới.

1. Tải mô hình:
   - Tải pretrained model từ các thư viện như torchvision (PyTorch), tensorflow_hub (TensorFlow), hoặc transformers (Hugging Face).
   - Ví dụ: *model = torchvision.models.resnet50(pretrained=True)* hoặc *model = BertModel.from_pretrained('bert-base-uncased')*.
2. Chuẩn bị dữ liệu:
   - Tiền xử lý dữ liệu đầu vào (input) sao cho phù hợp với định dạng của pretrained model (ví dụ: chuẩn hóa ảnh, tokenize văn bản).
   - Đảm bảo dữ liệu đầu vào có cùng kích thước, định dạng mà mô hình yêu cầu.
3. Chạy dự đoán:
   - Chuyển dữ liệu qua mô hình để nhận kết quả (logits, probabilities, embeddings, v.v.).
   - Ví dụ: Trong PyTorch, sử dụng ***model.eval()*** và ***with torch.no_grad()***: để tắt gradient trong inference.
4. Xử lý kết quả:
   - Chuyển đổi đầu ra của mô hình thành định dạng mong muốn (ví dụ: lớp dự đoán, nhãn, hoặc đặc trưng).
   - Áp dụng các kỹ thuật như softmax, argmax để lấy nhãn cuối cùng.

ML IoT
HCMUT EE MACHINE LEARNING & IOT LAB

THANK YOU

CONTACT US

403.1 H6, BKHCM Campus 2

mlandiotlab@gmail.com

mliotlab.github.io

facebook.com/hcmut.ml.iot.lab

youtube.com/@mliotlab