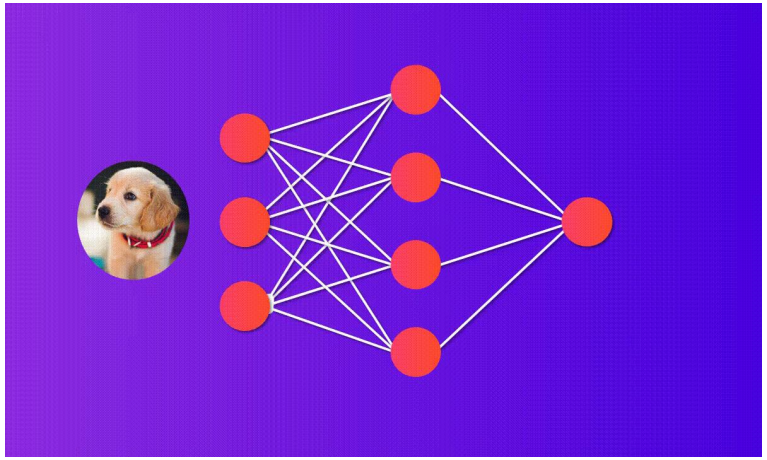
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Fundamentals of ANNs

(2026)

Lecturer: Xue Yang

yangxue.site



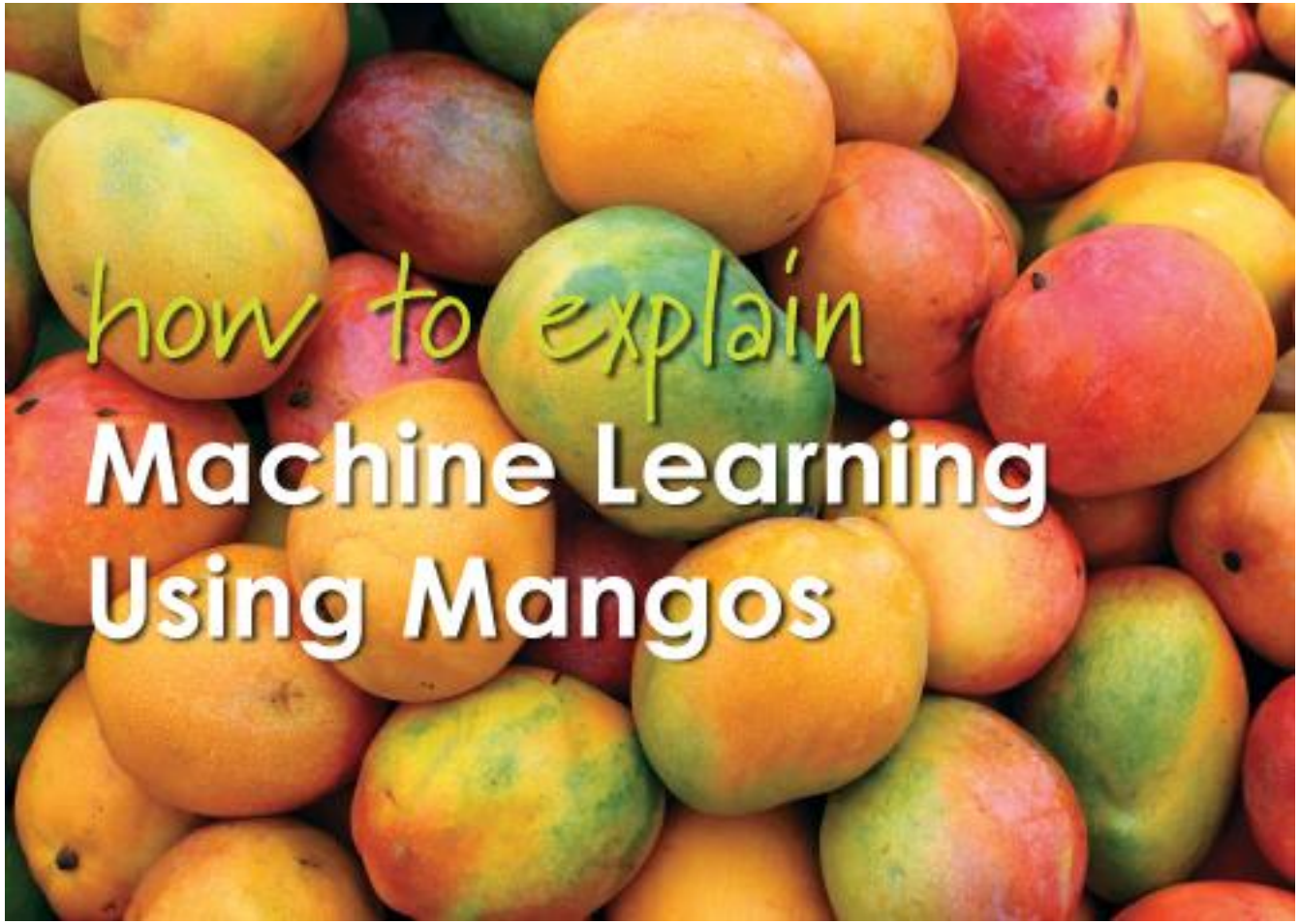
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Contents

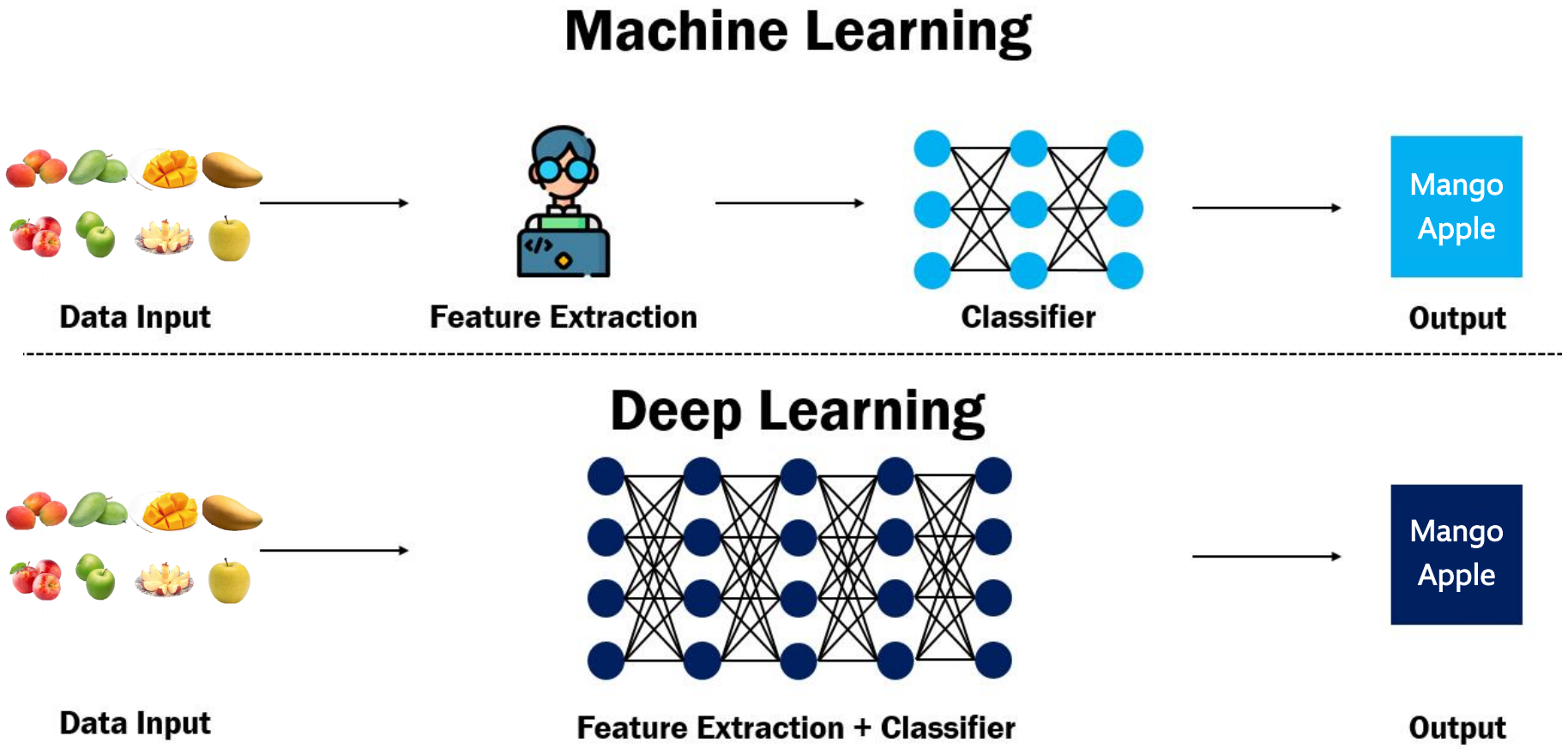
- **Introduction & Overview**
- Multi-Layer Perceptron
- Training Neural Networks
- Case Analysis

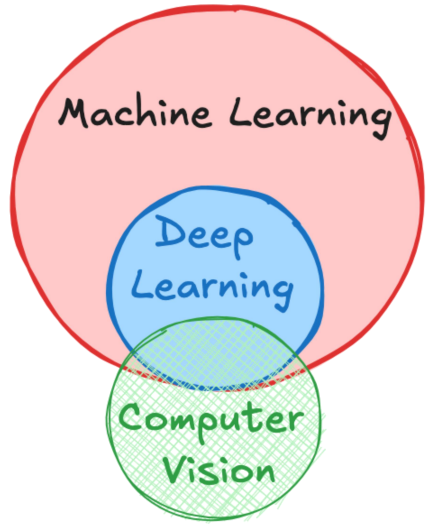
Mango Selection

- How can you tell if it's a mango?
- How can you tell if a mango is sweet?

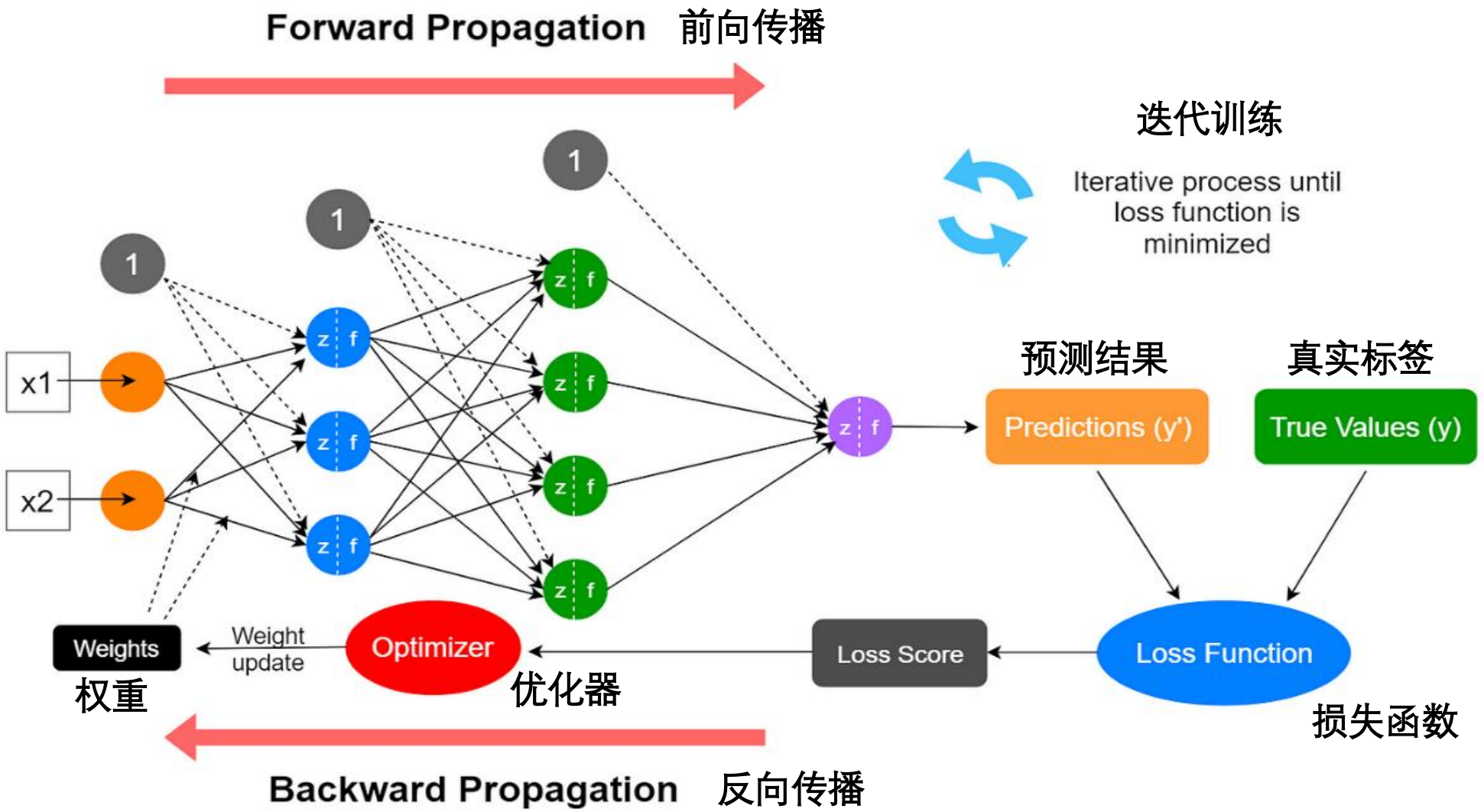


Introduction & Overview



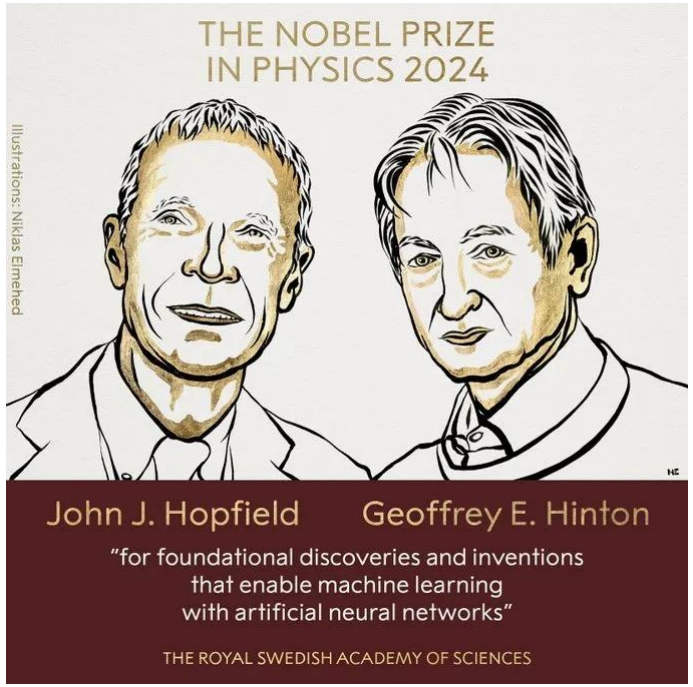


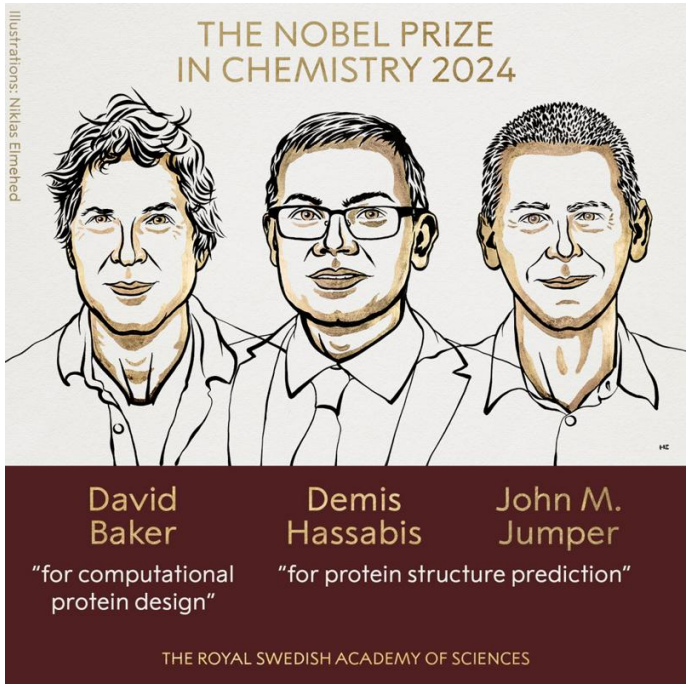
Introduction & Overview



Origin: Artificial Intelligence

- In recent years, AI has become the biggest winner of the Nobel Prize/Turing Award. As an irresistible force, it is driving a paradigm shift in scientific research.





Turing Award, 2018/2024

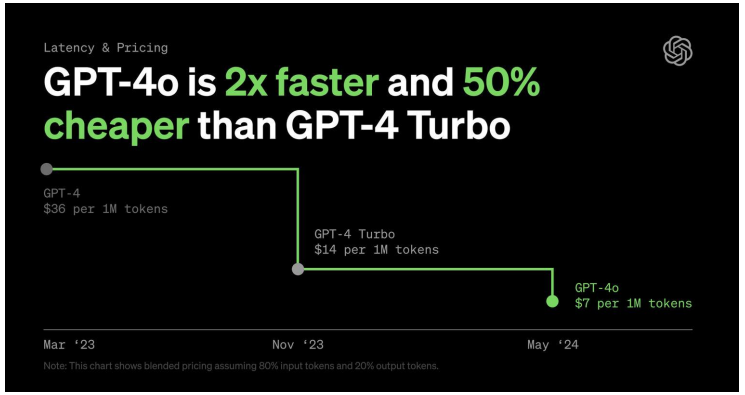
Origin: Artificial Intelligence

自动驾驶

生物制药

人形机器人

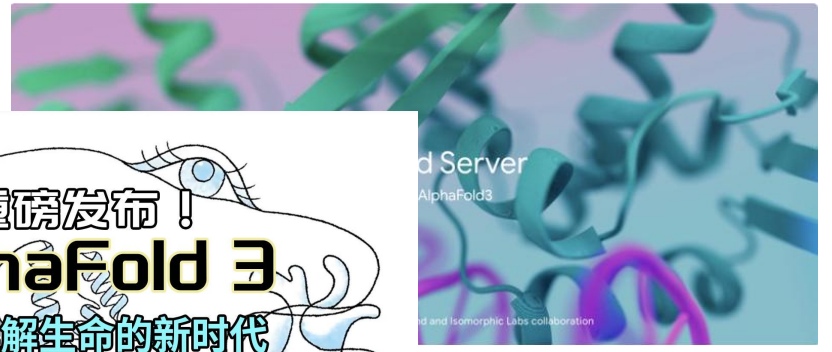
大模型



上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Origin: Artificial Intelligence

自动驾驶



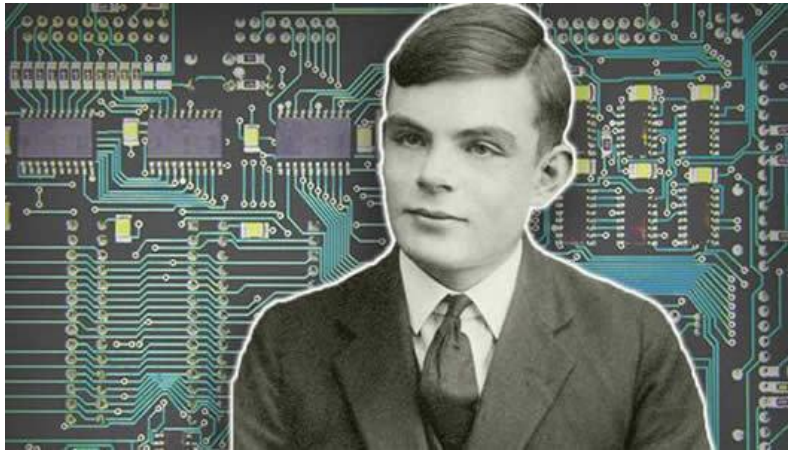
生物制药

人形机器人

大模型

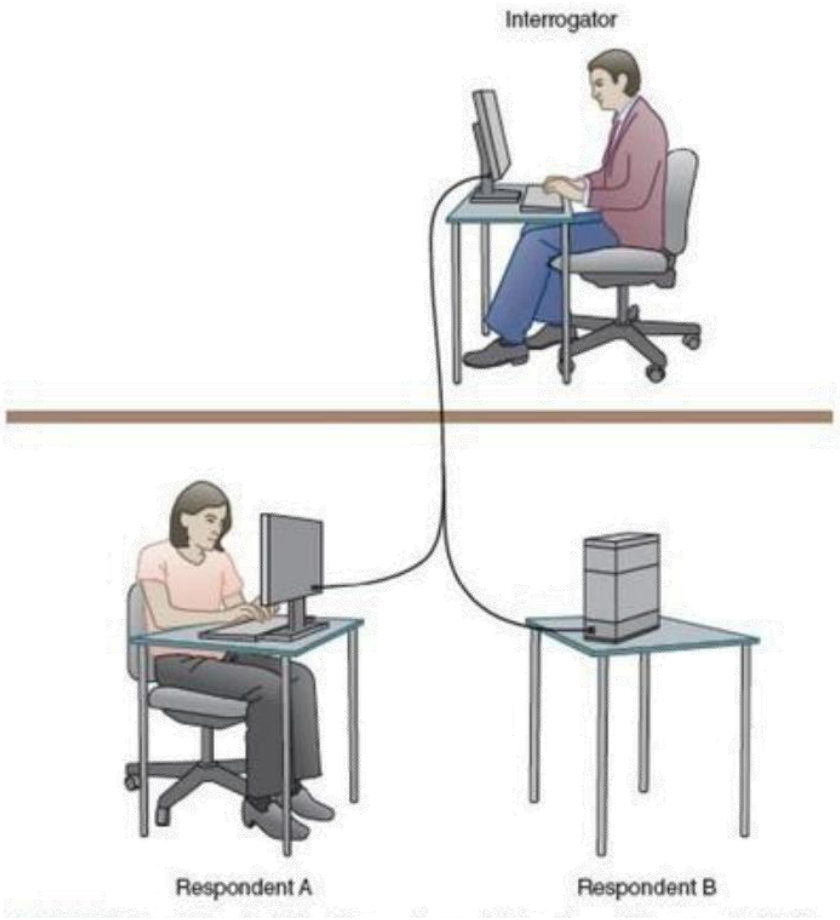
Origin: Artificial Intelligence

- Alan Turing (1912-1954)
  - ✓ 1931 Admitted to King's College, Cambridge, with a scholarship in mathematics.
  - ✓ 1938 Earned his PhD from the Institute for Advanced Study at Princeton, USA.
  - ✓ 1939 Called up to work for the British Foreign Office, cracking enemy codes.
  - ✓ 1945 Awarded the highest government honor, the Order of the British Empire.
  - ✓ 1954 Died by suicide.



Origin: Artificial Intelligence

- The Turing Test (1950s)
  - ✓ **Separate** humans from machines.
  - ✓ Humans ask questions via **electronic communication devices**.
  - ✓ After multiple rounds of Q&A, **if over 30% of people can't distinguish whether the respondent is human or machine**, the machine passes the test and is considered to possess human-level intelligence.



The very essence of the Turing test.

A和B哪个更像人类的回答？为什么？

A

问：你会下国际象棋吗？
答：是的。
问：你会下国际象棋吗？
答：是的。
问：你会下国际象棋吗？
答：是的。

B

问：你会下国际象棋吗？
答：是的。
问：你会下国际象棋吗？
答：是的，我不是已经说过了吗？
问：你会下国际象棋吗？
答：你烦不烦，干嘛老是提同样的问题。

A和B哪个更像人类的回答？为什么？

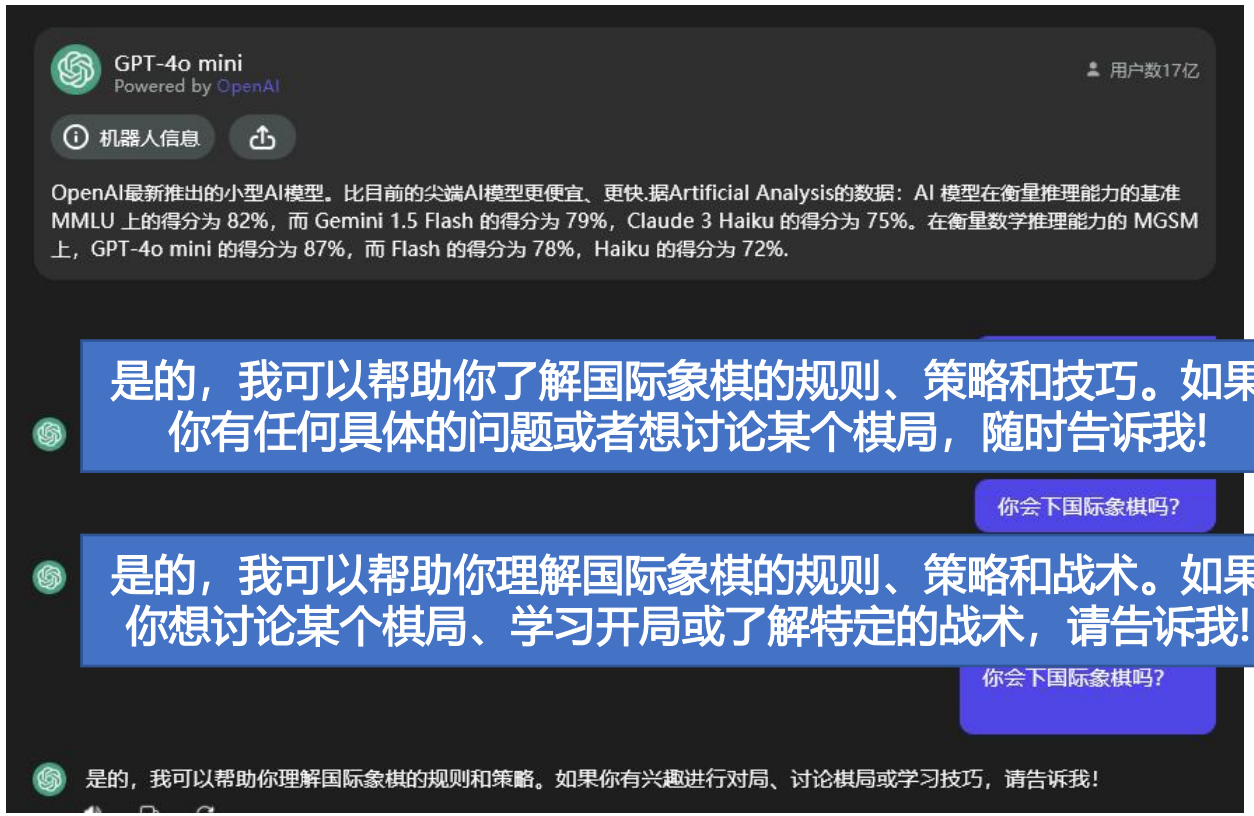
遵照特定规则回答
机械式回答

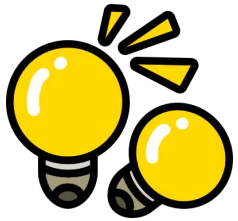
A

具有综合分析和理解的能力
能探知提问者在反复提出相同的问题

B

当今的大模型是否具备人类智能呢?





一个有趣的实验


原作者: YouTube 频道 Space Kangaroo
翻译/时轴/校对/压制: silverion
Tap to cancel


弱人工智能
强人工智能
通用智能
认知 & 理解
学习 & 推理

人工智能的主要流派

香农、图灵

兴起

图灵测试

熵&信息论

马尔可夫过程

符号主义

符号表达，数理逻辑推理

规则系统、专家系统

乔姆斯基生成文法

感知机、特征工程

知识图谱

联结主义

模拟人脑神经元的连接方式

Yann LeCun：LeNet

Geoffrey Hinton：BP、DBN

Yoshua Bengio：GAN

深度学习之父

2018年图灵奖

深度学习

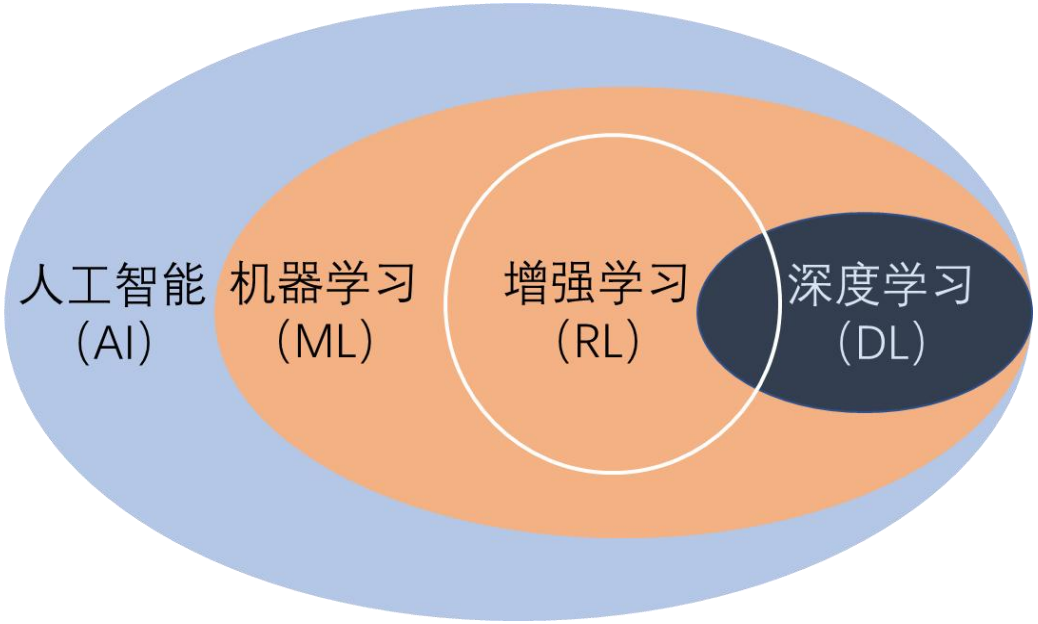
神经机器翻译

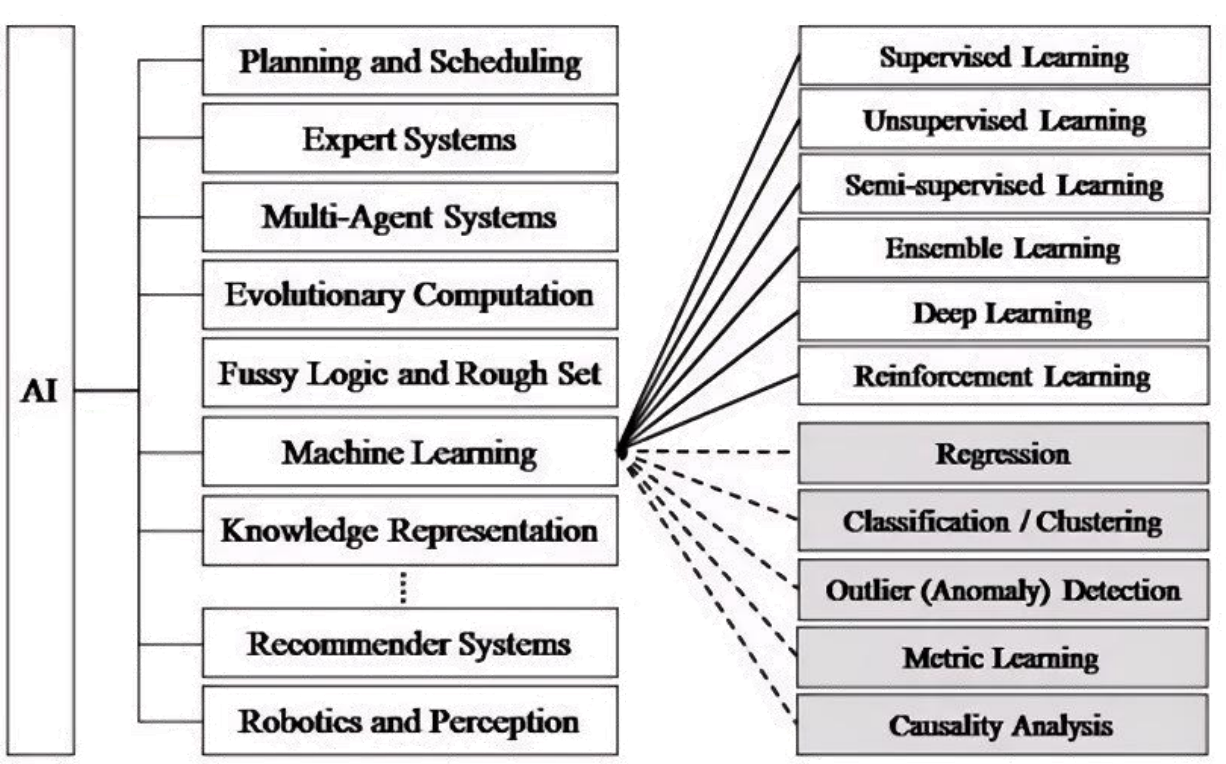
机器阅读理解

预训练LLM

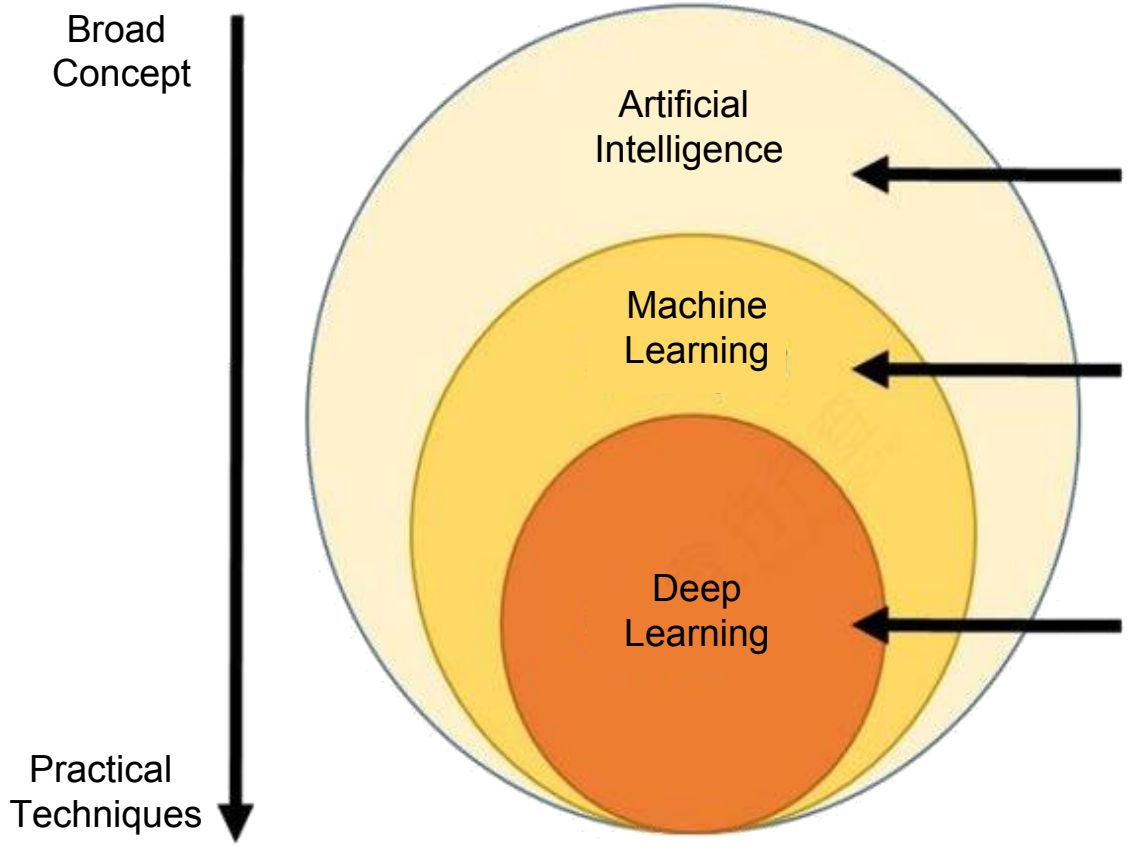
Artificial Intelligence → Deep Learning

- The differences and connections between AI and DL





Artificial Intelligence → Deep Learning



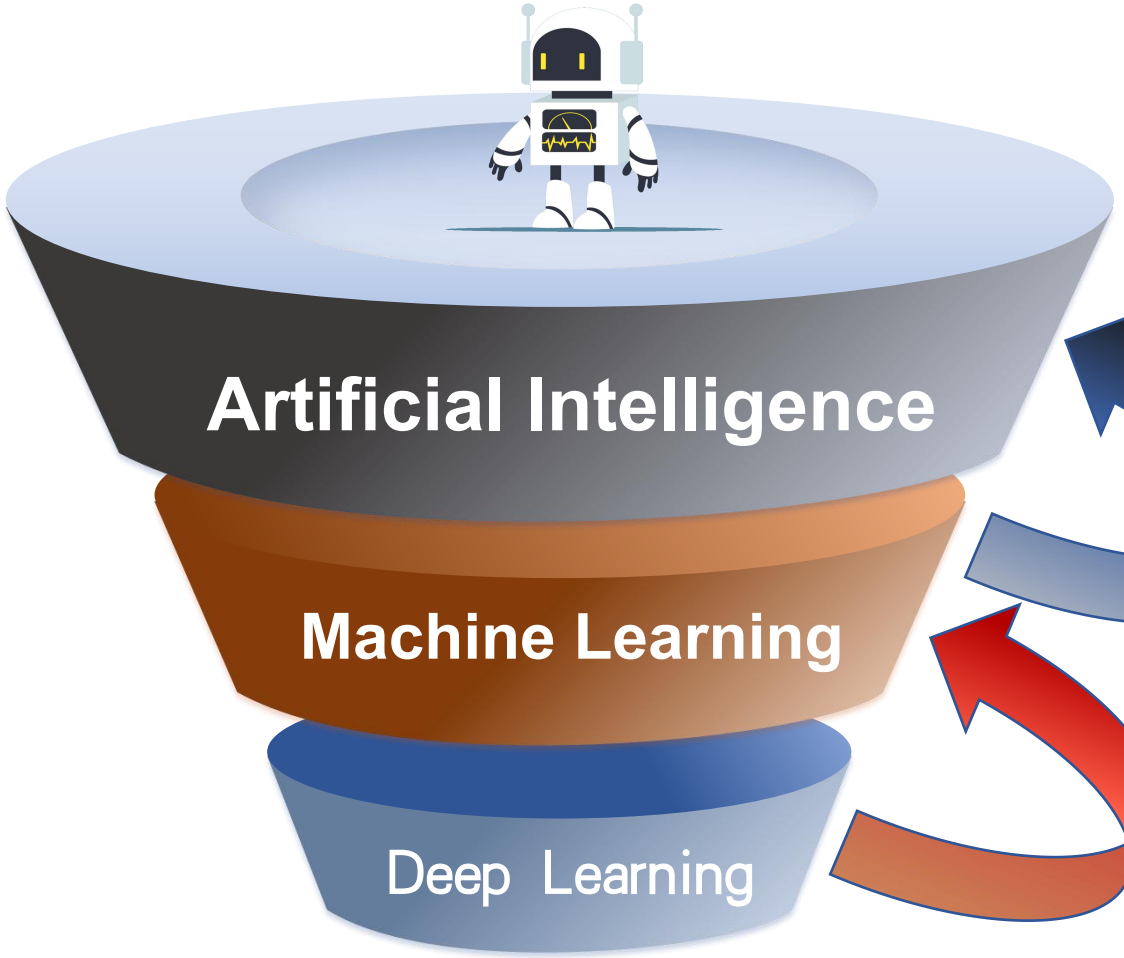
Making computers work like humans

Uses algorithms to analyze data, learn from it, and measure or predict real-world events

A specific implementation technology

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

Artificial Intelligence → Deep Learning



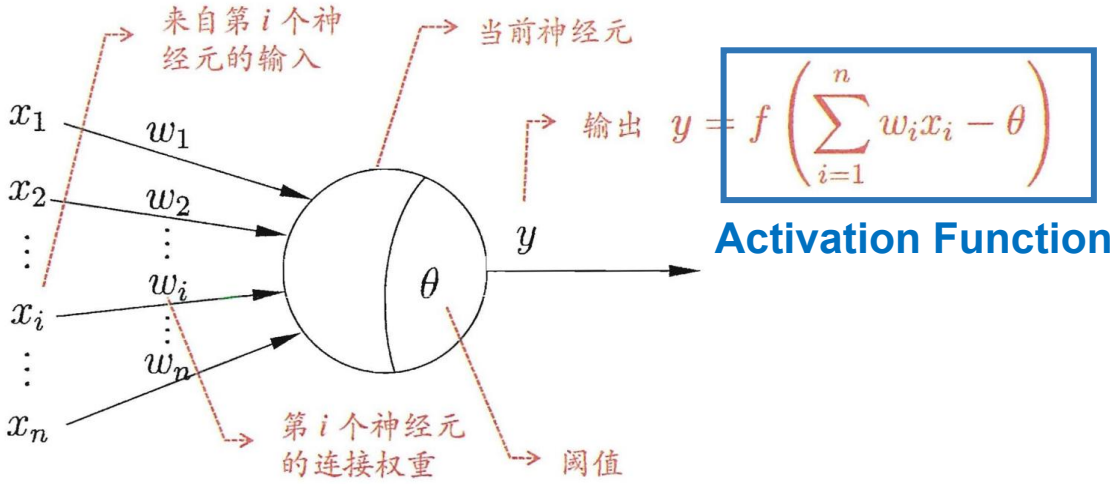
Providing methods for implementing artificial intelligence

Providing technology for implementing machine learning

The Development of Deep Learning

- When a neuron is "excited," it sends chemicals to connected neurons, altering their electrical potential.
- If the potential exceeds a threshold (θ), the neuron is activated.
- **Learned knowledge is embedded in the connection weights (ω) and threshold (θ).**

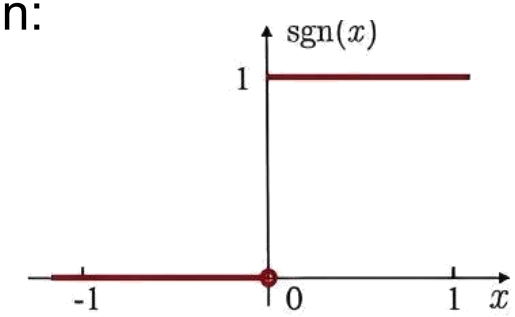
M-P Neuron Model [McCulloch and Pitts, 1943]



Budding → **1943** → **Future**

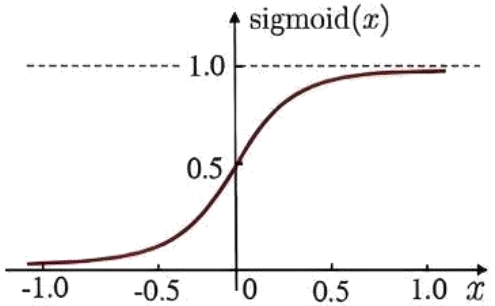
The Development of Deep Learning

- The ideal activation function is the step function:
  - ✓ 0: Inhibits the neuron
  - ✓ 1: Activates the neuron
- The step function is discontinuous and non-smooth.
- Commonly used alternative: Sigmoid function



$$\mathrm{sgn}(x) = \begin{cases} 1, & \text{if } x \geq 0; \\ 0, & \text{if } x < 0. \end{cases}$$

(a) 阶跃函数



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

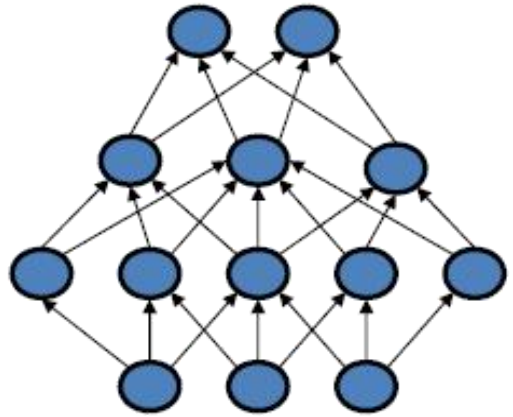
(b) Sigmoid 函数

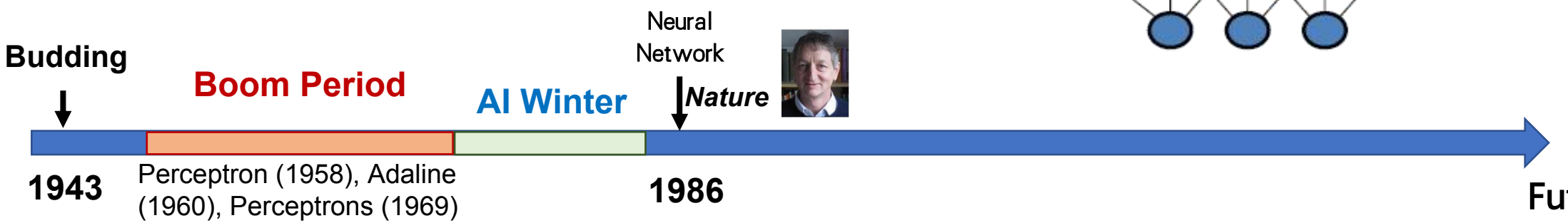
Artificial Intelligence → Deep Learning

- **First Winter** (1976-1982), faced skepticism due to
  - ✓ Insufficient computing power,
  - ✓ High computational complexity,
  - ✓ Great implementation difficulty

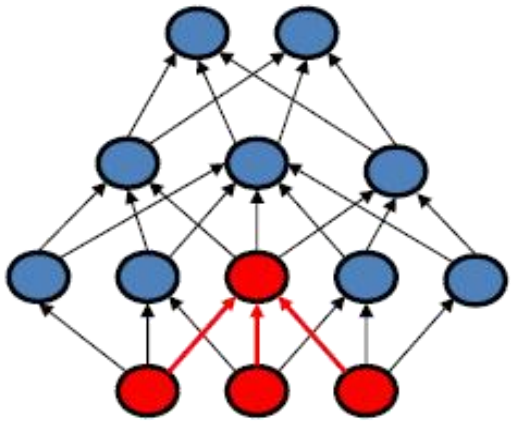
The Development of Deep Learning

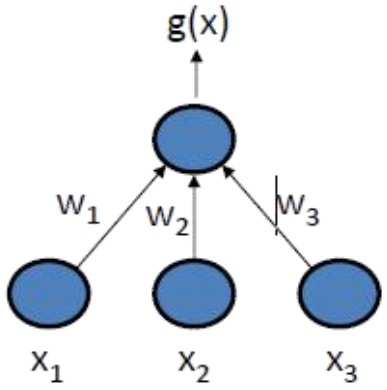
- A neural network is a widely interconnected network of adaptive simple units.
- Its organization can simulate the interaction responses of biological nervous systems to real-world objects. [T. Kohonen, NN88]
  - ✓ Solving general learning problems
  - ✓ Integrated with biological systems



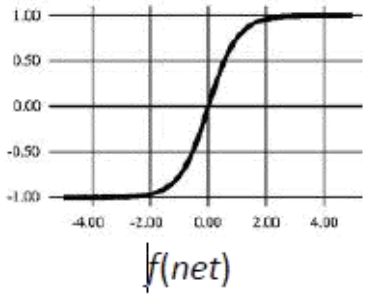


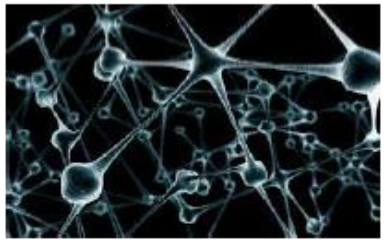
The Development of Deep Learning

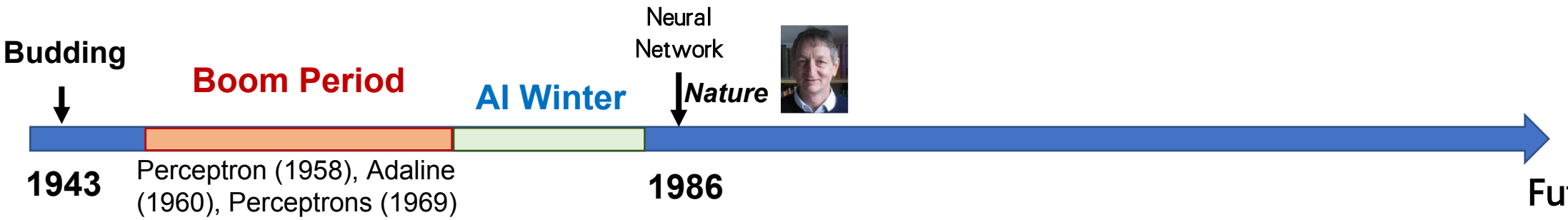




$$g(\mathbf{x}) = f(\sum_{i=1}^{d} x_i w_i + w_0) = f(\mathbf{w}^t \mathbf{x})$$

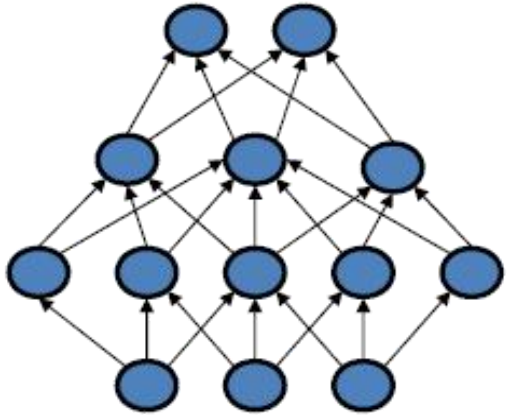


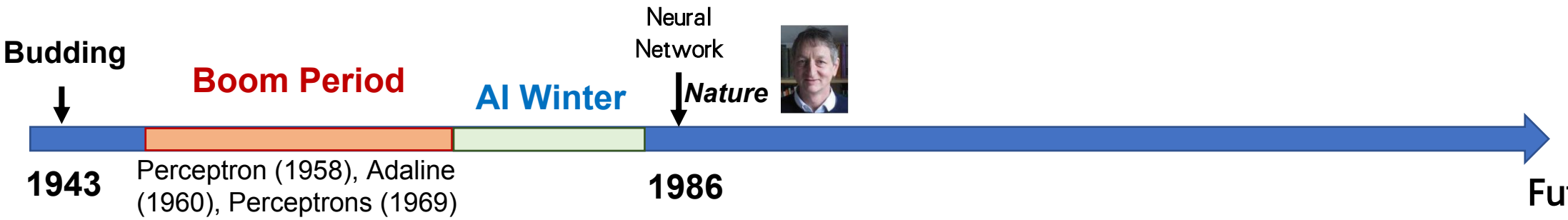




The Development of Deep Learning

- High training difficulty
- Insufficient computational power of computers
- Limited training data volume
- Poor performance



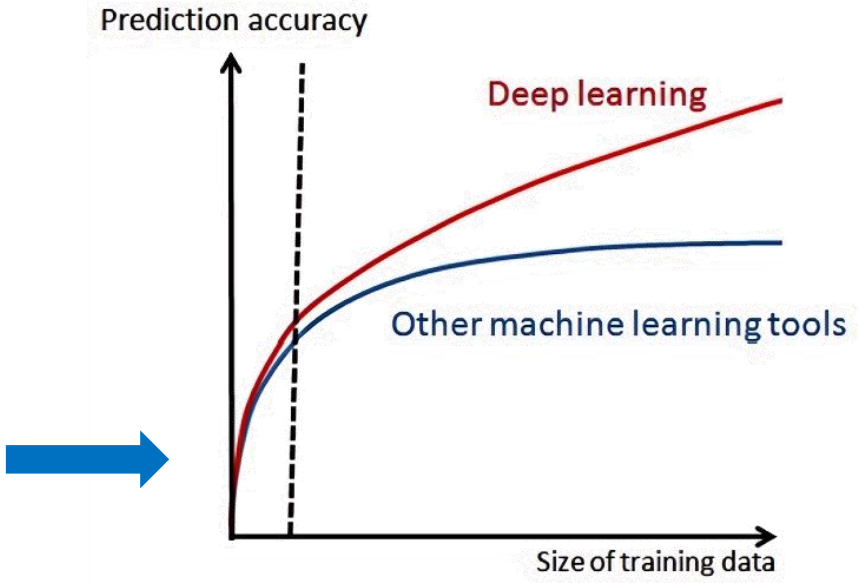


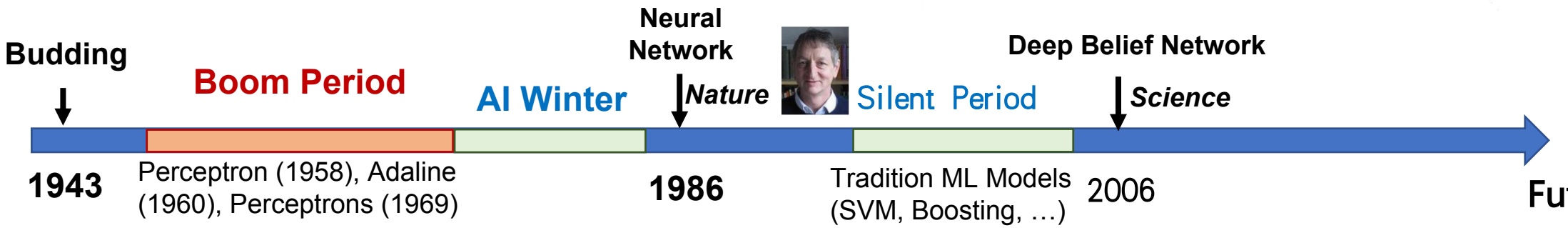
The Development of Deep Learning

- Better model design
- Greater computer computing power
- Large-scale database collection

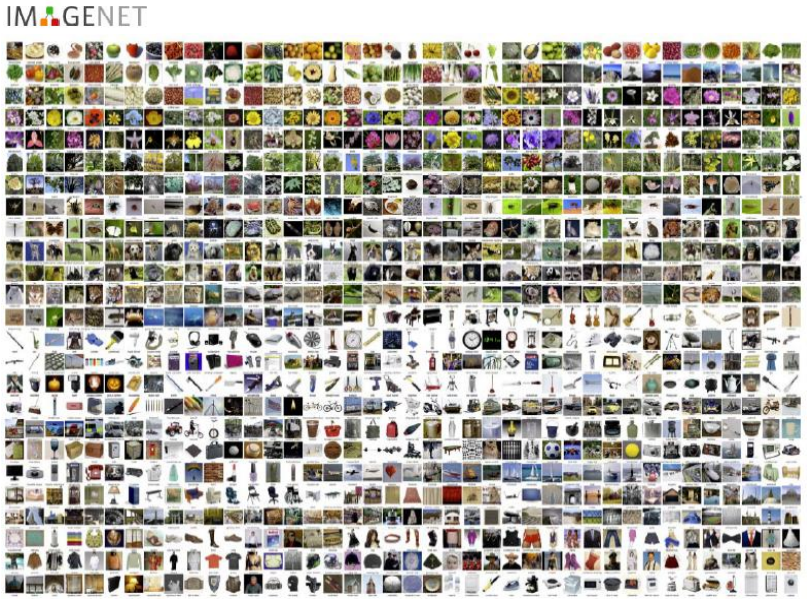
Significant performance improvement

Big data + increased model complexity



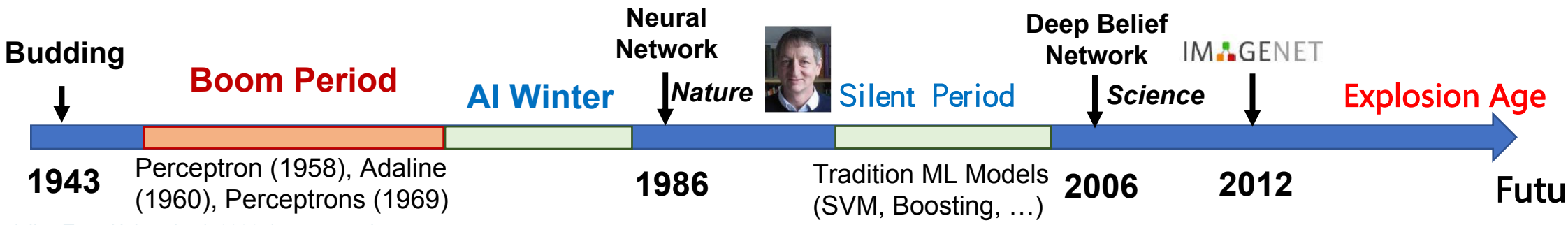


The Development of Deep Learning

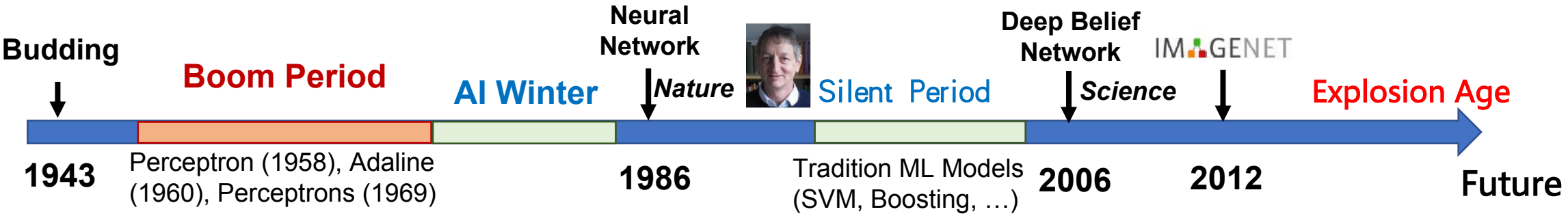


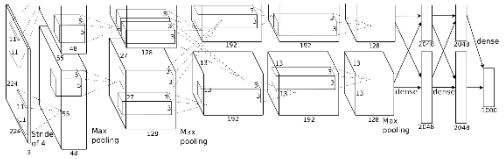
Comparison of object recognition effects
(1,000,000 images and 1,000 categories)

Rank	Institute	Error rate	Description
1	**U. Toronto**	**0.15315**	**Deep Learning**
2	U. Tokyo	0.26172	Hand-crafted features and learning models. Bottleneck.
3	U. Oxford	0.26979	
4	Xerox/INRIA	0.27058	

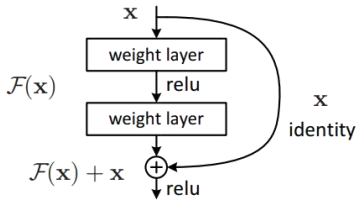


Introduction & Overview



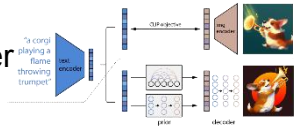


AlexNet, the beginning of deep learning



ResNet, residual solution extends network depth

BERT, bidirectional transformer
GPT-1, the first generation of large models



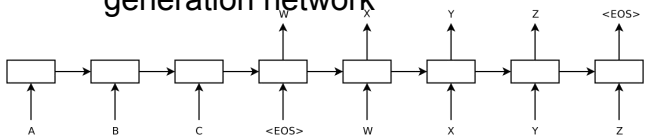
DALL-E, test-to-image

GPT-4, DALL-E3
Stable Diffusion XL

Stable Diffusion 3
GPT-4o, Support multimodality

deepseek
Deep thinking

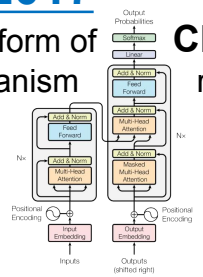
2012 **Seq2Seq**, sequence generation network



2014 **GANs**

2015

2017 **Transformer**, reform of the attention mechanism NLP model

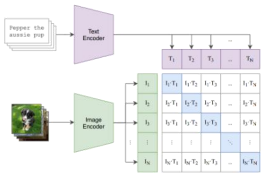


2018 **BERT**

2019 GPT-2

2020 GPT-3

2021 **CLIP**, Multimodal retrieval model



2021 **DALL-E**

2022 **ChatGPT**, Human-computer dialogue
Stable Diffusion, Text-Image Diffusion Model

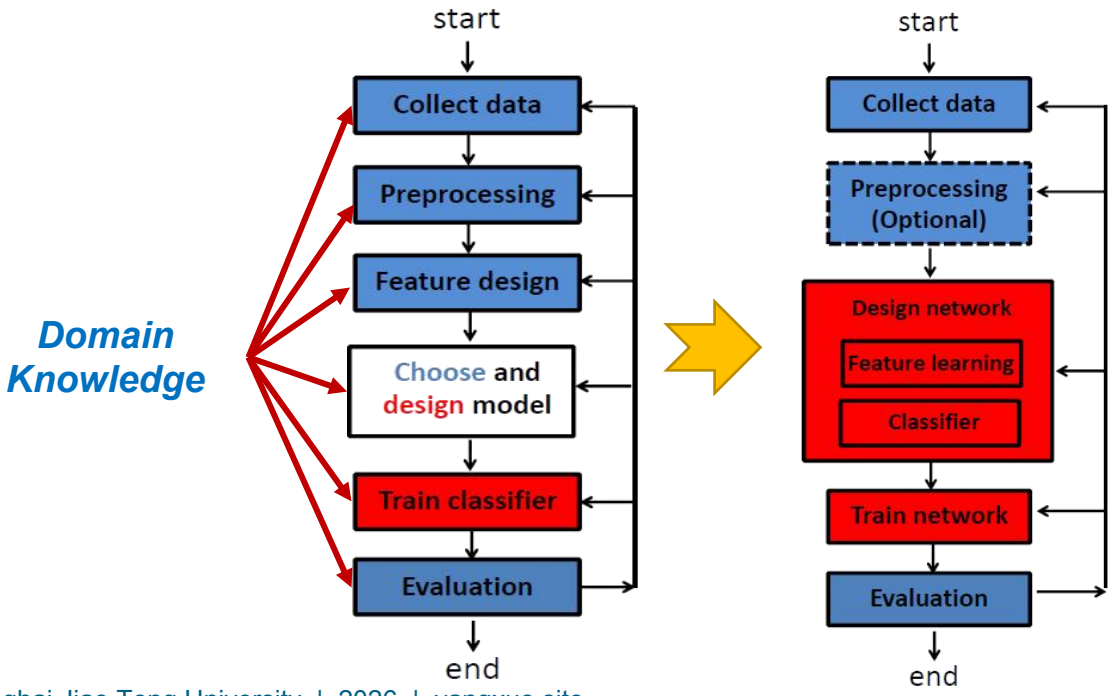
2023 **GPT-4o**

2024 **OpenAI o1**

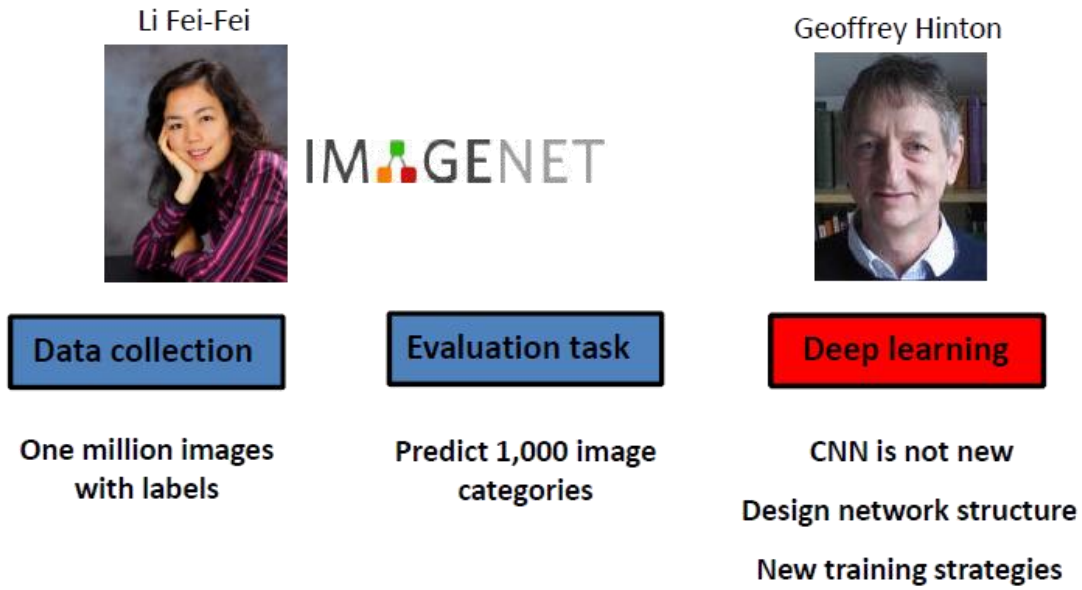
2025

Introduction & Overview

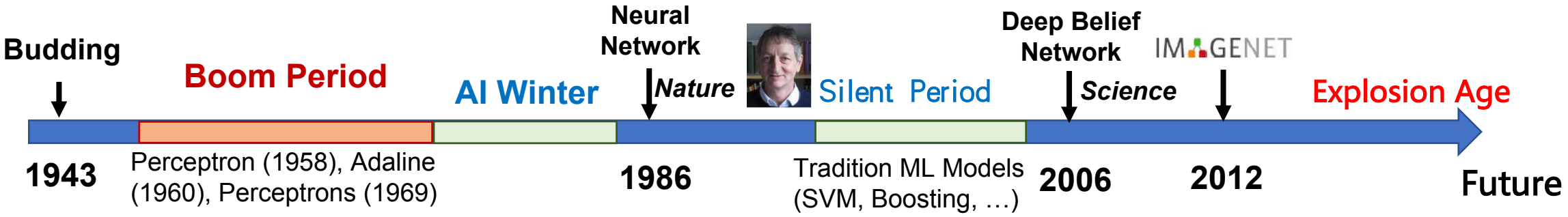
- **New algorithm design patterns**



- **Key factors for success: Big data + new learning models**



Introduction & Overview



- **Key factors for success: Big data + new learning models**

Li Fei-Fei

IMAGENET

Data collection

One million images with labels

Evaluation task

Predict 1,000 image categories

Geoffrey Hinton

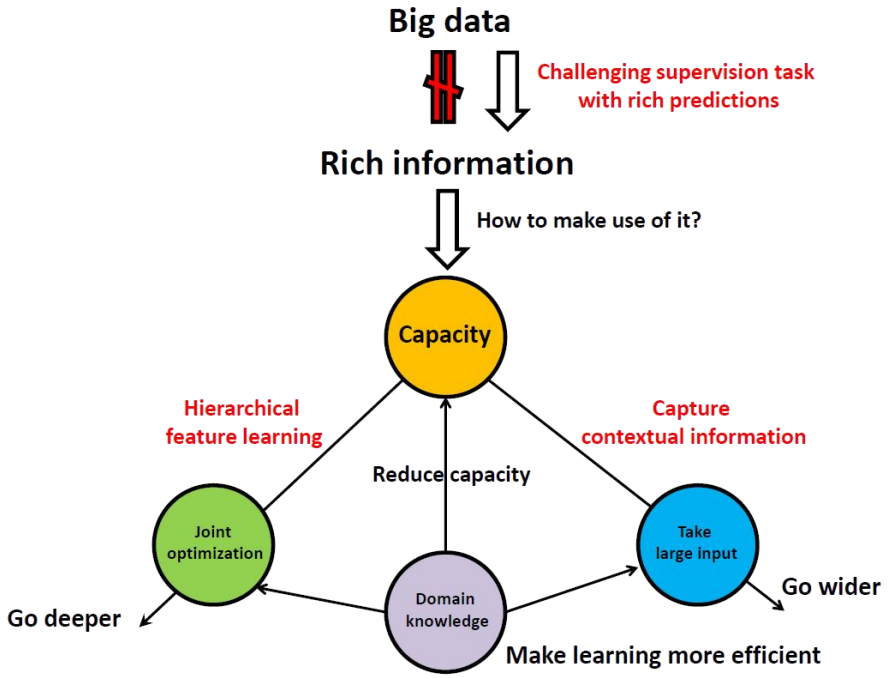
Deep learning

CNN is not new

Design network structure

New training strategies

- **Development direction**



Introduction & Overview

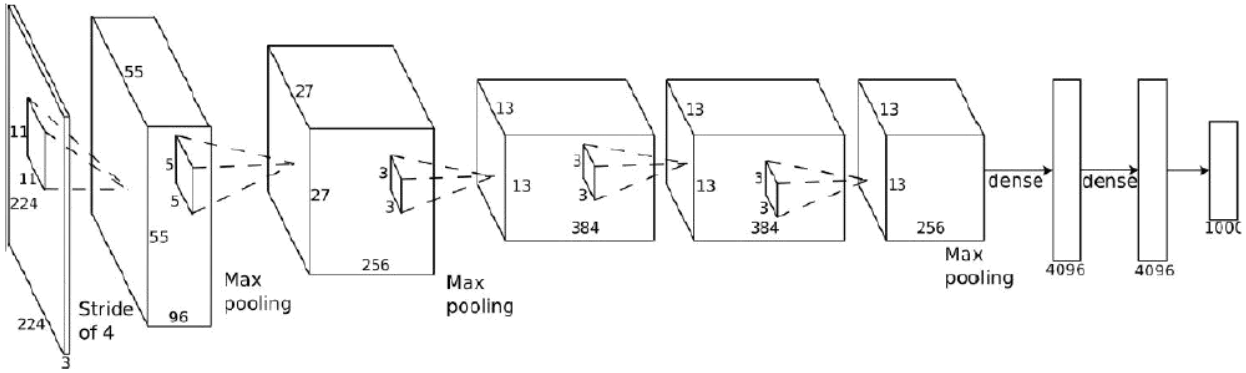
- **Feature learning**

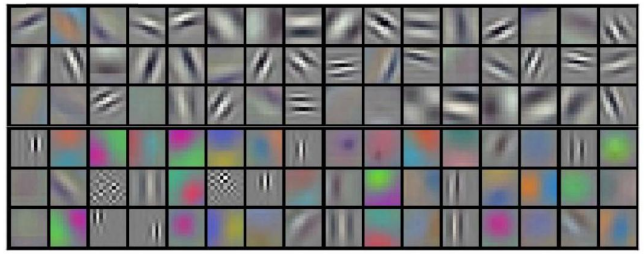
✓ Jointly optimize the "feature transformer + classifier" approach

✓ Leverage computer-generated learning of feature representations

✓ Better leverage big data

✓ Quickly acquire features suitable for new applications





96 learned low-level filters

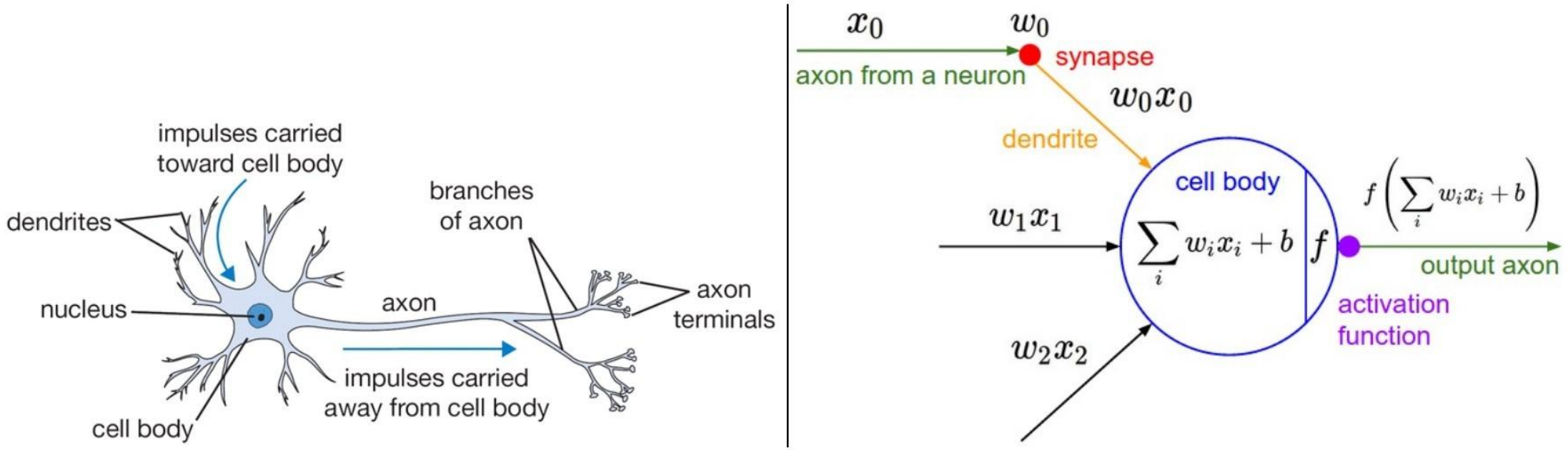
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Contents

- Introduction & Overview
- **Multi-Layer Perceptron**
- Training Neural Networks
- Case Analysis

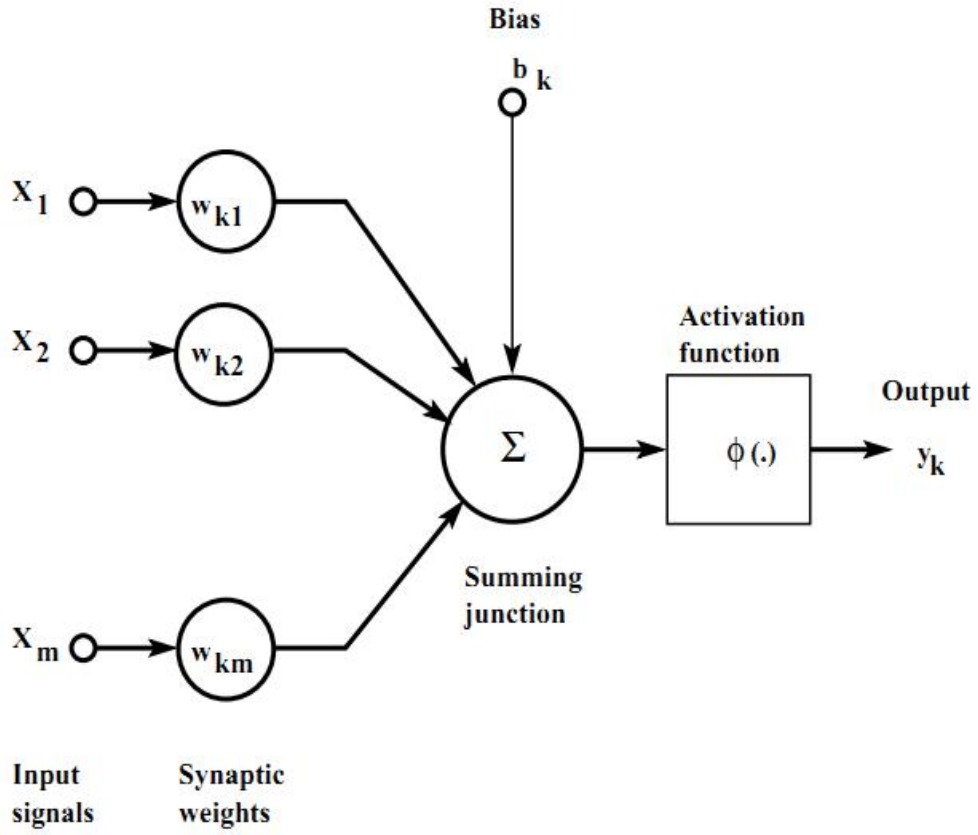
Modeling One Neuron

- Biological (left) vs. artificial neuron analogy (right)

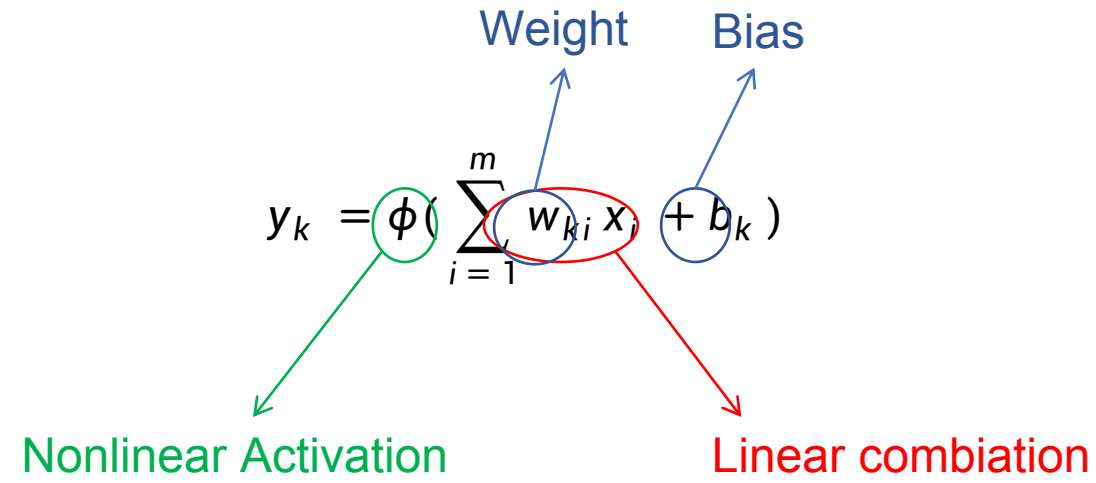


A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Single Neuron Analogy



output of k-th neuron analogy y_k:



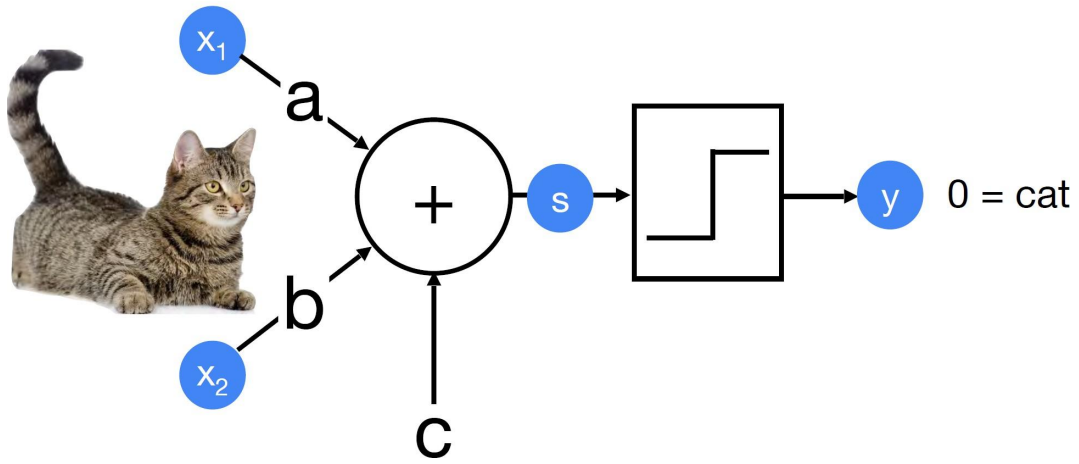
Single Neuron as a Linear Classifier

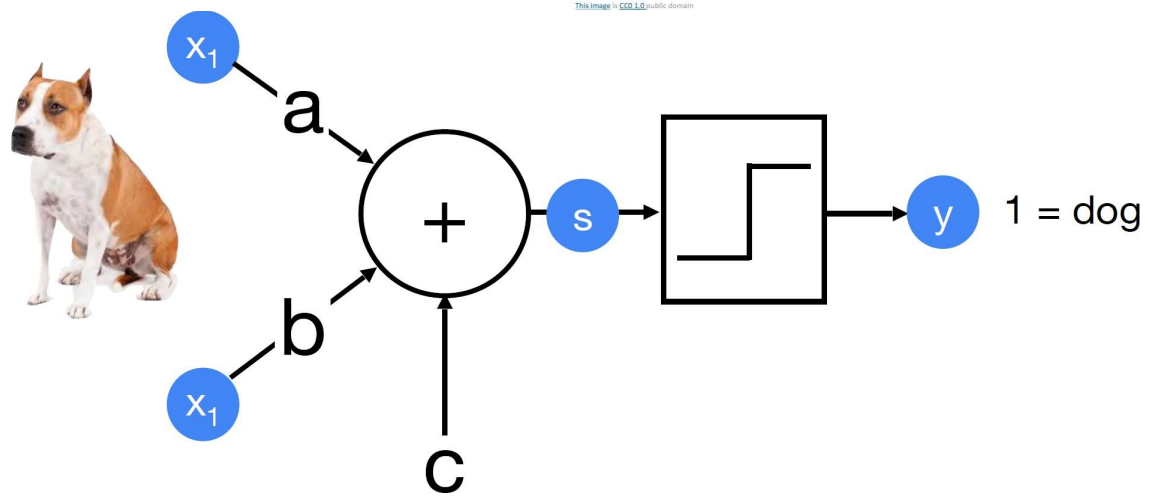
Input: $x \in \mathbb{R}^D$

Output: $Wx + b = f(x) \in \mathbb{R}^C$

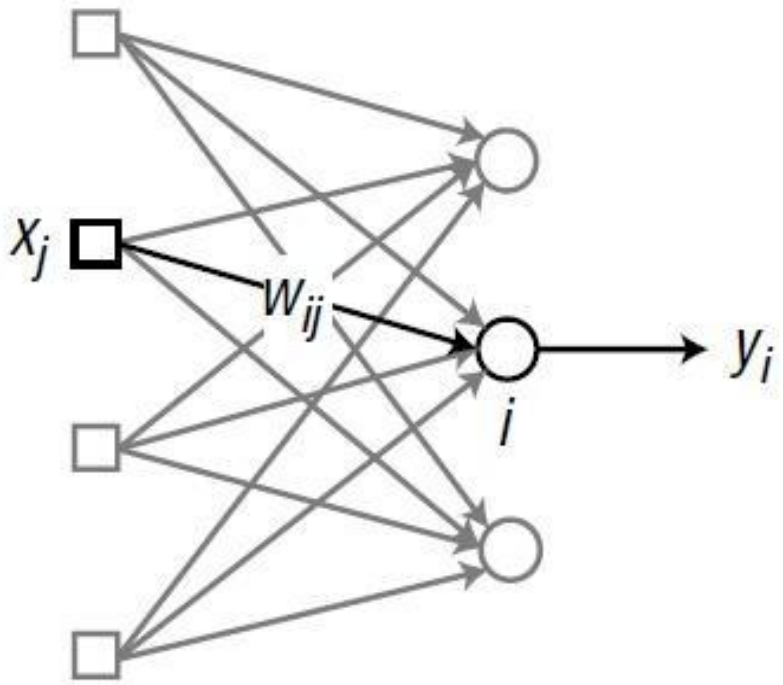
Learnable Parameters: $W \in R^{C \times D}, b \in \mathbb{R}^C$

Neural Network





Single Neurons stack → Neural Network Layer



$$y_j = \phi\left(\sum_{i=1}^{4} w_{ij}\, x_i + b_j\right)$$

Output stack

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \phi\left(\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right)$$

Weight Matrix

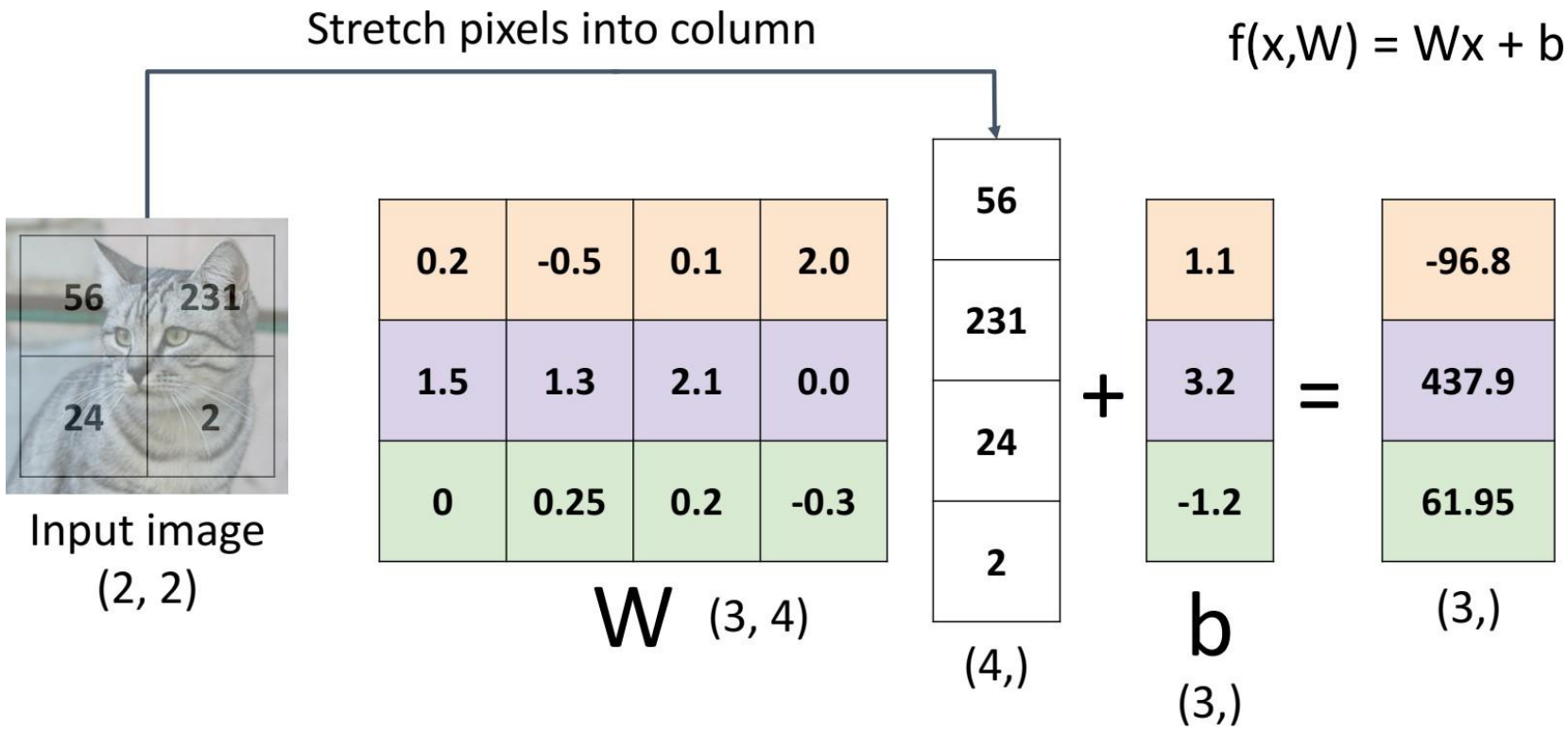
$$Y = W X + b$$

Output Vector

Input Vector

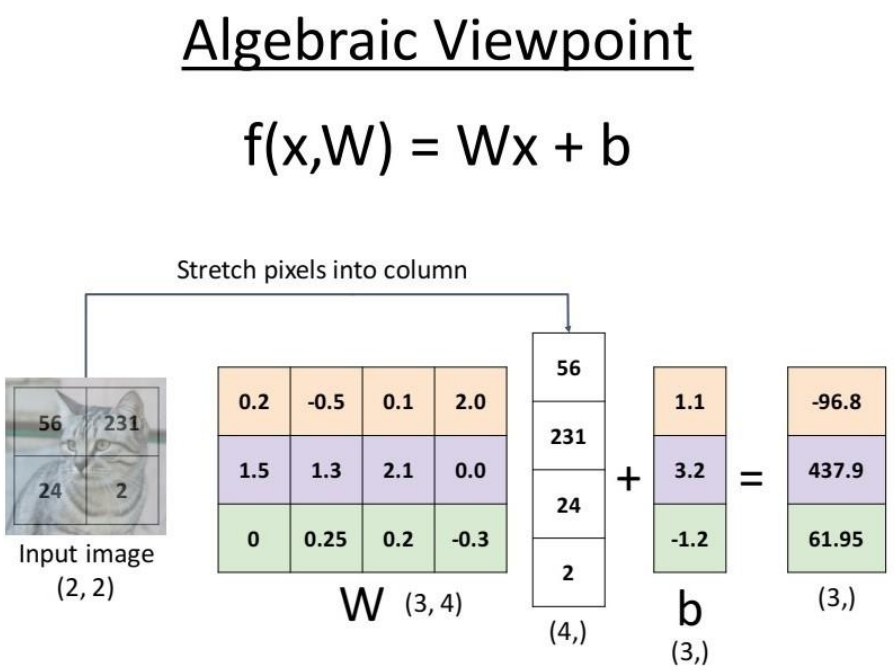
A Linear Classifier Example

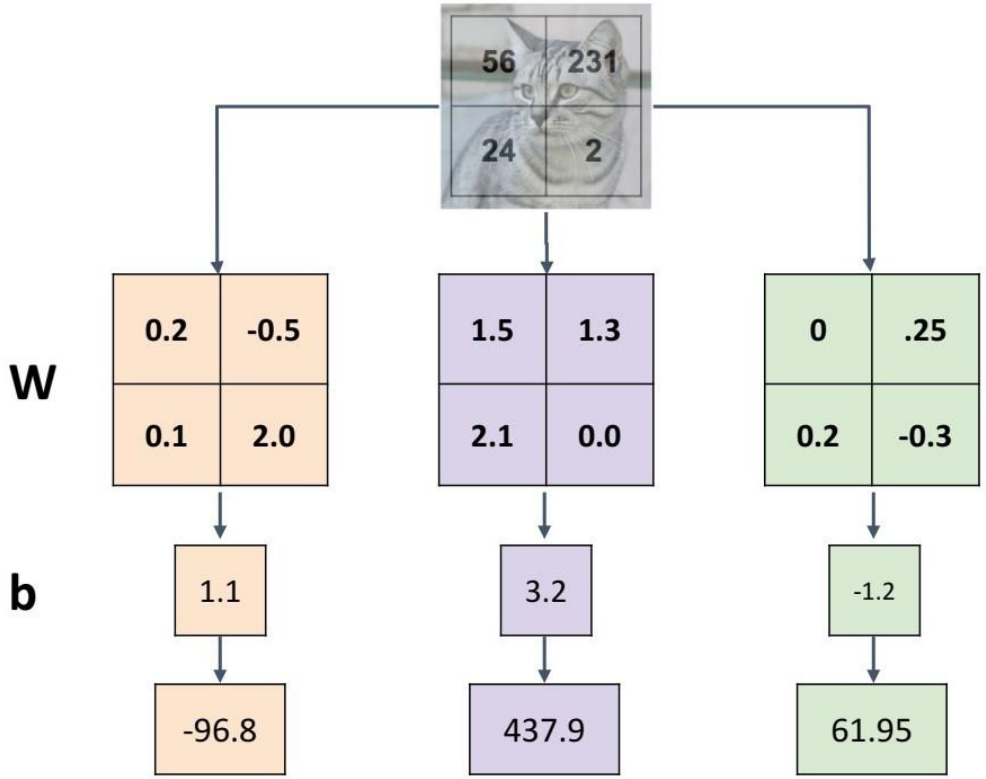
- Example for 2x2 image, 3 classes (cat/dog/ship)



A Linear Classifier Example

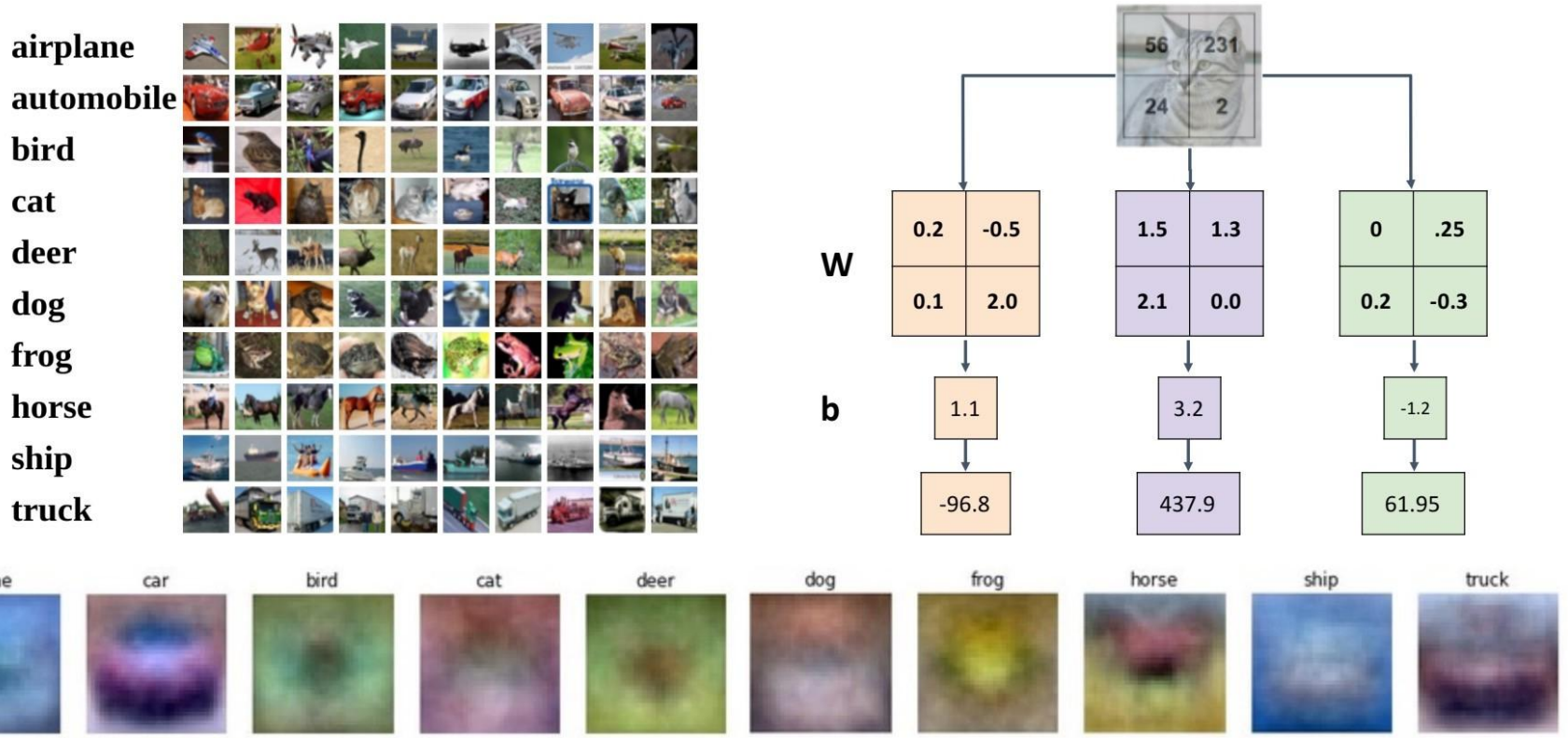
- Interpreting an linear classifier: **Algebraic viewpoint**





A Linear Classifier Example

- Interpreting an linear classifier: **Algebraic viewpoint**



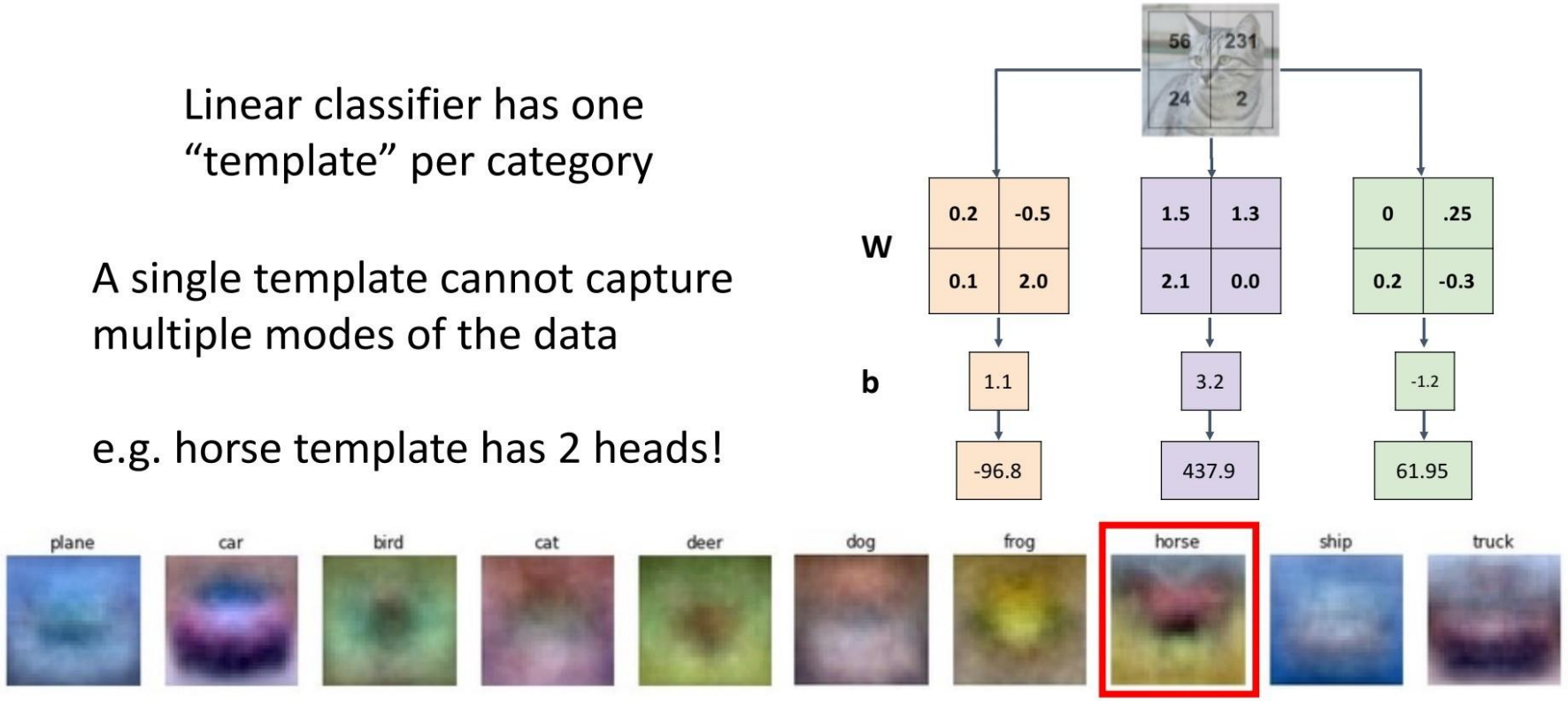
A Linear Classifier Example

- Interpreting an linear classifier: **Algebraic viewpoint**

Linear classifier has one "template" per category

A single template cannot capture multiple modes of the data

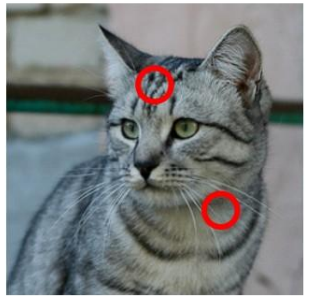
e.g. horse template has 2 heads!



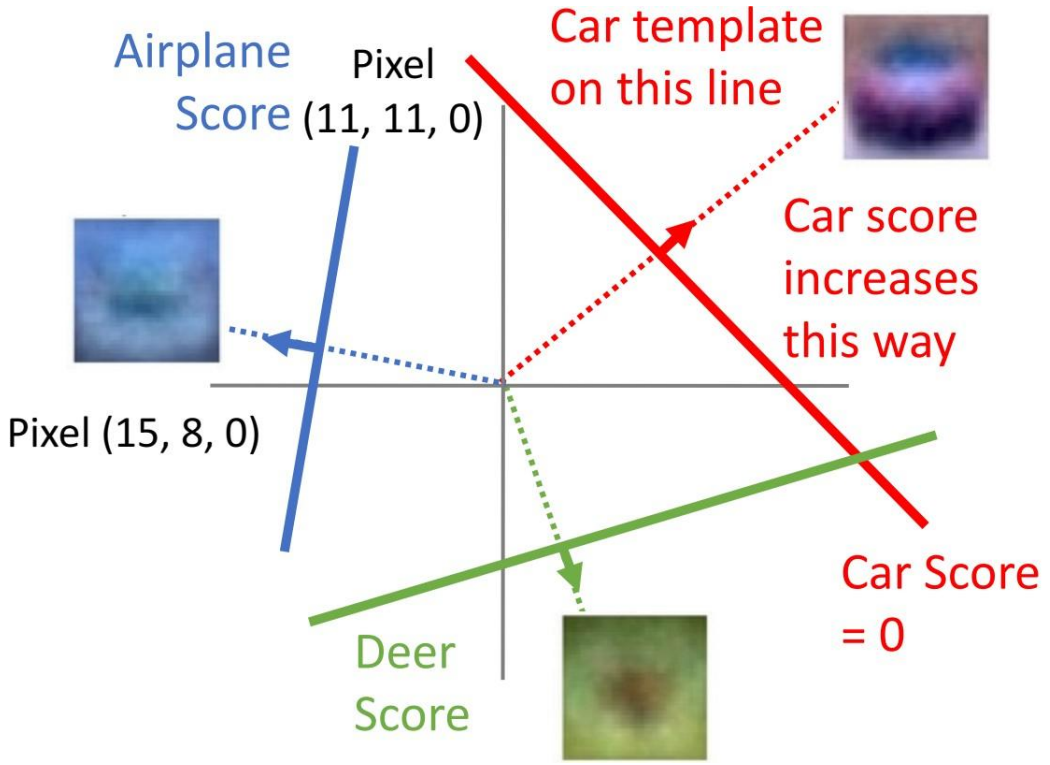
A Linear Classifier Example

- Interpreting an linear classifier: **Geometric viewpoint**

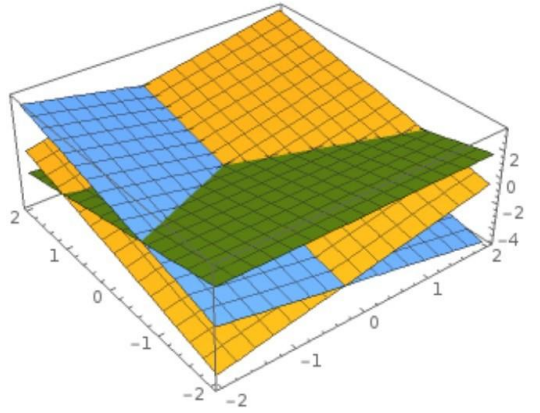
$$f(x,W) = Wx + b$$



Array of **32x32x3** numbers
(3072 numbers total)

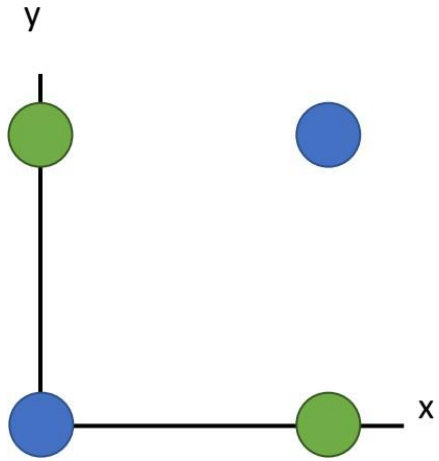


Hyperplanes carving up a high-dimensional space



Hard Cases for a Linear Classifier

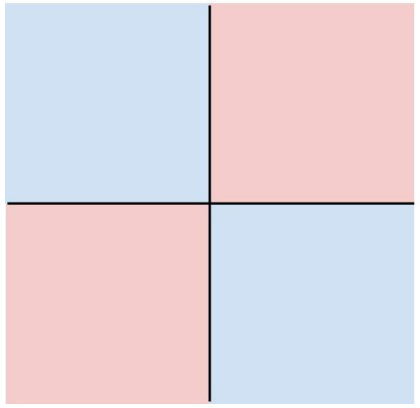
- A single neuron cannot solve many problems!
- Solution: Multi-Layer Perceptron (MLP w/ activation functions)



Famous XOR problem:
No linear separator exists!

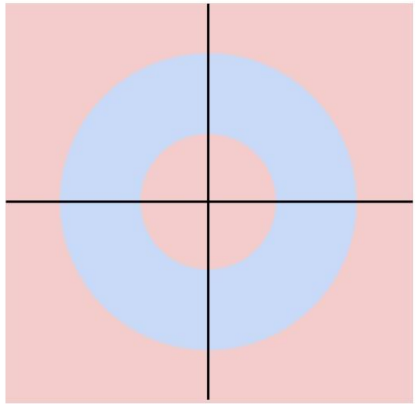
Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants



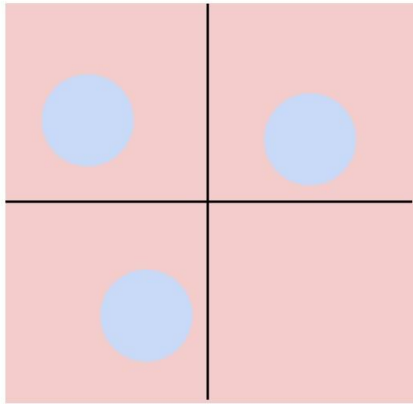
Class 1:
1 <= L2 norm <= 2

Class 2:
Everything else



Class 1:
Three modes

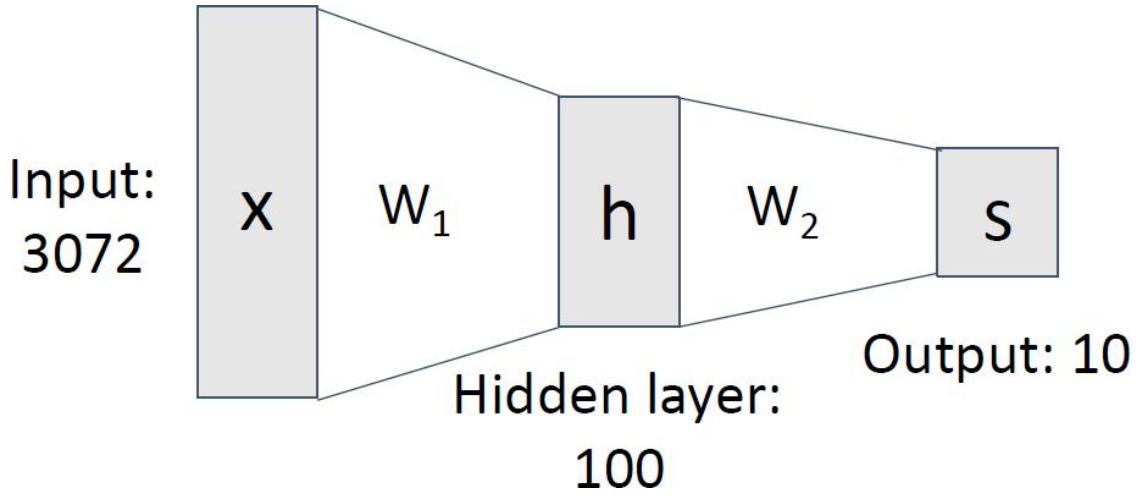
Class 2:
Everything else



X	Y	F(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

2-layer Neural Networks

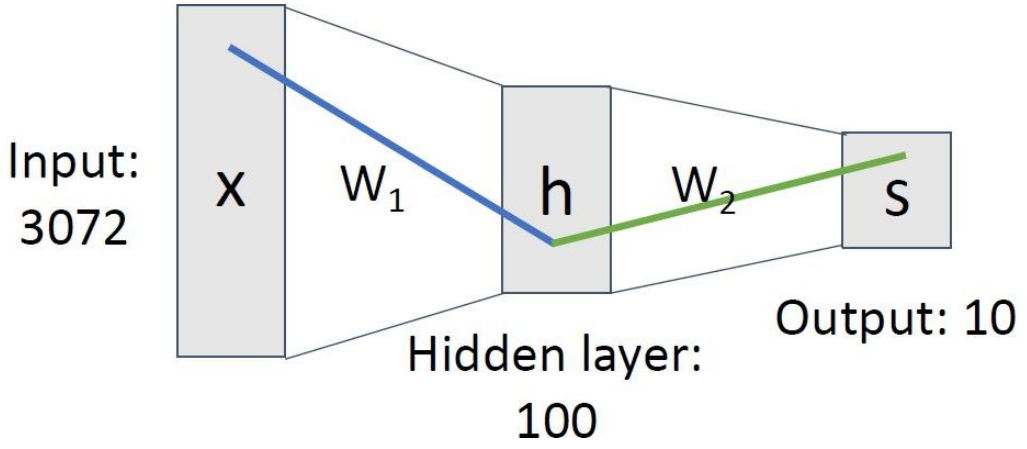
- **Input**: $x \in \mathbb{R}^D$
- **Output**: $W_2 max\,(0, W_1 x + b_1) + b_2 = f(x) \in \mathbb{R}^C$
- **Learnable Parameters**: $W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}, b_1 \in \mathbb{R}^H, b_2 \in \mathbb{R}^C$



2-layer Neural Networks

- **Input**: $x \in \mathbb{R}^D$
- **Output**: $W_2 max\,(0, W_1 x + b_1) + b_2 = f(x) \in \mathbb{R}^C$
- **Learnable Parameters**: $W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}, b_1 \in \mathbb{R}^H, b_2 \in \mathbb{R}^C$

Element (i, j) of W_1 gives the effect on h_j from x_i



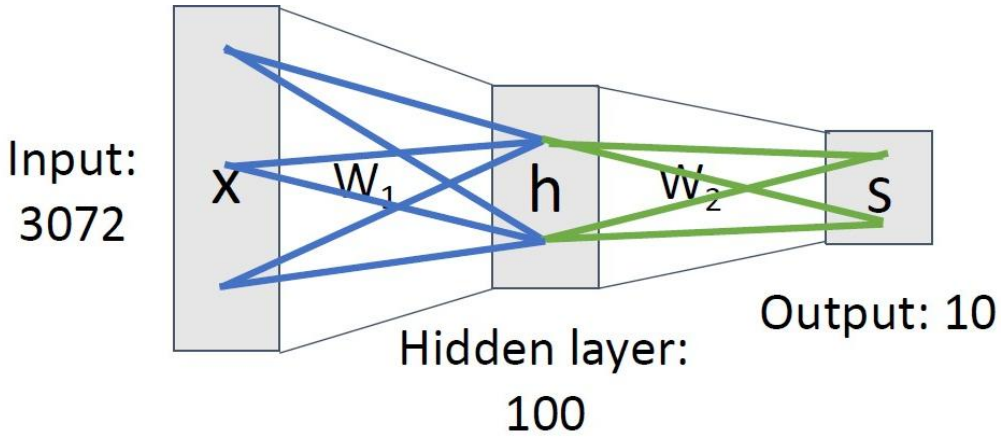
Element (j, k) of W_2 gives the effect on s_k from h_j

2-layer Neural Networks

- **Input**: $x \in \mathbb{R}^D$
- **Output**: $W_2 max\,(0, W_1 x + b_1) + b_2 = f(x) \in \mathbb{R}^C$
- **Learnable Parameters**: $W_1 \in \mathbb{R}^{H \times D}, W_2 \in \mathbb{R}^{C \times H}, b_1 \in \mathbb{R}^H, b_2 \in \mathbb{R}^C$

Element (i, j) of W_1 gives the effect on h_j from x_i

All elements of x affect all elements h

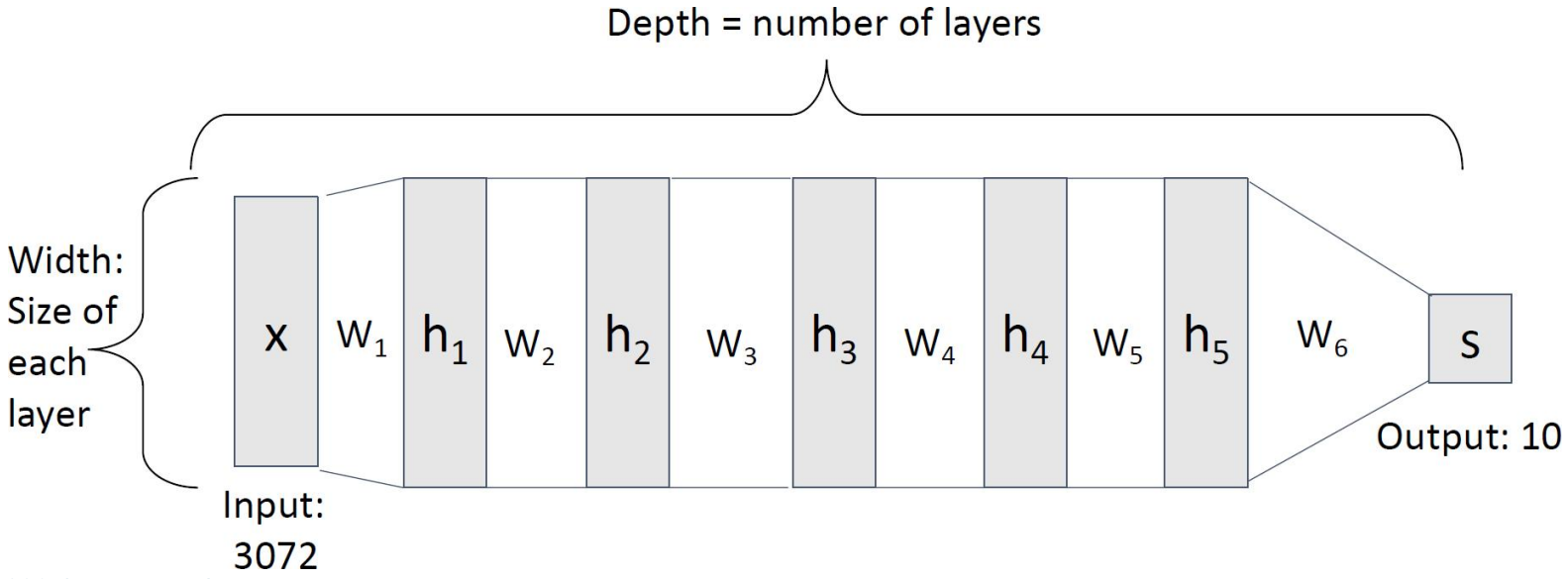


Element (j, k) of W_2 gives the effect on s_k from h_j

All elements of h affect all elements s

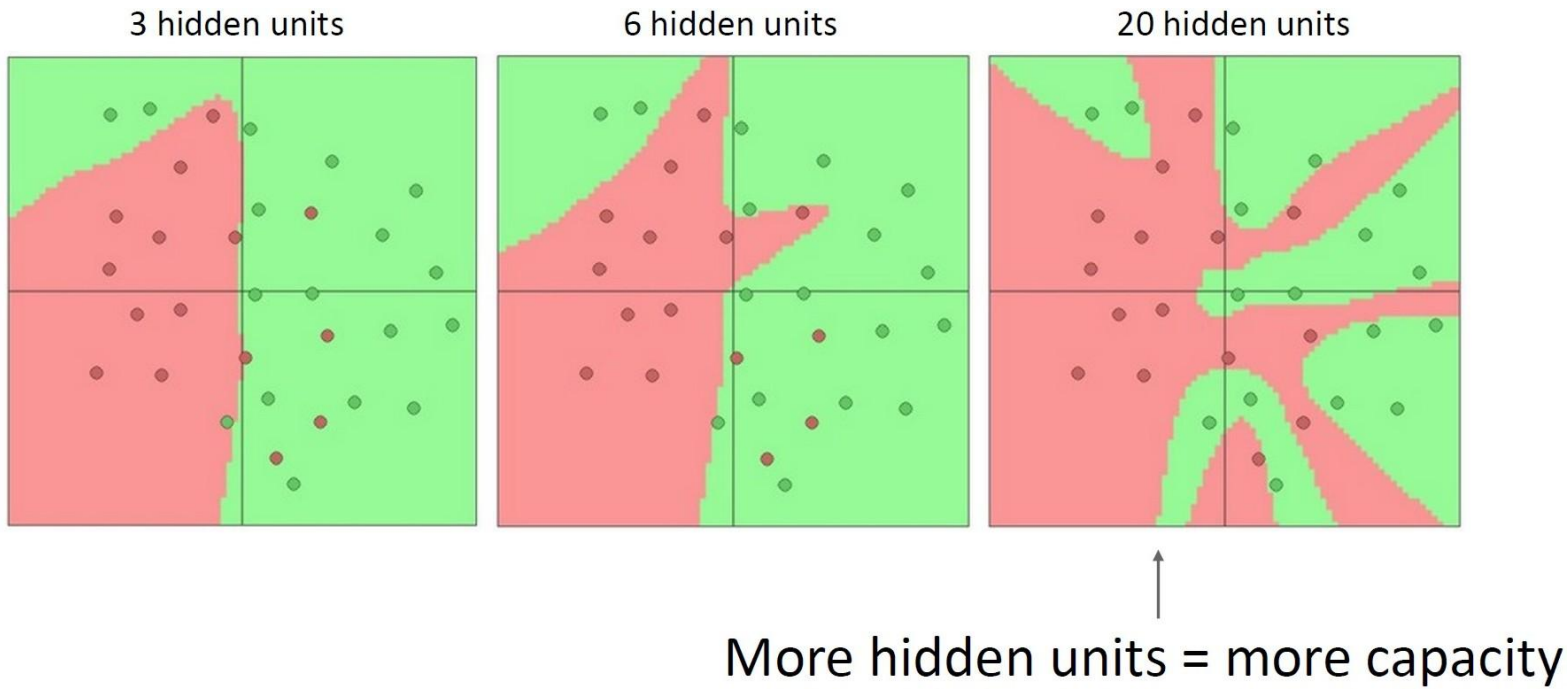
Deep Neural Networks

- **Input**: $x \in \mathbb{R}^D$
- **Output**: $f(x) \in \mathbb{R}^C$
- **Learnable Parameters**: $W_1, \ldots W_6, b_1, \ldots b_6$

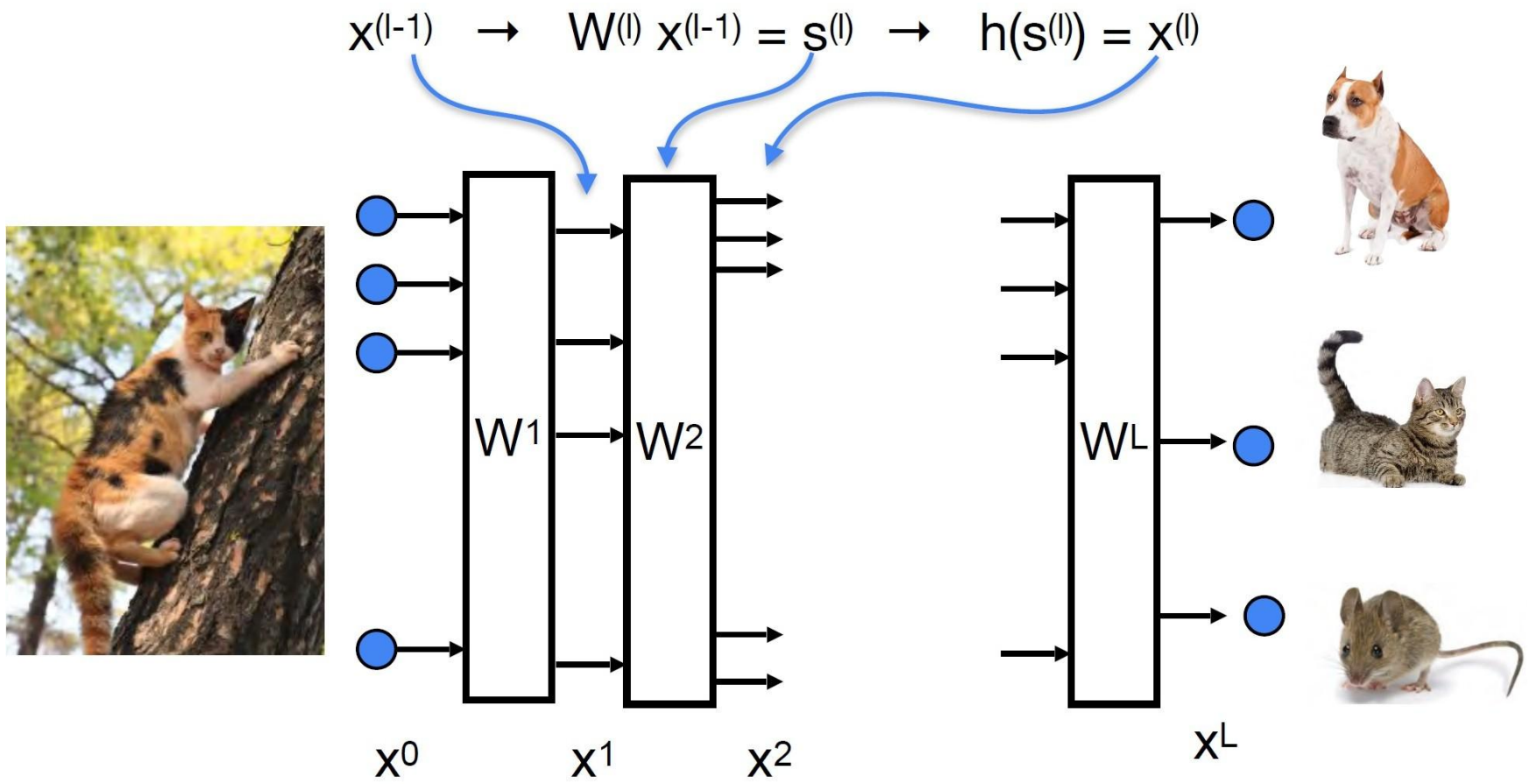


Deep Neural Networks

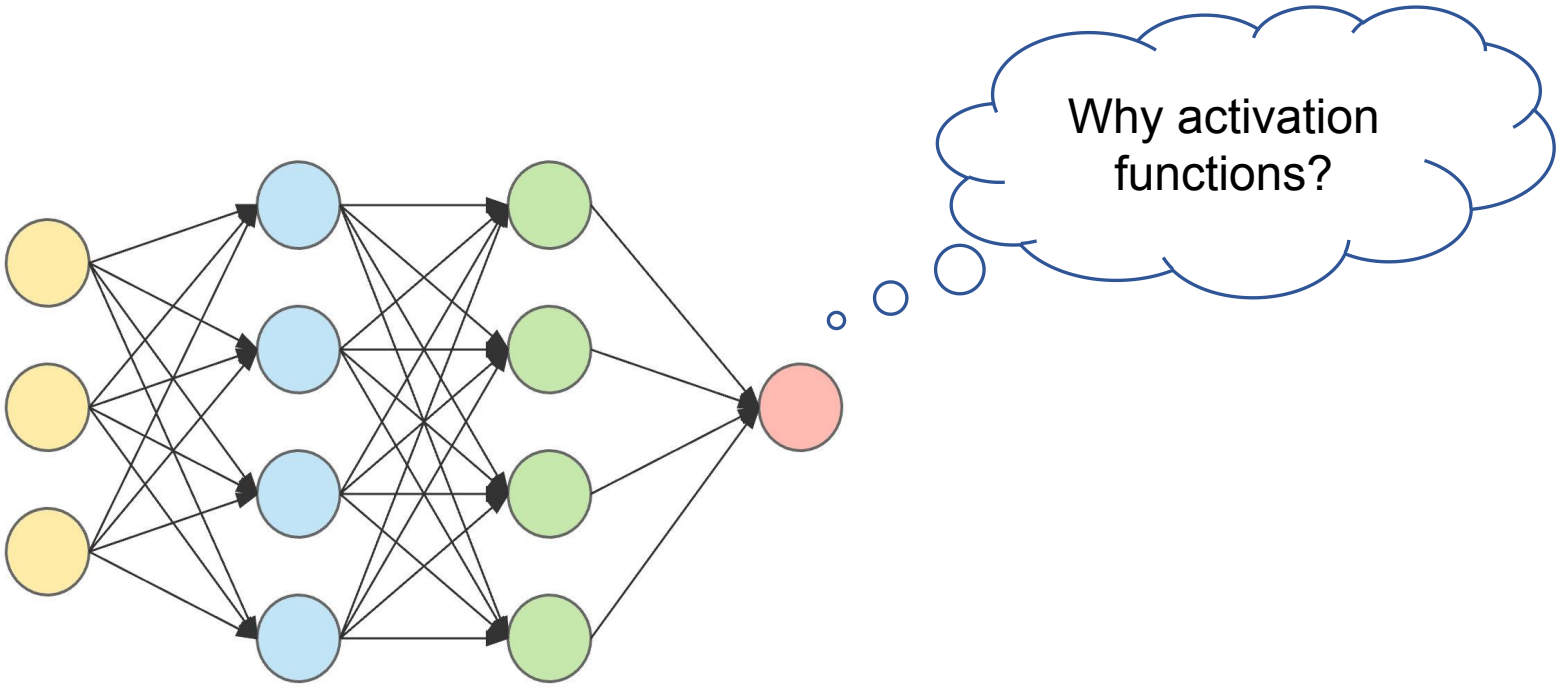
- Setting number of layers and their sizes
  - ✓ overfitting
  - ✓ generalization



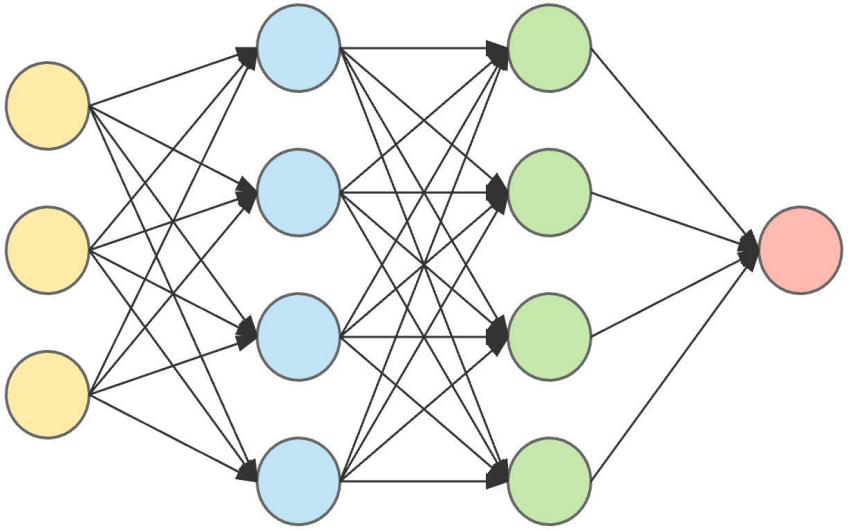
Deep Neural Networks Example



Activation Functions



Activation Functions



Suppose we remove all activation functions former layer output Y_k:

$$Y_k = W_k^T X_k + b_k$$

current layer output Y_{k+1}:

$$Y_{k+1} = W_{k+1}^T X_{k+1} + b_{k+1}$$

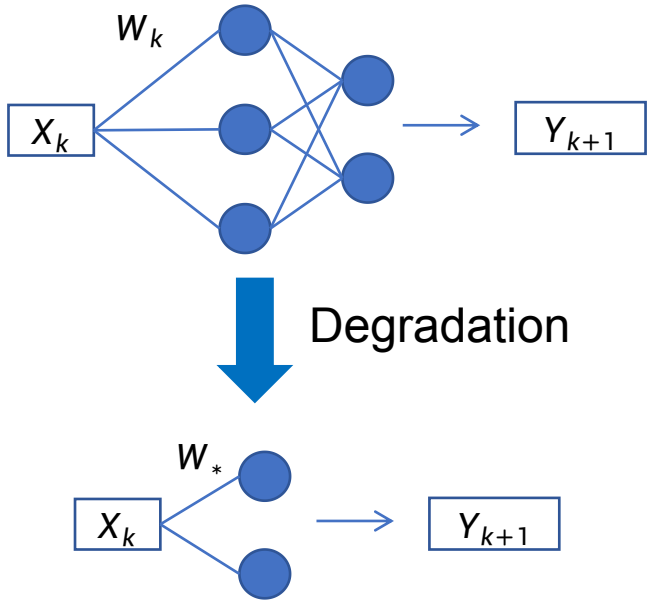
while $X_{k+1} = Y_k$

$$Y_{k+1} = W_{k+1}^T (W_k^T X_k + b_k) + b_{k+1}$$

Activation Functions

$$Y_{k+1} = W_{k+1}^T(W_k^T X_k + b_k) + b_{k+1}$$

$$Y_{k+1} = W_*^T X_k + b_*$$



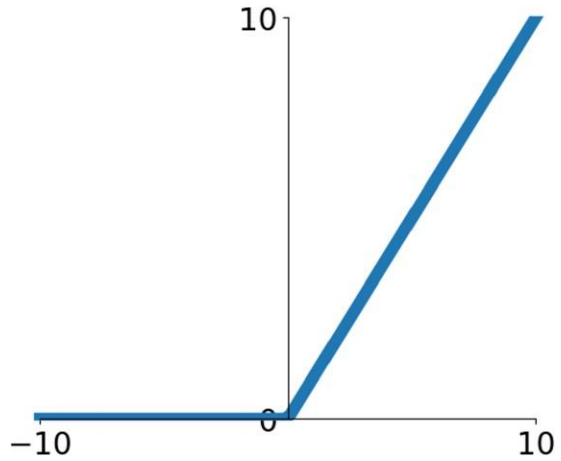
Conclusion: The linear superposition property makes stacking multiple linear layers meaningless, as they can always be reduced to a single layer.

Activation functions are necessary !

Activation Functions

- **Input**: $x \in \mathbb{R}^D$
- **Output**: $W_2 max(0, W_1 x + b_1) + b_2 = f(x) \in \mathbb{R}^C$

The function $ReLU(z) = \max(0, z)$ is called "**Rectified Linear Unit**"



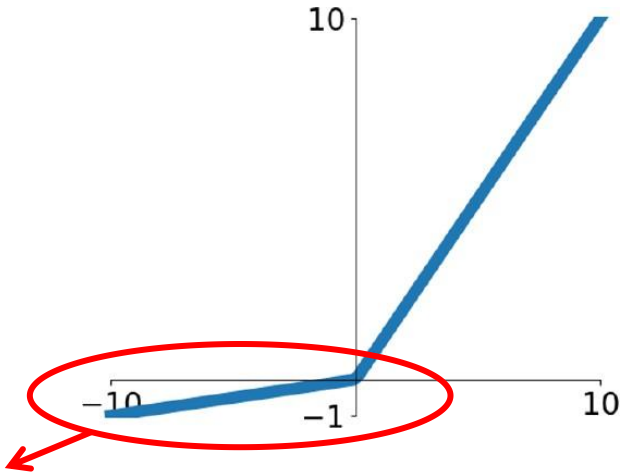
Activation Functions

- Pros and Cons to using the ReLUs:
  - ✓ **Sparsity**: Zero output for negative inputs promotes network **sparsity and generalization**.
  - ✓ **Mitigates Vanishing Gradients**: Gradient is exactly 1 in the positive region, **preventing decay** in deep networks.
  - ✓ **Dying ReLU Problem**: Neurons can get "stuck" outputting 0, making **gradients permanently zero**.

Activation Functions

- **Leaky ReLU**: Leaky ReLUs are one attempt to fix the “dying ReLU” problem.

$$\text{LeakyRELU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \text{negative\_slope} \times x, & \text{otherwise} \end{cases}$$



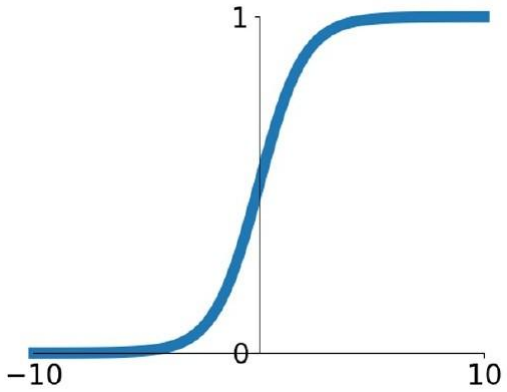
Adds a **small slope** to negative inputs to prevent "death"

Activation Functions

- **Sigmoid**:

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

- Sigmoids saturate and **kill gradients**. When the neuron's activation saturates at either tail of 0 or 1, the gradient at these regions is almost zero

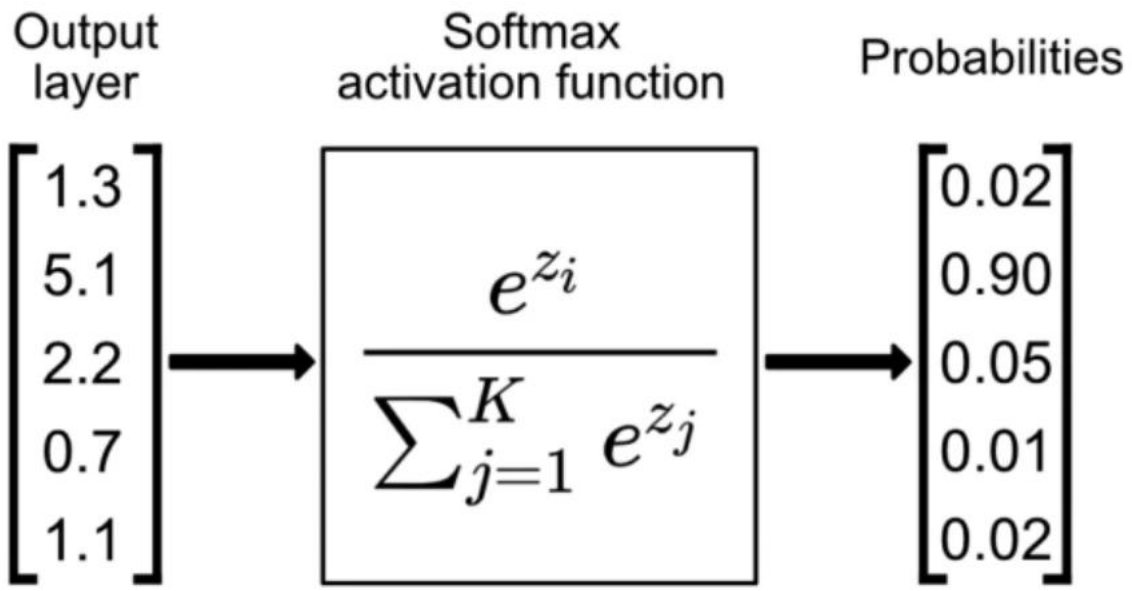


Activation Functions

- Pros and Cons to using the Sigmoid:
  - ✓ **Probabilistic Interpretation**: Directly outputs values (0, 1) interpretable as probabilities, so it's ideal for for **binary classification**.
  - ✓ **Smooth Gradient**: **Differentiable everywhere**, enabling gradient-based optimization.
  - ✓ **Vanishing Gradients**: Gradients ≈ 0 for extreme inputs ($|x| \gg 0$), **stalling learning** in deep networks.

Activation Functions

- **Softmax**:



$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

The sum is 1.

Activation Functions

- Pros and Cons to using the Softmax:
  - ✓ **Probabilistic Interpretation**: The output directly represents the probability of a sample belonging to each class, making it suitable for **multi-class classification tasks**.
  - ✓ **Gradient Behavior**: The derivative involves interactions between the probabilities of different classes, enabling **efficient gradient computation** during back propagation.

Activation Functions

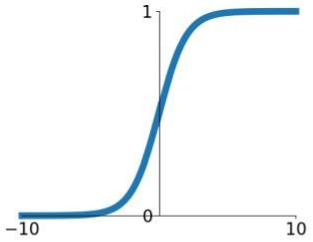
ReLU	Sigmoid	Softmax
$\max(0, x)$	$\frac{1}{1 + e^{-x}}$	$\frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$
• Sparse Activation • Solving Vanishing Gradient • Dying Neurons	• Smooth Function • Vanishing Gradient • Binary Classification	• Exponential Function • Probability Distribution • Multi—class classification

Activation Functions

- Other functions:

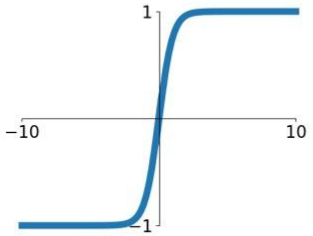
Sigmoid

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



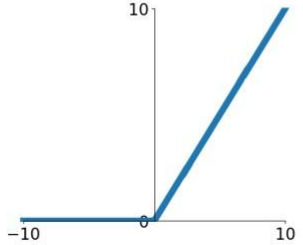
tanh

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$



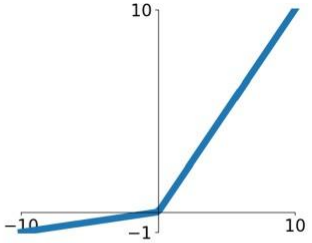
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.2x, x)$$

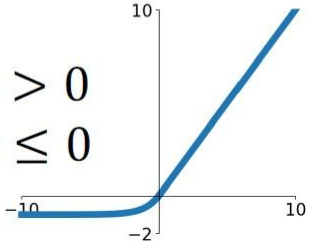


Softplus

$$\log(1 + \exp(x))$$

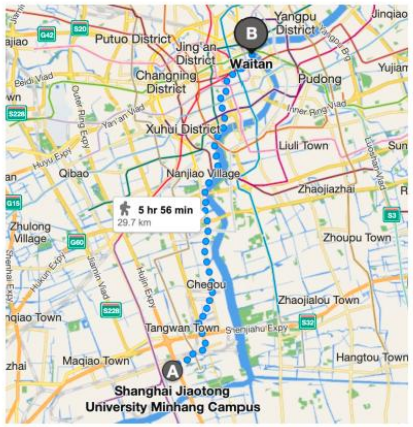
ELU

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha(\exp(x) - 1), & x \leq 0 \end{cases}$$



Summary: an example

- Suppose we want to predict the commute time into city center.



Dist (km)	Commute Time (min)
10.1	18
2.9	5
19.3	29
14.5	23
35.2	39.5

$$\hat{y} = f(x;\theta) = \theta_0 + \theta_1 x_1$$

$$\hat{y} = X\theta = \begin{bmatrix} 1 & 10.1 \\ 1 & 2.9 \\ 1 & 19.3 \\ 1 & 14.5 \\ 1 & 35.2 \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \Longrightarrow y = \begin{bmatrix} 18 \\ 5 \\ 29 \\ 23 \\ 39.5 \end{bmatrix}$$

Features: $X \in \mathbb{R}^{5\times 2}$ **Parameters:** $\theta \in \mathbb{R}^2$

- Define the loss function:

$$\mathcal{L}(\theta) = \frac{1}{2}\sum_{i=1}^{N}(f(x_i;\theta) - y_i)^2$$
$$= \frac{1}{2}(X\theta - y)^T(X\theta - y)$$

Find the parameters that can minimize this objective?

Summary: An Example

- Compute the partial derivative:

$$
\begin{aligned}
\nabla_\theta \mathcal{L}(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - y)^T(X\theta - y) \\
&= \frac{1}{2}\nabla_\theta((X\theta)^T X\theta - (X\theta)^T y - y^T(X\theta) + y^T y) && (1) \\
&= \frac{1}{2}\nabla_\theta(\theta^T(X^T X)\theta - y^T(X\theta) - y^T(X\theta)) && (2) \\
&= \frac{1}{2}\nabla_\theta(\theta^T(X^T X)\theta - 2(X^T y)^T\theta) && (3) \\
&= \frac{1}{2}(2X^T X\theta - 2X^T y) && (4) \\
&= X^T X\theta - X^T y && (5)
\end{aligned}
$$

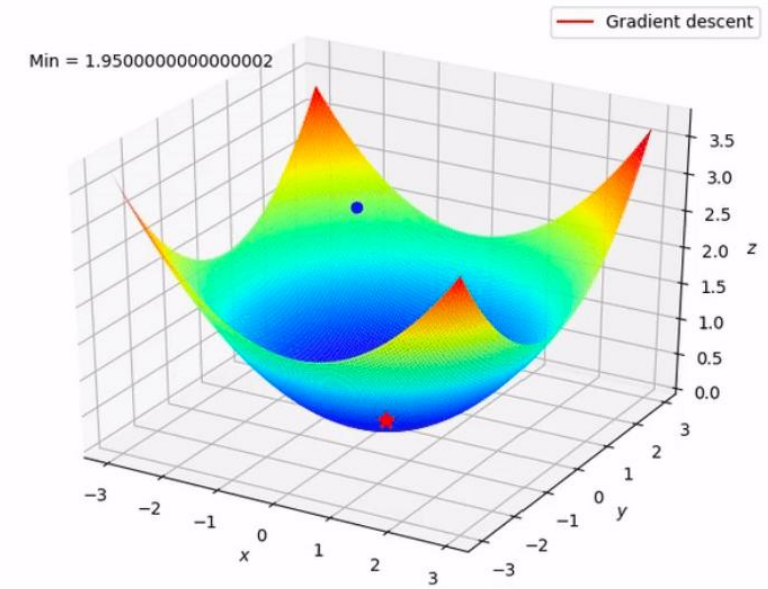
$$\nabla_\theta \mathcal{L}(\theta) = 0 \quad \Longrightarrow \quad \theta = (X^T X)^{-1} X^T y$$

Normal Equations

- Gradient Decent:

$$\nabla_\theta \mathcal{L}(\theta) = X^T X\theta - X^T y$$

$$\theta_j := \theta_j - \alpha \nabla_\theta \mathcal{L}(\theta)$$



Summary: An Example

- **What other variables would be useful?**
  - ✓ Distance to city center,
  - ✓ Day of the week,
  - ✓ Transportation,
  - ✓ etc.

Dist (km)	Day	Trans.	Commute Time (min)
2.7	Friday	Bus	25
4.1	Tuesday	Bike	20
2.9	Monday	Bus	18
3.5	Saturday	Bus	22
3.8	Friday	Car	20

⇨

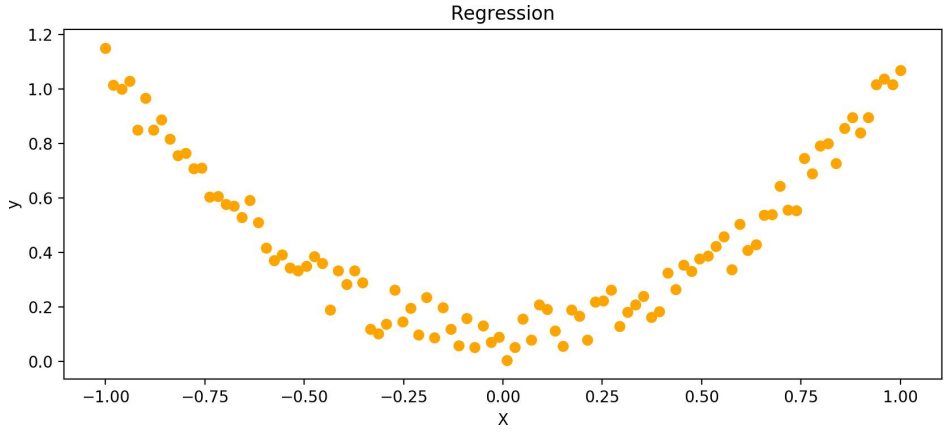
Dist (km)	Day	Trans.	Commute Time (min)
2.7	5	1	25
4.1	2	2	20
2.9	1	1	18
3.5	6	1	22
3.8	5	3	25

Represent the day as 1-7
Represent the Transportation type as: Bus : 1, Bike : 2, Car: 3

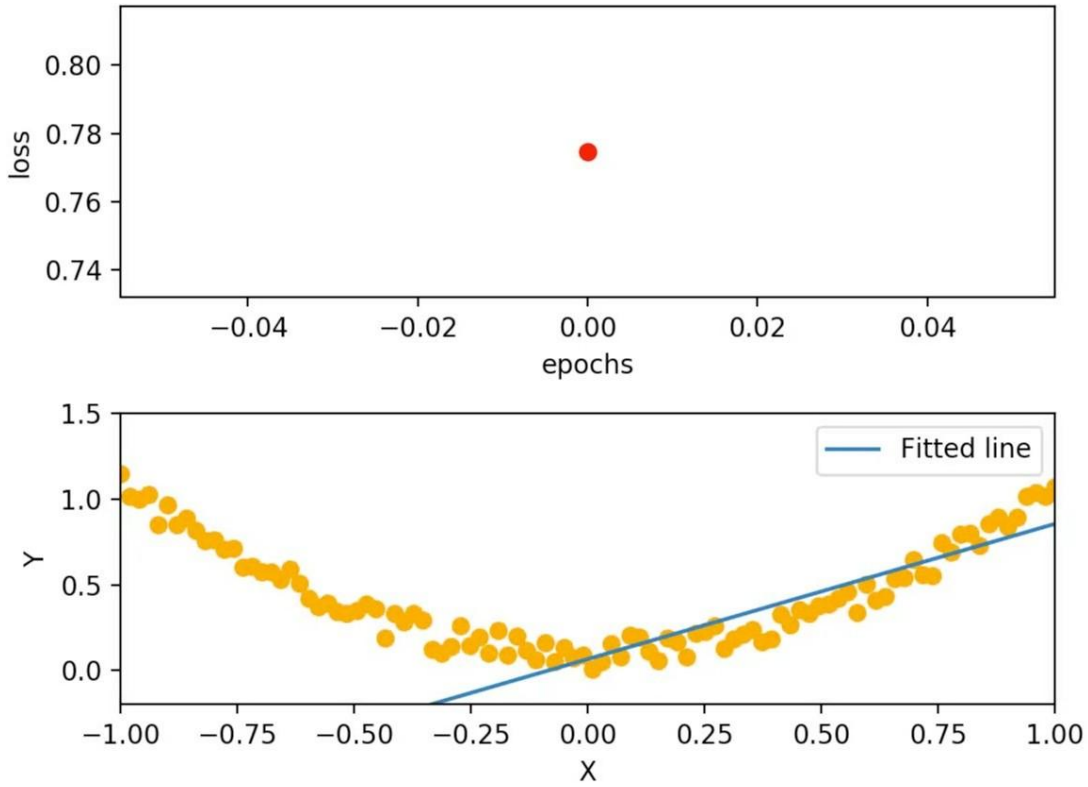
$$\hat{y} = X\theta = \begin{bmatrix} 1 & 2.7 & 5 & 1 \\ 1 & 4.1 & 2 & 2 \\ 1 & 2.9 & 1 & 1 \\ 1 & 3.5 & 6 & 1 \\ 1 & 3.8 & 5 & 3 \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Summary: An Example

- Fitting Non-Linear Model



$$\hat{y} = X\theta = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ 1 & x_5 \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

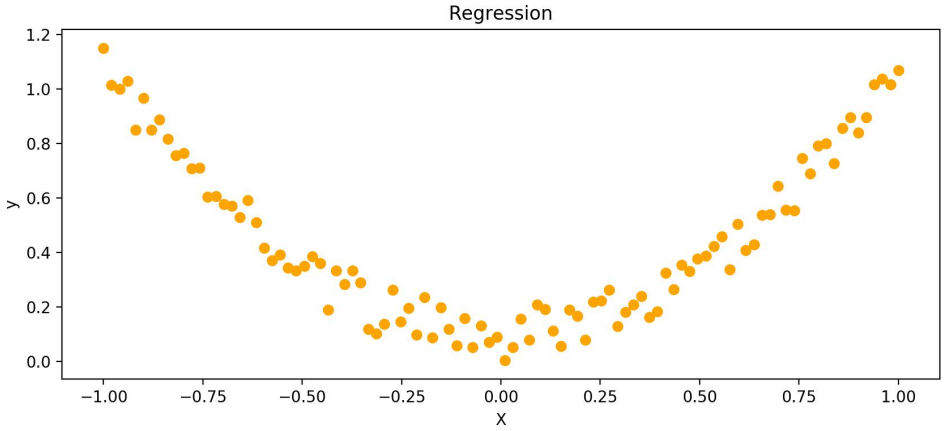


linear model is too limited, right?

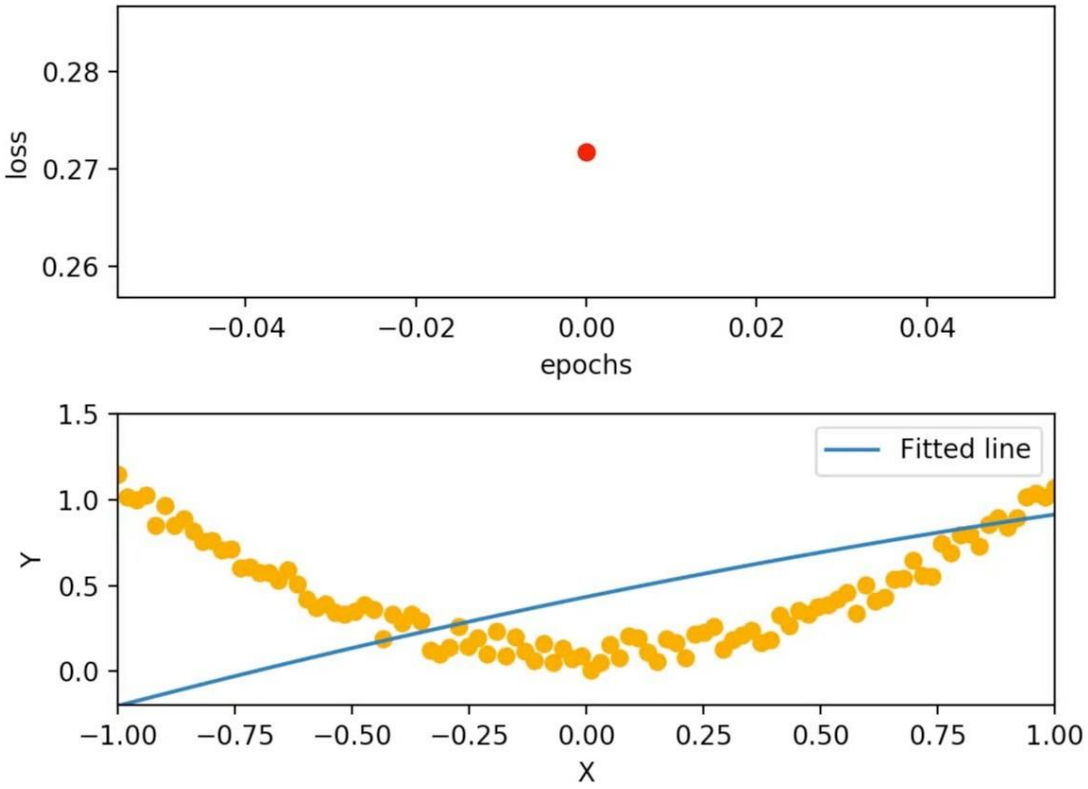
Maybe not, the key is really how to design features.

Summary: An Example

- Fitting Non-Linear Model



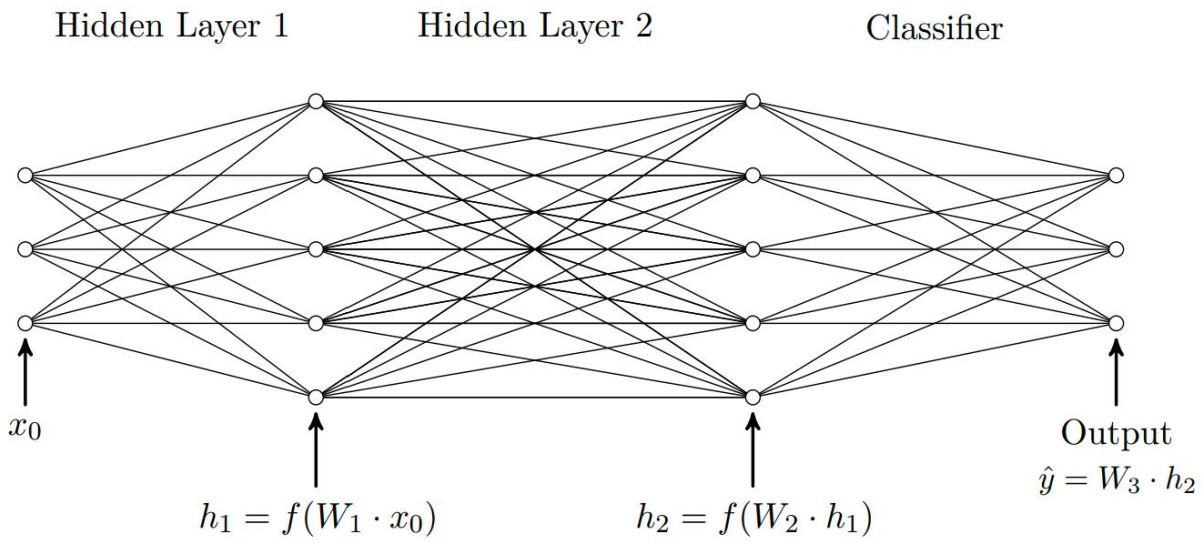
$$\hat{y} = X\theta = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ 1 & x_4 & x_4^2 \\ 1 & x_5 & x_5^2 \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$



Still linear model, just design some better features.

Summary: An Example

- Fitting MLPs:
  - ✓ Can we learn “good” features from data automatically?
  - ✓ Yes ! **Multi-layer Perceptrons (MLPs)** :



- Define loss function:

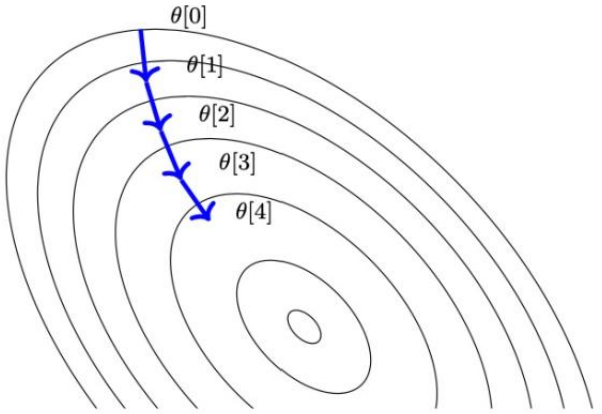
$$\mathcal{L}(W) = \frac{1}{2}(\hat{y} - y)^T(\hat{y} - y)$$

- GD with derivatives (**Back Propagation**)

$$W_1 := W_1 - \alpha \nabla_{W_1} \mathcal{L}(W)$$

$$W_2 := W_2 - \alpha \nabla_{W_2} \mathcal{L}(W)$$

$$W_3 := W_3 - \alpha \nabla_{W_3} \mathcal{L}(W)$$



上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Contents

- Introduction & Overview
- Multi-Layer Perceptron
- **Training Neural Networks**
- Case Analysis

Problem Restatement

- Given the training data $\mathrm{D} = \{\mathrm{x}^{(\mathrm{n})}, y^{(\mathrm{n})}\}$, $i \in [1, N]$.
- Given the neural networks $f(x, \theta)$ with parameters θ and loss functions 1.

How to solve this optimization problem?

- Compute the **empirical risk** to approximate the expected risk:

$$\mathcal{R}_{\mathcal{D}}^{emp}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(y^{(n)}, f(x^{(n)}, \theta))$$

- Minimizing the empirical loss leads to an optimization problem:

$$\theta^* = \arg\min_{\theta} \mathcal{R}_{\mathcal{D}}^{emp}(\theta)$$

Preliminary: Matrix Calculus

- The partial derivative of a scalar $y \in \mathbb{R}$ with respect to a vector $x \in \mathbb{R}^p$:

$$\frac{\partial y}{\partial \boldsymbol{x}} = [\frac{\partial y}{\partial x_1}, \cdots, \frac{\partial y}{\partial x_p}]^{\mathrm{T}}$$

- The partial derivative of a vector $y \in \mathbb{R}^q$ with respect to a vector $x \in \mathbb{R}^p$:

$$\frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_q}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial y_1}{\partial x_p} & \cdots & \frac{\partial y_q}{\partial x_p} \end{bmatrix} \in \mathbb{R}^{p \times q}$$

Preliminary: Chain Rules

- If $x \in \mathbb{R}, u = u(x) \in \mathbb{R}^s, g = g(u) \in \mathbb{R}^t$, then

$$\frac{\partial g}{\partial x} = \frac{\partial u}{\partial x}\frac{\partial g}{\partial u} \quad \in \mathbb{R}^{1\times t}.$$

- If $x \in \mathbb{R}^p, y = g(x) \in \mathbb{R}^s, z = f(y) \in \mathbb{R}^t$, then

$$\frac{\partial z}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial z}{\partial y} \quad \in \mathbb{R}^{p\times t}.$$

- If $X \in \mathbb{R}^{p\times q}$ is a matrix, $y = g(X) \in \mathbb{R}^s, z = f(y) \in \mathbb{R}$, then

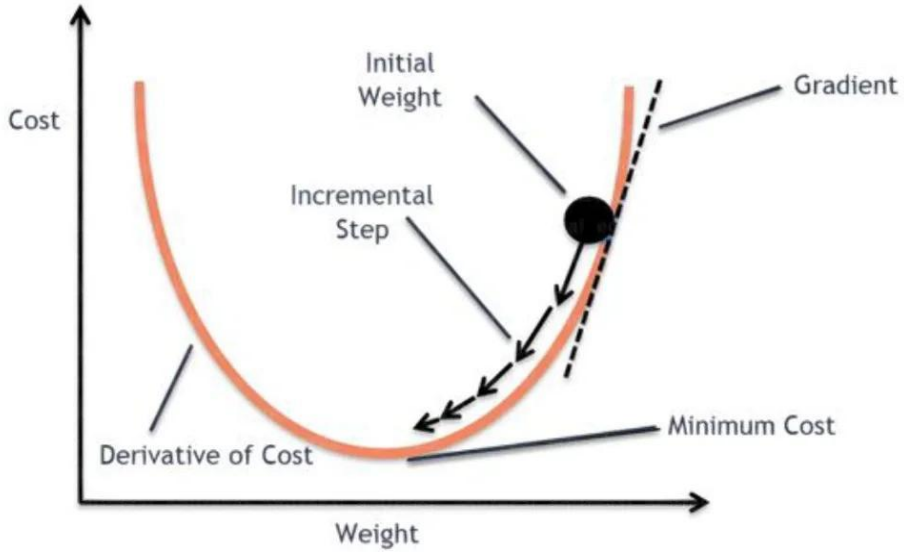
$$\frac{\partial z}{\partial X_{ij}} = \frac{\partial y}{\partial X_{ij}}\frac{\partial z}{\partial y} \quad \in \mathbb{R}.$$

Preliminary: Gradient Descent

- Following **negative gradient direction** of a function ensures the fastest decrease in the function value.

- By computing the derivative of the empirical risk $\mathcal{R}$ with respect to the learnable parameters W, we can determine the **update direction** for the parameters.

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial \mathcal{R}(W, \mathbf{b})}{\partial W^{(l)}}$$



Training MLPs

- Linear Regression didn't work

$$\hat{y} = X\theta = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \\ 1 & x_5 \end{bmatrix} \cdot \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

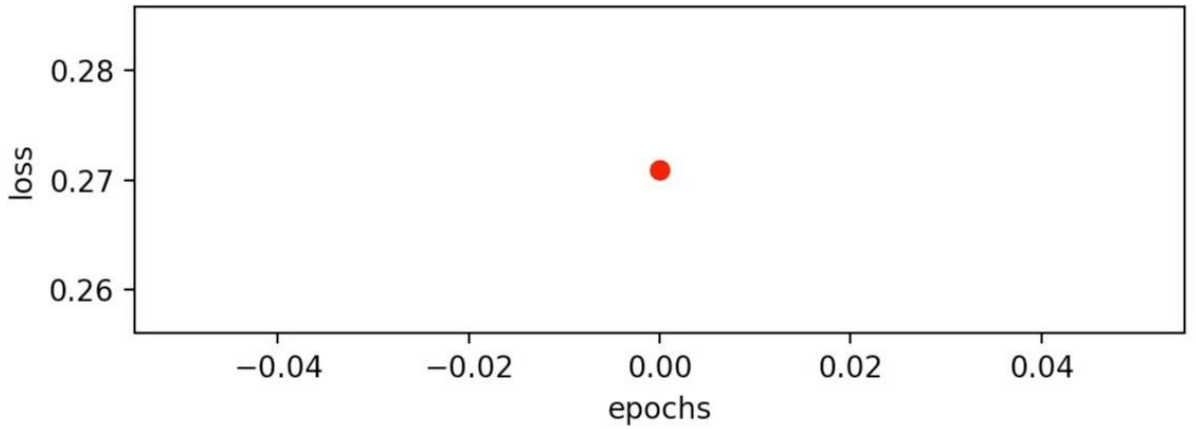
- What about MLPs?

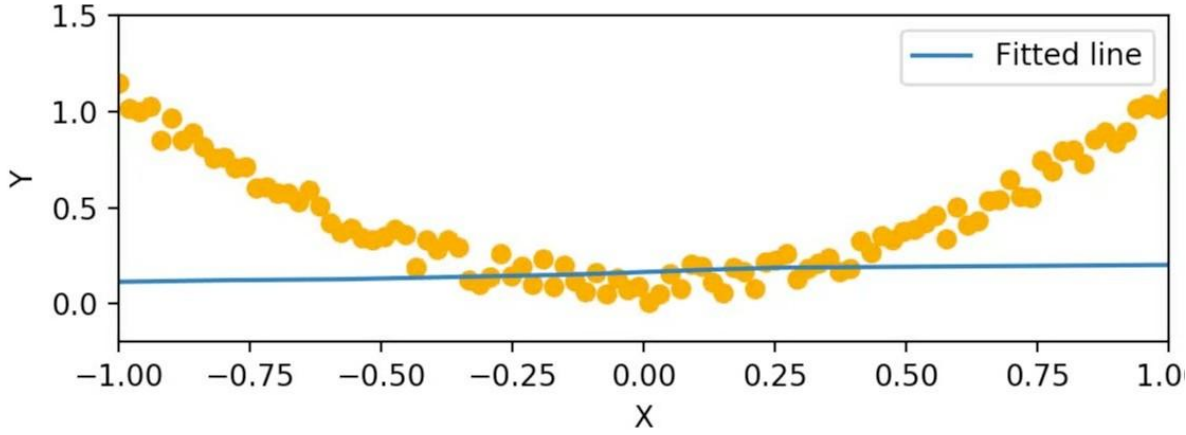
$$\begin{aligned} \hat{y} &= W_3 \cdot h_2 \\ &= W_3 \cdot f(W_2 \cdot h_1) \\ &= W_3 \cdot f(W_2 \cdot f(W_1 \cdot X)) \end{aligned} \qquad \mathcal{L}(W) = \frac{1}{2}(\hat{y} - y)^T(\hat{y} - y)$$

$$W_i := W_i - \alpha \nabla_{W_i} \mathcal{L}(W)$$

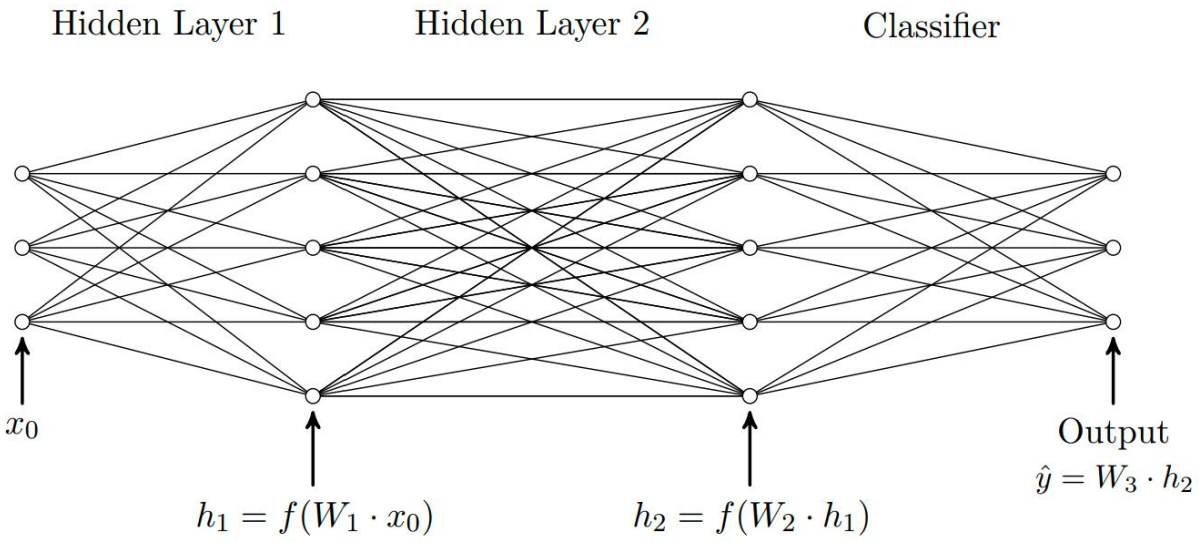
```python
# this is one way to define a network
class Net(torch.nn.Module):
    def __init__(self, n_feature=1, n_hidden=8, n_output=1):
        super(Net, self).__init__()
        self.hidden1 = torch.nn.Linear(n_feature, n_hidden)  # hidden layer
        self.hidden2 = torch.nn.Linear(n_hidden, n_hidden)  # hidden layer
        self.predict = torch.nn.Linear(n_hidden, n_output)  # output layer

    def forward(self, x):
        x = F.relu(self.hidden1(x))  # activation function for hidden layer
        x = F.relu(self.hidden2(x))  # activation function for hidden layer
        x = self.predict(x)  # linear output
        return x
```





Back Propagation



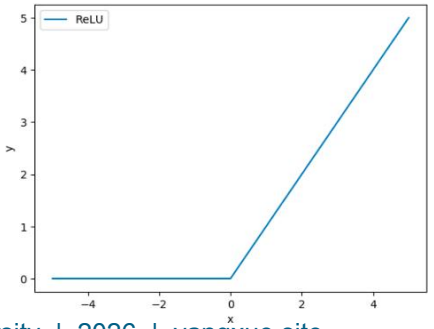
$$\mathcal{L}(W) = \frac{1}{2}(\hat{y} - y)^T(\hat{y} - y)$$

$$\begin{aligned}\hat{y} &= W_3 \cdot h_2 \\ &= W_3 \cdot f(W_2 \cdot h_1) \\ &= W_3 \cdot f(W_2 \cdot f(W_1 \cdot x_0))\end{aligned}$$

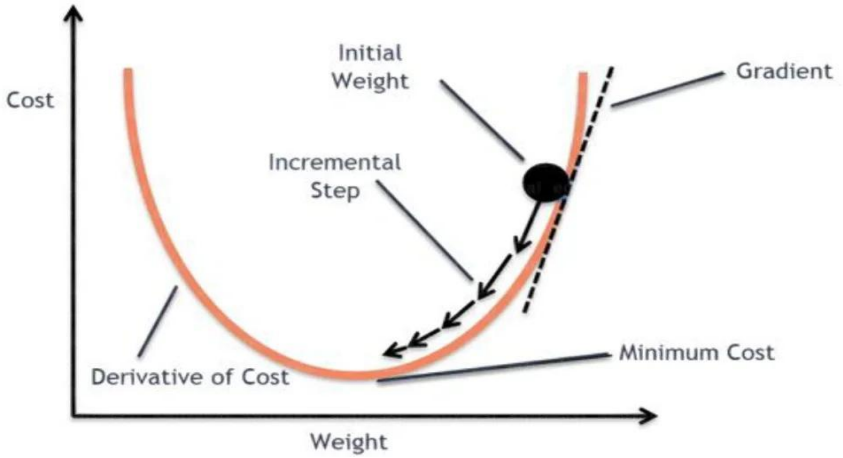
$$W_1 := W_1 - \alpha \nabla_{W_1} \mathcal{L}(W)$$

$$\begin{aligned}\nabla_{W_1} \mathcal{L}(W) &= \frac{\partial \mathcal{L}}{\partial h_1} \cdot \frac{\partial h_1}{\partial W_1} \\ &= \frac{\partial \mathcal{L}}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial W_1} \\ &= \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial W_1}\end{aligned}$$

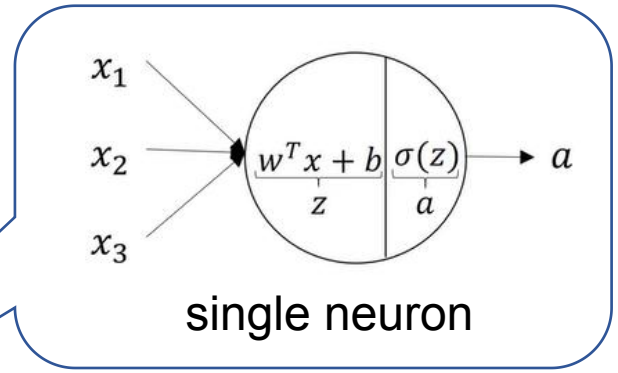
$\hat{y} - y \quad W_3 \quad f' \cdot W_2 \quad f' \cdot x_0$



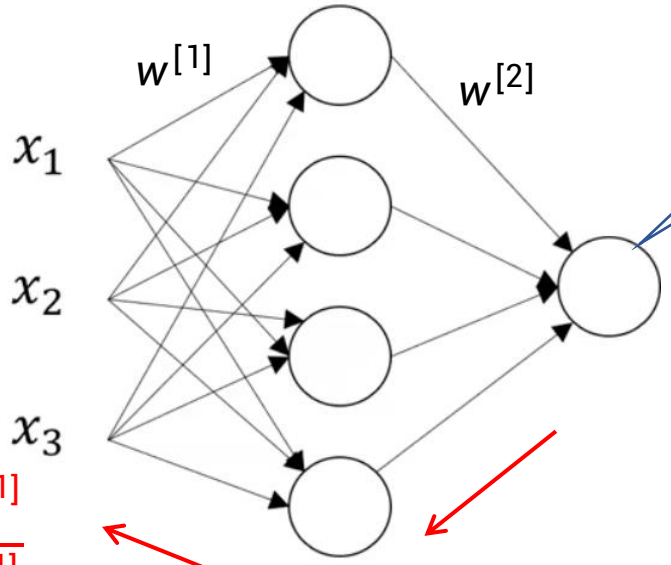
$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$



Back Propagation



Chain Rule!



$\hat{p}$ → Loss function J

$\frac{\partial J}{\partial \hat{p}}$ ← $\frac{\partial J}{\partial \hat{p}} \frac{\partial \hat{p}}{\partial z^{[2]}}$

$\frac{\partial J}{\partial \hat{p}} \frac{\partial \hat{p}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial w^{[2]}}$

$\frac{\partial J}{\partial \hat{p}} \frac{\partial \hat{p}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a_i^{[1]}}$

$\frac{\partial J}{\partial \hat{p}} \frac{\partial \hat{p}}{\partial z^{[2]}} \frac{\partial z_i^{[2]}}{\partial a_i^{[1]}} \frac{\partial a_i^{[1]}}{\partial z_i^{[1]}}$

$\frac{\partial J}{\partial \hat{p}} \frac{\partial \hat{p}}{\partial z^{[2]}} \frac{\partial z_i^{[2]}}{\partial a_i^{[1]}} \frac{\partial a_i^{[1]}}{\partial z_i^{[1]}} \frac{\partial z_i^{[1]}}{\partial w^{[1]}}$

→ Forward propagation

← Backward propagation

Back Propagation

- Input sample: $\vec{a} = (x_1, x_2)$
- The parameters of the 1st layer:

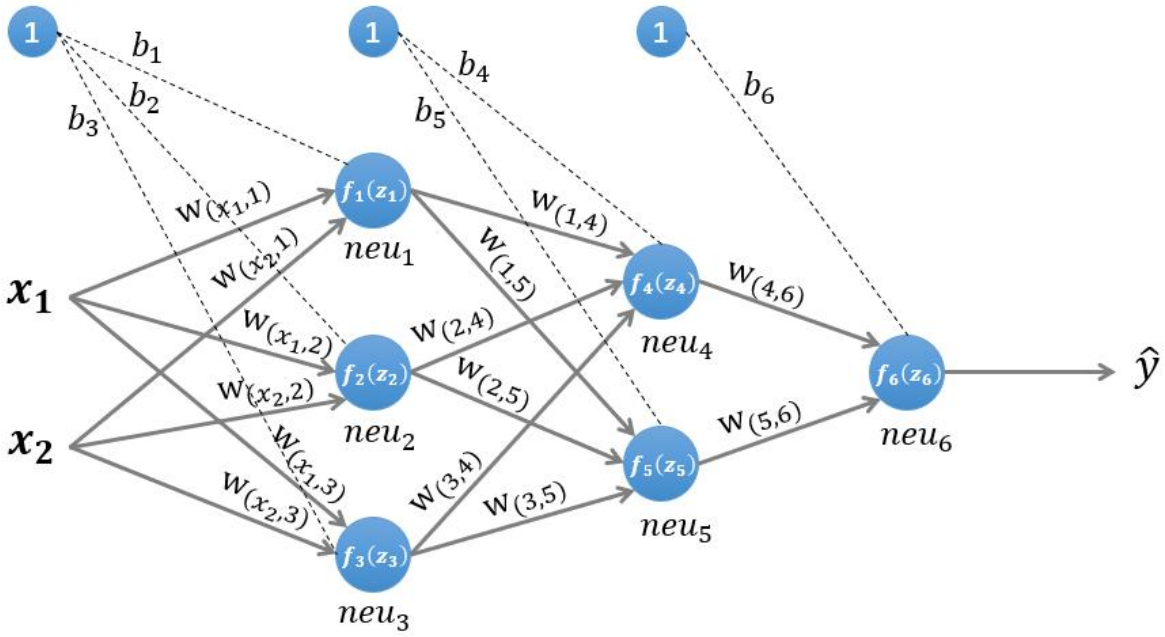
$$W^{(1)} = \begin{bmatrix} w_{(x_1,1)} & w_{(x_2,1)} \\ w_{(x_1,2)} & w_{(x_2,2)} \\ w_{(x_1,3)} & w_{(x_2,3)} \end{bmatrix}, \; b^{(1)} = [b_1, b_2, b_3]^T$$

- The parameters of the 2nd layer:

$$W^{(2)} = \begin{bmatrix} w_{(1,4)} & w_{(2,4)} & w_{(3,4)} \\ w_{(1,5)} & w_{(2,5)} & w_{(3,5)} \end{bmatrix}, \; b^{(2)} = [b_4, b_5]^T$$

- The parameters of the 3rd layer:

$$W^{(3)} = [w_{(4,6)}, w_{(5,6)}], \; b^{(3)} = [b_6]$$



https://panchuang.net/2018/03/20/backpropagation-algorithm/

Back Propagation

- The input of the first layer is:

$$z^{(1)} = W^{(1)} * (\vec{a})^T + (b^{(1)})^T$$

✓ The input of *neu1* is:

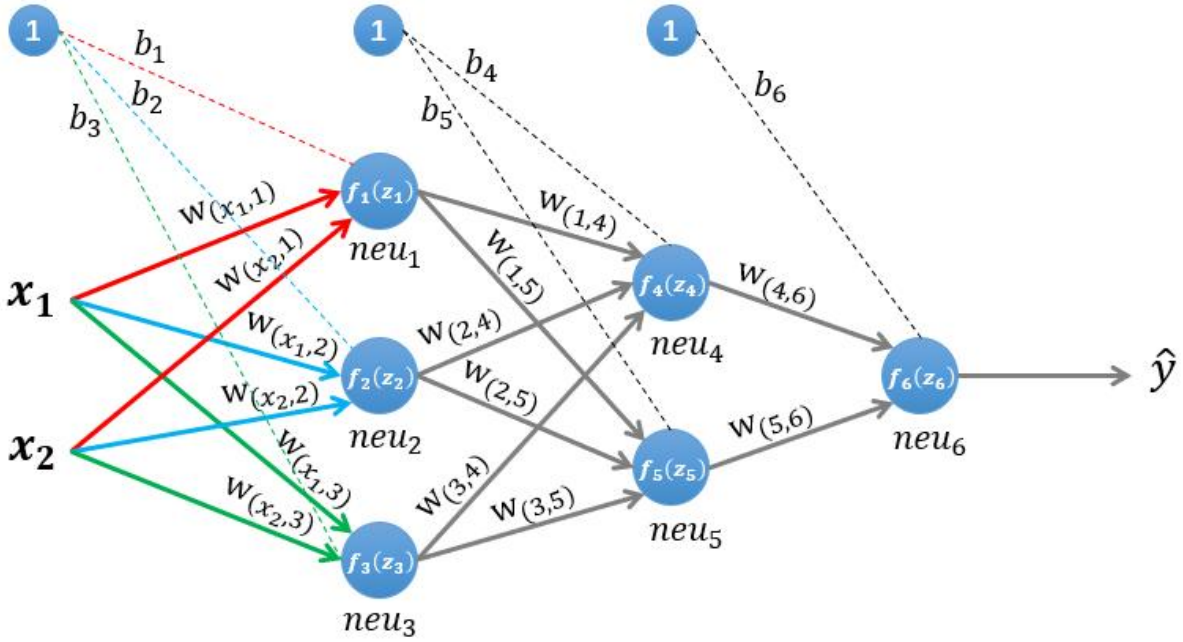
$$z_1 = w_{(x_1,1)} * x_1 + w_{(x_2,1)} * x_2 + b_1$$

✓ The input of *neu2* is:

$$z_2 = w_{(x_1,2)} * x_1 + w_{(x_2,2)} * x_2 + b_2$$

✓ The input of *neu3* is:

$$z_3 = w_{(x_1,3)} * x_1 + w_{(x_2,3)} * x_2 + b_3$$



Back Propagation

- The output of the first layer is:

$$f^{(1)}(z^{(1)}) = f^{(1)}(W^{(1)} * (\vec{a})^T + (b^{(1)})^T)$$

✓ The output of *neu1* is:

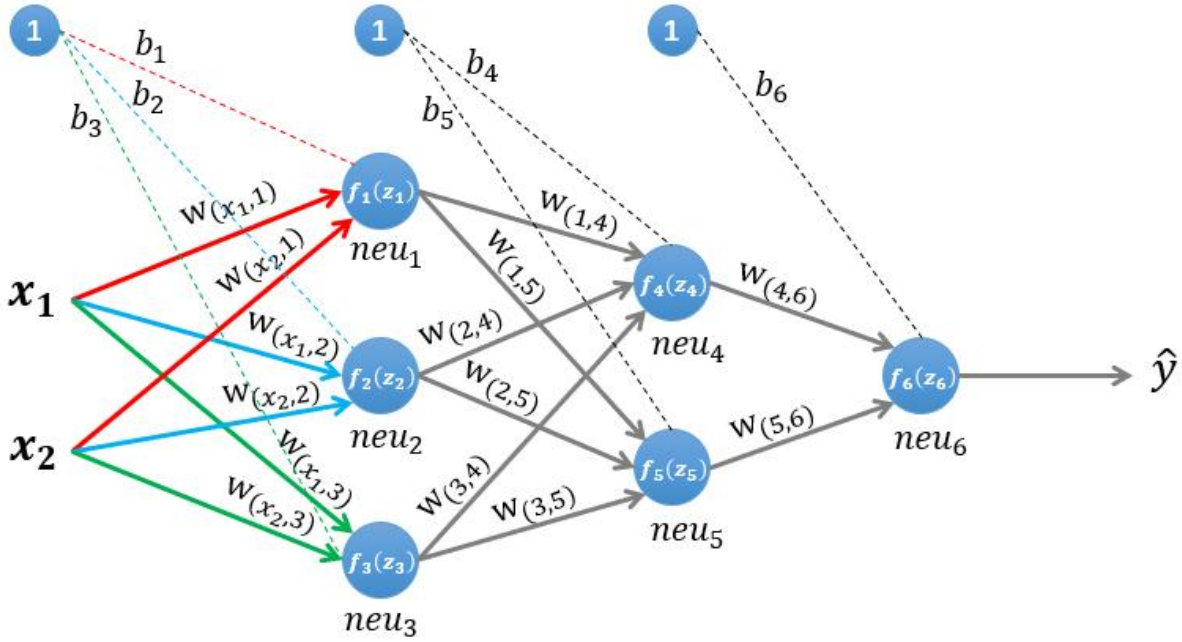
$$f_1(z_1) = f_1(w_{(x_1,1)} * x_1 + w_{(x_2,1)} * x_2 + b_1)$$

✓ The output of *neu2* is:

$$f_2(z_2) = f_2(w_{(x_1,2)} * x_1 + w_{(x_2,2)} * x_2 + b_2)$$

✓ The output of *neu3* is:

$$f_3(z_3) = f_3(w_{(x_1,3)} * x_1 + w_{(x_2,3)} * x_2 + b_3)$$



Back Propagation

- The input of the second layer is:

$$z^{(2)} = W^{(2)} * f^{(1)}([z_1, z_2, z_3]^T) + (b^{(2)})^T$$

✓ The input of *neu4* is:

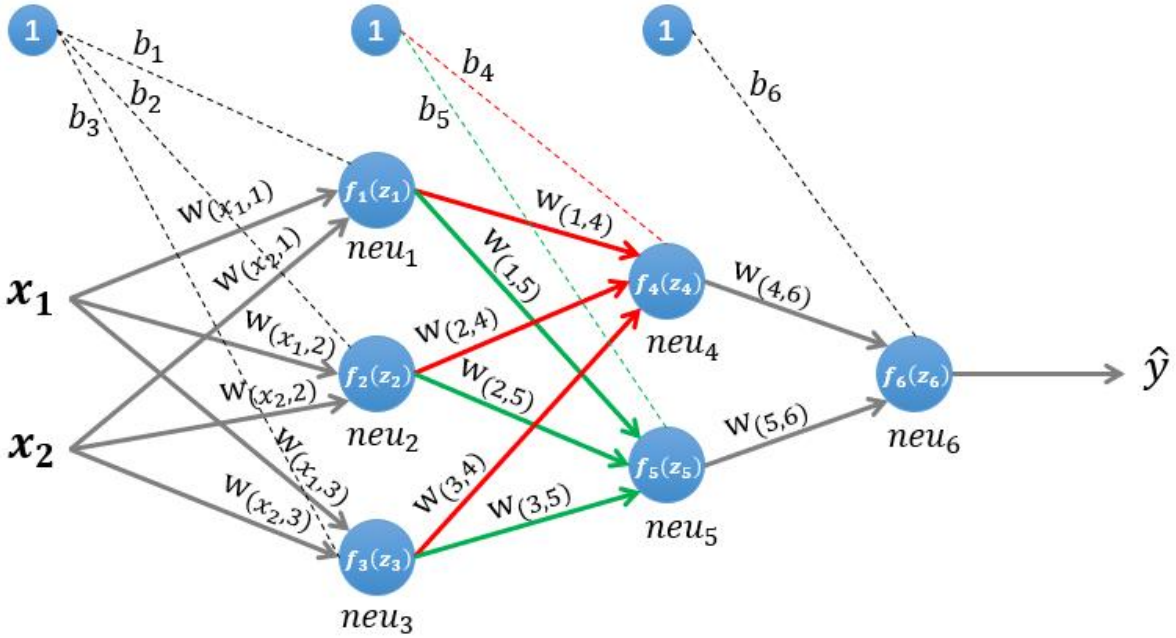
$$z_4 = w_{(1,4)} * f_1(z_1) + w_{(2,4)} * f_2(z_2)$$

$$+ w_{(3,4)} * f_3(z_3) + b_4$$

✓ The input of *neu5* is:

$$z_5 = w_{(1,5)} * f_1(z_1) + w_{(2,5)} * f_2(z_2)$$

$$+ w_{(3,5)} * f_3(z_3) + b_5$$



Back Propagation

- The output of the second layer is:

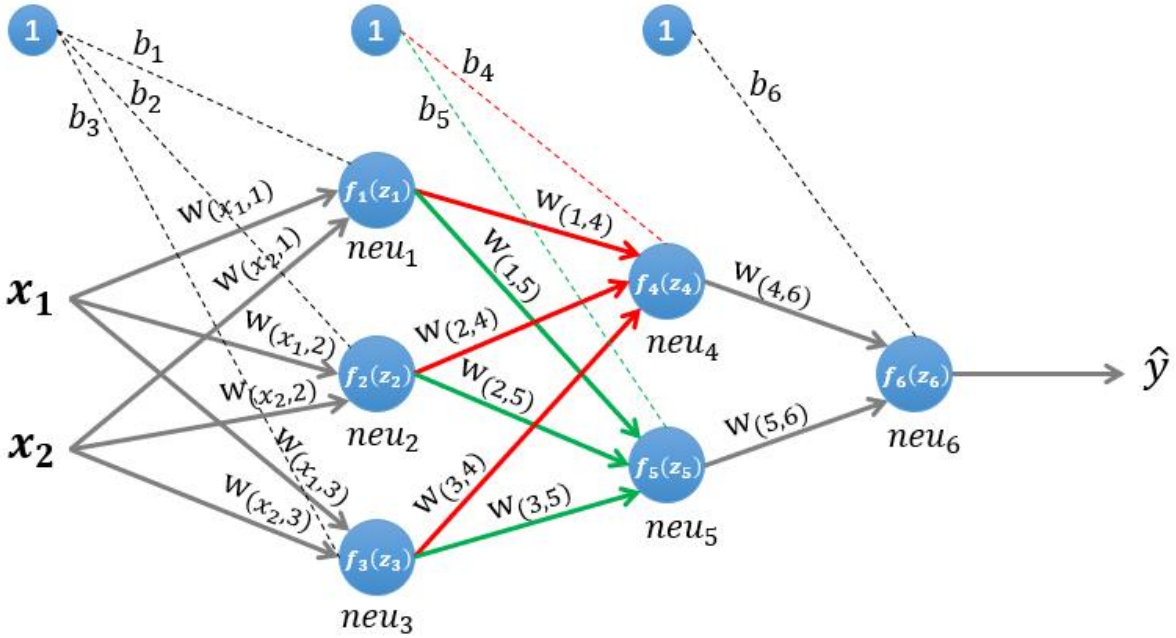
$$f^{(2)}(z^{(2)}) = f^{(2)}(W^{(2)} * f^{(2)}([z_1, z_2, z_3]^T) + (b^{(2)})^T)$$

✓ The output of *neu4* is:

$$f_4(z_4) = f_4(w_{(1,4)} * f_1(z_1) + w_{(2,4)} * f_2(z_2) + w_{(3,4)} * f_3(z_3) + b_4)$$

✓ The output of *neu5* is:

$$f_5(z_5) = f_5(w_{(1,5)} * f_1(z_1) + w_{(2,5)} * f_2(z_2) + w_{(3,5)} * f_3(z_3) + b_5)$$



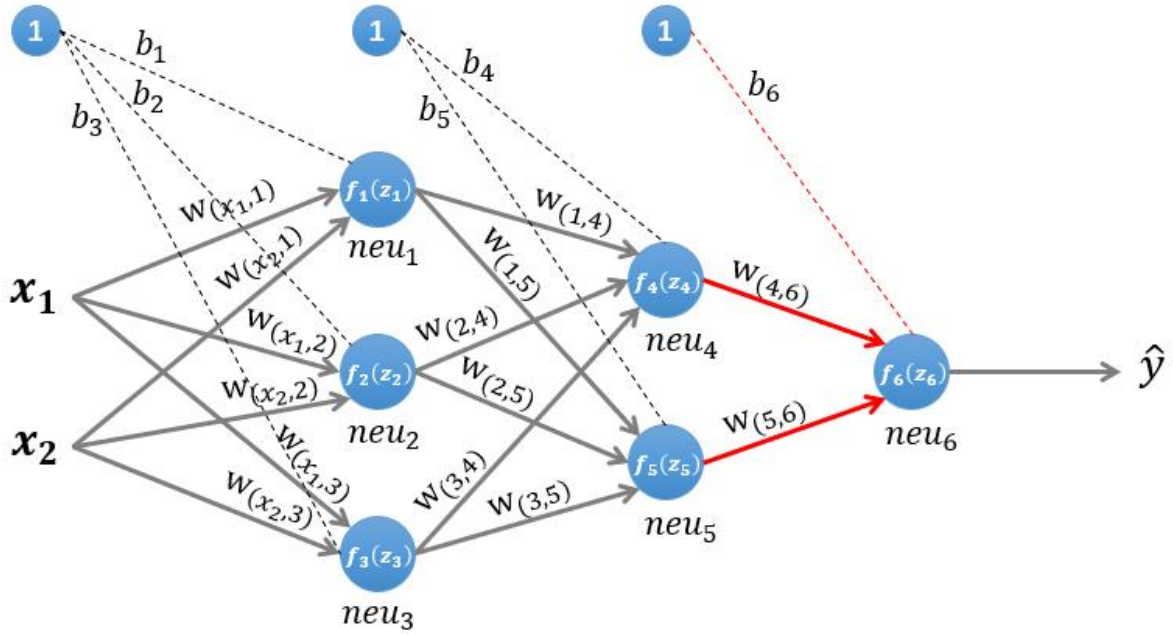
Back Propagation

- The input of the third layer is:

$$z^{(3)} = W^{(3)} * f^{(2)}([z_4, z_5]^T) + (b^{(3)})^T$$

✓ The input of *neu6* is:

$$z_6 = w_{(4,6)} * f_4(z_4) + w_{(5,6)} * f_5(z_5) + b_6$$



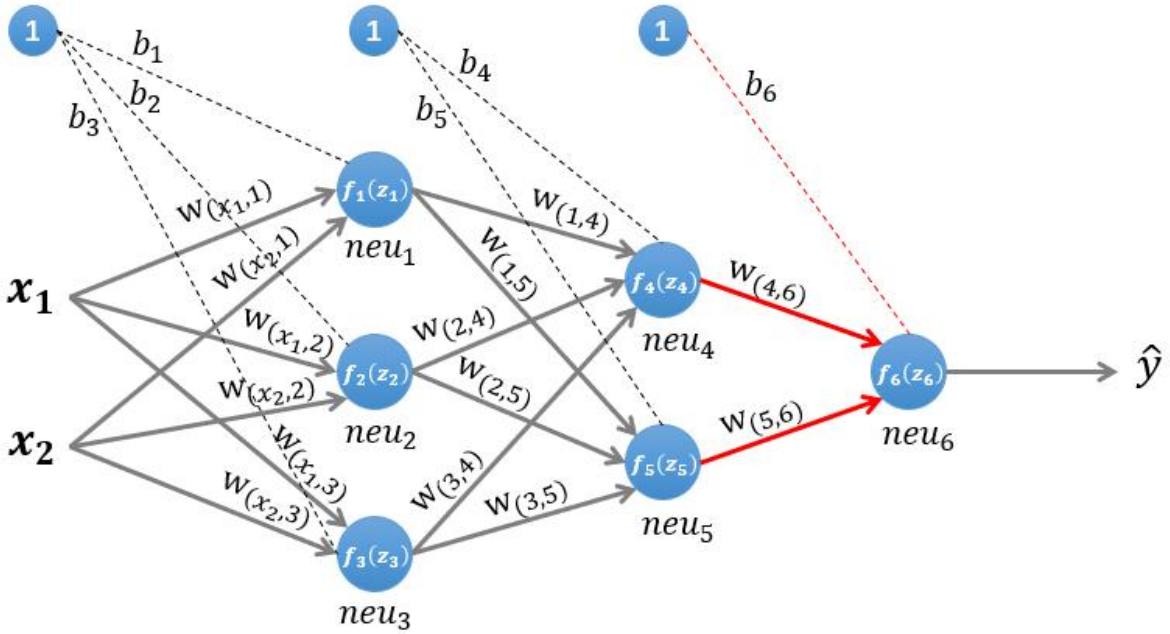
Back Propagation

- The output of the third layer is:

$$f^{(3)}(z^{(3)}) = f^{(3)}(W^{(3)} * f^{(2)}([z_4, z_5]^T) + (b^{(3)})^T)$$

✓ The output of *neu6* is:

$$f_6(z_6) = f_6(w_{(4,6)} * f_4(z_4) + w_{(5,6)} * f_5(z_5) + b_6)$$



Back Propagation

- Assuming we want to compute the partial derivatives of the parameters $W^{(k)}$ and $b^{(k)}$ for the k^{th} hidden layer, that is, finding $\frac{\partial L(y,\hat{y})}{\partial W^{(k)}}$ and $\frac{\partial L(y,\hat{y})}{\partial b^{(k)}}$.
- Suppose $z^{(k)}$ represents the input to the k^{th} layer neurons, i.e. $z^{(k)} = W^{(k)} * n^{(k-1)} + b^{(k)}$, where n^{k-1} is the output of the previous layer neurons. According to the chain:

$$\frac{\partial L(y,\hat{y})}{\partial W^{(k)}} = \frac{\partial L(y,\hat{y})}{\partial z^{(k)}} * \frac{\partial z^{(k)}}{\partial W^{(k)}}, \quad \frac{\partial L(y,\hat{y})}{\partial b^{(k)}} = \frac{\partial L(y,\hat{y})}{\partial z^{(k)}} * \frac{\partial z^{(k)}}{\partial b^{(k)}}$$

- Thus, we only need to compute the partial derivatives $\frac{\partial L(y,\hat{y})}{\partial z^{(k)}}$, $\frac{\partial z^{(k)}}{\partial W^{(k)}}$ and $\frac{\partial z^{(k)}}{\partial b^{(k)}}$.

Back Propagation

- Compute the partial derivative $\frac{\partial z^{(k)}}{\partial b^{(k)}}$:

Since the bias b is a constant term, the computation of the partial derivative is simple:

$$\frac{\partial z^{(k)}}{\partial b^{(k)}} = \begin{bmatrix} \frac{\partial (W_1^{(k)} * n^{k-1} + b_1)}{\partial b_1} & \cdots & \frac{\partial (W_1^{(k)} * n^{k-1} + b_1)}{\partial b_m} \\ \vdots & \cdots & \vdots \\ \frac{\partial (W_m^{(k)} * n^{k-1} + b_m)}{\partial b_1} & \cdots & \frac{\partial (W_m^{(k)} * n^{k-1} + b_m)}{\partial b_m} \end{bmatrix}$$

Still taking the neurons of the first hidden layer as an example, we have:

$$\frac{\partial z^{(1)}}{\partial b^{(1)}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Back Propagation

- Compute the partial derivative $\frac{\partial L(y,\hat{y})}{\partial z^{(k)}}$ (**error term**):

According to the forward calculation in the first section, we know the relationship between the input to the $(k+1)^{\text{th}}$ layer and the output of the k^{th} layer is:

$$z^{(k+1)} = W^{(k+1)} * n^{(k)} + b^{(k+1)}$$

Also, since $n^{(k)} = f_k(z^{(k)})$, according to the chain rule, we can obtain $\delta^{(k)}$ as:

$$\delta^{(k)} = \frac{\partial L(y,\hat{y})}{\partial z^{(k)}} = \frac{n^{(k)}}{\partial z^{(k)}} * \frac{\partial z^{(k+1)}}{\partial n^{(k)}} * \frac{\partial L(y,\hat{y})}{\partial z^{(k+1)}} = \frac{n^{(k)}}{\partial z^{(k)}} * \frac{\partial z^{(k+1)}}{\partial n^{(k)}} * \delta^{(k+1)}$$

$$= f_k'(z^{(k)}) * ((W^{(k+1)})^T * \delta^{(k+1)})$$

Back Propagation

- Compute the partial derivative $\frac{\partial L(y,\hat{y})}{\partial z^{(k)}}$ (**error term**):

$$\delta^{(k)} = \frac{\partial L(y,\hat{y})}{\partial z^{(k)}} = \frac{n^{(k)}}{\partial z^{(k)}} * \frac{\partial z^{(k+1)}}{\partial n^{(k)}} * \frac{\partial L(y,\hat{y})}{\partial z^{(k+1)}} = \frac{n^{(k)}}{\partial z^{(k)}} * \frac{\partial z^{(k+1)}}{\partial n^{(k)}} * \delta^{(k+1)}$$

$$= f_k'(z^{(k)}) * ((W^{(k+1)})^T * \delta^{(k+1)})$$

- Error term $\delta^{(k)}$ of the k^{th} layer neurons is obtained by multiplying the error term of the $(k+1)^{\text{th}}$ layer by the weights of the $(k+1)^{\text{th}}$ layer, and then multiplying by the derivative of the activation function of the k^{th} layer (gradient). This is the backpropagation of the error.)

第k层神经元的误差项$\delta^{(k)}$是由第$k+1$层的误差项乘以第$k+1$层的权重，再乘以第k层激活函数的导数（梯度）得到的。这就是误差的反向传播。

Back Propagation

- Now that we have calculated the partial derivatives $\frac{\partial L(y,\hat{y})}{\partial z^{(k)}}$, $\frac{\partial z^{(k)}}{\partial W^{(k)}}$ and $\frac{\partial z^{(k)}}{\partial b^{(k)}}$, then $\frac{\partial L(y,\hat{y})}{\partial W^{(k)}}$ and $\frac{\partial L(y,\hat{y})}{\partial b^{(k)}}$ can be represented as:

$$\frac{\partial L(y,\hat{y})}{\partial W^{(k)}} = \frac{\partial L(y,\hat{y})}{\partial z^{(k)}} * \frac{\partial z^{(k)}}{\partial W^{(k)}} = \delta^{(k)} * (n^{(k-1)})^T$$

$$\frac{\partial L(y,\hat{y})}{\partial b^{(k)}} = \frac{\partial L(y,\hat{y})}{\partial z^{(k)}} * \frac{\partial z^{(k)}}{\partial b^{(k)}} = \delta^{(k)}$$

Back Propagation: An Example

- Input sample: $\vec{a} = (x_1, x_2) = (1,2)$
- The parameters of the 1st layer:

$$W^{(1)} = \begin{bmatrix} w_{(x_1,1)} & w_{(x_2,1)} \\ w_{(x_1,2)} & w_{(x_2,2)} \\ w_{(x_1,3)} & w_{(x_2,3)} \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \\ 3 & 2 \end{bmatrix}, \; b^{(1)} = [b_1, b_2, b_3]^T = [1,2,3]^T$$

- The parameters of the 2nd layer:

$$W^{(2)} = \begin{bmatrix} w_{(1,4)} & w_{(2,4)} & w_{(3,4)} \\ w_{(1,5)} & w_{(2,5)} & w_{(3,5)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 2 \end{bmatrix}, \; b^{(2)} = [b_4, b_5]^T = [2,1]^T$$

- The parameters of the 3rd layer:

$$W^{(3)} = [w_{(4,6)}, w_{(5,6)}] = [1,3], \; b^{(3)} = [b_6] = [2]$$

- Assuming all activation functions are Logistic functions:

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

$$f^{(k)'}(x) = f^{(k)}(x)(1 - f^{(k)}(x))$$

- Using the mean squared error as the loss function:

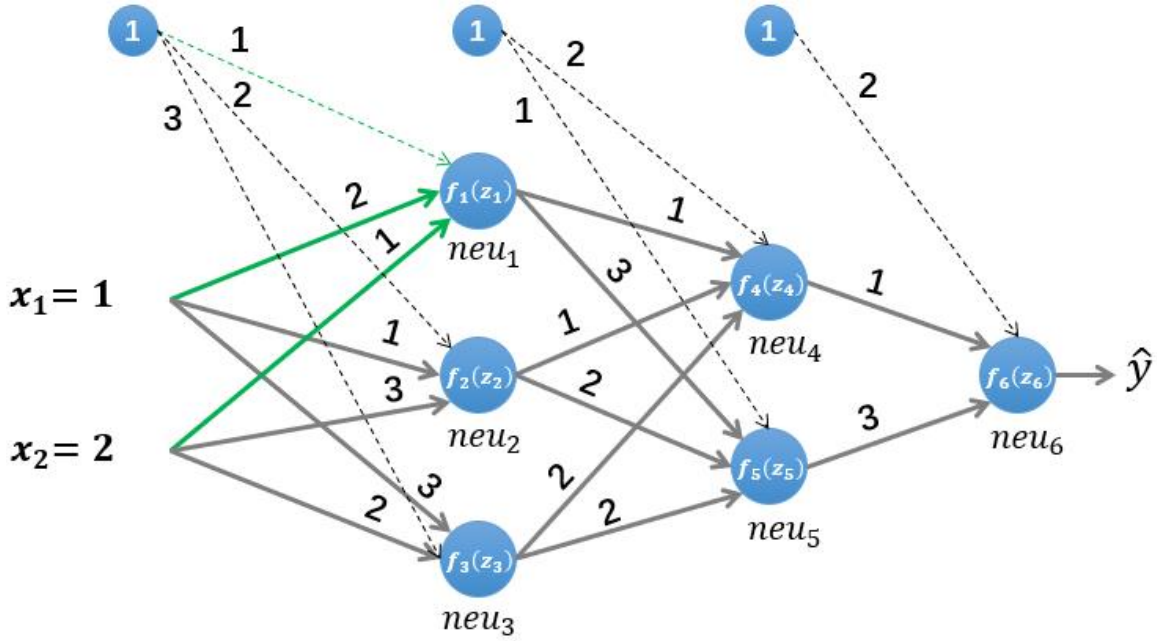
$$L(y, \hat{y}) = \frac{1}{2} \sum (y - \hat{y})^2$$

Back Propagation: An Example

- First, initialize the parameters of the neural network and calculate the first layer of neurons:

$$z_1 = w_{(x_1,1)} * x_1 + w_{(x_2,1)} * x_2 + b_1$$
$$= 2 * 1 + 1 * 2 + 1 = 5$$

$$f_1(z_1) = \frac{1}{1 + e^{-z_1}} = 0.993307149075715$$



Back Propagation: An Example

- Similarly:

$$z_2 = w_{(x_1,2)} * x_1 + w_{(x_2,2)} * x_2 + b_2$$

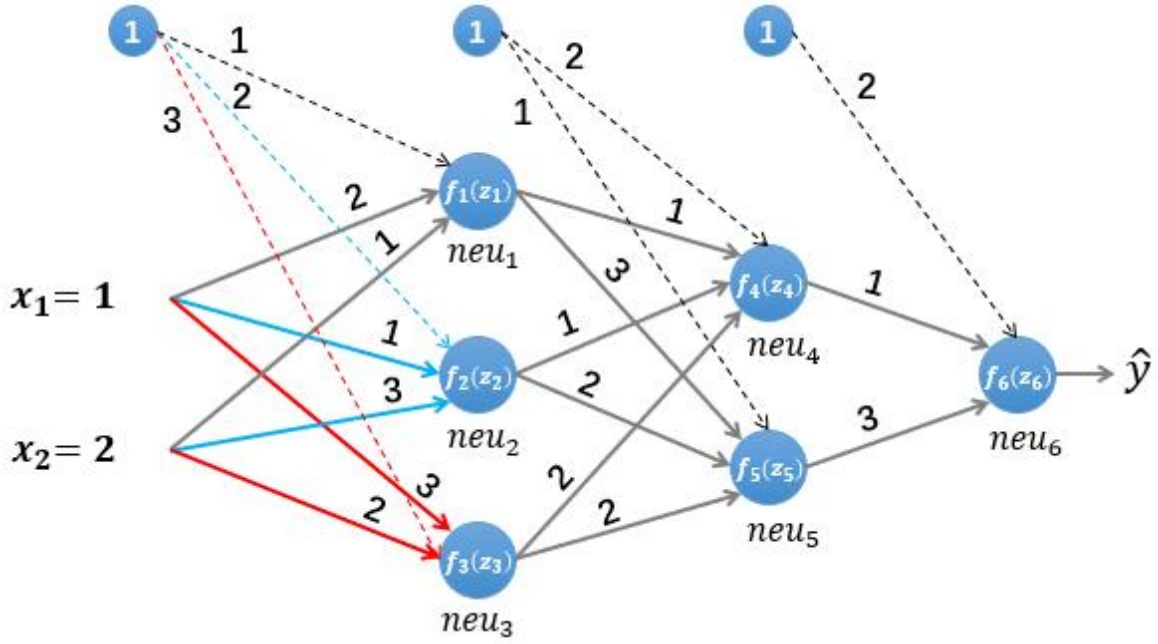
$$= 1 * 1 + 3 * 2 + 2 = 9$$

$$f_2(z_2) = \frac{1}{1 + e^{-z_2}} = 0.999876605424014$$

$$z_3 = w_{(x_1,3)} * x_1 + w_{(x_2,3)} * x_2 + b_3$$

$$= 3 * 1 + 2 * 2 + 3 = 10$$

$$f_3(z_3) = \frac{1}{1 + e^{-z_3}} = 0.999954602131298$$

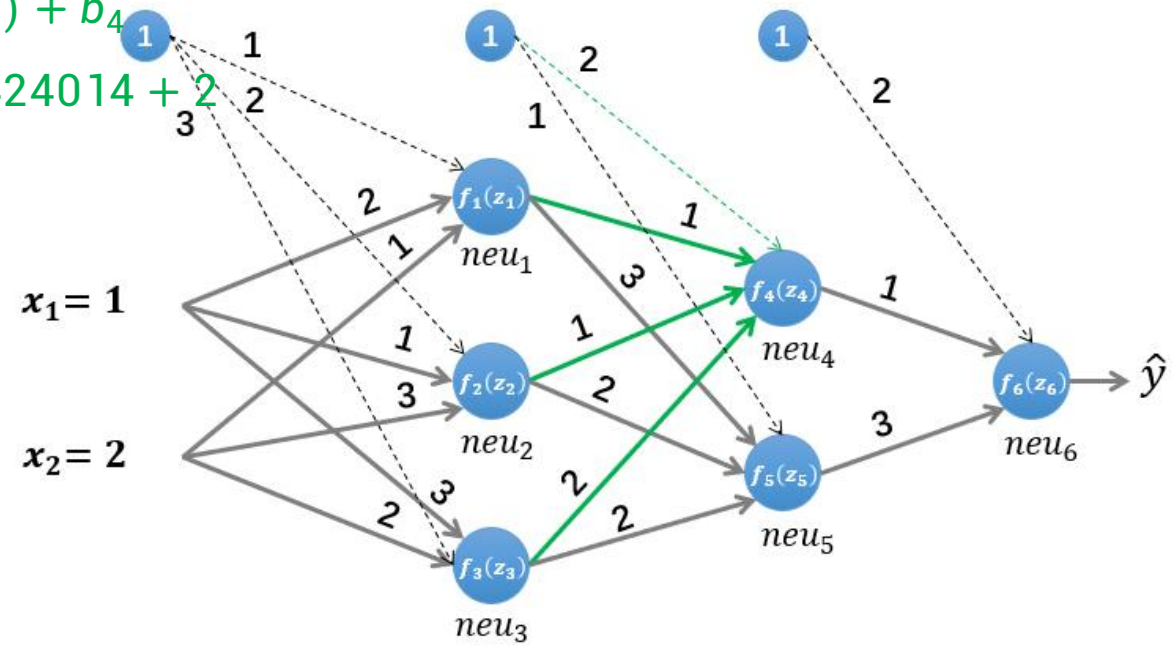


Back Propagation: An Example

- Calculate the second layer of neurons:

$$z_4 = w_{(1,4)} * f_1(z_1) + w_{(2,4)} * f_2(z_2) + w_{(3,4)} * f_3(z_3) + b_4$$
$$= 1 * 0.993307149075715 + 1 * 0.999876605424014 + 2 * 0.999954602131298 + 2$$
$$= 5.993092958762325$$

$$f_4(z_4) = \frac{1}{1 + e^{-z_4}} = 0.997510281884102$$

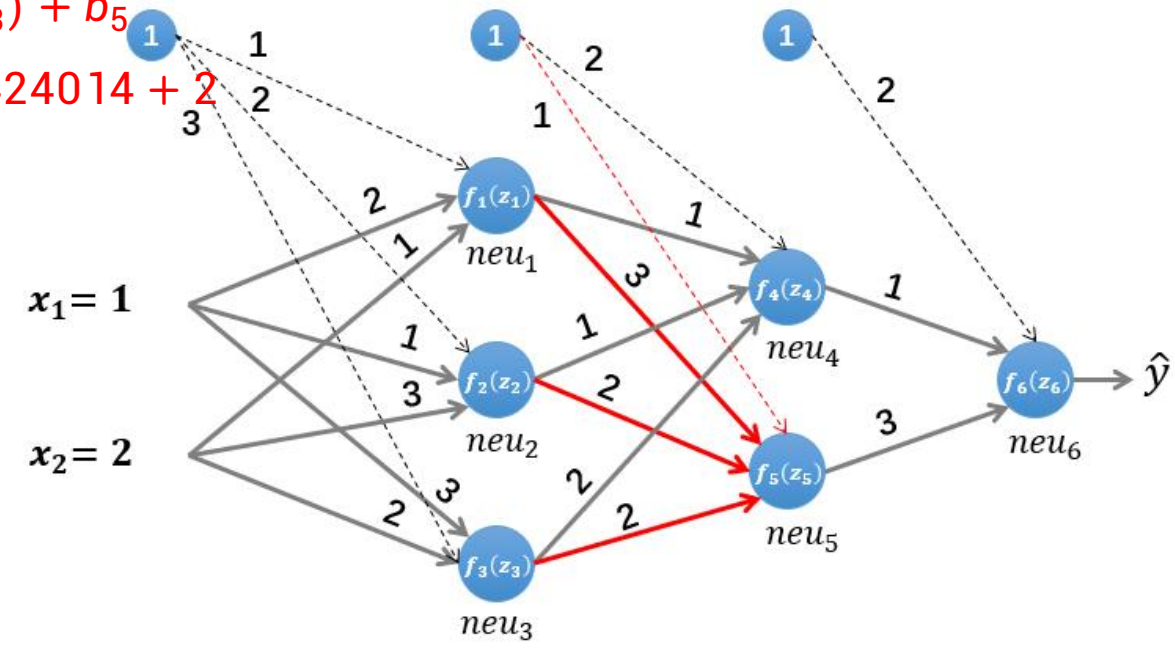


Back Propagation: An Example

- Calculate the second layer of neurons:

$$z_5 = w_{(1,5)} * f_4(z_4) + w_{(2,5)} * f_2(z_2) + w_{(3,5)} * f_3(z_3) + b_5$$
$$= 3 * 0.993307149075715 + 2 * 0.999876605424014 + 2$$
$$* 0.999954602131298 + 1$$
$$= 7.979583862337769$$

$$f_5(z_5) = \frac{1}{1 + e^{-z_5}} = 0.9996577353143582$$

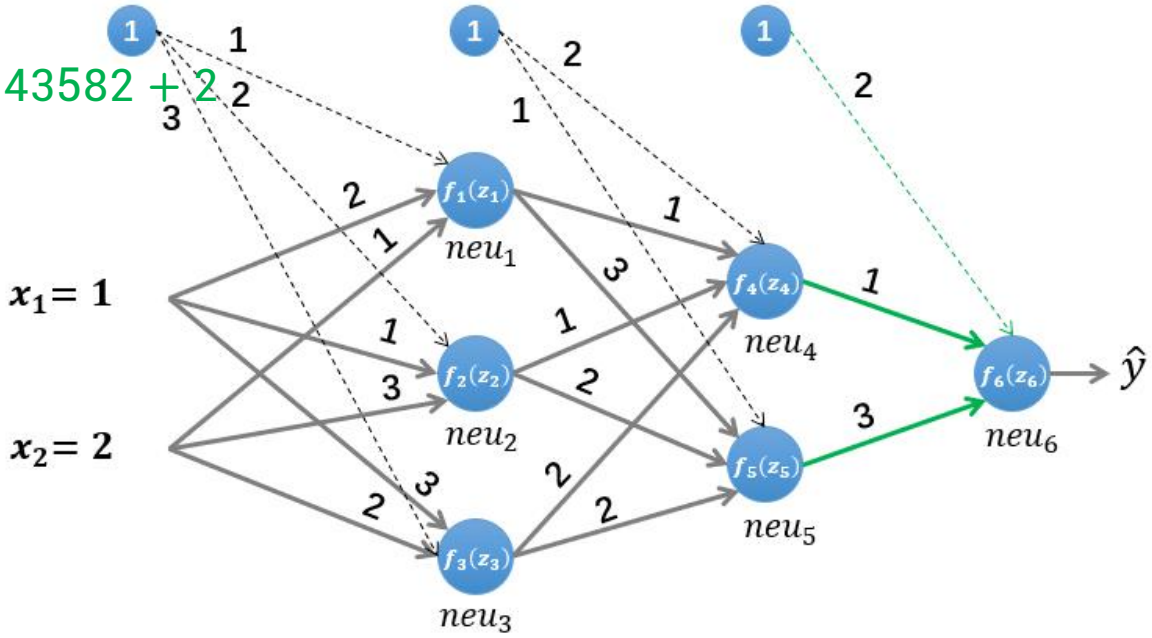


Back Propagation: An Example

- Calculate the second layer of neurons:

$$z_6 = w_{(4,6)} * f_4(z_4) + w_{(5,6)} * f_5(z_5) + b_6$$
$$= 1 * 0.9975102818841 02 + 3 * 0.9996577353143582 + 2$$
$$= 3.996483487827 1764$$

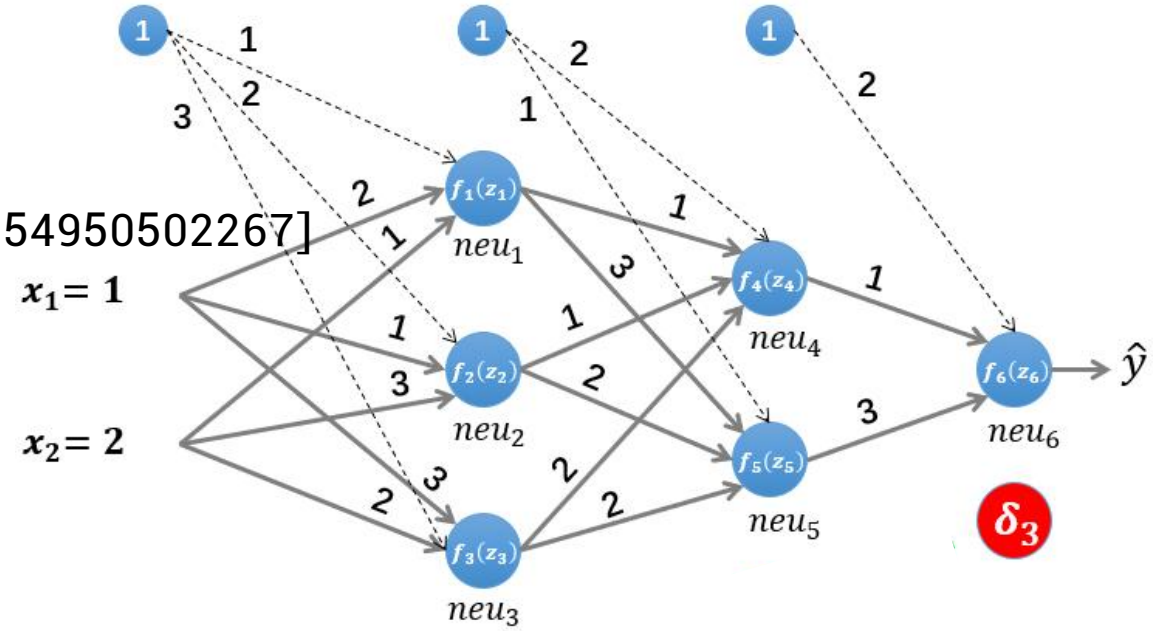
$$f_6(z_6) = \frac{1}{1 + e^{-z_6}} = 0.9975186881409559$$



Back Propagation: An Example

- The error function is $L(y,\hat{y}) = \frac{1}{2}\sum (y-\hat{y})^2$. Since the label of this sample is "1".

$$\delta^{(3)} = \frac{\partial L(y,\hat{y})}{\partial z^{(3)}} = \frac{\partial L(y,\hat{y})}{\partial n^{(3)}} * \frac{n^{(3)}}{\partial z^{(3)}}$$
$$= [-0.0024813118590440997\,] * f^{(3)'}(z^{(3)})$$
$$= [-0.0024813118590440997] * [0.0024751549505022 67]$$
$$= [-0.00000614163133165298]$$



注：$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x) * (1-f(x))$

Back Propagation: An Example

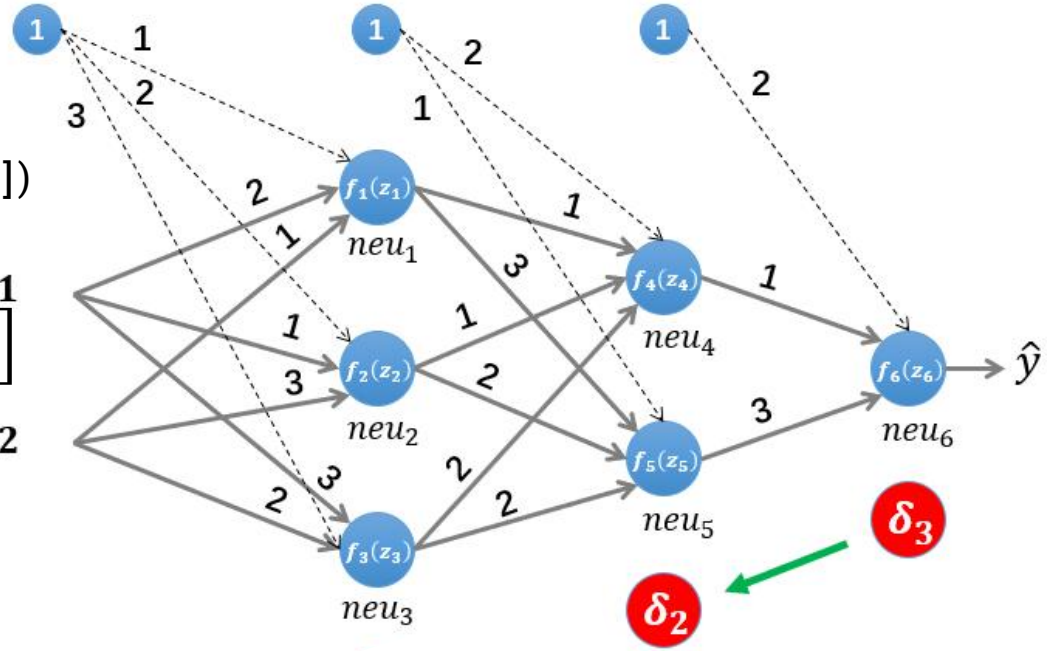
- The error function is $L(y,\hat{y}) = \frac{1}{2}\sum (y-\hat{y})^2$. Since the label of this sample is "1".

$$\delta^{(2)} = \frac{\partial L(y,\hat{y})}{\partial z^{(2)}} = f^{(2)'}(z^{(2)})((W^{(3)})^T * \delta^{(3)})$$

$$= \begin{bmatrix} f'_4(z_4) & 0 \\ 0 & f'_5(z_5) \end{bmatrix} * \left(\begin{bmatrix} 1 \\ 3 \end{bmatrix} * [-0.0000061416313316529 8]\right)$$

$$= \begin{bmatrix} 0.0024835194196014 12 & 0 \\ 0 & 0.0003421475405267646 \end{bmatrix} * \begin{bmatrix} -0.0000061416313316529 8 \\ -0.0001842489399495894 \end{bmatrix}$$

$$= \begin{bmatrix} -0.0000000152528606801 9266 \\ -0.0000000063040321648 41557 \end{bmatrix}$$



Back Propagation: An Example

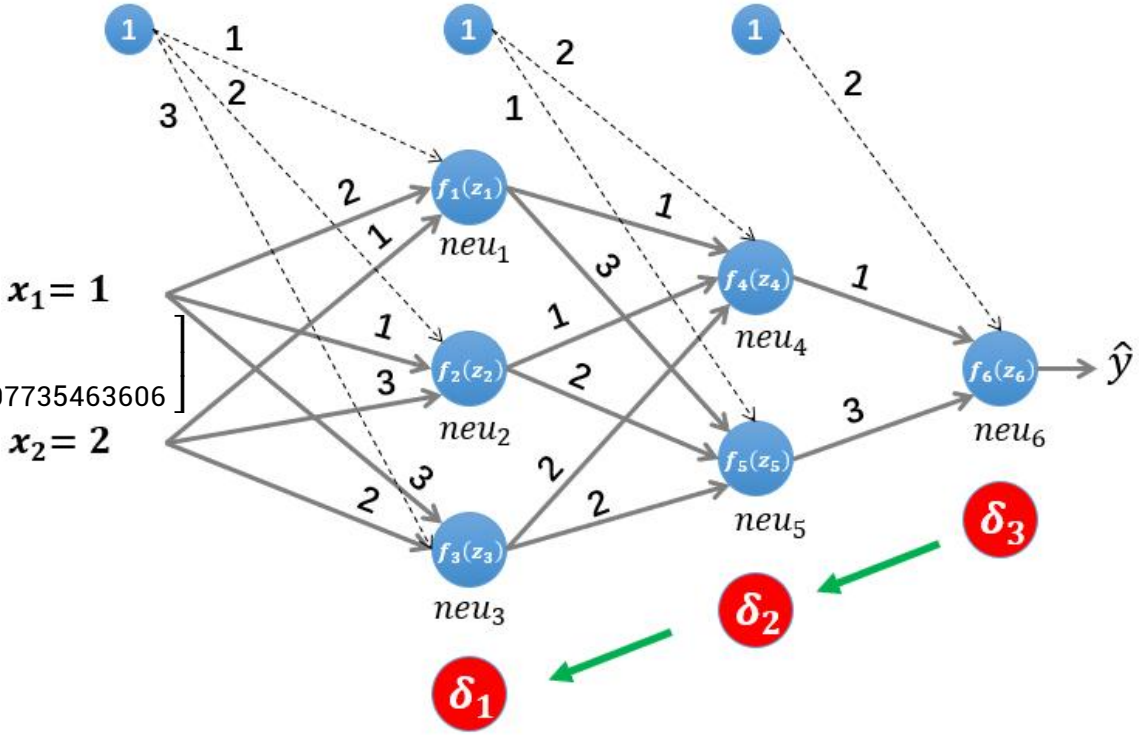
- The error function is $L(y,\hat{y}) = \frac{1}{2}\sum (y-\hat{y})^2$. Since the label of this sample is "1".

$$\delta^{(1)} = \frac{\partial L(y,\hat{y})}{\partial z^{(1)}} = f^{(1)'}(z^{(1)})((W^{(2)})^T * \delta^{(2)})$$

$$= \begin{bmatrix} f'_1(z_1) & 0 & 0 \\ 0 & f'_2(z_2) & 0 \\ 0 & 0 & f'_3(z_3) \end{bmatrix} ((W^{(2)})^T * \delta^{(2)})$$

$$= \begin{bmatrix} 0.006648056670790253 & 0 & 0 \\ 0 & 0.0001233793497645972 6 & 0 \\ 0 & 0 & 0.00004539580773546360 6 \end{bmatrix} * (\begin{bmatrix} 1 & 1 & 2 \\ 3 & 2 & 2 \end{bmatrix})^T * \begin{bmatrix} -0.000000015252860680 19266 \\ -0.000000006304032164841557 \end{bmatrix}$$

$$= \begin{bmatrix} -0.000000000227130571452642 87 \\ -0.000000000003437462811558678 3 \\ -0.000000000001957185125934328 4 \end{bmatrix}$$



Back Propagation: An Example

- Update parameters, taking the first layer as an example ($a = 0.1$):

$$W^{(k)} = W^{(k)} - a\delta^{(k)}(n^{(k-1)})^T$$

$$W^{(1)} = W^{(1)} - 0.1 * \delta^{(1)}(n^{(0)})^T$$

$$= \begin{bmatrix} 2 & 1 \\ 1 & 3 \\ 3 & 2 \end{bmatrix} - 0.1 * \begin{bmatrix} -0.000000000227130571452642 87 \\ -0.000000000003437462811558 6783 \\ -0.000000000001957185125934 3284 \end{bmatrix} [x_1, x_2]$$

$$= \begin{bmatrix} 2 & 1 \\ 1 & 3 \\ 3 & 2 \end{bmatrix} - 0.1 * \begin{bmatrix} -0.000000000227130571452642 87 \\ -0.000000000003437462811558 6783 \\ -0.000000000001957185125934 3284 \end{bmatrix} [1,2]$$

$$= \begin{bmatrix} 2.000000000022713 & 1.000000000045426 \\ 1.0000000000003437 & 3.0000000000006875 \\ 3.000000000000196 & 2.0000000000003912 \end{bmatrix}$$

Back Propagation: An Example

- Update parameters, taking the first layer as an example ($a = 0.1$):

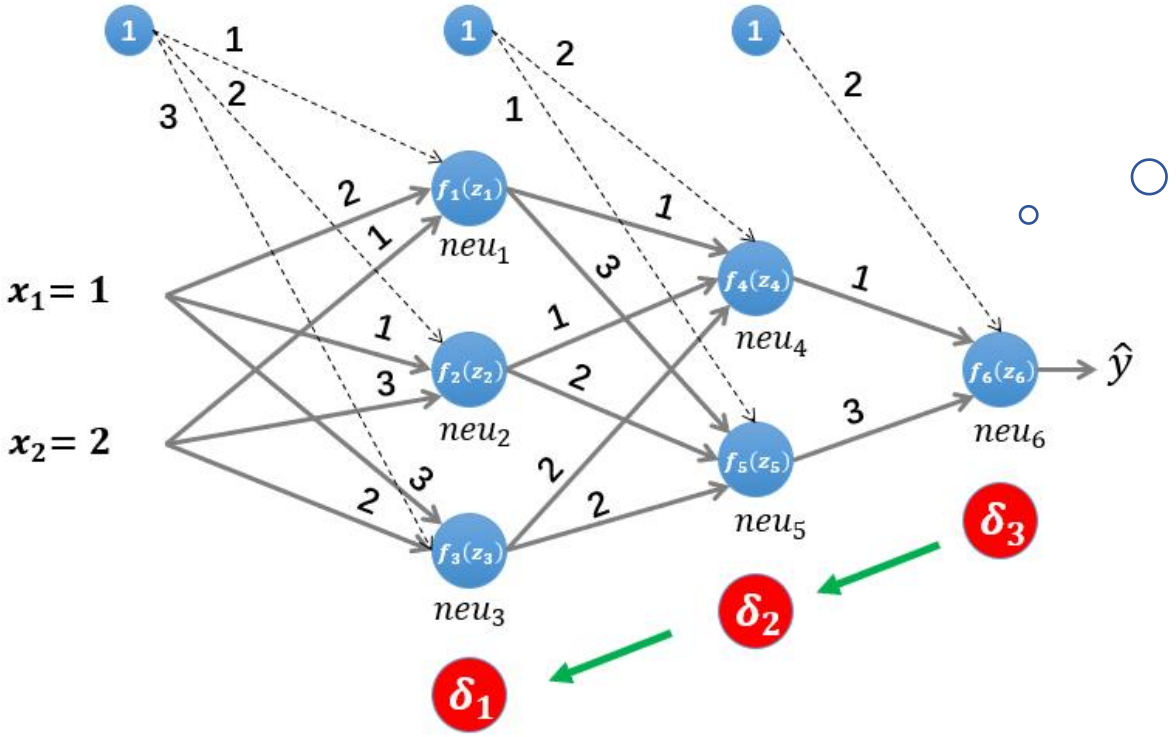
$$b^{(k)} = b^{(k)} - a\delta^{(k)}$$

$$b^{(1)} = b^{(1)} - 0.1 * \delta^{(1)}$$

$$= [1,2,3]^T - 0.1 * \begin{bmatrix} -0.00000000022713057145264287 \\ -0.00000000000343746281155867 83 \\ -0.00000000000019571851259343284 \end{bmatrix}$$

$$= \begin{bmatrix} 1.00000000000227130571452642 87 \\ 2.00000000000034374628115586783 \\ 3.00000000000019571851259343284 \end{bmatrix}$$

Back Propagation



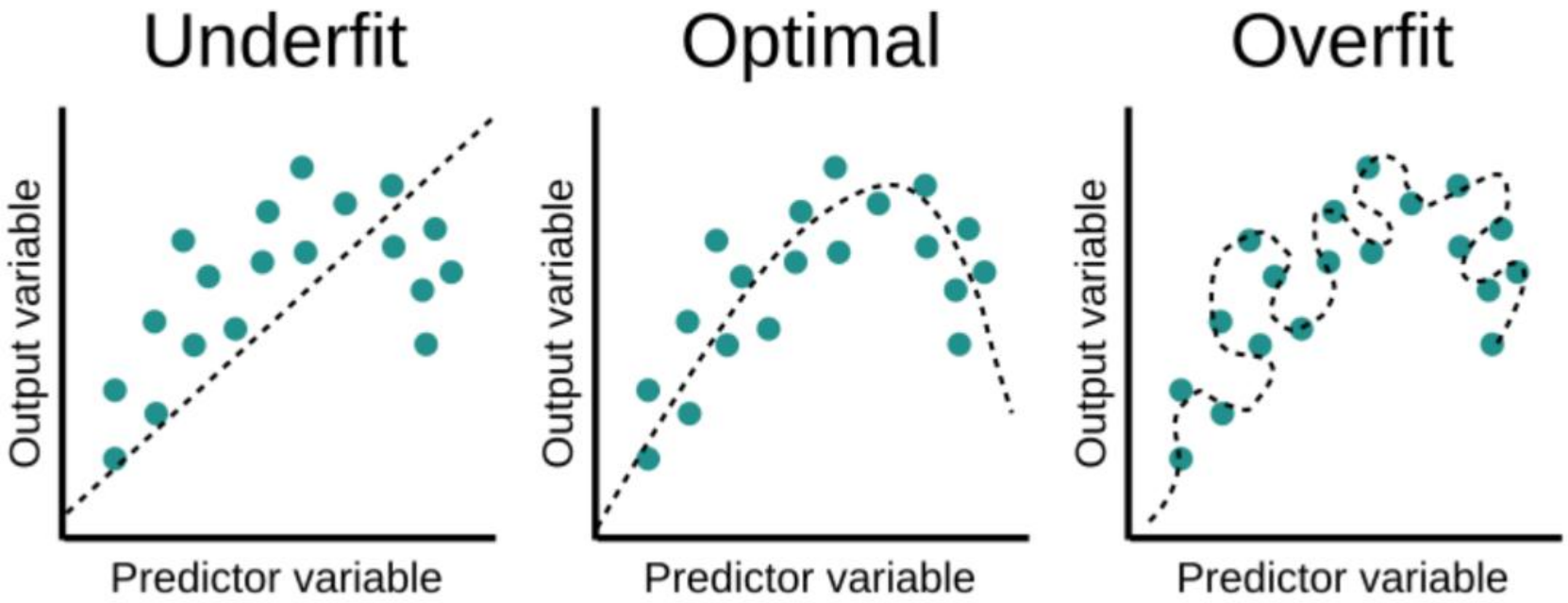
Feel Too complex to compute manually? 😭

You may need deep learning frameworks like PyTorch 😆
(code here)

Note:

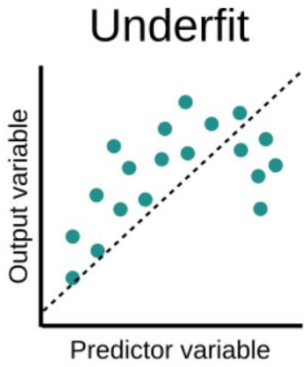
- ✓ nn.MSELoss() in PyTorch does not have the 1/2 part (i.e. $L(y, \hat{y}) = \sum (y - \hat{y})^2$), so the gradient is twice the size of the previous example.
- ✓ The default label of this sample is “5”.

Overfitting & Underfitting



Overfitting & Underfitting

- **Underfitting**: poor performance on both training and test set.



• Model is too simple
e.g., linear model for nonlinear data

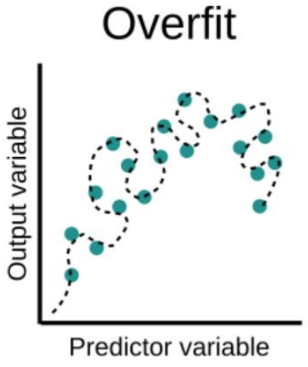
• Inadequate training
e.g., too few iterations

Solution

• Complicate the structure

• Increase training iterations

- **Overfitting**: poor performance on test set while excellent performance on training set



• Model is too complex
e.g., Using high-order models to fit low-order functions

• Training data is insufficient

Solution

• Simplify the structure

• Regularization

Regularization

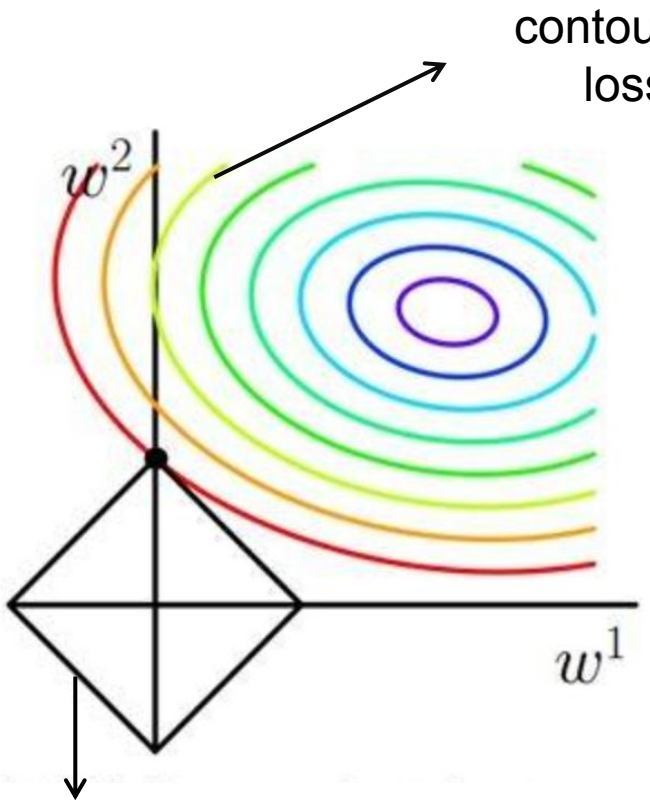
- **L1 Regularization**

Objective function $min\, J(w)$

$$J = L(w)$$

L1 Regularization

$$J = L(w) + \lambda \|w\|_1$$



https://www.zhihu.com/question/38081976/answer/2330394725

Regularization

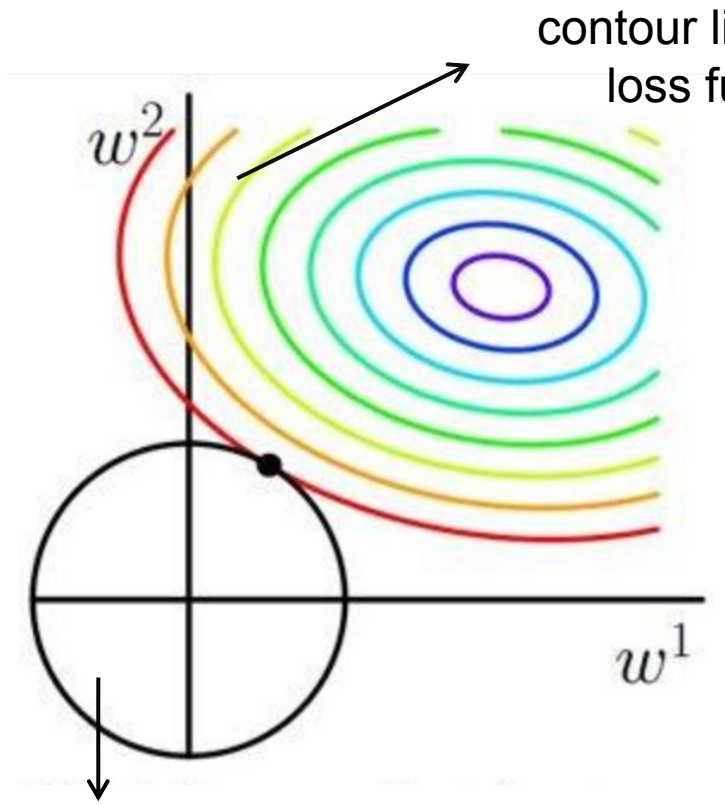
- **L2 Regularization**

Objective function $min\, J(w)$

$$J = L(w)$$

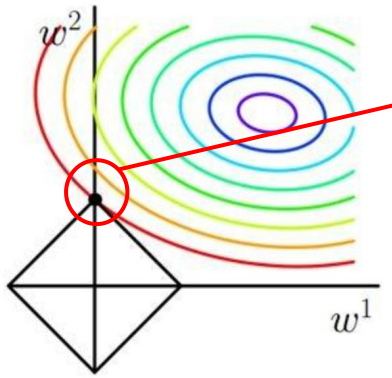
L2 Regularization

$$J = L(w) + \frac{\lambda}{2}\|w\|_2^2$$



Regularization

L1 Regularization



- Sparsity

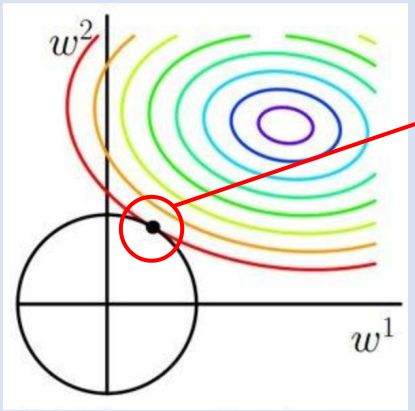
Intersects the loss contour at "corners," forcing some weights to zero.

↓

Tends to shrink some weights to zero

- Not differentiable at zero

L2 Regularization



- Density

Hypersphere has no conner, so it likely to touch contours away from the axes.

↓

Shrinks weights uniformly but very unlikely to zero

- Smooth & differentiable

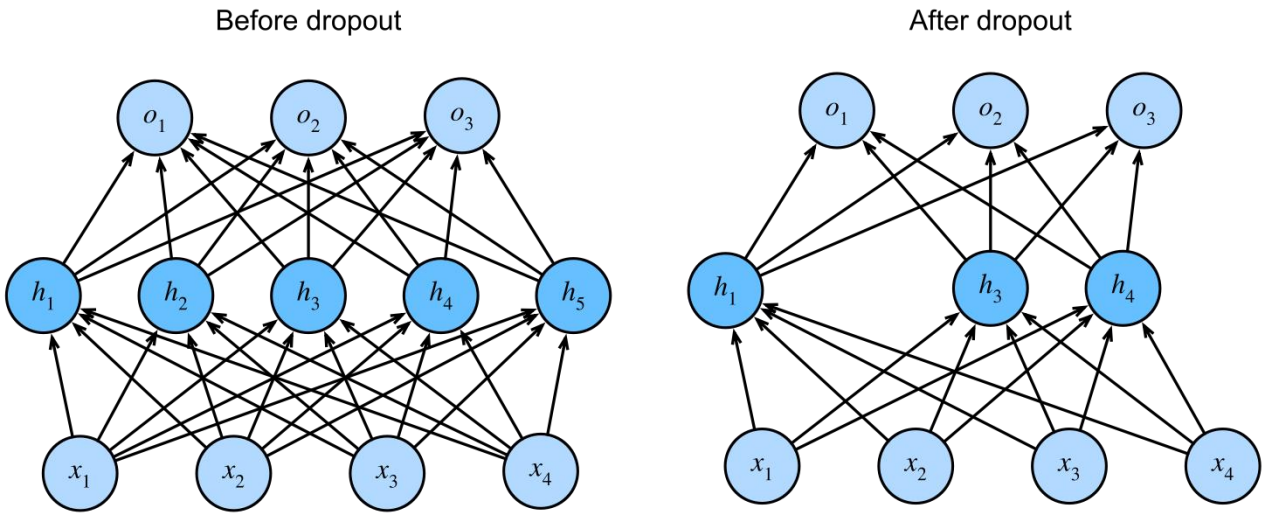
Regularization

- Dropout

At each training epoch, neurons have a **certain probability** of being dropped out.

Any feature has a certain probability of being masked out

The network cannot rely too heavily on any single feature, forcing it to learn robust, distributed representations.

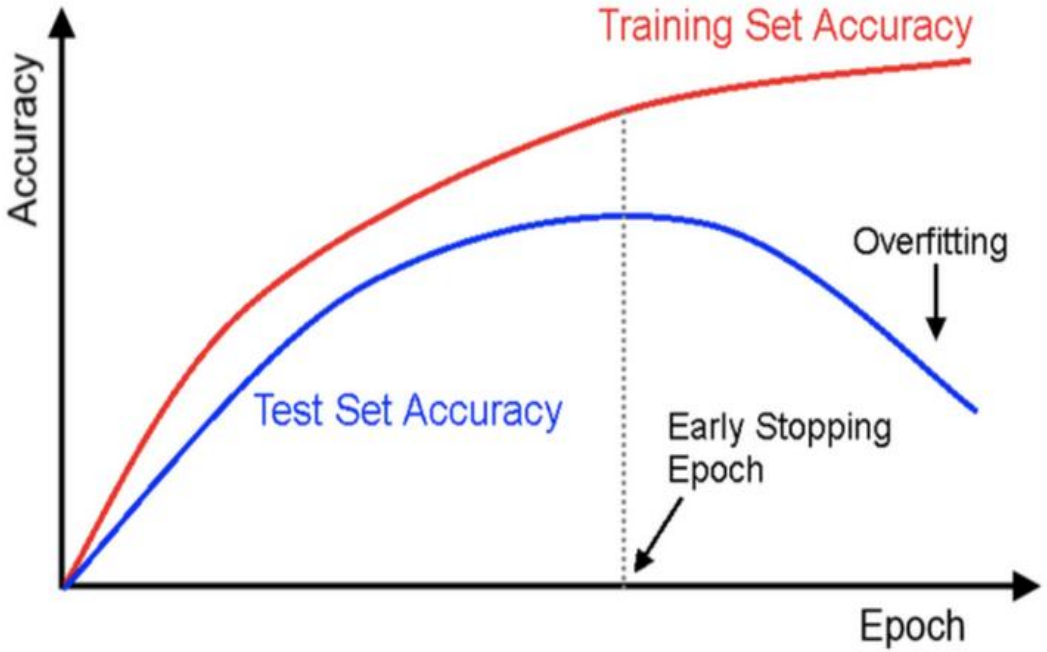


Regularization

- **Early Stopping**

✓ **Training accuracy:** may increase monotonically

✓ **Test accuracy**: might initially increase but start decreasing after reaching a certain threshold.



Regularization

- **Data Augmentation**

Insufficient data cause overfitting → Data augmentation to increase data diversity

original image

affine, shear, rotate, flip, blur, color transform ...

Optimizer: BGD, SGD, Mini-Batch GD

$$\theta = \theta - \eta \cdot \Delta_\theta J(\theta; Data)$$

The data used to compute the gradient at each parameter update

Batch Gradient Descent (BGD)

$$\theta = \theta - \eta \cdot \Delta_\theta J(\theta; X, Y)$$

Stochastic Gradient Descent (SGD)

$$\theta = \theta - \eta \cdot \Delta_\theta J(\theta; x^{(i)}, y^{(i)})$$

Mini-batch Gradient Descent (Mini-batch GD)

$$\theta = \theta - \eta \cdot \Delta_\theta J(\theta; x^{(i:i+n)}, y^{(i:i+n)})$$

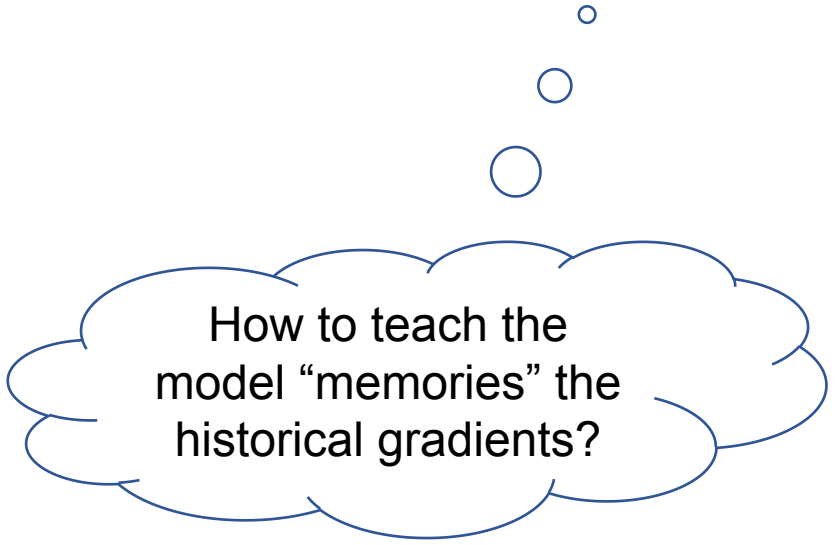
Optimizer: BGD, SGD, Mini-Batch GD

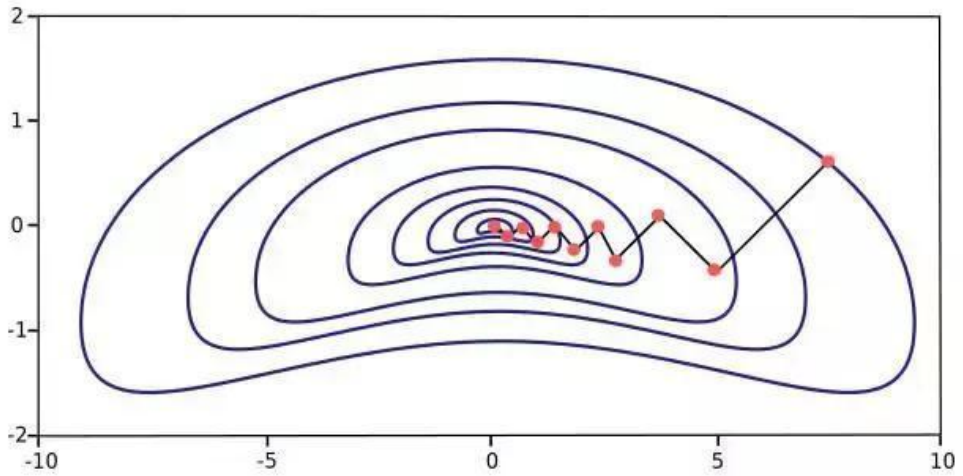
Method	BGD	SGD	Mini-batch GD
Data Used per Update	Entire training dataset	Single random sample	Small batch (e.g., 32, 64)
Characteristics	• Stable convergence • High computational cost per update • Slow and memory-intensive for large datasets	• Fast per iteration • Unstable convergence • High variance due to the noise	• Balances BGD and SGD • Commonly used

Optimizer

- The training data used in each iteration is typically a small batch.

- The update direction to exhibit jagged or even random oscillations





Preliminary: Exponentially Weighted Average (EMA)

- Given a sequence $\{\theta_1, \theta_2, \theta_3, \ldots, \theta_t\}$, calculate the average:

$$v_t = \frac{\theta_1, \theta_2, + \ldots + \theta_{t-1}}{t} + \frac{\theta_t}{t}$$

older observations

current observations

- Key formula of EMA:

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

Decay rate, higher decay rate means smoother but slower adaptation

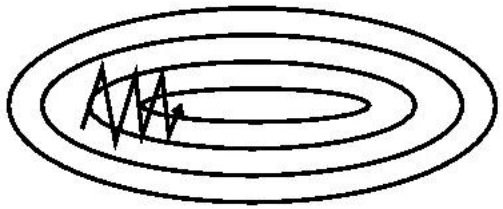
Optimizer: Momentum

- Momentum enhances gradient descent by **accumulating historical gradients to accelerate convergence and dampen oscillations**.

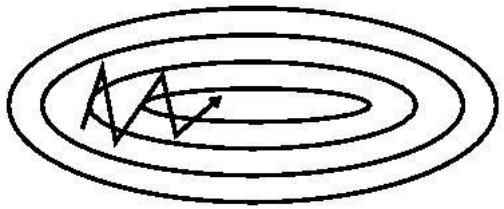
historical gradients

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_{t-1})$$ current gradients

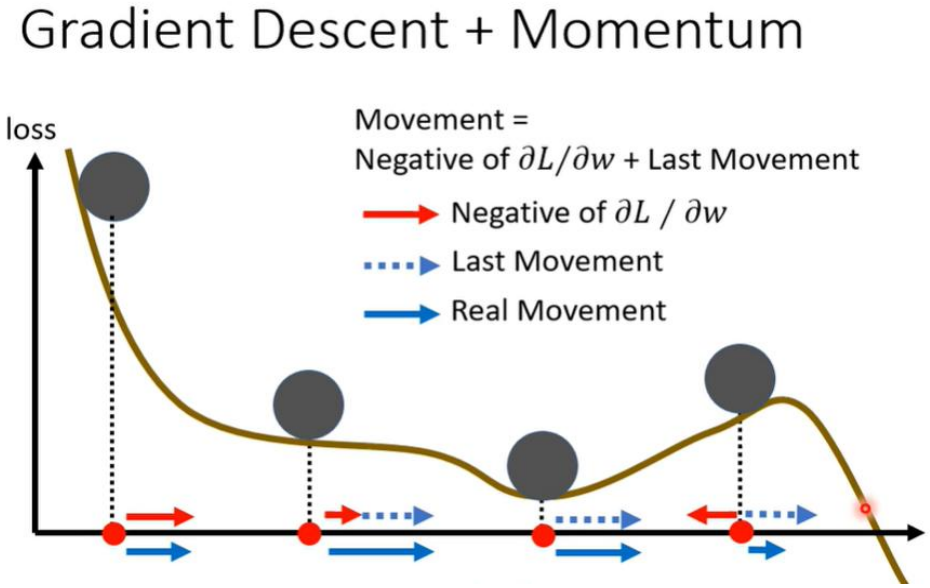
$$\theta_t = \theta_{t-1} - v_t$$



SGD



SGD with Momentum



https://www.cnblogs.com/fromll77/p/14956375.html

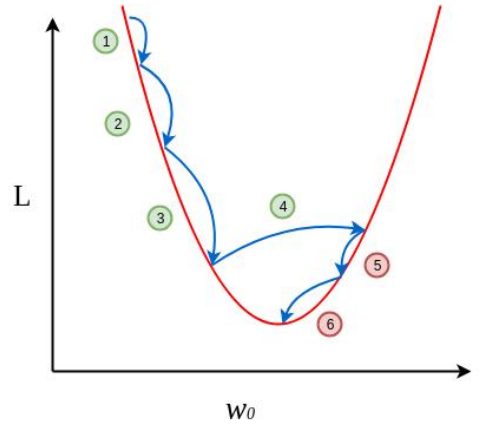
Optimizer: Nesterov Acceleration

- Momentum is not smart enough to **predict possible future scenarios**.

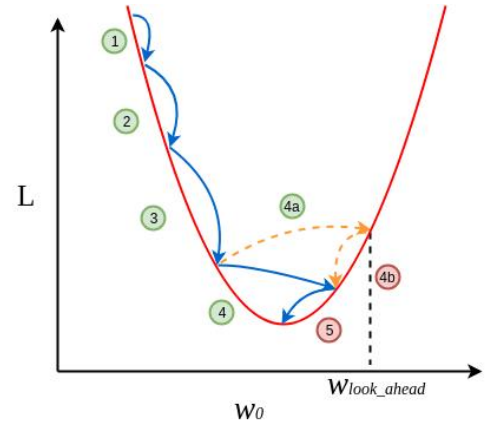
$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta_{t-1} - \gamma v_{t-1})$$

$$\theta_t = \theta_{t-1} - v_t$$

Approximately predict future weights



(a) Momentum-Based Gradient Descent



(b) Nesterov Accelerated Gradient Descent

$\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Positive(+)}$ $\bigcirc \Longrightarrow \frac{\partial L}{\partial w_0} = \frac{Negative(-)}{Negative(-)}$

Optimizer: Adagrad

- Adagrad is an **adaptive gradient optimizer that uses different learning rates for different parameters based on the update frequency**. For each parameter:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \qquad g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i})$$

- $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix:

$$G_{t,ii} = G_{t-1,ii} + g_{t,i}^2 = \sum_{k=0}^{t} g_{k,i}^2$$

- The overall formula is:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_{t-1} + \epsilon}} \odot g_{t-1}$$

Optimizer: Adadelta & RMSprop

- Adadelta is an extended algorithm of Adagrad to handle the problem of **monotonically decreasing Adagrad learning rate**.
- Accumulation of squared gradients: $E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$
- The step size of each weight update is as follows:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t = -\frac{\eta}{RMS[g]_t} g_t$$

- The exponentially weighted average is also used to calculate the root mean square of the step length:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

Optimizer: Adadelta & RMSprop

- The final Adadelta weight update formula is:

$$\theta_t = \theta_{t-1} - \frac{\text{RMS}[\Delta\theta]_{t-2}}{\text{RMS}[g]_{t-1}} g_{t-1}$$

- **Adadelta does not even require a specified learning rate**. The idea of RMSprop is roughly the same, and its update formula is as follows:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\text{RMS}[g]_{t-1}} g_{t-1}$$

Optimizer: Adam

- Adaptive Moment Estimation is a bit like a **combination of Momentum and RMSProp**.

  First, an exponentially weighted average is applied to the **gradient** and **the square of the gradient**:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

- Then perform bias correction:

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t} \qquad \widehat{v}_t = \frac{v_t}{1-\beta_2^t}$$

- The final weight update formula is as follows:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\widehat{v}_{t-1} + \epsilon}} \widehat{m}_{t-1}$$

Optimizer: AdamW

- AdamW believes that Adam is not used correctly for **weight decay**. When we use L2 regularization, a square term of the weight is added after the loss:

$$loss = loss_{orig}(\theta) + \frac{\lambda}{2}\theta^2$$

- When differentiating the loss function, the weight decay term appears in the gradient:

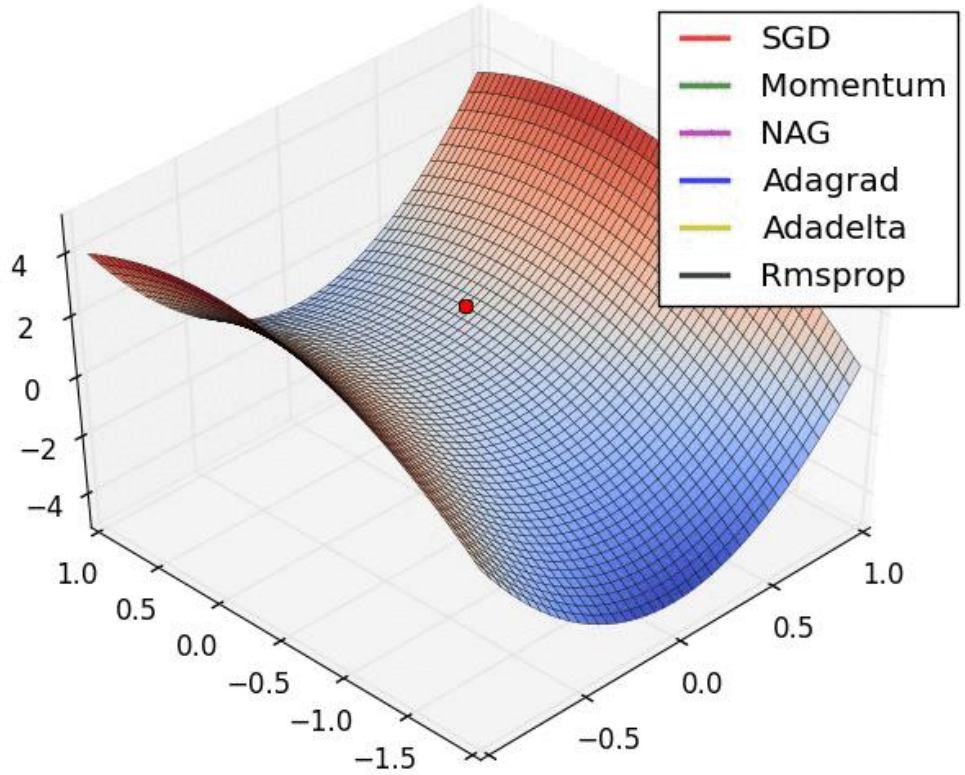
$$\nabla loss = \nabla loss_{orig}(\theta) + \lambda\theta$$

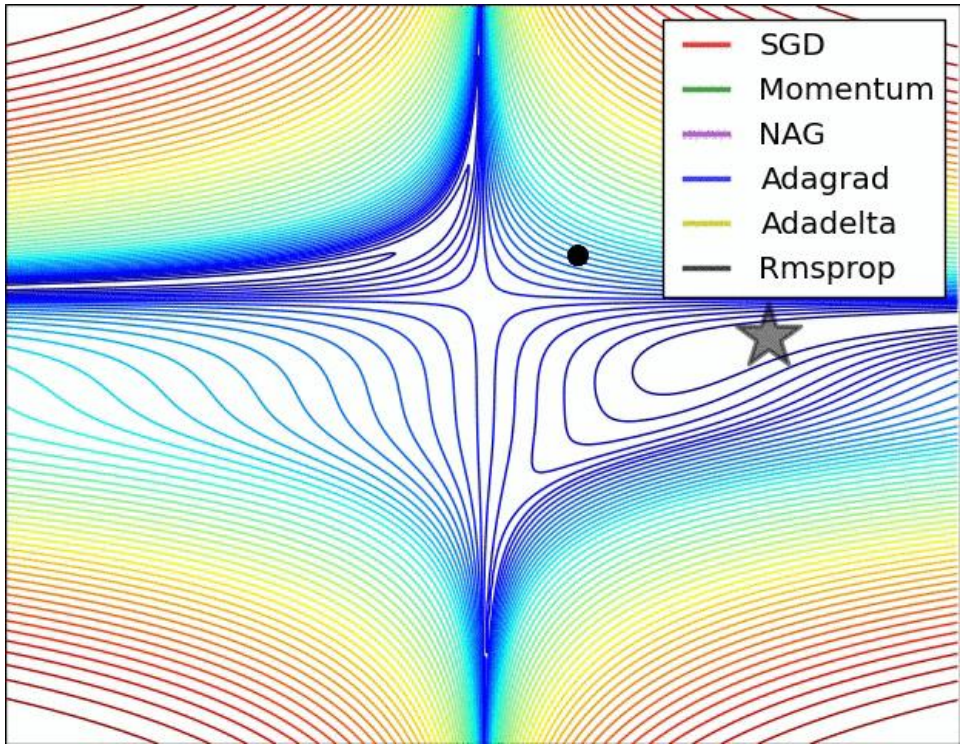
- Therefore, the parameter update formula of the Adam optimizer is

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_{t-1} + \epsilon}}\hat{m}_{t-1} - \eta\lambda\theta_{t-1}$$

Optimizer

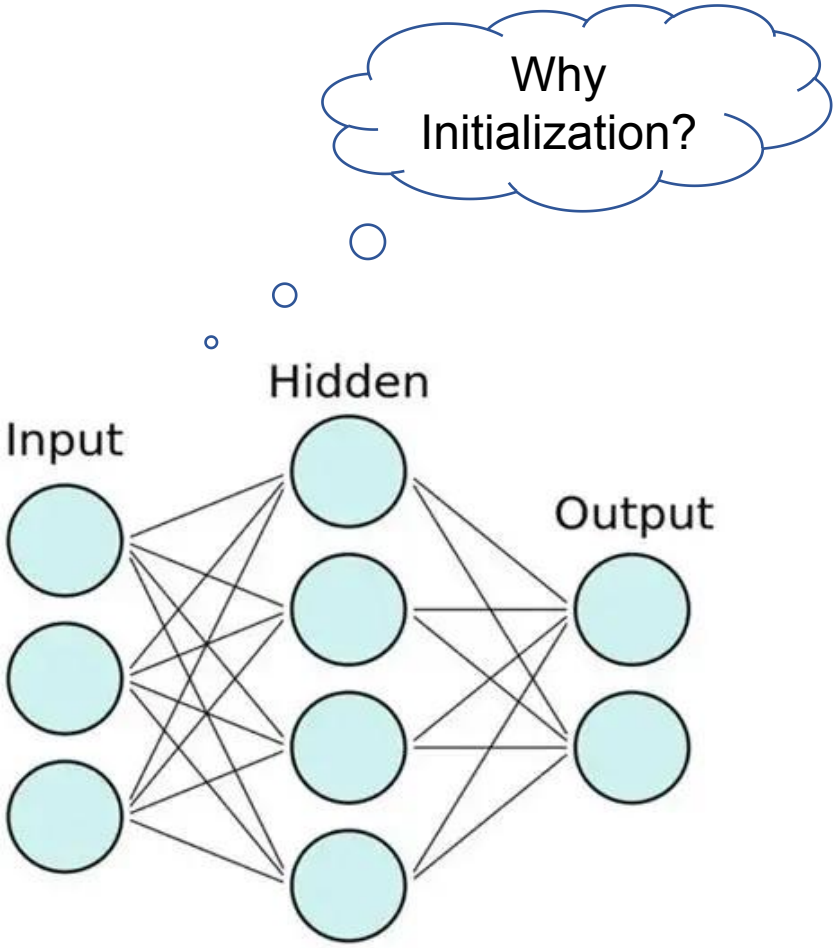
- Visualization diagrams of several optimizers:





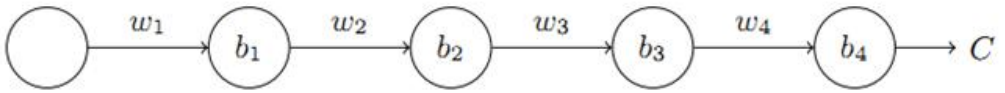
Initialization

- Avoid vanishing/exploding gradients in deep networks.
- Enable faster convergence during training.
- Ensure neurons start with diverse, non-zero activations.

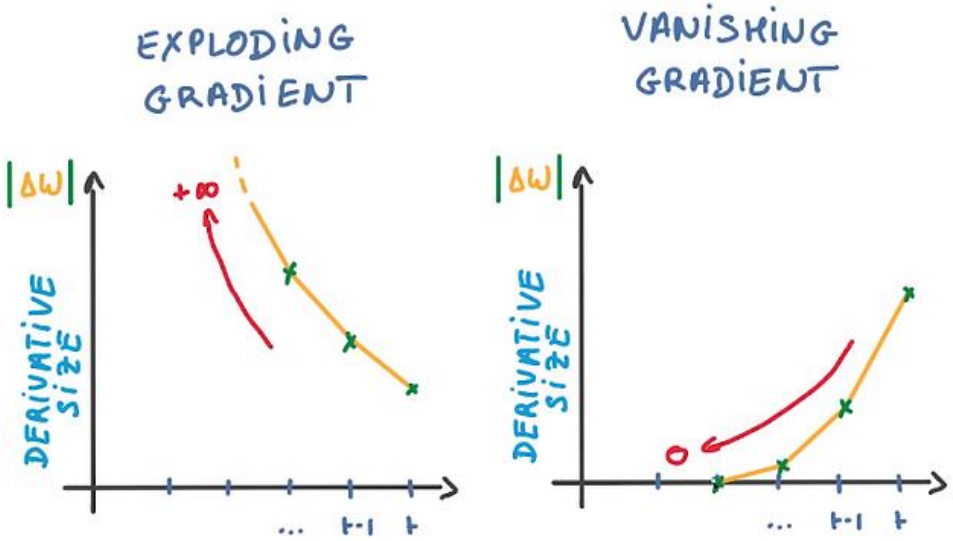


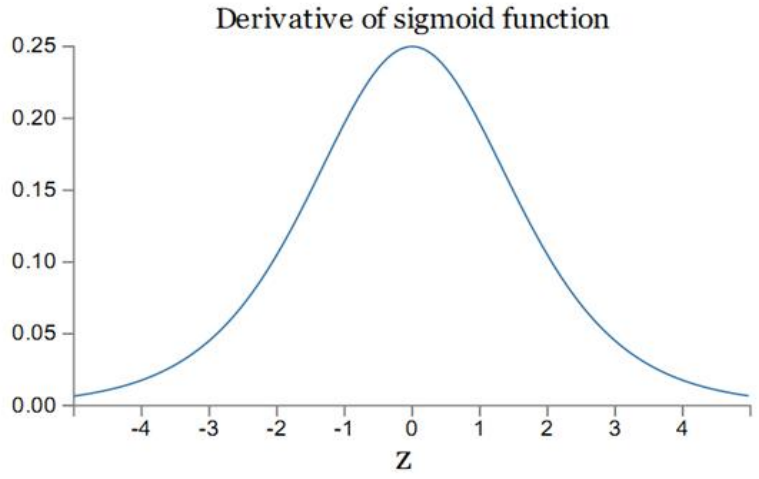
Initialization

- Avoid vanishing/exploding gradients in deep networks.
- Enable faster convergence during training.
- Ensure neurons start with diverse, non-zero activations.



$$\frac{\partial C}{\partial b_1} = \sigma'(z_1)\, w_2 \sigma'(z_2)\, w_3 \sigma'(z_3)\, w_4 \sigma'(z_4)\, \frac{\partial C}{\partial a_4}$$





Initialization

- Sample weights from a uniform distribution U or normal distribution N
  - ✓ Xavier Initialization (Linear activation function)

$$U[-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}], \quad N[0, \sqrt{\frac{2}{n_{in}+n_{out}}}]$$

  - ✓ He Initialization (Considering ReLU)

$$U[-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}}], \quad N[0, \sqrt{\frac{2}{n_{in}}}]$$

Xavier Initialization (Linear activation function)

1. 基础假设

- 输入 x_i 已经去均值（零均值）：$E[X_i] = 0$
- 权重 W_i 也是零均值：$E[W_i] = 0$
- X_i 与 W_i 相互独立

Xavier Initialization (Linear activation function)

2. 单乘积项的方差

由两个独立变量的乘积的方差公式：

$$\mathrm{Var}(W_i X_i) = [E(X_i)]^2 \mathrm{Var}(W_i) + [E(W_i)]^2 \mathrm{Var}(X_i) + \mathrm{Var}(X_i)\mathrm{Var}(W_i)$$

因为 $E[X_i] = 0$ 且 $E[W_i] = 0$，前两项为 0，剩下：

$$\mathrm{Var}(W_i X_i) = \mathrm{Var}(X_i) \cdot \mathrm{Var}(W_i)$$

Xavier Initialization (Linear activation function)

3. 整层输出的方差

令 $y = \sum_{i=1}^{n} W_i X_i$，由于各个 $W_i X_i$ 相互独立，有：

$$\mathrm{Var}(y) = \sum_{i=1}^{n} \mathrm{Var}(W_i X_i) = \sum_{i=1}^{n} \mathrm{Var}(X_i)\mathrm{Var}(W_i)$$

假设所有输入的方差相同：$\mathrm{Var}(X_i) = \mathrm{Var}(x)$，且所有权重初始化的方差相同：$\mathrm{Var}(W_i) = \mathrm{Var}(W)$，则：

$$\mathrm{Var}(y) = n \cdot \mathrm{Var}(x) \cdot \mathrm{Var}(W)$$

其中 $n = n_{\mathrm{in}}$，即输入的维度。

Xavier Initialization (Linear activation function)

4. 保持方差不变的条件

希望 $\mathrm{Var}(y) = \mathrm{Var}(x)$，于是：

$$n \cdot \mathrm{Var}(x) \cdot \mathrm{Var}(W) = \mathrm{Var}(x)$$

若 $\mathrm{Var}(x) \neq 0$，两边约去，得：

$$n \cdot \mathrm{Var}(W) = 1$$

所以：

$$\mathrm{Var}(W) = \frac{1}{n_{\text{in}}}$$

这就是 **Caffe 中 Xavier 初始化** 正向传播的方差设置。

Xavier Initialization (Linear activation function)

5. 反向传播的考虑（Glorot & Bengio 论文）

反向传播时，该层接收到的梯度相当于“反向的输入”，其维度是输出神经元的个数 n_{out}。
为了保持梯度方差在反向传播时不变，类似推导可得：

$$\text{Var}(W) = \frac{1}{n_{\text{out}}}$$

Xavier Initialization (Linear activation function)

6. 折中方案

Glorot & Bengio 同时考虑正向和反向传播，取两者的调和平均或直接取 $n_{\text{in}} + n_{\text{out}}$ 相关形式（如 $\text{Var}(W) = \frac{2}{n_{\text{in}}+n_{\text{out}}}$）。

均匀分布中用的 $\sqrt{\frac{6}{n_{\text{in}}+n_{\text{out}}}}$ 即是由此方差假设推导得到。

Normalization

- When input scales differ, the computed gradient scales also vary, making it difficult to determine an appropriate update step size.
  - ✓ Min-Max Scaling

$$x' = \frac{x - min(X)}{max(X) - min(X)}$$

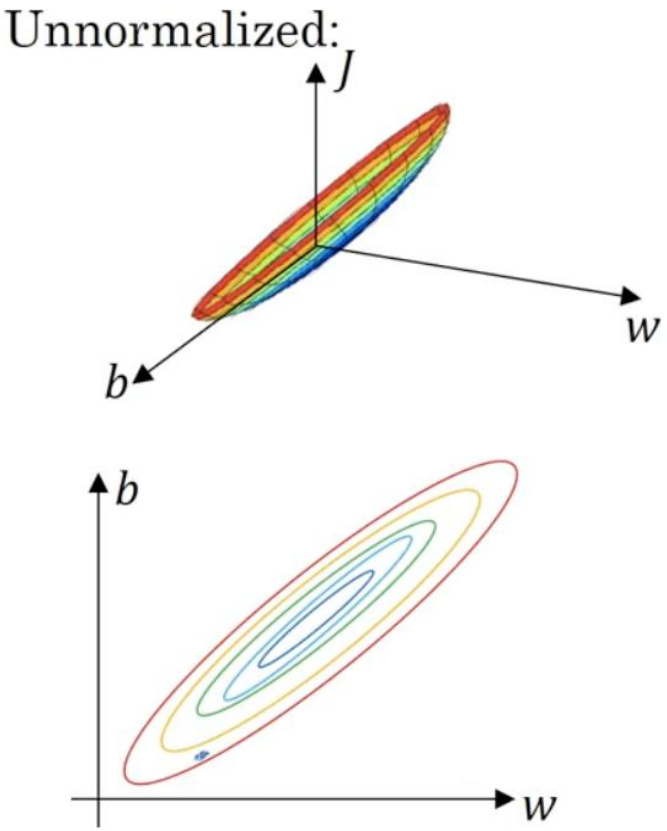
  - ✓ Standardization

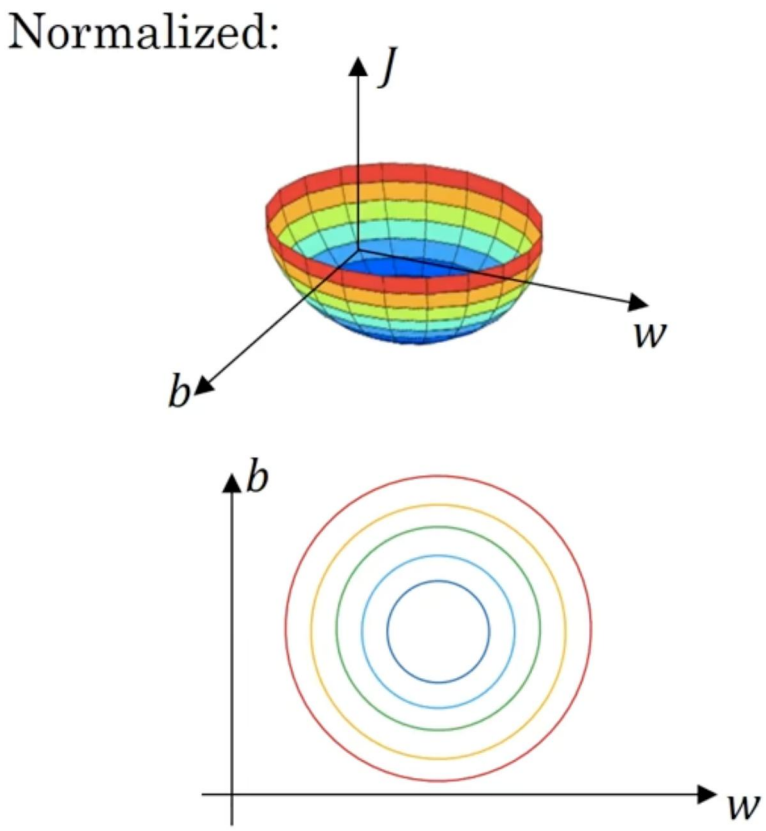
$$x' = \frac{x - \mu(X)}{\sigma(X)}$$

$\mu(X)$ is the dataset mean, $\sigma(X)$ is the dataset variance

Normalization

- Visualization.





上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Contents

- Introduction & Overview
- Multi-Layer Perceptron
- Training Neural Networks
- **Case Analysis**

任务：预测病人是否患有糖尿病

确诊依据是什么？

历史病例数据

ID	年龄	姓名	体重(kg)	身高(cm)	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	45	张三	76	165	28.1	6.8	85	是	是
2	29	李四	NULL	175	NULL	5.1	64	否	否
3	61	王五	82	160	32.0	15.3	142	是	是
4	35	赵六	71	170	24.5	5.6	NULL	否	否
5	52	林七	83	166	30.2	6.2	88	是	是

任务：预测病人是否患有糖尿病

- **基本术语**

ID	年龄	姓名	体重(kg)	身高(cm)	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	45	张三	76	165	28.1	6.8	85	是	是
2	29	李四	NULL	175	NULL	5.1	64	否	否
3	61	王五	82	160	32.0	15.3	142	是	是
4	35	赵六	71	170	24.5	5.6	NULL	否	否
5	52	林七	83	166	30.2	6.2	88	是	是

数据、数据集

任务：预测病人是否患有糖尿病

- **基本术语**

样本

ID	年龄	姓名	体重(kg)	身高(cm)	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	45	张三	76	165	28.1	6.8	85	是	是
2	29	李四	NULL	175	NULL	5.1	64	否	否
3	61	王五	82	160	32.0	15.3	142	是	是
4	35	赵六	71	170	24.5	5.6	NULL	否	否
5	52	林七	83	166	30.2	6.2	88	是	是

家族史：特征、属性
{是,否}：特征值、属性值

任务：预测病人是否患有糖尿病

- **基本术语**

是否患病：标签属性、标签
{是,否}：标签值、样本标签

ID	年龄	姓名	体重(kg)	身高(cm)	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	45	张三	76	165	28.1	6.8	85	是	是
2	29	李四	NULL	175	NULL	5.1	64	否	否
3	61	王五	82	160	32.0	15.3	142	是	是
4	35	赵六	71	170	24.5	5.6	NULL	否	否
5	52	林七	83	166	30.2	6.2	88	是	是

任务：预测病人是否患有糖尿病

是否所有特征都有用？

无用信息让模型“学偏了方向”

ID	年龄	姓名	体重(kg)	身高(cm)	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	45	张三	76	165	28.1	6.8	85	是	是
2	29	李四	NULL	175	NULL	5.1	64	否	否
3	61	王五	82	160	32.0	15.3	142	是	是
4	35	赵六	71	170	24.5	5.6	NULL	否	否
5	52	林七	83	166	30.2	6.2	88	是	是

冗余的信息留着有影响吗

冗余特征让模型“重复学习”

为了让模型专注于真正有用的信息，需要进行特征筛选

任务：预测病人是否患有糖尿病

- **特征筛选**

与BMI冗余，删除

ID	年龄	姓名	体重(kg)	身高(cm)	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	45	张三	76	165	28.1	6.8	85	是	是
2	29	李四	NULL	175	NULL	5.1	64	否	否
3	61	王五	82	160	32.0	15.3	142	是	是
4	35	赵六	71	170	24.5	5.6	NULL	否	否
5	52	林七	83	166	30.2	6.2	88	是	是

对预测糖尿病无用，删除

- **特征筛选**：通过剔除无关或冗余信息，提升模型的效率与准确性

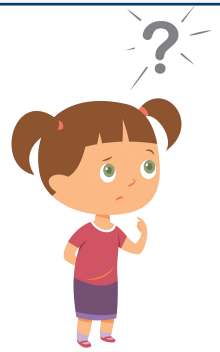
任务：预测病人是否患有糖尿病

缺失的值怎么处理？

缺失信息会让模型难以学习

正常空腹血糖一般<7.0 mmol/L
是**录入错误**？还是**真实高危**？

极端值可能严重拉偏模型参数



ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	45	28.1	6.8	85	是	是
2	29	NULL	5.1	64	否	否
3	61	32.0	15.3	142	是	是
4	35	24.5	NULL	NULL	否	否
5	52	30.2	6.2	88	是	是

6.2和88数值相差悬殊，怎么办？

尺度不一致会让模型过度关注大数值特征，忽略重要但值小的特征

为了让模型准确理解数据，要先对原始数据进行清洗

任务：预测病人是否患有糖尿病

- **数据清洗**

标记为异常值

删除该样本

用平均值填充

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	45	28.1	6.8	85	是	是
2	29	NULL	5.1	64	否	否
3	61	32.0	15.3	142	是	是
4	35	24.5	NULL	NULL	否	否
5	52	30.2	6.2	88	是	是

- **数据清洗**：处理缺失、异常或不一致的数据，提升数据质量与可靠性
- **缺失值处理**：通过填充、删除或建模方式补全数据
- **异常值处理**：识别并合理处理以确保数据分布稳定

任务：预测病人是否患有糖尿病

- **数据清洗**

相较于血糖指标数值偏大

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	45	28.1	6.8	85	是	是
2	29	30.1	5.1	64	否	否
3	61	32.0	15.3	142	是	是
5	52	30.2	6.2	88	是	是

- 归一化：将数据缩放到 [0, 1] 区间，避免数值大的特征主导模型学习

  归一化后的数据 = (原值 - 最小值) / (最大值 - 最小值)

任务：预测病人是否患有糖尿病

- **数据清洗**

归一化

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	0.50	28.1	6.8	85	是	是
2	0.00	30.1	5.1	64	否	否
3	1.00	32.0	15.3	142	是	是
5	0.71	30.2	6.2	88	是	是

- 归一化：将数据缩放到 [0, 1] 区间，避免数值大的特征主导模型学习

  归一化后的数据 = (原值 - 最小值) / (最大值 - 最小值)

任务：预测病人是否患有糖尿病

- **数据清洗**

不同检测指标存在分布差异

归一化

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	0.50	28.1	6.8	85	是	是
2	0.00	30.1	5.1	64	否	否
3	1.00	32.0	15.3	142	是	是
5	0.71	30.2	6.2	88	是	是

- **归一化**：将数据缩放到 [0, 1] 区间，避免数值大的特征主导模型学习

  归一化后的数据 = (原值 - 最小值) / (最大值 - 最小值)

- **标准化**：将数据变换为均值为0、标准差为1的分布，避免偏态影响模型判断

  标准化后的数据 = (原值 - 均值) / 标准差

任务：预测病人是否患有糖尿病

- **数据清洗**

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	0.50	−1.43	6.8	−0.319	是	是
2	0.00	0.00	5.1	−1.010	否	否
3	1.00	+1.36	15.3	+1.548	是	是
5	0.71	+0.07	6.2	−0.219	是	是

归一化　标准化　标准化

- **归一化**：将数据缩放到 [0, 1] 区间，避免数值大的特征主导模型学习

  归一化后的数据 = (原值 - 最小值) / (最大值 - 最小值)

- **标准化**：将数据变换为均值为0、标准差为1的分布，避免偏态影响模型判断

  标准化后的数据 = (原值 - 均值) / 标准差

任务：预测病人是否患有糖尿病

- **数据清洗**

离散化

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	0.50	−1.43	正常	−0.319	是	是
2	0.00	0.00	低	−1.010	否	否
3	1.00	+1.36	高	+1.548	是	是
5	0.71	+0.07	正常	−0.219	是	是

- **离散化**：将连续变量划分为类别，以提高模型解释性

任务：预测病人是否患有糖尿病

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	0.50	−1.43	正常	−0.319	是	是
2	0.00	0.00	低	−1.010	否	否
3	1.00	+1.36	高	+1.548	是	是
5	0.71	+0.07	正常	−0.219	是	是

数据已清洗并准备完毕，模型训练可以开始了！

任务：预测病人是否患有糖尿病

- **模型训练**

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	0.50	−1.43	正常	−0.319	是	是
2	0.00	0.00	低	−1.010	否	否
3	1.00	+1.36	高	+1.548	是	是
5	0.71	+0.07	正常	−0.219	是	是

- **预测模型**：函数“f(病人体征) = 是否患病”
- **训练**：模型从“历史病例”中学习上述函数的过程
- **测试**：给定新病患数据，通过训练后的预测模型判断是否患病过程

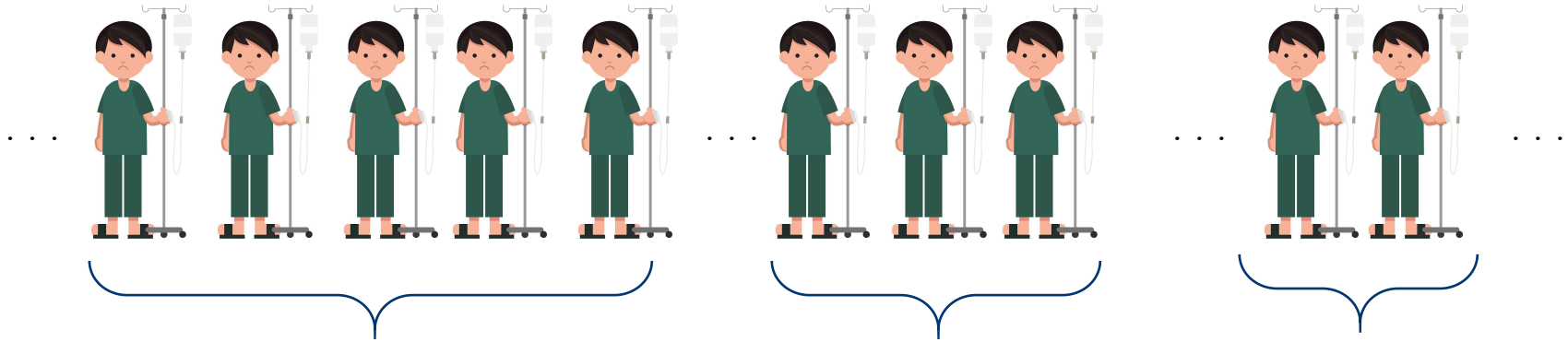
训练数据

测试数据

任务：预测病人是否患有糖尿病

- **模型训练**

100个病例：



训练集 (70例)
教模型判断的病例

验证集 (20例)
供模型“查缺补漏”

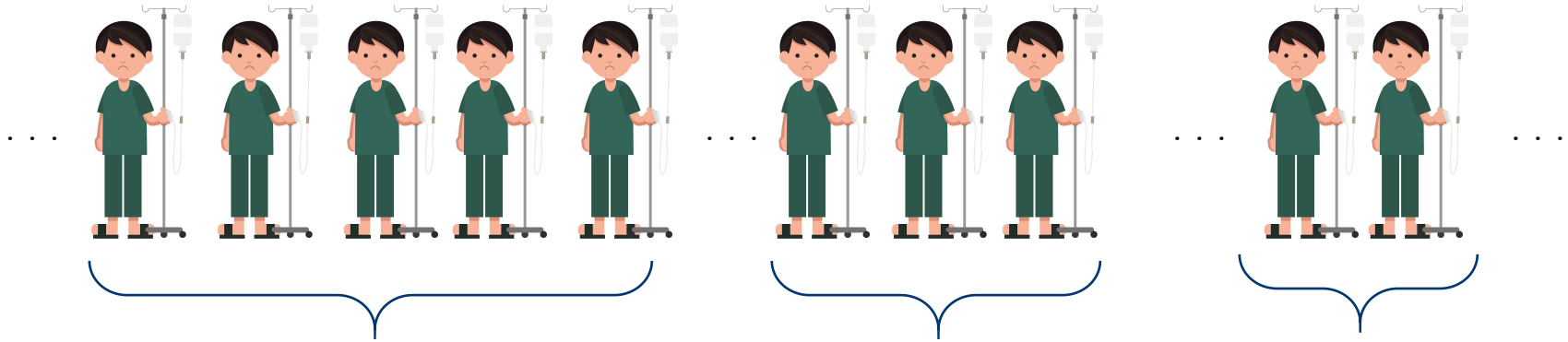
测试集 (10例)
评估模型性能

- **训练集**：训练数据的集合，用于学习模型参数
- **验证集**（可选）：训练过程中对模型进行选择与评估
- **测试集**：测试数据的集合，模型最终服务对象，用于评估模型性能

任务：预测病人是否患有糖尿病

- **模型训练**

100个病例：



训练集 (70例)
教模型判断的病例

验证集 (20例)
供模型“查缺补漏”

测试集 (10例)
评估模型性能

- **泛化能力**：模型对新样本（未来样本）的预测能力
- **测试集性能** 估计 **泛化性能**

ATTENTION

三个集合的数据不重叠

提升模型的泛化性能，是所有机器学习模型的终极目标

任务：预测病人是否患有糖尿病

eg.模型学到的每个病症权重=模型参数

- **模型训练**

eg.每次模型学习的病例数=超参数

x0.7

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	0.50	−1.43	正常	−0.319	是	是
2	0.00	0.00	低	−1.010	否	否
3	1.00	+1.36	高	+1.548	是	是
5	0.71	+0.07	正常	−0.219	是	是

- **模型参数**：通过训练数据优化更新，由学习过程确定
- **算法参数**：由人工在学习开始前根据先验知识定义，称“超参数”

所有参数都可能对模型最终性能有关键影响

任务：预测病人是否患有糖尿病

- **模型训练**

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病	患病程度
1	0.50	−1.43	正常	−0.319	是	是	0.6
2	0.00	0.00	低	−1.010	否	否	0.2
3	1.00	+1.36	高	+1.548	是	是	0.8
5	0.71	+0.07	正常	−0.219	是	是	0.7
						分类任务	回归任务

- **分类任务**：对“是否患病”的类别进行预测（离散值）
- **回归任务**：对“患病”的程度如0.8，0.32进行预测（连续值）

任务：预测病人是否患有糖尿病

- **模型训练**

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	0.50	−1.43	正常	−0.319	是	是
2	0.00	0.00	低	−1.010	否	否
3	1.00	+1.36	高	+1.548	是	是
5	0.71	+0.07	正常	−0.219	是	是

生成式任务

6	0.75	+1.25	正常	+1.021	是	是

- **判别式任务**：模型学习 X → Y 的映射（如输入特征 → 是否患病）
- **生成式任务**：生成式模型关注于建模数据的分布，能“生成”出新的样本

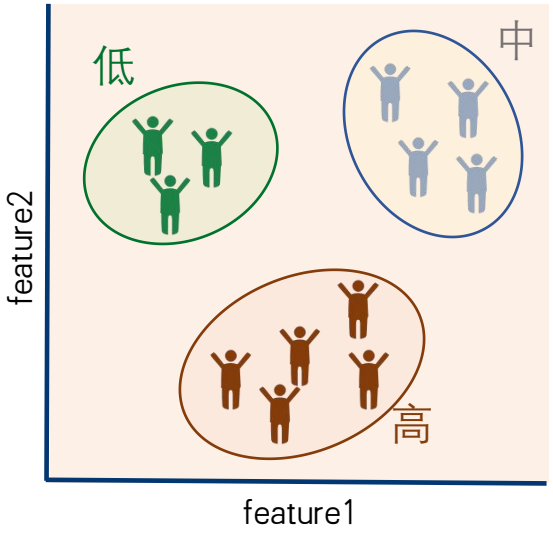
任务：预测病人是否患有糖尿病

无标签，找出数据中的内在规律/分群

- **模型训练**

ID	年龄	BMI	空腹血糖 (mmol/L)	血清胰岛素	家族史	是否患病
1	0.50	−1.43	正常	−0.319	是	是
2	0.00	0.00	低	−1.010	否	否
3	1.00	+1.36	高	+1.548	是	是
5	0.71	+0.07	正常	−0.219	是	是

糖尿病风险群体划分

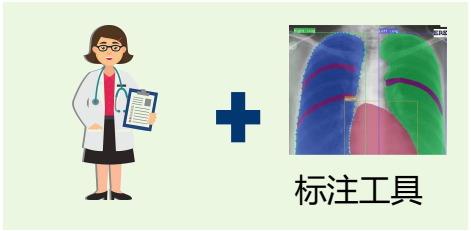


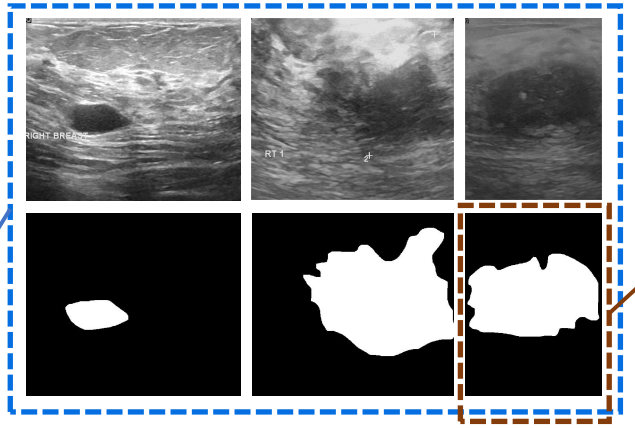
- **有监督学习**：训练数据带有标签信息（模型训练直接获得标签监督反馈）
- **无监督学习**：训练过程无法获得训练数据标签信息

（低目标性，寻找数据隐藏结构，常用于先驱任务，聚类、降维等）

Case Analysis

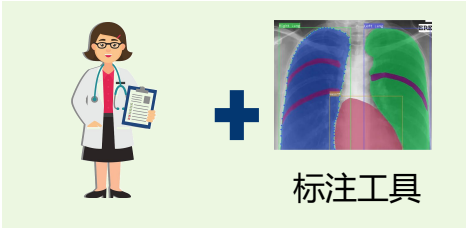
明确问题 → 获取数据 → 特征选择 → 数据清洗 → 模型训练 → 上线运行

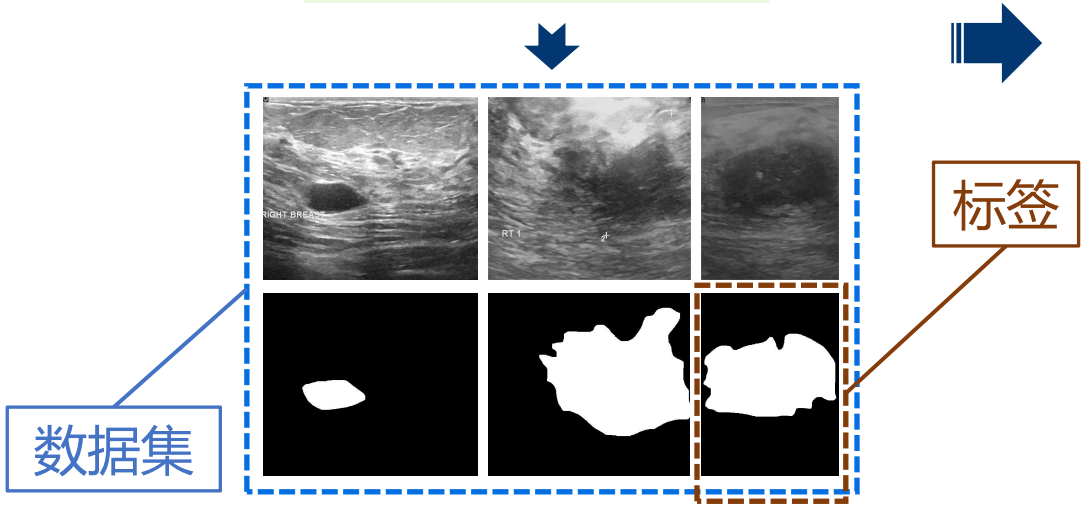




Case Analysis

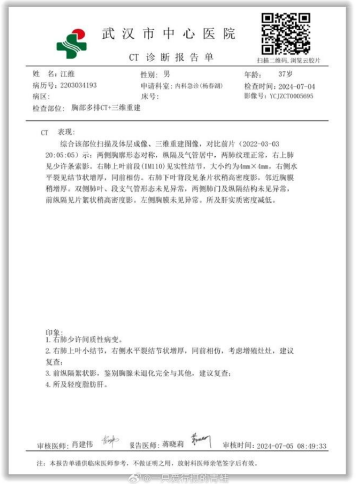
明确问题 → 获取数据 → 特征选择 → 数据清洗 → 模型训练 → 上线运行



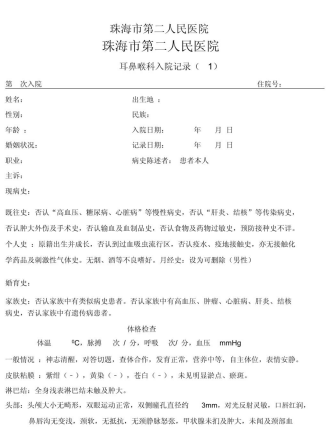


标签的获取方式：

- **粗略标注**：依据诊断报告或电子病历（EMR）自动提取标签（如“良/恶性”“疾病类型”）



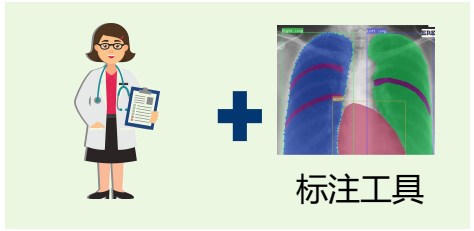
武汉市中心医院
CT诊断报告单

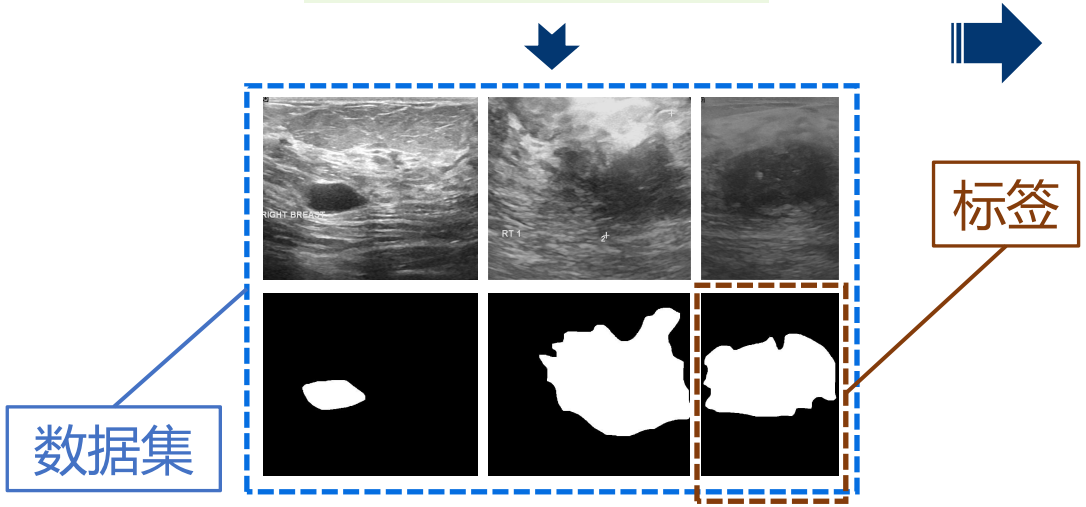


珠海市第二人民医院
珠海市第二人民医院
耳鼻喉科入院记录（ 1）

Case Analysis

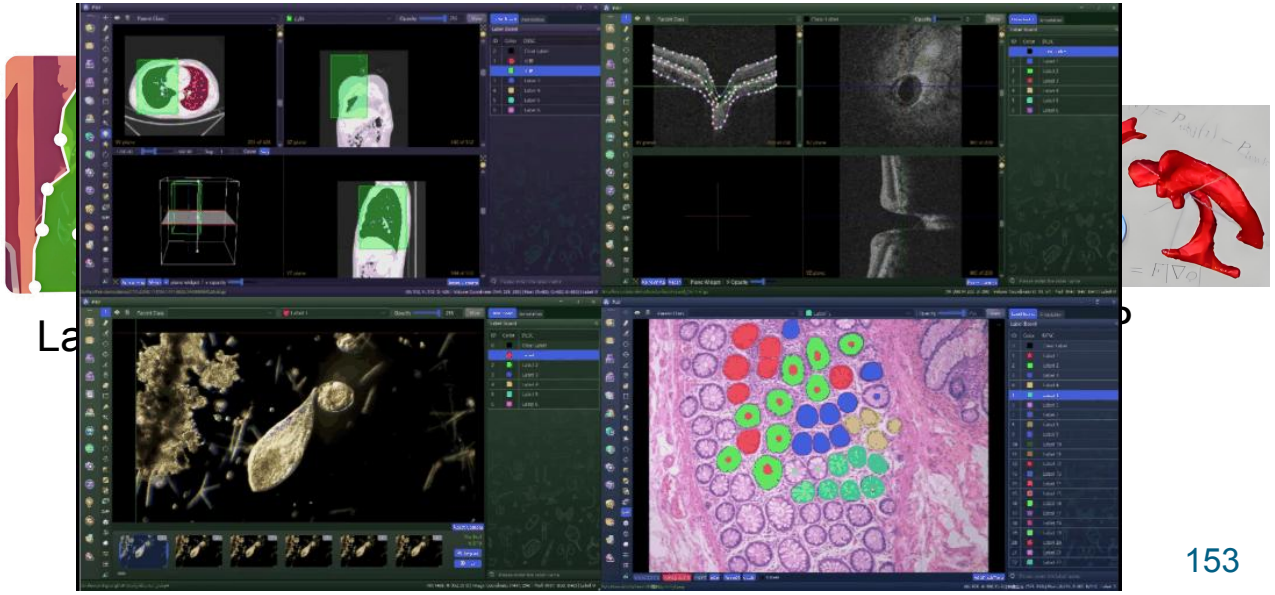
明确问题 → 获取数据 → 特征选择 → 数据清洗 → 模型训练 → 上线运行





标签的获取方式：

- **细致标注**：医生使用**标注工具**逐像素勾画器官或病灶区域

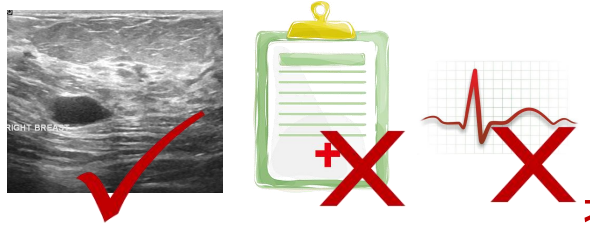


Case Analysis

明确问题 → 获取数据 → 特征选择 → 数据清洗 → 模型训练 → 上线运行

Optional

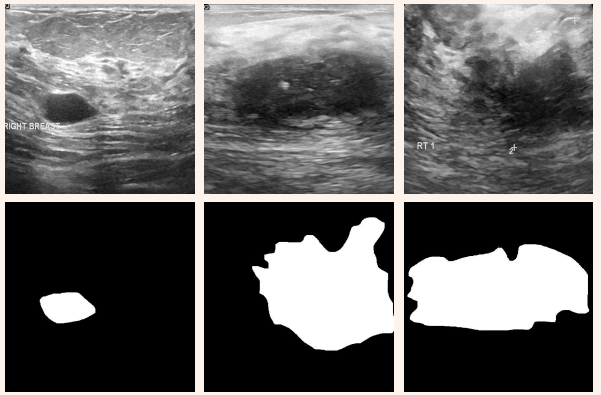
选择与问题相关的特征



提高数据质量

归一化、标准化、离散化、因子化、去除共线性、缺失值处理、……

eg.



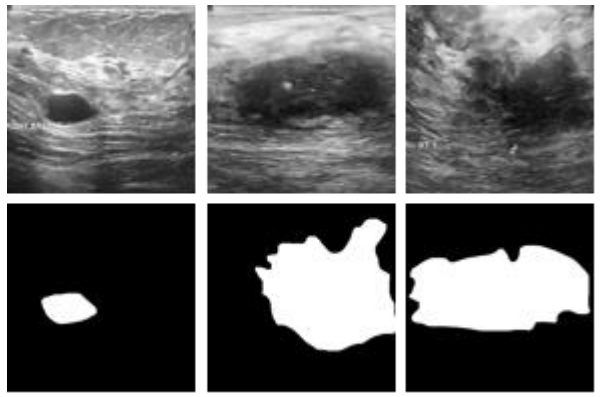
- **图像裁剪**：统一大小，去除无关背景
- **灰度归一化**：统一图像亮度/对比度
- **去除噪声**：消除伪影、扫描误差

Case Analysis

明确问题 → 获取数据 → 特征选择 → 数据清洗 → 模型训练 → 上线运行

Optional

- [x] 有监督学习
- [ ] 半监督学习
- [ ] 无监督学习



有监督学习：图像+标签

DATASET
训练集

MODEL

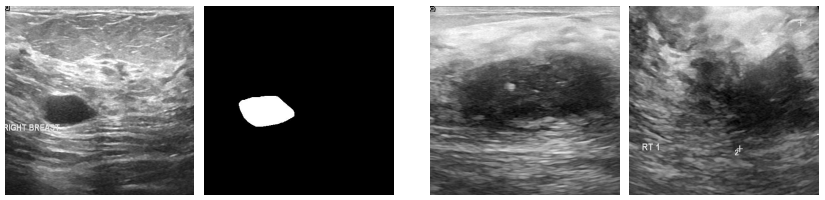
给模型大量有标签的数据，模型不断学习**“影像特征”**与**“肿瘤区域”**之间的对应关系，最终学会自动从图像中精准分割出肿瘤。

Case Analysis

明确问题 → 获取数据 → 特征选择 → 数据清洗 → 模型训练 → 上线运行

Optional

- ☐ 有监督学习
- ☑ 半监督学习
- ☐ 无监督学习



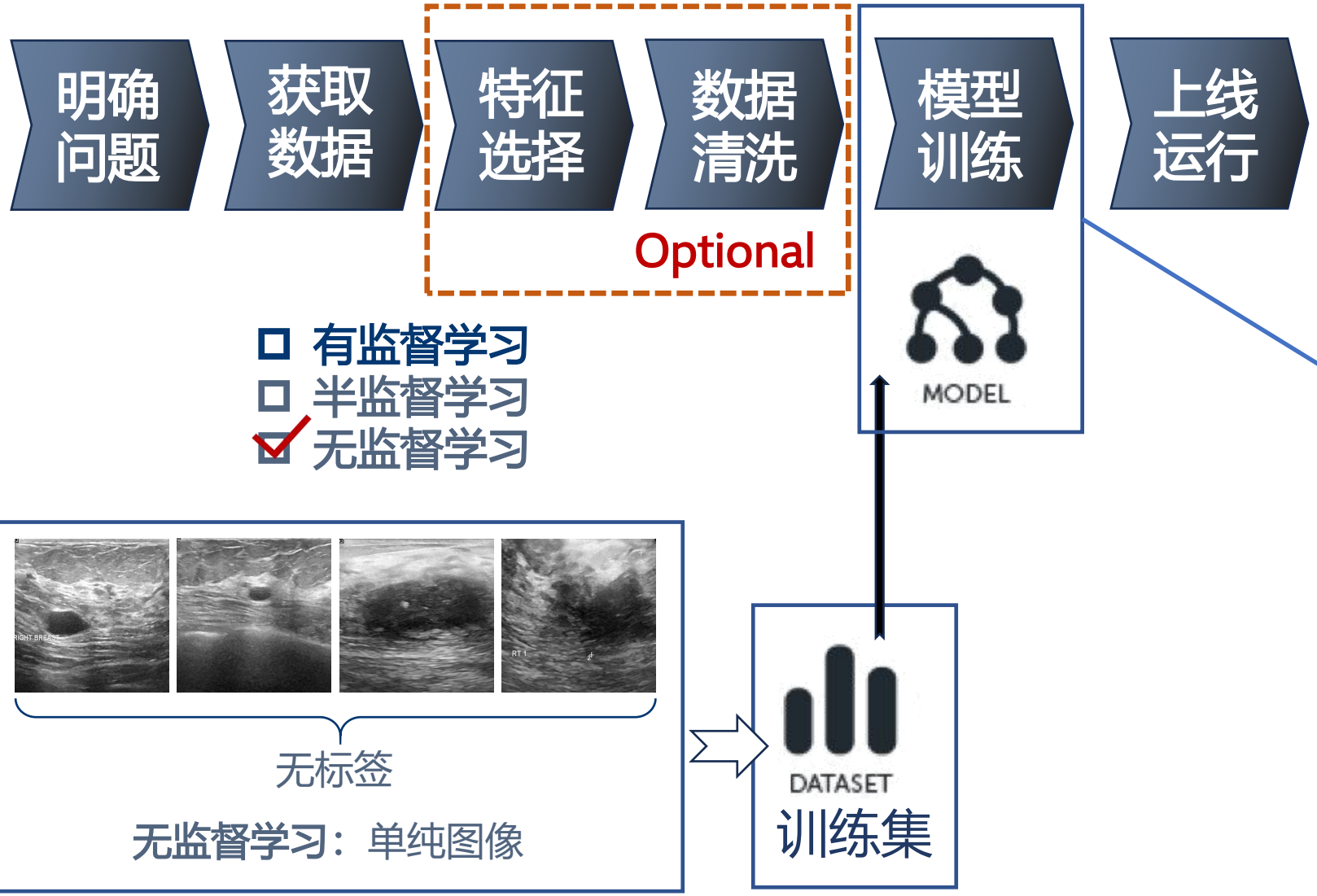
有标签　　无标签

半监督学习：少量标注+大量未标注

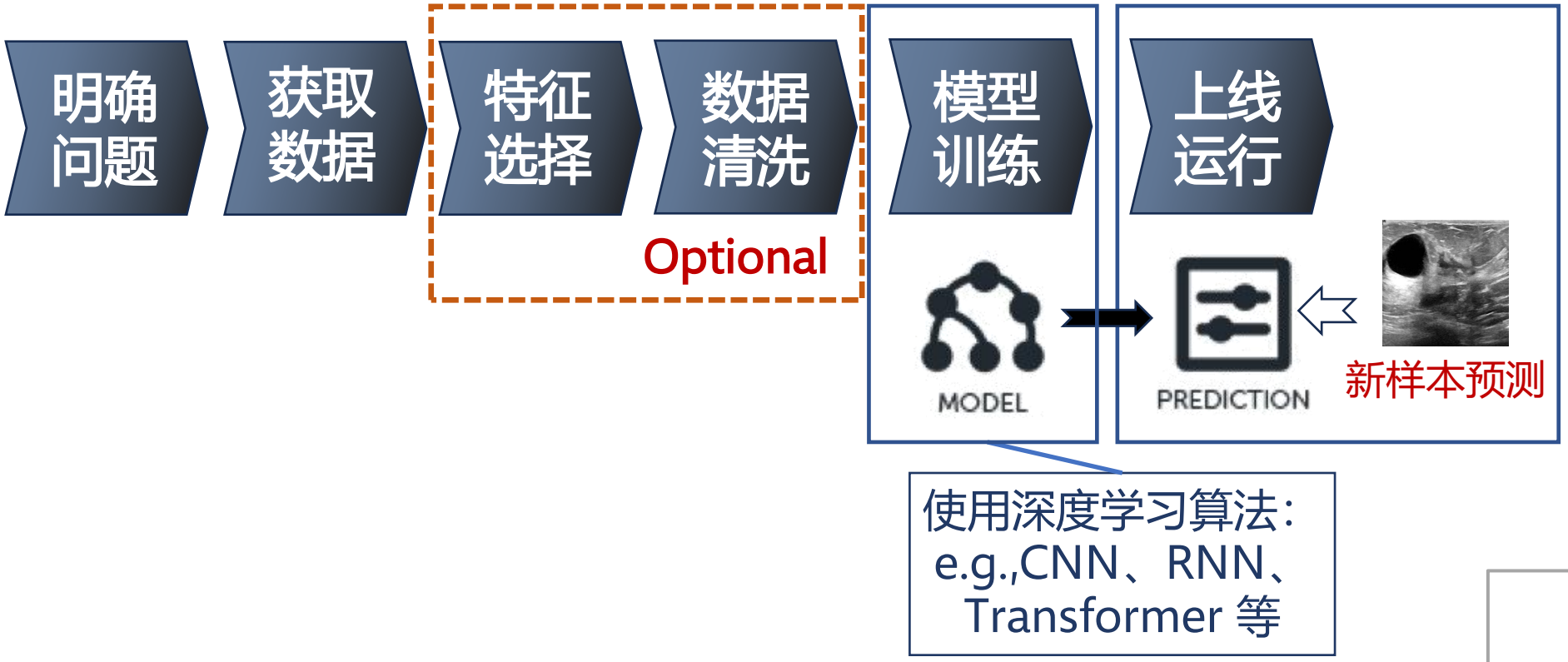
训练集

让模型在“**看懂部分图像**”的基础上学会处理“没标注”的图像，提高整体性能、减少人工标注成本

Case Analysis

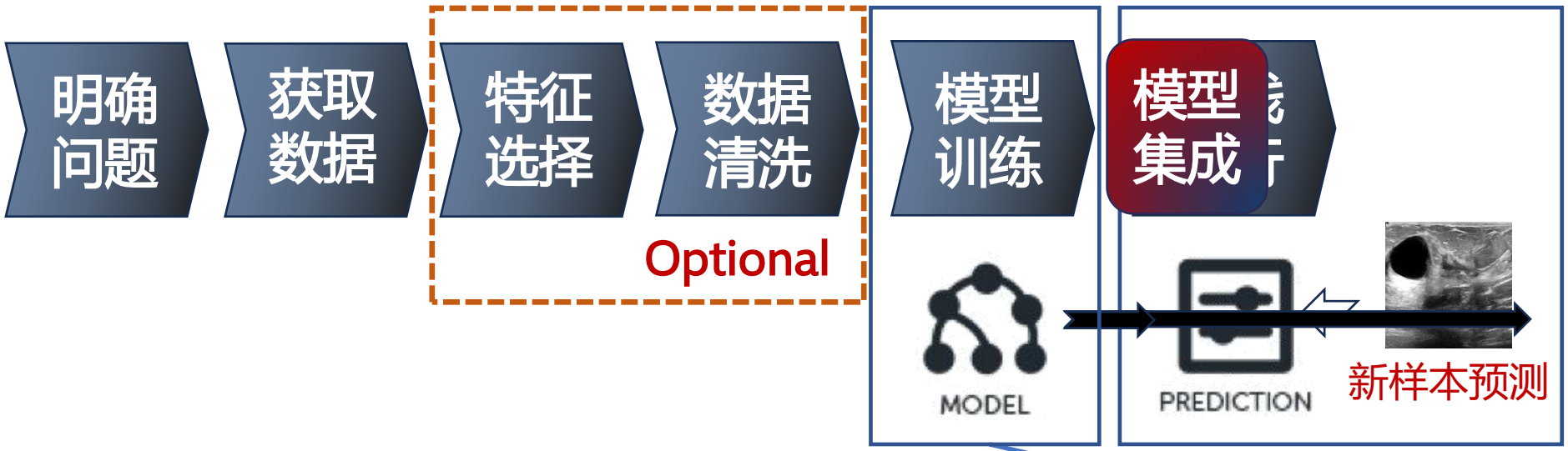


Case Analysis



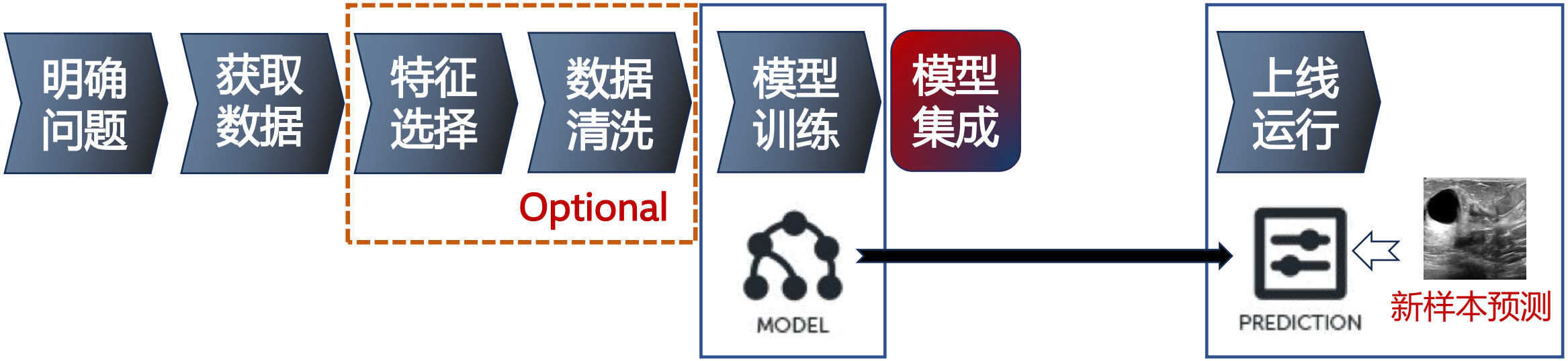
性能如何提

Case Analysis



性能如何提

Case Analysis



模型集成：构建并结合多个弱学习器（模型）来提升性能

（三个臭皮匠顶个诸葛亮）

两种常见集成策略：

- **Bagging**：个体分类器的训练相互独立
- **Boosting**：依次训练不同分类器，后者基于前者进行性能提升

性能如何提

Case Analysis

明确问题 → 获取数据 → 特征选择 → 数据清洗 → 模型训练 → 模型集成 → 模型评估与选择 → 上线运行

Optional

e.g. 定位乳腺病变?

选择与问题相关的特征

提高数据质量

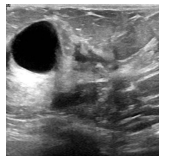
归一化、标准化、离散化、因子化、去除共线性、缺失值处理、……

DATASET
训练集

使用深度学习算法:
e.g.,CNN、RNN、Transformer 等

MODEL

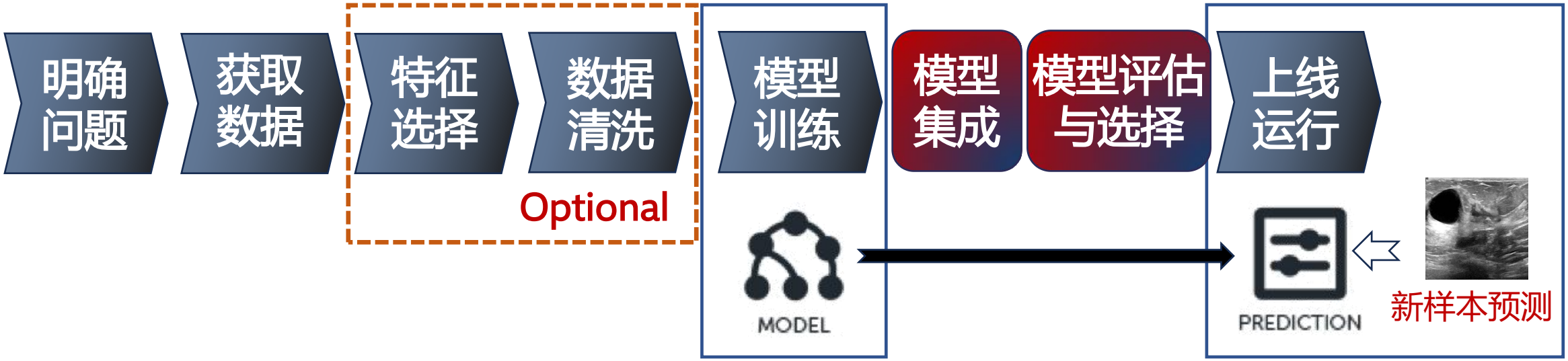
PREDICTION



新样本预测

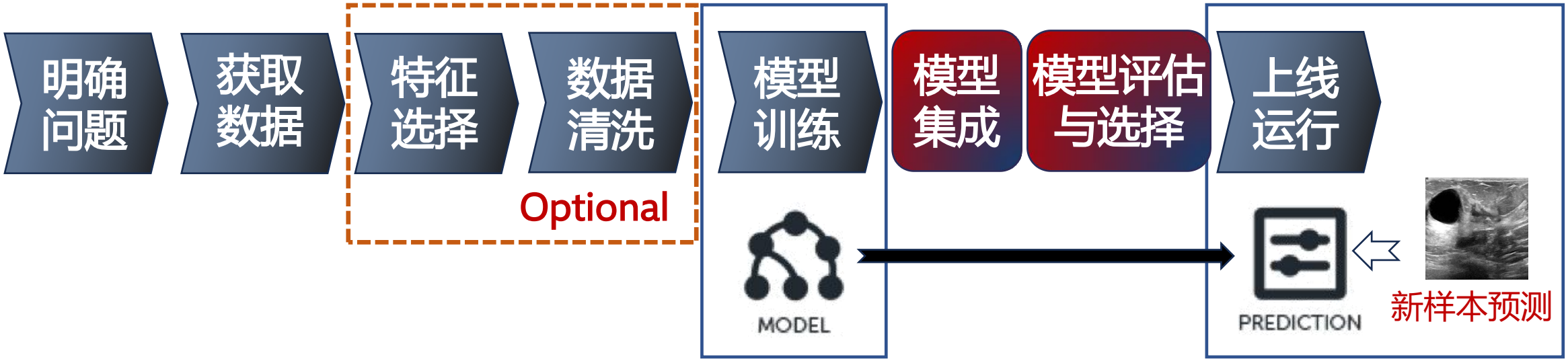
性能好?

Case Analysis



- **模型评估**：衡量模型在样本上的适用性（误差小）
- **模型选择**：基于评估选择更适合的模型（泛化能力强）

Case Analysis



- **模型评估**：衡量模型在样本上的适用性（误差小）
- **模型选择**：基于评估选择更适合的模型（泛化能力强）

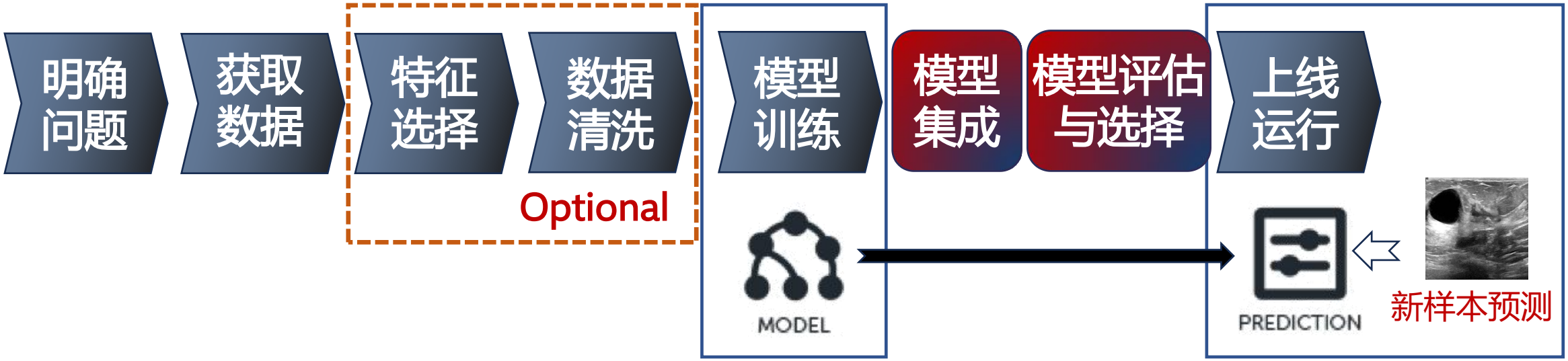
1.如何获得测试结

2.如何评估性能优

评估方法

性能好？

Case Analysis



- **模型评估**：衡量模型在样本上的适用性（误差小）
- **模型选择**：基于评估选择更适合的模型（泛化能力强）

1.如何获得测试结

评估方法

2.如何评估性能优

性能度量

性能好？

Thanks

Welcome to join VisionXLab

yangxue.site