中国科学院大学
University of Chinese Academy of Sciences

自然语言处理

第2讲 神经网络基础

王 石 资康莉 刘 瑜

2026年春季课程

https://ictkc.github.io/teaching/

第二讲 神经网络基础

神经网络语言模型

（3）输出每个词的概率以及最可能的词

（2）神经网络

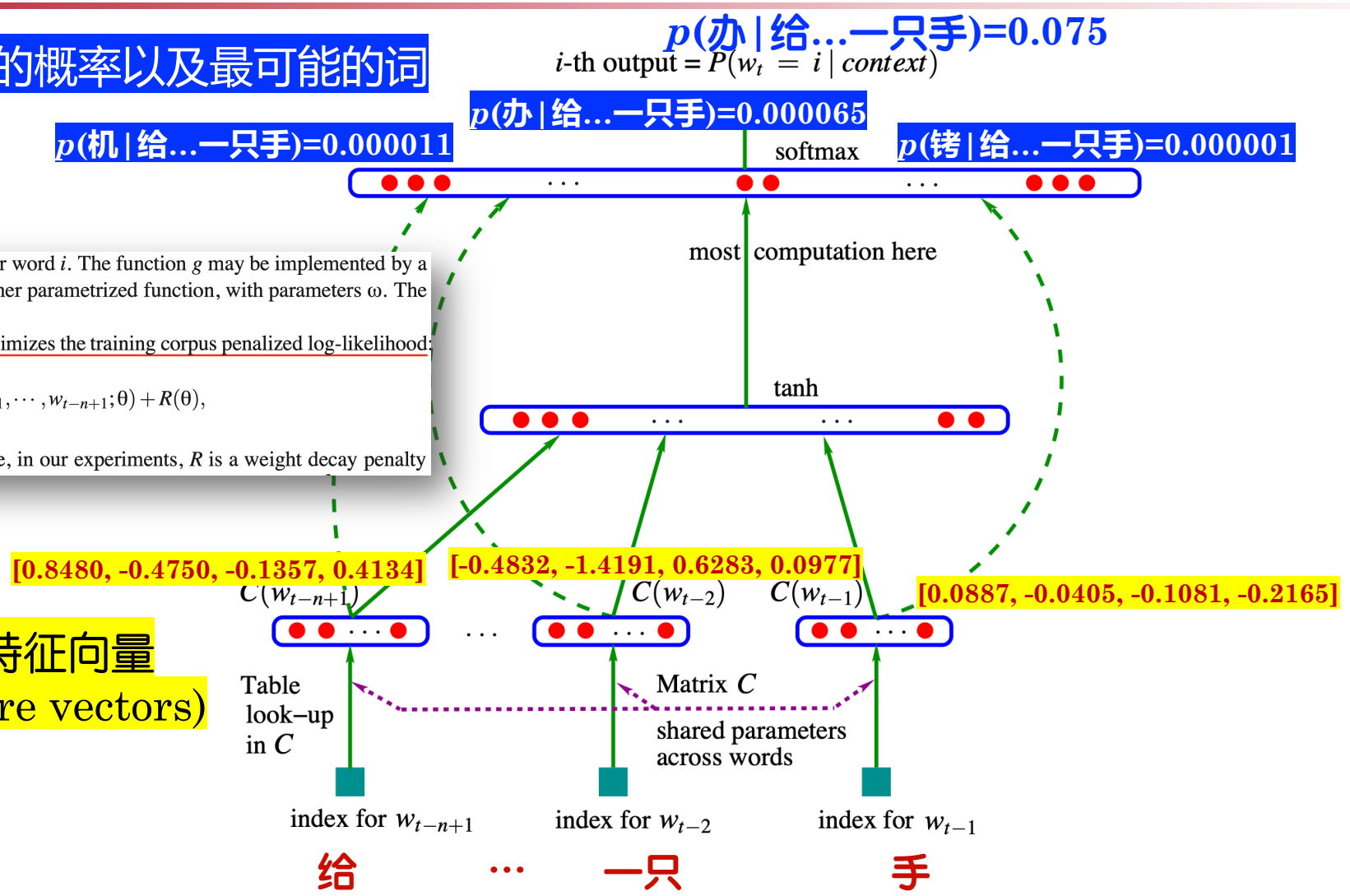
matrix C whose row i is the feature vector $C(i)$ for word i. The function g may be implemented by a feed-forward or recurrent neural network or another parametrized function, with parameters ω. The overall parameter set is $\theta = (C, \omega)$.

Training is achieved by looking for θ that maximizes the training corpus penalized log-likelihood:

$$L = \frac{1}{T} \sum_t \log f(w_t, w_{t-1}, \cdots, w_{t-n+1}; \theta) + R(\theta),$$

where $R(\theta)$ is a regularization term. For example, in our experiments, R is a weight decay penalty

（1）词分布式特征向量
(distributed feature vectors)



神经元（神经细胞）

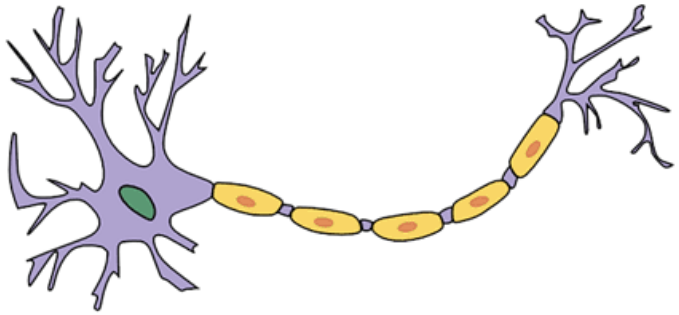
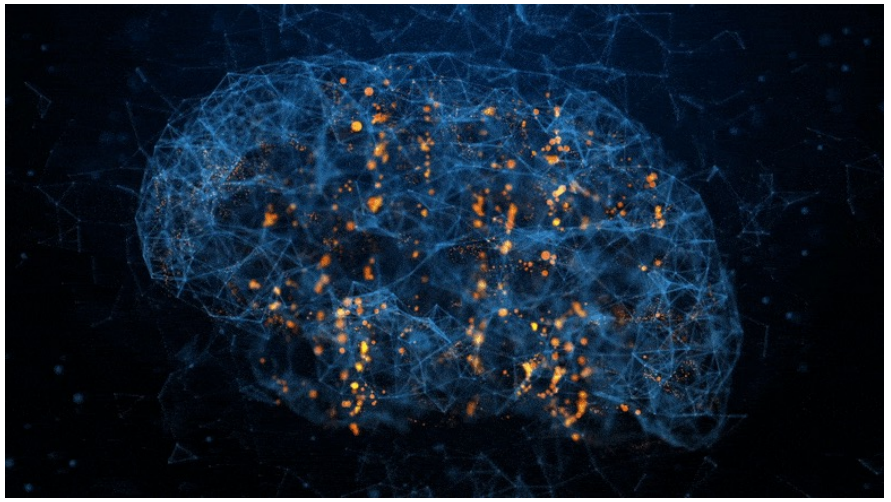
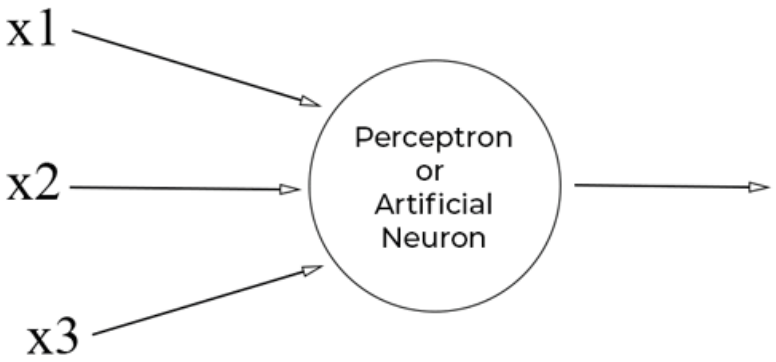
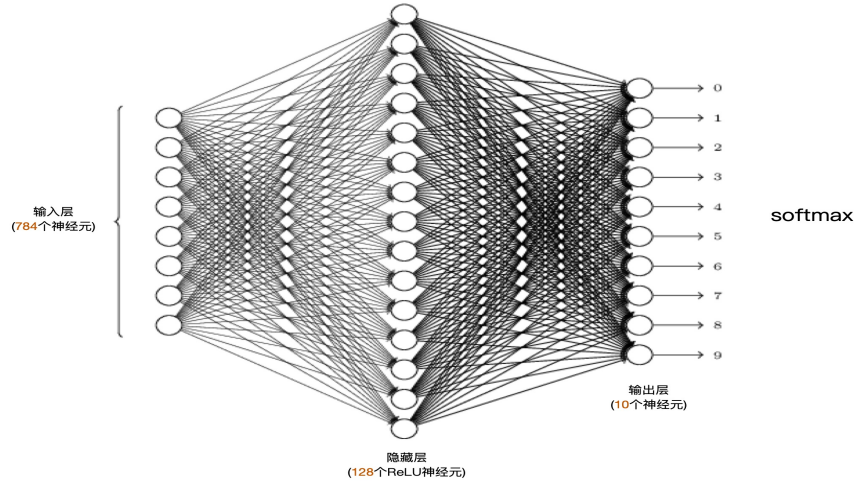


Fig: Biological Neuron

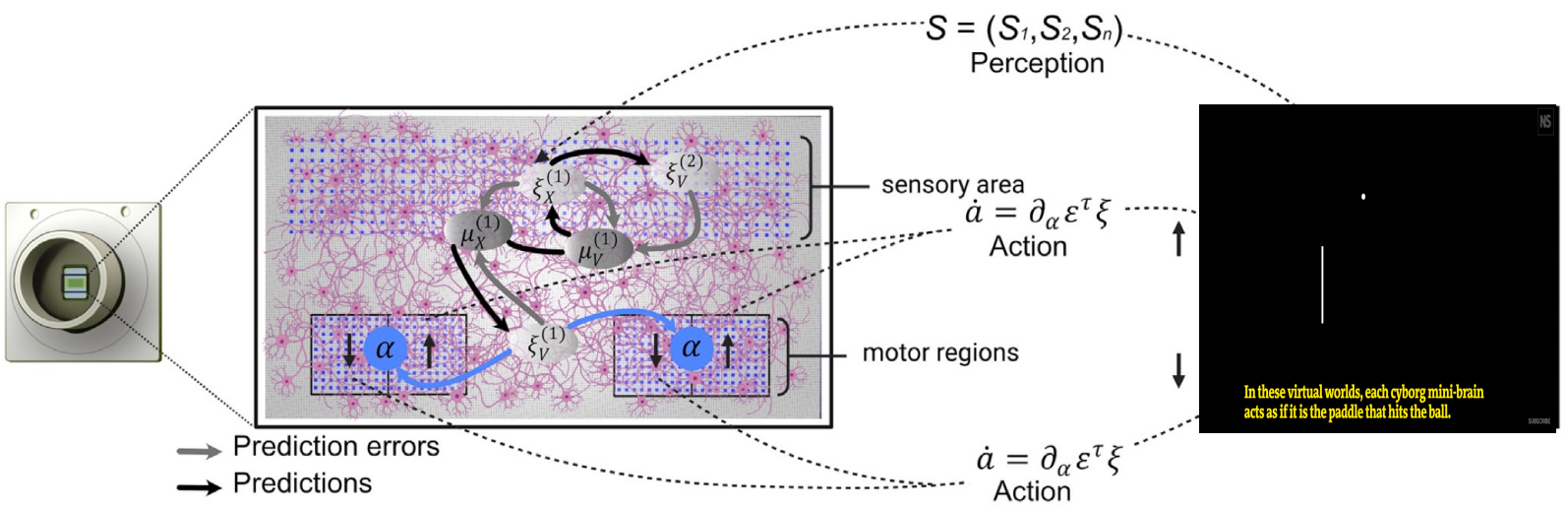


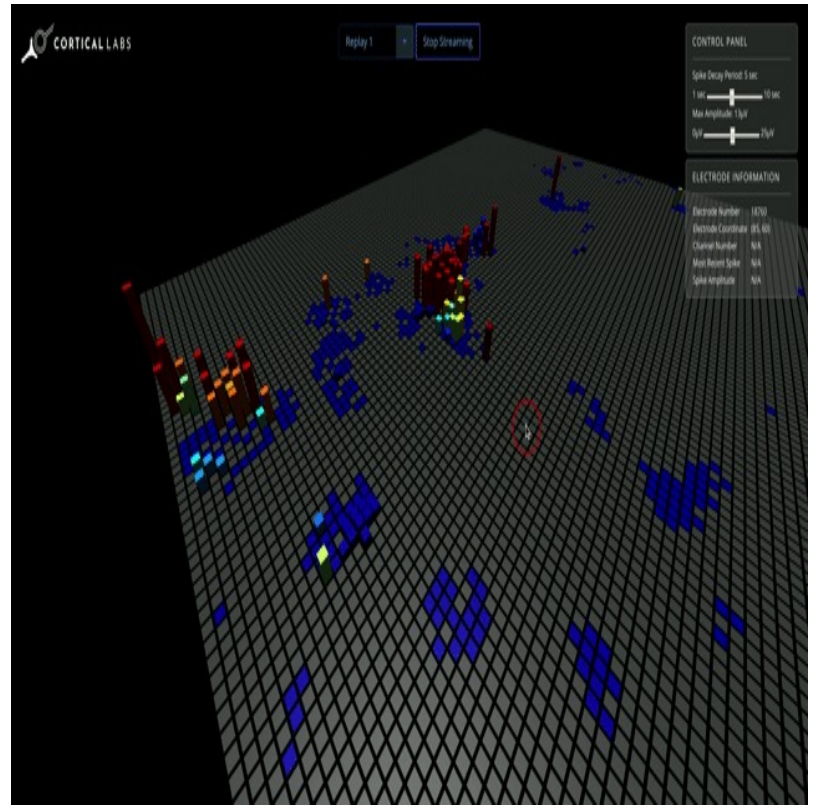




神经网络

归根结底，人类智能的物质基础是约千亿级神经元和百万亿神经突触连接的生物组织





100万个神经细胞在培养皿中经“电击”训练，学会了弹球游戏

Kagan B J, Kitchen A C, Tran N T, et al. In vitro neurons learn and exhibit sentience when embodied in a simulated game-world[J]. Neuron, 2022, 110(23): 3952-3969. e8.

中国科学院大学
University of Chinese Academy of Sciences

目 录

神经元的简化数学模型：感知机（Perceptron）

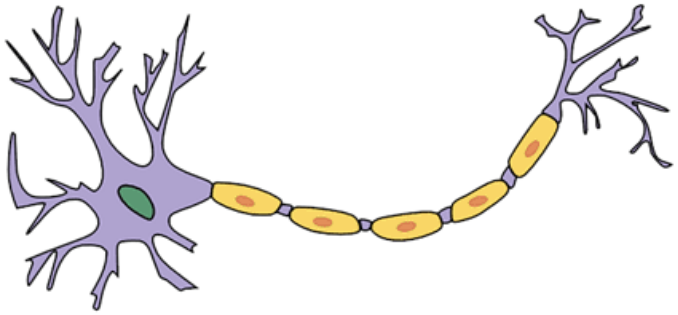
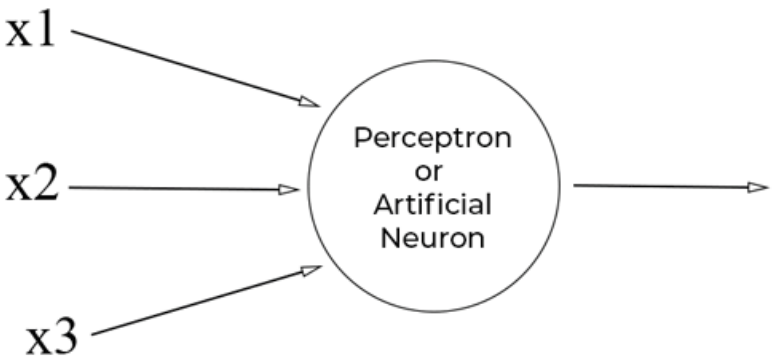
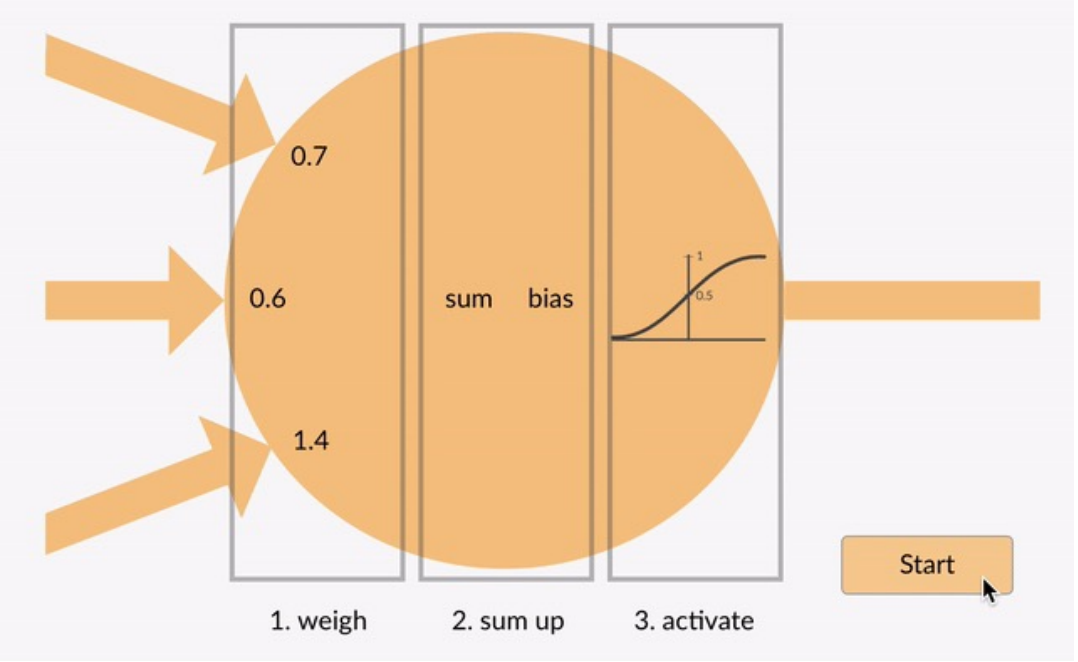
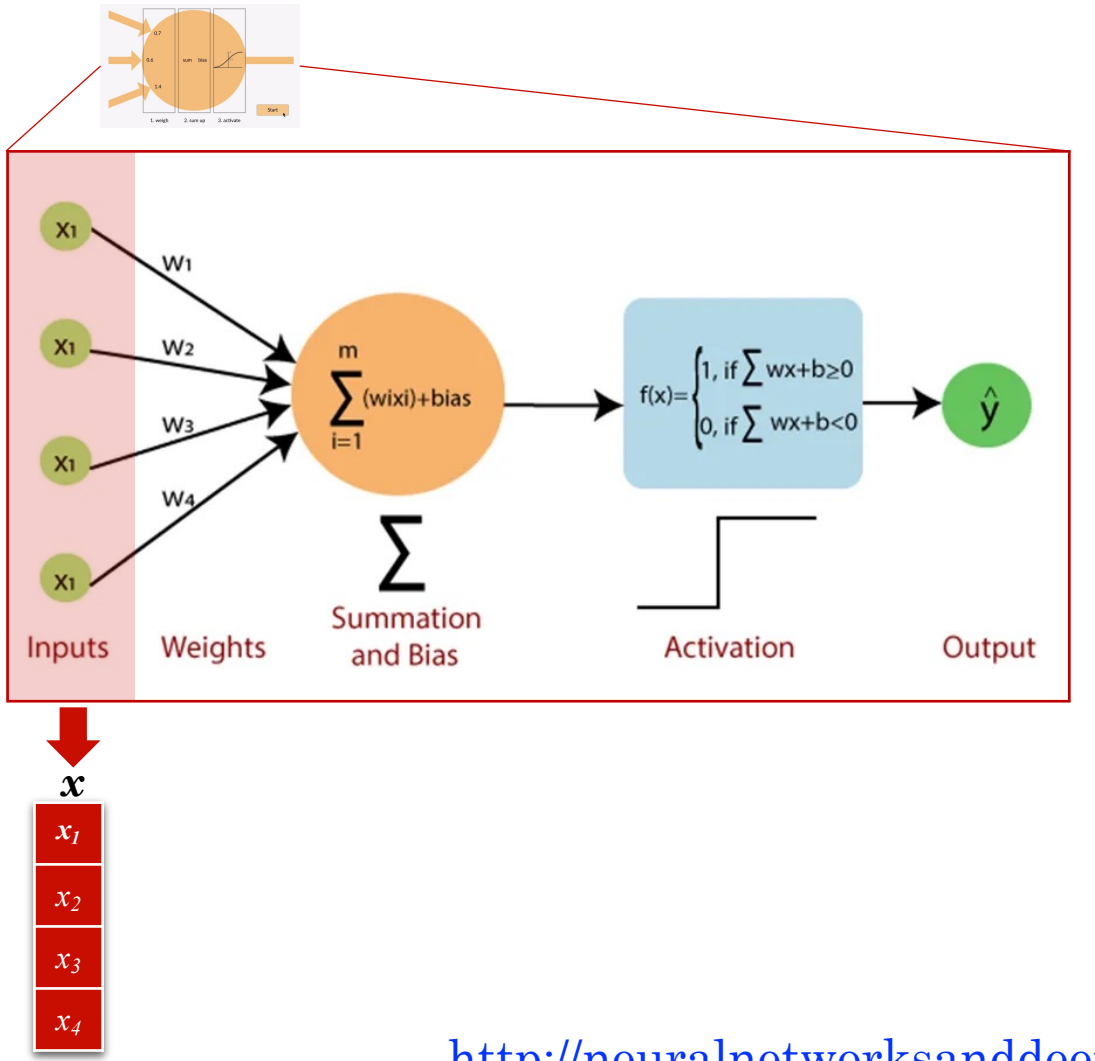


Fig: Biological Neuron





感知机（Perceptron）

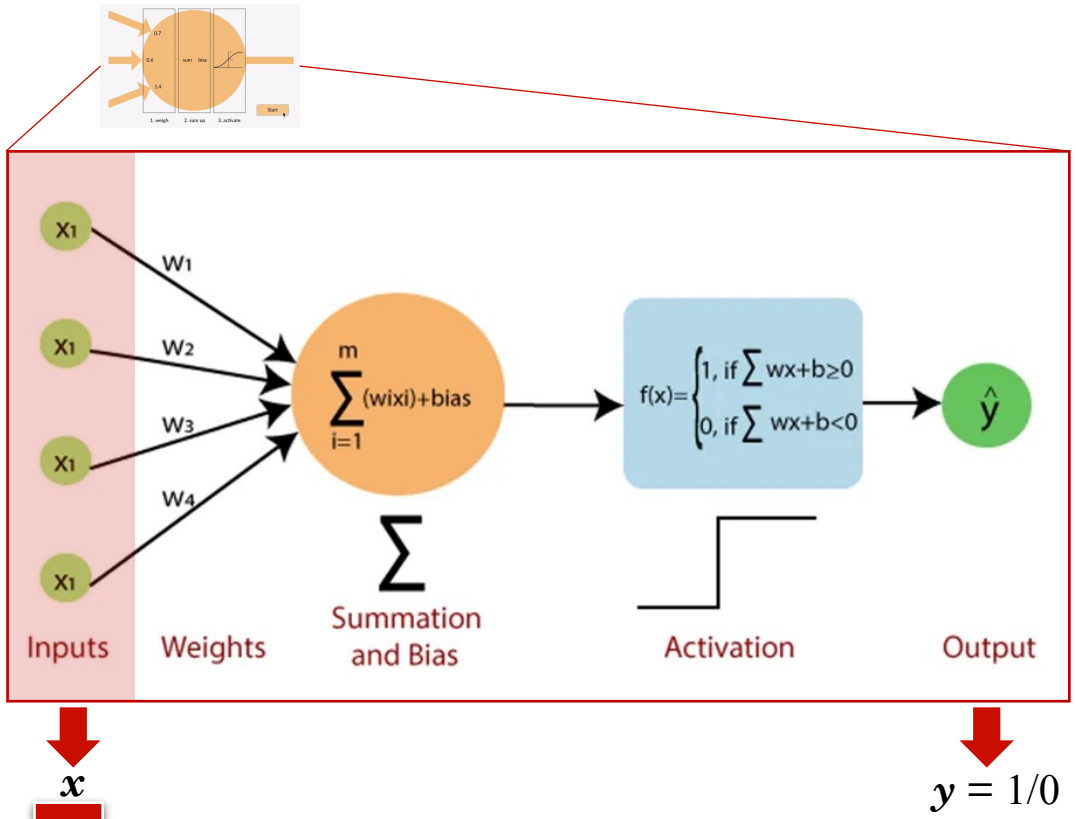


- 本质是一个**分类器**
- 输入：分类要素
  - 例如：分类是否要出游
    - x_1：天气是否晴天？ (1/0)
    - x_2：朋友是否有空？ (1/0)
    - x_3：论文是否录用？ (1/0)
    - x_4：兜里是否有钱？ (1/0)
  - 表示形式：4X1的矩阵（向量）

$$\boldsymbol{x} = [x_1, x_2, x_3, x_4]^{-1}$$

http://neuralnetworksanddeeplearning.com/index.html

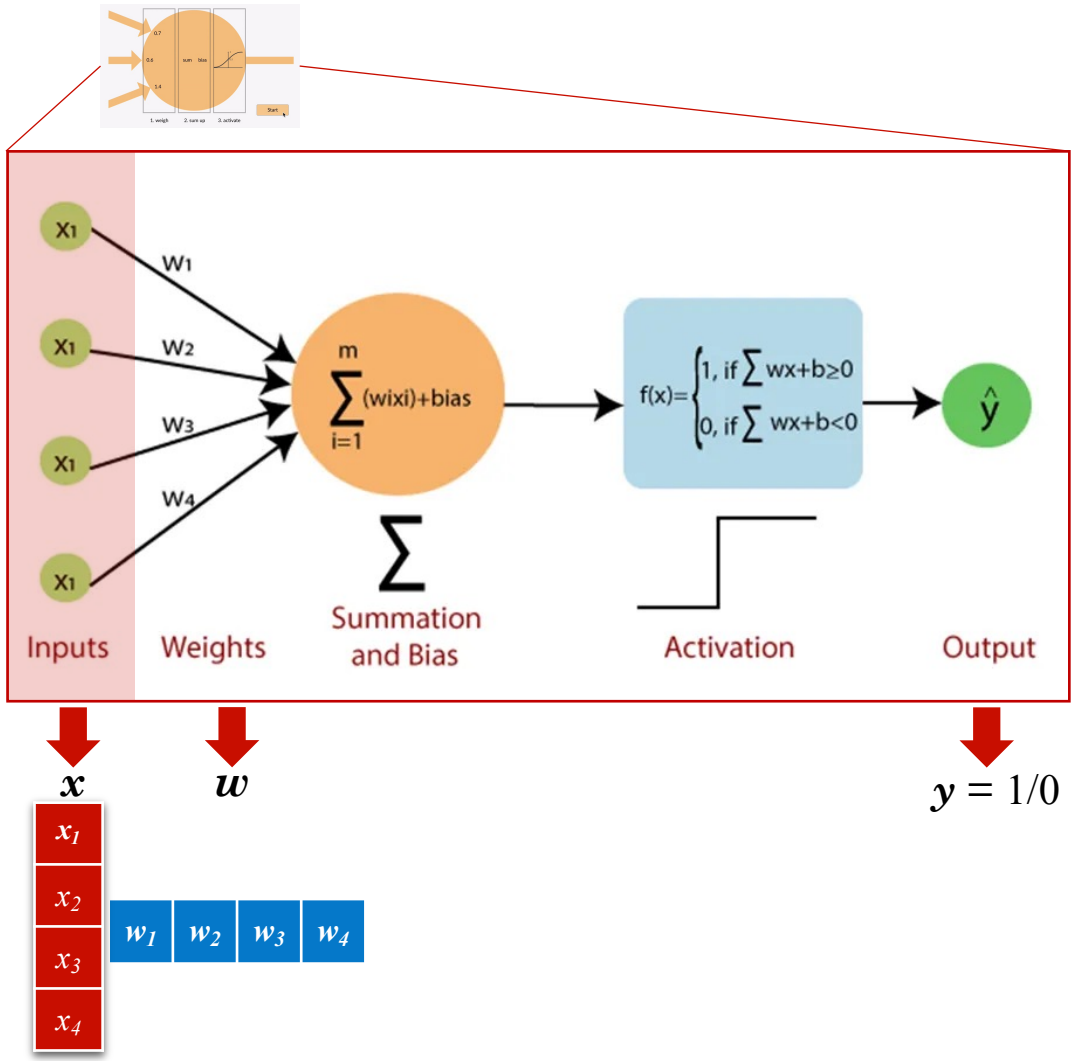
感知机（Perceptron）



- 本质是一个**分类器**
- 输出：分类标签
  - 例如：1/0
  - 表示形式：1个数字，标量

$$y = 1/0$$

感知机（Perceptron）

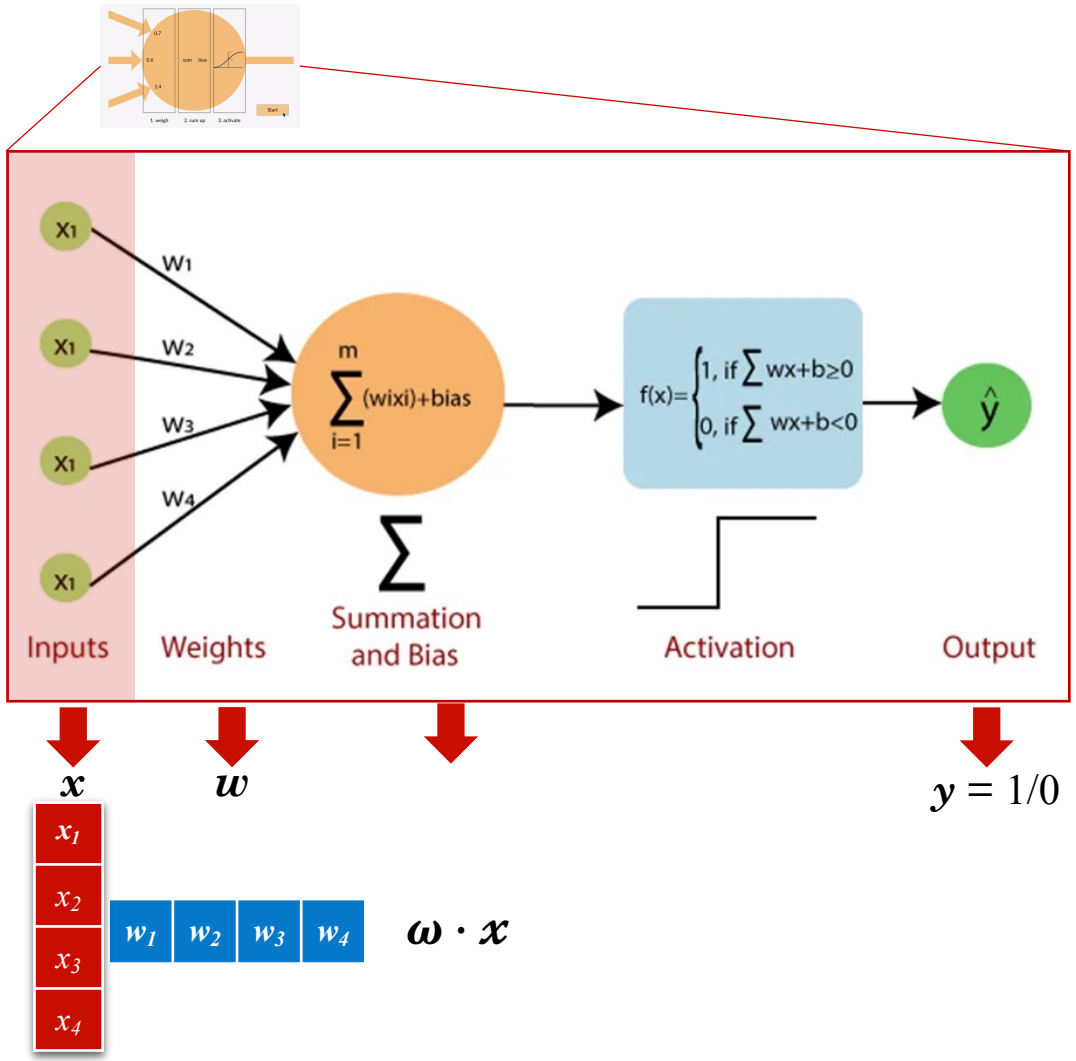


- 本质是一个**分类器**
- 决策过程：
  - 1 – 评估各要素权重
    - x_1：天气是否晴天？ w_1=0.1
    - x_2：朋友是否有空？ w_2=0.1
    - x_3：论文是否录用？ w_3=0.2
    - x_4：兜里是否有钱？ w_4=**0.7**

  - 表示形式：4X1的矩阵

$$\boldsymbol{w} = [w_1, w_2, w_3, w_4]$$

感知机（Perceptron）

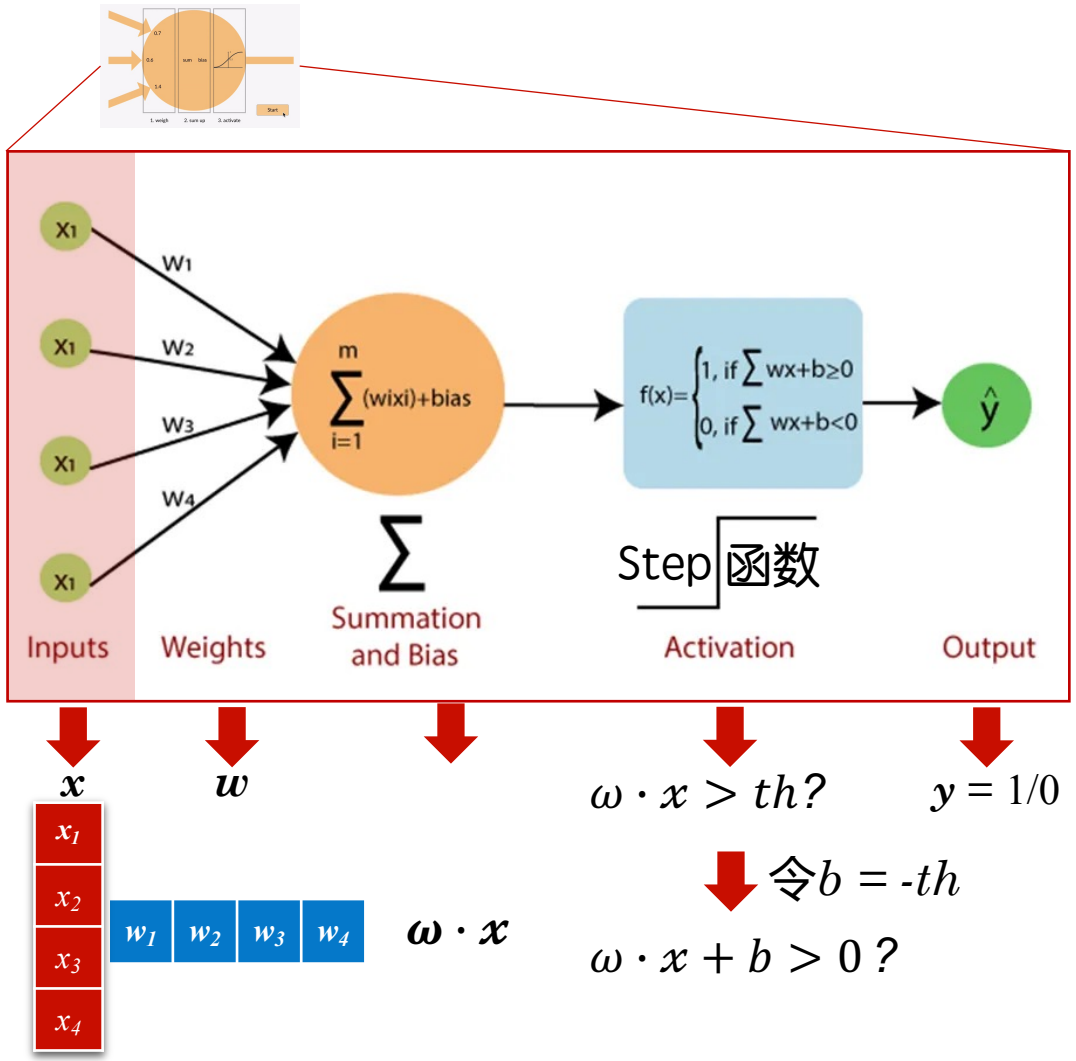


- 本质是一个**分类器**
- 决策过程：
  - 2 – 多要素综合权衡
    - x_1：天气是否晴天？ $1 \times w_1=0.1$
    - x_2：朋友是否有空？ $0 \times w_2=0.1$
    - x_3：论文是否录用？ $0 \times w_3=0.2$
    - x_4：兜里是否有钱？ $1 \times w_4=\mathbf{0.7}$

    $\sum = 0.8$
  - 表示形式：矩阵乘法

$$\omega \cdot x = \sum_{i=1}^{4} w_i x_i$$

感知机（Perceptron）

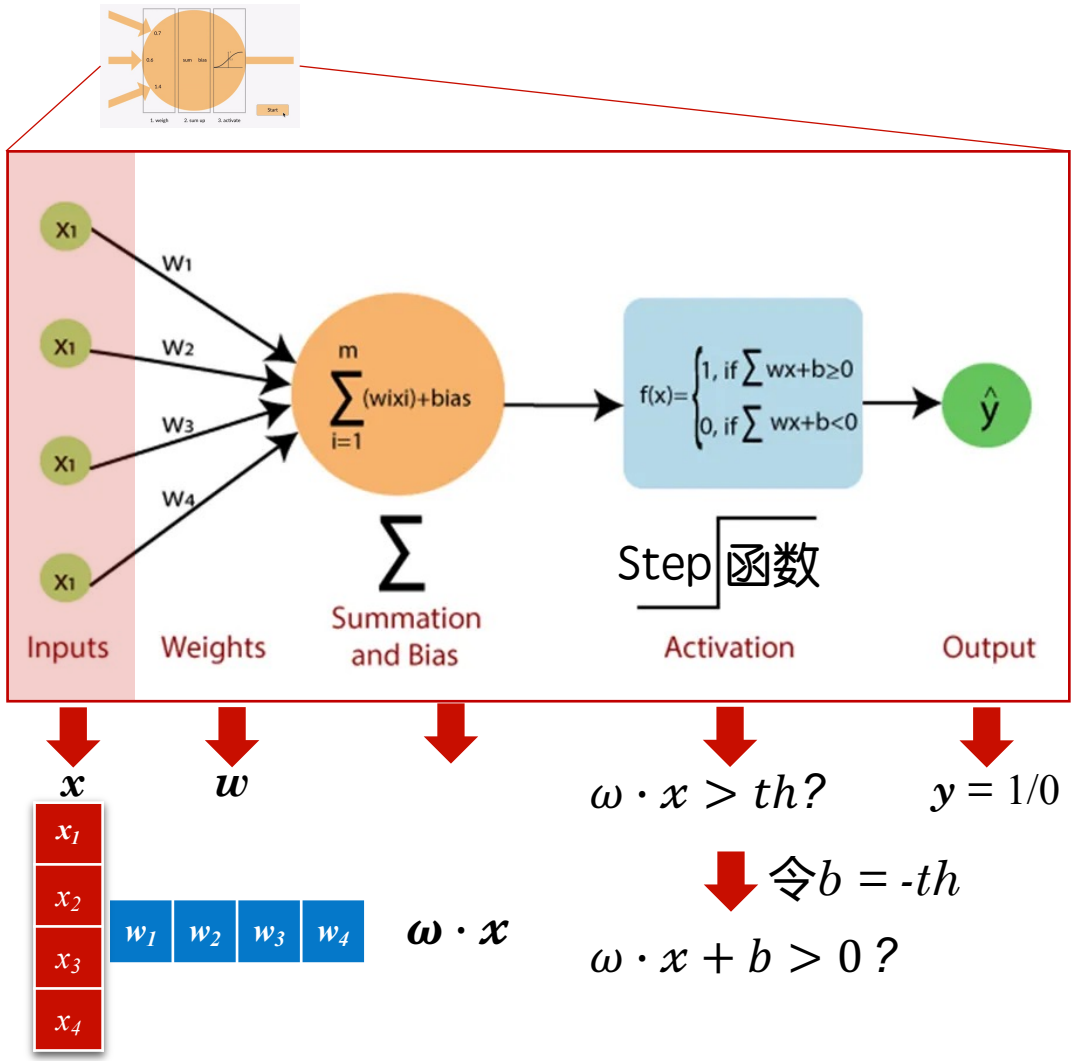


- 本质是一个**分类器**
- 决策过程：
  - 3 – 抉择（激活）
    - x_1：天气是否晴天？ $1 \times w_1$=0.1
    - x_2：朋友是否有空？ $0 \times w_2$=0.1
    - x_3：论文是否录用？ $0 \times w_3$=0.2
    - x_4：兜里是否有钱？ $1 \times w_4$=**0.7**

    $\sum = 0.8$
  - 假设心理阈值=0.6：走！

$$f(x) = \begin{cases} 1, & if\ (\omega \cdot x \geq 0.6) \\ 0, & if\ (\omega \cdot x < 0.6) \end{cases}$$

感知机（Perceptron）



- 本质是一个0/1**分类器**

$$\omega \cdot x + b > 0 \,?\, 1 : 0$$

*b*ias：生物神经元的“激活门禁”

大脑神经元传递为什么有个临界电位?

这是一个非常棒的生物物理学问题。我们可以把神经元的点火过程想象成一个**“带有门禁的开关”**。

大脑神经元之所以设定一个**临界电位**（Threshold Potential），本质上是为了实现**信息的精准过滤**和**信号的远程传输**。

1. 防止“噪音”干扰（信号过滤）

大脑内部始终存在着微小的电波动。如果神经元对任何微弱的刺激都有反应，大脑就会陷入永无止境的“尖叫”中，无法区分什么是重要的感官输入，什么是背景噪音。

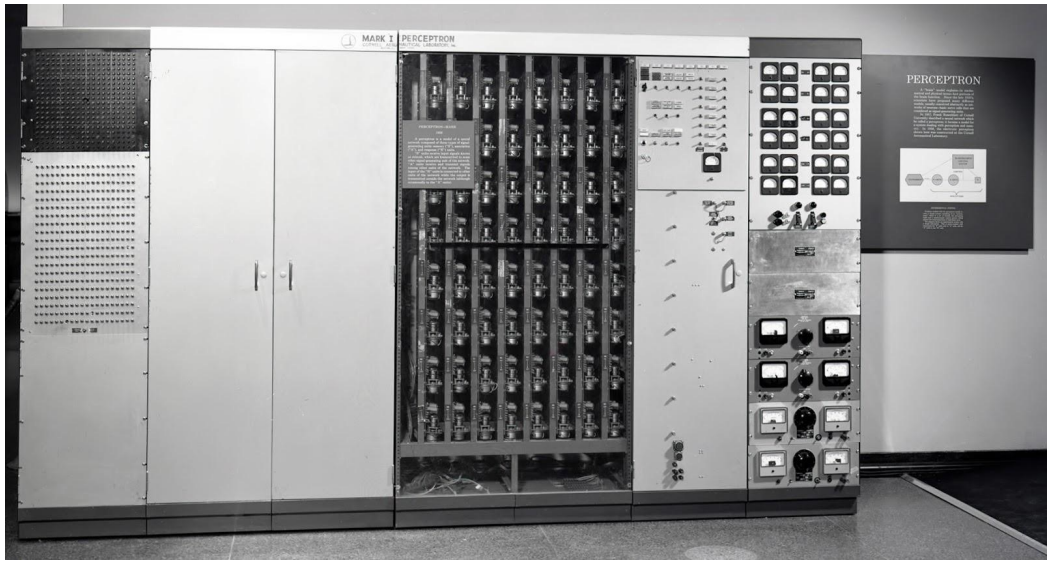
- **分级电位：** 神经元接收到的微弱刺激只会引起局部的电压波动，这种波动会随距离衰减。
- **全或无原则：** 只有当刺激足够强，使膜电位从静息状态（约 -70mV）上升到临界点（约 -55mV）时，神经元才会决定：“这是一个值得发送的信号！”

感知机的发展历程

Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN[1]

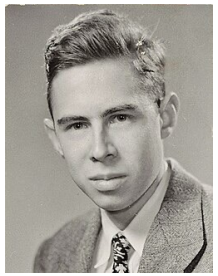
F. ROSENBLATT

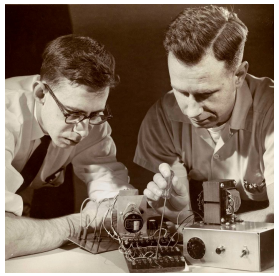


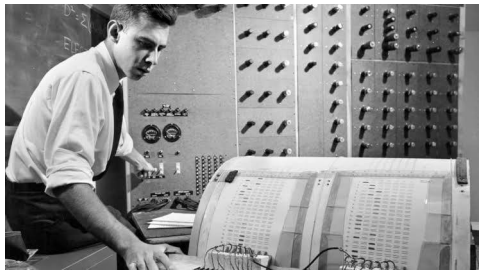
“首个类似人脑思维的机器”感知机：美国海军资助，400个光传感器模拟视网膜，连接约1000个神经元处理信息，输出单一结果

https://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf

Frank Rosenblatt，1957年感知器的创造者







早年生活和教育

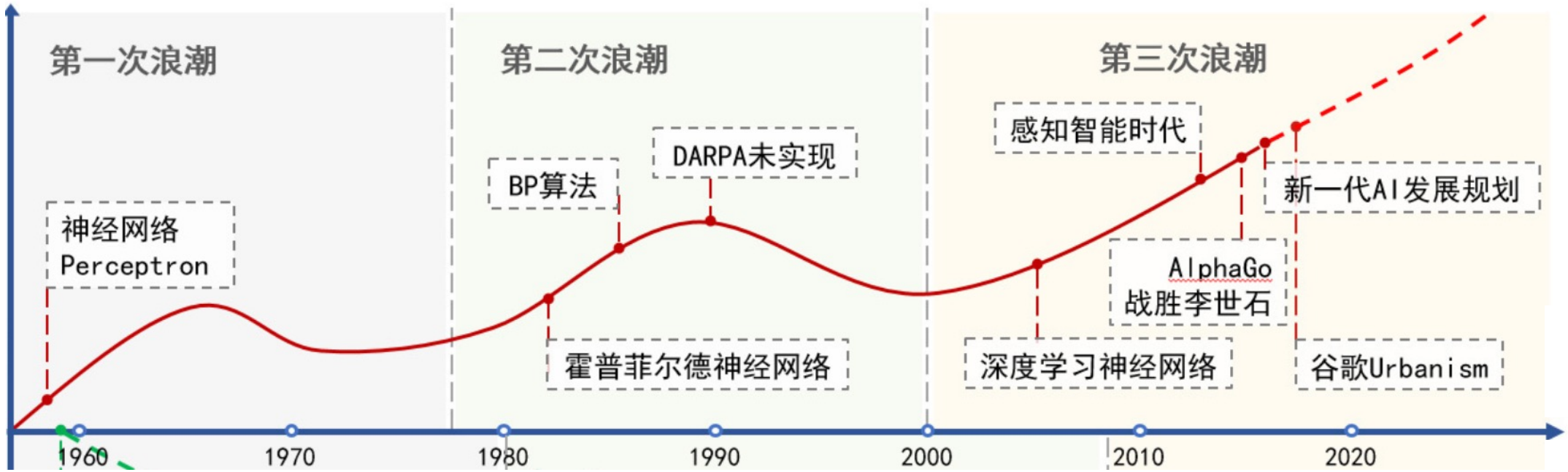
Frank Rosenblatt于1928年7月11日出生于美国纽约州新罗谢尔。他的父亲Samuel Rosenblatt是一名律师，他的母亲Helen Rosenblatt是一名家庭主妇。Rosenblatt的早年生活以对科学和数学的浓厚兴趣为标志，这受到父母的鼓励。

感知器模型的开发

感知器模型由Frank Rosenblatt在20世纪50年代首次引入，是一个单层神经网络，旨在将输入分为兩類之一。该模型由输入层、处理层和输出层组成，它们之间的连接根据预测和实际输出之间的误差进行调整。

Frank Rosenblatt在人工智能（AI）研究方面的开创性工作对人工智能系统的发展产生了持久的影响。他引入了多层感知器，这是一种前馈神经网络，为现代机器学习算法奠定了基础。尽管他不幸去世，年仅41岁，但他的工作继续影响着今天的人工智能研究，许多现代深度学习模型借鉴了他半个多世纪前引入的概念。Rosenblatt的

感知机一定程度上引发了AI第一次浪潮



国防高级研究项目局（DARPA）等在20世纪60年代为人工智能研究项目提供了巨额资金

单个感知机的致命缺陷：异或问题

$$y(x1, x2) = f(\omega1 * x1 + \omega2 * x2 - b)$$

逻辑运算	w_1	w_2	b	验证
与（$x_1 \wedge x_2$）	1	1	1.5	1*1+1*1-1.5>0 1*0+1*1-1.5<0
或（$x_1 \vee x_2$）	1	1	0.5	1*0+1*1-0.5>0 1*0+1*0-0.5<0
非（$\sim x_1$）	-1	0	-0.5	-1*0+0*1/0+0.5>0 -1*1+0*1/0+0.5<0

单个感知机的致命缺陷：异或问题

$$y(x1, x2) = f(\omega 1 * x1 + \omega 2 * x2 - b)$$ 1969年，Minsky

逻辑运算	x_1	x_2	y
异或（XOR）	1	1	0
	1	0	1
	0	1	1
	0	0	0

11: $w_1 + w_2 - b < 0$ → $b > w_1 + w_2$

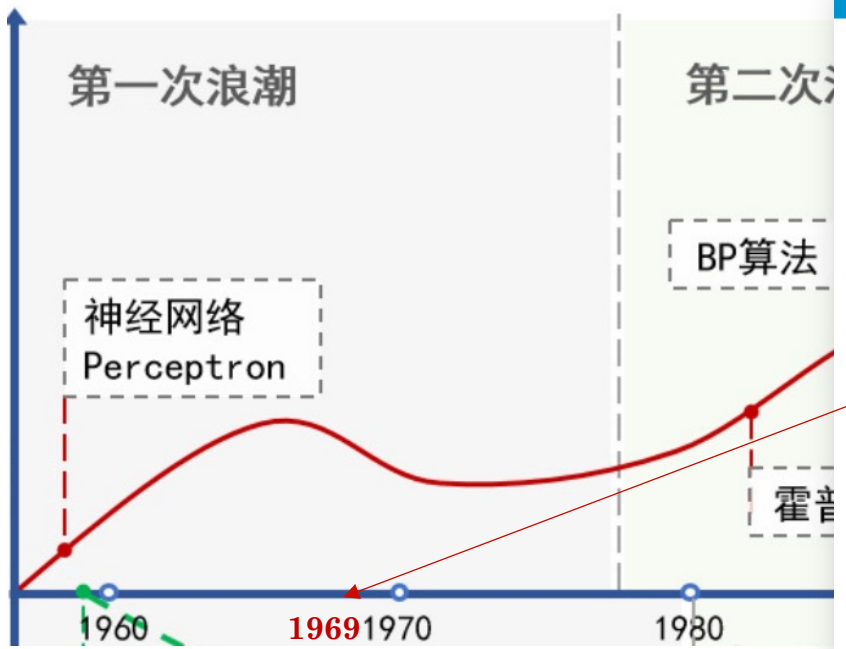
10: $w_1 + 0 - b \geq 0$ → $b \leq w_1$

01: $0 + w_2 - b \geq 0$ → $b \leq w_2$

00: $0 + 0 - b < 0$ → $b > 0$

矛盾，无解！

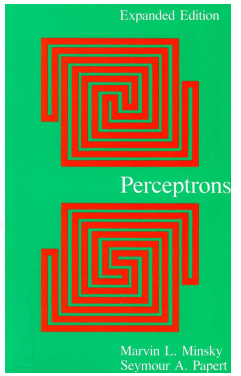
XOR缺陷导致了AI第一次寒冬?



Minsky和Papert的《*感知电子子*》一书抨击了Frank Rosenblatt关于感知电子的工作，并成为人工神经网络分析的基础作品。這本書是人工智慧史上爭議的中心，有人声称它大大阻礙了20世紀70年代的神經網路研究，並促成了所謂的「人工智慧冬天」。

https://amturing.acm.org/award_winners/minsky_7440781.cfm

XOR缺陷导致了AI第一次寒冬?

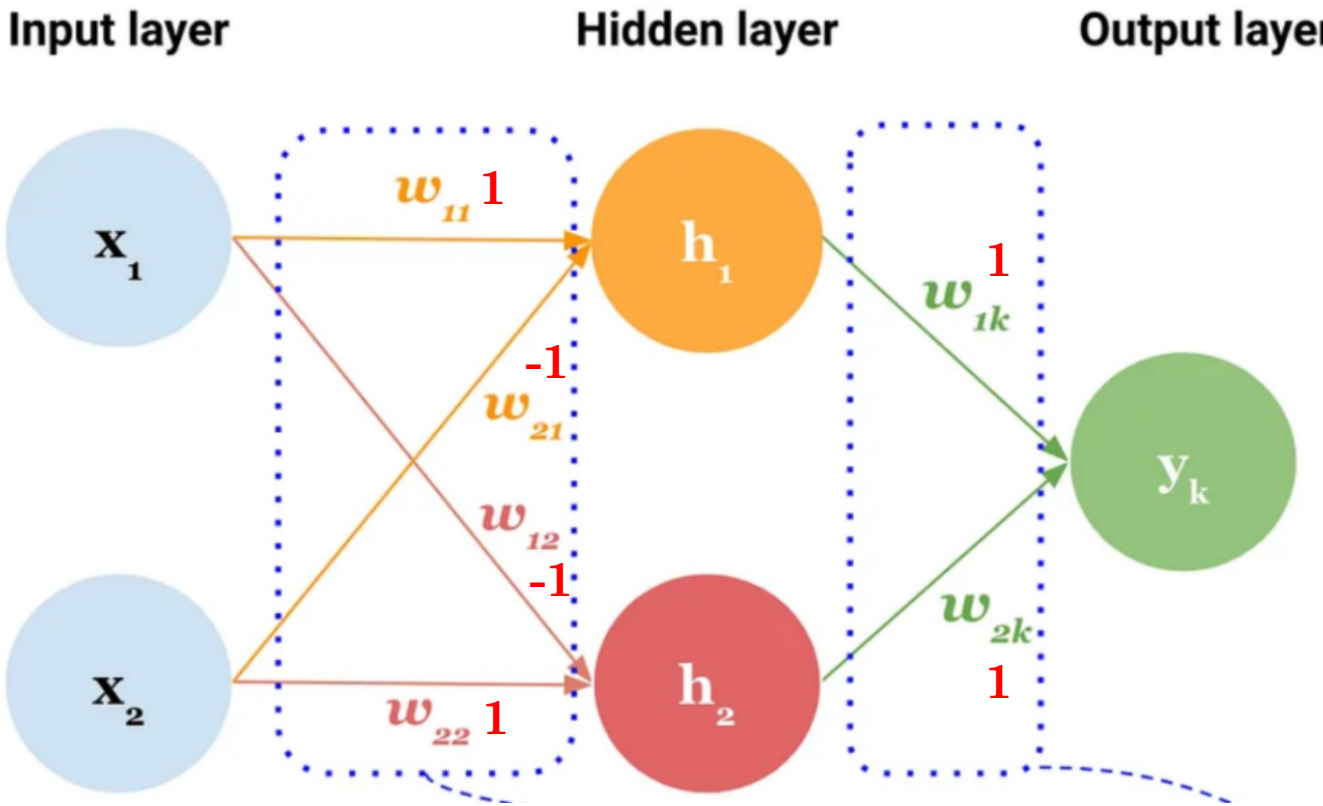


XOR事件 [编辑] https://rodsmith.nz/wp-content/uploads/Minsky-and-Papert-Perceptrons.pdf

这本书的一些评论家[*需要引用*]指出，作者暗示，由于单个人工神经元无法实现一些功能，如XOR逻辑功能，因此更大的网络也有类似的限制，因此应该放弃。对三层感知器的研究展示了如何实现此类功能。Rosenblatt在他的书中证明，具有*先验*无限数量的隐藏层A元素（神经元）和一个输出神经元的"元素感知器"可以解决任何分类问题。（存在定理。[19]）Minsky和Papert使用了具有限制隐藏层A元素输入数量和局部条件的感知器：隐藏层的每个元素接收来自小圆圈的输入信号。这些受限的感知器无法定义图像是连通的图形还是图像中的像素数偶数（奇偶校验谓词）。

这个故事有很多错误[*需要引用*]。尽管一个神经元实际上只能计算少量的逻辑谓词，但它是众所周知的[*需要引用*]此类元素的网络可以计算任何可能的布尔函数。沃伦·麦卡洛赫和沃尔特·皮茨都知道这一点，他们甚至提出了如何用他们的正式神经元创建图灵机（第三节[20]），在罗森布拉特的书中提到，在1961年的一篇典型论文中提到（图15[21]），甚至在《感知器》一书中被提及。[22]Minsky在他的《*计算：有限和无限机器*》一书的第3章中，还广泛使用形式神经元来创建简单的理论计算机。

多层感知器 (Multilayer Perceptron ， MLP)

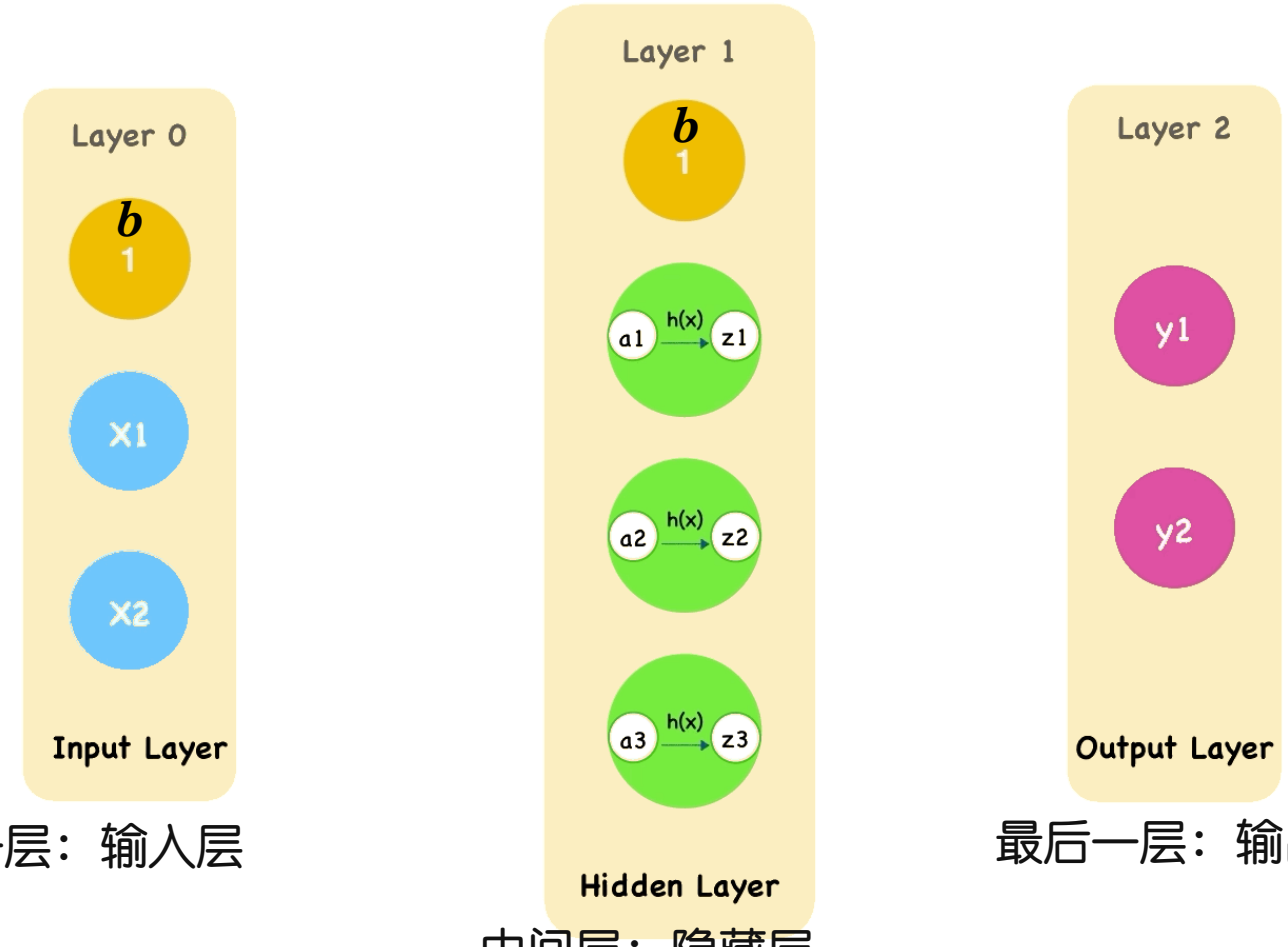


所有b=0.5

x_1	x_2	h_1	h_2	y
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

多层感知器 (Multilayer Perceptron ， MLP)

注意：模型层数指权重层数（边的层数），而神经元不带权重



第一层：输入层

中间层：隐藏层

最后一层：输出层

❑ 多层感知机

- 多个神经元构成一“层”(layer)
- 神经元的输出依然只有一个，但可发送给多个后层神经元
- 信号发送是从前到后单向的，因此MLP也称为“前馈神经网络 (Feed-Forward Neural Network)”
- 工作过程：信号从输入层向后逐层计算，最终从最后一层得到分类结果

多层感知器 (Multilayer Perceptron，MLP)

- 代码实现（代码加注释）

```
import numpy as np

class MLP:
    def __init__(self, input_size, hidden_size, output_size):
        """input_size:输入层节点数, hidden_size:隐藏层节点数, output_size:输出层节点数"""
        self.W1 = np.random.randn(input_size, hidden_size) # 输入层 -> 隐藏层 权重矩阵
        self.b1 = np.zeros((1, hidden_size)) # 隐藏层偏置
        self.W2 = np.random.randn(hidden_size, output_size) # 隐藏层 -> 输出层 权重矩阵
        self.b2 = np.zeros((1, output_size)) # 输出层偏置

    def forward(self, X):
        """ 前向传播过程"""
        self.z1 = np.dot(X, self.W1) + self.b1 # 输入层 -> 隐藏层
        self.z2 = np.dot(self.z1, self.W2) + self.b2 # 隐藏层 -> 输出层
        return self.z2
```

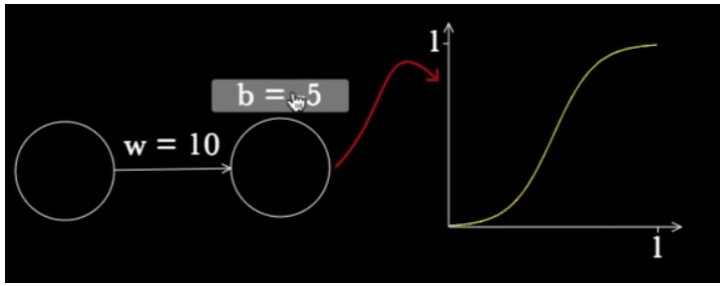
通用近似定理（Universal approximation theorem）

Approximation by superpositions of a sigmoidal function

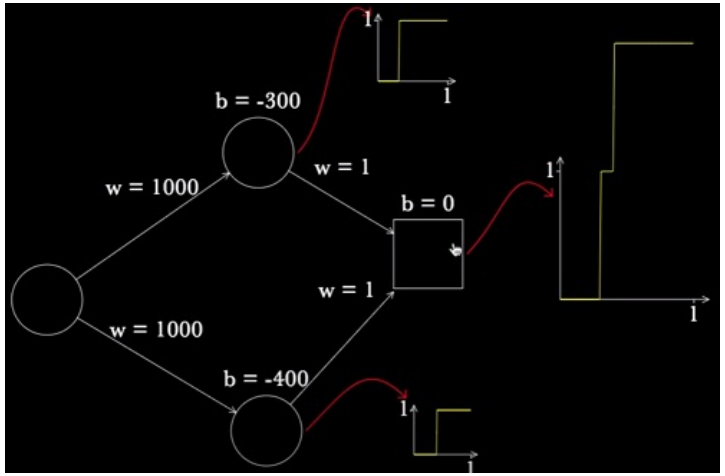
G Cybenko - Mathematics of control, signals and systems, 1989 - Springer

… Theorem 2, we can find a summation of the form of G … approximation theory and statistics. Some recent progress concerned with the relationship between a function being approximated …

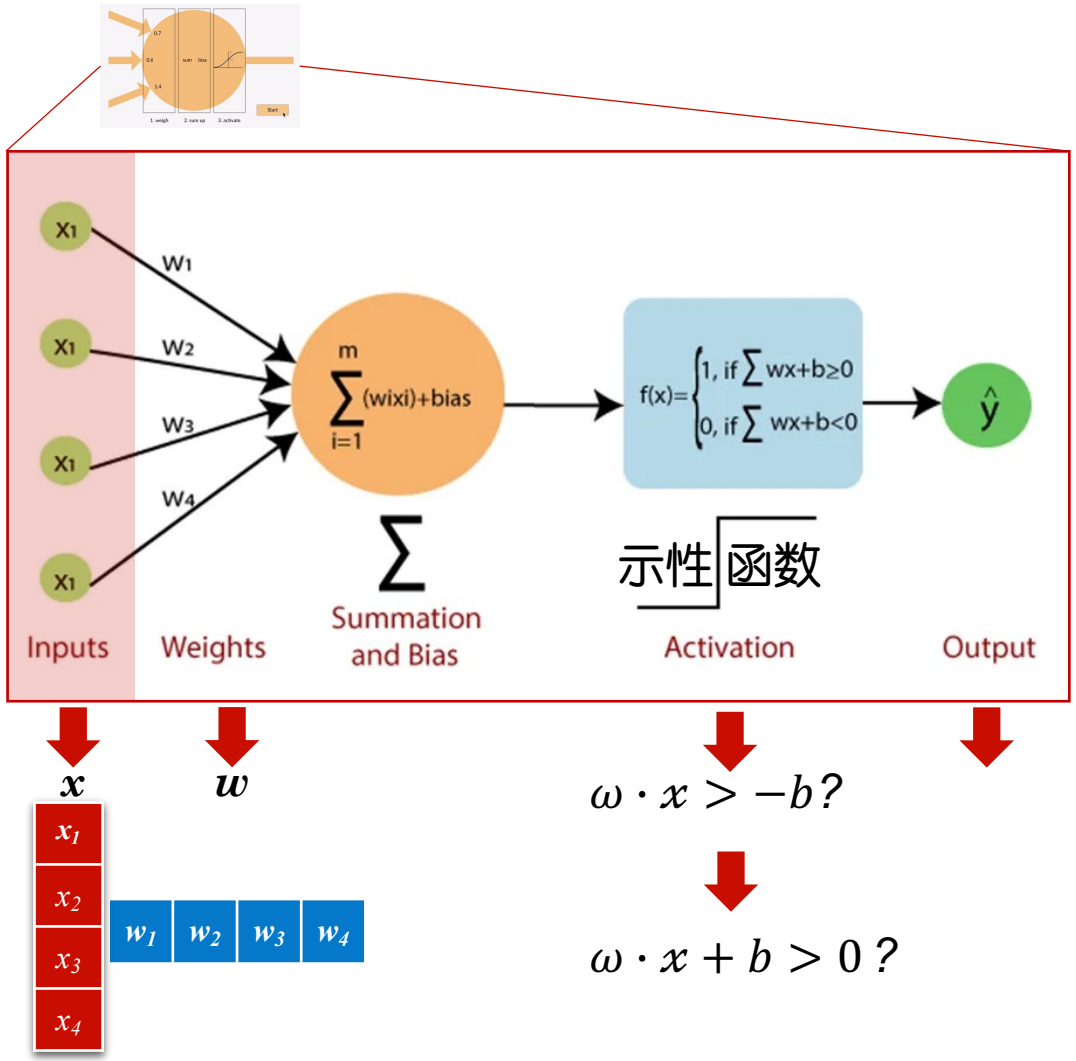
☆ 保存 99 引用 被引用次数：23089 相关文章 所有 19 个版本



通用近似定理是神经网络的理论基础。它证明了一个具有包含足够多但有限数量神经元隐藏层的MLP，结合某些激活函数，可以合理的精度近似任何连续函数



然而……



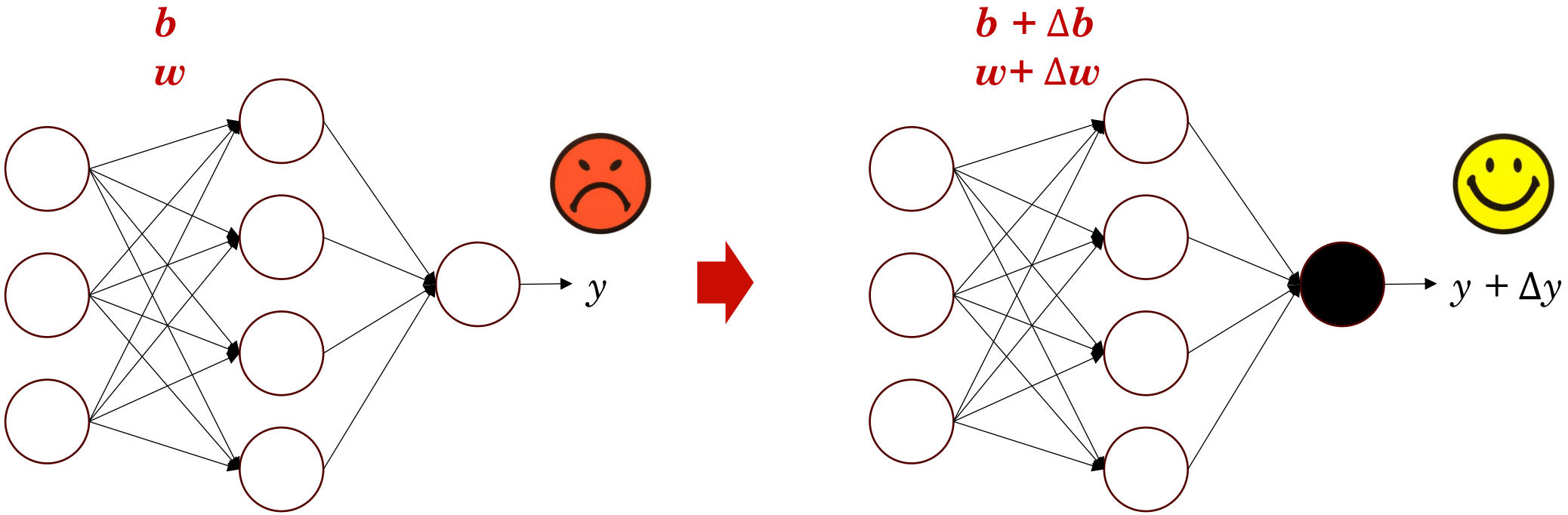
ω, b怎么设置?

中国科学院大学
University of Chinese Academy of Sciences

目 录

试错法

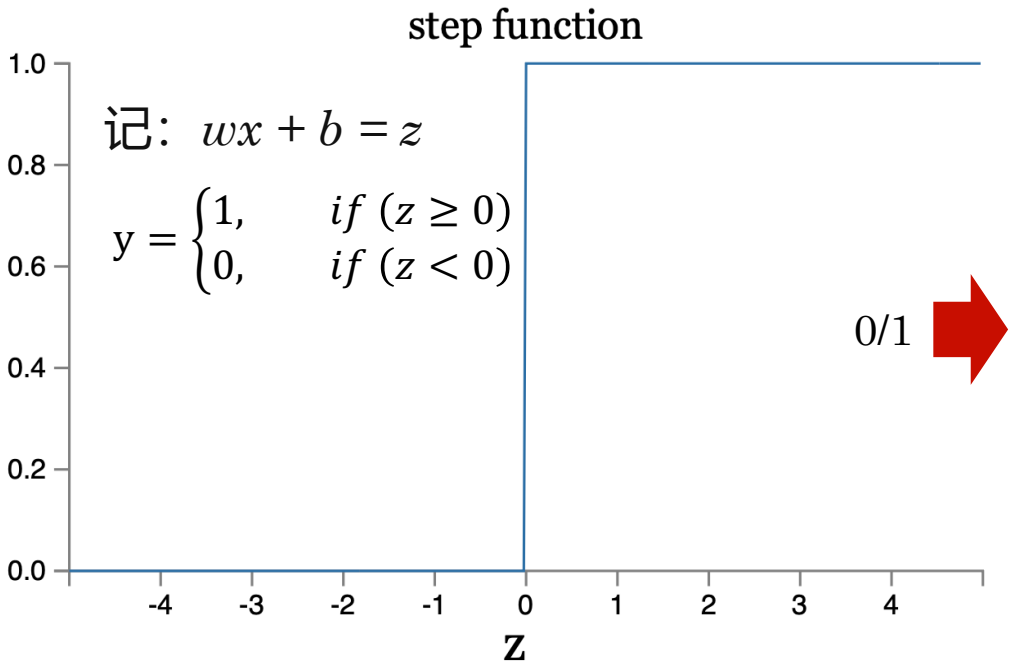
- 对w、b作微小变动，看y对不对？不对就总结规律，继续改进！



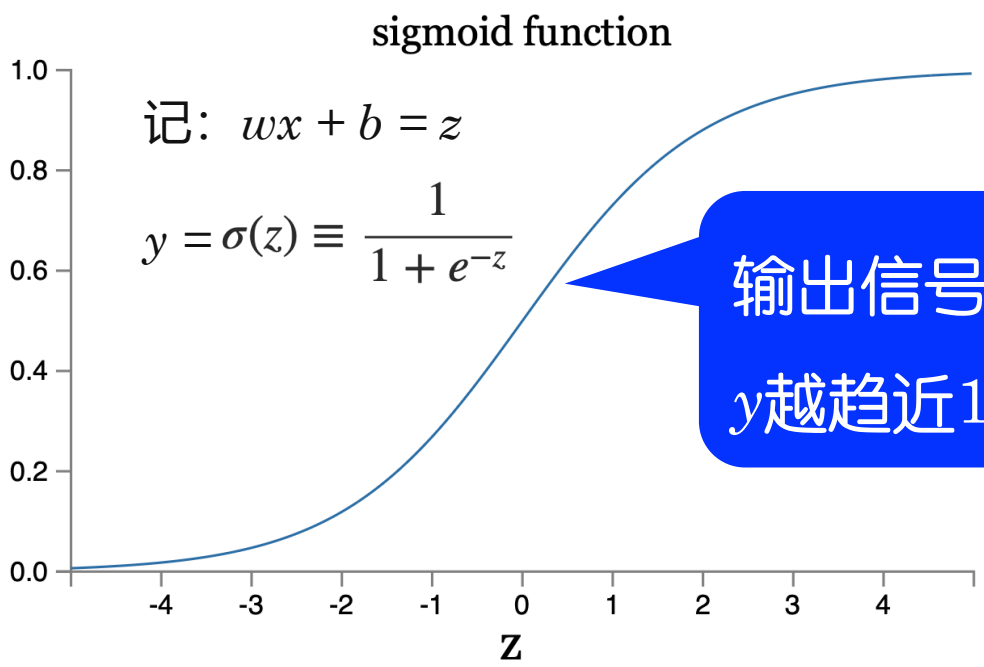
http://neuralnetworksanddeeplearning.com/chap2.html

y走向成熟：从0/1到(0,1)

- 对w、b作微小变动，但y的变动太猛了（非黑即白），不利于总结规律



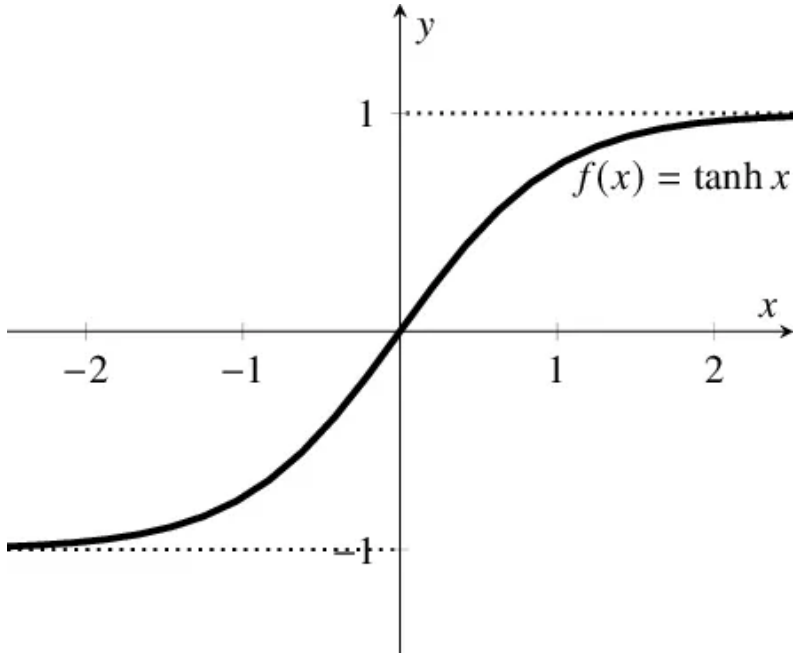
0/1 ➡ (0,1)



输出信号z越大，y越趋近1

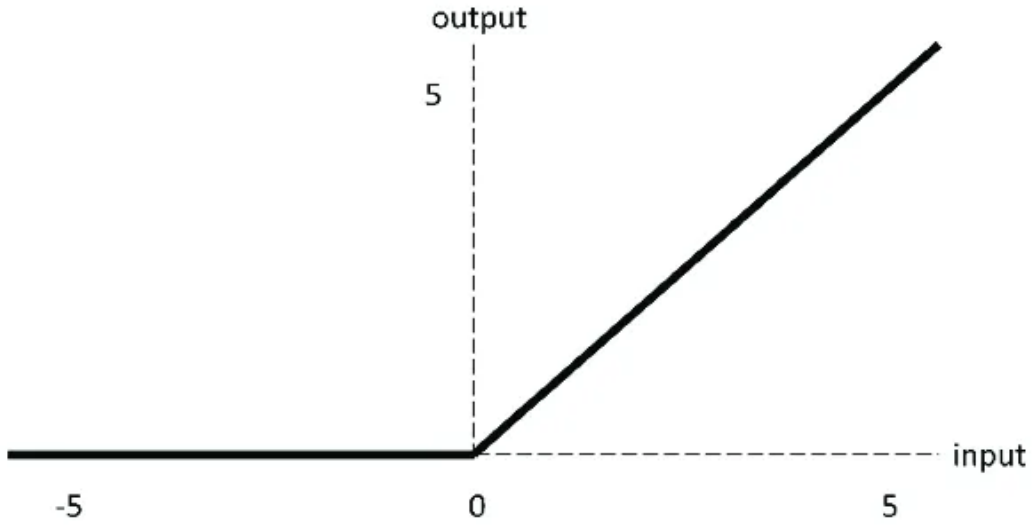
其他常见激活函数

❑ tanh（双曲正切函数）



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

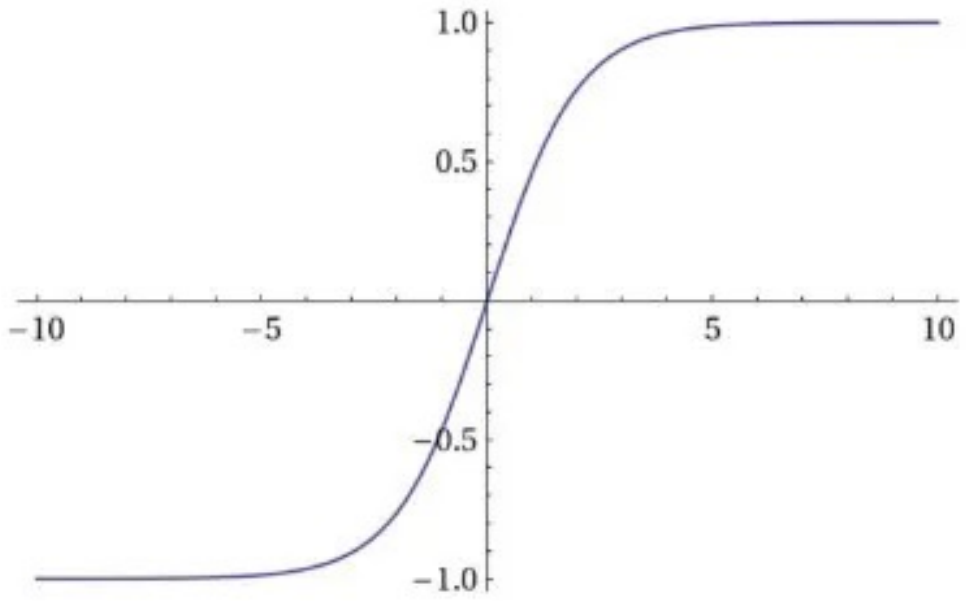
❑ ReLU（修正线性单元）

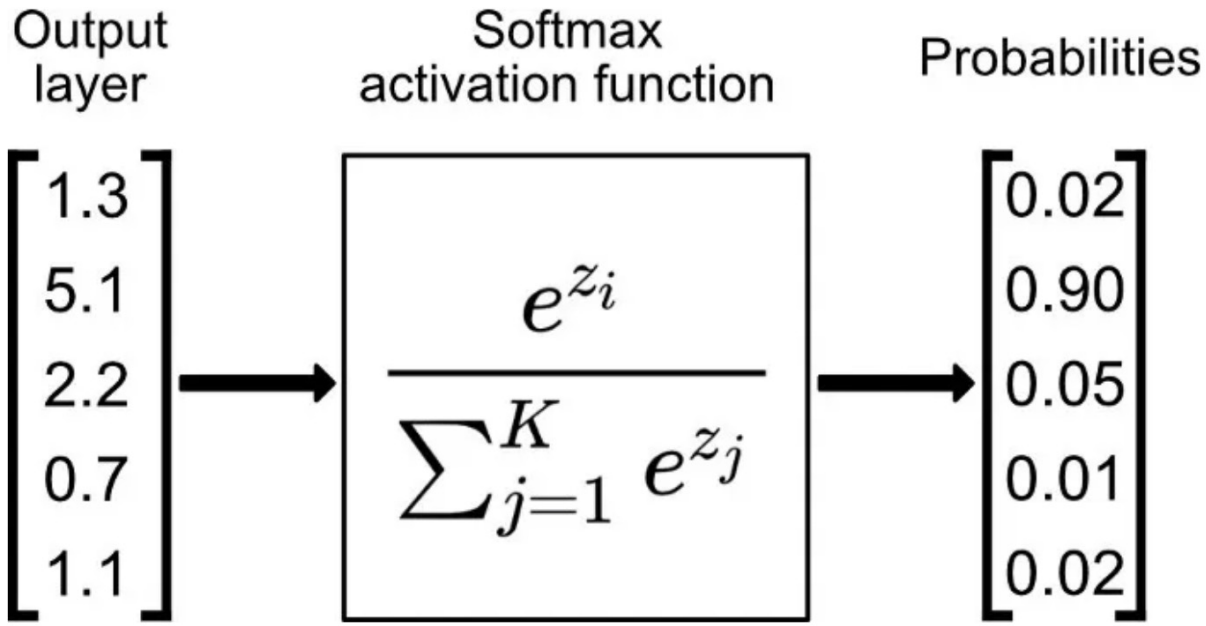


$$\sigma(x) = \begin{cases} max(0,x) & ,x >= 0 \\ 0 & ,x < 0 \end{cases}$$

其他常见激活函数

- Softmax





预测$\hat{y}$与真实y的差距：损失（也称成本）

$\hat{y}=f(x)$ 为预测值

- 单样本

$$L = |y_i - \hat{y}_i|$$

- 多样本
  - 平均绝对误差 (MAE)

$$L = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  - 均方误差 (MSE)

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- 向量型多样本
  - 交叉熵

$$L = -\sum_{i=1}^{n} y_i \cdot log(\hat{y}_i)$$

训练

- 寻找一组w,b参数，使得在整个训练集上L最小

$$argmin_{w,b}L$$

梯度下降（gradient descent）

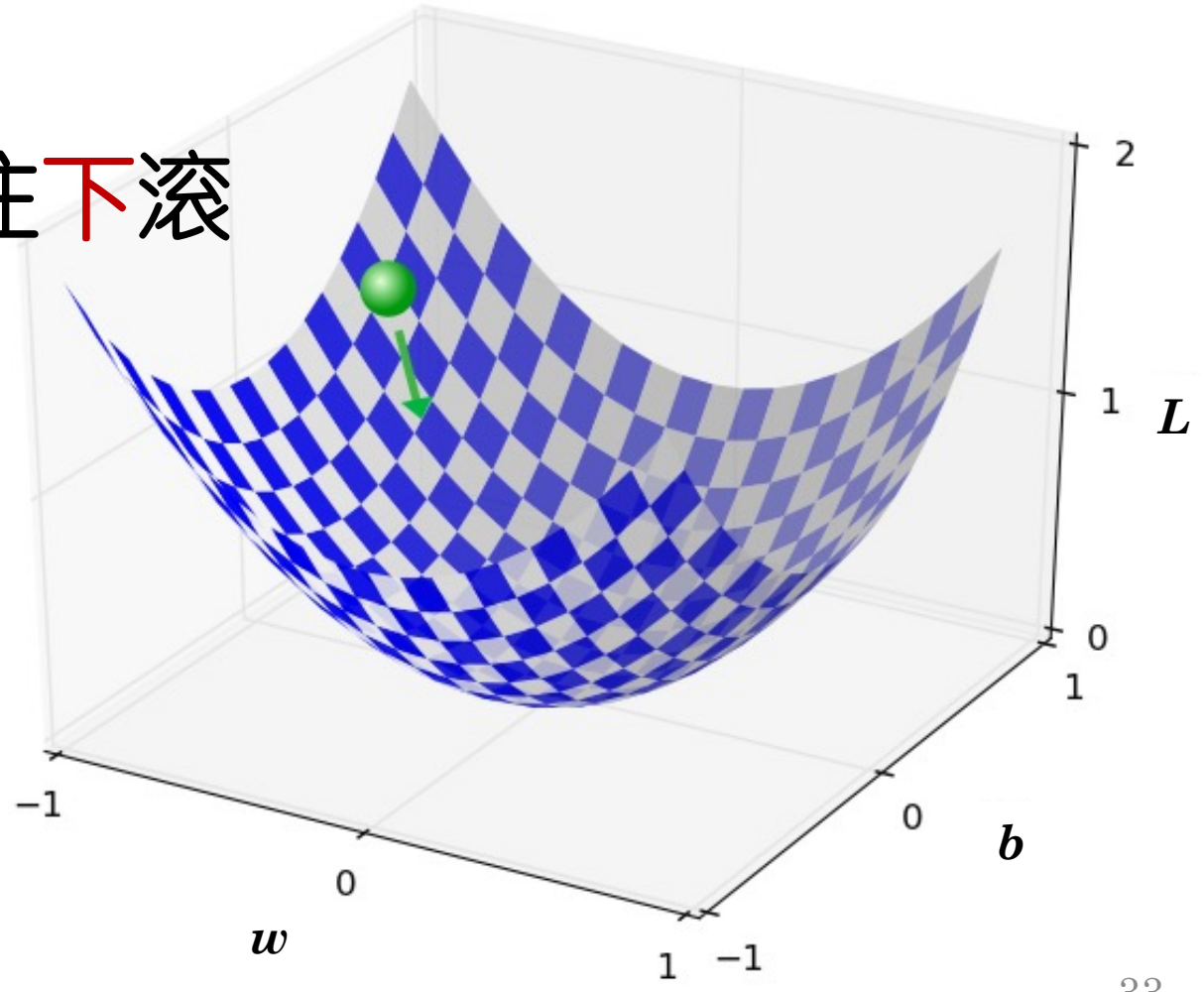
问：如何达到最低点?

答：不管身处何处，一直往下滚

问：哪儿是下?

答：导的反方向

$$-\left(\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right)$$



梯度下降（gradient descent）

证明：记滚动距离ΔL

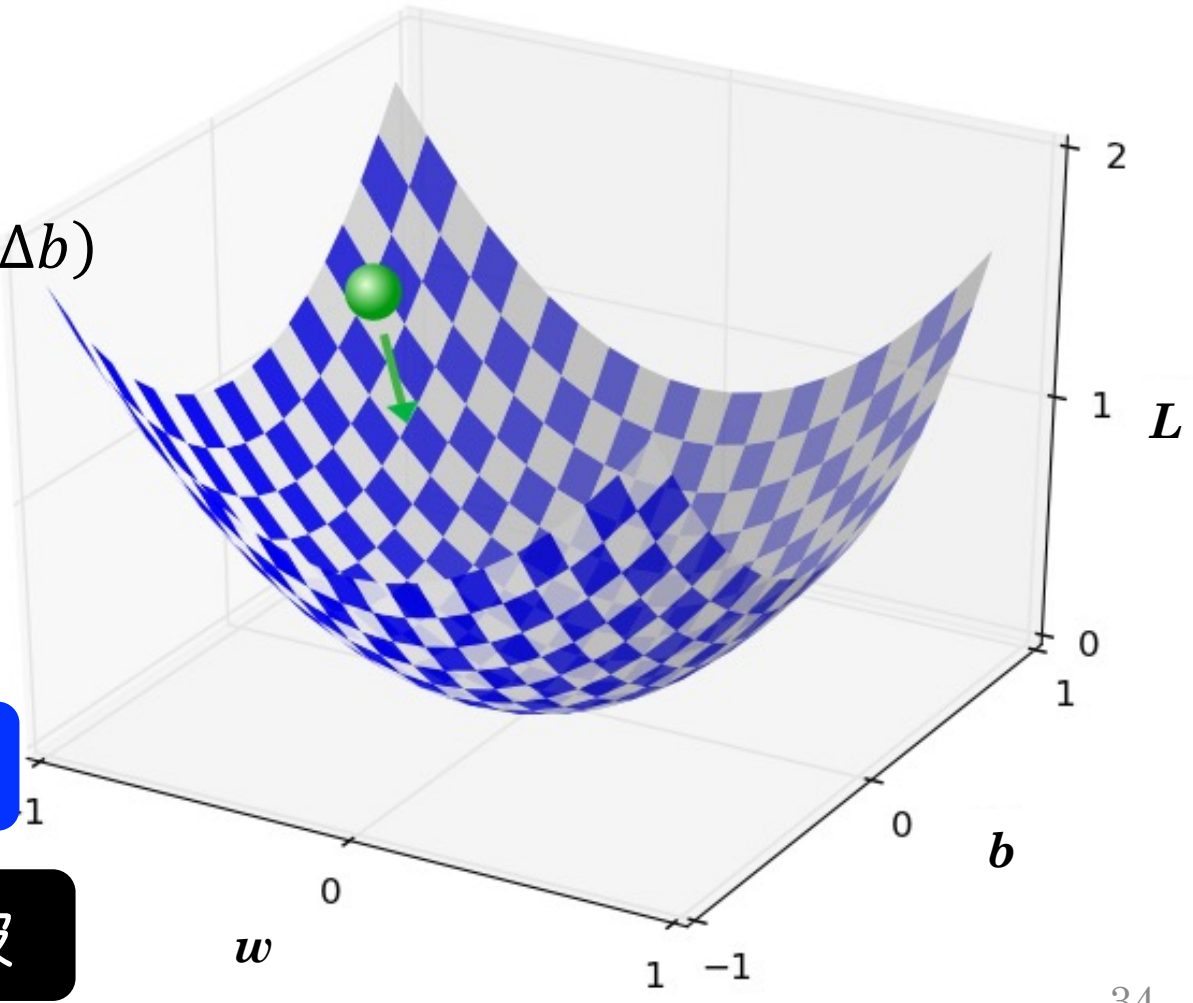
$$\Delta L = \frac{\partial L}{\partial w}\Delta w + \frac{\partial L}{\partial b}\Delta b = \left(\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right)^T (\Delta w, \Delta b)$$

记$\left(\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right)^T = \nabla L$， $\Delta\theta = (\Delta w, \Delta b)$

则$\Delta L = \nabla L \Delta\theta$ 导方向

若$\Delta\theta = -\eta\nabla L\ (\eta > 0)$ 走导反方向

则$\Delta L = -\eta(\nabla L)^2$， < 0 就一定下坡



梯度下降（gradient descent）

证明：记滚动距离ΔL

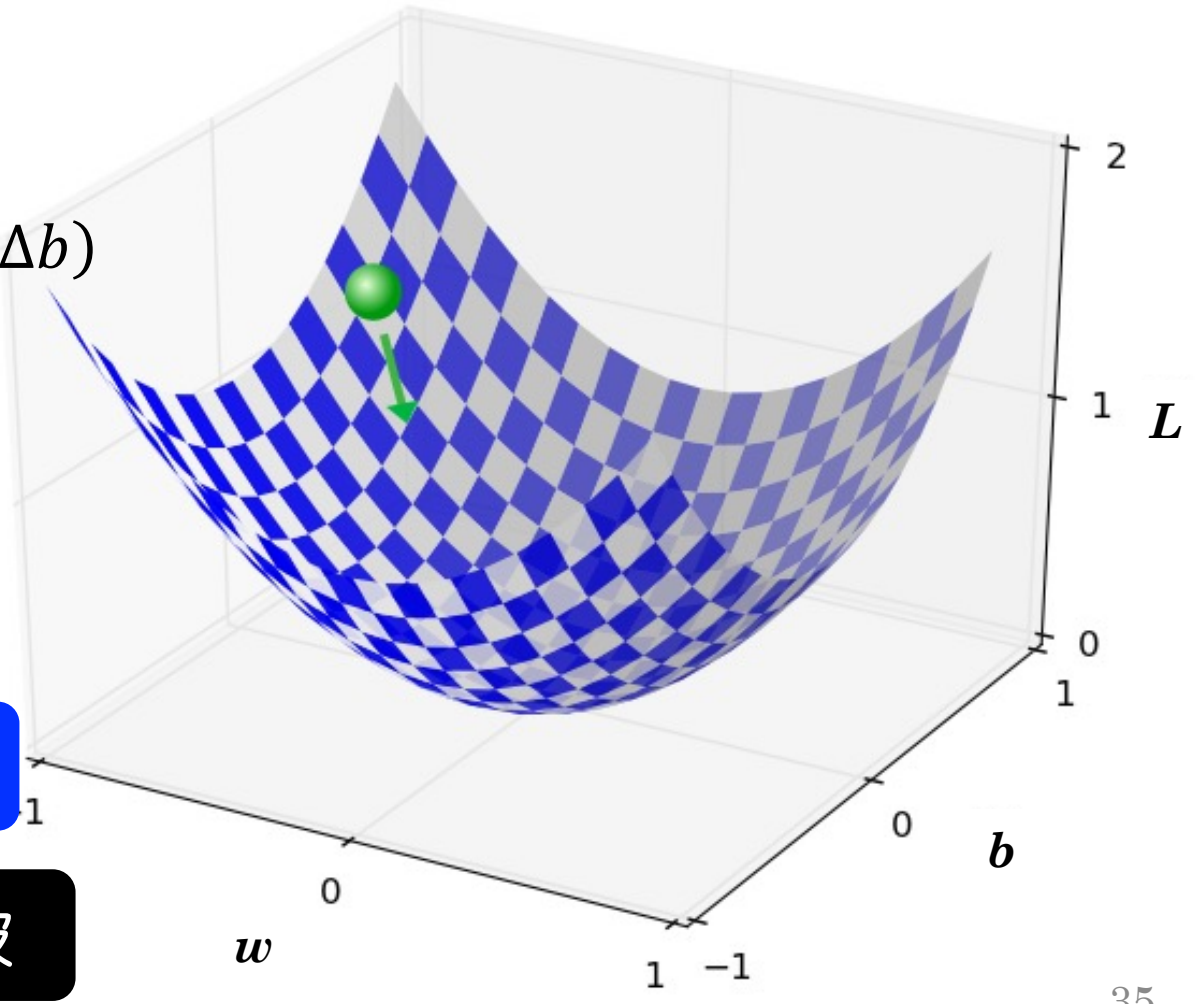
$$\Delta L = \frac{\partial L}{\partial w}\Delta w + \frac{\partial L}{\partial b}\Delta b = \left(\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right)^T (\Delta w, \Delta b)$$

记$\left(\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right)^T = \nabla L$， $\Delta\theta = (\Delta w, \Delta b)$

则$\Delta L = \nabla L \Delta\theta$ 导方向

若$\Delta\theta = -\eta\nabla L\ (\eta > 0)$ 走导反方向

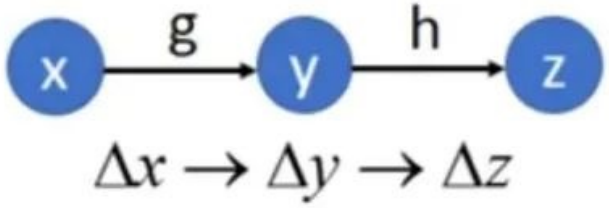
则$\Delta L = -\eta(\nabla L)^2$， < 0 就一定下坡



求导的链式法则

Case 1

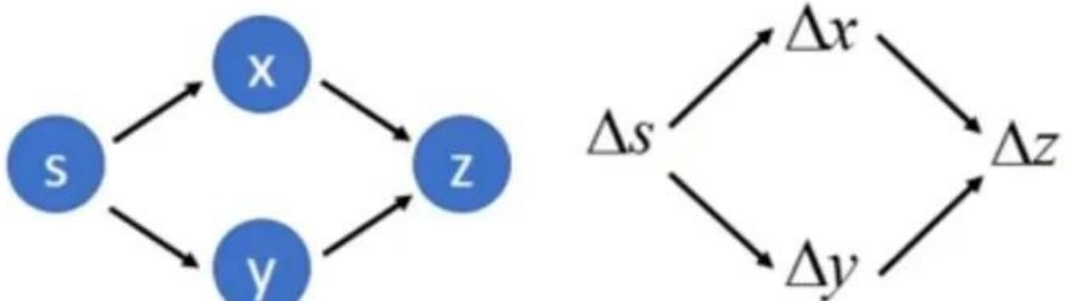
$$z = f(x) \Rightarrow y = g(x) \quad z = h(y)$$



$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

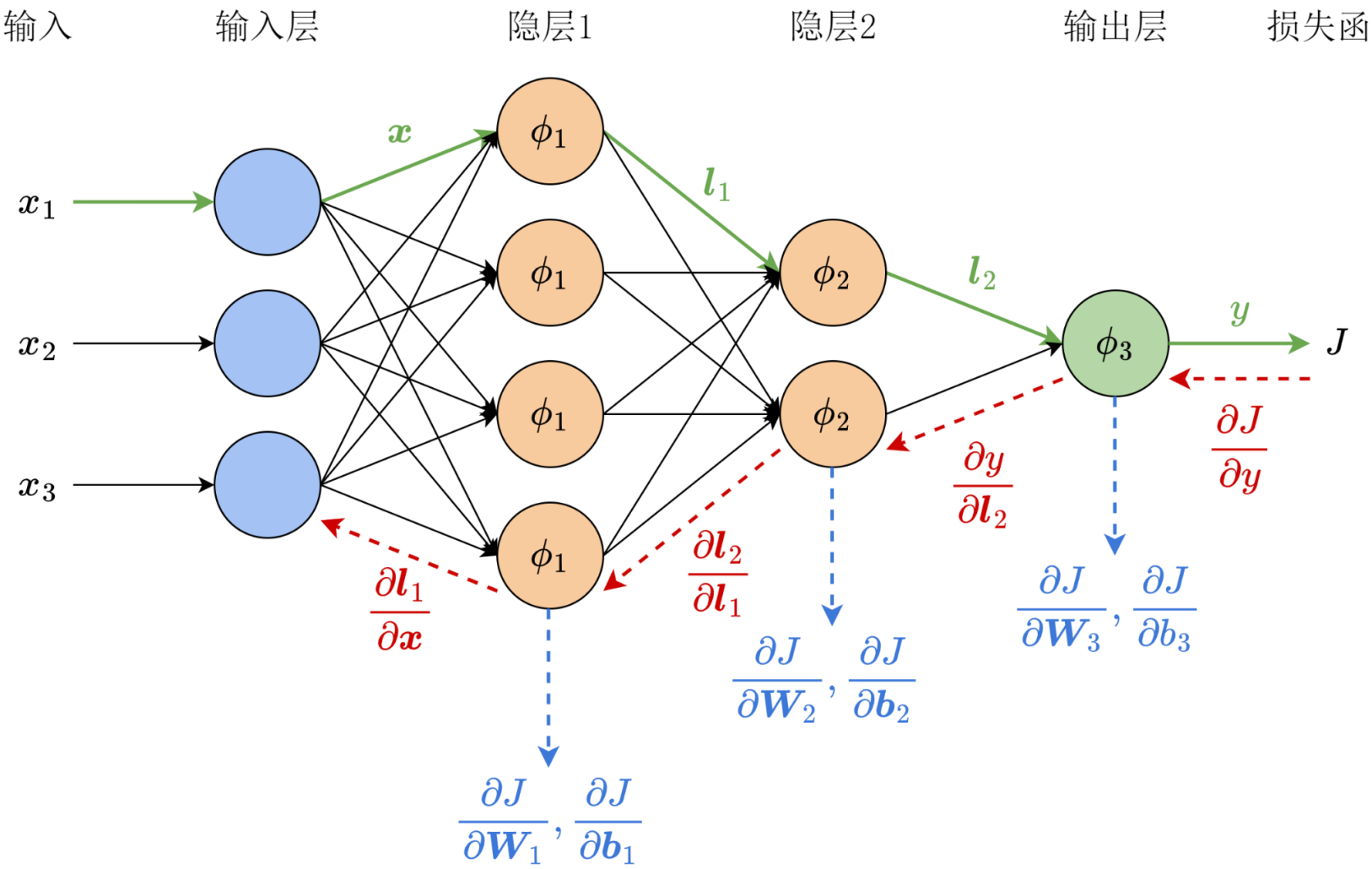
Case 2

$$z = f(s) \Rightarrow x = g(s) \quad y = h(s) \quad z = k(x, y)$$

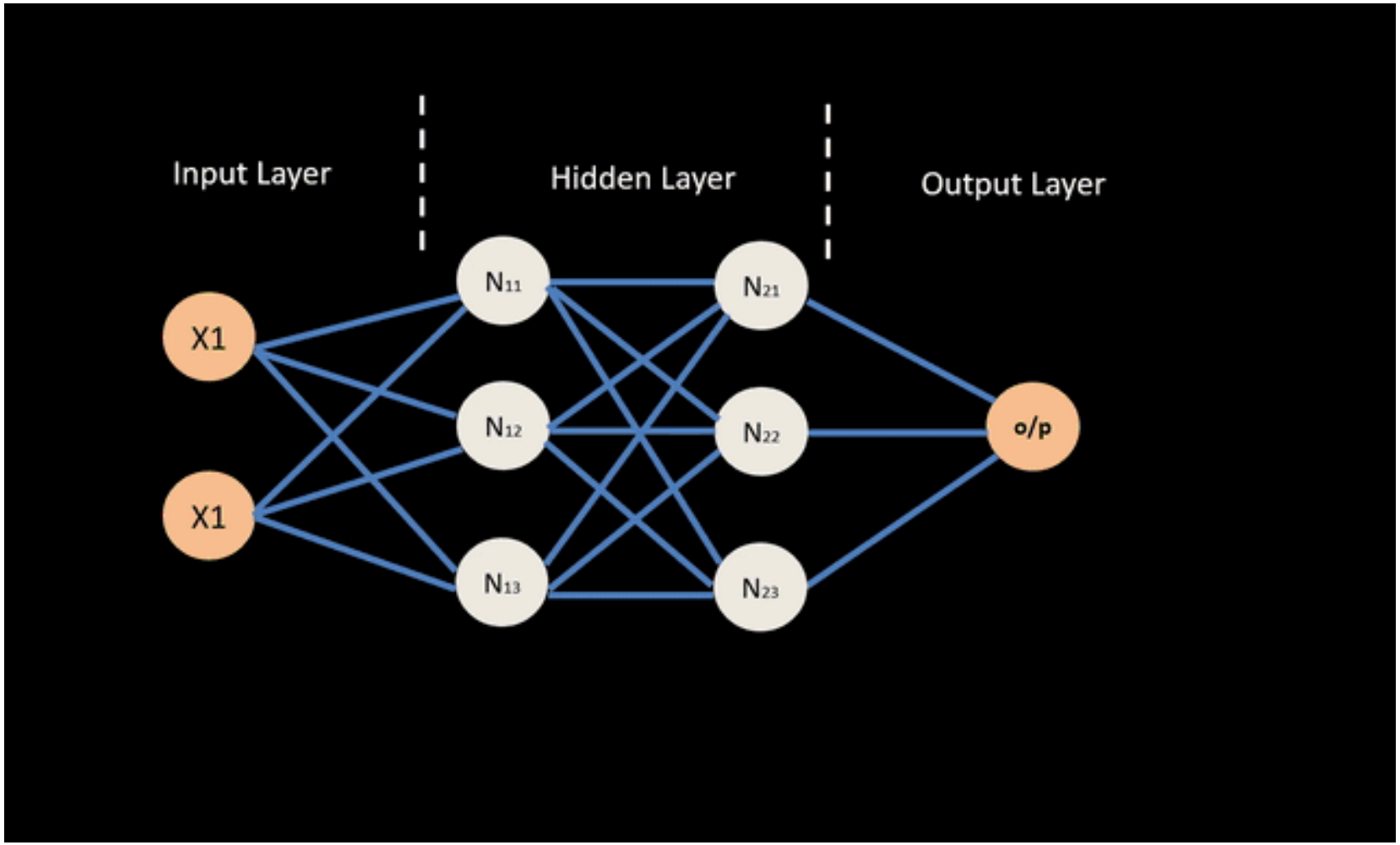


$$\frac{dz}{ds} = \frac{\partial z}{\partial x}\frac{dx}{ds} + \frac{\partial z}{\partial y}\frac{dy}{ds}$$

反向传播的参数更新


输入
输入层
隐层1
隐层2
输出层
损失函数
x_1
x_2
x_3
$\boldsymbol{x}$
ϕ_1
$\boldsymbol{l}_1$
ϕ_2
$\boldsymbol{l}_2$
ϕ_3
y
J
$\frac{\partial \boldsymbol{l}_1}{\partial \boldsymbol{x}}$
$\frac{\partial \boldsymbol{l}_2}{\partial \boldsymbol{l}_1}$
$\frac{\partial y}{\partial \boldsymbol{l}_2}$
$\frac{\partial J}{\partial y}$
$\frac{\partial J}{\partial \boldsymbol{W}_1}, \frac{\partial J}{\partial \boldsymbol{b}_1}$
$\frac{\partial J}{\partial \boldsymbol{W}_2}, \frac{\partial J}{\partial \boldsymbol{b}_2}$
$\frac{\partial J}{\partial \boldsymbol{W}_3}, \frac{\partial J}{\partial b_3}$

多层网络训练过程：反向传播 (back propagation)



多层网络训练过程：反向传播 (back propagation)

- 代码实现（代码加注释）

```
class MLP:
    def backward(self, X, y, output):
        """X: 输入数据, y: 真实标签, output: 模型输出"""
        # 1. 输出层误差
        error = output - y
        dZ2 = error  # 输出层没有激活函数
        dW2 = np.dot(self.a1.T, dZ2)
        db2 = np.sum(dZ2, axis=0, keepdims=True)
        # 2. 传播到隐藏层
        dZ1 = np.dot(dZ2, self.W2.T) * sigmoid_der(self.a1)
        dW1 = np.dot(X.T, dZ1)
        db1 = np.sum(dZ1, axis=0, keepdims=True)
        # 3. 更新权重和偏置
        self.W2 -= self.lr * dW2
        self.b2 -= self.lr * db2
        self.W1 -= self.lr * dW1
        self.b1 -= self.lr * db1
```

$$z_1 = x\,W_1 + b_1,\ \ a_1 = f(z_1)$$
$$z_2 = a_1 W_2 + b_2,\ \ \hat{y} = g(z_2)$$
$$L = 1/2(y - \hat{y})\text{^}2$$

$$d_{\hat{y}} = \frac{\partial L}{\partial \hat{y}} = \hat{y} - y$$
$$d_{z_2} = \frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2} = (\hat{y} - y)\, g'(z_2)$$
$$d_{W_2} = \frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_2} = a_1^T d_{z_2}$$
$$d_{b_2} = \frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial b_2} = d_{z_2}$$

$$d_{z_1} = \frac{\partial L}{\partial z_1} = (\frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1}) \cdot \frac{\partial a_1}{\partial z_1} = (d_{z_2} W_2^T) \cdot f'(z_1)$$
$$d_{W_1} = \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_2}{\partial W_1} = x^T d_{z_1}$$
$$d_{b_1} = \frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1} = d_{z_1}$$

中国科学院大学
University of Chinese Academy of Sciences

目 录

中国科学院大学
University of Chinese Academy of Sciences

目 录

循环神经网络 (Recurrent Neural Network，RNN)

❑ 具有“记忆”，擅长处理时序数据

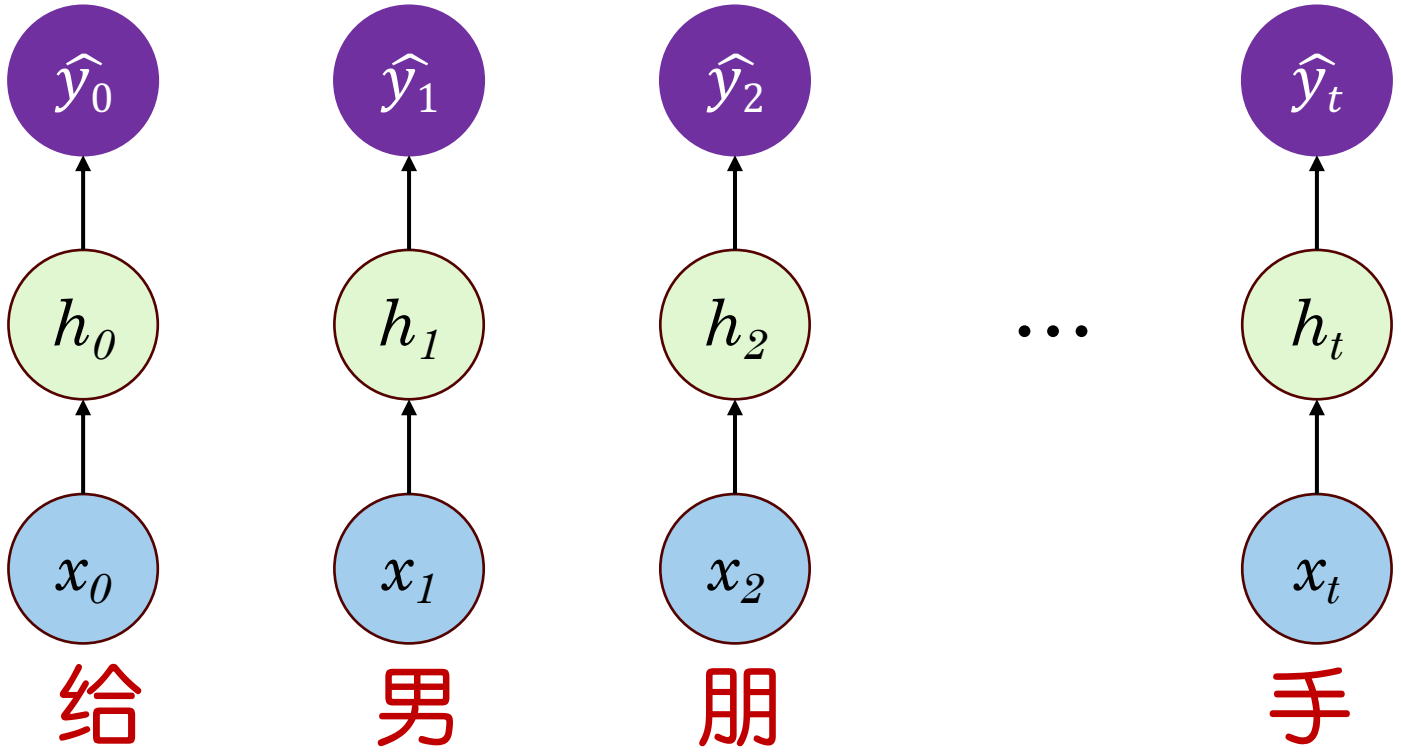
时序数据（Time Series Data）是按时间顺序记录、反映事物随时间变化的一组有序数据集合，如人类语言

人类语言是时序的

《降临》中外星人语言时空一体的

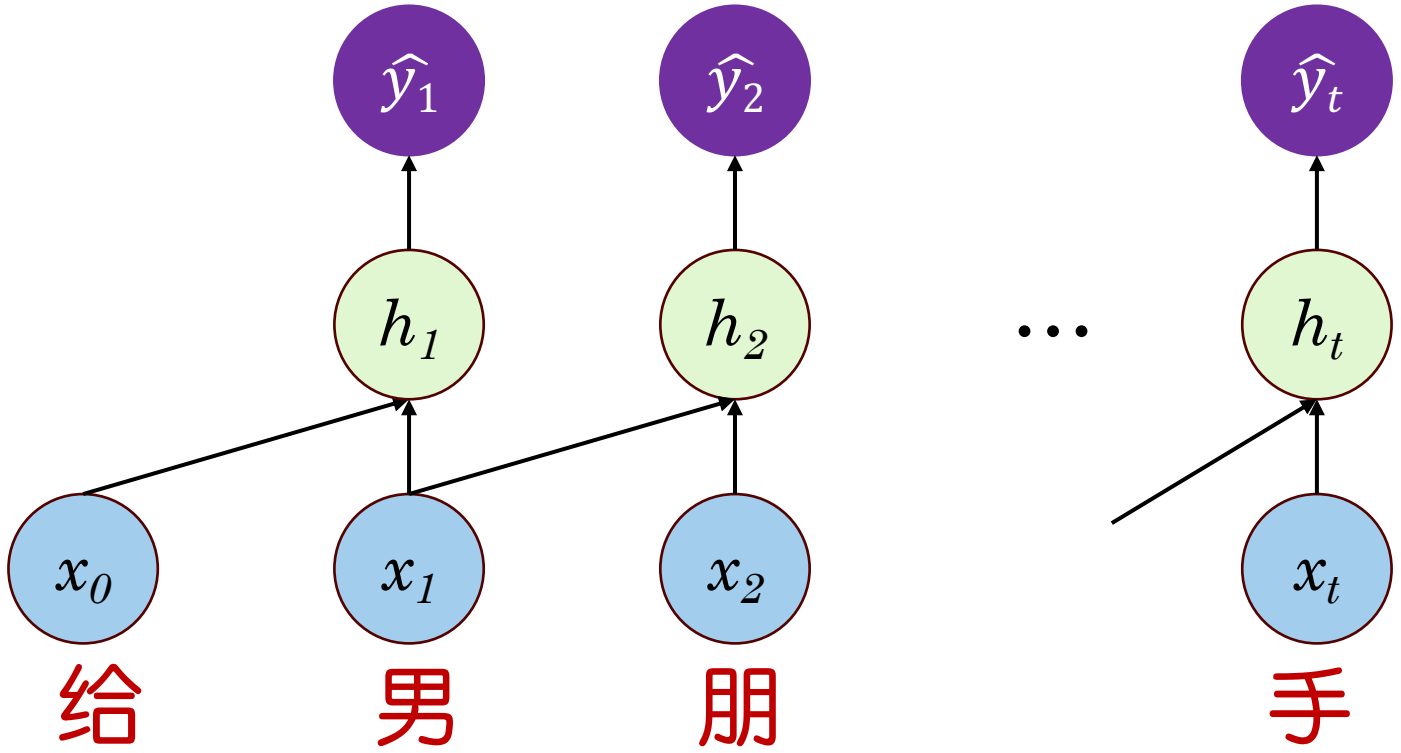
循环神经网络 (Recurrent Neural Network，RNN)

❑ Bi-Gram Language Model



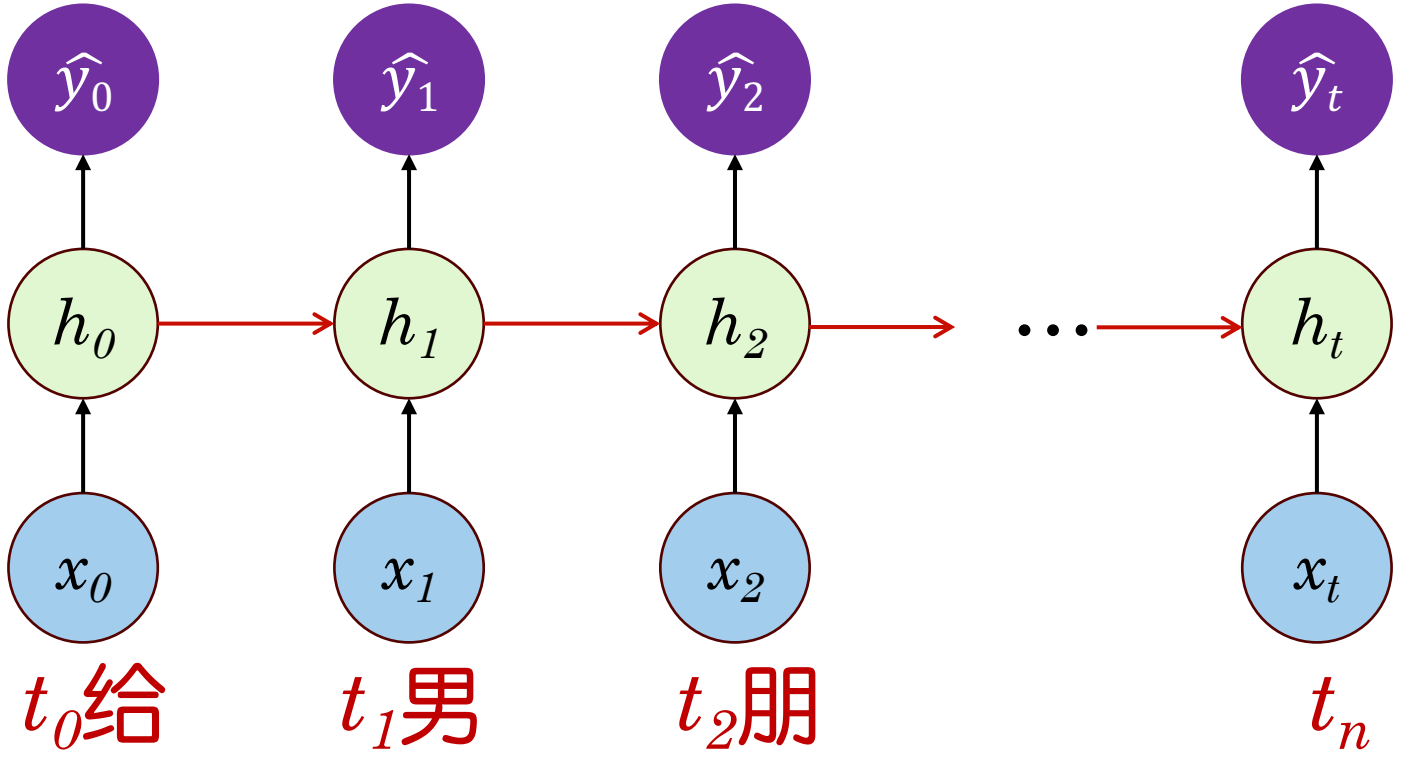
循环神经网络 (Recurrent Neural Network，RNN)

❑ Tri-Gram Language Model



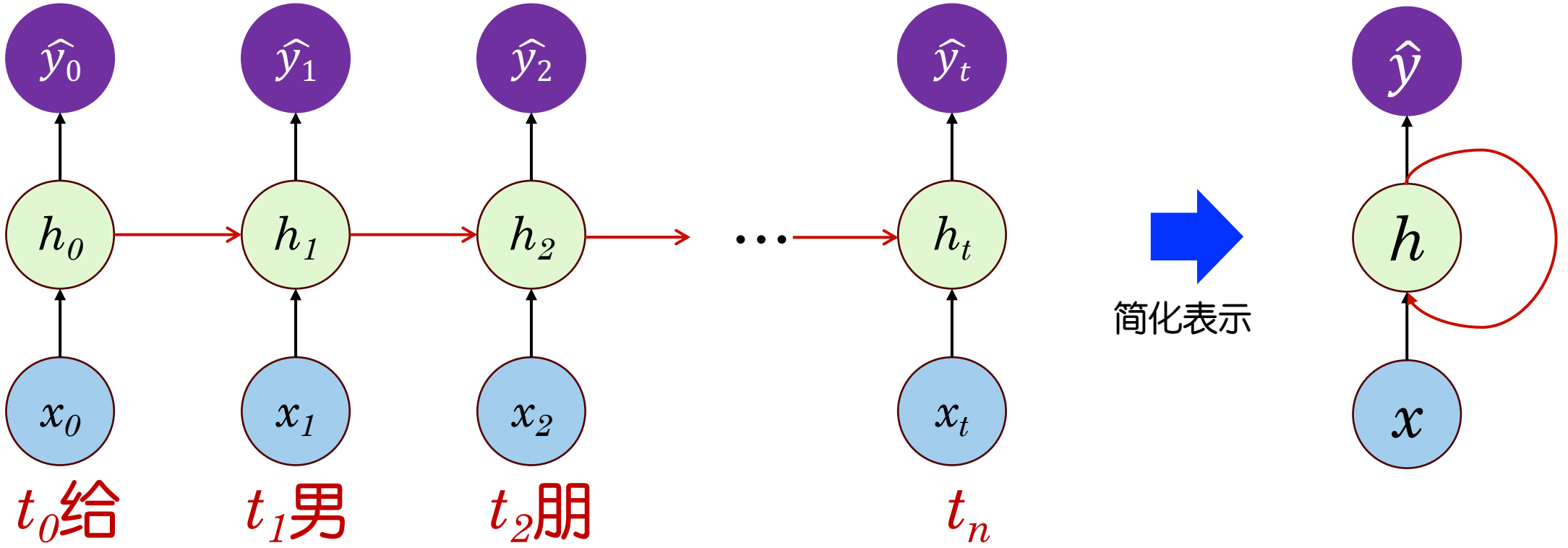
循环神经网络 (Recurrent Neural Network，RNN)

- RNN：t时刻的输出取决于当前及之前的状态
  - 之前的状态“封印”在隐藏层中



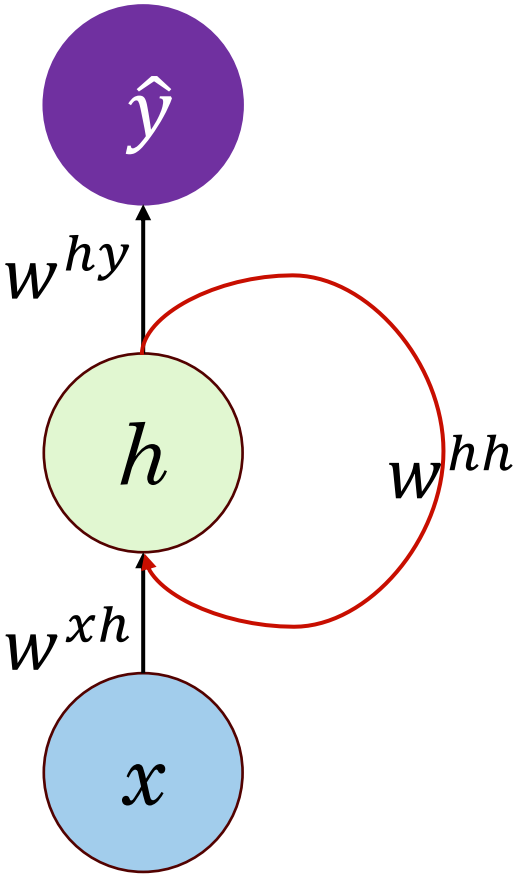
循环神经网络 (Recurrent Neural Network，RNN)

- RNN：t时刻的输出取决于当前及之前的状态
  - 之前的状态“封印”在隐藏层中



循环神经网络 (Recurrent Neural Network，RNN)

- 具有“记忆”，擅长处理时序数据



$$h_t = \sigma\left(w^{hh}h_{t-1} + w^{xh}x_t\right)$$

t层状态

t-1层状态

t层输入

隐层-隐层权重

隐层-输入权重

各自都只有一个矩阵，
多步间共享

循环神经网络 (Recurrent Neural Network，RNN)

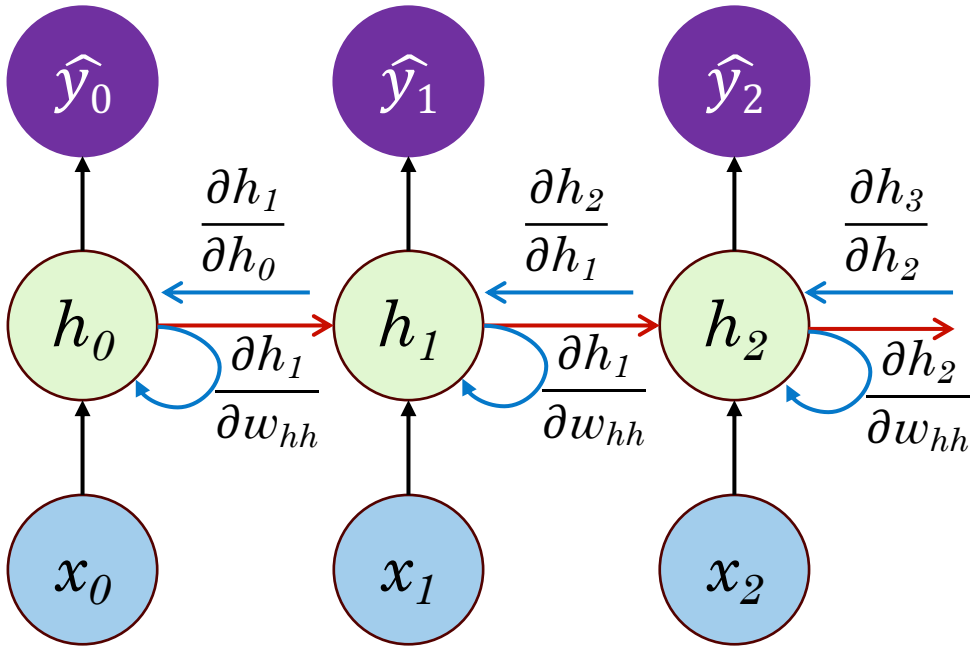
- 代码实现（代码加注释）

```
class RNN:
    def __init__(self, input_size, hidden_size, output_size):
        """input_size: 输入维度, hidden_size: 隐藏状态维度, output_size: 输出维度"""
        self.Wxh = np.random.randn(input_size, hidden_size) # 输入 -> 隐藏层
        self.Whh = np.random.randn(hidden_size, hidden_size) # 隐藏层 -> 隐藏层
        self.Why = np.random.randn(hidden_size, output_size) # 隐藏层 -> 输出层
        self.bh = np.zeros((1, hidden_size)) # 隐藏层偏置
        self.by = np.zeros((1, output_size)) # 输出层偏置

    def forward(self, inputs):
        """inputs: 一个序列 [x1, x2, ..., xt]"""
        h = np.zeros((1, self.Whh.shape[0])) # 初始隐藏状态
        self.hidden_states, outputs = [], []
        for x in inputs: # RNN核心公式
            h = np.tanh(np.dot(x, self.Wxh) + np.dot(h, self.Whh) + self.bh)
            y = np.dot(h, self.Why) + self.by
            self.hidden_states.append(h)
            outputs.append(y)
        return outputs
```

循环神经网络 (Recurrent Neural Network，RNN)

❑ 训练中的梯度消失/爆炸

- 直观理解：梯度更新时，梯度被近距离梯度主导，模型倾向于用近距离梯度更新参数，导致难以学到长距离依赖关系



$$\frac{dh_t}{d\theta} = \frac{\partial h_t}{\partial h_{t-1}}\frac{dh_{t-1}}{d\theta} + \frac{\partial h_t}{\partial \theta}$$

$$= \frac{\partial h_t}{\partial \theta} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial \theta}$$

$$+ \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\frac{\partial h_{t-2}}{\partial \theta} + \cdots$$

$$+ \cdots + \underbrace{\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_2}{\partial h_1}}_{\approx 0/\infty}\frac{\partial h_1}{\partial \theta}$$

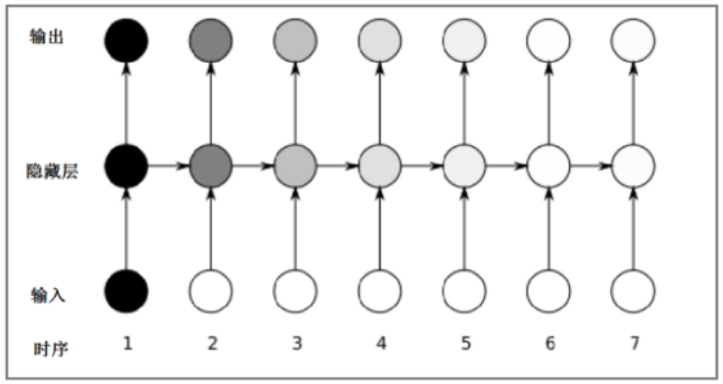
中国科学院大学
University of Chinese Academy of Sciences

目 录

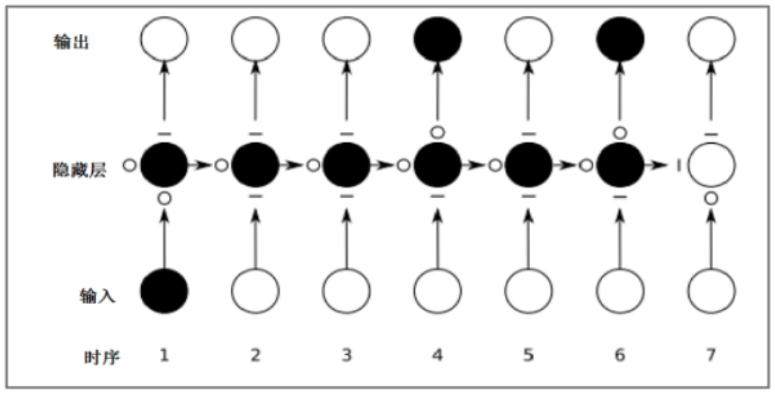
长短期记忆网络(Long Short-Term Memory, LSTM)

- 原理：增加“记忆细胞”来解决RNN的梯度消失/爆炸

LSTM单元不仅接受 x_t 和 h_{t-1}，还需建立一个机制(维持一个细胞状态*Ct*) 能保留前面远处结点信息在长距离传播中不会被丢失



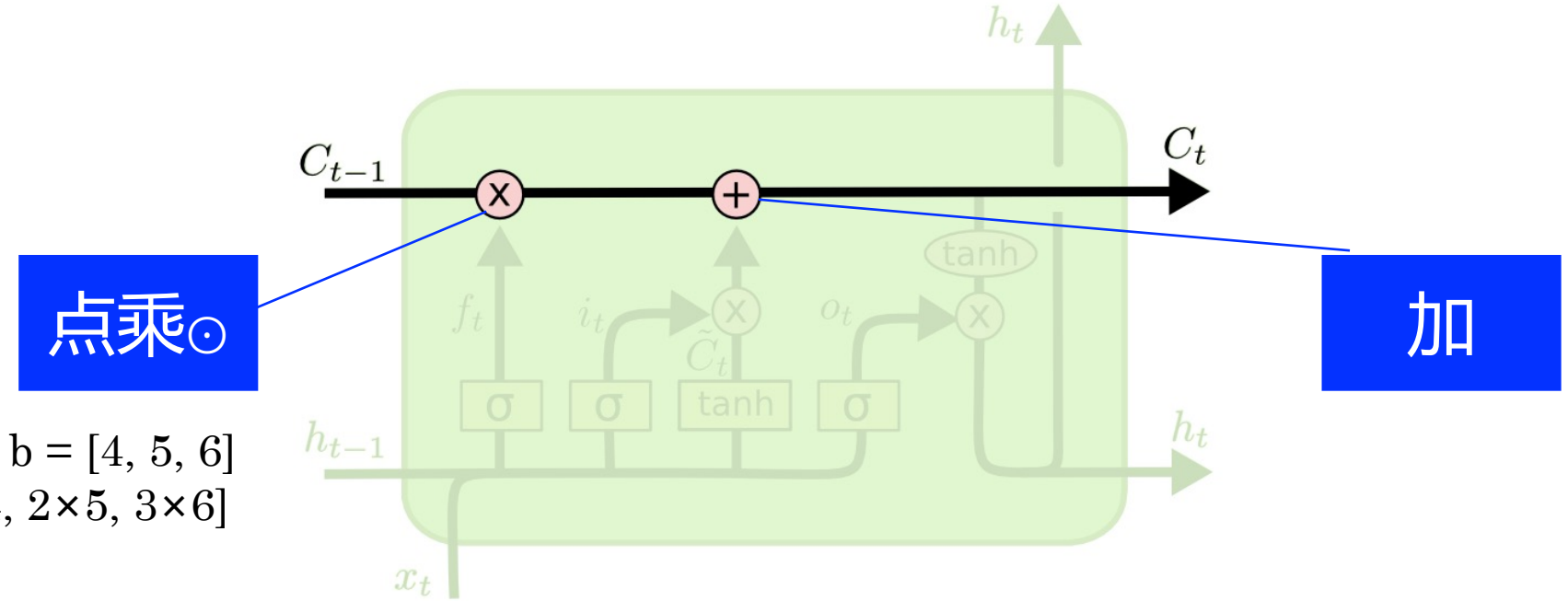
RNN



LSTM

长短期记忆网络(Long Short-Term Memory, LSTM)

记忆细胞可以实现**有选择性的长期记忆**，信息在上面方便保持

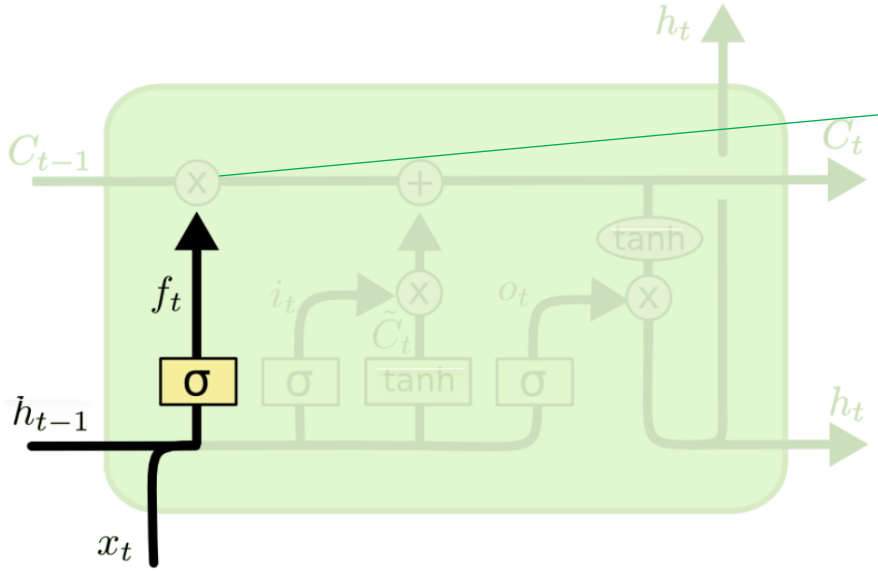


a = [1, 2, 3]， b = [4, 5, 6]
a ⊙ b = [1×4, 2×5, 3×6]

LSTM 通过设计“门”结构实现记忆选择

长短期记忆网络(Long Short-Term Memory, LSTM)

- 遗忘门（Forget Gate）：决定从细胞状态中“遗忘”哪些信息



$C_{t-1} = [1, 2, 3]$，$f_t = [0.4, 0.5, 0.6]$
$C_{t-1} \odot f_t = [0.4, 1.0, 1.8]$

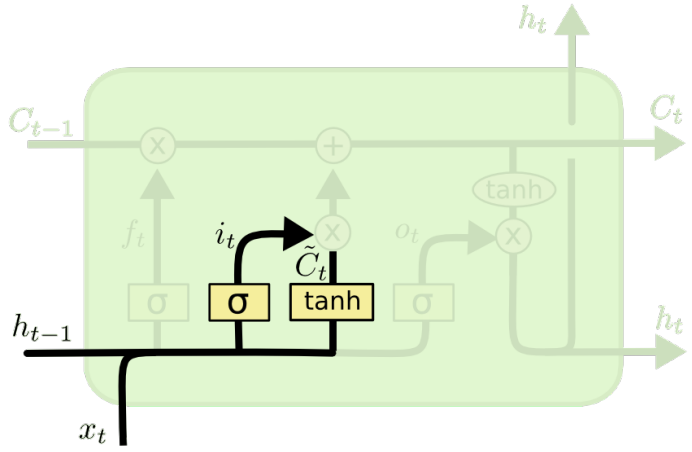
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \in (0,1)^h$$

h维的(0,1)实数向量，用于逐维决定细胞状态保留多少

例如：在预测下一个词时，细胞状态可能包含了主语男女性别的代词(“他”或“她”)，当看到新的代词时，可以考虑忘记旧的代词

长短期记忆网络(Long Short-Term Memory, LSTM)

- 输入门（Input Gate）：决定哪些新信息“更新/存入”细胞状态

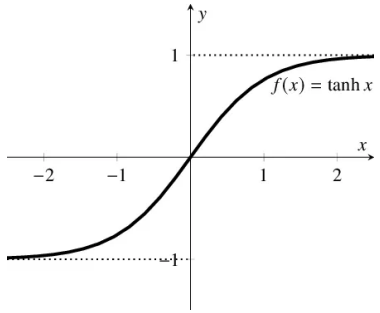


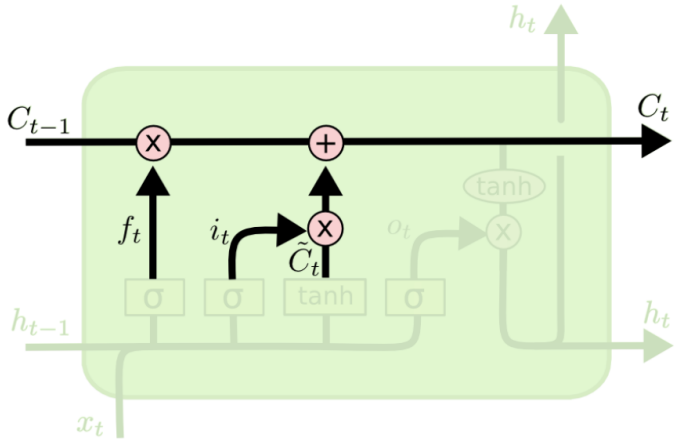
Sigmoid: h维的(0,1)实数向量，用于**决定哪些值需要更新**

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right) \in (0,1)^h$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C) \in (-1,1)^h$$

Tanh: h维的(-1,1)实数向量，用于生成**新的候选值**





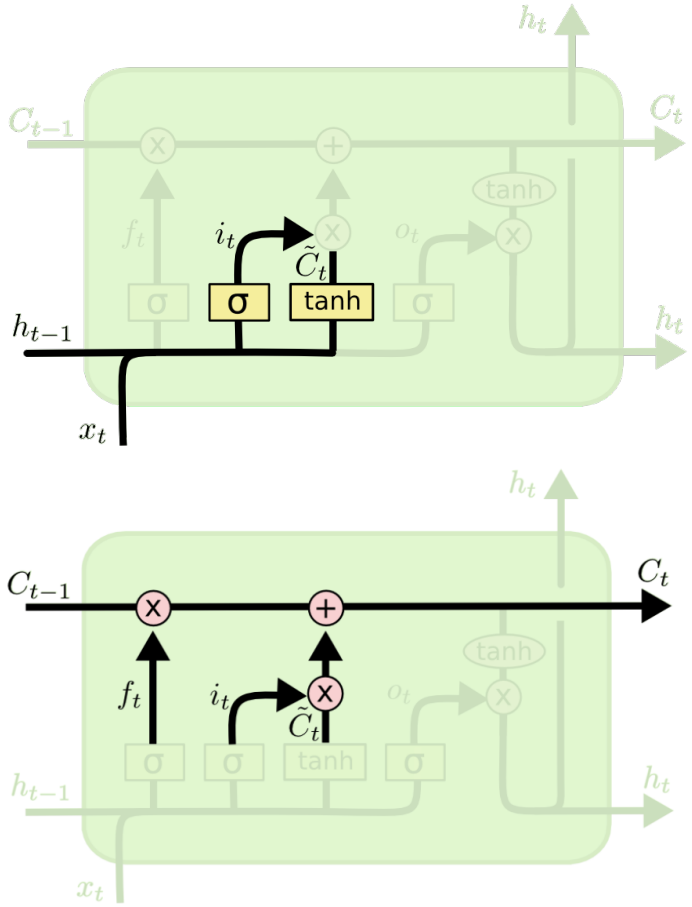
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

新的细胞状态：遗忘掉了一些值，同时又更新了一些

例如：当看到新代词时，需将新代词更新入细胞状态

长短期记忆网络(Long Short-Term Memory, LSTM)

- 输入门（Input Gate）：决定哪些新信息“更新/存入”细胞状态



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right) \in (0,1)^h$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C) \in (-1,1)^h$$

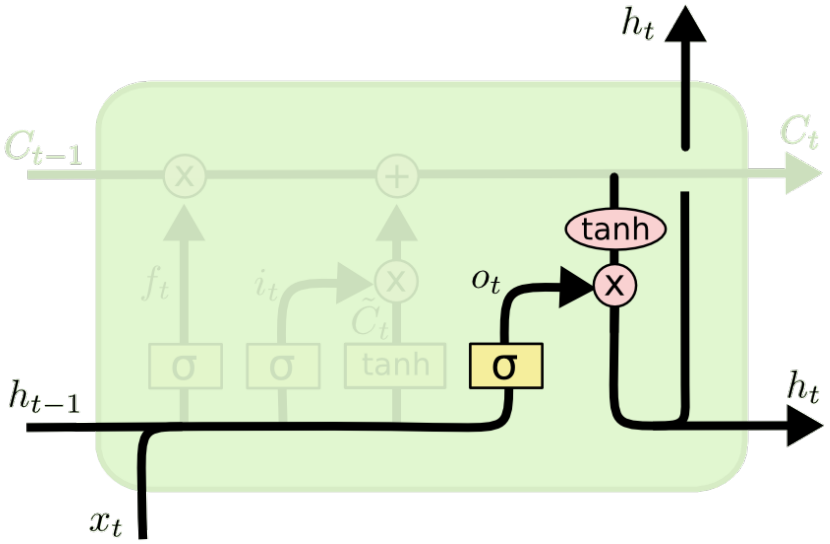
如果**遗忘门始终为1（都不遗忘）且输入门始终为0（不更新信息）**，则过去的细胞状态**（记忆）**将随时间被保存，并传递到当前时间步**（长距离依赖）**

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

例如：当看到新代词时，需将新代词更新入细胞状态

长短期记忆网络(Long Short-Term Memory, LSTM)

- 输出门（Output Gate）：基于细胞状态和当前输入，决定输出哪些信息到隐藏状态



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t \odot \tanh\left(C_t\right)$$

细胞状态影响隐藏状态

例如：根据细胞状态中的性别信息和当前输入，确定下一个词

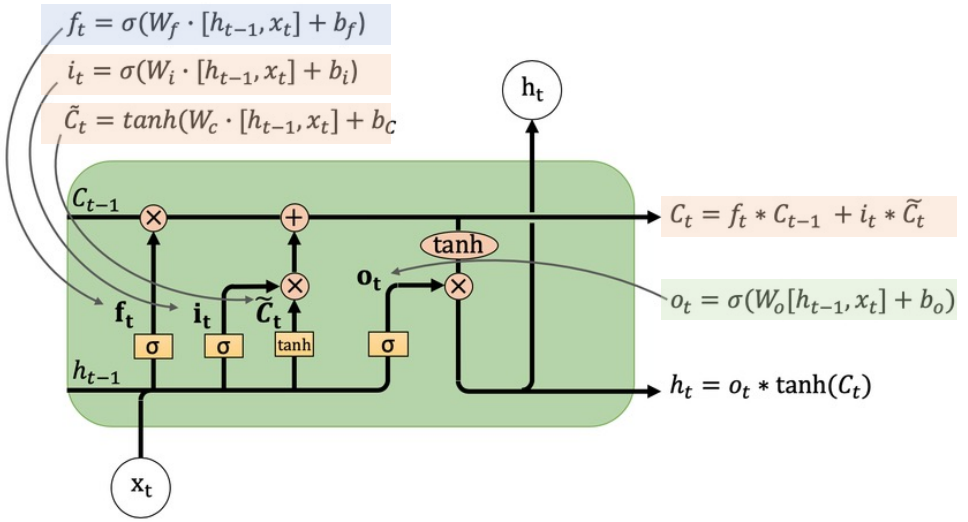
长短期记忆网络(Long Short-Term Memory, LSTM)

- 代码实现（代码加注释）

```
class LSTM:
    def __init__(self, input_size, hidden_size):
        """input_size: 输入维度, hidden_size: 隐藏状态维度"""
        # 初始化权重矩阵
        self.Wf = np.random.randn(input_size + hidden_size, hidden_size) # 遗忘门
        self.Wi = np.random.randn(input_size + hidden_size, hidden_size) # 输入门
        self.Wc = np.random.randn(input_size + hidden_size, hidden_size) # 候选记忆
        self.Wo = np.random.randn(input_size + hidden_size, hidden_size) # 输出门
        # 偏置
        self.bf = np.zeros((1, hidden_size))
        self.bi = np.zeros((1, hidden_size))
        self.bc = np.zeros((1, hidden_size))
        self.bo = np.zeros((1, hidden_size))
```

长短期记忆网络(Long Short-Term Memory, LSTM)

- 代码实现（代码加注释）

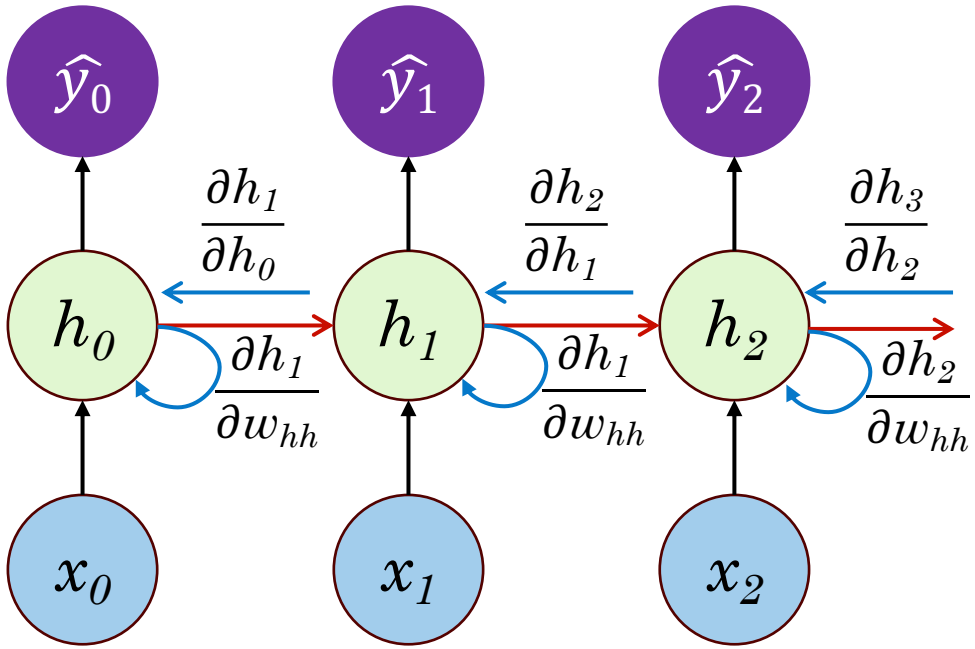



```
class LSTM:
    def __init__(self, input_size, hidden_size):
        . . . . . .(前一页)
    def forward(self, inputs):
        """inputs: 输入序列 [x1, x2, ..., xt]"""
        h = np.zeros((1, self.hidden_size)) # 隐藏状态
        c = np.zeros((1, self.hidden_size)) # 记忆状态
        outputs = []
        for x in inputs:
            combined = np.concatenate((h, x), axis=1) # 拼接输入和上一时刻隐藏状态
            f = self.sigmoid(np.dot(combined, self.Wf) + self.bf) # 遗忘门
            i = self.sigmoid(np.dot(combined, self.Wi) + self.bi) # 输入门
            c_tilde = np.tanh(np.dot(combined, self.Wc) + self.bc)# 候选记忆
            c = f * c + i * c_tilde # 更新记忆
            o = self.sigmoid(np.dot(combined, self.Wo) + self.bo) # 输出门
            h = o * np.tanh(c) # 更新隐藏状态
            outputs.append(h)
        return outputs
```

回忆：RNN

❑ 训练中的梯度消失/爆炸

- 直观理解：梯度更新时，梯度被近距离梯度主导，模型倾向于用近距离梯度更新参数，导致难以学到长距离依赖关系



$$\frac{dh_t}{d\theta} = \frac{\partial h_t}{\partial h_{t-1}}\frac{dh_{t-1}}{d\theta} + \frac{\partial h_t}{\partial \theta}$$

$$= \frac{\partial h_t}{\partial \theta} + \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial \theta}$$

$$+ \frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\frac{\partial h_{t-2}}{\partial \theta} + \cdots$$

$$+ \cdots + \underbrace{\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots \frac{\partial h_2}{\partial h_1}}_{\approx 0/\infty}\frac{\partial h_1}{\partial \theta}$$

长短期记忆网络(Long Short-Term Memory, LSTM)

- 梯度问题分析

$$
\begin{aligned}
f_t &= \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \\
i_t &= \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right) \\
o_t &= \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right) \\
\hat{c}_t &= \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \hat{c}_t \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

$$
\frac{\partial h_t}{\partial h_{t-1}} \rightarrow \frac{\partial c_t}{\partial c_{t-1}}
$$

$$
\frac{\partial c_t}{\partial c_{t-1}} = f_t + c_{t-1}\frac{\partial f_t}{\partial c_{t-1}} + \hat{c}_t \frac{\partial i_t}{\partial c_{t-1}} + i_t \frac{\partial \hat{c}_t}{\partial c_{t-1}}
$$

$\frac{\partial i_t}{\partial c_{t-1}}$ <<1, $\frac{\partial \hat{c}_t}{\partial c_{t-1}}$ <<1

$$
c_{t-1}\frac{\partial f_t}{\partial c_{t-1}} = f_t\left(1-f_t\right)o_{t-1}\left(1-\tanh^2 c_{t-1}\right)c_{t-1}U_f
$$

f_t <1, $(1-f_t)$ <1, o_{t-1} <1, $(1-\tanh^2 c_{t-1})$ <0.45

整体 <<1

长短期记忆网络(Long Short-Term Memory, LSTM)

- 梯度问题分析

$$
\begin{aligned}
f_t &= \sigma\left(W_f x_t + U_f h_{t-1} + b_f\right) \\
i_t &= \sigma\left(W_i x_t + U_i h_{t-1} + b_i\right) \\
o_t &= \sigma\left(W_o x_t + U_o h_{t-1} + b_o\right) \\
\hat{c}_t &= \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \hat{c}_t \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

$$
\frac{\partial h_t}{\partial h_{t-1}} \Rightarrow \frac{\partial c_t}{\partial c_{t-1}}
$$

$$
\frac{\partial c_t}{\partial c_{t-1}} = f_t + c_{t-1}\frac{\partial f_t}{\partial c_{t-1}} + \hat{c}_t\frac{\partial i_t}{\partial c_{t-1}} + i_t\frac{\partial \hat{c}_t}{\partial c_{t-1}} \ll 1
$$

直观理解：

- 不应该被遗忘的（f_t=1）被一直保存
- 该被遗忘的（f_t=0）梯度消失

长短期记忆网络(Long Short-Term Memory)

短期记忆 [编辑]

文A 38种语言

条目 讨论 汉 漢 大陆简体 阅读 编辑 查看历史 工具

维基百科，自由的百科全书

短期记忆（英语：**Short-term memory**，也称为**primary memory**或者**active memory**）是记忆的一种类型，它可以在头脑中让少量信息保持激活状态，在短时间内可以使用。短期记忆的持续时间（在没有复述或者激活的情况下）以秒计算，通常在5－20秒。与长期记忆相比，短期记忆对信息的储存时间较短，信息储存的容量也很有限。关于短期记忆的容量，一个常常引用的数字是7 ± 2个元素。

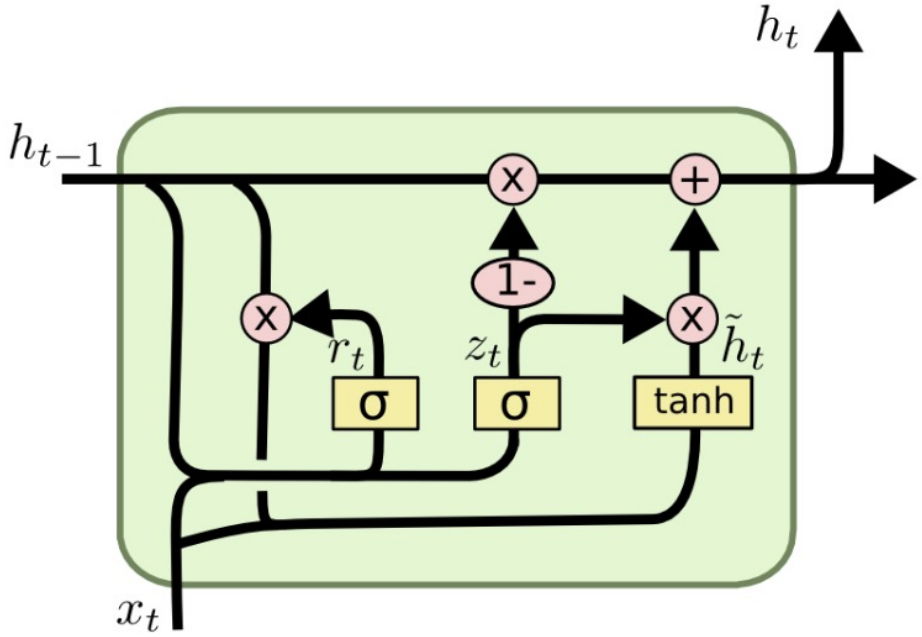
Corsi 块敲击测试：https://www.labvanced.com/player.html?id=74622

长期记忆 [编辑]

长期记忆是能够保持几天到几年的记忆。它与工作记忆以及短期记忆不同，后二者只保持几秒到几小时。生物学上来讲，短期记忆是神经连接的暂时性强化，生理上的结构是反响回路（reverberatory circuit），而通过巩固后、可变为长期记忆。

GRU（Gate Recurrent Unit）：简化版LSTM

GRU 将 LSTM 的输入门和遗忘门合为更新门（update gate）



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

参数比LSTM 少，但是却也能够达到与 LSTM 相当的功能

中国科学院大学
University of Chinese Academy of Sciences

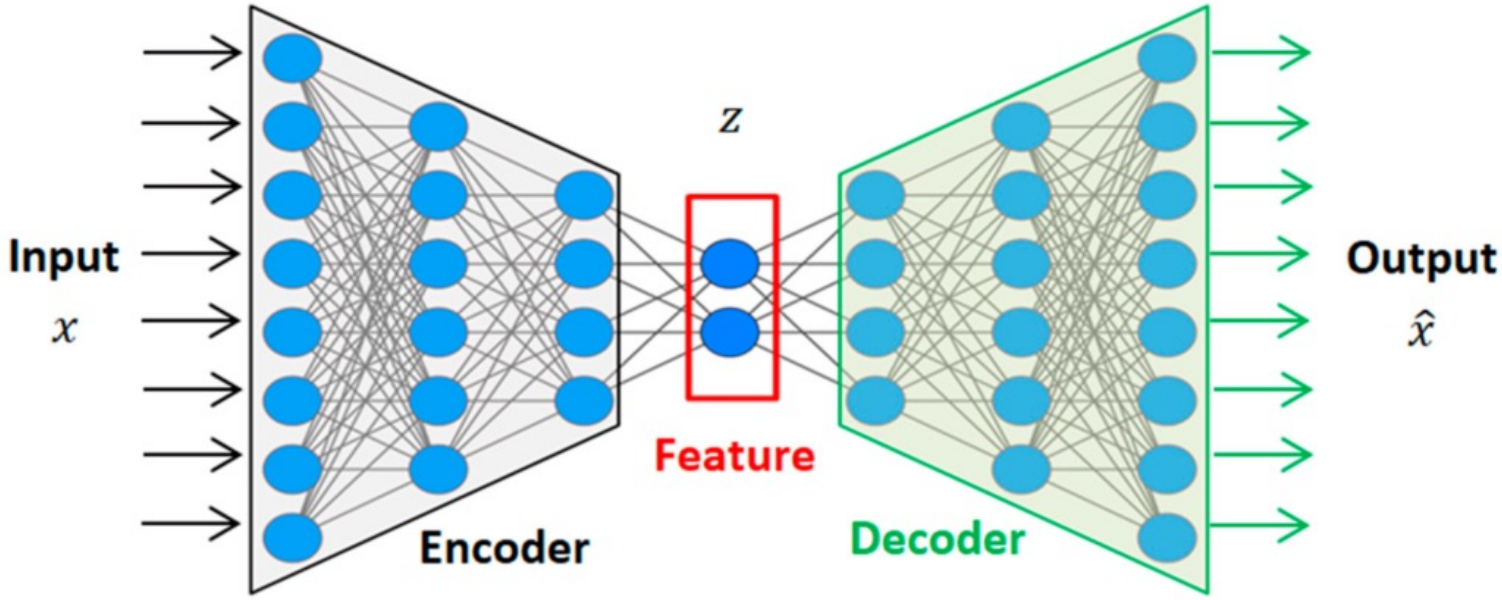
目 录

编码器-解码器（Encoder-Decoder）

- 广泛用于处理序列到序列（Seq2Seq）任务的架构
  1. 将输入序列先通过编码器压缩为一个中间表示
  2. 再由解码器根据该表示逐步生成输出序列
  3. 编码器、解码器都是神经网络（如RNN）

编码器-解码器（Encoder-Decoder）

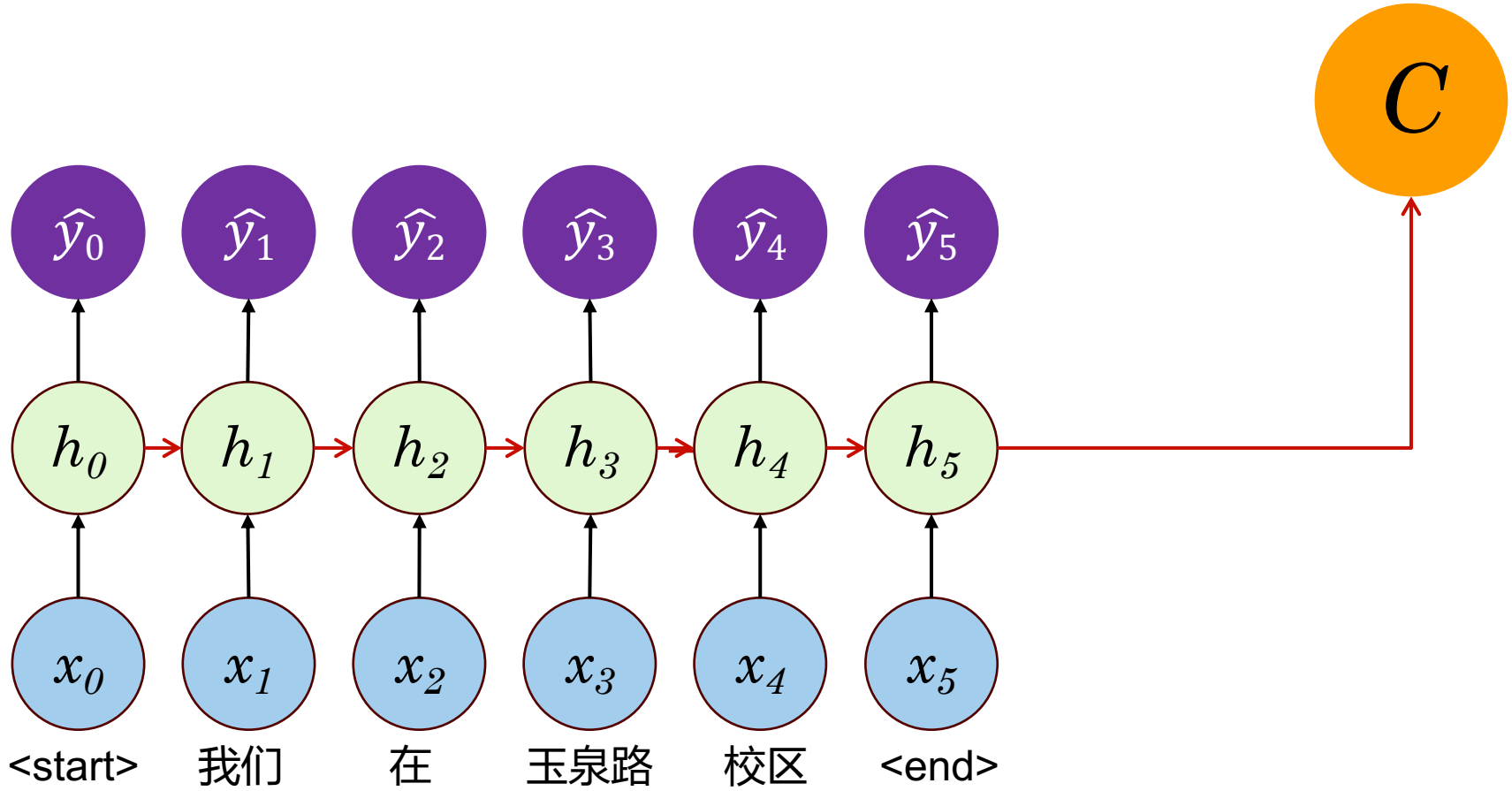
- 广泛用于处理序列到序列（Seq2Seq）任务的架构
  1. 编码器（NN）先将输入序列通压缩为一个中间表示
  2. 解码器（NN）再根据该中间表示逐步生成输出序列



Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems 27 (2014).

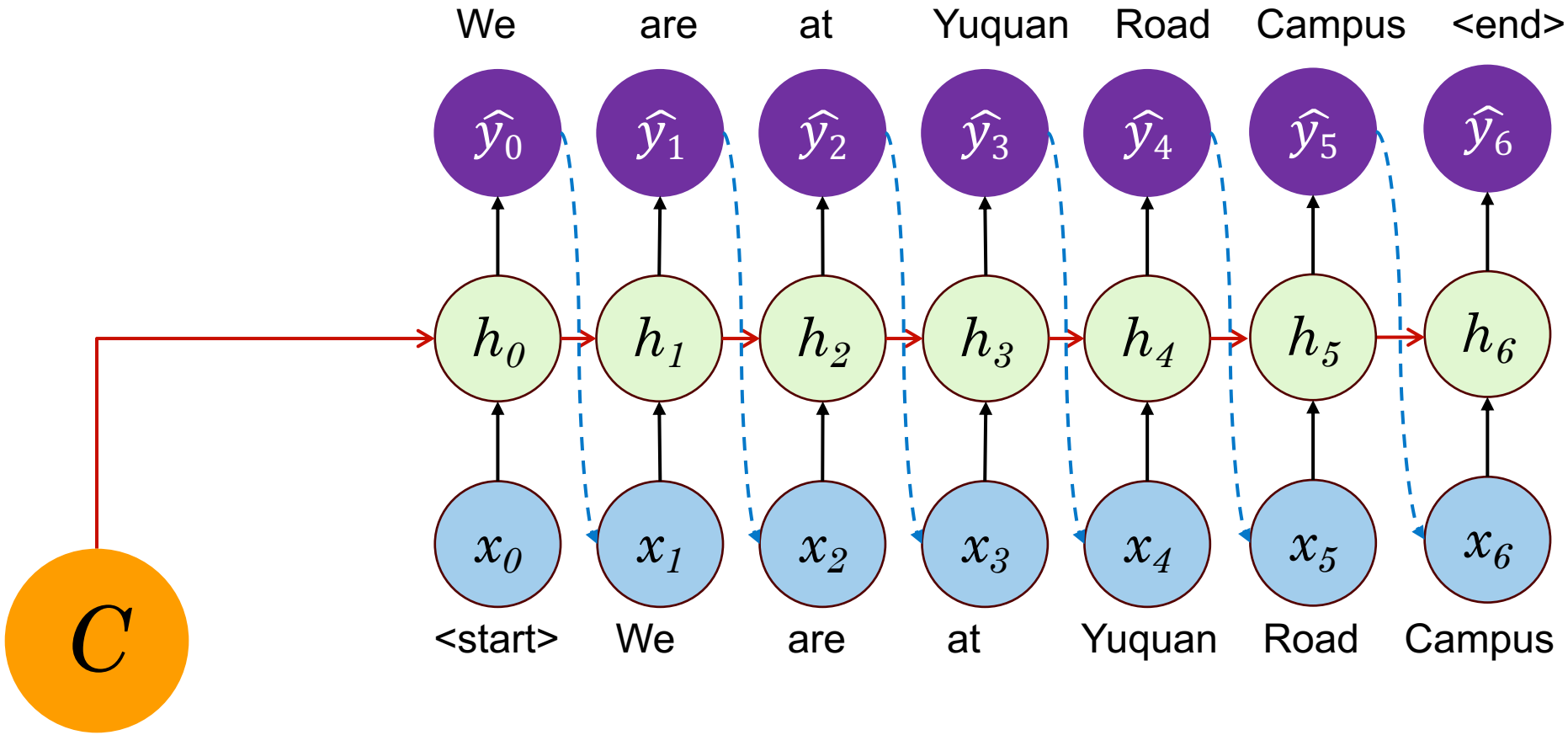
Encoder：先整体理解

- **理解**：将输入压缩为包含所有语义的固定长度向量



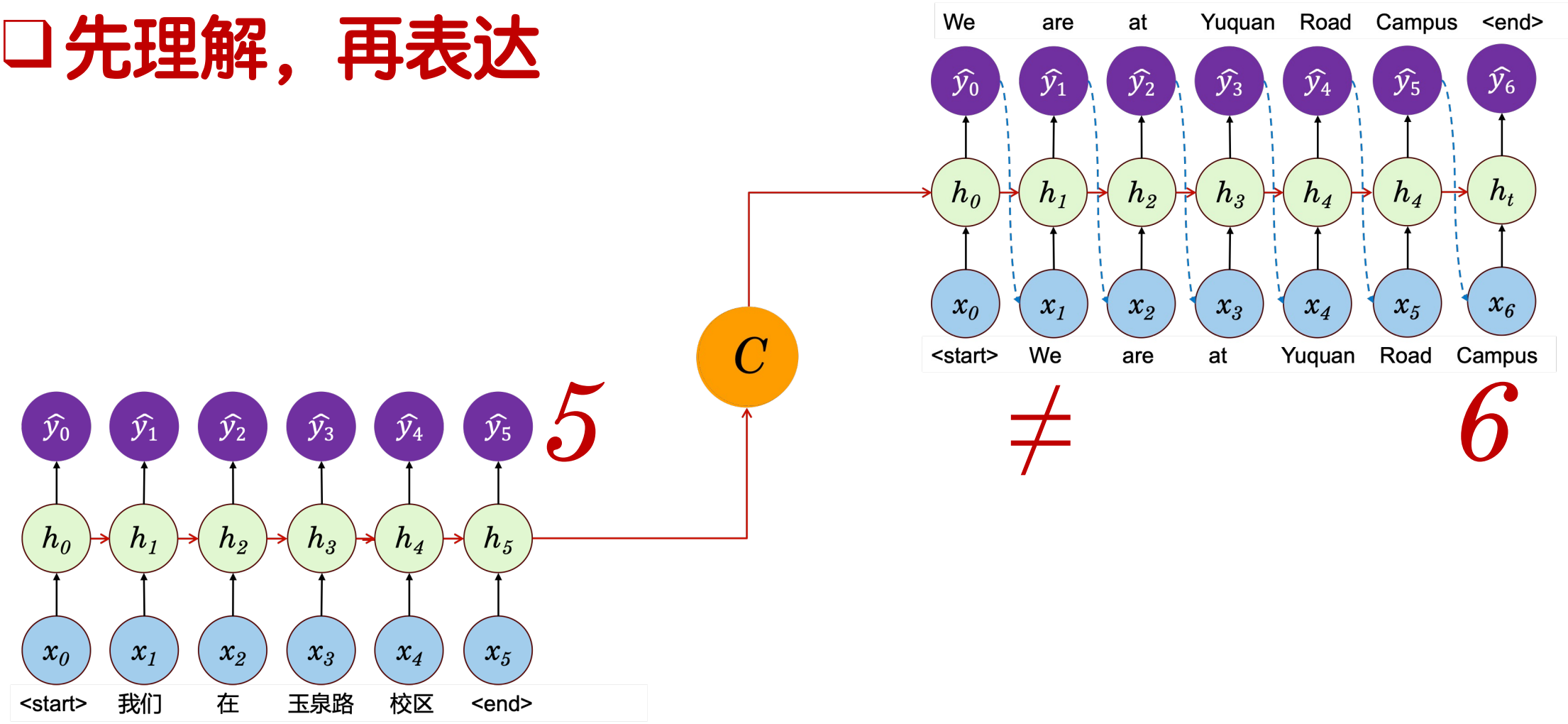
Decoder：在逐字输出

- **表达**：将“理解向量”按需（如英文）逐字输出



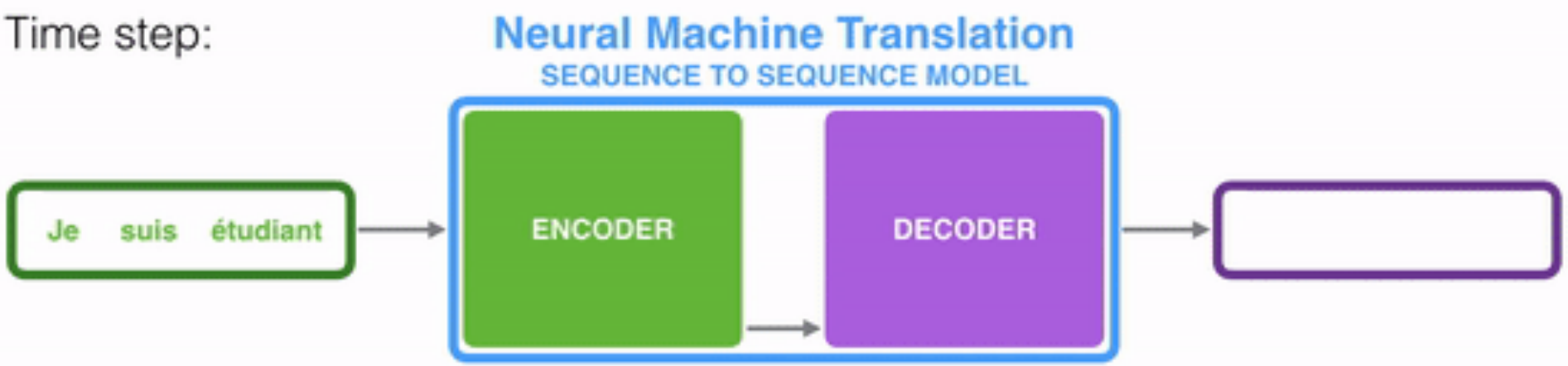
Encoder-Decoder

- 先理解，再表达



Encoder-Decoder

- 先整体理解，再逐字表达



Encoder-Decoder

- 代码实现（代码加注释）

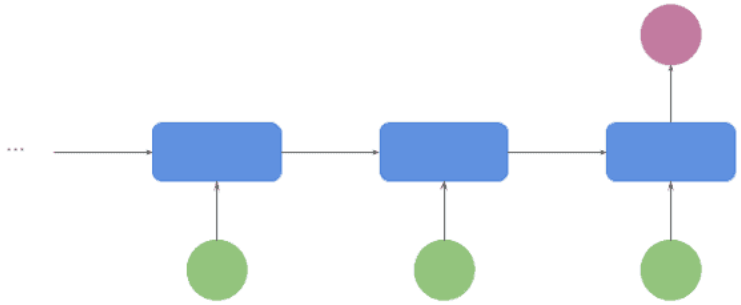
```
class EncoderDecoder:
    """基于RNN实现，包含编码器和解码器"""
    def __init__(self, input_size, hidden_size, output_size):
        """input_size : 输入维度, hidden_size : 隐藏状态维度"""
        # Encoder参数
        self.Wxh = np.random.randn(input_size, hidden_size) # 输入 -> 隐藏层
        self.Whh = np.random.randn(hidden_size, hidden_size) # 隐藏层-> 隐藏层
        self.bh = np.zeros((1, hidden_size)) # 偏置

        # Decoder参数
        self.Wyh = np.random.randn(output_size, hidden_size) # 输出 -> 隐藏层
        self.Whh_dec = np.random.randn(hidden_size, hidden_size) # 隐藏层-> 隐藏层
        self.by = np.zeros((1, hidden_size)) # 偏置

        # 输出层
        self.Who = np.random.randn(hidden_size, output_size)
        self.bo = np.zeros((1, output_size))
```

Encoder-Decoder

- 代码实现（代码加注释）

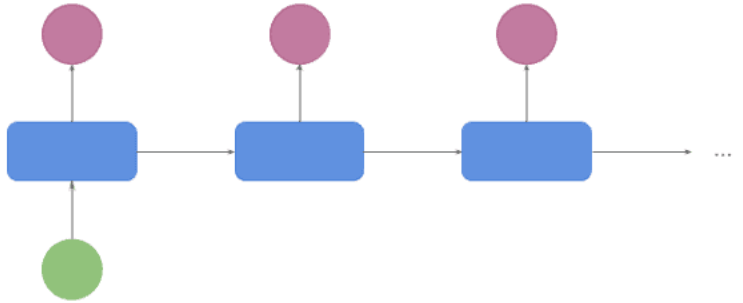
```
class EncoderDecoder:
    def __init__(self, input_size, hidden_size, output_size):
        . . . . . .(前一页)
    def forward(self, inputs, target_len):
        """Encoder: 将输入序列编码为中间表示"""
        h = np.zeros((1, self.hidden_size)) # 初始隐藏状态
        for x in inputs: # RNN核心公式
            h = tanh(np.dot(x, self.Wxh) + np.dot(h, self.Whh) + self.bh)

        """Decoder: 根据中间表示生成输出序列"""
        y = np.zeros((1, self.Who.shape[1])) # 初始输入
        outputs = []
        for _ in range(target_len):
            h = tanh(np.dot(y, self.Wyh) + np.dot(h, self.Whh_dec) + self.by)
            o = softmax(np.dot(h, self.Who) + self.bo) # 输出层
            outputs.append(o) # 统计输出结果
            y = o # 将当前输出作为下一步输入
        return outputs
```

Encoder-Decoder类型



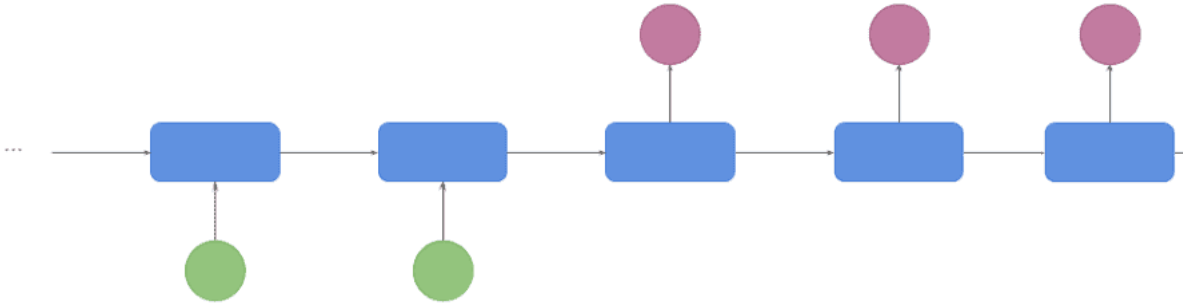
(1) N-1

情感分类



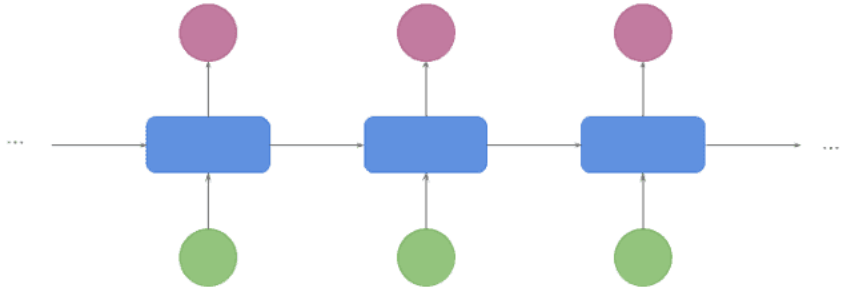
(2) 1-N

语音合成、图像描述



(3) N-M (M可等于N)

翻译、问答、摘要

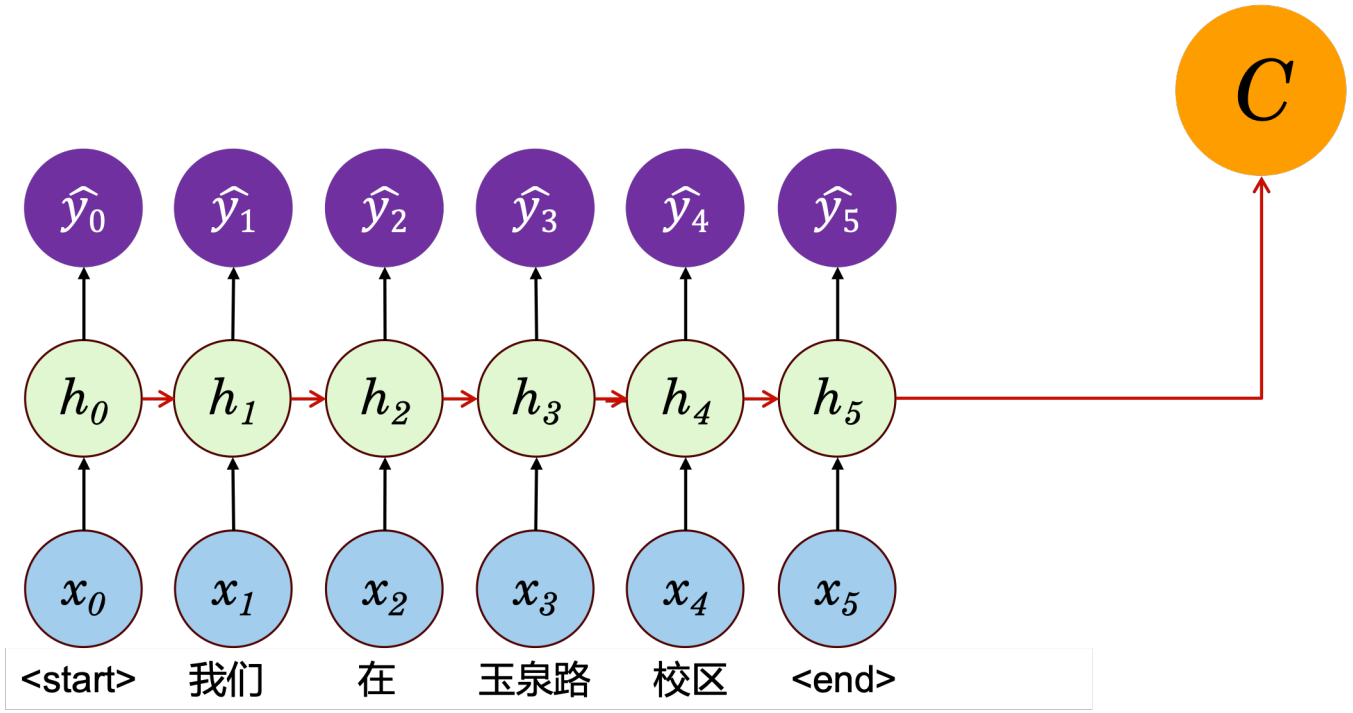


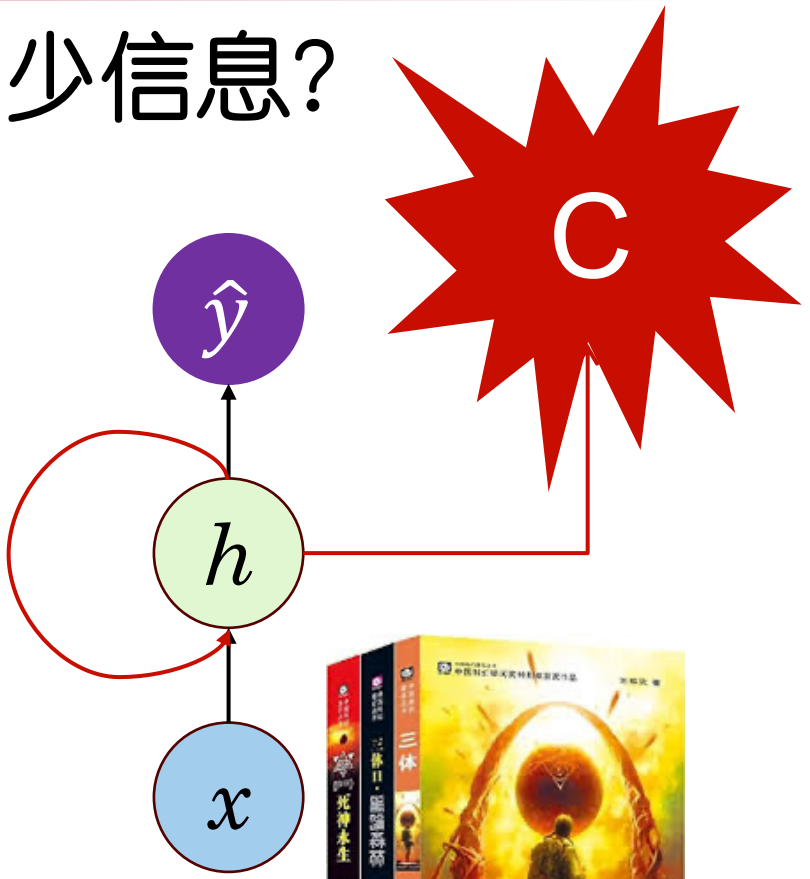
(4) N-N

Tagging

不足之处

- 区区一个固定长度的C，能容下多少信息？





Attention

Coming Soon …

本节复习

- 感知机、前馈、w/b，$h/x/y$
- 梯度下降、反向传播、$argmin_{w,b}L$
- RNN、LSTM、梯度消失/爆炸
- 有RNN为啥还需要Encoder-Decoder?

参考文献

- Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015
- Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems 27 (2014).

思考：《降临》中外星人的语言适合哪个模型?

CNN?

致 谢

- 胡玥、曹亚男、方芳：国科大《自然语言处理基础》
- 曹亚男、任昱冰：国科大《深度学习与自然语言处理概述》


THANKS
https://ictkc.github.io/teaching/2026spring-nlp