

2023-2024学年第一学期本科生课程

《神经网络与深度学习》

第三节：激活函数与损失函数

主讲人：戴金晟(副教授，博士生导师)

daijincheng@bupt.edu.cn

神经网络与深度学习课程组



北京邮电大学
Beijing University of Posts and Telecommunications

内容导览



多层感知机

通用近似定理

常用激活函数

损失函数

内容导览



多层感知机

通用近似定理

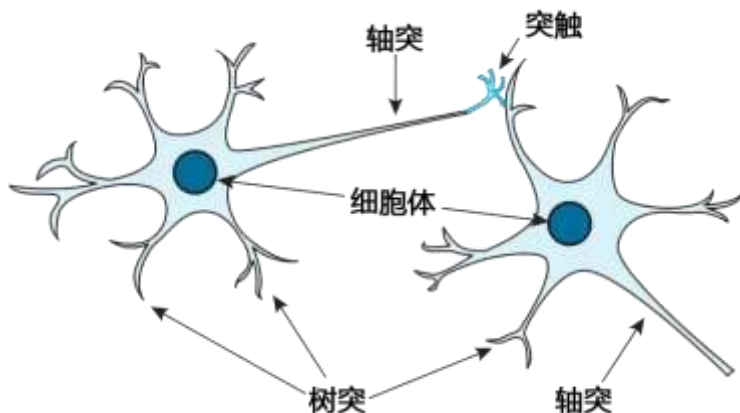
常用激活函数

损失函数

从生物神经元到人工神经元

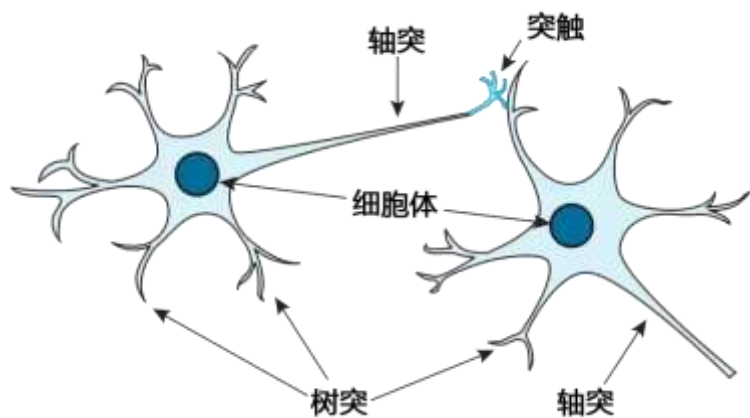
□生物神经元的特点

- 人脑大约由 140 亿个神经元组成，神经元互相连接成神经网络
- 当神经元细胞体通过轴突传到突触前膜的脉冲幅度达到一定强度，即超过其阈值电位后，突触前膜将向突触间隙释放神经传递的化学物质
- 突触有两种：兴奋性突触和抑制性突触



从生物神经元到人工神经元

□ 1943年，美国神经生物学家沃伦·麦卡洛克和数学家沃尔特·皮茨对生物神经元进行建模，首次提出人工神经元模型（M-P模型）



生物神经元

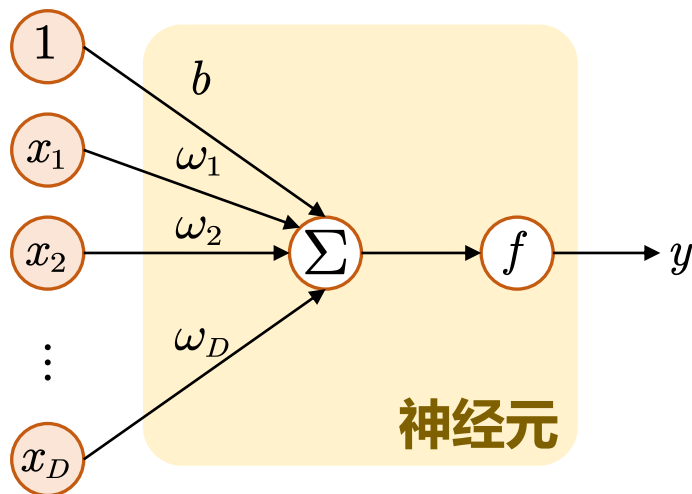


人工神经元

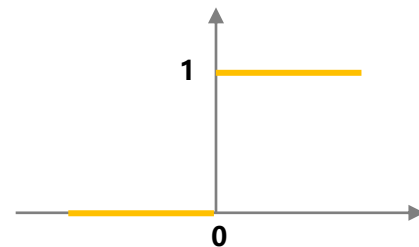
基本人工神经元结构

- M-P模型具有线性运算能力，能够简单处理一些分类问题，模型权重需先给定，不用学习，激活函数是比较简单的阈值函数

输入 权重 求和 激活函数 输出 阶跃函数实现激活功能



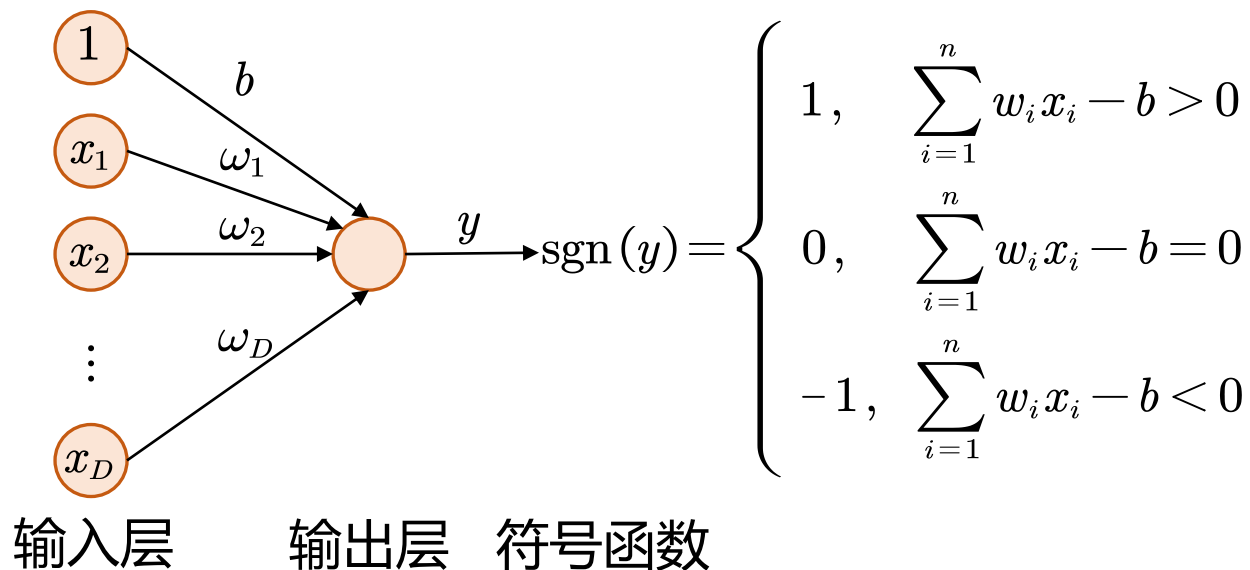
10	→	1
5	→	1
-1	→	0
-8	→	0



$$w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b = \sum_{i=1}^n w_i x_i + b$$
$$y = f\left(\sum_{i=1}^n w_i x_i + b\right) = \begin{cases} 0, & \sum_{i=1}^n w_i x_i < b \\ 1, & \sum_{i=1}^n w_i x_i \geq b \end{cases}$$

感知机 (Perceptron)

- 1957年, Frank Roseblatt 最早提出可以模拟人类感知能力的神经网络模型, 称之为感知机, 感知机是神经网络的起源, 引发了神经网络第一次高潮
- 由两层神经网络组成, 输入层接收外界输入信号, 输出层是 MP 神经元
- 权重和偏置可以通过 “学习” 得到



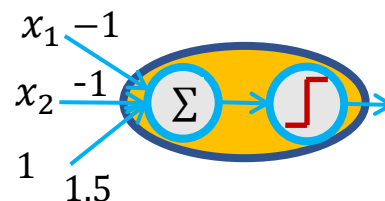
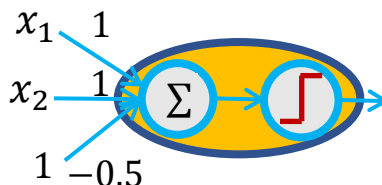
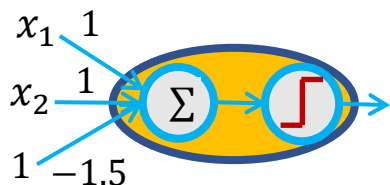
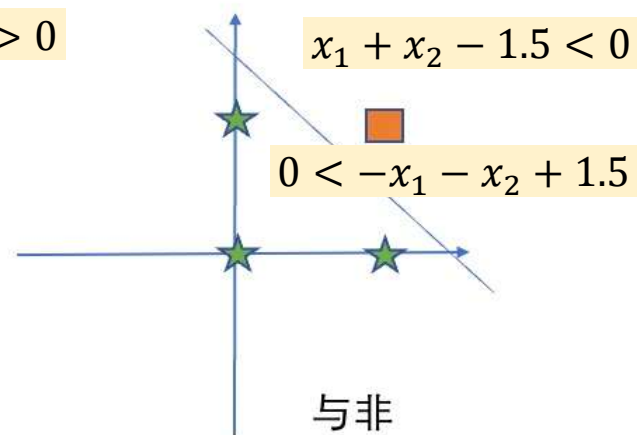
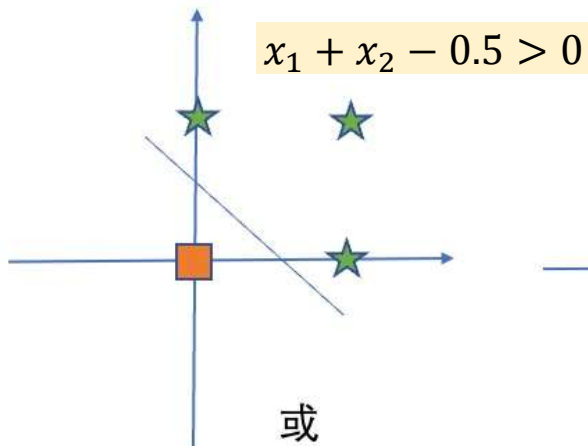
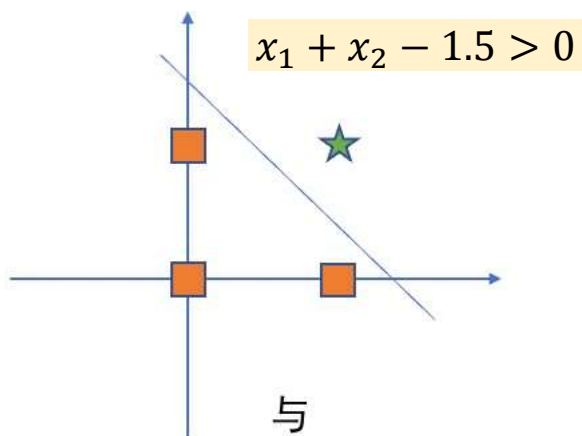
Frank Roseblatt 7

感知机解决线性可分问题

x_1	x_2	x_1 AND x_2
0	0	0
0	1	0
1	0	0
1	1	1

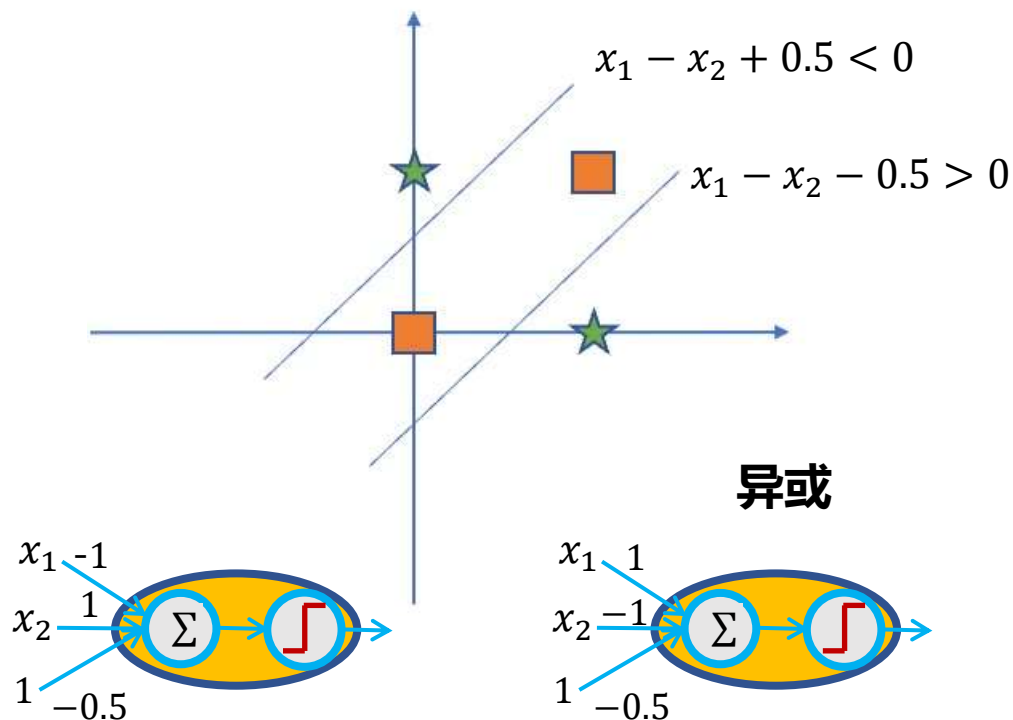
x_1	x_2	x_1 OR x_2
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	x_1 NAND x_2
0	0	1
0	1	1
1	0	1
1	1	0



线性不可分问题

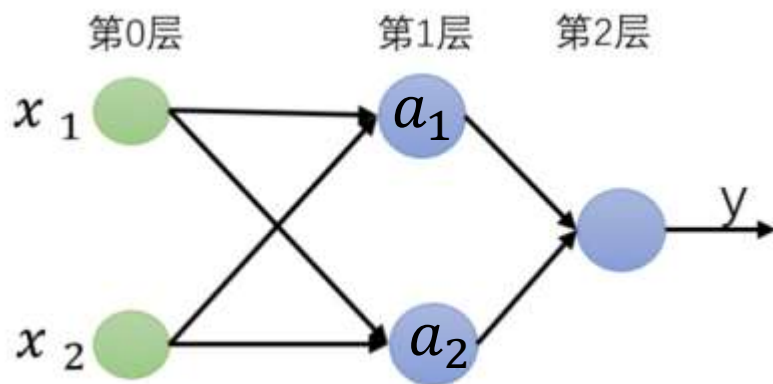
x_1	x_2	$x \text{ XOR } y$
0	0	0
0	1	1
1	0	1
1	1	0



感知机无法解决异或问题!!!

两层感知机解决异或问题

- 可以通过增加层数来解决单层感知机无法实现异或电路的问题
- 单层感知机只能对线性可分的数据进行分类，而多层感知机可以表示非线性空间，理论上具备解决任何分类问题的能力



真值表

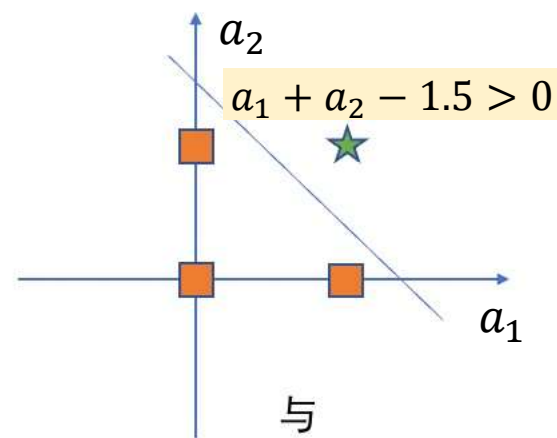
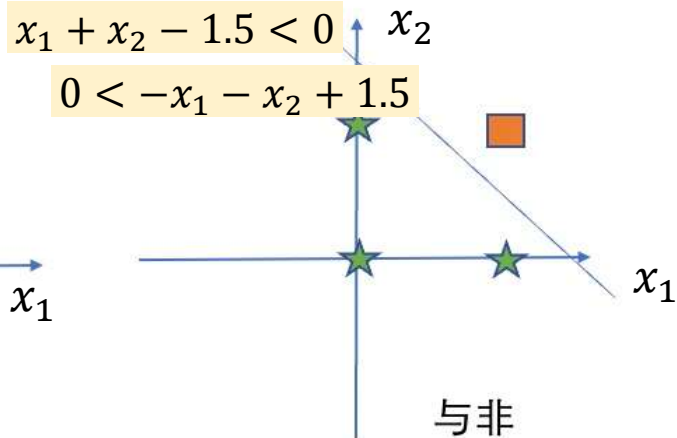
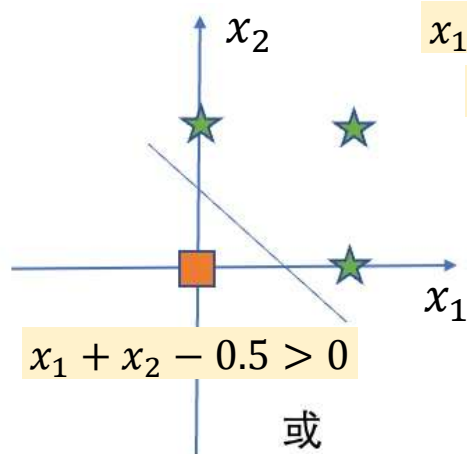
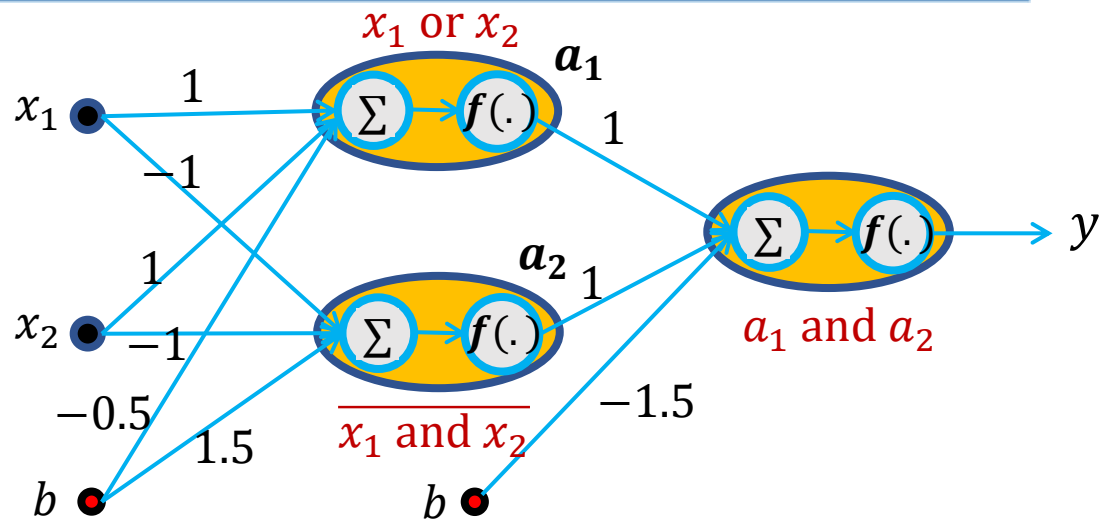
x_1	x_2	a_1	a_2	y
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0

异或运算真值表

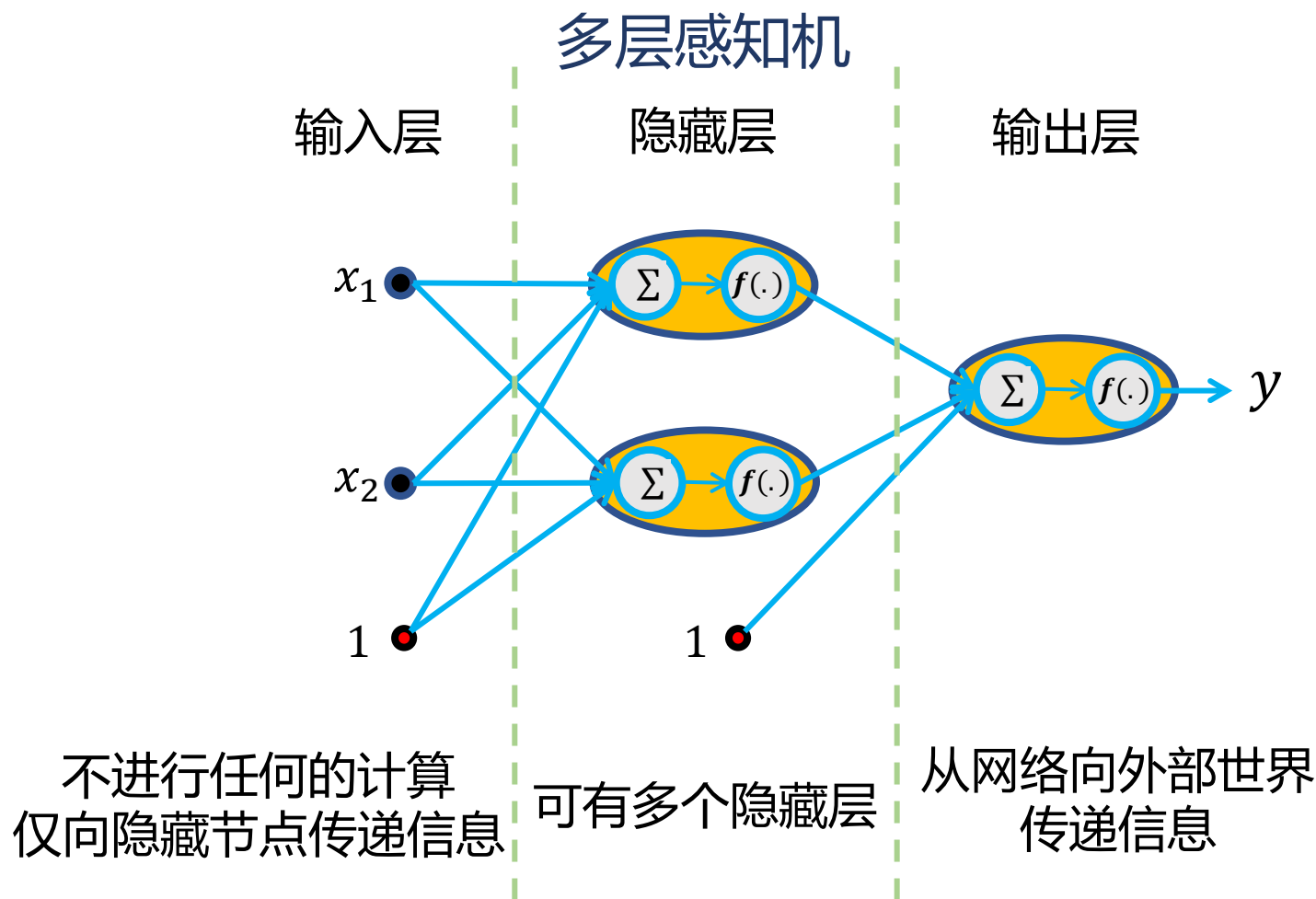
x_1	x_2	$x \text{ XOR } y$
0	0	0
0	1	1
1	0	1
1	1	0

两层感知机解决异或问题

x_1	x_2	a_1	a_2	y
0	0	0	1	0
0	1	1	1	1
1	0	1	1	1
1	1	1	0	0



前馈神经网络 (Feed Forward Neural Network)



信息从输入到输出流动，没有循环或回路

多层感知机 (Multilayer Perceptron)

□ 多层感知机模型是一种**前馈神经网络**

➤ 前馈：信息从输入层到输出层单向流动

□ 多层感知机模型是一个**有监督学习模型**

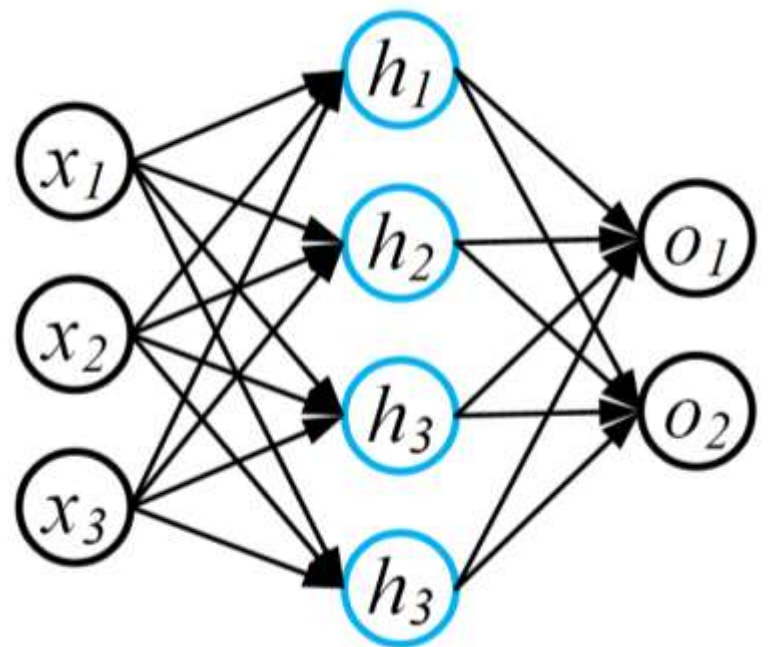
➤ 所有的参数 θ 就是各个层之间的连接权重以及偏置

➤ 通过在训练集中寻找最优参数 $\hat{\theta}$ ，从而得到函数 $f(x, \theta)$ 的最优近似 $f(x, \hat{\theta})$

□ 将含有一层隐含层的感知机网络推广到多层网络结构，数学表达式为：

$$f(x) = f^{(n)}(f^{(n-1)}(\cdots f^1(x)))$$

深度神经网络



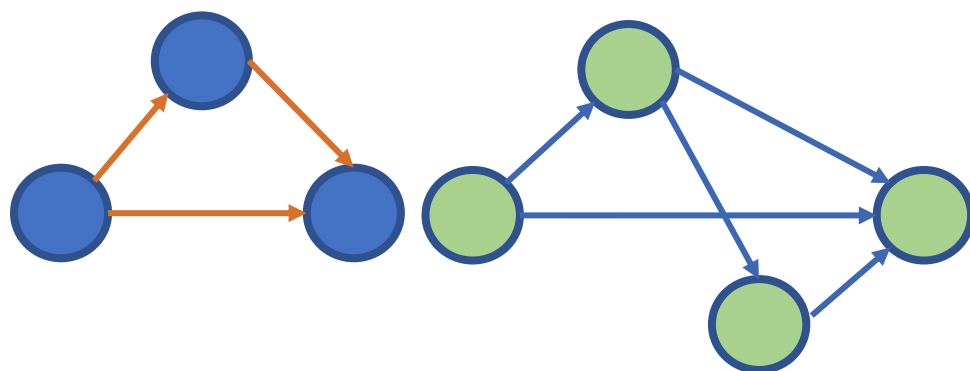
输入层

隐藏层

输出层

“深度”神经网络

深度 ≥ 2



Depth = 2

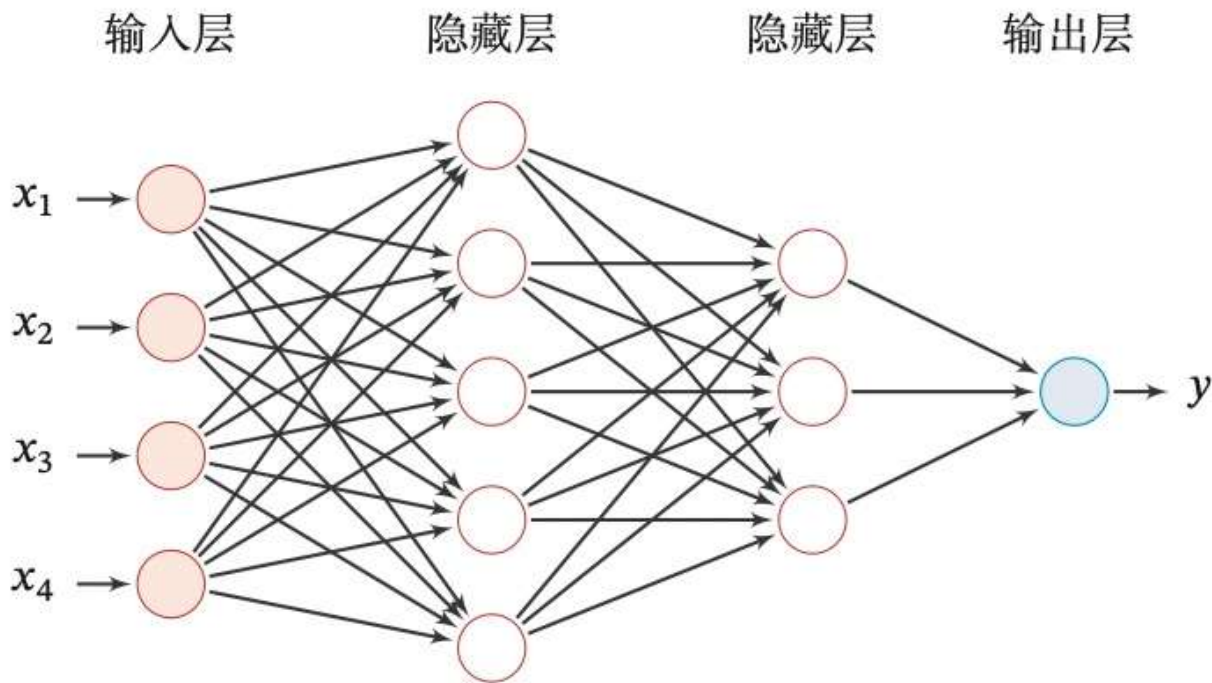
Depth = 3

要求

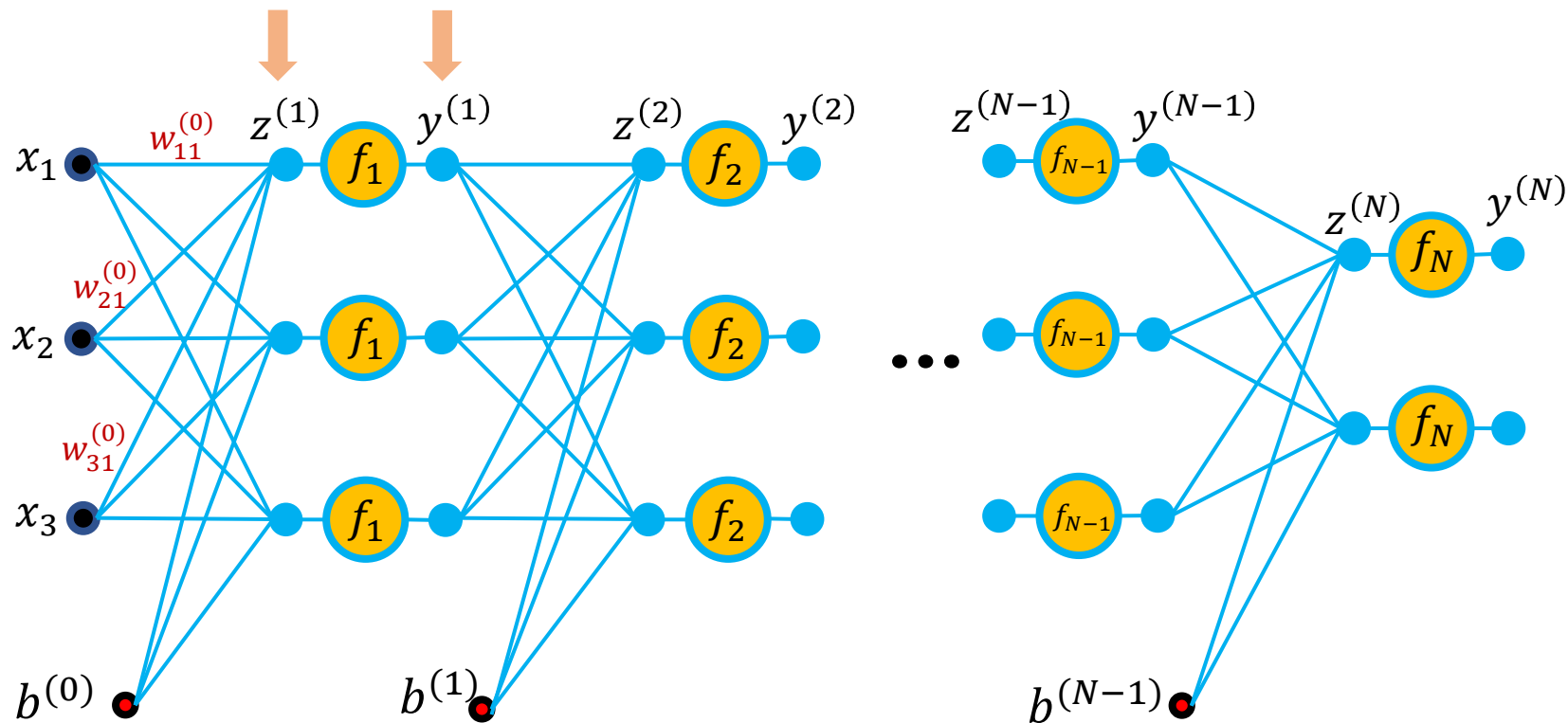
- 每一层捕获不同特征
- 不丢失信息

前向传播 (Forward Propagation)

- 从输入经过一层层隐层计算最后得到输出的过程即为前向传播
- 在前馈神经网络中，每一层的神经元可以接收前一层神经元的信号，并产生信号输出到下一层，整个网络中无反馈，信号从输入层向输出层单向传播



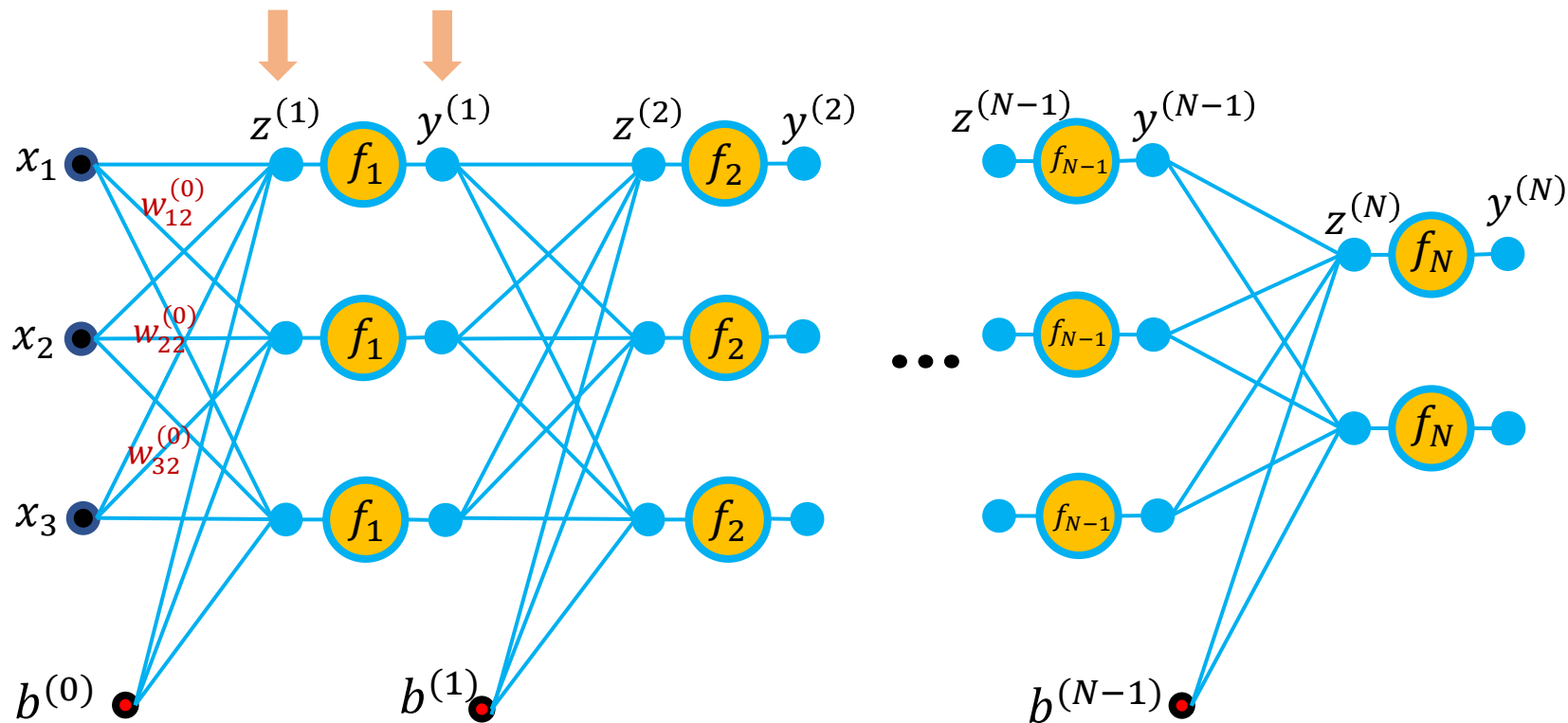
前向传播示例



$$z_j^{(1)} = \sum_i w_{ij}^{(0)} x_i + b_j^{(0)}$$

$$y_j^{(1)} = f_1(z_j^{(1)})$$

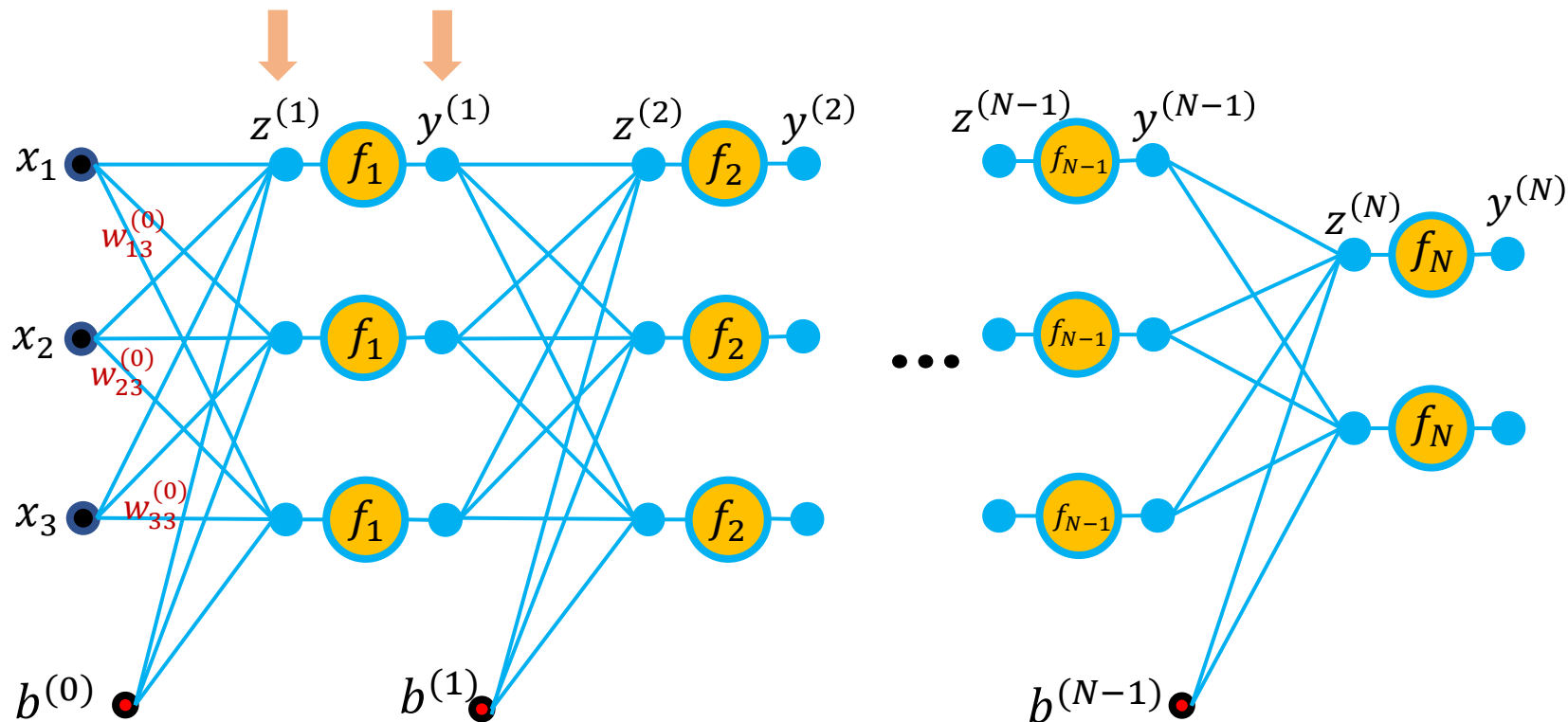
前向传播示例



$$z_j^{(1)} = \sum_i w_{ij}^{(0)} x_i + b_j^{(0)}$$

$$y_j^{(1)} = f_1(z_j^{(1)})$$

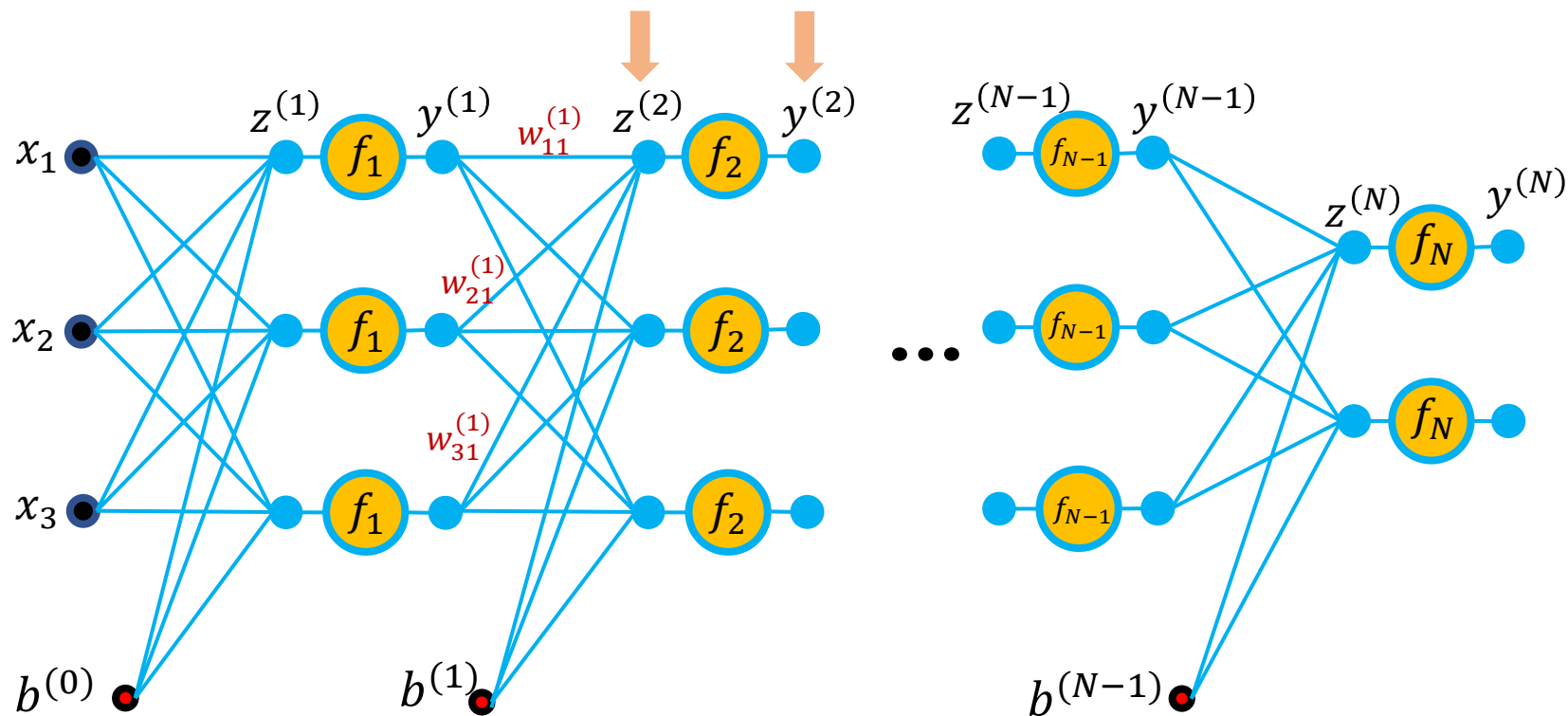
前向传播示例



$$z_j^{(1)} = \sum_i w_{ij}^{(0)} x_i + b_j^{(0)}$$

$$y_j^{(1)} = f_1(z_j^{(1)})$$

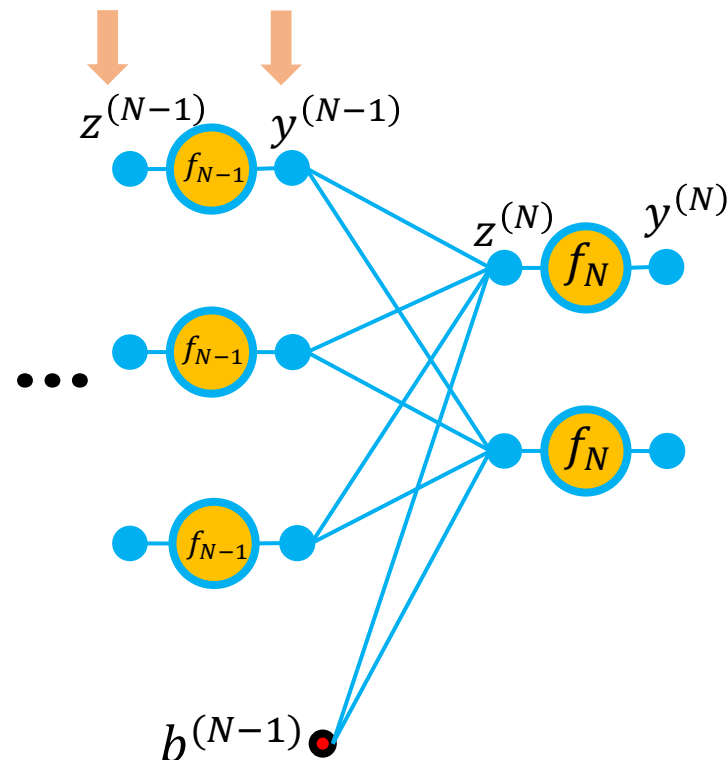
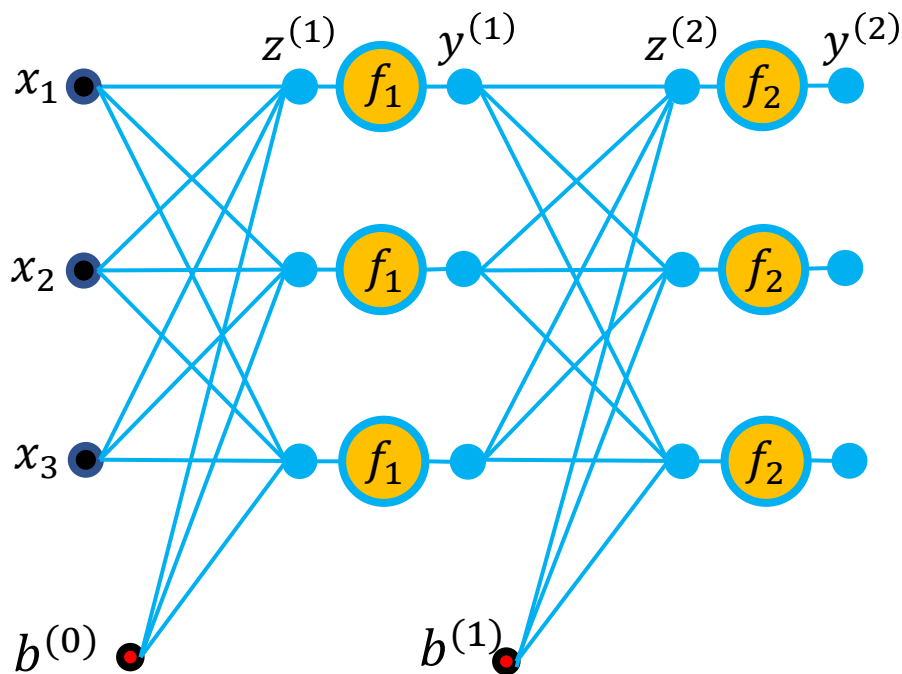
前向传播示例



$$z_j^{(1)} = \sum_i w_{ij}^{(0)} x_i + b_j^{(0)} \quad z_j^{(2)} = \sum_i w_{ij}^{(1)} y_i^{(1)} + b_j^{(1)}$$

$$y_j^{(1)} = f_1(z_j^{(1)}) \quad y_j^{(2)} = f_2(z_j^{(2)})$$

前向传播示例



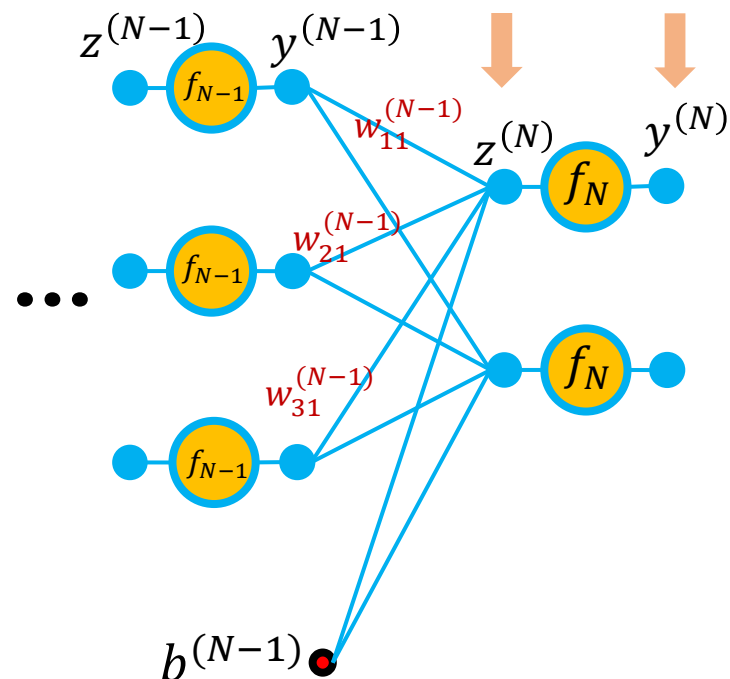
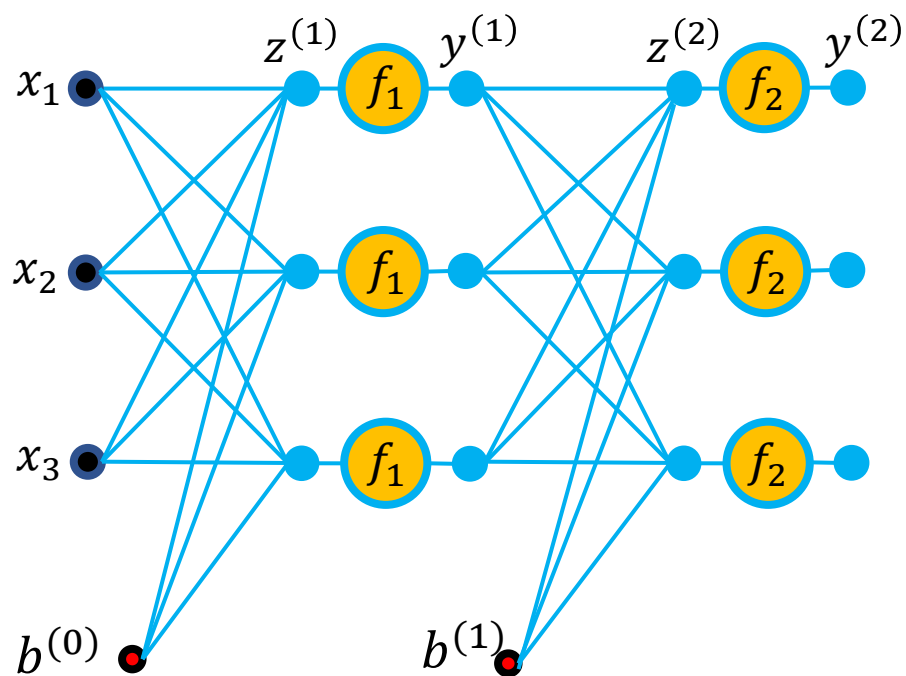
$$z_j^{(1)} = \sum_i w_{ij}^{(0)} x_i + b_j^{(0)} \quad z_j^{(2)} = \sum_i w_{ij}^{(1)} y_i^{(1)} + b_j^{(1)} \quad z_j^{(N-1)} = \sum_i w_{ij}^{(N-2)} y_i^{(N-2)} + b_j^{(N-2)}$$

$$y_j^{(1)} = f_1(z_j^{(1)})$$

$$y_j^{(2)} = f_2(z_j^{(2)})$$

$$y_j^{(N-1)} = f_{N-1}(z_j^{(N-1)})$$

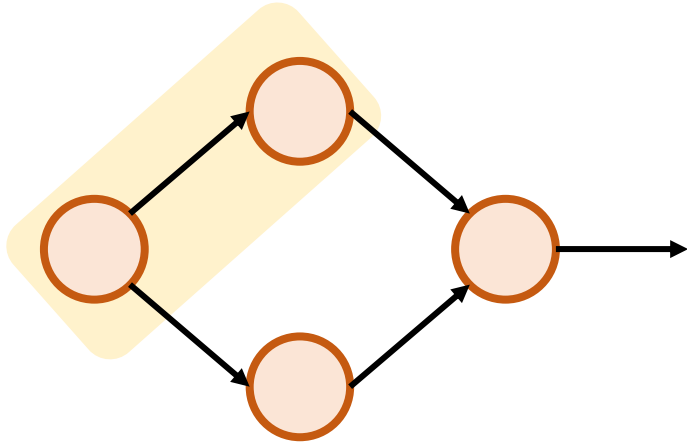
前向传播示例



$$z_j^{(N)} = \sum_i w_{ij}^{(N-1)} y_i^{(N-1)} + b_j^{(N-1)}$$

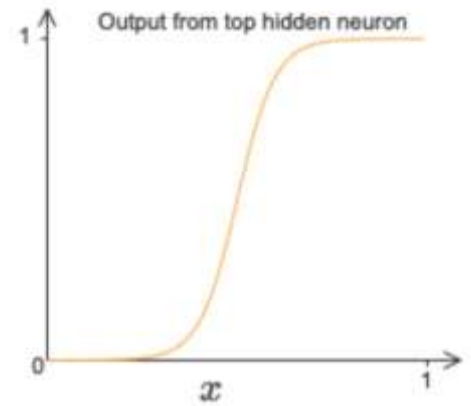
$$y_j^{(N)} = f_N(z_j^{(N)})$$

考虑一个简单的两层神经网络

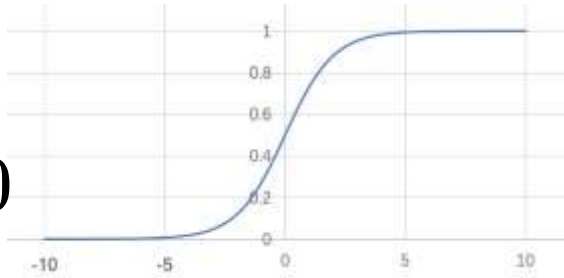


$$z = wx + b$$

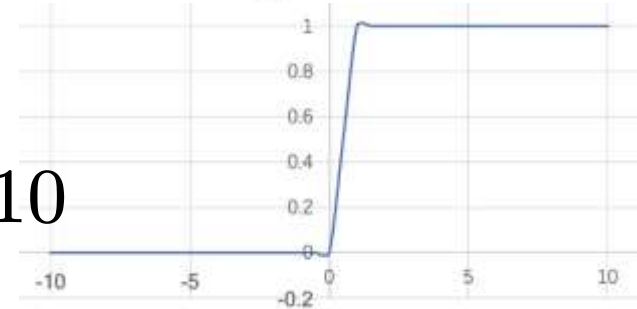
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



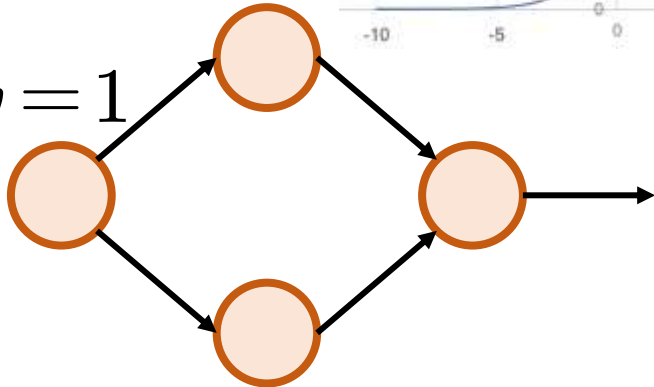
$$b = 0$$



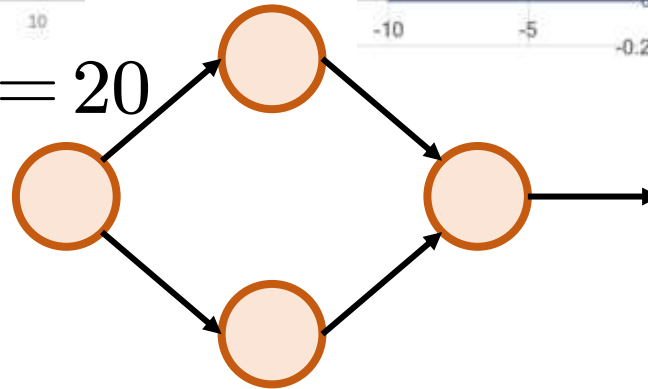
$$b = -10$$



$$w = 1$$

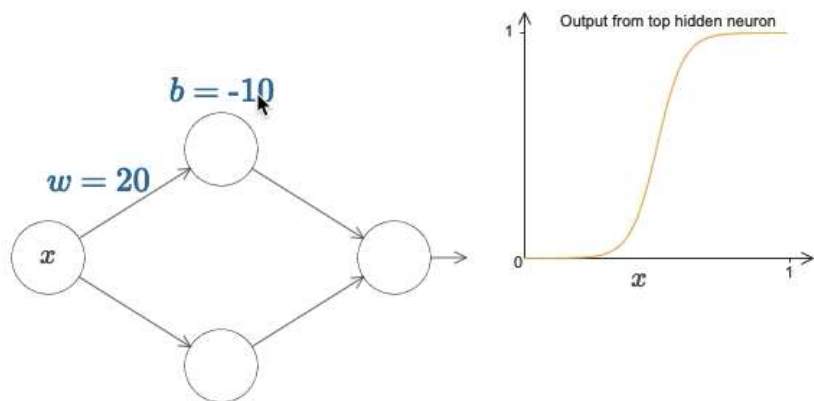


$$w = 20$$

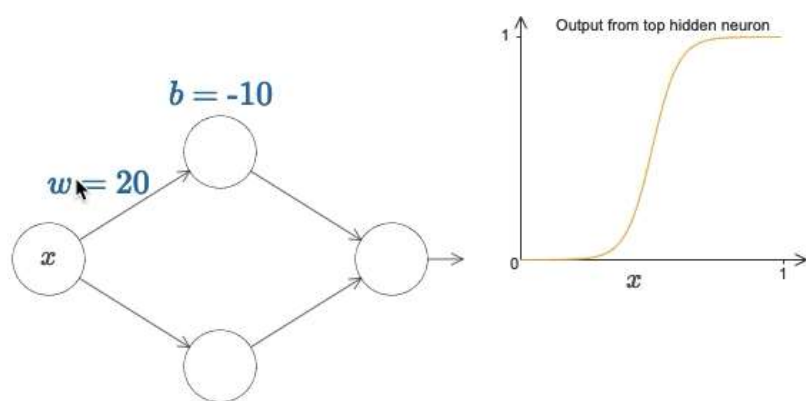


Refer: A visual proof that neural nets can compute any function
<http://neuralnetworksanddeeplearning.com/chap4.html>

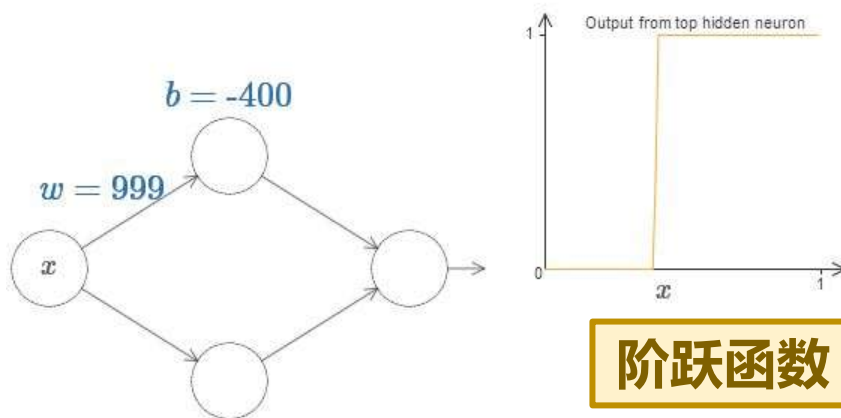
改变权重和偏置的影响



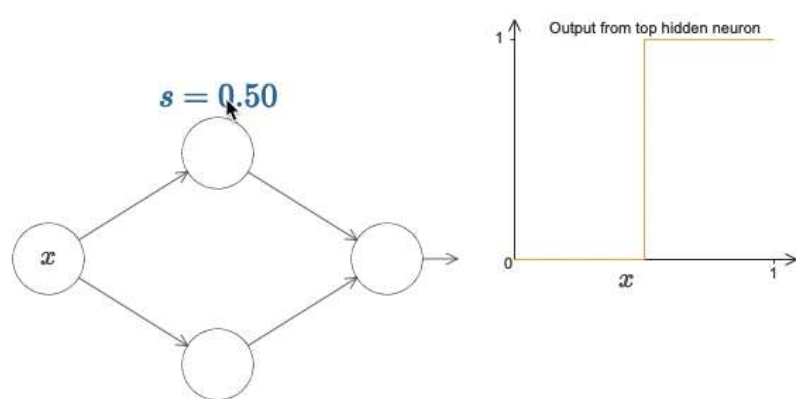
改变偏置



改变权重

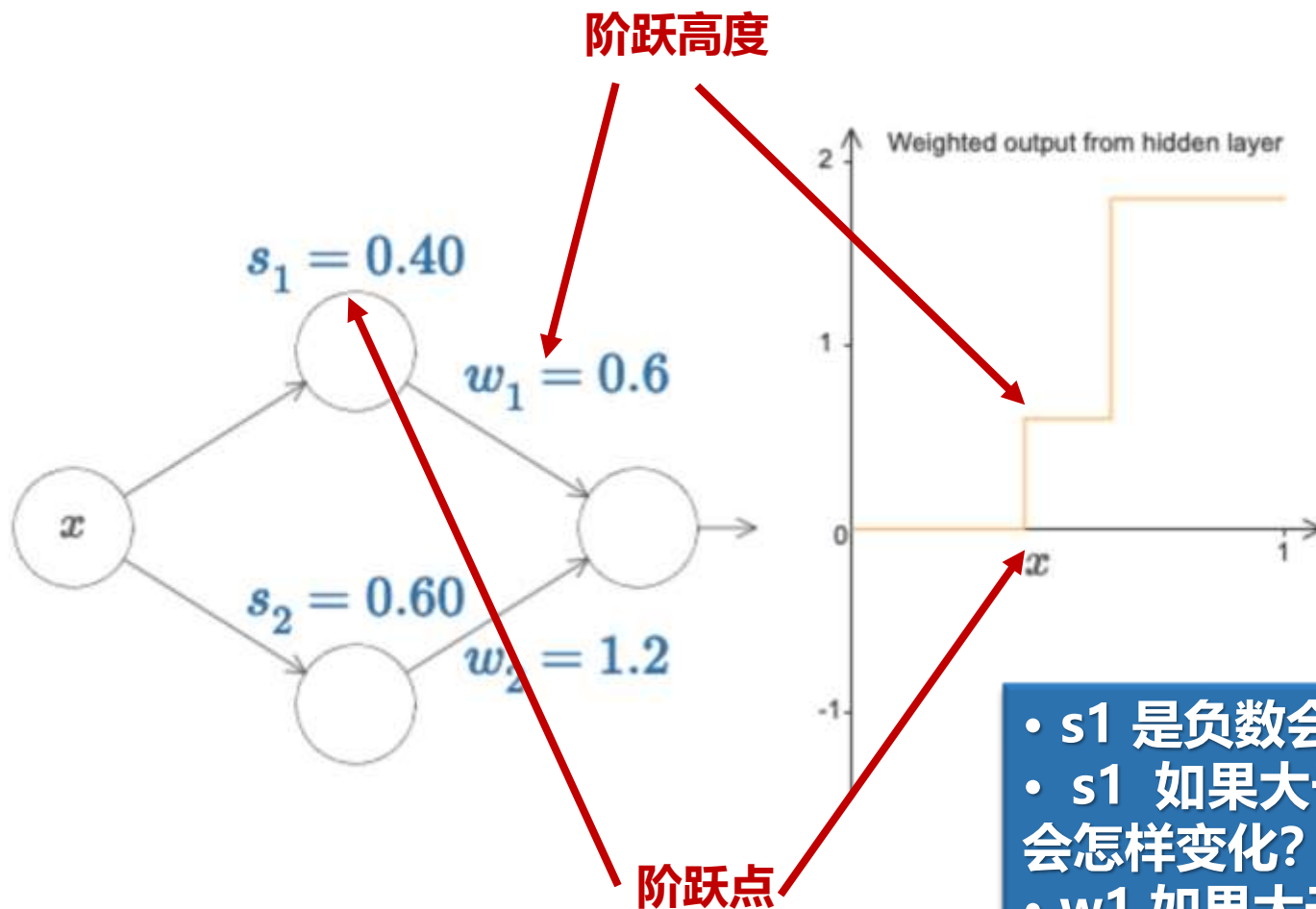


阶跃函数



$$s = -\frac{b}{w} \text{ 是阶跃函数的阈值}$$

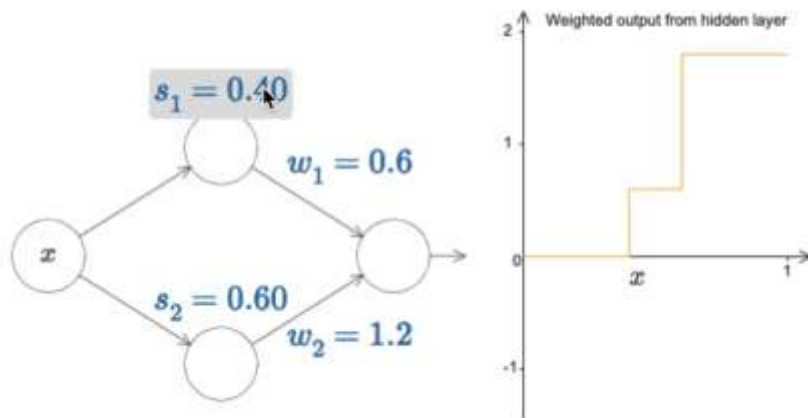
隐藏层到输出层的参数如何影响输出？



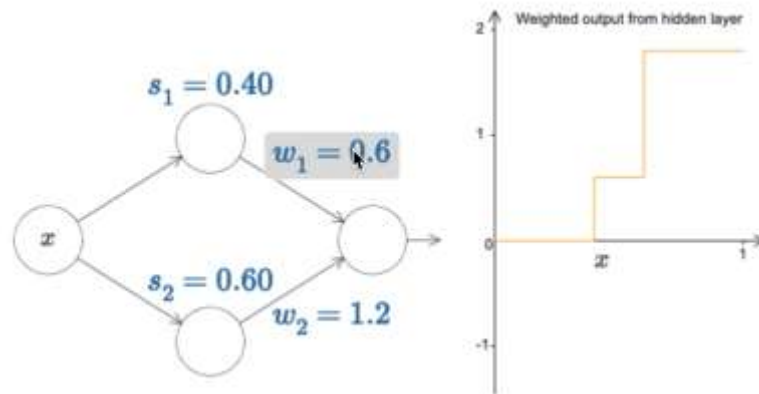
- s_1 是负数会如何？
- s_1 如果大于 s_2 ，那么图形会怎样变化？
- w_1 如果大于 w_2 会如何？
- w_1 是负数会如何？

隐藏层神经元的相互作用

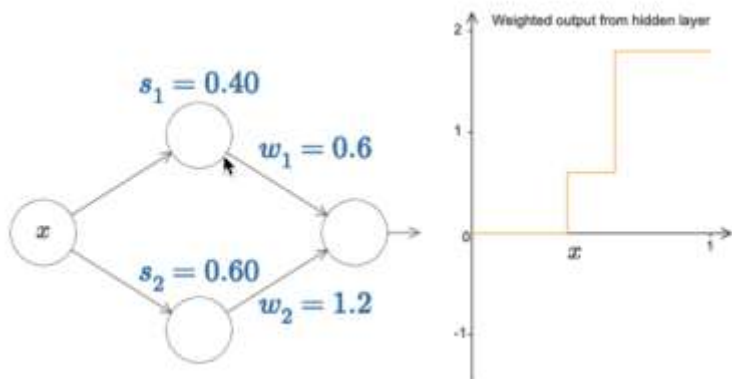
s_1 变为负数



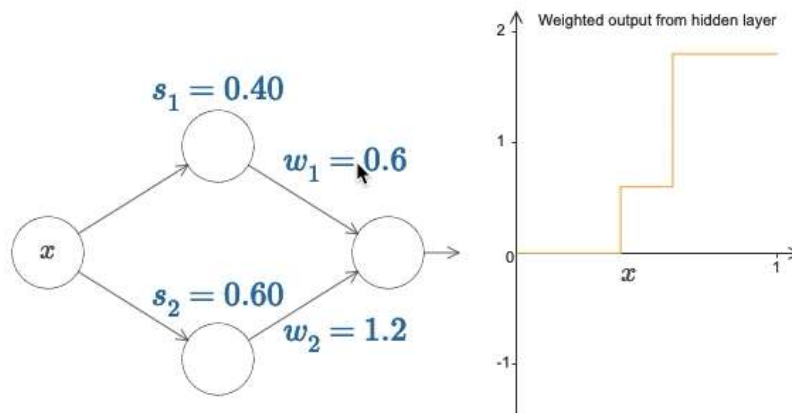
w_1 大于 w_2



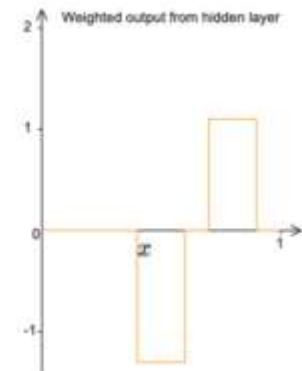
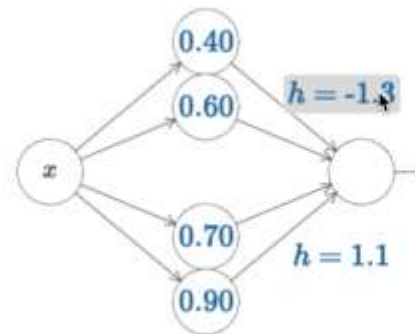
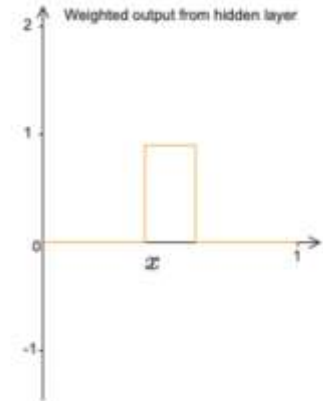
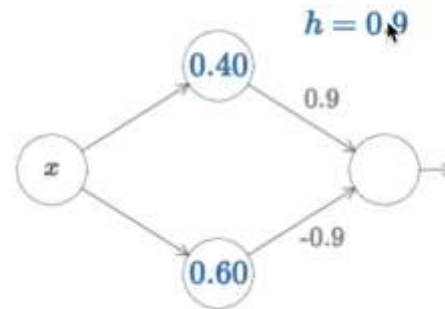
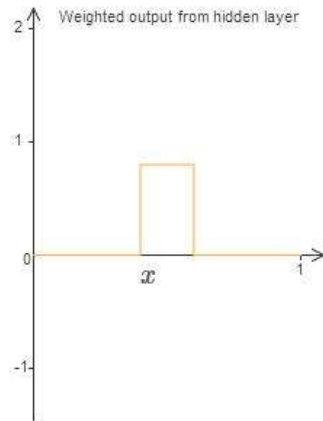
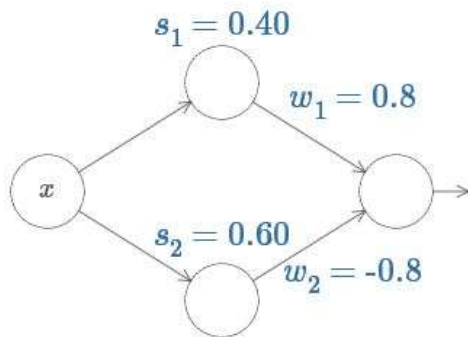
s_1 大于 s_2



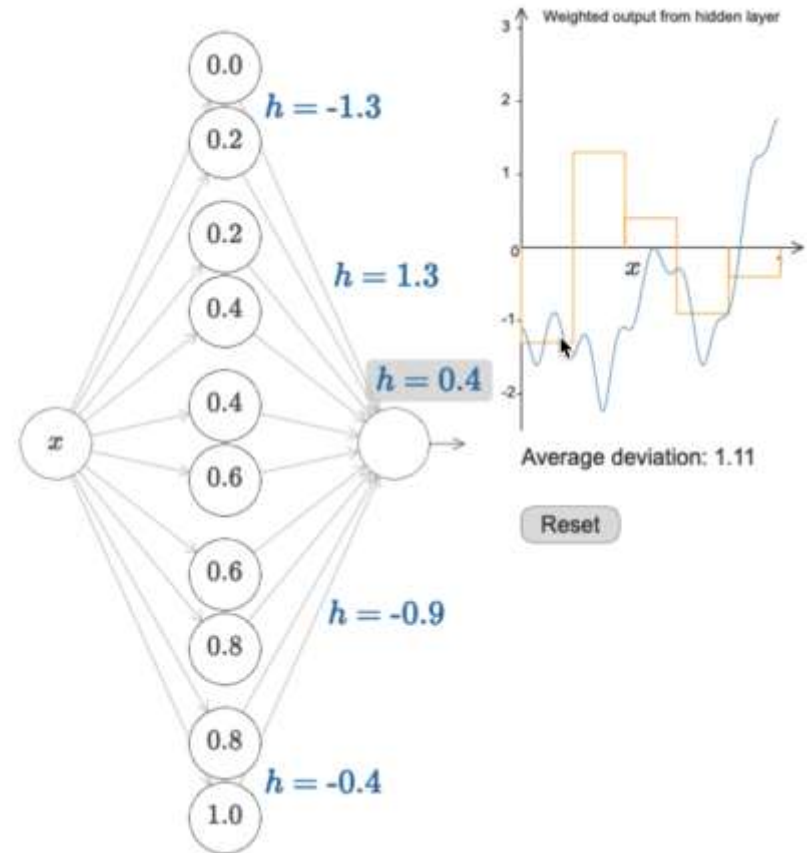
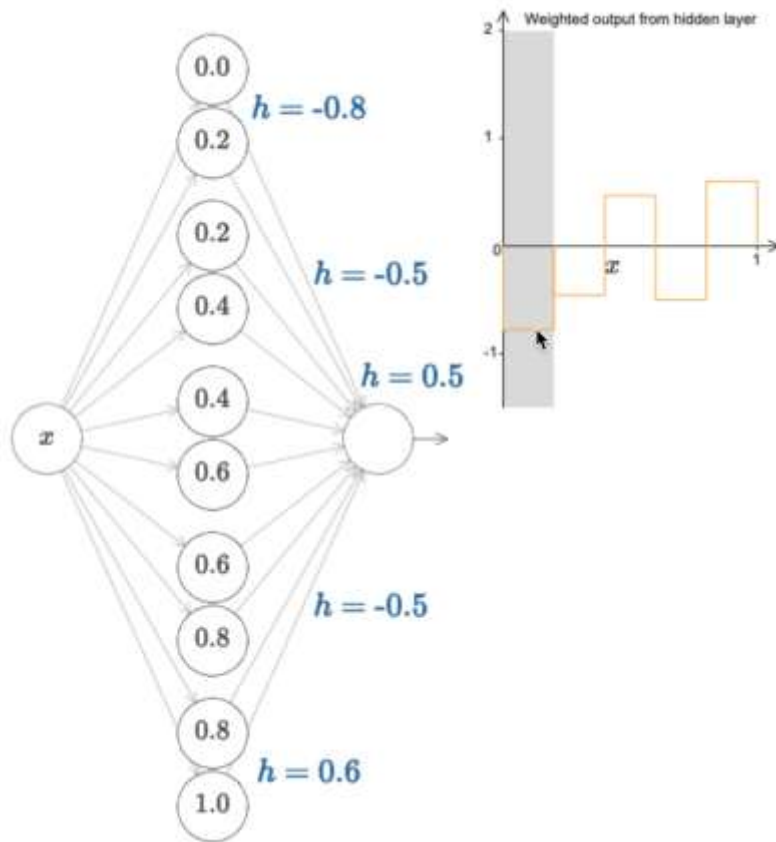
w_1 是负数



固定权重比，生成脉冲函数



固定权重比，生成脉冲函数



内容导览



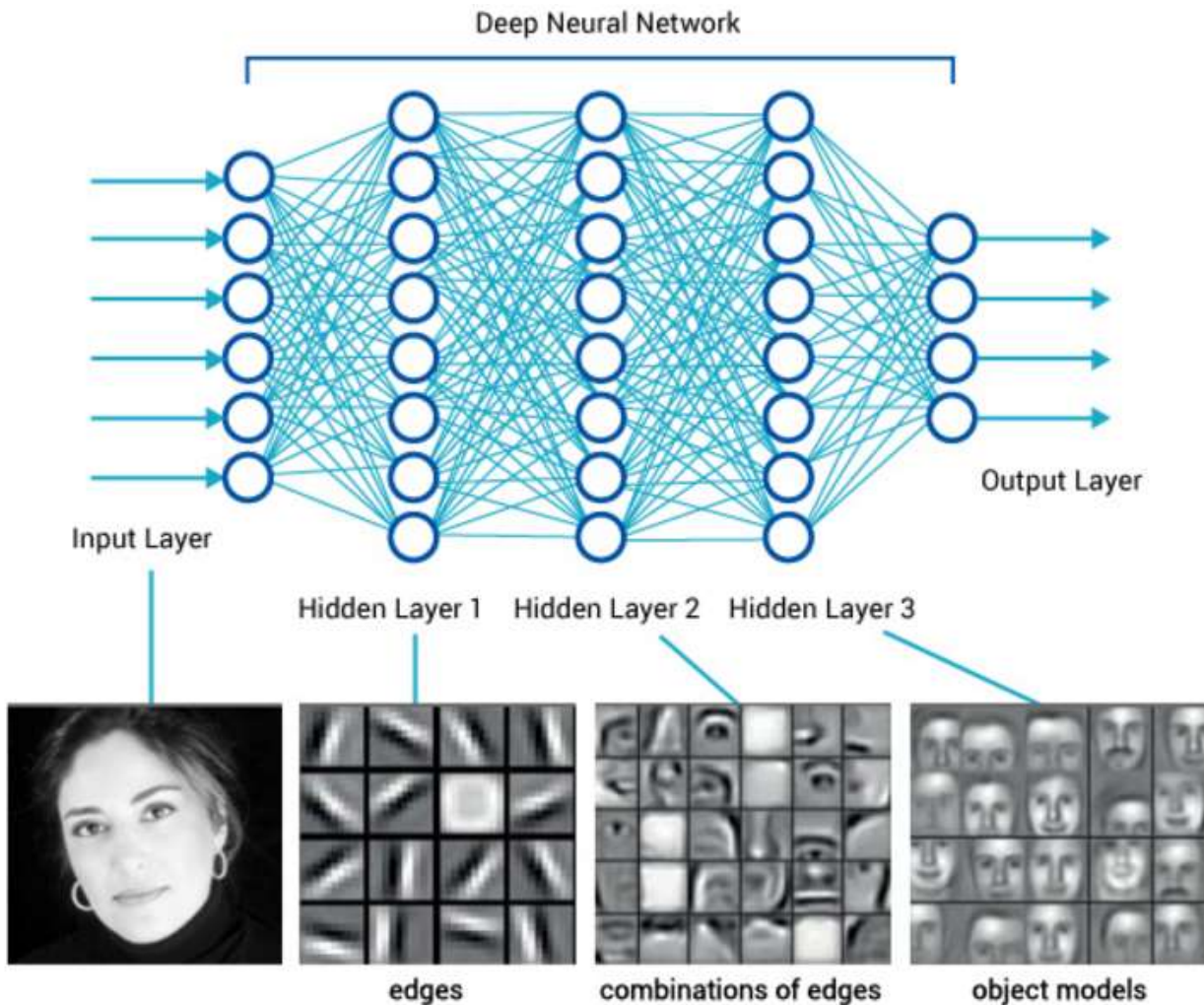
多层感知机

通用近似定理

常用激活函数

损失函数

深层前馈神经网络(MLP)



通用近似定理

定理 4.1 – 通用近似定理 (Universal Approximation Theorem)

[Cybenko, 1989, Hornik et al., 1989]: 令 $\varphi(\cdot)$ 是一个非常数、有界、单调递增的连续函数, \mathcal{I}_d 是一个 d 维的单位超立方体 $[0, 1]^d$, $C(\mathcal{I}_d)$ 是定义在 \mathcal{I}_d 上的连续函数集合。对于任何一个函数 $f \in C(\mathcal{I}_d)$, 存在一个整数 m , 和一组实数 $v_i, b_i \in \mathbb{R}$ 以及实数向量 $\mathbf{w}_i \in \mathbb{R}^d$, $i = 1, \dots, m$, 以至于我们可以定义函数

$$F(\mathbf{x}) = \sum_{i=1}^m v_i \varphi(\mathbf{w}_i^T \mathbf{x} + b_i), \quad (4.33)$$

作为函数 f 的近似实现, 即

$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon, \forall \mathbf{x} \in \mathcal{I}_d. \quad (4.34)$$

其中 $\epsilon > 0$ 是一个很小的正数。

根据通用近似定理, 对于具有**线性输出层和至少一个使用“挤压”性质的激活函数的隐藏层组成的前馈神经网络**, **只要其隐藏层神经元的数量足够, 它可以以任意的精度来近似任何从一个定义在实数空间中的有界闭集函数。**

应用到机器学习

- 神经网络可以作为一个“万能”函数来使用，可以用来进行复杂的特征转换，或逼近一个复杂的条件分布

$$\hat{y} = g(\varphi(\mathbf{x}), \theta)$$

分类器(一般比较简单)

神经网络(一般比较复杂)

- 如果 $g(\cdot)$ 为Logistic回归，那么Logistic回归分类器可以看成神经网络的最后一层

内容导览



多层感知机

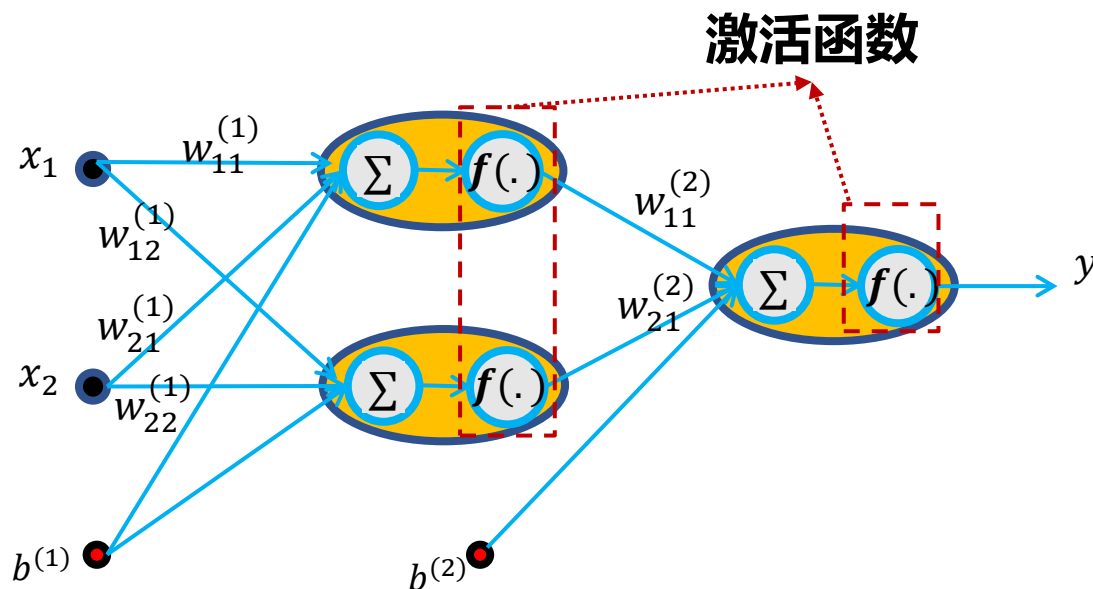
通用近似定理

常用激活函数

损失函数

激活函数的定义与作用

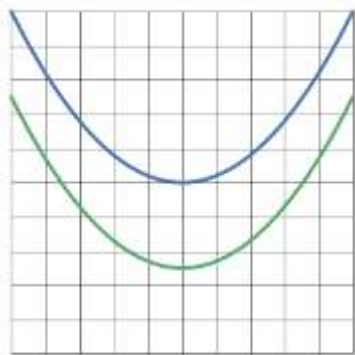
- 激活函数都是非线性函数，经过激活函数处理后，就可以将输入的数据非线性化，以此来拟合更为复杂的函数
- 如果一个网络中每个神经元的输出只是所有输入的加权和，那么无论神经网络有多少层，这个网络就只是一个线性模型，无法形成一个复杂的表达空间，为了提升网络的表达能力，需要加入**非线性映射**



激活函数的作用

□ 用线性变换跟随着非线性变化将输入空间投向另一个空间

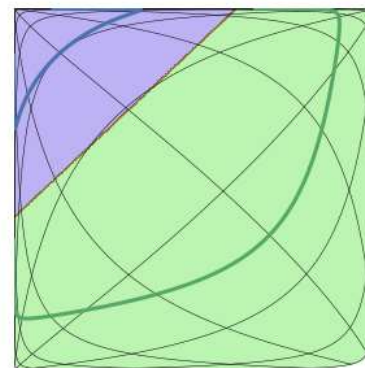
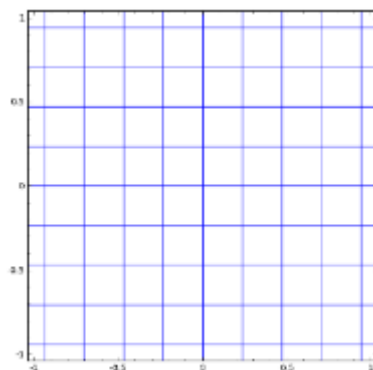
当前空间的数据



线性不可分数据



投影到另一个空间



线性可分数据



常用激活函数：Sigmoid函数

□ Sigmoid函数

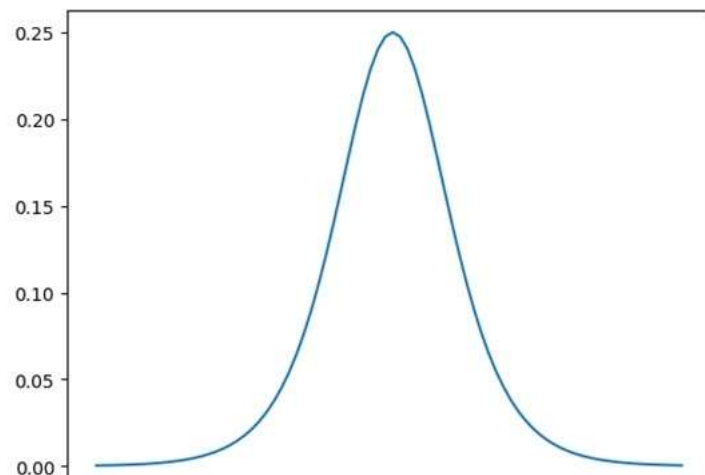
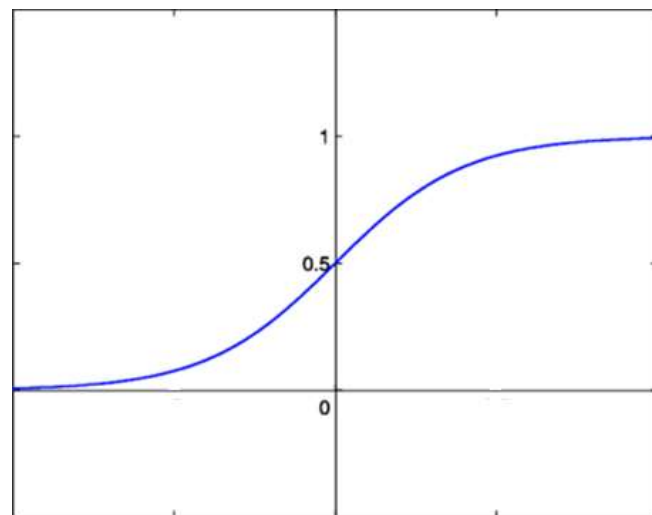
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

□ 优点

- 可解释性较好
- 输出范围有限
- 单调连续

□ 缺点

- 函数饱和，梯度小
- 输出不以0为中心
- 执行指数运算，计算机运行得较慢

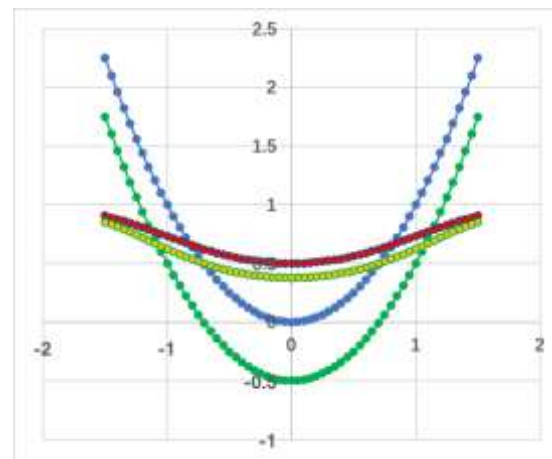
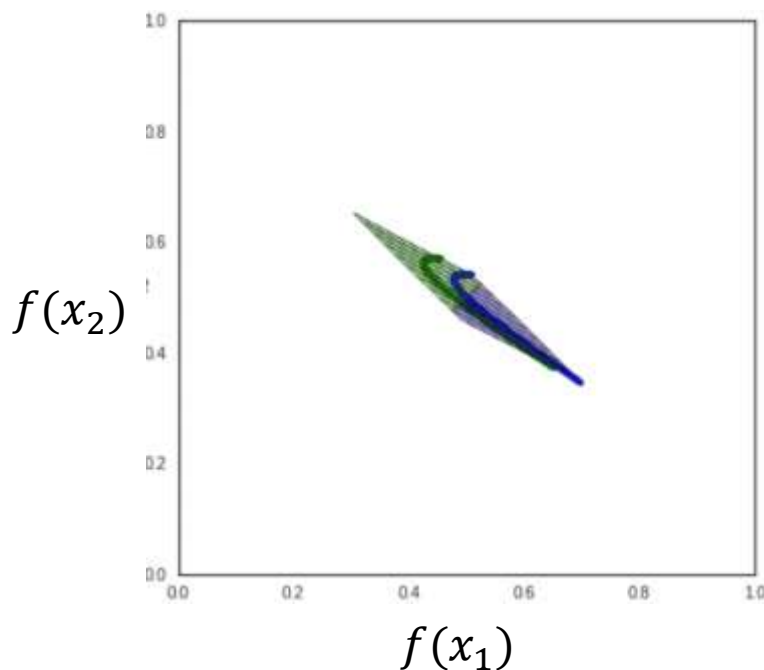


$$\sigma'(x) = \sigma(x) (1 - \sigma(x))$$

三维视角的 Sigmoid 函数

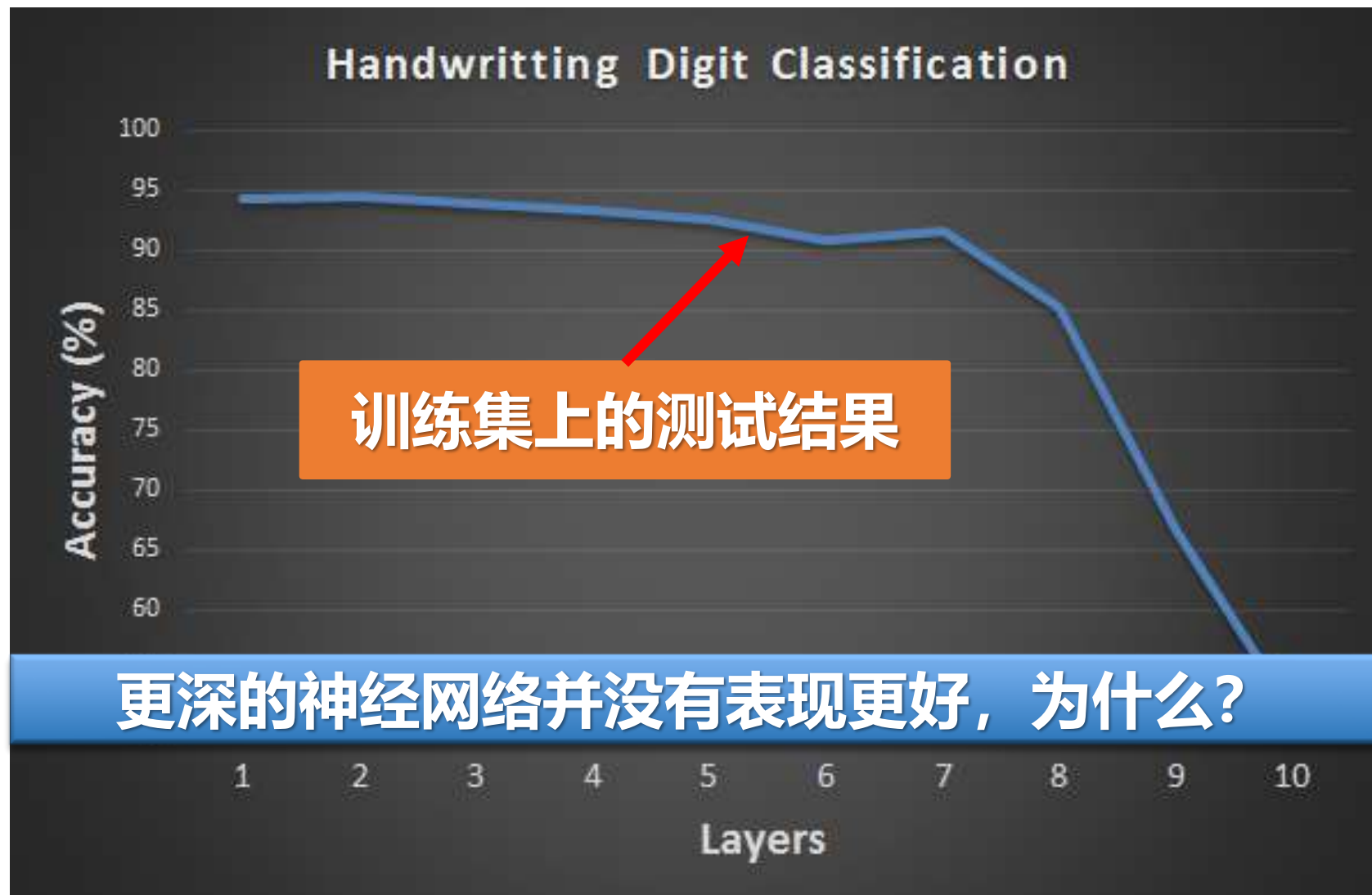
□ 横轴和纵轴都是隐藏层激活函数之前的输出

- epoch 15–40: 典型的 Sigmoid “挤压” 过程发生在横轴上
- epoch 40–65: 纵轴上特征空间转换有一个 “加宽” 的过程
- epoch 65–100: “加宽” 过程更明显
- 最终, 成功分开了两条曲线

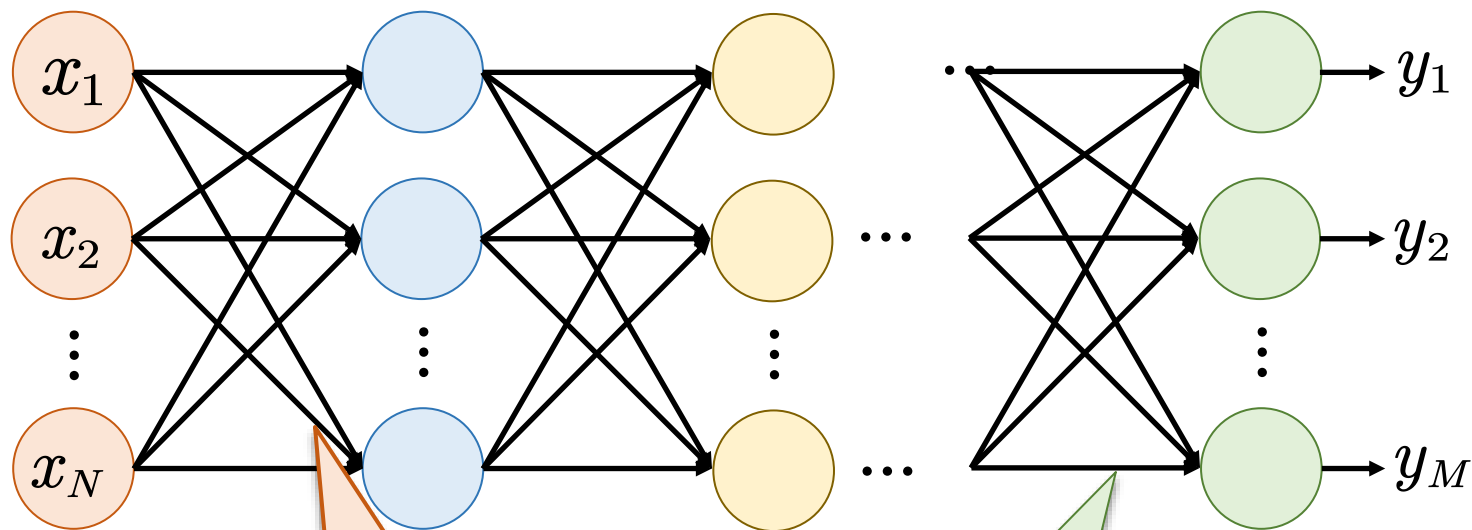


输入图像被压扁

思考



复习：梯度消失问题



梯度更小

学习缓慢

大多数随机

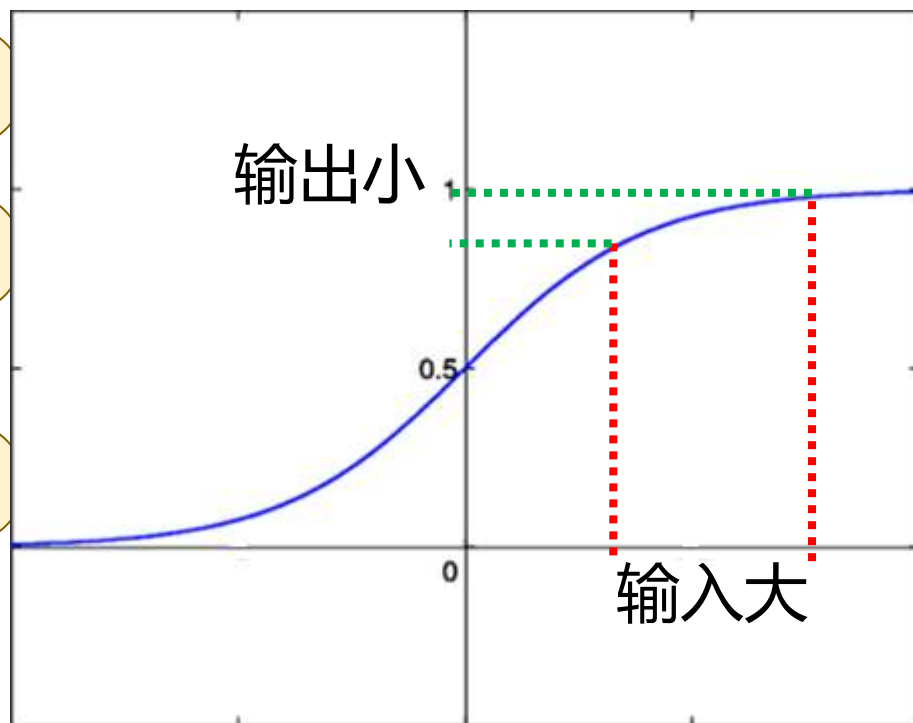
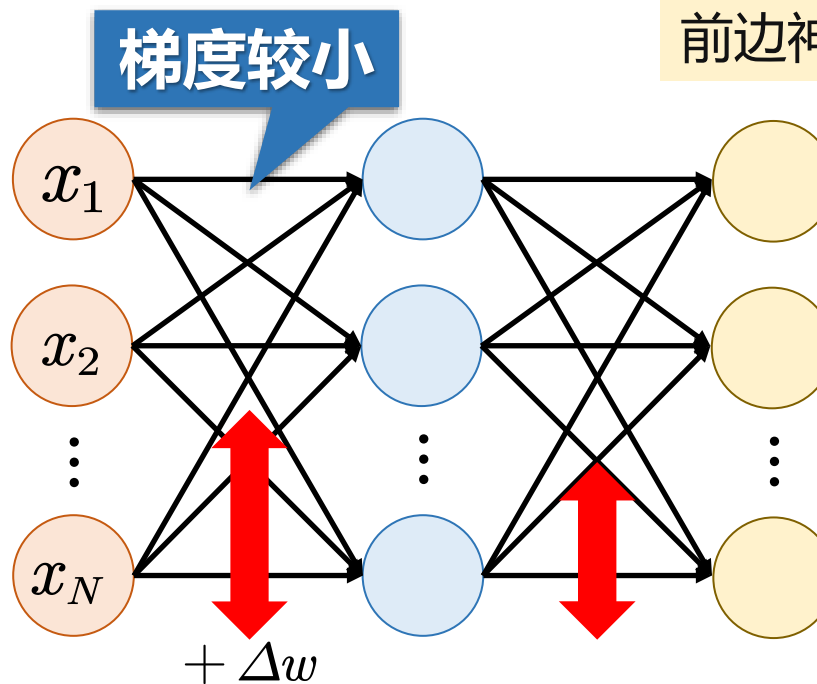
梯度更大

学习快速

已经收敛

复习：梯度消失问题

Sigmoid函数导数最大值为0.25，因链式法则需要连乘，故进行反向传播时容易导致**梯度消失**。前边神经元的权重可能得不到更改。



计算导数的直观方法：

$$\frac{\partial l}{\partial w} = ? \quad \frac{\Delta l}{\Delta w}$$

常用激活函数：Tanh函数

□Tanh函数

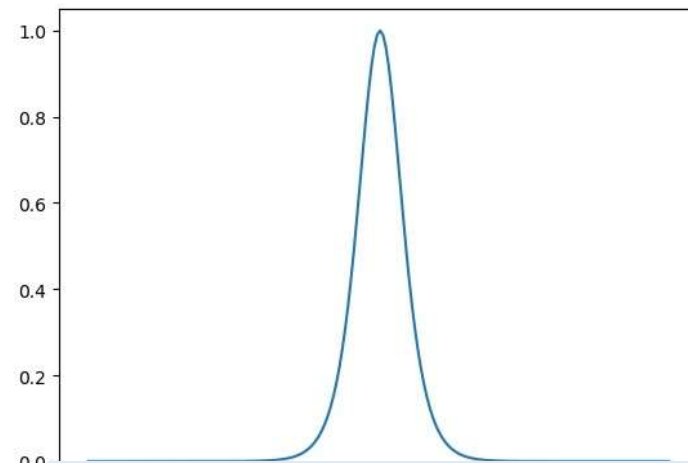
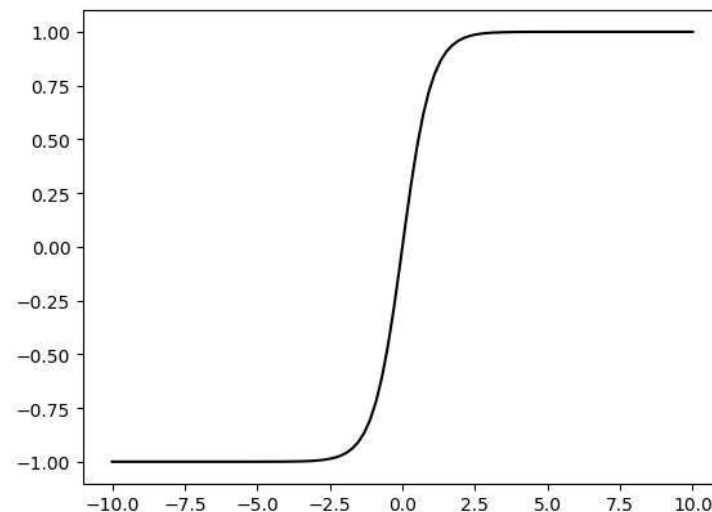
$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

□优点

- 比Sigmoid收敛速度快
- 输出以0为中心
- 负输入映射为负，零输入被映射为接近零

□缺点

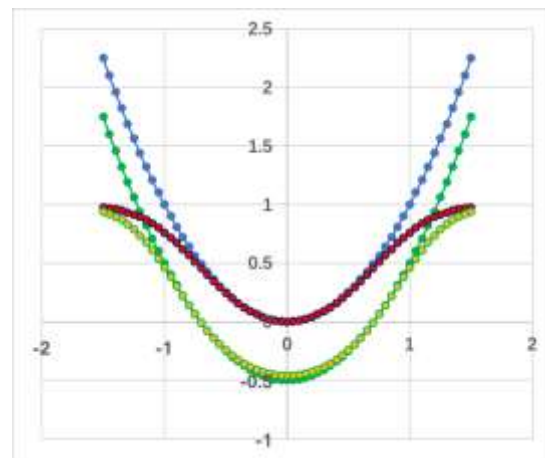
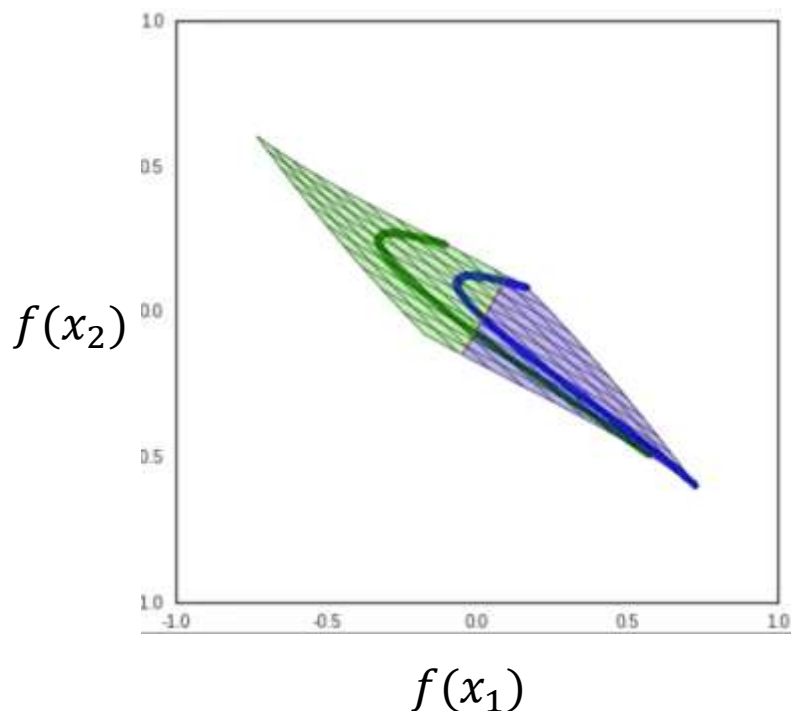
- 函数饱和，梯度仍然容易消失
- 执行指数运算，计算机运行较慢



$$\tanh'(x) = 1 - \tanh^2(x)$$

三维视角的 Tanh 函数

- epoch 10-40: 横轴上发生了Tanh “挤压” 过程，不过它不如之前Sigmoid那样明显
- epoch 40-55: 纵轴上出现了特征空间转换的“加宽”过程
- epoch 65-100: “加宽”过程更明显
- 最终，以更快的速度分开了两条线



输入图像被压扁

常用激活函数：ReLU函数

□ ReLU函数

$$f(x) = \max(0, x)$$

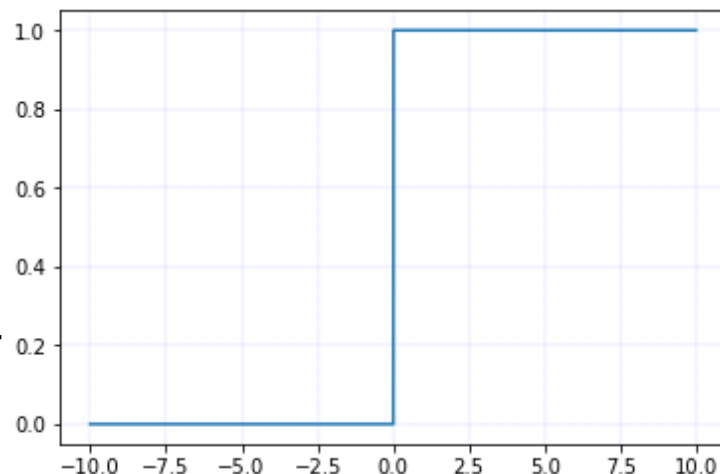
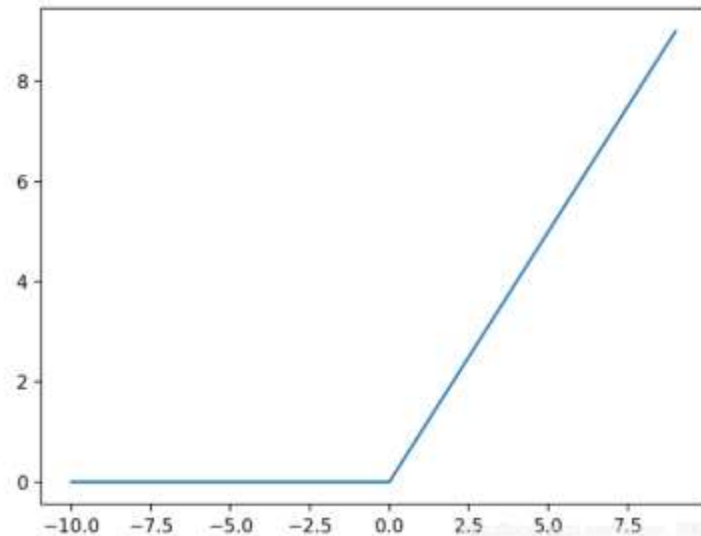
□ 优点

- 收敛速度比上述激活函数更快
- 计算简单
- **避免梯度消失问题**



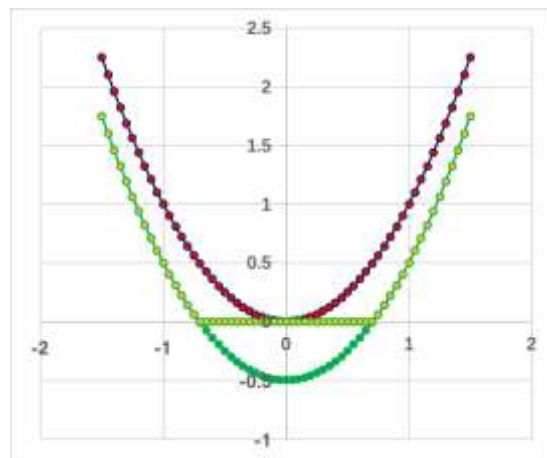
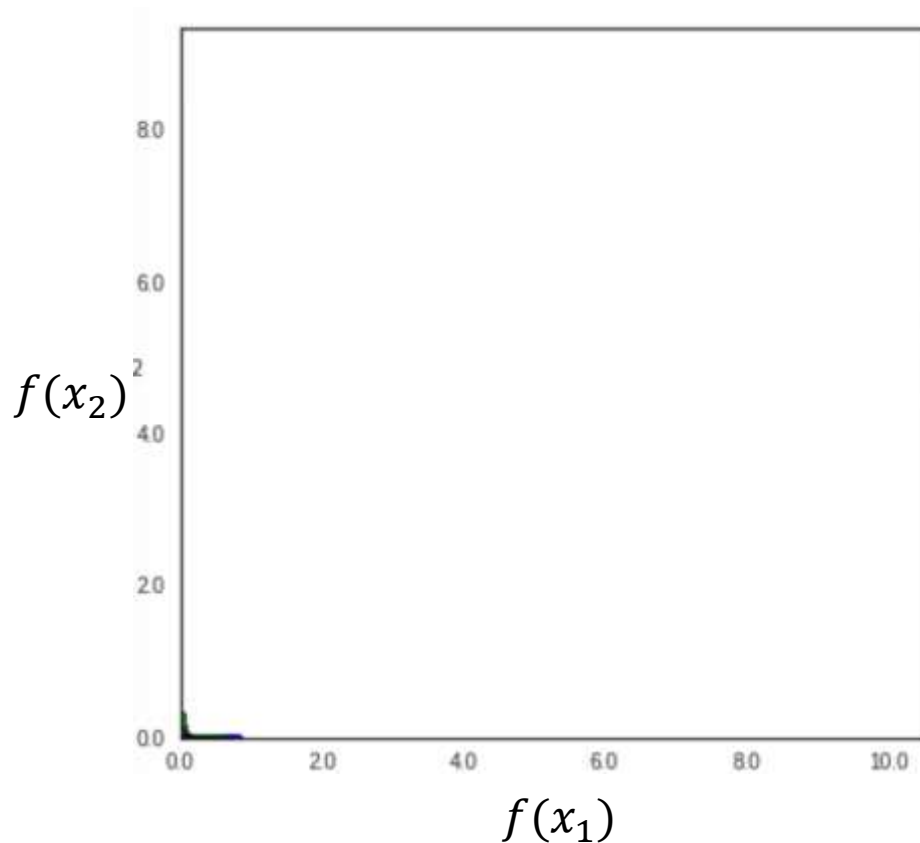
□ 缺点

- 输出不以0为中心
- 当输入小于0时，参数无法更新



三维视角的 ReLU 函数

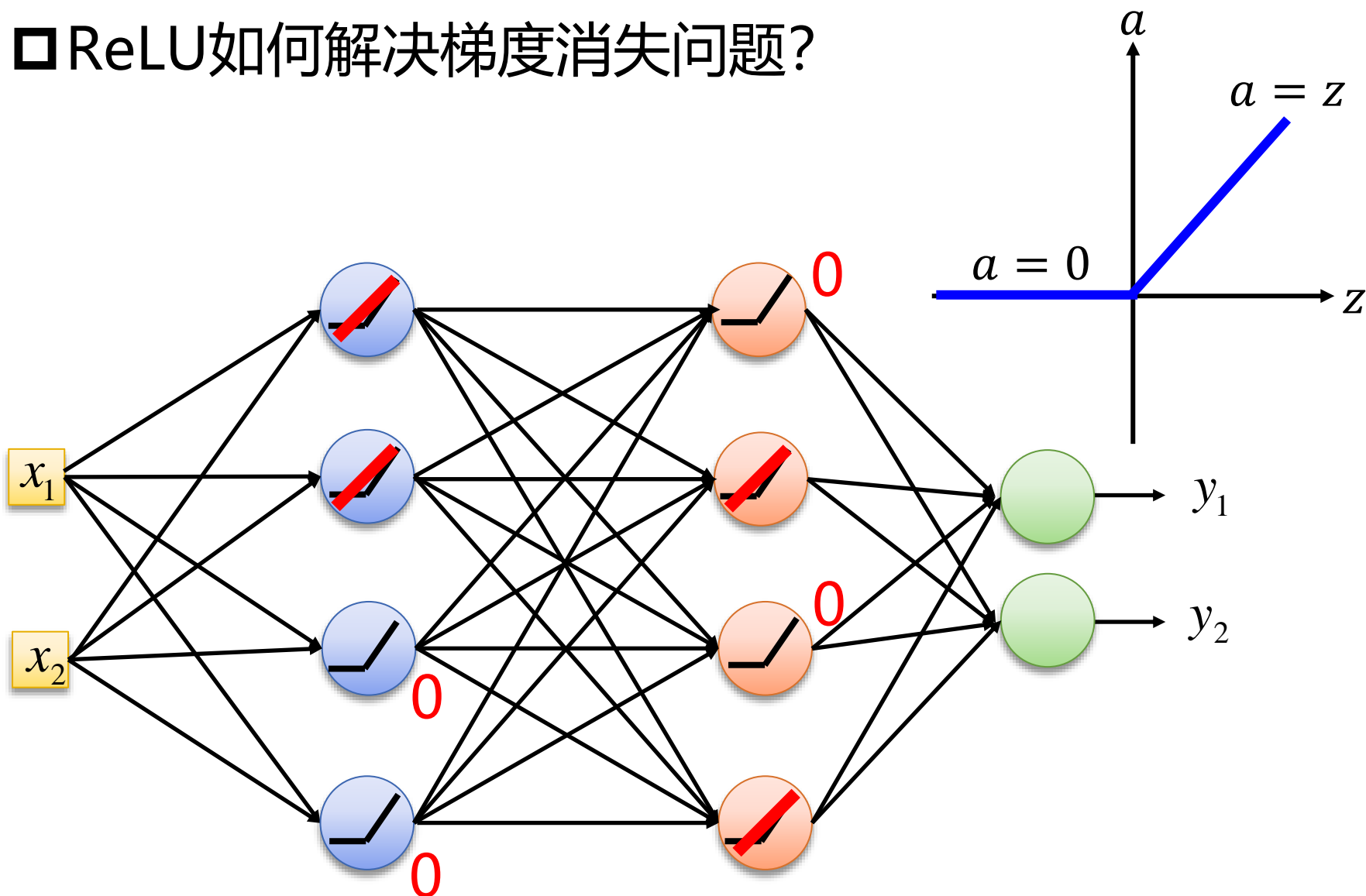
- 非常迅速地地将两条线分类，只使用了 Tanh函数 75% 的时间



大于 0 的部分被完整输出

常用激活函数：ReLU函数

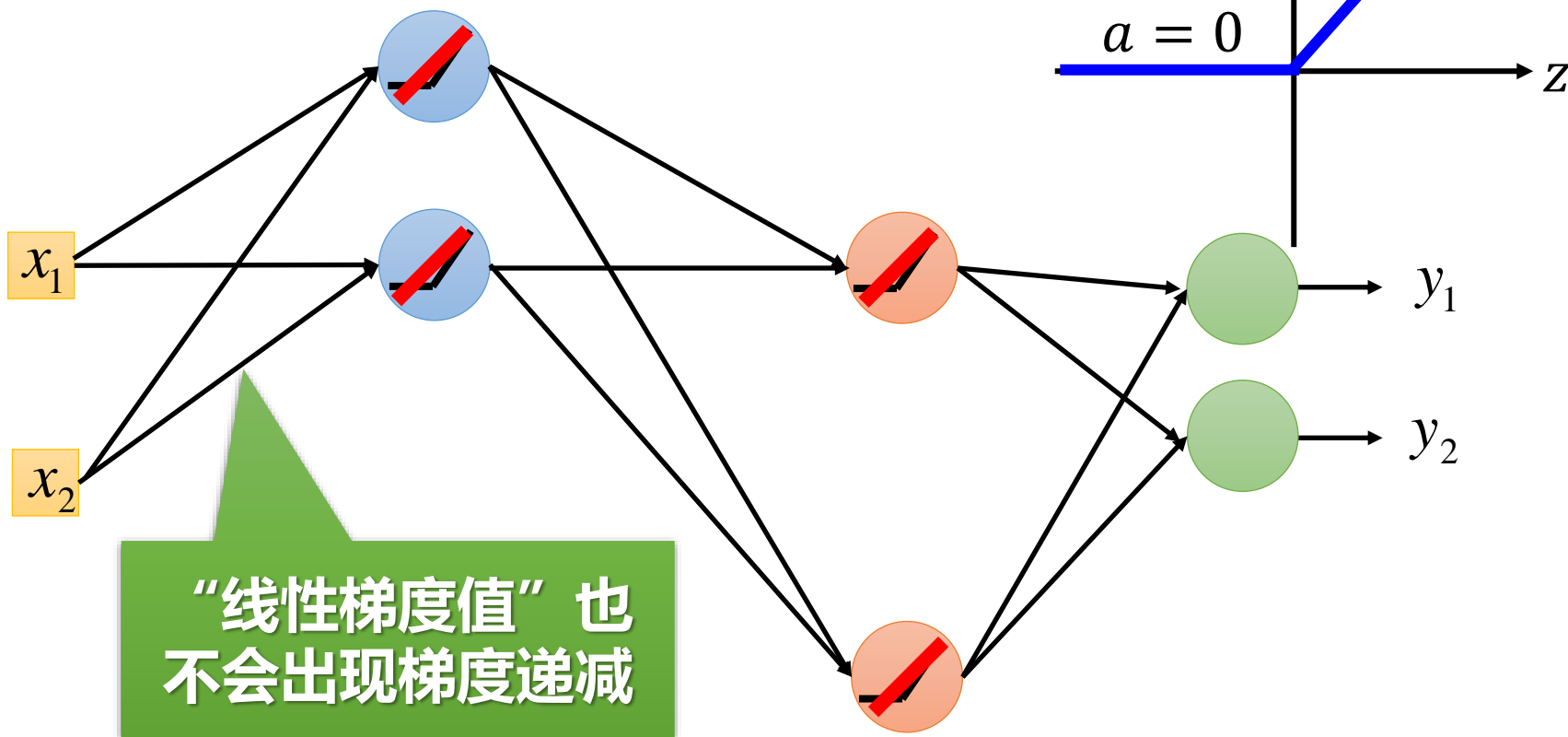
□ ReLU如何解决梯度消失问题？



常用激活函数：ReLU函数

□ ReLU如何解决梯度消失问题？

更“瘦”的线性网络



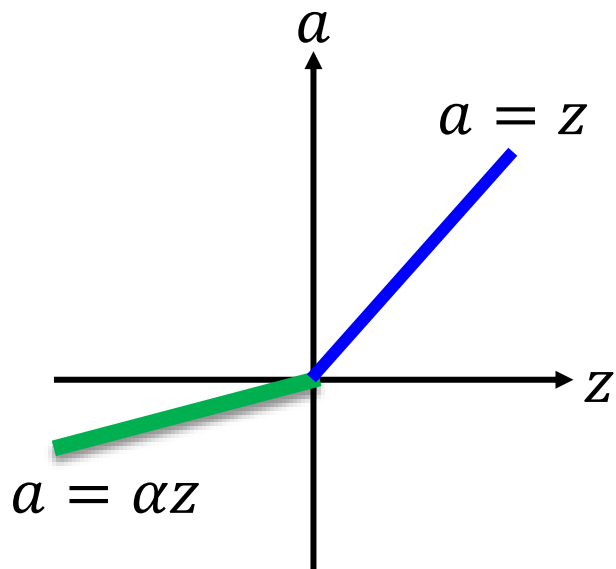
常用激活函数：ReLU变体

□ Leaky-ReLU函数

$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$

□ 优点

- 收敛速度快
- 解决神经元死亡现象
- 函数范围是负无穷到正无穷



□ 缺点

- 实际应用中效果不稳定

α 同样通过梯度
下降学习到

常用激活函数：ReLU变体

□ ELU函数

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

□ 优点

- 收敛速度快
- **缓解**梯度消失
- 对输入变化更鲁棒

□ 缺点

- 计算量更大

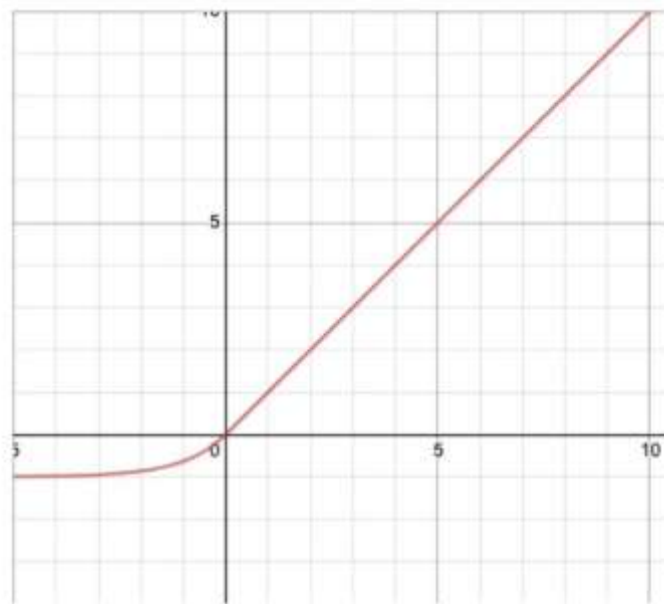
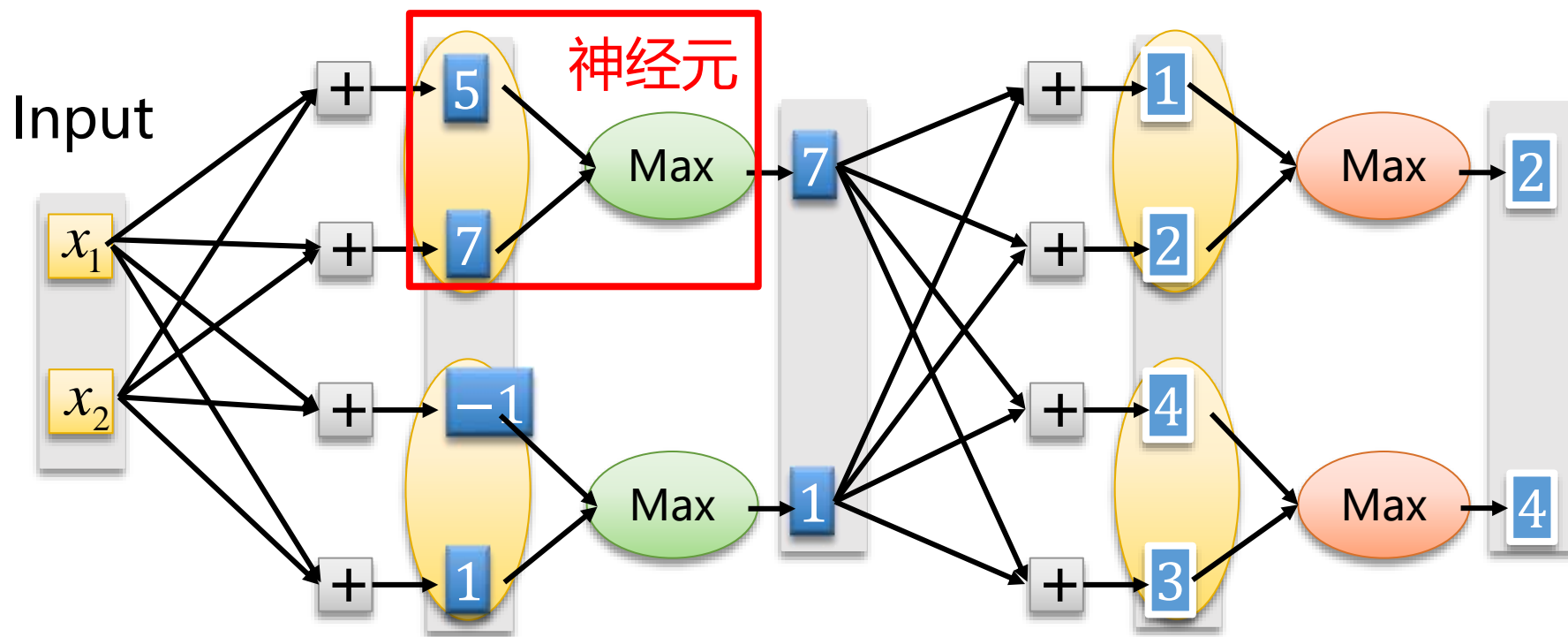


Fig 1. Exponential Linear Unit or ELU activation function

可学习的激活函数：Maxout

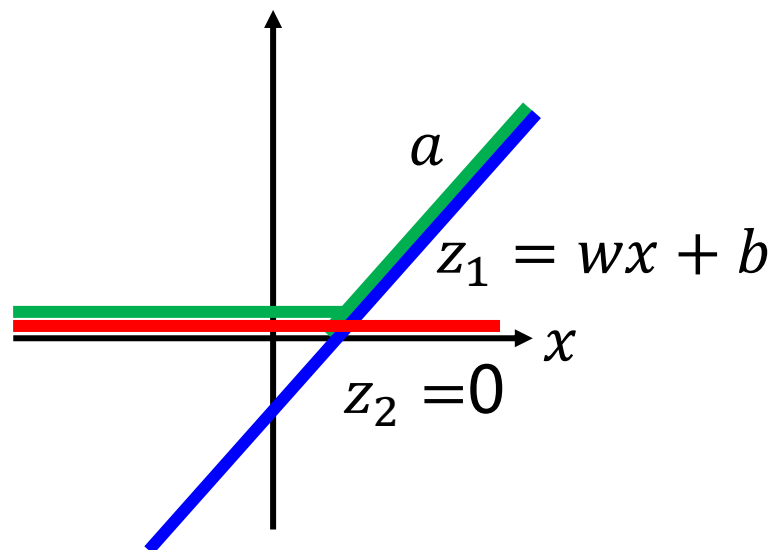
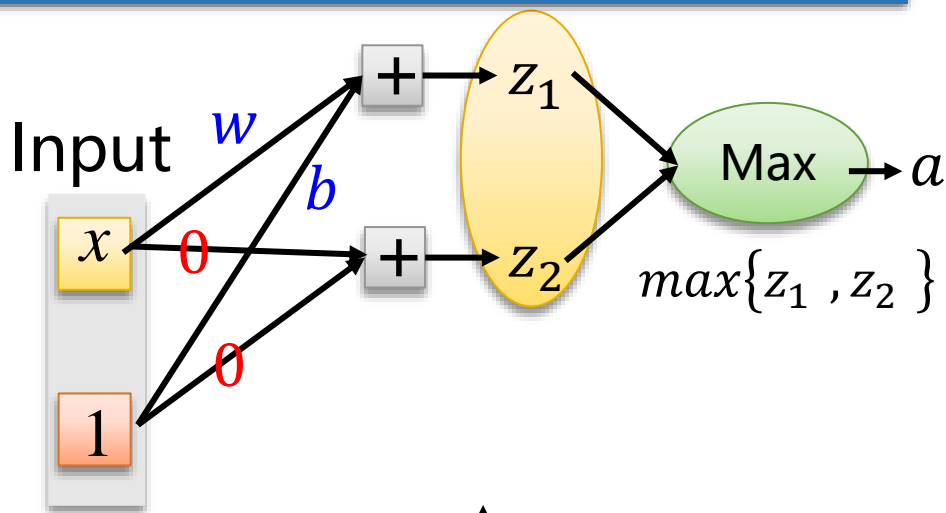
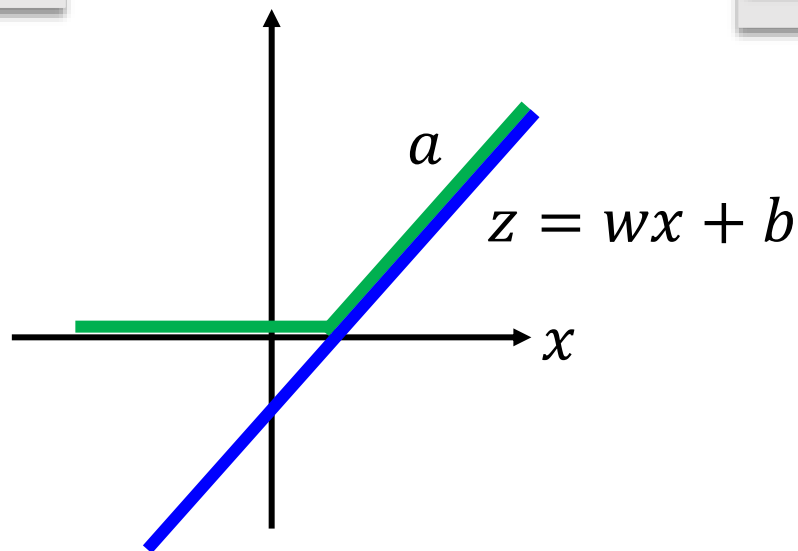
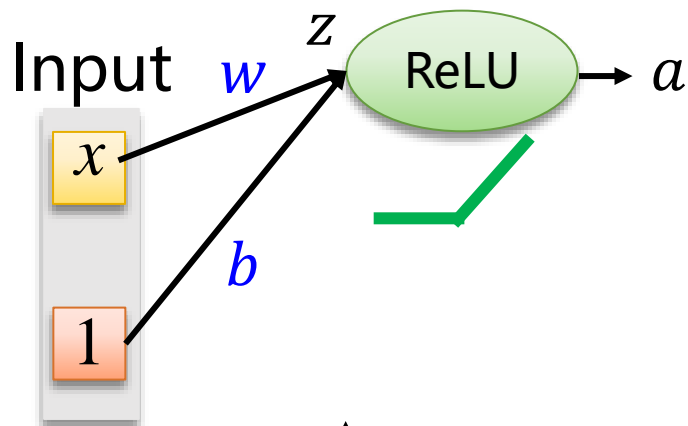
ReLU 是 Maxout 的一种特例



一组中可以不只两个元素

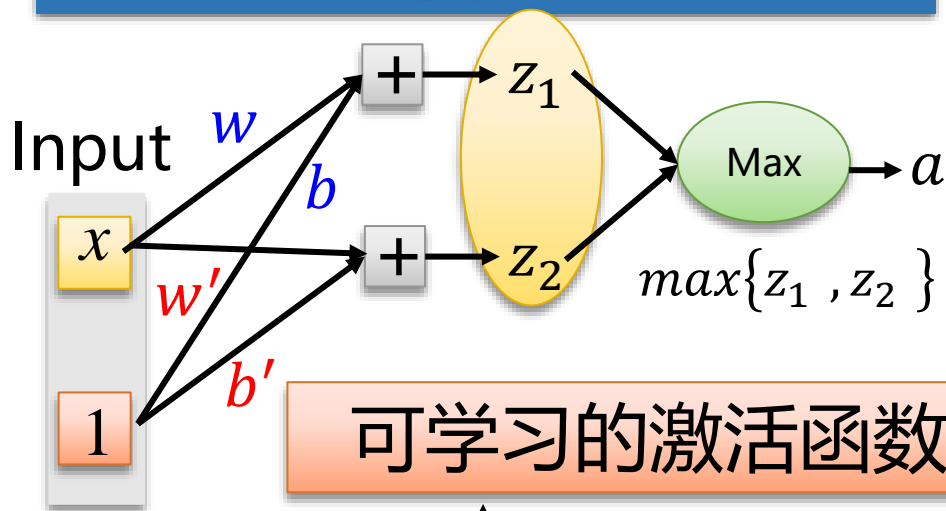
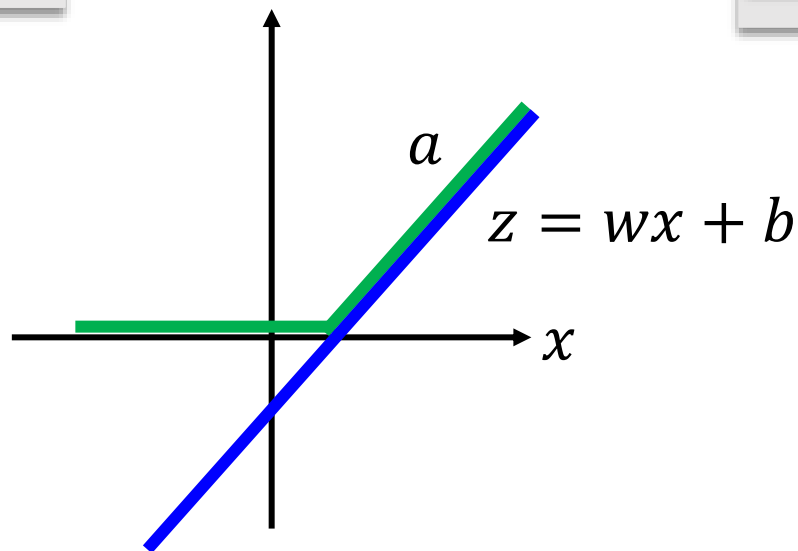
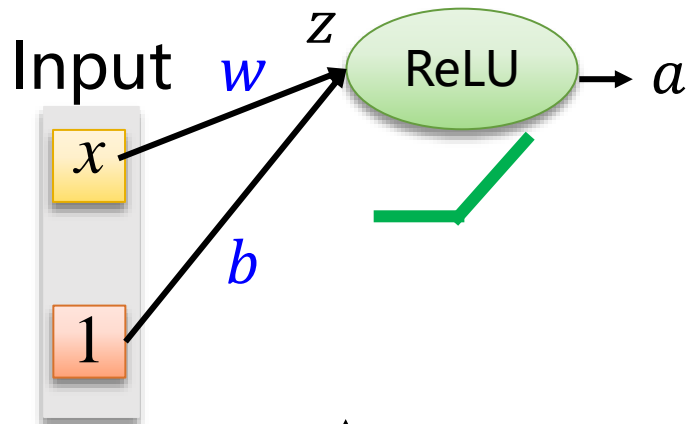
可学习的激活函数：Maxout

ReLU 是 Maxout 的一种特例

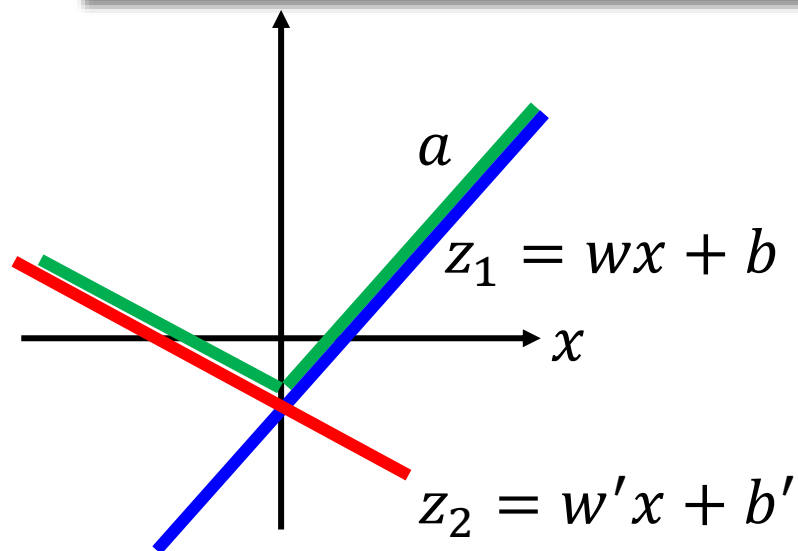


可学习的激活函数：Maxout

Maxout不仅仅是ReLU



可学习的激活函数



可学习的激活函数：Maxout

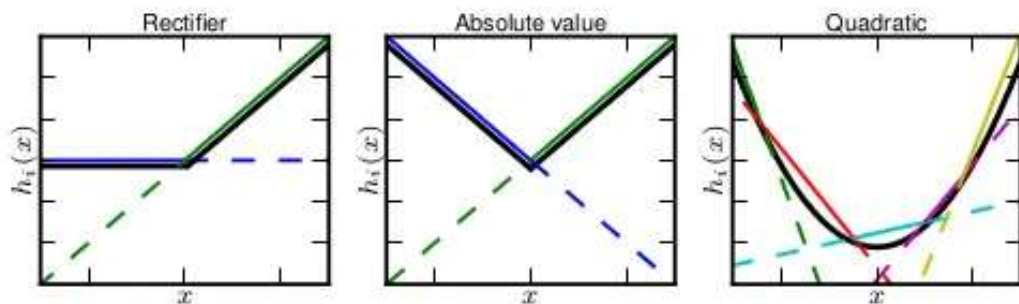
□ Maxout函数(可学习的分段线性函数)

$$f(x) = \max_{i=1,2,\dots,n} (\omega_i^T x + b_i)$$

一组5个元素

□ 优点

- 收敛速度快
- 不饱和
- 可以拟合任意凸函数



一组2个元素

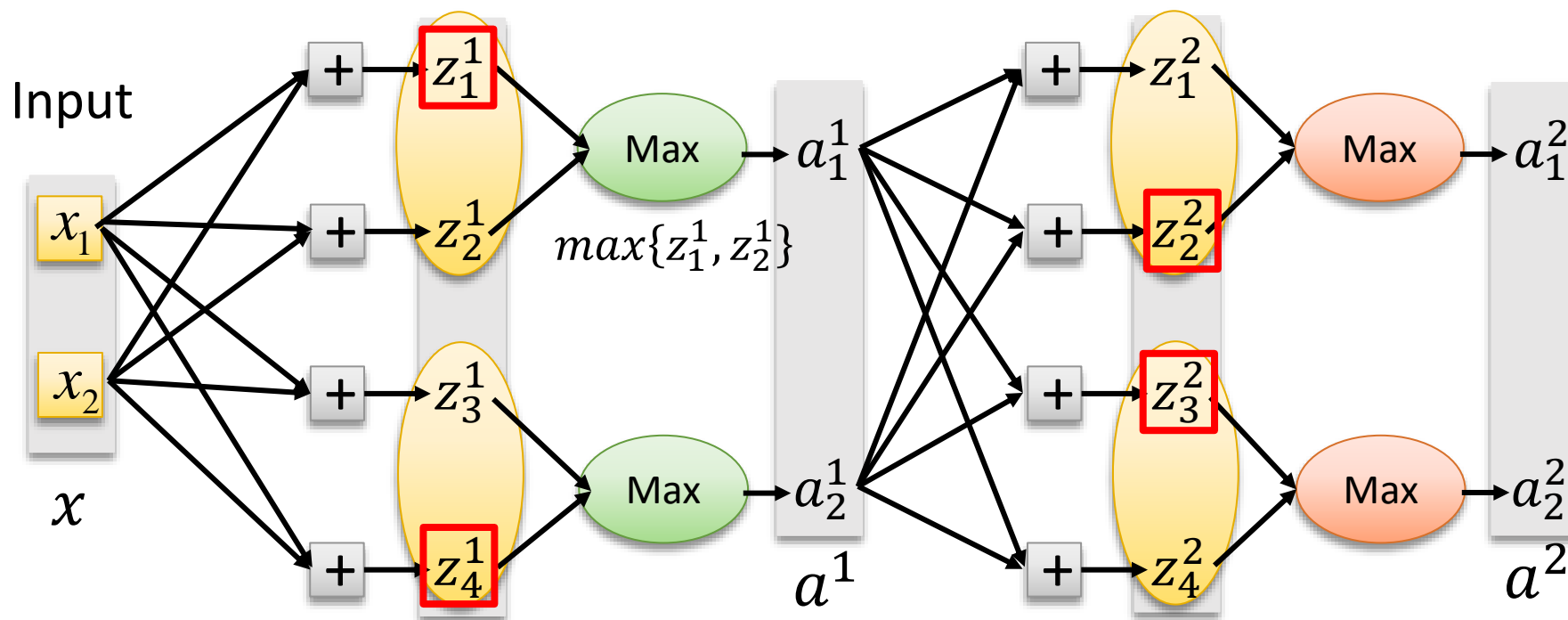
□ 缺点

段的数目取决于一组中有多少元素

- 参数量增加，计算复杂度增加

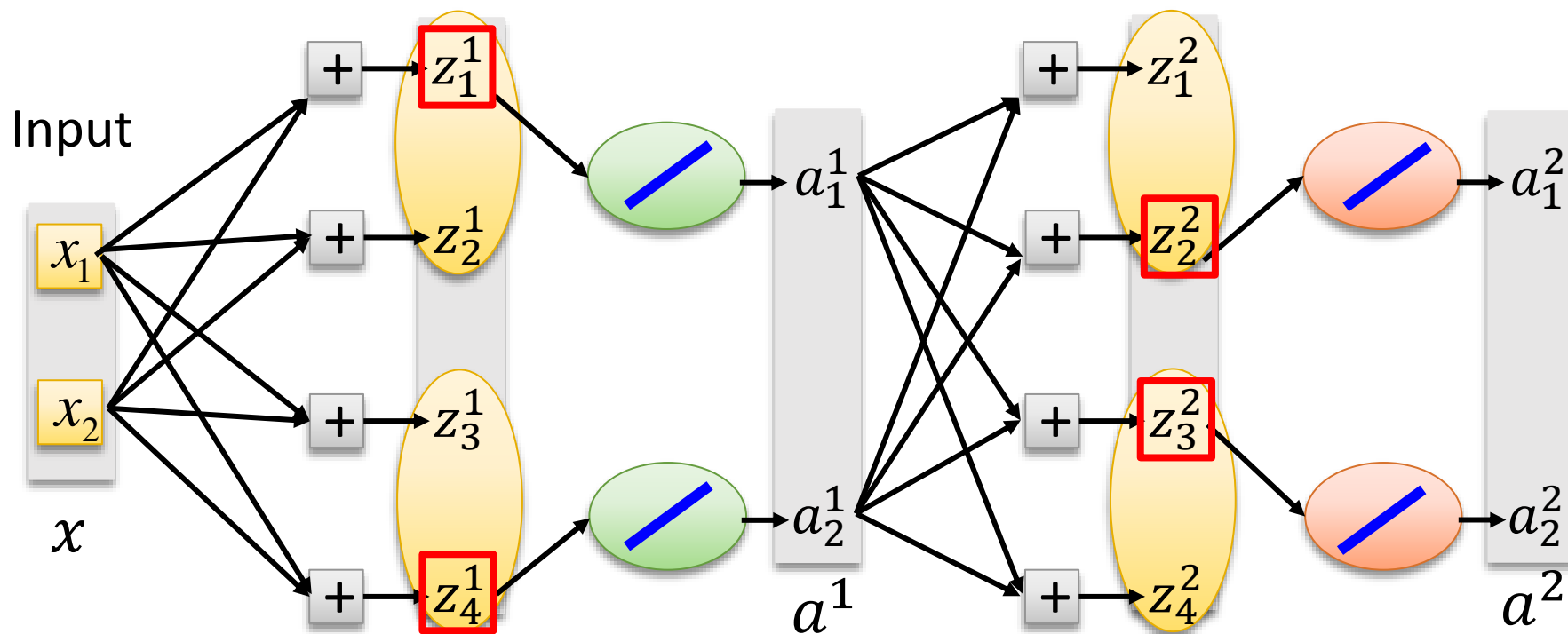
Maxout函数-训练

□ 给定训练数据 x , 易知哪些 z 是最大的



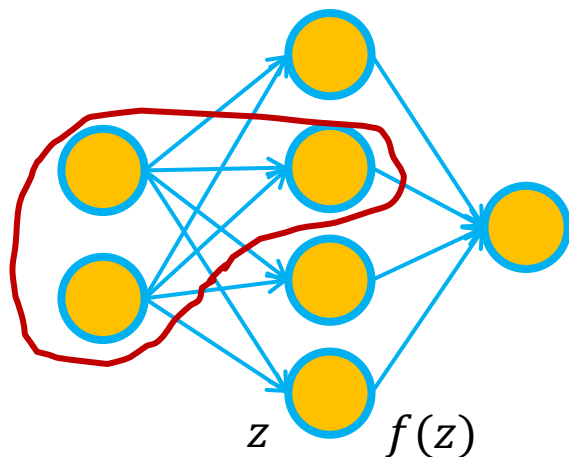
Maxout函数-训练

□ 给定训练数据 x , 易知哪些 z 是最大的

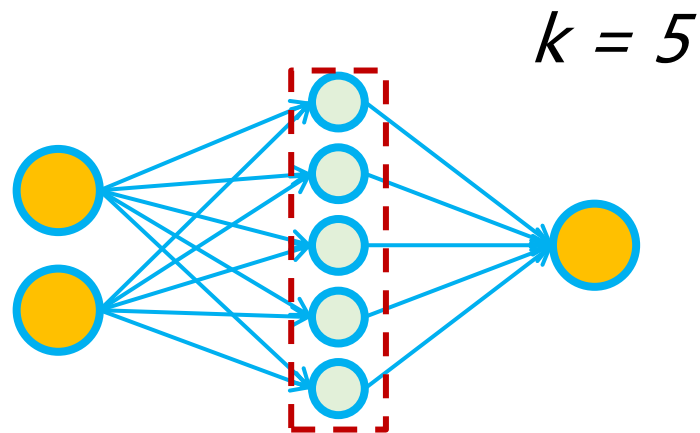


对于不同的样本，得到的网络也会不同

Maxout 函数的复杂度



Maxout
 $k = 5$



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

$$\text{out} = f(\mathbf{z})$$

1 组参数

采用 Maxout 函数, 参数个数成 k 倍增加

$$z_1 = \mathbf{w}_1^T \mathbf{x} + b_1$$

$$z_2 = \mathbf{w}_2^T \mathbf{x} + b_2$$

$$z_3 = \mathbf{w}_3^T \mathbf{x} + b_3$$

$$z_4 = \mathbf{w}_4^T \mathbf{x} + b_4$$

$$z_5 = \mathbf{w}_5^T \mathbf{x} + b_5$$

隐藏层中的
“隐隐层”

5 组参数

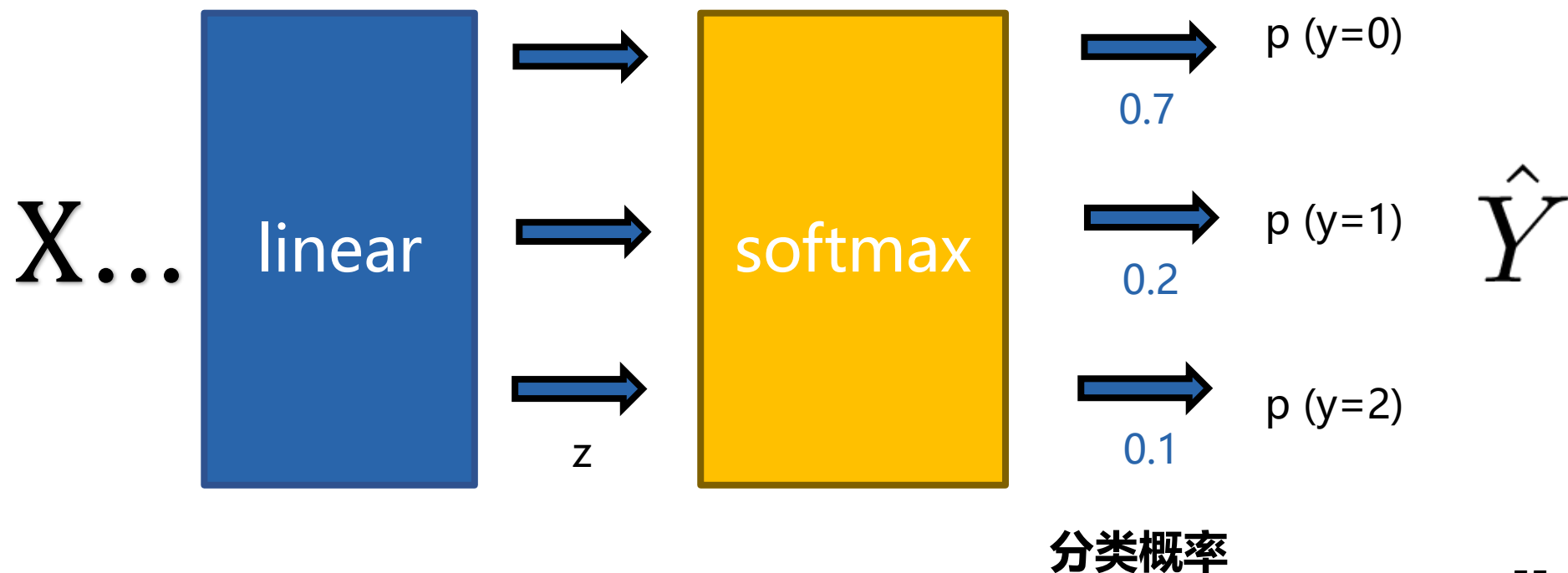
$$\text{out} = f(z_1, z_2, z_3, z_4, z_5)$$

常见的激活函数：Softmax函数

- 将多个神经元输出映射到区间 $(0, 1)$ ，常用于多分类问题中

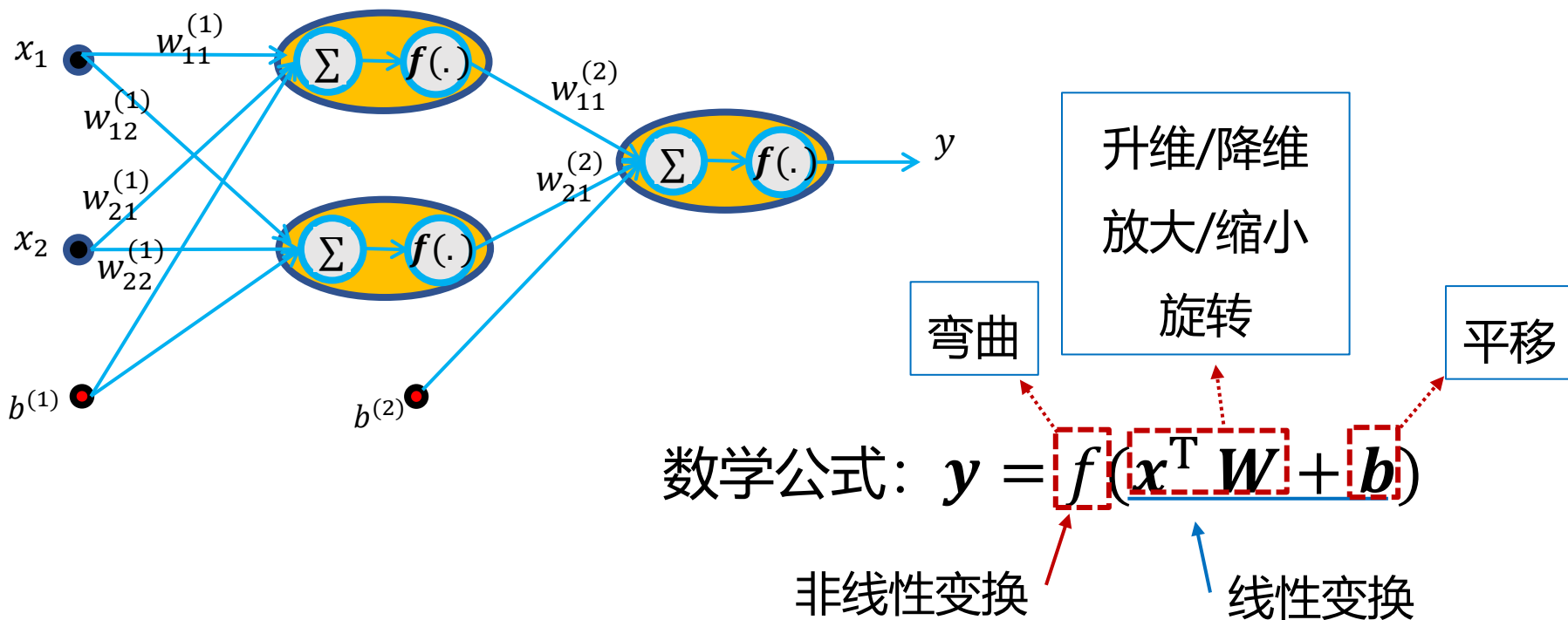
$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_i e^{z_i}}$$

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

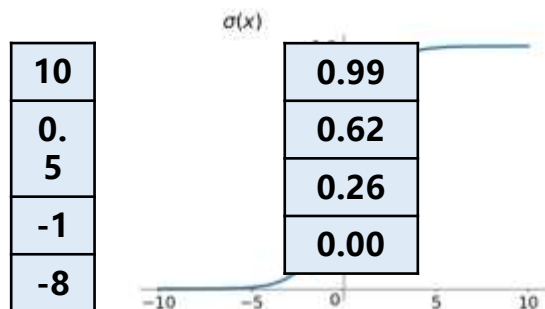


激活函数给神经网络增加了什么能力？

输入空间到输出空间的变换

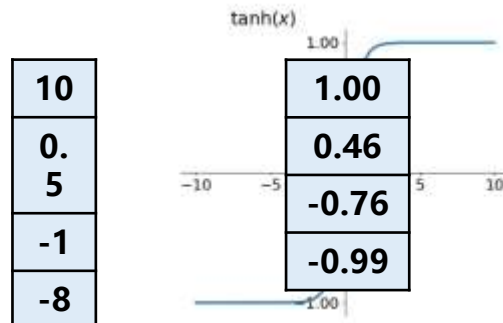


激活函数如何发挥作用？



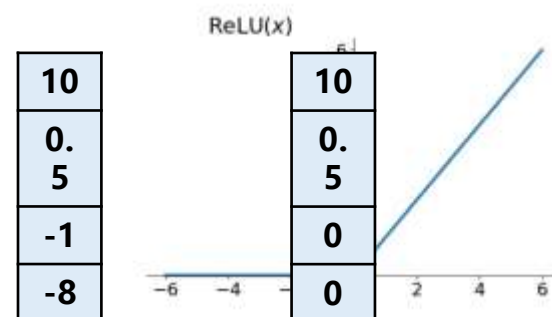
Sigmoid

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



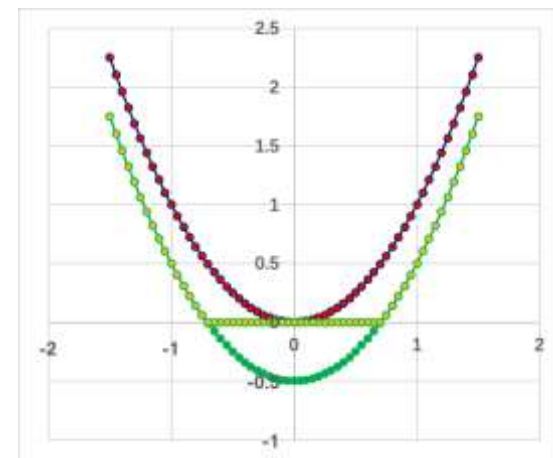
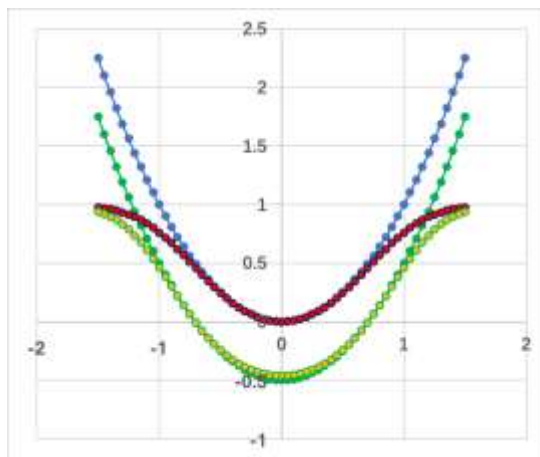
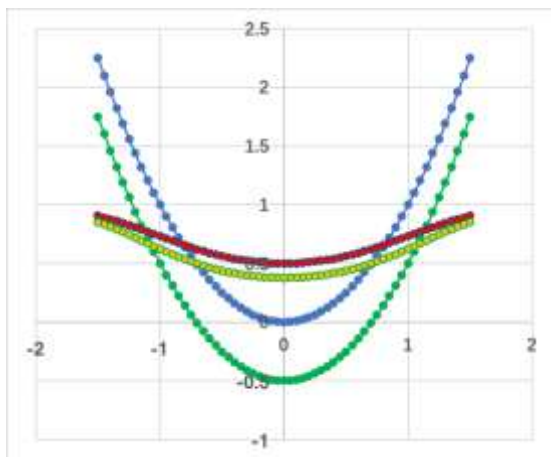
Tanh

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

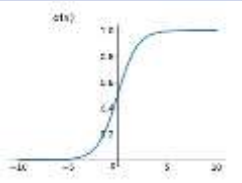
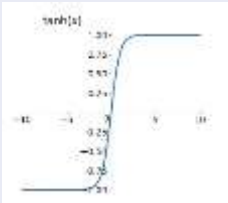
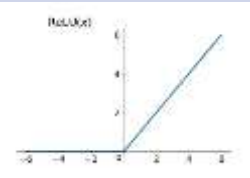


ReLU

$$f(x) = \max(0, x)$$



如何选择激活函数

激活函数	图形	公式	导数	优点	缺点	应用场景
Sigmoid		$\frac{1}{1 + e^{-x}}$	$f(x)(1 - f(x))$	<ul style="list-style-type: none"> • 单调连续 • 输出范围有限 • 优化稳定 • 物理意义上最为接近生物神经元 • 求导容易 	<ul style="list-style-type: none"> • 梯度消失 • 其输出不以0为中心 	特征相差 <ul style="list-style-type: none"> • 比较复杂 • 不是特别大
Tanh		$\frac{1 - e^{-2x}}{1 + e^{-2x}}$	$1 - f(x)^2$	<ul style="list-style-type: none"> • 比 Sigmoid 函数收敛速度更快 • 输出以0为中心 	<ul style="list-style-type: none"> • 梯度消失 	特征相差明显
ReLU		$\max(0, x)$	$\begin{cases} 0 & \text{for } x < 0 \\ 0 & \text{for } x = 0 \\ 1 & \text{for } x > 0 \end{cases}$	<ul style="list-style-type: none"> • 计算简单 • 缓解梯度消失 • 稀疏激活性 	<ul style="list-style-type: none"> • 神经元死亡 • 权重无法更新 	绝大多数场景

激活函数特性

□连续并可导的非线性函数（允许少数点上不可导）

- 反向传播时需要计算激活函数的偏导数

□单调性

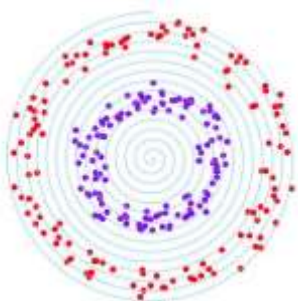
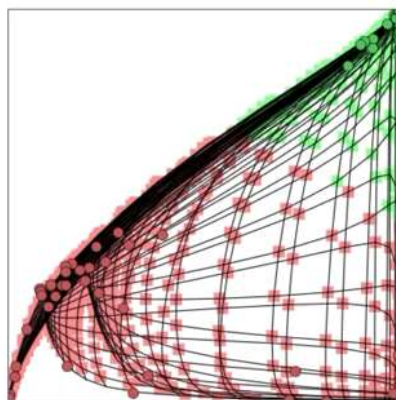
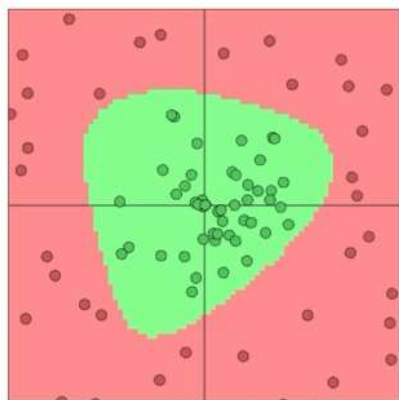
- 激活函数单调，导数符号不变，单层网络能保证是凸函数

□输出值范围

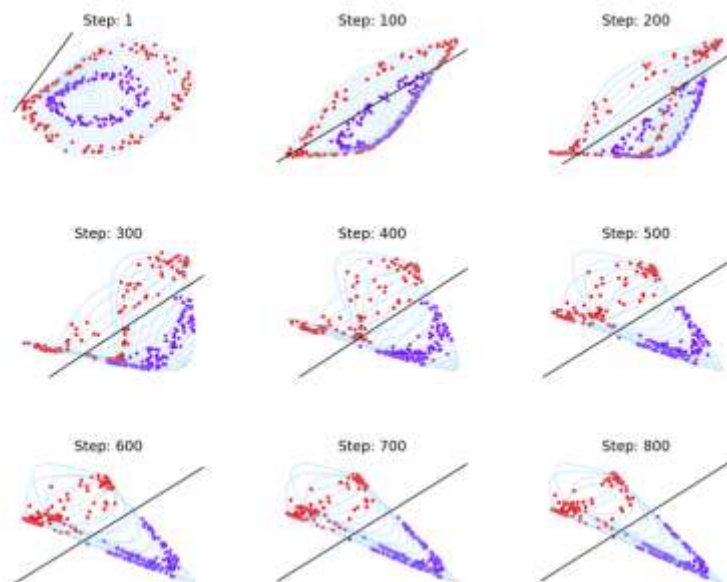
- 当激活函数输出值是 **有限** 的时候，基于梯度的优化方法会更加稳定，因为特征的代表受有限权值的影响更显著
- 当激活函数输出值是 **无限** 的时候，模型的训练会更加高效，不过在这种情况下，一般需要更小的学习率

激活函数小结

□ 神经网络是去学习如何利用矩阵的线性变换加激活函数的非线性变换，将原始输入空间投向线性可分/稀疏的空间去分类/回归



环形数据



内容导览



多层感知机

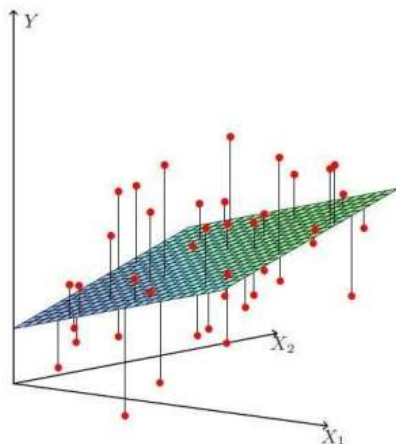
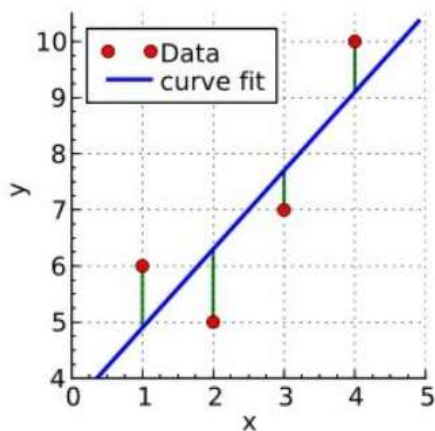
通用近似定理

常用激活函数

损失函数

如何判断函数拟合效果

□ 如何衡量拟合函数与真实值（Ground Truth）之间的偏离程度？



一元变量、二元变量线性回归

➤ Mean Absolute Error: 平均绝对误差

$$\text{MAE}(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^m |y_i - \hat{y}_i|$$

➤ Mean Square Error: 均方误差

$$\text{MSE}(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2$$

➤ Root Mean Square Error: 均方根误差

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{1}{m} \sum_{i=1}^m (y_i - \hat{y}_i)^2}$$

损失函数和目标函数

□ 损失函数 (Loss Function) / 代价函数 (Cost Function)

- 给定一个模型输出 \hat{y} 和一个真实 y , 使用损失函数计算一个实值损失 $f(y_i, \hat{y}_i)$, 代价函数输出整个训练集 (或者mini-batch) 的总损失 $J = \sum_{i=1}^N f(y_i, \hat{y}_i)$

□ 目标函数 (Object Function)

- 一个更通用的术语, 表示任意希望被优化的函数, 用于机器学习领域和非机器学习领域 (比如运筹优化)

损失函数

□ 用来估量模型的输出 \hat{y} 与真实值 y 之间的偏离程度

- 反映神经网络对数据集的拟合程度，拟合越差，损失函数值越大
- 损失函数比较大时，其对应的梯度也要比较大，这样可以更快速地更新模型参数

□ 损失函数特征

- 非负性
- 预测值和真实值接近时，损失函数值趋于零

回归问题

Mean Square Error/Quadratic Loss
均方误差/二次损失

Mean Absolute Error
平均绝对误差

Huber
Loss/Smooth
Mean Absolute
Error
Huber 损失/平滑平
均绝对误差

Log Cosh Loss
对数余弦损失

Quantile Loss
分位数损失

分类问题

KL Divergence/Relative Entropy
KL 散度/相对熵

Hinge Loss
合页损失

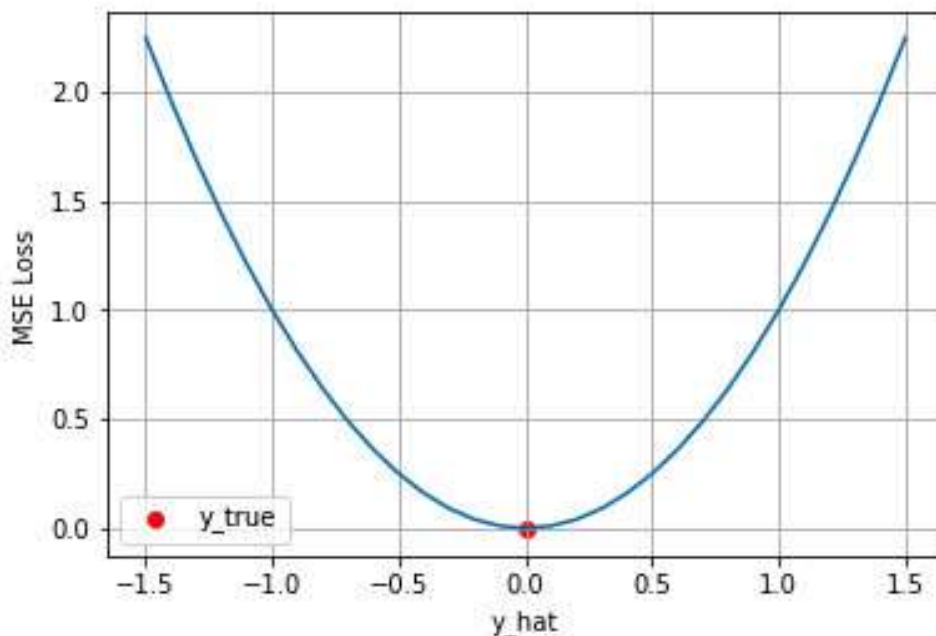
Exponential Loss
指数损失

回归损失函数：均方误差

□ 均方误差 Mean Square Error (MSE)

□ L2 Loss：随着误差增加，损失呈二次方的增加

$$L_{\text{MSE}} = \frac{1}{2N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

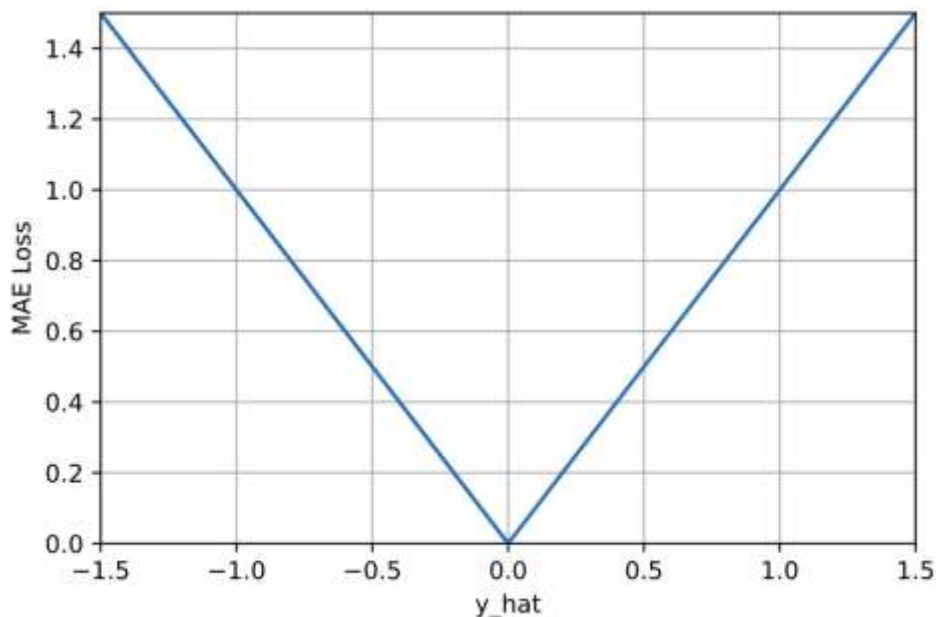


回归损失函数：平均绝对误差

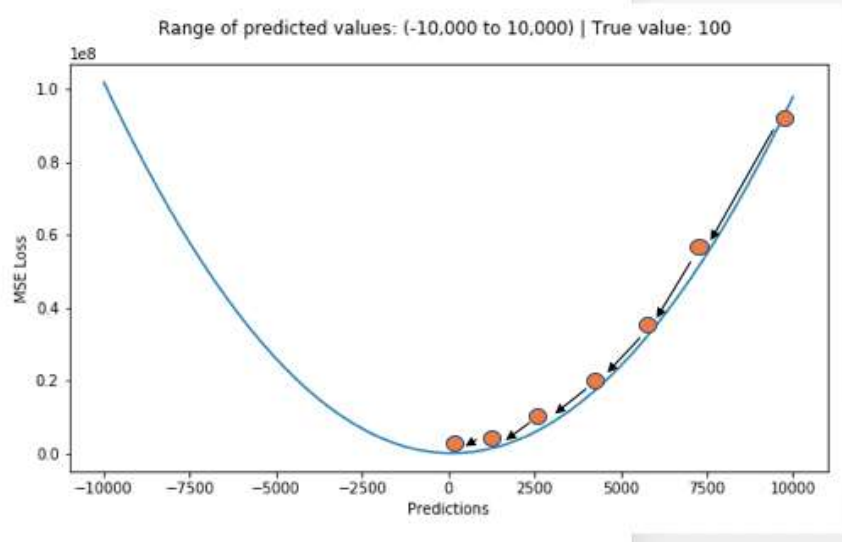
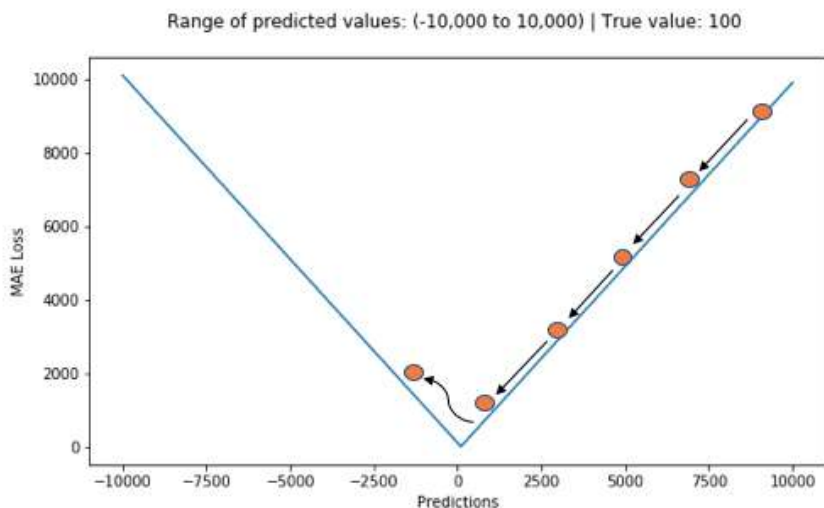
□ 平均绝对误差 Mean Absolute Error (MAE)

□ L1 Loss：随着误差增加，损失呈线性增长

$$L_{\text{MAE}} = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i|$$



回归损失函数—MSE 与 MAE 的对比



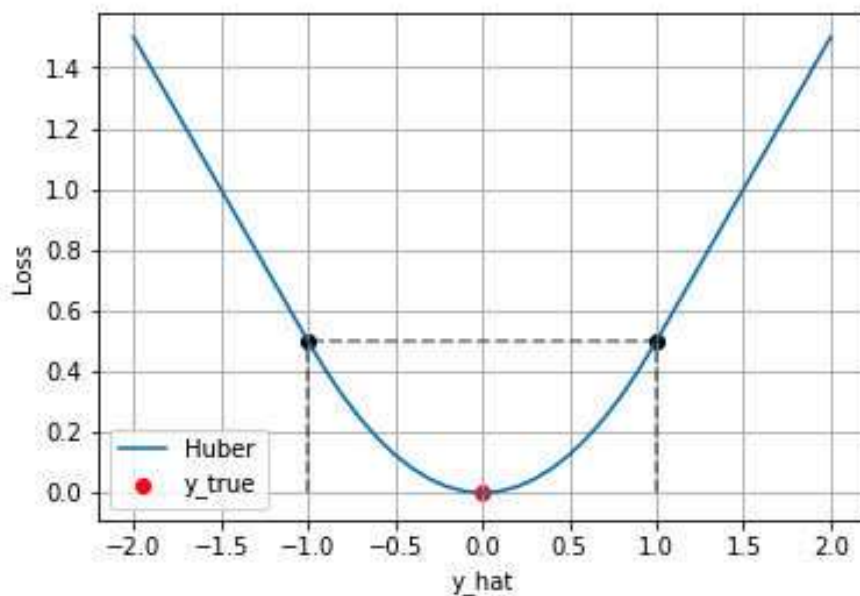
- MAE 损失的梯度始终是相同，对于小的损耗值，梯度也会很大
- 对异常值处理效果更好

- MSE 损失的梯度随损失增大而增大，而损失趋于 0 时则会减小
- 对异常值更敏感

回归损失函数：Huber 损失

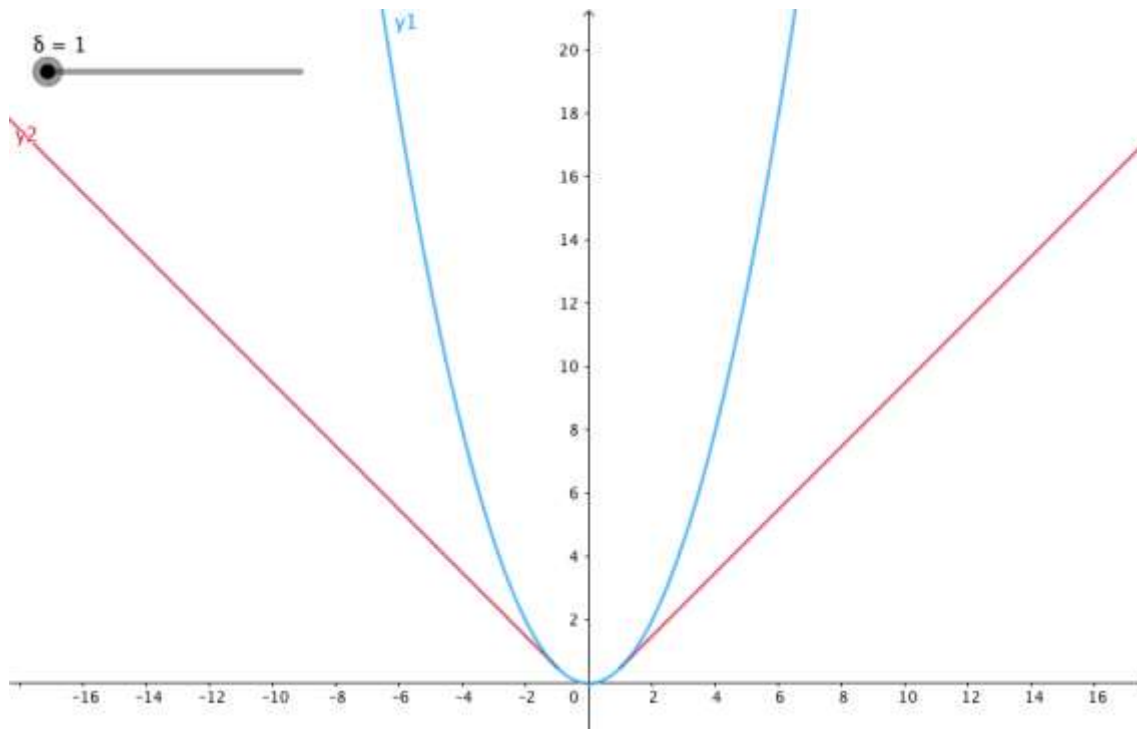
- 在误差接近 0 时使用 MSE，使损失函数可导并且梯度更加稳定
- 误差较大时使用 MAE，可以降低异常值的影响

$$L_{\text{SMAE}} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{|y_i - \hat{y}_i| \leq \delta} \frac{(y_i - \hat{y}_i)^2}{2} + \mathbb{I}_{|y_i - \hat{y}_i| > \delta} (\delta |y_i - \hat{y}_i| - \frac{1}{2} \delta^2)$$



回归损失函数：Huber 损失

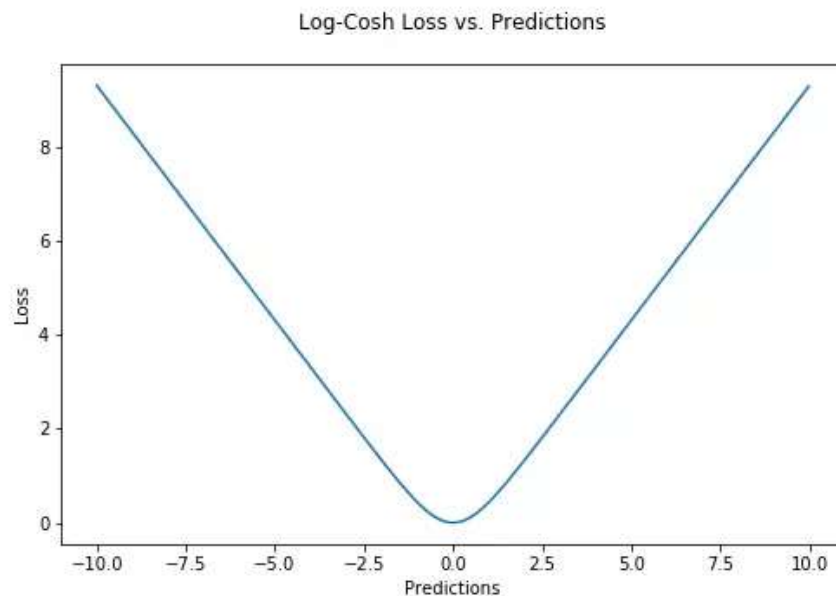
$$L_{\text{SMAE}} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{|y_i - \hat{y}_i| \leq \delta} \frac{(y_i - \hat{y}_i)^2}{2} + \mathbb{I}_{|y_i - \hat{y}_i| > \delta} (\delta |y_i - \hat{y}_i| - \frac{1}{2} \delta^2)$$



回归损失函数：对数余弦损失

- 对于较小误差 $|y - \hat{y}|$ ，近似于MSE，收敛下降较快
- 对于较大误差，近似等于 $|y - \hat{y}| - \log(2)$ ，类似于 MAE，不会受到离群点的影响
- 有 Huber 损失所有优点，且不需要设定超参数
- 求导比较复杂，计算量较大，在深度学习中使用不多

$$L_{\log-\cosh} = \frac{1}{N} \sum_{i=1}^N \log(\cosh(\hat{y}_i - y_i))$$



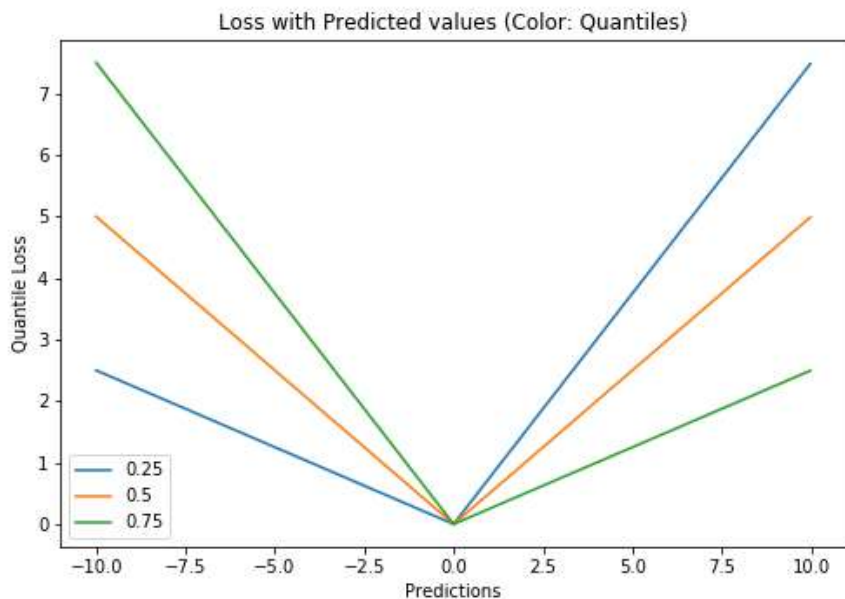
回归损失函数：分位数损失

- 一个分段函数， γ 为分位数系数
- 预测出一个取值区间，控制预测的值偏向哪个区间
- 使用不同系数来控制高估和低估在整个损失值的权重

$$L_{quant} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}_{\hat{y}_i \geq y_i} (1 - \gamma) |y_i - \hat{y}_i| + \mathbb{I}_{\hat{y}_i < y_i} \gamma |y_i - \hat{y}_i|$$

高估

低估



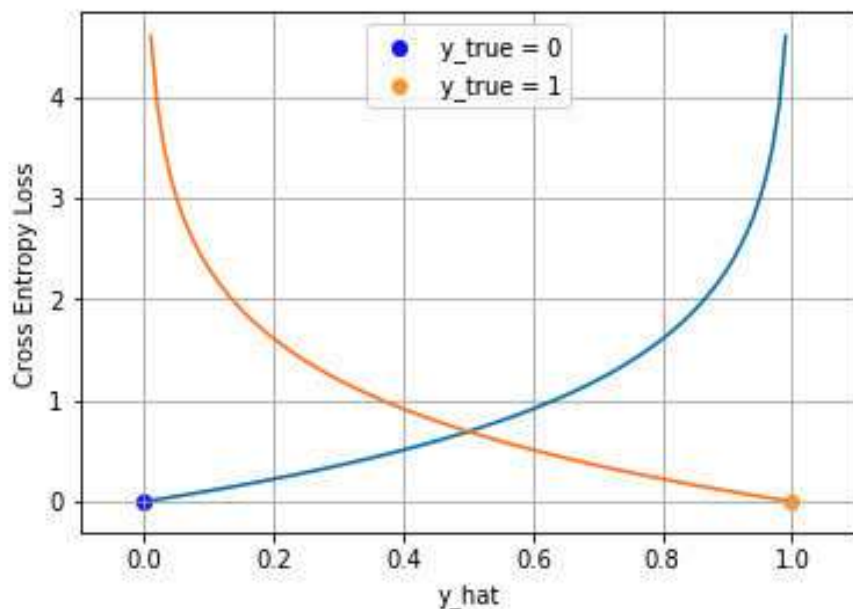
分类损失函数：交叉熵

□描述两个概率分布之间的距离，越小说明越接近

□随着误差变大，损失呈指数增长

➤二分类： $L_{CE} = -\sum_{i=1}^N (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$

➤多分类： $L_{CE} = -\sum_{i=1}^N \sum_{j=1}^K y_i^{c_j} \log(\hat{y}_i^{c_j})$



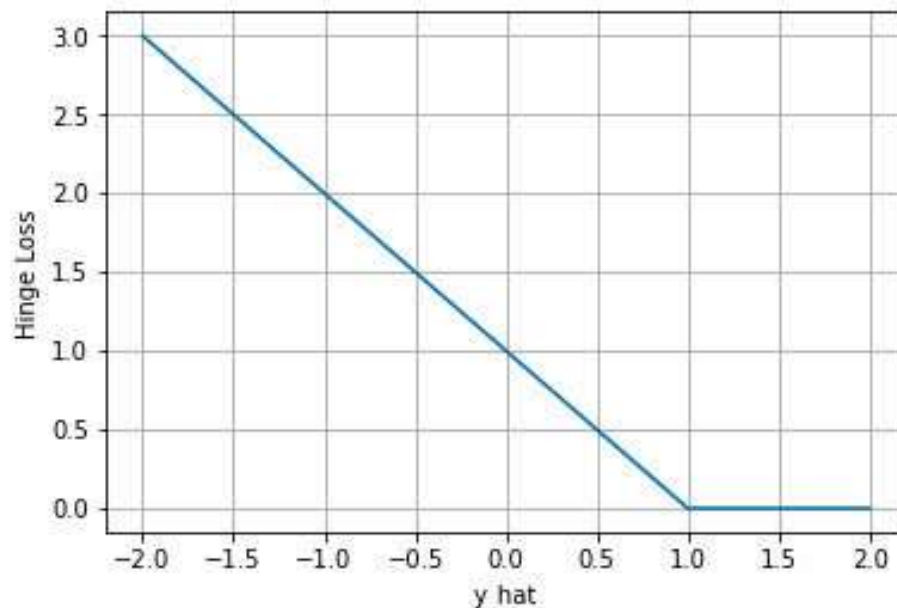
$y_i^{c_j}$ 表示第 i 个样本，
属于第 j 个类别

分类损失函数：合叶损失

- 不仅要分类正确，而且确信度足够高时损失才是 0
- 模型输出负值会有较大惩罚，输出在 $(0, 1)$ 区间时，会有一个较小惩罚

$$\text{sgn}(y_i) = 1$$

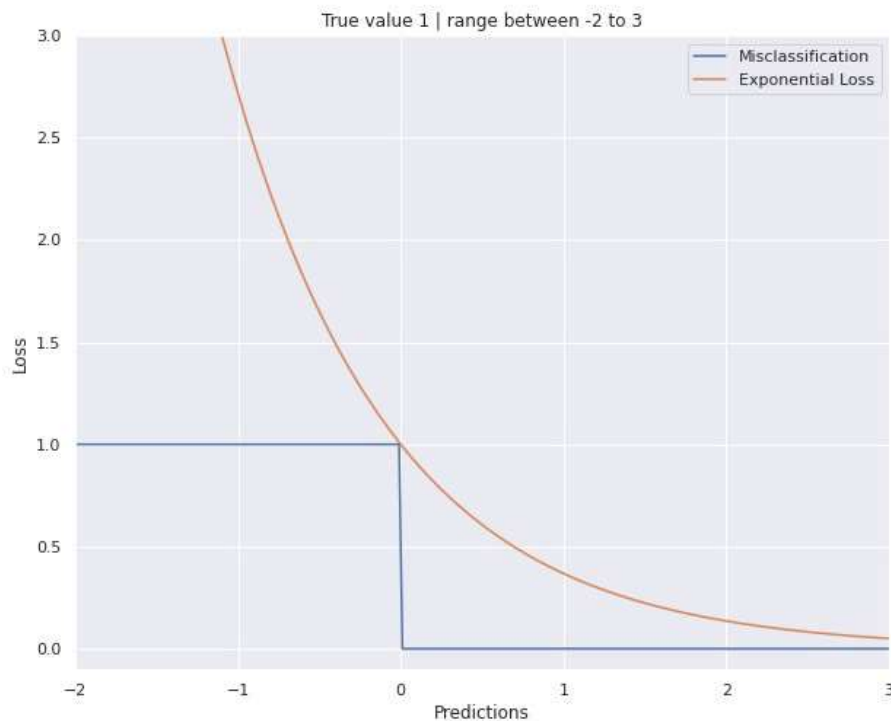
$$L_{\text{hinge}} = \sum_{i=1}^N \max(0, 1 - \text{sgn}(y_i) \hat{y}_i)$$



分类损失函数：指数损失

- 0/1 损失的一种近似
- 对离群点、噪声非常敏感
- 主要用于 AdaBoost 算法

$$L_{EL} = \frac{1}{N} \sum_{i=1}^N e^{-y_i \cdot \hat{y}_i}$$



分类损失函数：焦点损失

□ 解决样本的类别不平衡问题

□ 解决简单/困难样本不平衡问题

$$CE(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1 - p) & \text{otherwise} \end{cases} \quad p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise} \end{cases}$$



$$CE(p, y) = CE(p_t) = -\log(p_t)$$



增加权重参数

解决类别不平衡问题

平衡交叉熵 (Balanced Cross Entropy)

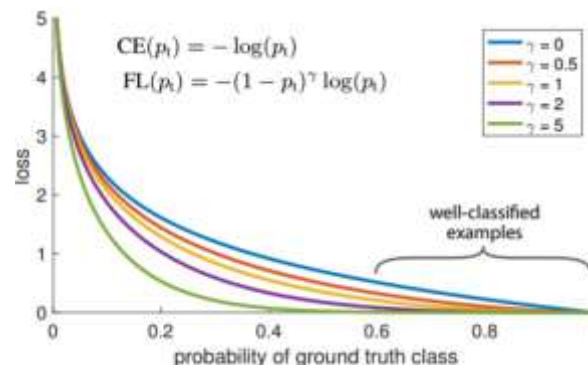
$$CE(p_t) = -\alpha_t \log(p_t) \quad \alpha_t = \begin{cases} \alpha & \text{if } y = 1 \\ 1 - \alpha & \text{otherwise} \end{cases}$$



样本越简单权重越小

焦点损失 (Focal Loss)

$$FL(p_t) = -\alpha_t (1 - p_t)^\gamma \log(p_t)$$



代理损失函数 (Surrogate Loss Function)

- 分类损失函数：0-1 损失函数 $L(y, f(x)) = \begin{cases} 0, & y \neq f(x) \\ 1, & y = f(x) \end{cases}$
- 当预测错误时，损失函数为 1，当预测正确时，损失函数值为 0
 - 该损失函数不考虑预测值和真实值的误差程度。只要错误，就是 1
 - 0-1 损失函数非凸、非连续 **(梯度下降法失效)**，导致目标函数难以直接求解，需要使用其它性能较好的函数（代理损失函数）进行替换

模型	代理损失函数	公式
支持向量机	合页损失 Hinge loss	$\max(0, 1 - yf(x))$
逻辑回归	对数损失 Log loss	$\log(1 + \exp(-yf(x)))$
AdaBoost	指数损失 Exponential loss	$\exp(-yf(x))$
支持向量机	感知损失 Perceptron loss	$\max(0, -f(x))$

