《深度学习》

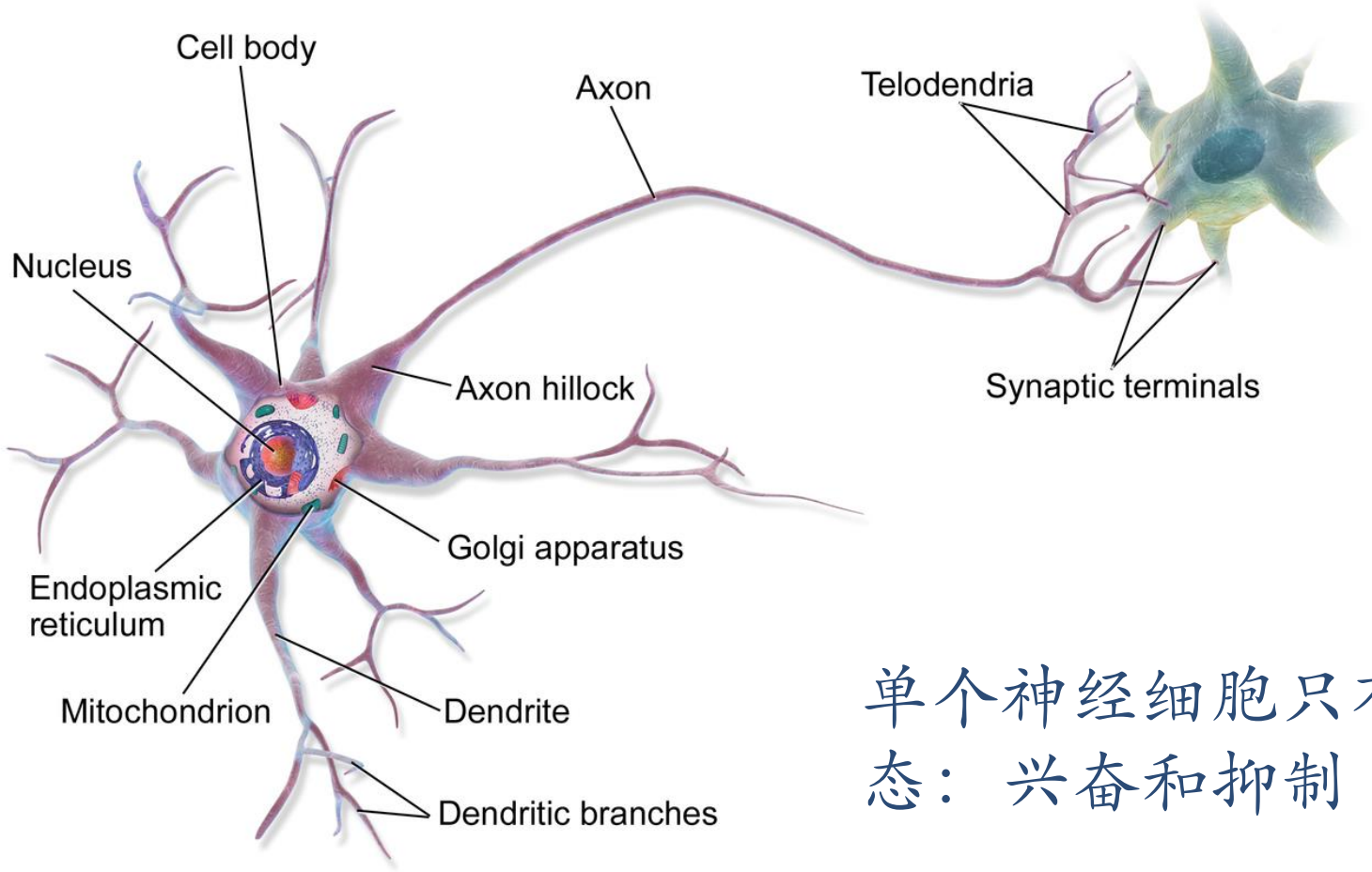
前馈神经网络

内容

- 1.神经元
  - 法乎自然，神经元
  - 线性、非线性、XOR
  - 激活函数的性质
  - 常见激活函数（隐层）
  - 对比logistic和tanh
  - 早期网络MP、**Rosenblatt**
- 2.神经网络
  - 别名
  - 特征，三方面
  - 从三到万，网络结构
  - 深还是宽，万能近似
  - 数学表达（复合函数）
  - 画图表达 （计算图）
  - FP & BP
  - 如何解决贡献度问题（求梯度）
  - 链式法则与计算图的梯度传播——自动微分
  - 业内实现：动态图、静态图
  - 梯度消失和爆炸的根源

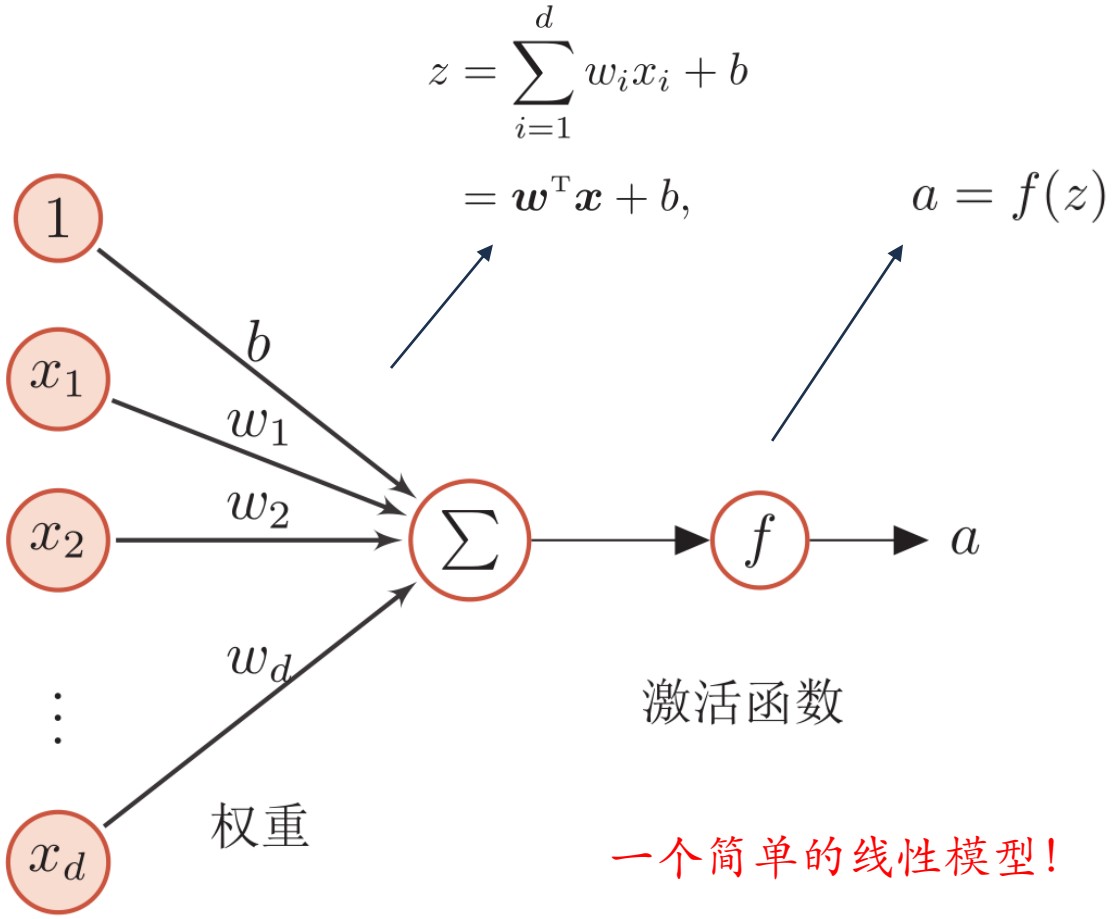
神经元

生物神经元



单个神经细胞只有两种状态：兴奋和抑制

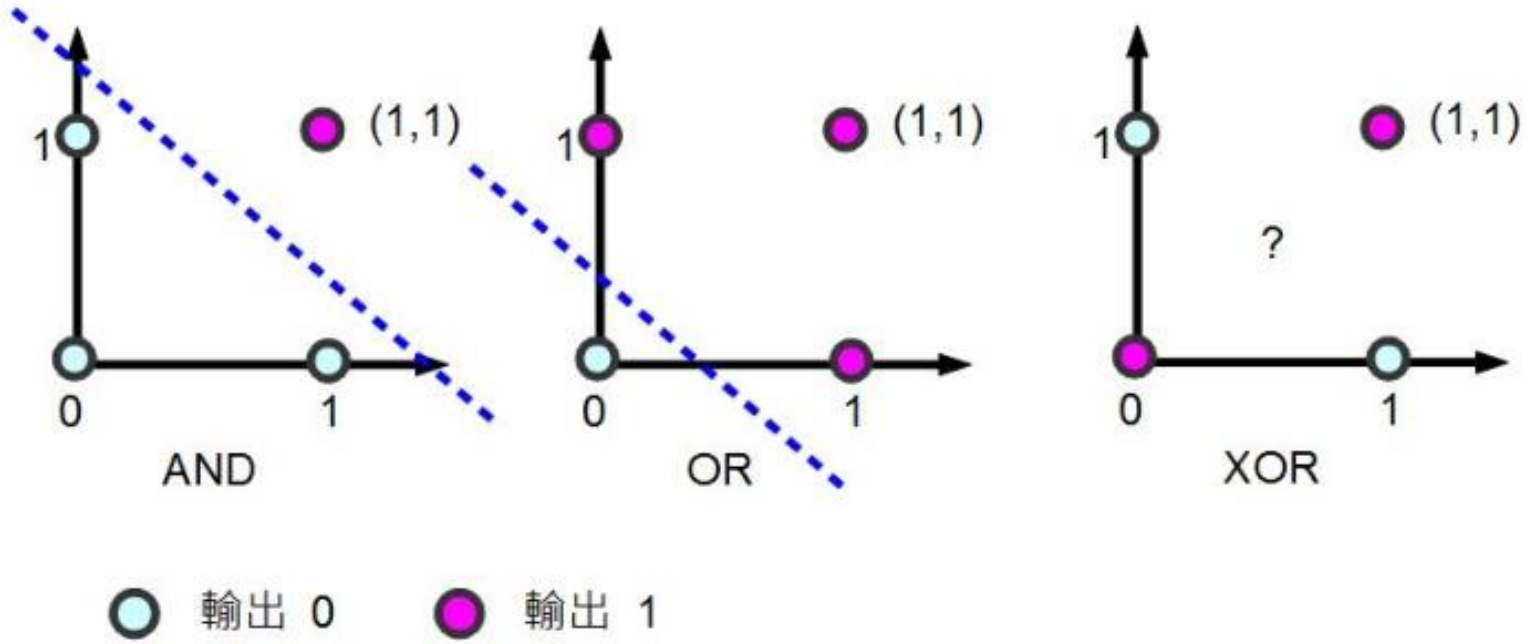
人工神经元


$z = \sum_{i=1}^{d} w_i x_i + b$
$= \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b,$
$a = f(z)$
1
x_1
x_2
x_d
b
w_1
w_2
w_d
$\sum$
f
a
激活函数
权重
一个简单的线性模型!

线性、非线性

- 两个层面：
  - 对于数据：
    - 线性可分与线性不可分。
  - 对于模型：
    - **线性方程**也称一次方程式。指未知数都是一次的方程。其一般的形式是ax+by+...+cz+d=0。线性方程的本质是等式两边乘以任何相同的非零数，方程的本质都不受影响。
    - 因为在笛卡尔坐标系上任何一个一次方程的表示都是一条**直线**。组成一次方程的每个项必须是常数或者是一个常数和一个变量的乘积。

XOR：一个简单的非线性问题



解决XOR问题

我们现在可以指明我们的整个网络是

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^{\top} \max\{0, \boldsymbol{W}^{\top}\boldsymbol{x} + \boldsymbol{c}\} + b.$$

我们现在可以给出 XOR 问题的一个解。令

$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\boldsymbol{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix},$$

$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

及 $b = 0$。

解决XOR问题

- 令 $\boldsymbol{X}$ 表示设计矩阵，它包含二进制输入空间中全部的四个点，每个样本占一行，那么矩阵表示为：X
- 神经网络的第一步是将输入矩阵乘以第一层的权重矩阵：XW
- 然后，我们加上偏置向量 $\boldsymbol{c}$，得到： $\mathrm{XW} + \boldsymbol{c}$
- 在当前这个空间中，所有的样本都处在一条斜率为 1 的直线上。

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

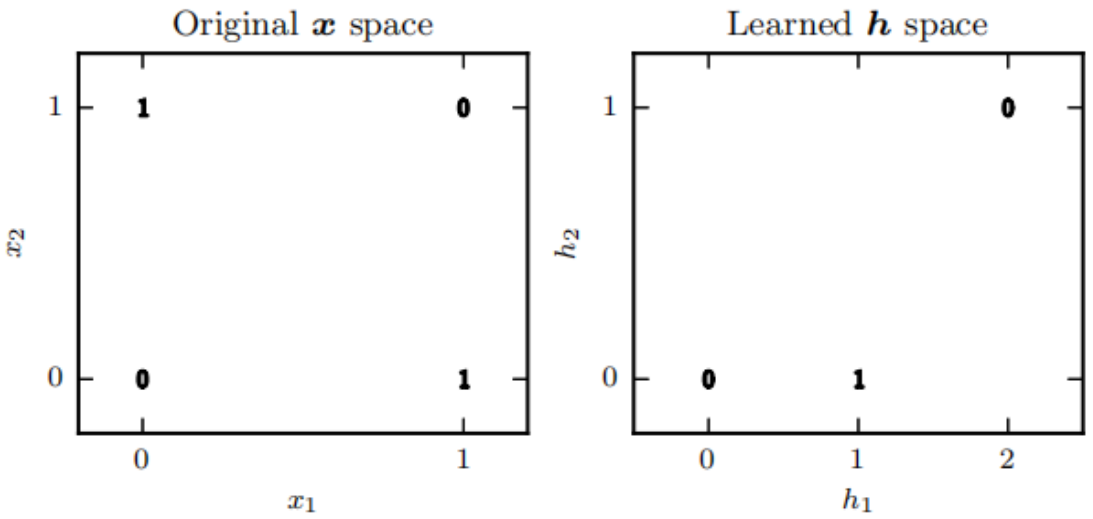
$$\boldsymbol{XW} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 2 & 2 \end{bmatrix}.$$

$$\begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}.$$

Relu

- 我们使用整流线性变换：
  - $\boldsymbol{h1} = \max\{0, \boldsymbol{W}\boldsymbol{x} + \boldsymbol{c}\}$
- 这个变换改变了样本间的关系。它们不再处于同一条直线上了。
-
- 它们现在处在一个可以用线性模型解决的空间上。

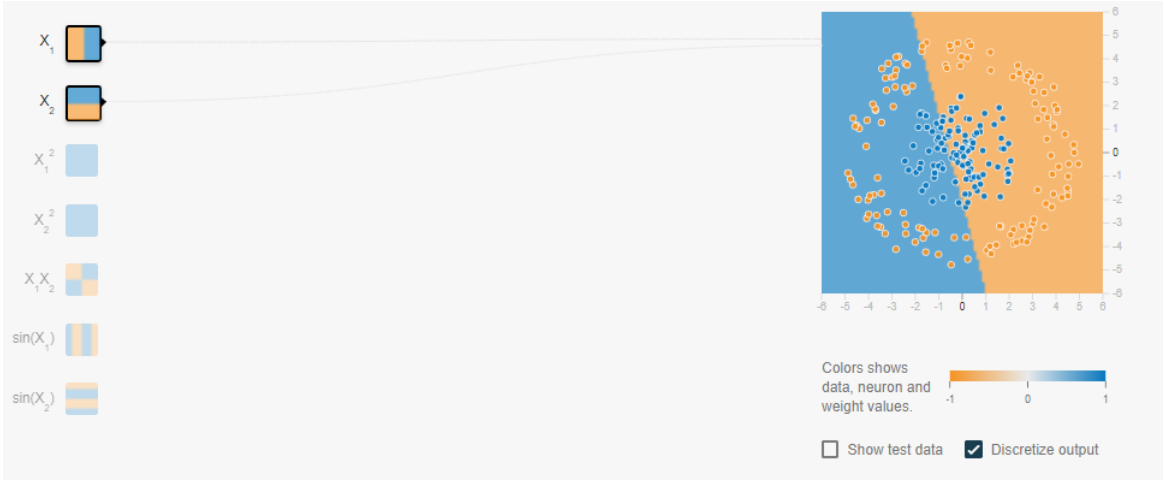
$$\begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}.$$

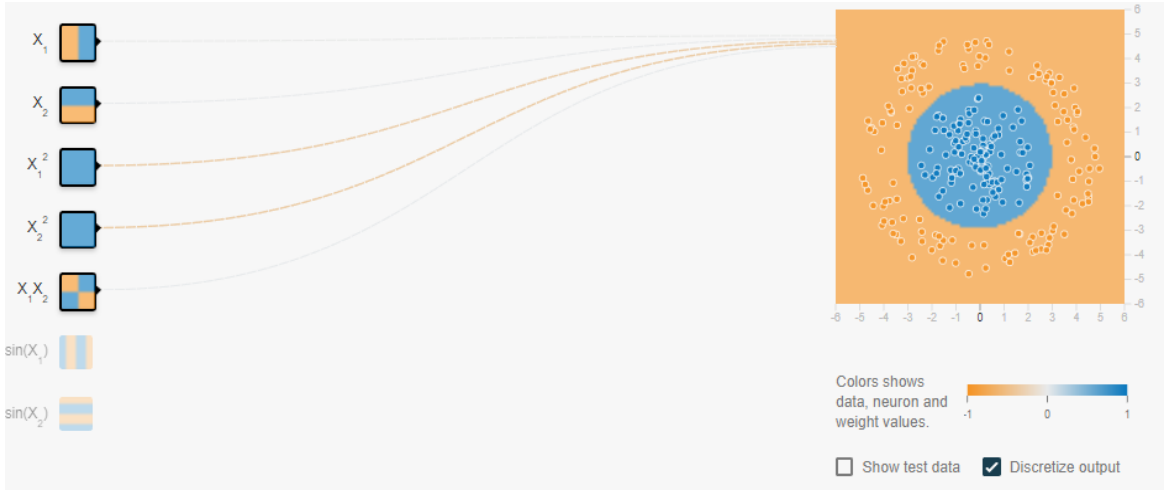


特征 or 激活

http://playground.tensorflow.org

如何处理非线性可分问题?





激活函数的性质

- 连续并可导（允许少数点上不可导）的非线性函数。
  - 可导的激活函数可以直接利用数值优化的方法来学习网络参数。
- 激活函数及其导函数要尽可能的简单
  - 有利于提高网络计算效率。
- 激活函数的导函数的值域要在一个合适的区间内
  - 不能太大也不能太小，否则会影响训练的效率和稳定性。

Sigmoid型激活函数

- **Sigmoid型函数**
- Sigmoid 型函数是指一类 S 型曲线函数，为两端饱和函数.
- 常用的 Sigmoid 型函数有Logistic函数和Tanh函数.

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

Logistic 函数的性质

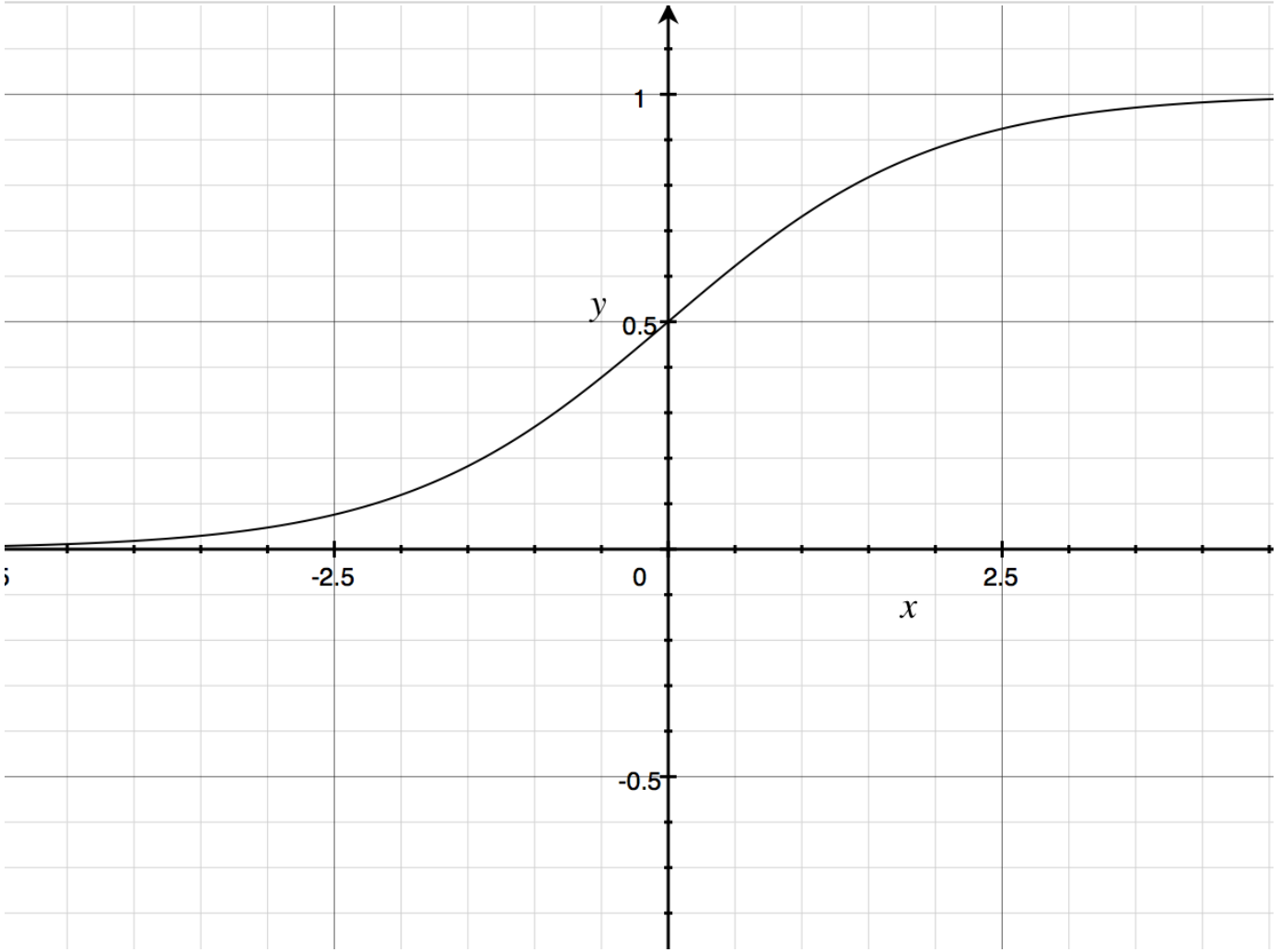
- Logistic 函数可以看成是一个“挤压”函数，把一个实数域的输入“挤压”到 $(0, 1)$. 当输入值在0附近时，Sigmoid型函数近似为线性函数；当输入值靠近两端时，对输入进行抑制. 输入越小，越接近于 0；输入越大，越接近于 1.

- 这样的特点 也和生物神经元类似，对一些输入会产生兴奋（输出为1），对另一些输入产生抑制（输出为0）.

- 和感知器使用的阶跃激活函数相比，Logistic函数是连续可导的，其数学性质更好.

Logistic

$$y=\frac{1}{1+e^{-x}}$$



Logistic 函数

数学小知识 | 饱和

对于函数 $f(x)$，若 $x \to -\infty$ 时，其导数 $f'(x) \to 0$，则称其为左饱和. 若 $x \to +\infty$ 时，其导数 $f'(x) \to 0$，则称其为右饱和. 当同时满足左、右饱和时，就称为两端饱和.

因为Logistic函数的性质，使得装备了Logistic激活函数的神经元具有以下两点性质：

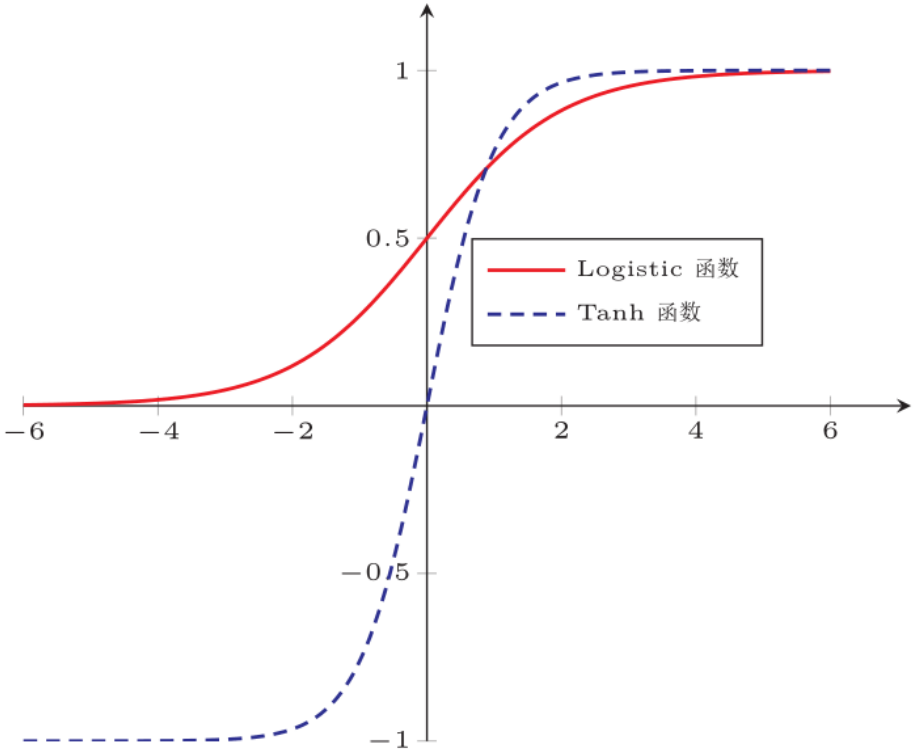
1）其输出直接可以看作概率分布，使得神经网络可以更好地和统计学习模型进行结合.

2）其可以看作一个软性门（Soft Gate），用来控制其他神经元输出信息的数量.

Tanh函数

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

Logistic 函数的输出恒大于 0，非零中心化的输出会使得其后一层的神经元的输入发生偏置偏移（bias shift），并进一步使得梯度下降的收敛速度变慢。



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

- 性质：
  - 饱和函数
  - Tanh函数是零中心化的，而logistic函数的输出恒大于0

Hard-Logistic函数

- Logistic函数和Tanh函数都是Sigmoid型函数，具有饱和性，但是计算开销较大．因为这两个函数都是在中间（0附近）近似线性，两端饱和．因此，这两个函数可以通过分段函数来近似．
- 以 Logistic 函数 在0附近的一阶泰勒展开（Taylor expansion）

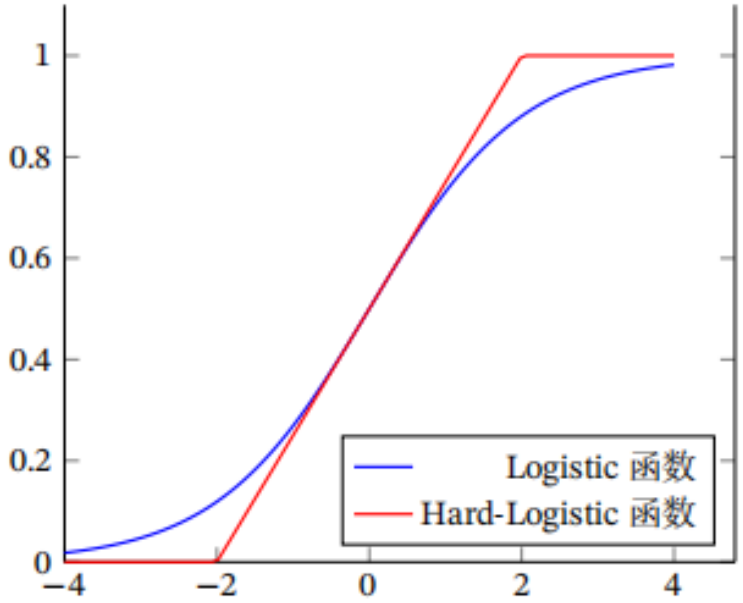
$$g_l(x) \approx \sigma(0) + x \times \sigma'(0)$$

$$= 0.25x + 0.5.$$

Hard-Logistic函数

- 这样Logistic函数可以用分段函数hard-logistic(x)来近似.

$$\text{hard-logistic}(x) = \begin{cases} 1 & g_l(x) \geq 1 \\ g_l & 0 < g_l(x) < 1 \\ 0 & g_l(x) \leq 0 \end{cases}$$
$$= \max\big(\min(g_l(x), 1), 0\big)$$
$$= \max\big(\min(0.25x + 0.5, 1), 0\big).$$



(a) Hard Logistic 函数

ReLU函数

- ReLU（Rectified Linear Unit，修正线性单元）[Nair et al., 2010]，也叫 Rectifier函数[Glorot et al., 2011]，是目前深度神经网络中经常使用的激活函数.
- ReLU实际上是一个斜坡（ramp）函数，定义为:

$$\begin{aligned}\mathrm{ReLU}(x) &= \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \\ &= \max(0, x).\end{aligned}$$

Relu的优点

- 采用 ReLU 的神经元只需要进行加、乘和比较的操作，计算上更加高效.
- ReLU 函数也被认为具有生物学合理性（Biological Plausibility）：
  - 单侧抑 制、
  - 宽兴奋边界（即兴奋程度可以非常高）.
  - 很好的稀疏性
- 在优化方面，相比于Sigmoid型函数的两端饱和，ReLU函数为左饱和函数，且在x> 0 时导数为1，在一定程度上缓解了神经网络的梯度消失问题，加速梯度下降的收敛速度.

Relu的缺点

- ReLU 函数的输出是非零中心化的，给后一层的神经网络引入偏置偏移，会影响梯度下降的效率.
- ReLU 神经元在训练时比较容易“死亡”. 在训练时，如果参数在一次不恰当的更新后，第一个隐藏层中的某个 ReLU 神经元在所有的训练数据上都不能被激活，那么这个神经元自身参数的梯度永远都会是 0. 在以后的训练过程中永远不能被激活. 这种现象称为死亡 ReLU 问题（Dying ReLU Problem），并且也有可能会发生在其他隐藏层.

- 在实际使用中，为了避免上述情况，有几种ReLU的变种也会被广泛使用.

带泄露的ReLU

- 带泄露的ReLU（Leaky ReLU）在输入 $x < 0$ 时，保持一个很小的梯度．这样当神经元非激活时也能有一个非零的梯度可以更新参数，避免永远不能被激活[Maas et al., 2013]．带泄露的ReLU的定义如下：

$$\begin{aligned}\text{LeakyReLU}(x) &= \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases} \\ &= \max(0, x) + \gamma \min(0, x),\end{aligned}$$

- 其中γ是一个很小的常数，比如0.01．当 $\gamma < 1$ 时，带泄露的ReLU也可以写为 $\text{LeakyReLU}(x) = \max(x, \gamma x)$

带参数的ReLU

- 带参数的 ReLU（Parametric ReLU，PReLU）引入一个可学习的参数不同神经元可以有不同的参数．对于第i个神经元，其 PReLU的定义为：

$$\begin{aligned}\mathrm{PReLU}_i(x) &= \begin{cases} x & \text{if } x > 0 \\ \gamma_i x & \text{if } x \le 0 \end{cases} \\ &= \max(0, x) + \gamma_i \min(0, x)\end{aligned}$$

- 其中 γi为x≤0 时函数的斜率．因此，PReLU 是非饱和函数．如果γi＝0，那么PReLU就退化为ReLU.
- 如果γi为一个很小的常数，则PReLU可以看作带泄露的ReLU.
- PReLU 可以允许不同神经元具有不同的参数，也可以一组神经元共享一个参数。

扩展

ELU（Exponential Linear Unit，指数线性单元）[Clevert et al., 2015]是一个近似的零中心化的非线性函数，其定义为

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma(\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \tag{4.23}$$

$$= \max(0, x) + \min(0, \gamma(\exp(x) - 1)), \tag{4.24}$$

其中$\gamma \geq 0$是一个超参数，决定$x \leq 0$时的饱和曲线，并调整输出均值在0附近.

Softplus 函数[Dugas et al., 2001]可以看作 Rectifier 函数的平滑版本，其定义为

$$\text{Softplus}(x) = \log(1 + \exp(x)). \tag{4.25}$$

Softplus 函数其导数刚好是 Logistic 函数. Softplus 函数虽然也具有单侧抑制、宽兴奋边界的特性，却没有稀疏激活性.

常见激活函数

$$\text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$
$$= \max(0, x).$$

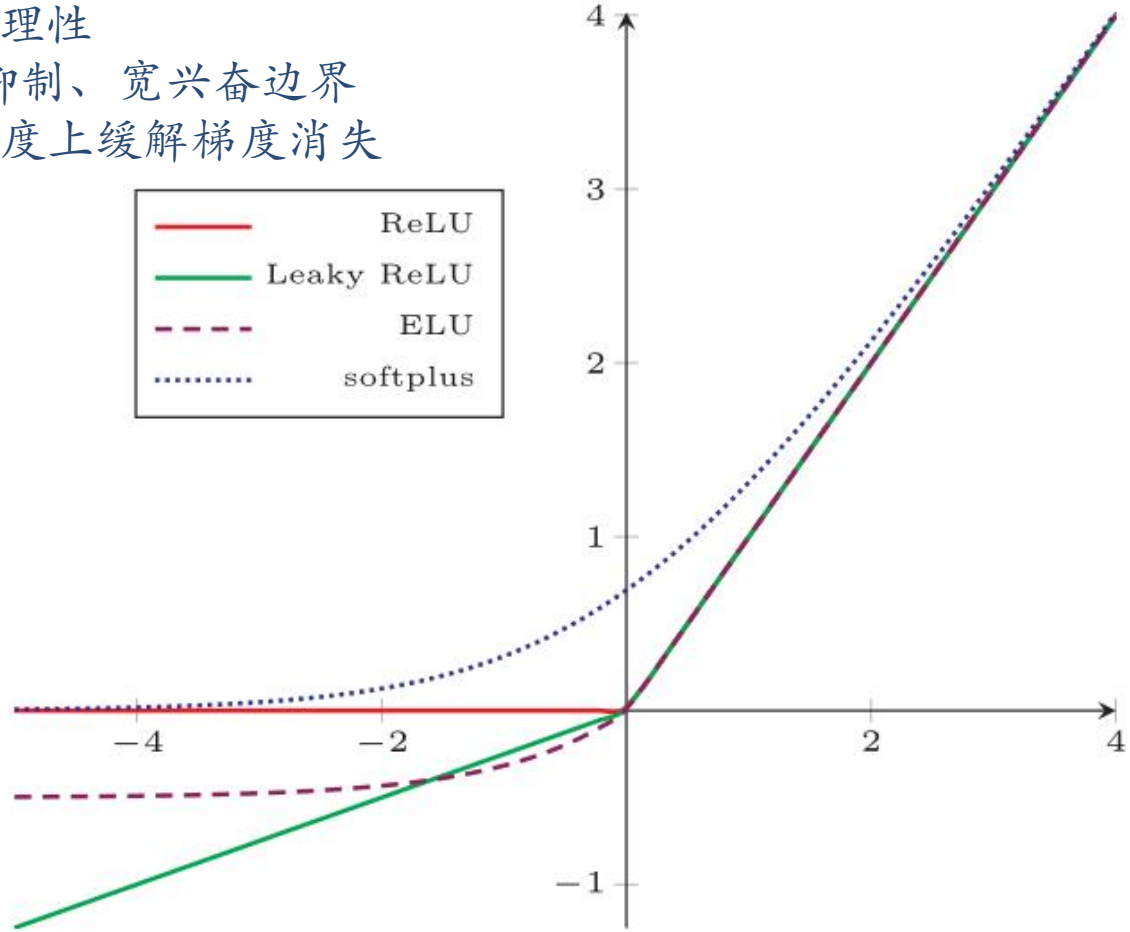
$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases}$$

$$\text{PReLU}_i(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma_i x & \text{if } x \leq 0 \end{cases}$$

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$
$$= \max(0, x) + \min(0, \gamma(\exp(x) - 1))$$

$$\text{softplus}(x) = \log(1 + \exp(x))$$

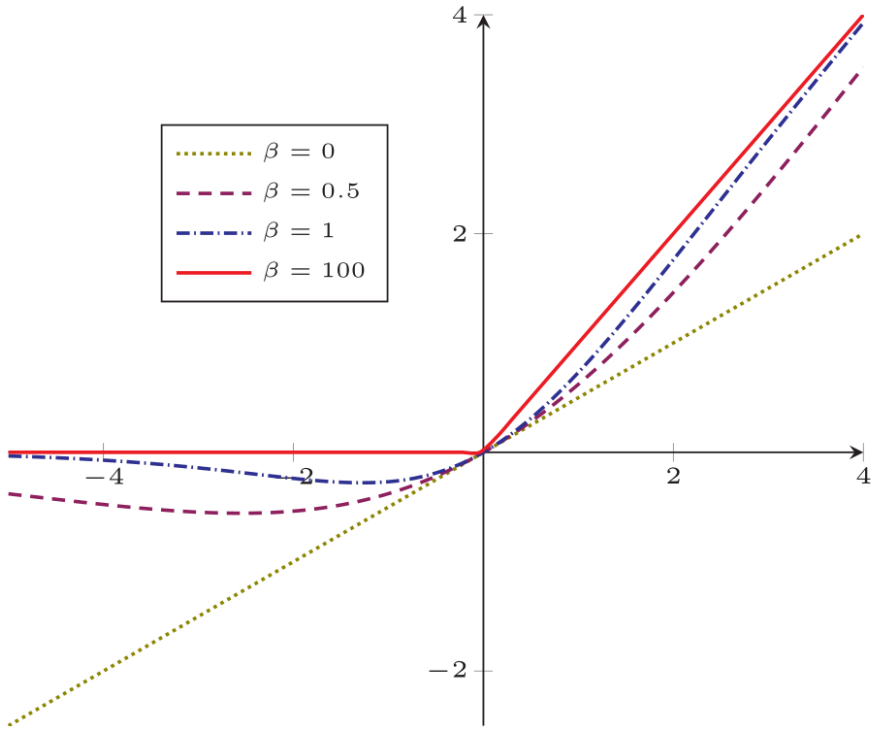
- 计算上更加高效
- 生物学合理性
  - 单侧抑制、宽兴奋边界
- 在一定程度上缓解梯度消失问题



死亡ReLU问题（Dying ReLU Problem）

常见激活函数

Swish函数 $\text{swish}(x) = x\sigma(\beta x)$



常见激活函数及其导数

激活函数	函数	导数
Logistic 函数	$f(x) = \frac{1}{1+\exp(-x)}$	$f'(x) = f(x)\big(1 - f(x)\big)$
Tanh 函数	$f(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	$f'(x) = 1 - f(x)^2$
ReLU 函数	$f(x) = \max(0, x)$	$f'(x) = I(x > 0)$
ELU 函数	$f(x) = \max(0, x) + \min\Big(0, \gamma\big(\exp(x) - 1\big)\Big)$	$f'(x) = I(x > 0) + I(x \leq 0) \cdot \gamma \exp(x)$
SoftPlus 函数	$f(x) = \log\big(1 + \exp(x)\big)$	$f'(x) = \frac{1}{1+\exp(-x)}$

感知器

早期的MP神经元和Rosenblatt感知器

Psychological Review
Vol. 65, No. 6, 1958

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

F. ROSENBLATT

Cornell Aeronautical Laboratory

HAVING told you about the giant digital computer known as I.B.M. 704 and how it has been taught to play a fairly creditable game of chess, we'd like to tell you about an even more remarkable machine, the perceptron, which, as its name implies, is capable of what amounts to original thought. The first perceptron has yet to be built,

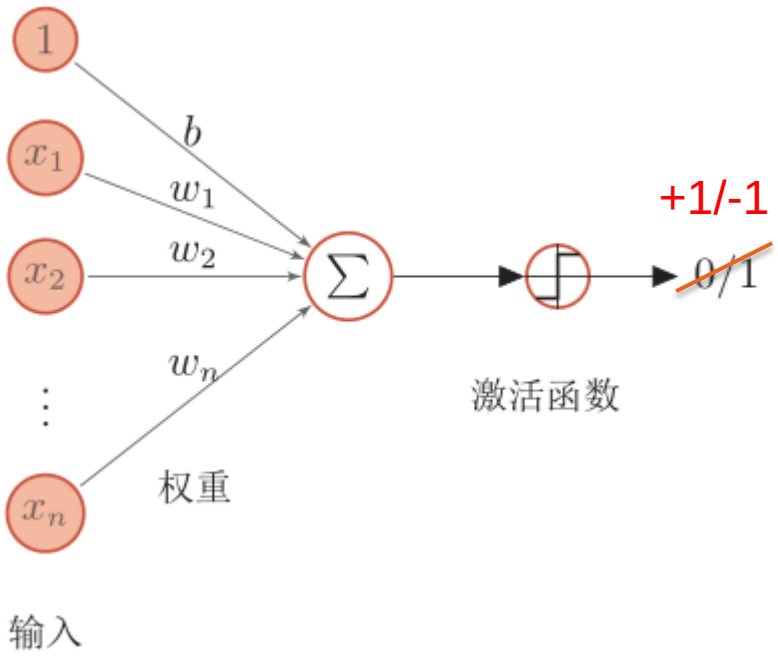
The New Yorker, December 6, 1958 P. 44

The IBM 704 computer

感知器

- 模拟生物神经元行为的机器，有与生物神经元相对应的部件，如权重（突触）、偏置（阈值）及激活函数（细胞体），输出为+1或-1。

$$\hat{y} = \begin{cases} +1 & \text{当} \mathbf{w}^{\mathrm{T}}\mathbf{x} > 0 \\ -1 & \text{当} \mathbf{w}^{\mathrm{T}}\mathbf{x} \leq 0 \end{cases},$$



感知器

- 学习算法
  - 一种错误驱动的在线学习算法：
  - 先初始化一个权重向量$W = 0$（通常是全零向量）；
  - 每次分错一个样本(X, y)时，即$yW_TX < 0$
  - 用这个样本来更新权重

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$$

  - 根据感知器的学习策略，可以反推出感知器的损失函数为

$$\mathcal{L}(\boldsymbol{w}; \boldsymbol{x}, y) = \max(0, -y\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})$$

感知器的学习过程

算法 3.1 两类感知器的参数学习算法

输入: 训练集 $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，最大迭代次数 T

初始化：$\boldsymbol{w}_0 \leftarrow 0, k \leftarrow 0, t \leftarrow 0$；
repeat
  对训练集 $\mathcal{D}$ 中的样本随机排序；
  for $n = 1 \cdots N$ **do**
    选取一个样本 $(\boldsymbol{x}^{(n)}, y^{(n)})$；
    if $\boldsymbol{w}_k^{\mathrm{T}}(y^{(n)}\boldsymbol{x}^{(n)}) \leq 0$ **then**
      $\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k + y^{(n)}\boldsymbol{x}^{(n)}$；
      $k \leftarrow k + 1$；
    end
    $t \leftarrow t + 1$；
    if $t = T$ **then** break； // 达到最大迭代次数
  end
until $t = T$；
输出: $\boldsymbol{w}_k$

表示分错

对比Logistic回归的更新方式：

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x}^{(n)} \left(y^{(n)} - \hat{y}_{\boldsymbol{w}_t}^{(n)} \right)$$

神经网络

别名

- 深度前馈网络（deep feedforward network），
- 也叫作 前馈神经网络（feedforward neural network）或者
- 多层感知机（multilayer perceptron, MLP），
- 是典型的深度学习模型。

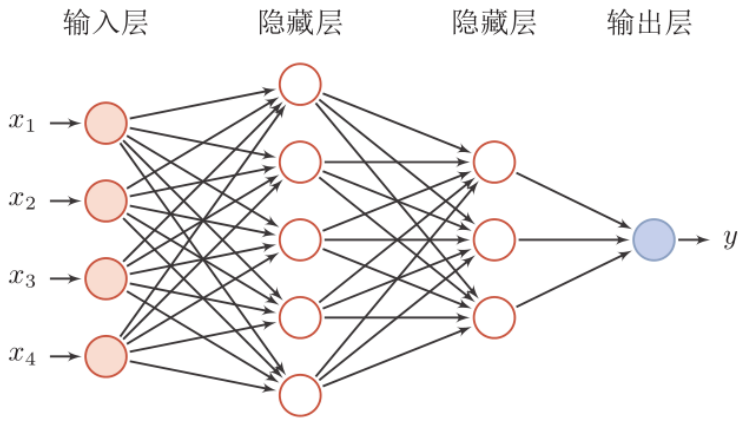
- 这种模型被称为 前向（feedforward）的，是因为信息流过 $\boldsymbol{x}$ 的函数，流经用于定义 f 的中间计算过程，最终到达输出 $\boldsymbol{y}$。在模型的输出和模型本身之间没有 反馈 feedback）连接。
- 当前馈神经网络被扩展成包含反馈连接时，它们被称为 循环神经网络（recurrent neural network）。

神经网络 的 特性

- 神经网络最早是作为一种主要的连接主义模型。
  - 20世纪80年代后期，最流行的一种连接主义模型是分布式并行处理（Parallel Distributed Processing，PDP）网络，其有3个主要特性：
    - 1）信息表示是分布式的（非局部的）；
    - 2）记忆和知识是存储在单元之间的连接上；
    - 3）通过逐渐改变单元之间的连接强度来学习新的知识。
  - 引入误差反向传播来改进其学习能力之后，神经网络也越来越多地应用在各种机器学习任务上。

神经网络的组成

- 前馈神经网络（全连接神经网络、多层感知器）
  - 各神经元分别属于不同的层，层内无连接。
  - 相邻两层之间的神经元全部两两连接。
  - 整个网络中无反馈，信号从输入层向输出层单向传播，可用一个有向无环图表示。



神经网络的组成

- 前馈网络的最后一层被称为 输出层（output layer）
- 因为训练数据并没有给出这些中间层所需的输出，所以这些层被称为 隐藏层（hidden layer）。
- 隐藏层的维数决定了模型的 宽度（width）。
- 链的全长称为模型的 深度（depth）。

ANN的万能近似性

万能近似定理(universal approximation theorem)： 一个前馈神经网络如果具有线性输出层和至少一层具有任何一种 “挤压” 性质的激活函数的隐藏层，只要给予网络足够数量的隐藏单元，它可以以任意的精度来近似任何从一个有限维空间到另一个有限维空间的 Borel 可测函数。在 Rn 的有界闭集上的任意连续函数是 Borel 可测的， 因此可以用神经网络来近似。

Author:WangQi
Email:wangqikaixin@gmail.com

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 $\mathbf{w}_i \in \mathbb{R}^d$，$i = 1, \cdots, m$，以至于我们可以定义函数

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

作为函数 f 的近似实现，即

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

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。

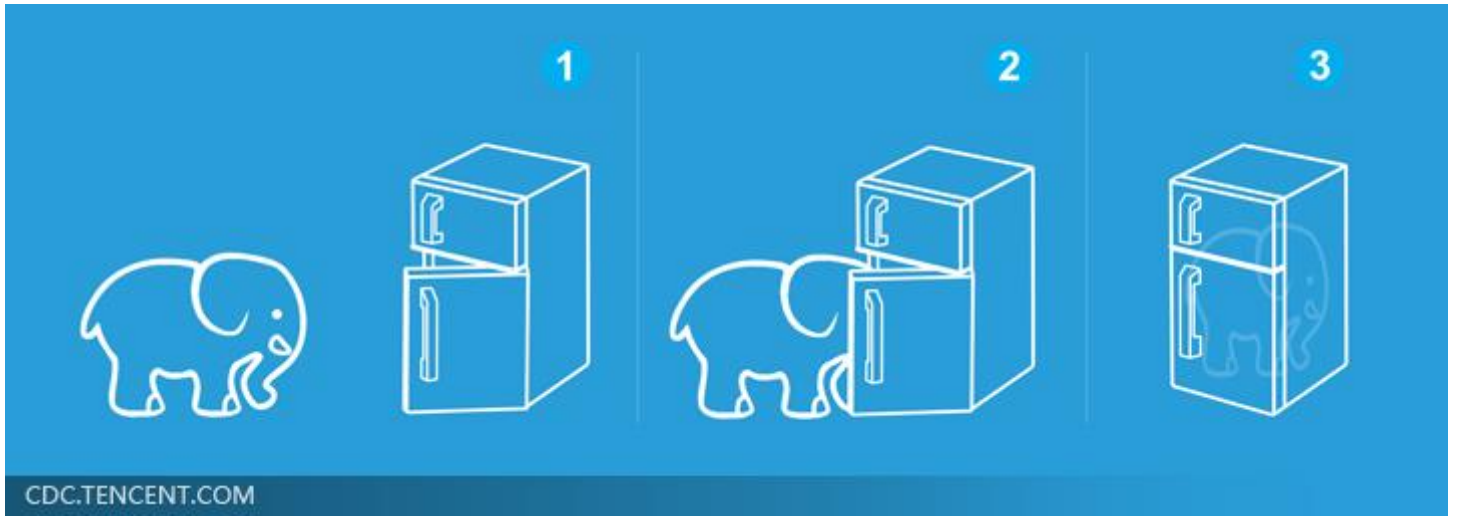
人工神经网络

- 人工神经网络主要由大量的神经元以及它们之间的有向连接构成。因此考虑三方面：

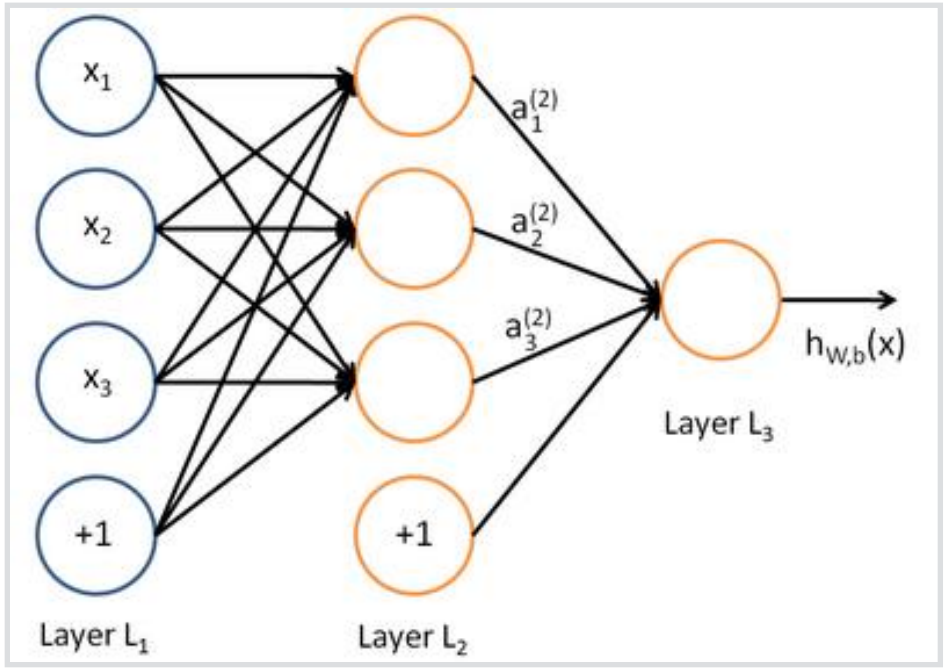
- 神经元的激活规则
  - 主要是指神经元输入到输出之间的映射关系，一般为非线性函数。
- 网络的拓扑结构
  - 不同神经元之间的连接关系。
- 学习算法
  - 通过训练数据来学习神经网络的参数。

深度学习的三个步骤

Deep Learning is so simple ……



数学表达，复合函数



$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

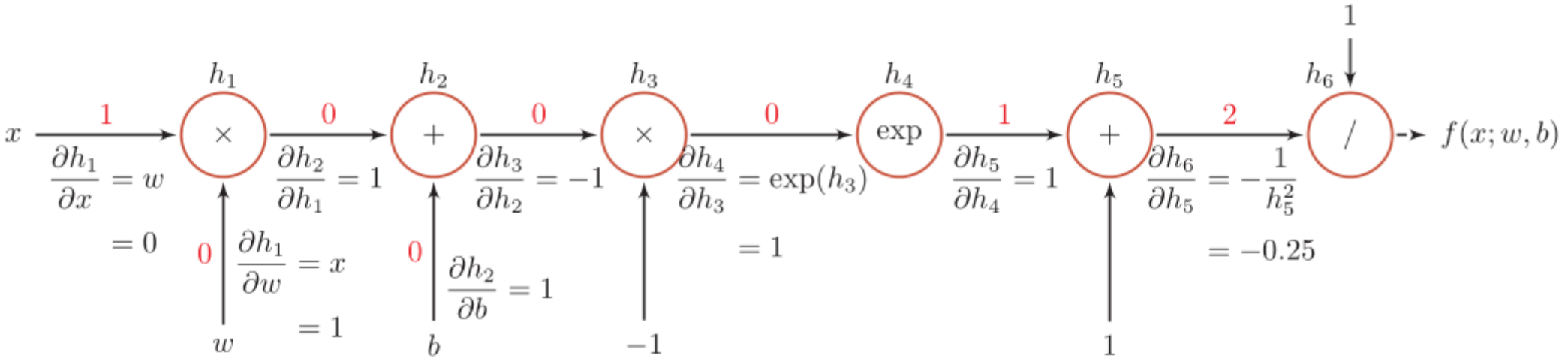
$$h_{w,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

计算图与自动微分

计算图

- 计算图

$$f(x;w,b)=\frac{1}{\exp(-(wx+b))+1}.$$



链式法则

- 链式法则（Chain Rule）是在微积分中求复合函数导数的一种常用方法。

（1）若 $x \in \mathbb{R}, \boldsymbol{u} = u(x) \in \mathbb{R}^s, \boldsymbol{g} = g(\boldsymbol{u}) \in \mathbb{R}^t$，则

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

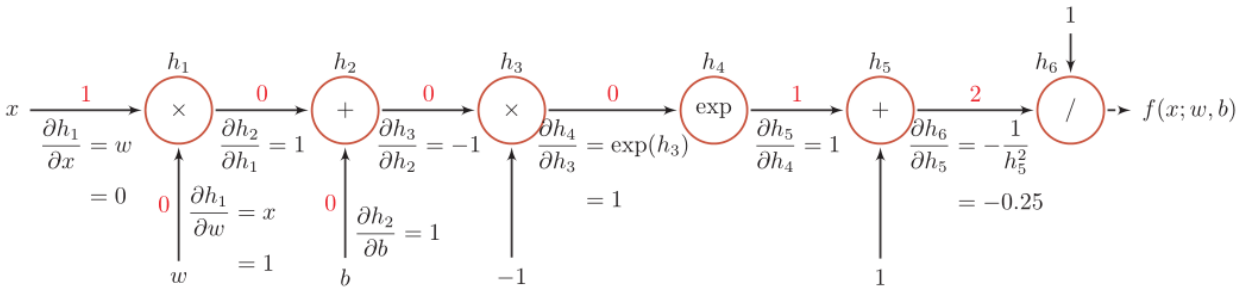
（2）若 $\boldsymbol{x} \in \mathbb{R}^p, \boldsymbol{y} = g(\boldsymbol{x}) \in \mathbb{R}^s, \boldsymbol{z} = f(\boldsymbol{y}) \in \mathbb{R}^t$，则

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

（3）若 $X \in \mathbb{R}^{p \times q}$ 为矩阵，$\boldsymbol{y} = g(X) \in \mathbb{R}^s, z = f(\boldsymbol{y}) \in \mathbb{R}$，则

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

计算图



函数	导数	
$h_1 = x \times w$	$\frac{\partial h_1}{\partial w} = x$	$\frac{\partial h_1}{\partial x} = w$
$h_2 = h_1 + b$	$\frac{\partial h_2}{\partial h_1} = 1$	$\frac{\partial h_2}{\partial b} = 1$
$h_3 = h_2 \times -1$	$\frac{\partial h_3}{\partial h_2} = -1$	
$h_4 = \exp(h_3)$	$\frac{\partial h_4}{\partial h_3} = \exp(h_3)$	
$h_5 = h_4 + 1$	$\frac{\partial h_5}{\partial h_4} = 1$	
$h_6 = 1/h_5$	$\frac{\partial h_6}{\partial h_5} = -\frac{1}{h_5^2}$	

当x = 1,w = 0,b = 0时，可以得到

$$\begin{aligned}\frac{\partial f(x;w,b)}{\partial w}|_{x=1,w=0,b=0} &= \frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w} \\ &= 1 \times -0.25 \times 1 \times 1 \times -1 \times 1 \times 1 \\ &= 0.25.\end{aligned}$$

自动微分

- 自动微分是利用链式法则来自动计算一个复合函数的梯度。

- 前向模式和反向模式
  - 反向模式和反向传播的计算梯度的方式相同

- 如果函数和参数之间有多条路径，可以将这多条路径上的导数再进行相加，得到最终的梯度。

前向传播——信息传递过程

- 前馈神经网络通过下面公式进行信息传播。

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)},$$
$$\boldsymbol{a}^{(l)} = f_l(\boldsymbol{z}^{(l)}).$$

- 前馈计算：

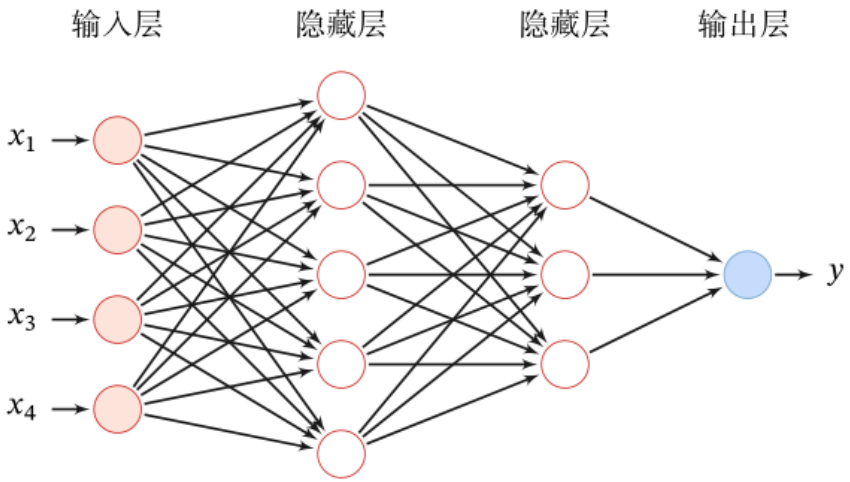
$$\boldsymbol{x} = \boldsymbol{a}^{(0)} \to \boldsymbol{z}^{(1)} \to \boldsymbol{a}^{(1)} \to \boldsymbol{z}^{(2)} \to \cdots \to \boldsymbol{a}^{(L-1)} \to \boldsymbol{z}^{(L)} \to \boldsymbol{a}^{(L)} = \phi(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}))$$

反向传播算法 (自动微分的反向模式)

- 前馈神经网络的训练过程可以分为以下三步
  - 前向计算每一层的状态和激活值，直到最后一层
  - 反向计算每一层的参数的偏导数
  - 更新参数

- 解决了贡献度问题。

前馈网络符号定义



给定一个前馈神经网络，用下面的记号来描述这样网络：

记号	含义
L	神经网络的层数
M_l	第 l 层神经元的个数
$f_l(\cdot)$	第 l 层神经元的激活函数
$\boldsymbol{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$	第 $l-1$ 层到第 l 层的权重矩阵
$\boldsymbol{b}^{(l)} \in \mathbb{R}^{M_l}$	第 $l-1$ 层到第 l 层的偏置
$\boldsymbol{z}^{(l)} \in \mathbb{R}^{M_l}$	第 l 层神经元的净输入（净活性值）
$\boldsymbol{a}^{(l)} \in \mathbb{R}^{M_l}$	第 l 层神经元的输出（活性值）

反向传播算法

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

$$\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} = \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}}\right]$$

$$= \left[0, \cdots, \frac{\partial(\mathbf{w}_{i:}^{(l)}\mathbf{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \cdots, 0\right]$$

$$= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0\right]$$

$$\triangleq \mathbb{I}_i(a_j^{(l-1)}) \quad \in \mathbb{R}^{m^{(l)}},$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

$$\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \quad \in \mathbb{R}^{m^{(l)}}$$

误差项

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{I}_{m^{(l)}} \quad \in \mathbb{R}^{m^{(l)} \times m^{(l)}}$$

计算 $\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \in \mathbb{R}^{m^{(l)}}$

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

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

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \frac{\partial f_l(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}} = \mathrm{diag}(f_l'(\mathbf{z}^{(l)}))$$

$$\frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} = (W^{(l+1)})^{\mathrm{T}}$$

$$\begin{aligned}
\delta^{(l)} &\triangleq \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \\
&= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}} \\
&= \mathrm{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^{\mathrm{T}} \cdot \delta^{(l+1)} \\
&= f_l'(\mathbf{z}^{(l)}) \odot \left((W^{(l+1)})^{\mathrm{T}} \delta^{(l+1)}\right),
\end{aligned}$$

反向传播算法

在计算出上面三个偏导数之后，公式 (4.49) 可以写为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \mathbb{I}_i(a_j^{(l-1)})\delta^{(l)} = \delta_i^{(l)} a_j^{(l-1)}.$$

进一步，$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 关于第 l 层权重 $W^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial W^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^{\mathrm{T}}.$$

同理，$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ 关于第 l 层偏置 $\mathbf{b}^{(l)}$ 的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}.$$

静态计算图和动态计算图

- 静态计算图是在编译时构建计算图，计算图构建好之后在程序运行时不能改变。
  - Theano和Tensorflow
- 动态计算图是在程序运行时动态构建。两种构建方式各有优缺点。
  - DyNet，Chainer和PyTorch

- 静态计算图在构建时可以进行优化，并行能力强，但灵活性比较低。动态计算图则不容易优化，当不同输入的网络结构不一致时，难以并行计算，但是灵活性比较高。

谢 谢