第5章 神经网络

中国矿业大学 计算机科学与技术学院

Neural networks

神经网络是一个很大的学科，本课程仅讨论它与机器学习的交集。

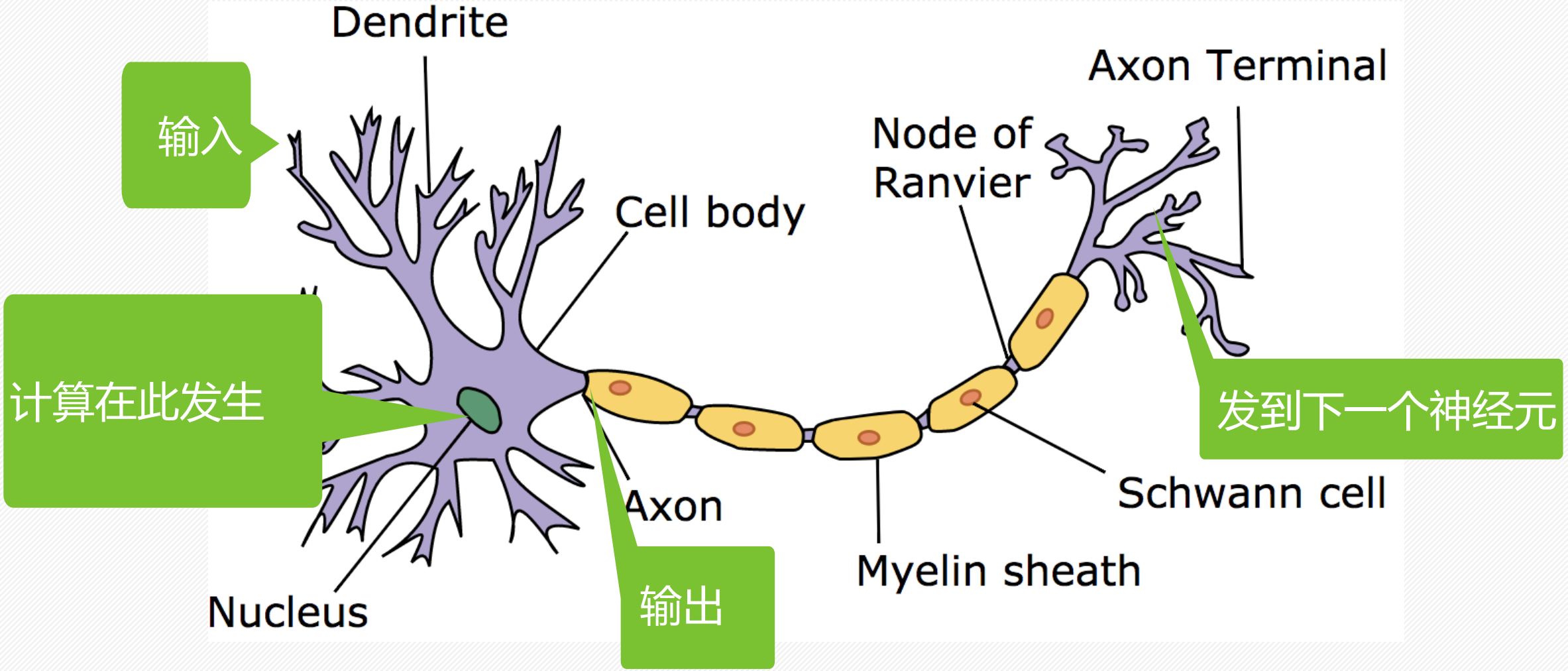
神经网络学得的知识蕴含在连接权与阈值中。

Neural networks are massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organizations which are intended to interact with the objects of the real world in the same way as biological nervous systems do.

[T. Kohonen, NN88]

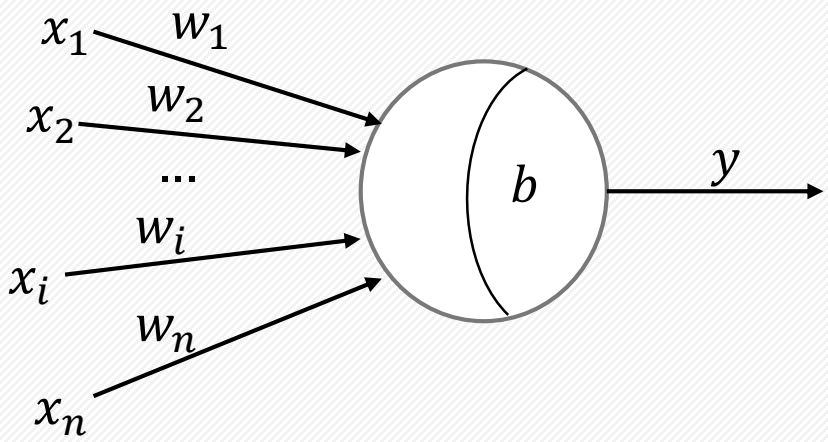
神经科学的灵感

生物神经元



5.1 神经元模型

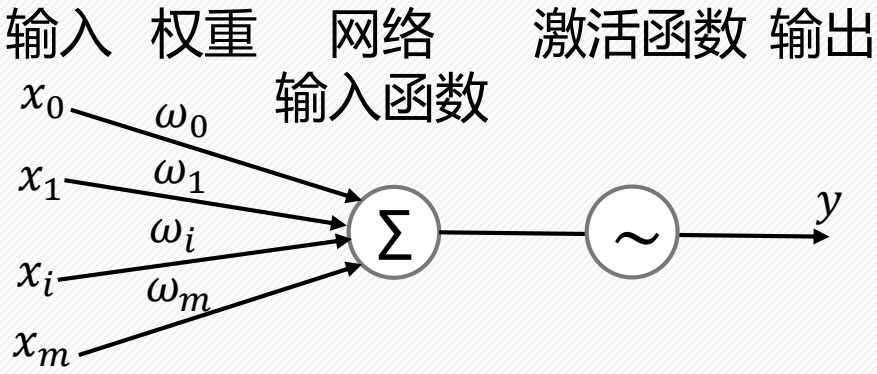
M-P 神经元模型 [McCulloch and Pitts, 1943]



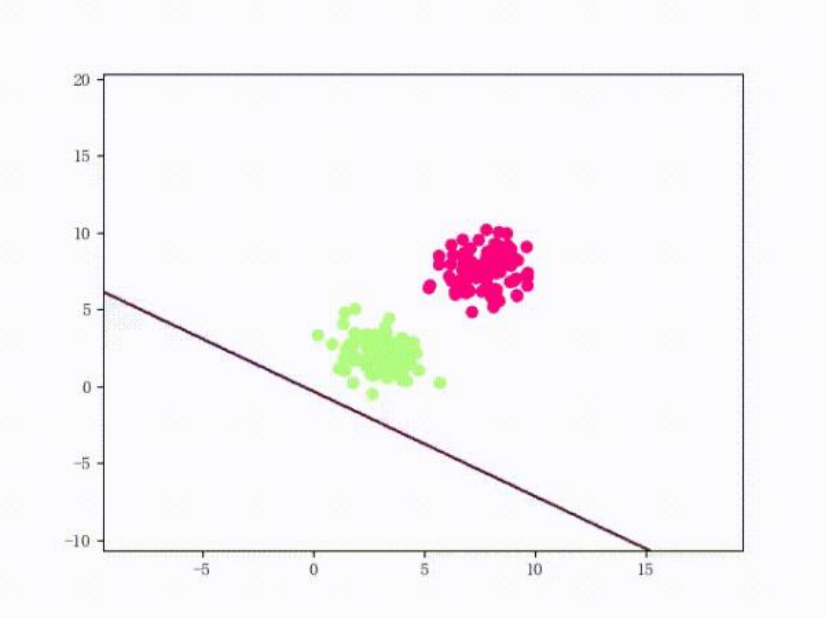
$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

5.2 感知机与多层网络

■**感知机(perceptron):** 基于M-P神经元模型发展而来的人工神经网络模型。

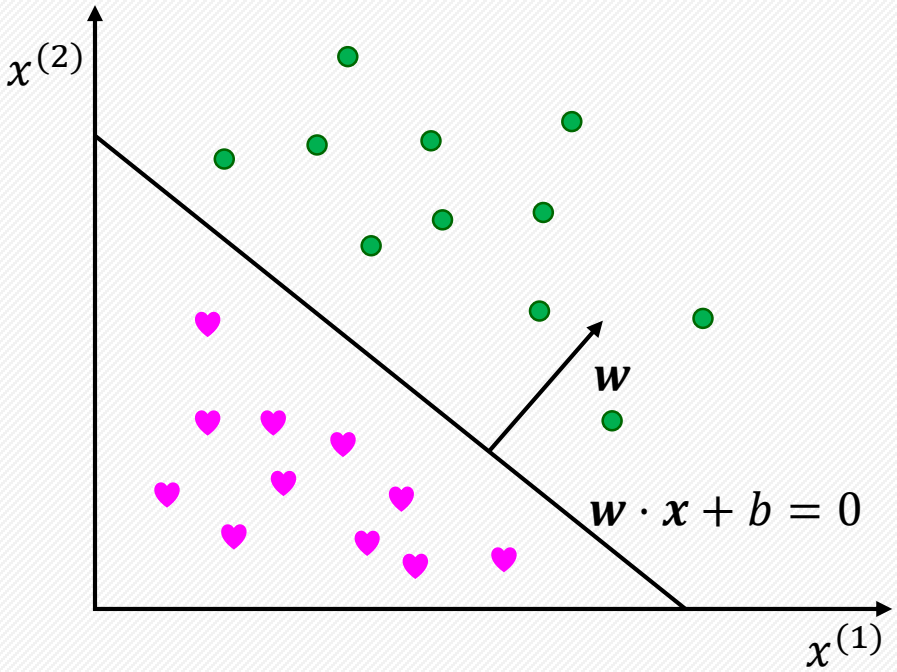


$$f(x) = \text{sign}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$



感知机是二分类的线性分类模型，其输入为实例的特征向量，输出为实例的类别。感知机对应于输入空间中将实例划分为正负两类的分离超平面。

5.2 感知机与多层网络



$$f(x) = sign(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

$$sign(x) = \begin{cases} -1 & x < 0 \\ +1 & x \geq 0 \end{cases}$$

对于分类正确的样本$(\boldsymbol{x_i}, y_i)$，当y_i=1时，预测$\boldsymbol{w} \cdot \boldsymbol{x_i} + b > 0$，当$y_i = -1$时，预测$\boldsymbol{w} \cdot \boldsymbol{x_i} + b < 0$

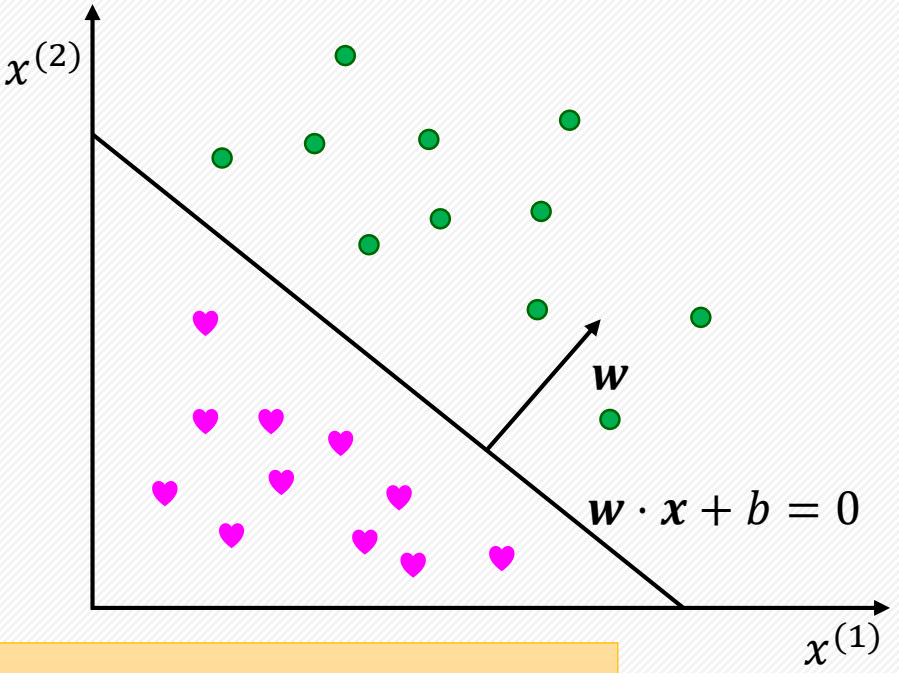
对于分类错误的样本$(\boldsymbol{x_i}, y_i)$，当y_i=1时，预测$\boldsymbol{w} \cdot \boldsymbol{x_i} + b < 0$，当$y_i = -1$时，预测$\boldsymbol{w} \cdot \boldsymbol{x_i} + b > 0$

损失函数

$$L(\boldsymbol{w}, b) = -\frac{1}{\|\boldsymbol{w}\|} \sum_{\boldsymbol{x_i} \in M} y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b)$$

$$L(\boldsymbol{w}, b) = -\sum_{\boldsymbol{x_i} \in M} y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b)$$

例5.1，求任意一点x到超平面$w \cdot x + b = 0$的距离d。



解：设x在平面S上的投影点为x'，则

$$w \cdot x' + b = 0$$

∵向量$\overrightarrow{xx_0}$与直线S的法向量平行

$$\left|w \cdot \overrightarrow{xx'}\right| = \|w\| \cdot \left\|\overrightarrow{xx'}\right\| = \|w\| \cdot d$$

$$\begin{aligned} \because \left|w \cdot \overrightarrow{xx'}\right| &= |w_1 (x_1 - x_1') + w_2 (x_2 - x_2') + \cdots + w_n (x_n - x_n')| \\ &= |(w_1 x_1 + w_2 x_2 + \cdots + w_n x_n) - (w_1 x_1' + w_2 x_2' + \ldots + w_n x_n')| \\ &= |w \cdot x + b - (w \cdot x' + b)| \\ &= |w \cdot x + b| \end{aligned}$$

$$\therefore d = \frac{|w \cdot x + b|}{\|w\|}$$

$$d_{\text{误分类}} = -\frac{y_i (w \cdot x_i + b)}{\|w\|}$$


ZHONGXUESHUXUEZAZHI 中学数学杂志 2009年第1期

点到直线的距离公式的十三种证明方法

湖北省鄂州市沼山中学 436061 余树林 袁新宝

算法5.1 感知器学习算法的原始形式

输入：

- 训练集$S=(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_N,y_N)$
- 学习率$\eta(0<\eta\leq 1)$

输出：权重向量$\boldsymbol{w}$和偏置b

过程：

第1步：初始化权重$\boldsymbol{w}$和偏置b

第2步：随机选择样本$(\boldsymbol{x}_i,y_i)\in S$

第3步：如果是误分类样本，则进行更新

if $y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i+b)\leq 0$ then

$$\boldsymbol{w}\leftarrow\boldsymbol{w}+\eta y_i\boldsymbol{x}_i$$

$$b\leftarrow b+\eta y_i$$

第4步：转到步骤2，直至训练集中没有误分类样本，

得到感知机模型$f(\boldsymbol{x})=\mathrm{sign}(\boldsymbol{w}\cdot\boldsymbol{x}+b)$

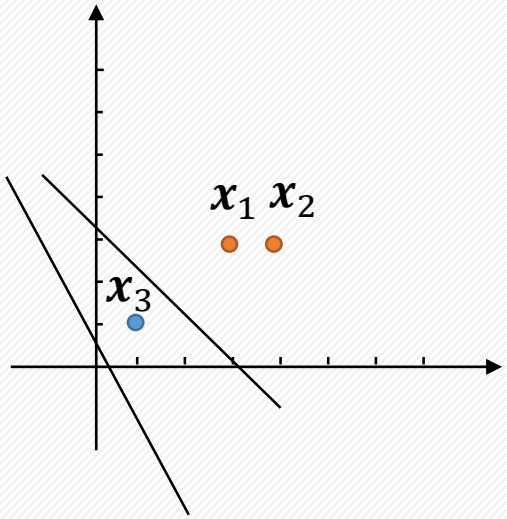
$$L(\boldsymbol{w},b)=-\sum_{\boldsymbol{x}_i\in M}y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i+b)$$

$$\frac{\partial L(\boldsymbol{w},b)}{\partial\boldsymbol{w}}=\frac{\partial}{\partial\boldsymbol{w}}\left[-\sum_{\boldsymbol{x}_i\in M}y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i+b)\right]=-\sum_{\boldsymbol{x}_i\in M}y_i\boldsymbol{x}_i$$

$$\frac{\partial L(\boldsymbol{w},b)}{\partial b}=\frac{\partial}{\partial b}\left[-\sum_{\boldsymbol{x}_i\in M}y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i+b)\right]=-\sum_{\boldsymbol{x}_i\in M}y_i$$

例5.2

例5.2 如右图所示的训练数据集，其正样本是$\boldsymbol{x}_1=(3,3)^{\mathrm{T}}, \boldsymbol{x}_2=(4,3)^{\mathrm{T}}$，负样本是$\boldsymbol{x}_3=(1,1)^{T}$，试用感知机算法求感知机模型$f(x)=sign(\boldsymbol{w}\cdot\boldsymbol{x}+b)$。这里，$\boldsymbol{w}=(w^{(1)},w^{(2)})^{T}$,$\boldsymbol{x}=(x^{(1)},x^{(2)})^{T}$



解 构建最优化问题：$\max\limits_{\boldsymbol{w},b} L(\boldsymbol{w},b)=-\sum\limits_{\boldsymbol{x_i}\in M} y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i+b)$

求解$\boldsymbol{w},b$，设$\eta=1$

(1) 取初值 $\boldsymbol{w}=(0,0)^{T}, b=0$

(2) 对于$\boldsymbol{x}_1=(3,3)^{\mathrm{T}}$,$y_1(\boldsymbol{w}\cdot\boldsymbol{x_1}+b)=0$, 未正确分类，更新$\boldsymbol{w},b$;

$\boldsymbol{w}=\boldsymbol{w}+y_1\boldsymbol{x_1}=(3,3)^{\mathrm{T}}, b=b+y_1=1$

得线性模型 $\boldsymbol{w}\cdot\boldsymbol{x}+b=3x^{(1)}+3x^{(2)}+1$

(3) 对于$\boldsymbol{x}_1,\boldsymbol{x}_2$，显然$y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i+b)>0$，被正确分类，不更新$\boldsymbol{w},b$;

对于$\boldsymbol{x}_3$，$y_3(\boldsymbol{w}\cdot\boldsymbol{x_3}+b)<0$，被错误分类，更新$\boldsymbol{w},b$;

$\boldsymbol{w}=\boldsymbol{w}+y_3\boldsymbol{x_3}=(2,2)^{\mathrm{T}}, b=b+y_1=0$

得线性模型 $\boldsymbol{w}\cdot\boldsymbol{x}+b=2x^{(1)}+2x^{(2)}$

$\boldsymbol{w}\leftarrow\boldsymbol{w}+\eta y_i\boldsymbol{x}_i$

$b\leftarrow b+\eta y_i$

例5.2 （续）

表5.1 例5.2 求解的迭代过程

迭代次数	误分样本	w	b	$w \cdot x + b$
0		$(0,0)^{\mathrm{T}}$	0	0
1	x_1	$(3,3)^{\mathrm{T}}$	1	$3x^{(1)} + 3x^{(2)} + 1$
2	x_3	$(2,2)^{\mathrm{T}}$	0	$2x^{(1)} + 2x^{(2)}$
3	x_3	$(1,1)^{\mathrm{T}}$	-1	$x^{(1)} + x^{(2)} - 1$
4	x_3	$(0,0)^{\mathrm{T}}$	-2	-2
5	x_1	$(3,3)^{\mathrm{T}}$	-1	$3x^{(1)} + 3x^{(2)} - 1$
6	x_3	$(2,2)^{\mathrm{T}}$	-2	$2x^{(1)} + 2x^{(2)} - 2$
7	x_3	$(1,1)^{\mathrm{T}}$	-3	$x^{(1)} + x^{(2)} - 3$
8	无	$(1,1)^{\mathrm{T}}$	-3	$x^{(1)} + x^{(2)} - 3$

直到 $\boldsymbol{w} = (1,1)^T, b = -3$， 此时所有数据点都被正确分类，不再有误分类点，损失函数达到极小。

分离超平面：$x^{(1)} + x^{(2)} - 3 = 0$

感知机模型：$f(\boldsymbol{x}) = \mathrm{sign}(x^{(1)} + x^{(2)} - 3)$

感知机算法的收敛性

将偏置b并入权重向量$\boldsymbol{w}$，记作$\hat{\boldsymbol{w}} = (\boldsymbol{w}^T, b)^T$，同样也将输入向量加以扩充，加进常数1，记作$\hat{\boldsymbol{x}} = (\boldsymbol{x}^T, 1)^T$。这样，$\hat{\boldsymbol{w}}, \hat{\boldsymbol{x}} \in \mathbb{R}^{n+1}$。显然，$\hat{\boldsymbol{w}} \cdot \hat{\boldsymbol{x}} = \boldsymbol{w} \cdot \boldsymbol{x} + b$。

定理2.1 (Novikoff) 设训练数据集$T = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_N, y_N)\}$是**线性可分**的，其中$\boldsymbol{x}_2 \in \mathcal{X} = \mathbb{R}^n$, $y_i \in \mathcal{Y} = \{-1, +1\}$，$i = 1,2 \cdots, N$，则

（1）存在满足条件$\|\hat{\boldsymbol{w}}_{\text{opt}}\| = 1$的超平面$\hat{\boldsymbol{w}}_{\text{opt}} \cdot \hat{\boldsymbol{x}} = \boldsymbol{w}_{\text{opt}} \cdot \boldsymbol{x} + b_{\text{opt}}$将训练数据集完全正确分开；且存在$\gamma > 0$，对所有$i = 1,2 \cdots, N$，有

$$y_i(\hat{\boldsymbol{w}}_{\text{opt}} \cdot \hat{\boldsymbol{x}}) = y_i(\boldsymbol{w}_{\text{opt}} \cdot \boldsymbol{x} + b_{\text{opt}}) \geq \gamma$$

(2)令$R = \max\limits_{1 \leq i \leq N} \|\hat{\boldsymbol{x}}_i\|$，则感知机算法5.1在训练数据集上的误分类次数$k$满足不等式

$$k \leq \left(\frac{R}{\gamma}\right)^2$$

感知机算法的收敛性

证明

(1)由于训练数据集是线性可分的，存在超平面可将训练数据集完全正确分开，取此超平面为$\hat{\boldsymbol{w}} \cdot \hat{\boldsymbol{x}} = \boldsymbol{w} \cdot \boldsymbol{x} + b = 0$，使$\left\|\hat{\boldsymbol{w}}_{\mathrm{opt}}\right\| = 1$。由于对有限的$i = 1,2 \cdots, N$，均有

$$y_i\left(\hat{\boldsymbol{w}}_{\mathrm{opt}} \cdot \hat{\boldsymbol{x}}\right) = y_i(\boldsymbol{w}_{\mathrm{opt}} \cdot \boldsymbol{x} + b_{\mathrm{opt}}) > 0$$

所以存在

$$\gamma = \min_i\{y_i(\boldsymbol{w}_{\mathrm{opt}} \cdot \boldsymbol{x} + b_{\mathrm{opt}})\}$$

使

$$y_i\left(\hat{\boldsymbol{w}}_{\mathrm{opt}} \cdot \hat{\boldsymbol{x}}\right) = y_i(\boldsymbol{w}_{\mathrm{opt}} \cdot \boldsymbol{x} + b_{\mathrm{opt}}) \geq \gamma$$

感知机算法的收敛性

感知器算法从$\hat{w}_0 = 0$开始，如果实例被误分类，则更新权重。

令$\hat{w}_{k-1} = (w_{k-1}^T, b_{k-1})^T$

则第k个误分类实例的条件是 $y_i(\hat{w}_{k-1} \cdot \hat{x}_i) = y_i(w_{k-1} \cdot x + b_{k-1}) \leq 0$

若(x_i, y_i)是被$\hat{w}_{k-1} = (w_{k-1}^T, b_{k-1})^T$误分类的数据，则$(w, b)$的更新是

$$w_k \leftarrow w_{k-1} + \eta y_i x_i \qquad b_k \leftarrow b_{k-1} + \eta y_i$$

即 $\hat{w}_k = \hat{w}_{k-1} + \eta y_i \hat{x}_i$

$$\begin{aligned} \hat{w}_k \cdot \hat{w}_{opt} &= \hat{w}_{k-1} \cdot \hat{w}_{opt} + \eta y_i \hat{w}_{opt} \cdot \hat{x}_i \\ &\geq \hat{w}_{k-1} \cdot \hat{w}_{opt} + \eta\gamma \\ &\geq \hat{w}_{k-2} \cdot \hat{w}_{opt} + 2\eta\gamma \\ &\cdots \\ &\geq k\eta\gamma \end{aligned}$$

$$\begin{aligned} \|\hat{w}_k\|^2 &= \|\hat{w}_{k-1}\|^2 + 2\eta y_i \hat{w}_{k-1} \cdot \hat{x}_i + \eta^2 \|\hat{x}_i\|^2 \\ &\leq \|\hat{w}_{k-1}\|^2 + \eta^2 \|\hat{x}_i\|^2 \\ &\leq \|\hat{w}_{k-1}\|^2 + \eta^2 R^2 \\ &\leq \|\hat{w}_{k-2}\|^2 + 2\eta^2 R^2 \\ &\cdots \\ &\leq k\eta^2 R^2 \end{aligned}$$

$k^2\eta^2\gamma^2 \leq k\eta^2 R^2$ 得 $k \leq (\frac{R}{\gamma})^2$

结论：误分类的次数k是有上界的，经过有限次搜索可以找到将训练数据完全正确分开的分离超平面。即当训练数据集线性可分时，感知机学习算法原始形式迭代是收敛的。

感知机学习算法的对偶形式

基本想法：将$\boldsymbol{w}$和b表示为实例$\boldsymbol{x}_i$和标记y_i的线性组合的形式，通过求解其系数而求得$\boldsymbol{w}$和b. 不失一般性，在算法5.1中可假设初始值$\boldsymbol{w}$和b均为零。对误分类点$(\boldsymbol{x}_i, y_i)$通过

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y_i \boldsymbol{x}_i$$

$$b \leftarrow b + \eta y_i$$

逐步修改$\boldsymbol{w}$和b，则$\boldsymbol{w}$和b关于$(\boldsymbol{x}_i, y_i)$的增量分别是$\alpha_i y_i \boldsymbol{x}_i$和$\alpha_i y_i$。这里，$\alpha_i = n_i \eta$，$n_i$是$(x_i, y_i)$被误分的次数。

最后学习到的$\boldsymbol{w}$和b可以分别表示为

$$\boldsymbol{w} = \sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x}_i \qquad b = \sum_{i=1}^{N} \alpha_i y_i$$

$$\alpha_i \geq 0, i = 1,2 \cdots, N$$

算法5.2 感知机学习算法的对偶形式

输入：

- 训练集$S=(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_N,y_N)$
- 学习率$\eta(0<\eta\leq 1)$

输出：$\boldsymbol{\alpha}$和b；感知机模型$f(\boldsymbol{x})=\text{sign}(\sum_{i=1}^{N}\alpha_i y_i \boldsymbol{x}_i\cdot\boldsymbol{x}+b)$，其中$\boldsymbol{\alpha}=(\alpha_1,\alpha_2,\cdots,\alpha_N)^{\mathrm{T}}$

过程：

（1）$\boldsymbol{\alpha}\leftarrow\boldsymbol{0},b\leftarrow 0$

（2）在训练集中选取数据$(\boldsymbol{x}_i,y_i)$；

（3）if $y_i(\sum_{i=1}^{N}\alpha_i y_i \boldsymbol{x}_i\cdot\boldsymbol{x}+b)\leq 0$ then

$$\alpha_i\leftarrow\alpha_i+\eta$$

$$b\leftarrow b+\eta y_i$$

（4）转到步骤（2）直到没有误分类数据。

Gram矩阵：$G=[\boldsymbol{x}_i\cdot\boldsymbol{x}_j]_{N\times N}$

例5.3

数据同例5.2，正样本是$\boldsymbol{x}_1=(3,3)^{\mathrm{T}},\boldsymbol{x}_2=(4,3)^{\mathrm{T}}$，负样本是$\boldsymbol{x}_3=(1,1)^{T}$，试用感知机学习算法对偶形式求感知机模型。

解 按照算法5.2

(1) 取$\alpha_i=0,i=1,2,3,b=0,\eta$=1;

(2) 计算Gram矩阵; $G=\begin{bmatrix}18 & 21 & 6\\ 21 & 25 & 7\\ 6 & 7 & 2\end{bmatrix}$

(3) 误分条件 $y_i\left(\sum_{i=1}^{N}\alpha_i y_i\boldsymbol{x}_i\cdot\boldsymbol{x}+b\right)\le 0$

参数更新： $\alpha_i\leftarrow\alpha_i+1 \quad b\leftarrow b+y_i$

(4) 迭代，过程参见表5.2。

(5) $\boldsymbol{w}=2\boldsymbol{x}_1+0\boldsymbol{x}_2-5\boldsymbol{x}_3=(1,1)^T \quad b=-3$

分离超平面： $x^{(1)}+x^{(2)}-3=0$

感知机模型： $f(\boldsymbol{x})=sign(x^{(1)}+x^{(2)}-3)$

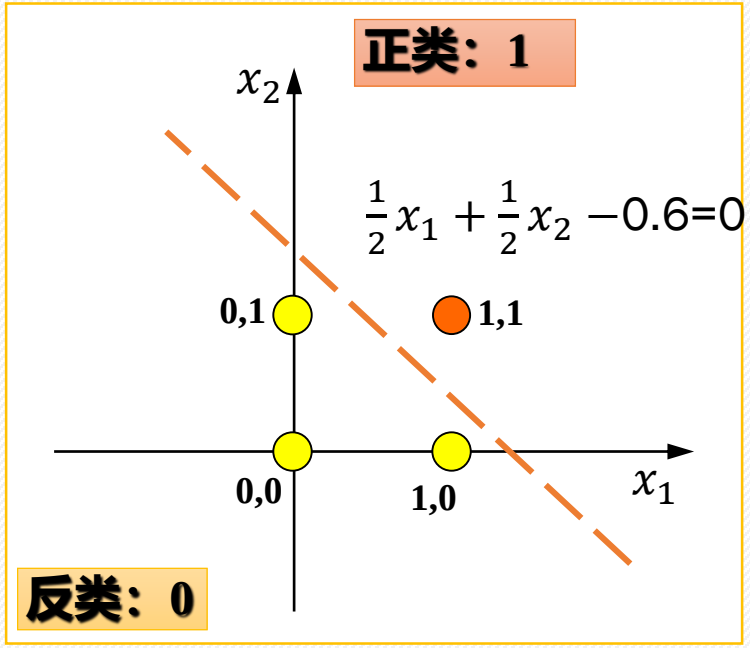
表5.2 例5.3求解的迭代过程

k	0	1	2	3	4	5	6	7
		x_1	x_3	x_3	x_3	x_1	x_3	x_3
α_1	0	1	1	1	1	2	2	2
α_2	0	0	0	0	0	0	0	0
α_3	0	0	1	2	3	3	4	5
b	0	1	0	-1	-2	-1	-2	-3

用感知器实现简单逻辑运算

(1) 逻辑“与” $(x_1 \text{ and } x_2)$

x_1	x_2	x_1 and x_2
0	0	0
0	1	0
1	0	0
1	1	1

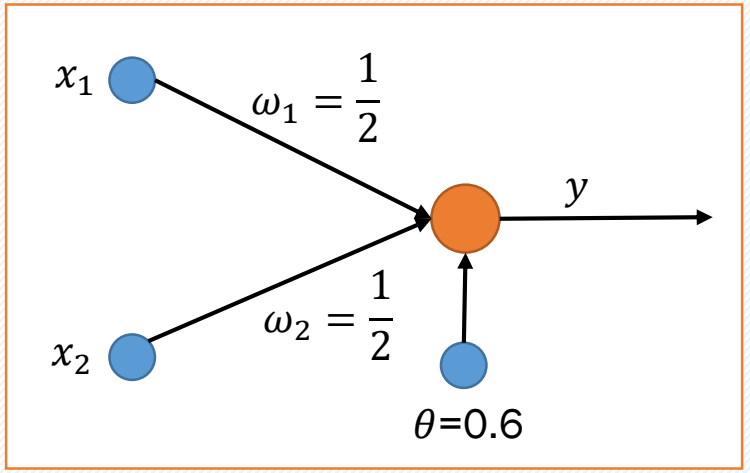


感知器模型

$$y=f(\omega_1 x_1+\omega_2 x_2+b)$$

分界线

$$\frac{1}{2}x_1+\frac{1}{2}x_2-0.6=0$$



用感知器实现简单逻辑运算

(2) 逻辑“或”(x_1 or x_2)

- 令$\omega_1 = \omega_2 = 1, b = -0.5$，则$y = f(1 \times x_1 + 1 \times x_2 - 0.5)$
- 只要x_1和x_2中有一个为1，则y的值为1；
- 只有当x_1和x_2同时为0时，y的值才为0。

(3) 逻辑“非”(not x_1)

- 令$\omega_1 = -1$，$\omega_2 = 0$，$b = 0.5$，则$y = f((-1) \times x_1 + 0 \times x_2 + 0.5))$
- 无论x_2为何值，x_1为1时，y的值都为0；
  x_1为0时，y的值为1。即y总等于- x_1。

(4) 逻辑“异或”(x_1 xor x_2)

- 单层感知器无法解决异或问题。

$$y(x_1, x_2) = \begin{cases} 0, & x_1 = x_2 \\ 1, & x_1 \neq x_2 \end{cases}$$

感知器模型

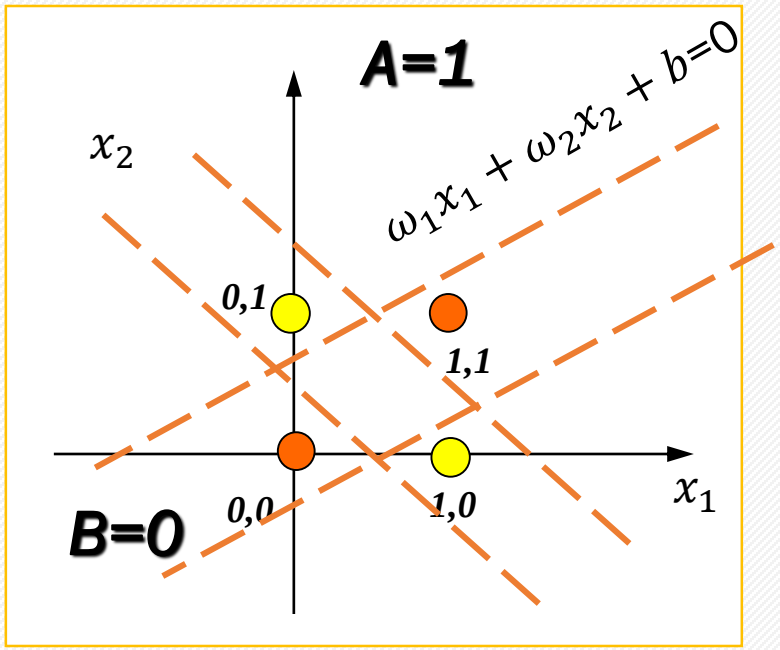
逻辑“异或”

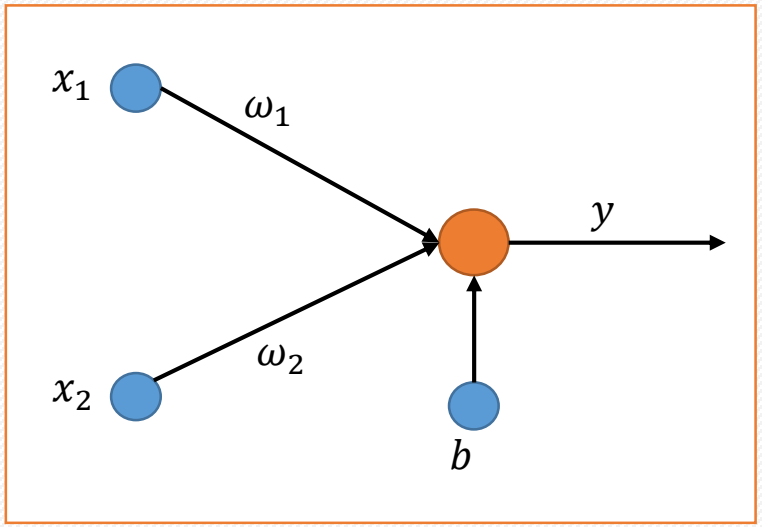
x_1	x_2	x_1 xor x_2
1	1	0
1	0	1
0	1	1
0	0	0

显然，无法找到一条线，将样本分为期望的两类。

ω=?

b=?





感知器模型

- 对于某个感知机
  - 输入为x_1，x_2，权值为 在ω_1，ω_2，偏置是b
- 为了学习这个函数，神经网络必须找到一组值，满足不等式方程：

$$\omega_1 * 1 + \omega_2 * 1 + b < 0, \quad \text{真值表的第一行；}$$

$$\omega_1 * 1 + \omega_2 * 0 + b > 0, \quad \text{真值表的第二行；}$$

$$\omega_1 * 0 + \omega_2 * 1 + b > 0, \quad \text{真值表的第三行；}$$

$$\omega_1 * 0 + \omega_2 * 0 + b < 0, \quad \text{最后一行。}$$

- 这组关于ω_1，ω_2和b的不等式方程组没有解，证明感知机不能解决异或问题。
- 异或问题不能用感知机来解决的原因：待识别的类别无法线性可分。

Marvin Lee Minsky（马文·明斯基）

1969年，他和佩波特的名著《感知机》证明了Rosenblatt感知机解决非线性问题能力薄弱，连XOR逻辑分类都做不到，只能作线性划分。更重要的是，他将关于单层感知器局限性的结论推广到了多层感知器。

原文如下：

尽管它有严重的局限性，感知器展示了它自身的研究价值。它有很多吸引人的优点：它的线性，迷人的学习法则，作为一类并行计算范例的清晰的简单性。没有任何理由认为这些优点能带到多层感知器中去。依据我们的直觉判断，即使推广到多层系统也不会有好的结果。但是，我们认为证明（或否定）这一点是一个很重要的需要研究的问题。

雨后彩虹

神经网络研究进入低谷期，一直持续了将近20年。

神经网络的反击：

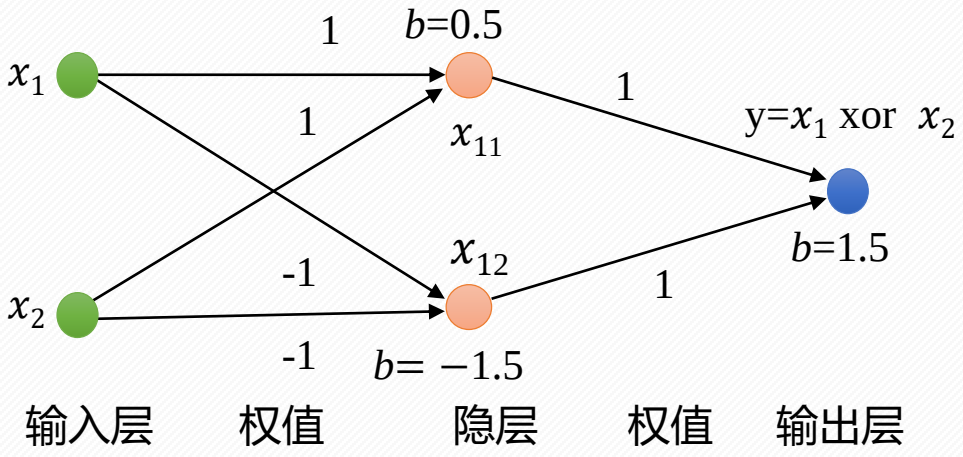
- 1988年，赛本柯(Cybenko)等证明：具有两个隐层的BP网络可以实现任何有界连续函数。
- 1989年又证明：单隐层BP网络可以实现任何有界连续函数。

明斯基对多层感知器的“判决”被彻底推翻。

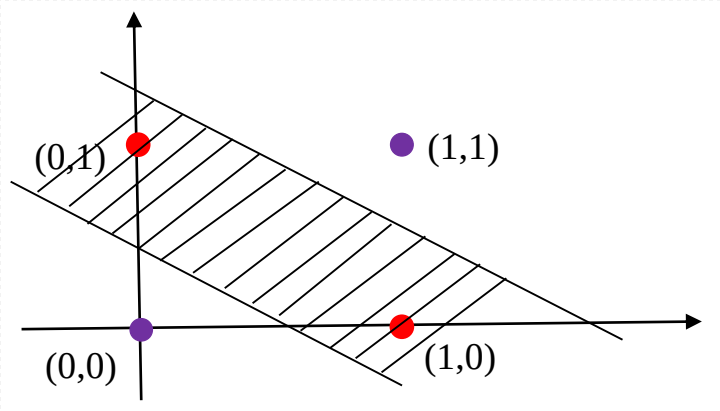
说明两点：

1. 不要随便预言科学的未来。好像从来就没准过。
2. 不要太迷信大师。

逻辑“异或”问题的多层感知器



两层感知器结构



分类区域

$y = x_1 \text{ XOR } x_2 = (x_1 \text{ or } x_2) \text{ AND } (- x_1 \text{ or } -x_2)$

- 隐层神经元x_{11}所确定的直线方程

$$1 \times x_1 + 1 \times x_2 - 0.5$$

- 隐层神经元x_{12}所确定的直线方程

$$-1 \times x_1 - 1 \times x_2 + 1.5$$

- 输出层神经元所确定的直线方程

$$1 \times x_{11} + 1 \times x_{12} - 1.5$$

相当于对隐层神经元x_{11}和x_{12}的输出作“逻辑与”运算，因此可识别由隐层已识别的两个半平面的交集所构成的一个凸多边形。

感知器模型小结

分类特性

感知器将输入信号分为两类：输出-1或1；

感知器的分界线(面)： $\boldsymbol{\omega}^{\boldsymbol{T}}\boldsymbol{x}+b=0$；

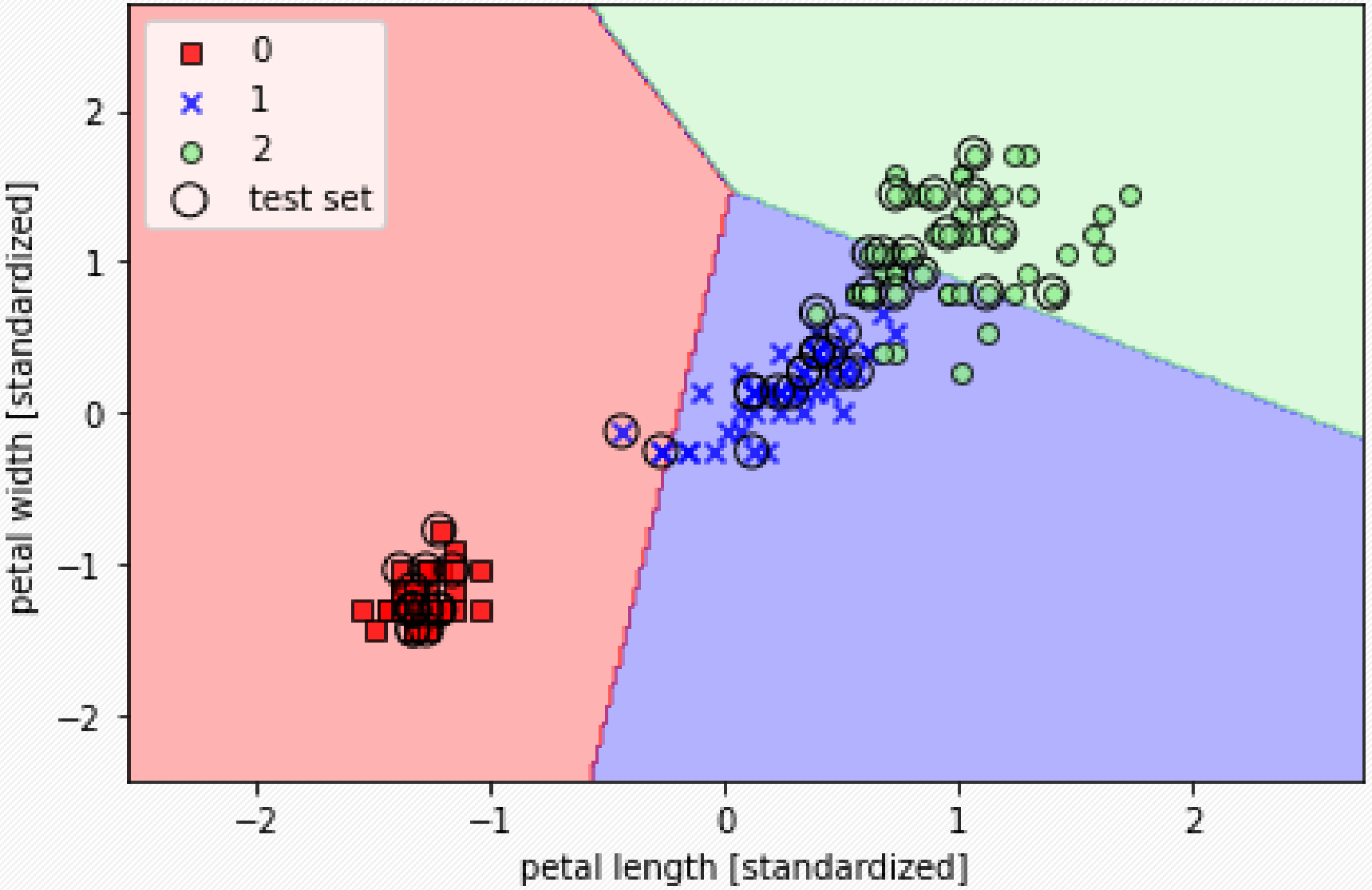
分类能力

单层感知器的分类能力有限；

只能对线性可分的样本进行分类；

对于非线性样本，则无法进行正确划分。

例5.2，感知机实现鸢尾花分类



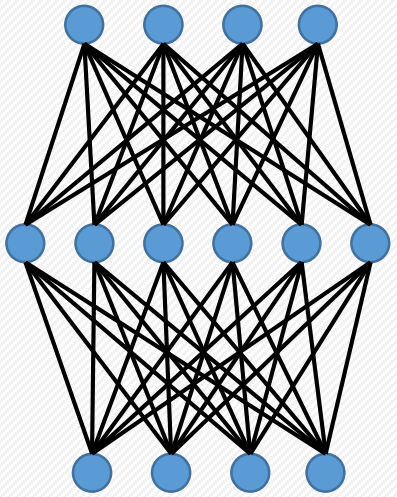
多层前馈网络结构

多层网络：包含隐层的网络

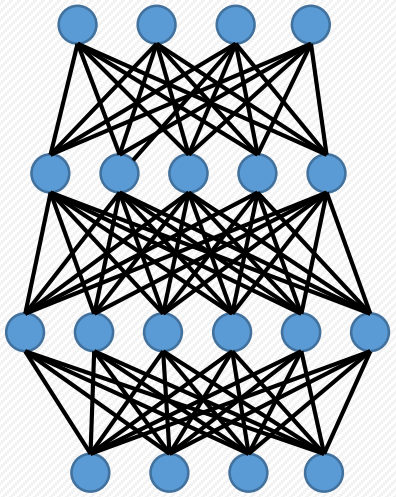
前馈网络：神经元之间不存在同层连接也不存在跨层连接

隐层和输出层神经元亦称“功能单元”(functional unit)

多层前馈网络有强大的表示能力



单隐层前馈网络



双隐层前馈网络

只需一个包含足够多神经元的隐层，多层前馈神经网络就能以任意精度逼近任意复杂度的连续函数 [Hornik et al., 1989]

但是，如何设置隐层神经元数是未决问题. 常用“试错法”

5.3 误差逆传播算法 (BP)

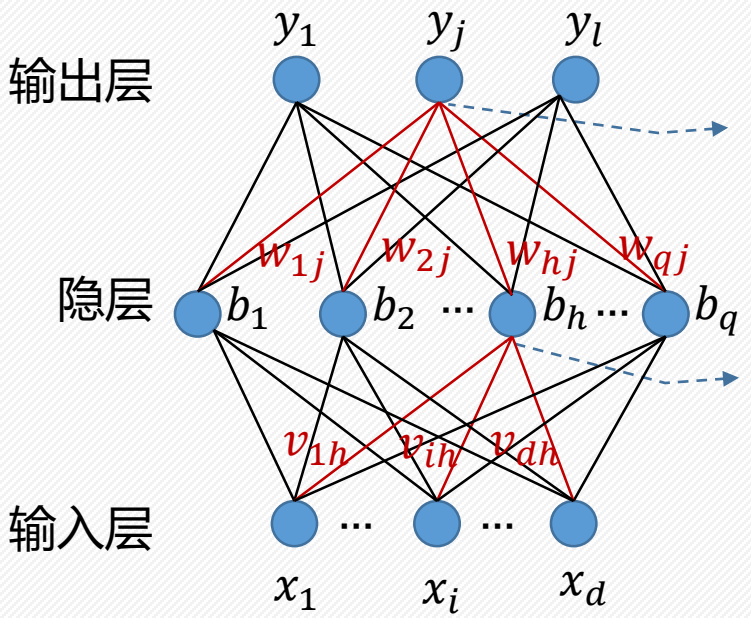
最成功、最常用的神经网络算法，被用于多种任务（不仅限于分类）

P. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral science. Ph.D dissertation, Harvard University, 1974

给定训练集 $D = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), \ldots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\}, \boldsymbol{x}_i \in \mathbb{R}^d, \boldsymbol{y}_i \in \mathbb{R}^l$

输入：d 维特征向量
输出：l个输出值
隐层：使用q个隐层神经元

输出层第j个神经元的阈值为θ_j，隐层第h个神经元的阈值为γ_h，输入第i个神经元与隐层第h个神经元之间的连接权为v_{ih}，隐层第h个神经元与输出层第j个神经元之间的连接权为w_{hj}，隐层和输出层均使用Sigmoid 激活函数。



第j个输出神经元的输入

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h$$

第h个隐层神经元的输入

$$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$$

BP 算法推导

对于训练样例$(\boldsymbol{x}_k, \boldsymbol{y}_k)$，假定网络的实际输出为$\hat{\boldsymbol{y}}_k = (\hat{y}_1^k, \hat{y}_2^k \ldots, \hat{y}_l^k)$

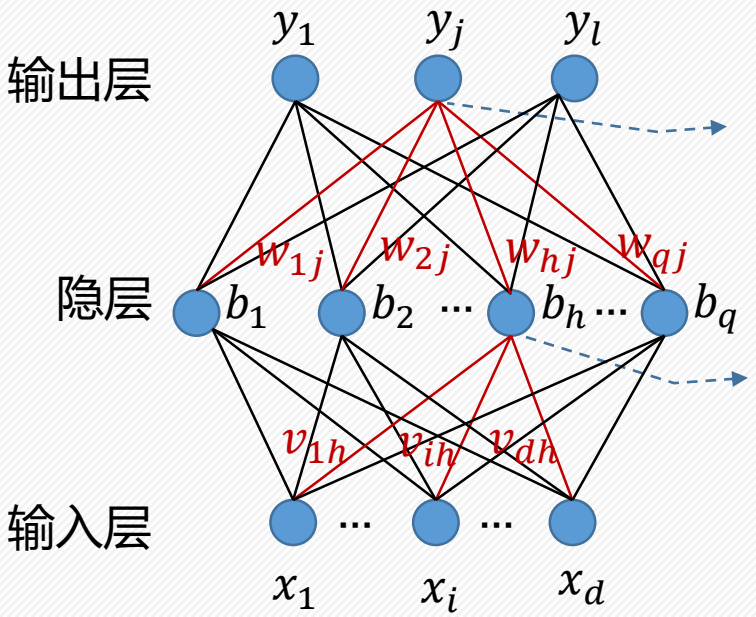
$$\hat{y}_j^k = f(\beta_j - \theta_j)$$

则网络在$(\boldsymbol{x}_k, \boldsymbol{y}_k)$上的均方误差为：

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$$

需通过学习确定的参数数目：

$$(d + l + 1)q + l$$



第j个输出神经元的输入

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h$$

第h个隐层神经元的输入

$$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$$

BP 是一个迭代学习算法，在迭代的每一轮中采用广义感知机学习规则

$$v \leftarrow v + \Delta v$$

BP 算法推导 (续)

BP 算法基于梯度下降策略，以目标的负梯度方向对参数进行调整

以ω_{hj}为例

对误差E_k，给定学习率η，有：

$$\Delta\omega_{hj}=-\eta\frac{\partial E_k}{\partial \omega_{hj}}$$

注意到ω_{hj}先影响到第j个神经元的输入β_j再影响到$\hat{y}_j^k$，然后才影响到E_k，有：

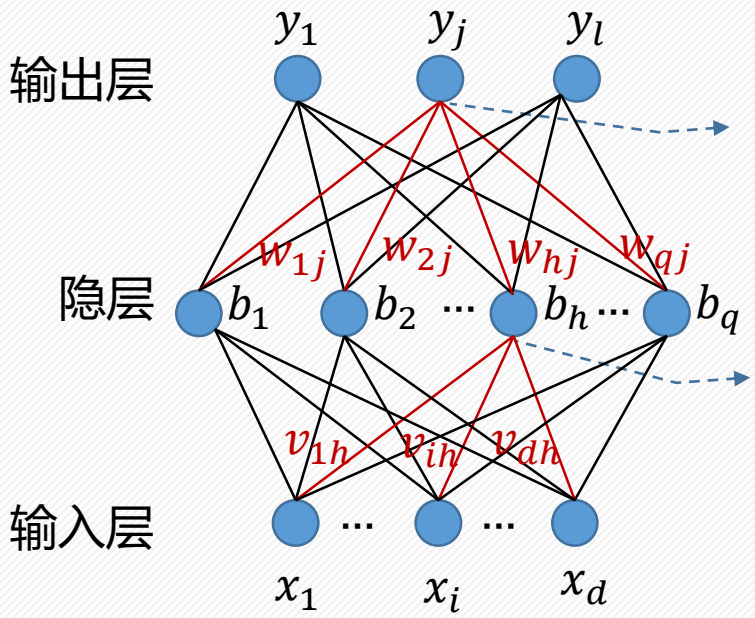
$$\frac{\partial E_k}{\partial \omega_{hj}}=\frac{\partial E_k}{\partial \hat{y}_j^k}\cdot\frac{\partial \hat{y}_j^k}{\partial \beta_j}\cdot\frac{\partial \beta_j}{\partial \omega_{hj}}$$

⇐ 链式法则

$$E_k=\frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k-y_j^k)^2$$

$$\hat{y}_j^k=f(\beta_j-\theta_j)$$

$$\beta_j=\sum_{h=1}^{q}\omega_{hj}b_h$$



第j个输出神经元的输入

$$\beta_j=\sum_{h=1}^{q}w_{hj}b_h$$

第h个隐层神经元的输入

$$\alpha_h=\sum_{i=1}^{d}v_{ih}x_i$$

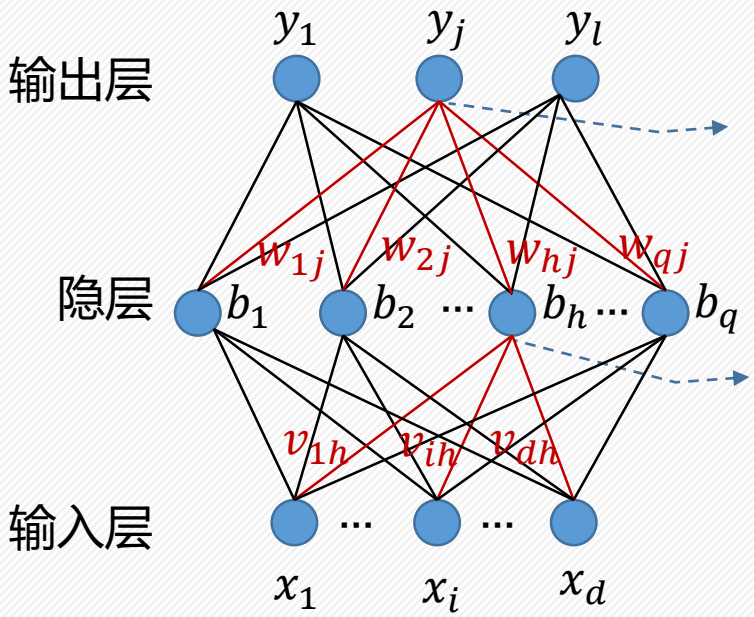
BP 算法推导 (续)

$$\frac{\partial E_k}{\partial \omega_{hj}} = \frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial \omega_{hj}}$$

$$\frac{\partial \beta_j}{\partial \omega_{hj}} = b_h$$

$$\begin{aligned} g_j &= -\frac{\partial E_k}{\partial \hat{y}_j^k} \cdot \frac{\partial \hat{y}_j^k}{\partial \beta_j} \\ &= -(\hat{y}_j^k - y_j^k) f'(\beta_j - \theta_j) \\ &= \hat{y}_j^k (1 - \hat{y}_j^k)(y_j^k - \hat{y}_j^k) \end{aligned}$$

$$\Delta \omega_{hj} = -\eta \frac{\partial E_k}{\partial \omega_{hj}} = \eta g_j b_h$$



第j个输出神经元的输入

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h$$

第h个隐层神经元的输入

$$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$$

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

$$f'(x) = f(x)(1 - f(x))$$

BP 算法推导 (续)

类似地，有：

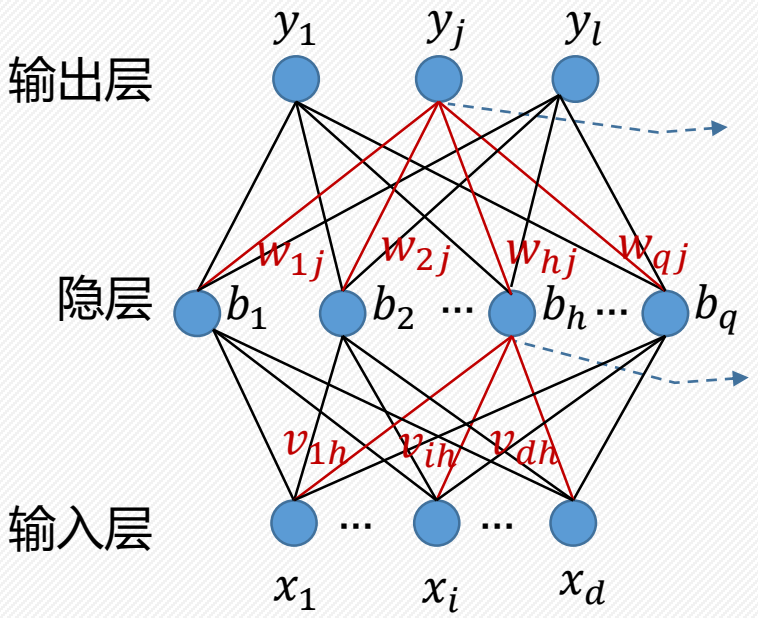
$$\Delta\theta_j = -\eta g_j$$

$$\Delta v_{ih} = \eta e_h x_i$$

$$\Delta\gamma_h = -\eta e_h$$

其中：

$$\begin{aligned} e_h &= -\frac{\partial E_k}{\partial b_h}\cdot\frac{\partial b_h}{\partial \alpha_h} \\ &= -\sum_{j=1}^{l}\frac{\partial E_k}{\partial \beta_j}\cdot\frac{\partial \beta_j}{\partial b_h} f'(\alpha_h-\gamma_h) \\ &= \sum_{j=1}^{l}\omega_{hj} g_j f'(\alpha_h-\gamma_h) \\ &= b_h(1-b_h)\sum_{j=1}^{l}\omega_{hj} g_j \end{aligned}$$



第j个输出神经元的输入

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h$$

第h个隐层神经元的输入

$$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$$

学习率η不能太大、不能太小。

BP 算法

输入：训练集 $D=\{(x_k,y_k)\}_k^m$；
学习率η.

过程：

1:在(0, 1)范围内随机初始化网络中所有连接权重和阈值
2:repeat
3: for all $(x_k,y_k)\in D$ do .
4: 根据当前参数和式(5.3)计算当前样本的输出$\hat{y}_k$;
5: 根据式(5.10)计算输出层神经元的梯度项g_j;
6: 根据式(5.15)计算隐层神经元的梯度项e_h;
7: 根据式(5.11)-(5.14)更新连接权ω_{hj}，v_{ih}与阈值θ_j,γ_h
8: end for
9: until 达到停止条件
输出：连接权与阈值确定的多层前馈神经网络

图5.8 误差逆传播算法

标准 BP 算法 vs. 累积 BP 算法

标准 BP 算法

- 每次针对单个训练样例更新权值与阈值
- 参数更新频繁，不同样例可能抵消，需要多次迭代

累积 BP 算法

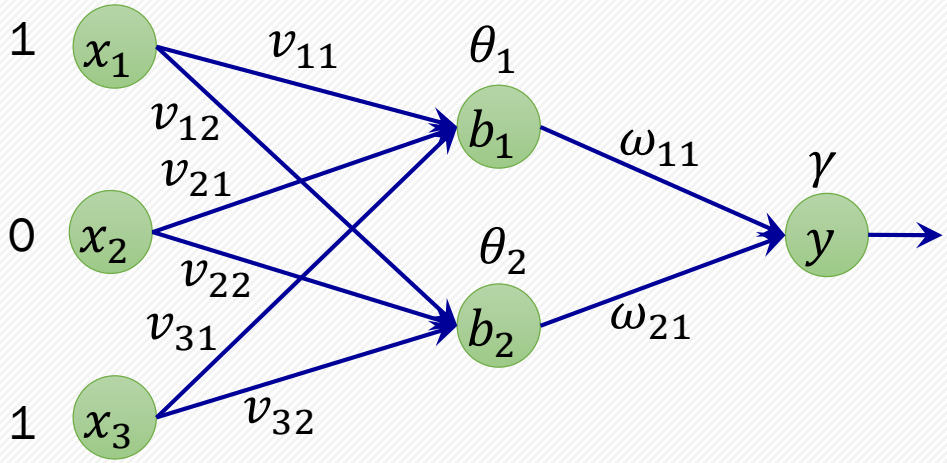
- 其优化目标是最小化整个训练集上的累计误差
- 读取整个训练集一遍才对参数进行更新，参数更新频率较低

在很多任务中，累计误差下降到一定程度后，进一步下降会非常缓慢，这时标准 BP算法往往会获得较好的解，尤其当训练集非常大时效果更明显。

例5.4，多层前馈神经网络示例。

设学习率η为0.1，该网络的初始权值和阈值如下表所示，样本为$\boldsymbol{x}=\{1,0,1\}$，期望输出为1。通过BP算法学习网络参数。

x_1	x_2	x_3	v_{11}	v_{12}	v_{21}	v_{22}	v_{31}	v_{32}	ω_{11}	ω_{21}	θ_1	θ_2	γ
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	0.4	-0.2	-0.1



例5.4(续)，多层前馈神经网络示例。

x_1	x_2	x_3	v_{11}	v_{12}	v_{21}	v_{22}	v_{31}	v_{32}	ω_{11}	ω_{21}	θ_1	θ_2	γ
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	0.4	-0.2	-0.1

(1) 向前计算：逐层计算节点输出

节点b_1：$\alpha_1 = 0.2 \times 1 + 0.4 \times 0 - 0.5 \times 1 - 0.4 = -0.7$

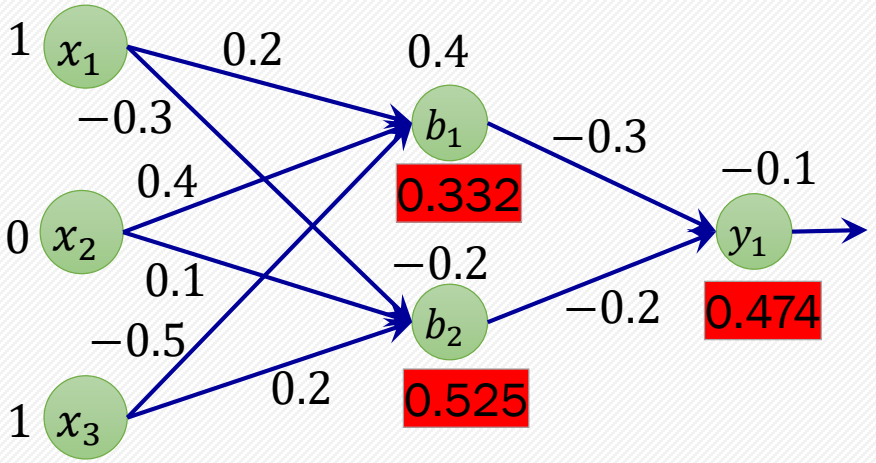
$b_1 = f(-0.7) = 0.332$

节点b_2：$\alpha_2 = -0.3 \times 1 + 0.1 \times 0 + 0.2 \times 1 + 0.2 = 0.1$

$b_2 = f(0.1) = 0.525$

节点y_1：$\beta_1 = -0.3 \times 0.332 - 0.2 \times 0.525 + 0.1 = -0.1046$

$y_1 = f(-0.1046) = 0.474$



例5.4(续)，多层前馈神经网络示例。

x_1	x_2	x_3	v_{11}	v_{12}	v_{21}	v_{22}	v_{31}	v_{32}	ω_{11}	ω_{21}	θ_1	θ_2	γ
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	0.4	-0.2	-0.1

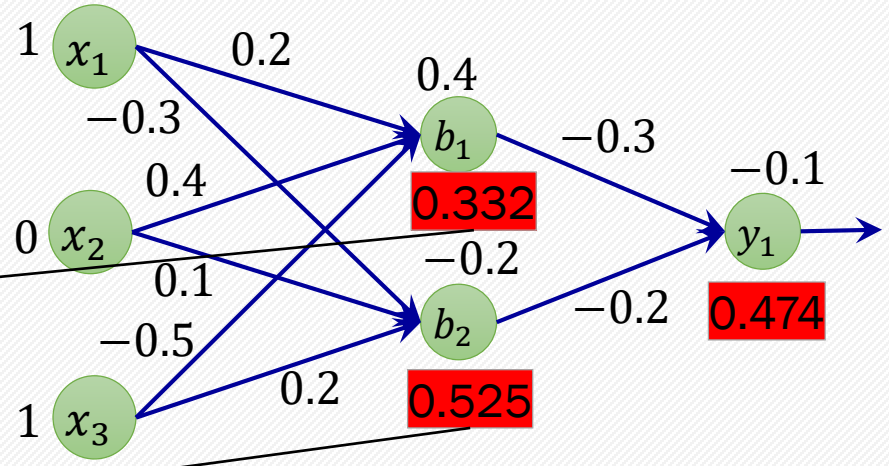
(2) 向后计算：逐层计算误差

$\hat{y}_1^1$　y_j^k

节点y_1梯度：$g_1 = 0.474 \times (1-0.474) \times (1-0.474)$
$=0.1311$

节点b_1梯度：$e_1 = 0.332 \times (1-0.332) \times (-0.3) \times 0.1311$
$=-0.0087$

节点b_2梯度：$e_2 = 0.525 \times (1-0.525) \times (-0.2) \times 0.1311$
$=-0.0065$



$$g_1 = \hat{y}_j^k(1-\hat{y}_j^k)(y_j^k-\hat{y}_j^k)$$

$$e_1 = b_h(1-b_h)\sum_{j=1}^{l}\omega_{hj}g_j$$

例5.4(续)，多层前馈神经网络示例。

x_1	x_2	x_3	v_{11}	v_{12}	v_{21}	v_{22}	v_{31}	v_{32}	ω_{11}	ω_{21}	θ_1	θ_2	γ
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	0.4	-0.2	-0.1

(2) 向后计算：修正权值、阈值

节点y_1权值、阈值：

η　g_1　b_1

$$\omega_{11} = -0.3 + 0.1 \times 0.1311 \times 0.332 = -0.296$$

$$\omega_{21} = -0.2 + 0.1 \times 0.1311 \times 0.525 = -0.193$$

$$\gamma = -0.1 - 0.1 \times 0.1311 = 0.11311$$

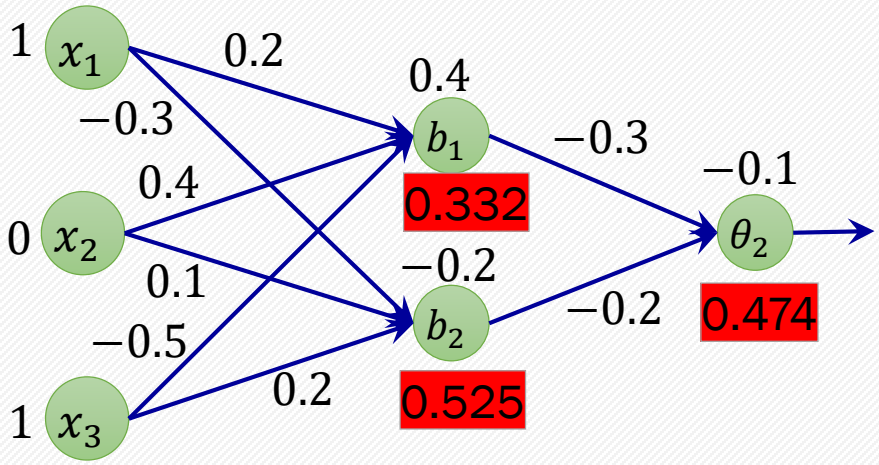
节点b_1权值、阈值：

$$v_{11} = 0.2 + 0.1 \times (-0.0087) \times 1 = 0.199$$

$$v_{21} = 0.4 + 0.1 \times (-0.0087) \times 0 = 0.4$$

$$v_{31} = -0.5 + 0.1 \times (-0.0087) \times 1 = -0.5009$$

$$\theta_1 = 0.4 - 0.1 \times (-0.0087) = 0.4009$$



$$\Delta\omega_{hj} = \eta g_j b_h$$

$$\Delta\theta_j = -\eta g_j$$

$$\Delta v_{ih} = \eta e_h x_i$$

$$\Delta\gamma_h = -\eta e_h$$

例5.4(续)，多层前馈神经网络示例。

x_1	x_2	x_3	v_{11}	v_{12}	v_{21}	v_{22}	v_{31}	v_{32}	ω_{11}	ω_{21}	θ_1	θ_2	γ
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	0.4	-0.2	-0.1

(2) 向后计算：修正权值、阈值

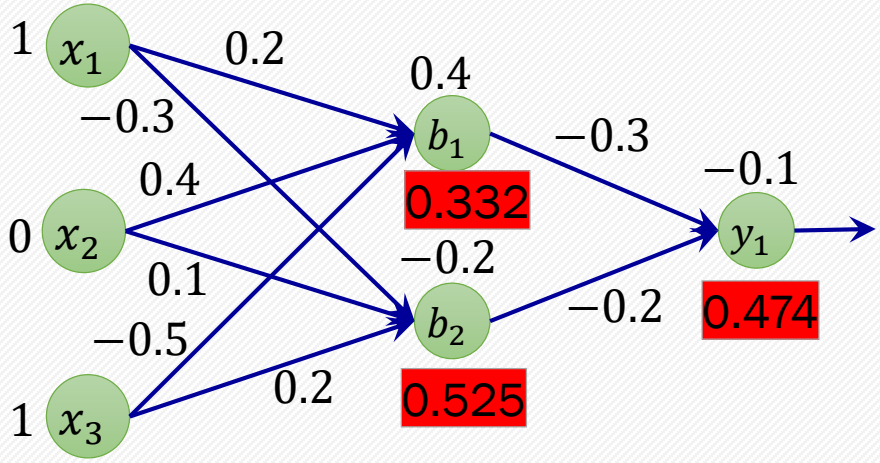
节点b_2权值、阈值：

$$v_{12} = -0.3 + 0.1 \times (-0.0065) \times 1 = 0.3007$$

$$v_{22} = 0.1 + 0.1 \times (-0.0065) \times 0 = 0.1$$

$$v_{32} = 0.2 + 0.1 \times (-0.0065) \times 1 = 0.1935$$

$$\theta_2 = -0.2 - 0.1 \times (-0.0065) = -0.1935$$



$$\Delta\omega_{hj} = \eta \frac{\partial E_k}{\partial \omega_{hj}} b_h$$

$$\Delta\theta_j = -\eta g_j$$

$$\Delta v_{ih} = \eta e_h x_i$$

$$\Delta\gamma_h = -\eta e_h$$

缓解过拟合

主要策略：

- **早停 (early stopping)**
  - 若训练误差连续 a 轮的变化小于 b, 则停止训练
  - 使用验证集：若训练误差降低、验证误差升高，则停止训练

- **正则化 (regularization)**
  - 在误差目标函数中增加一项描述网络复杂度

例如 $$E = \lambda \frac{1}{m} \sum_{i=1}^{m} E_k + (1-\lambda) \sum_{i} \omega_i^2$$

偏好比较小的连接权和阈值，使网络输出更“光滑”

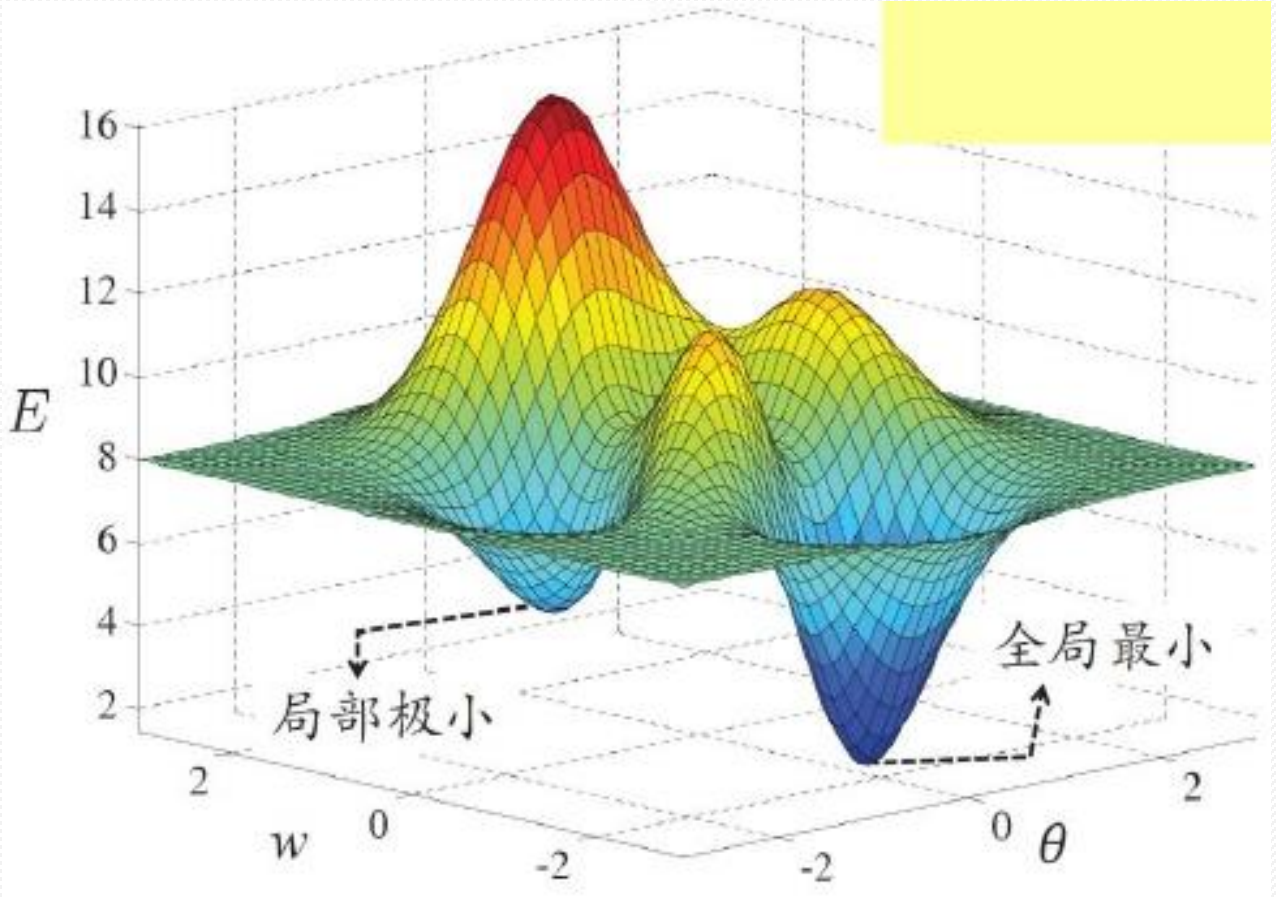
5.4 全局最小 与局部极小

神经网络的训练过程可看作参数寻优过程：在参数空间中，寻找一组最优参数使得误差最小

- 存在多个“局部极小”
- 只有一个“全局最小”

“跳出”局部极小的常见策略：

- 不同的初始参数
- 模拟退火
- 随机扰动
- 遗传算法
- ……



例5.4，【Pytorch实战】鸢尾花多分类

```
#网络类
class mynet(nn.Module):
    def __init__(self):
        super(mynet,self).__init__()
        self.fc=nn.Sequential( #添加层以及激活函数
            nn.Linear(4,20),
            nn.ReLU(),
            nn.Linear(20,30),
            nn.ReLU(),
            nn.Linear(30,3)
        )
        self.mse=nn.CrossEntropyLoss()
        self.optim=torch.optim.Adam(params=self.parameters(),lr=0.1)
```

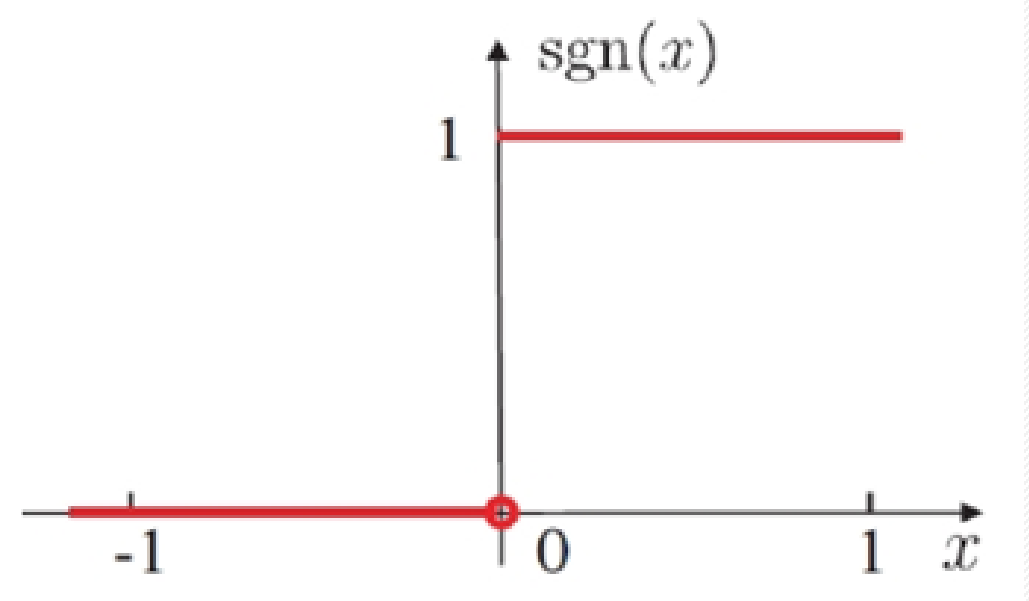
激活函数

- **目的**：为了增加神经网络模型的非线性表达能力。
- **基本要求**
  - 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应位于合适的区间，不能太大也不能太小，否则会影响训练的效率和稳定性。

激活函数

- **阶跃函数**（step function），0表示抑制，1表示激活。
- 具有不连续、不光滑等不好的性质 。

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



常见的激活函数

(1) Sigmoid函数

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

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

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

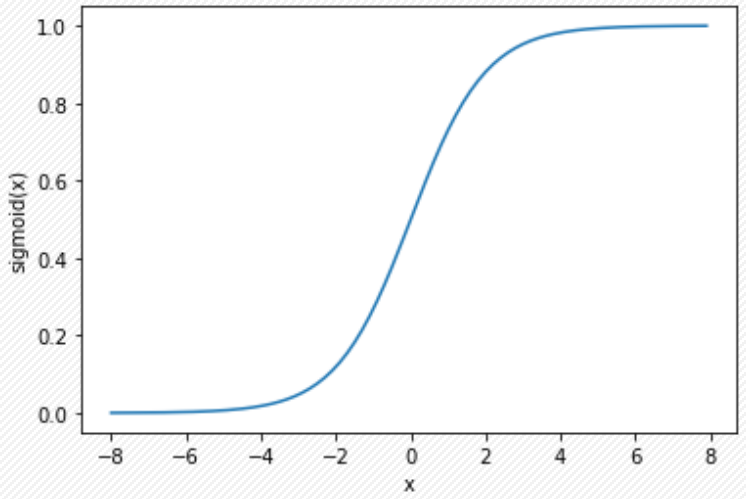
$$= s(x)(1-s(x))$$

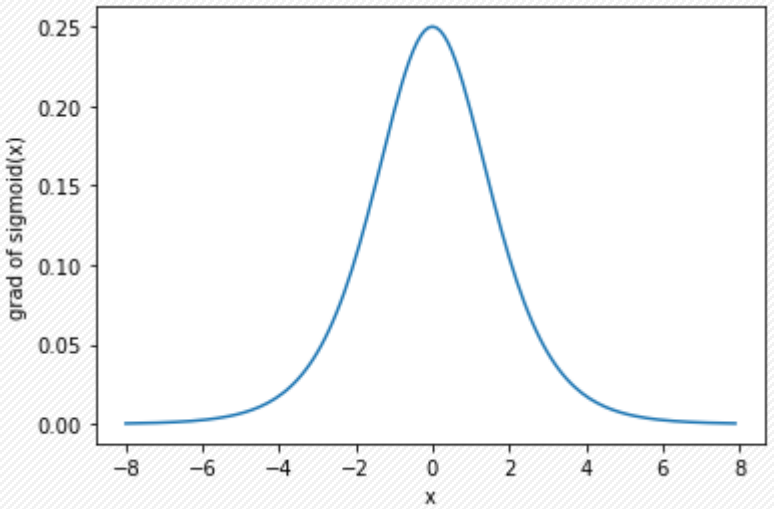
优点

- 输出范围为(0,1)，可以表示概率值
- 输出范围有限，数据在传递过程中不易发散
- 导数容易计算。

缺点

- saturate and kill gradients
- output 不以0为中心
- 计算复杂

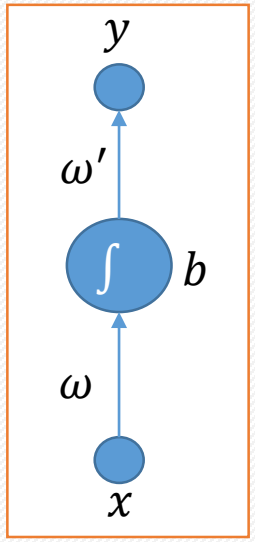





```
import torch
from torch import nn
input=torch.randn(1,1,2,2)
print(input)
s = nn.Sigmoid()
s(input)
```

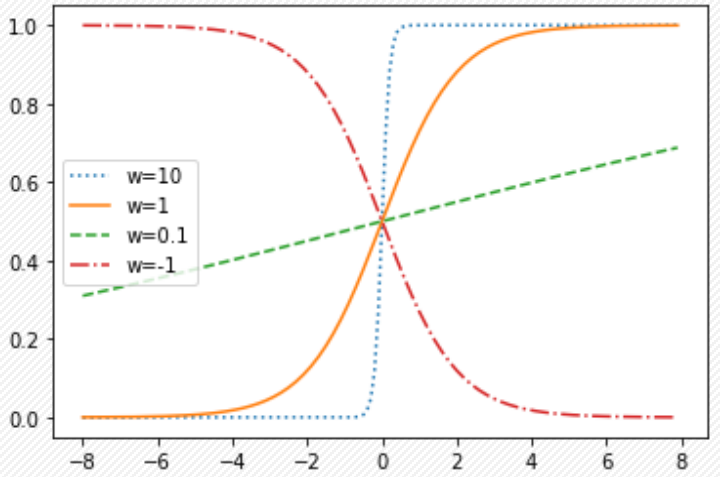
单神经元模型

$$y = s(x) = \omega'\sigma(\omega x + b)$$



保持$\omega' = 1$，$b = 0$

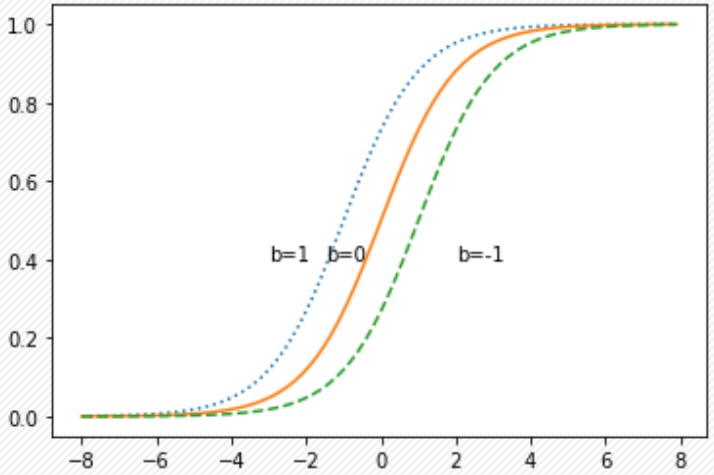
$$s(x) = \frac{1}{1 + e^{-\omega x}}$$



不同参数ω的sigmoid函数曲线

保持$\omega = \omega' = 1$

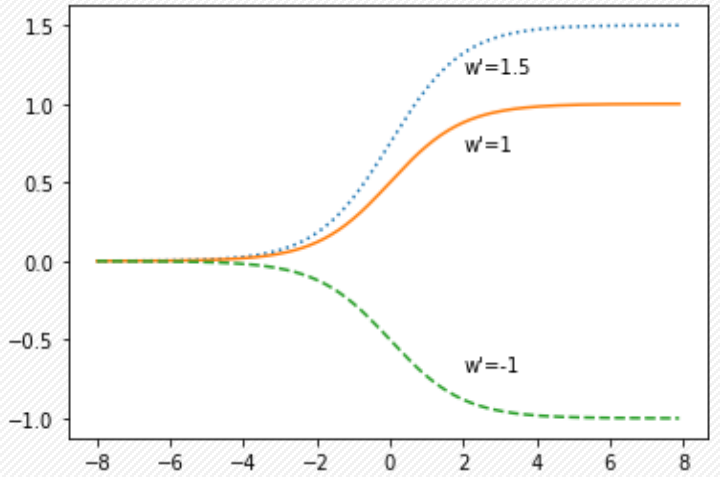
$$s(x) = \frac{1}{1 + e^{-(x+b)}}$$



不同参数b的sigmoid函数曲线

保持$\omega = 1$，$b = 0$

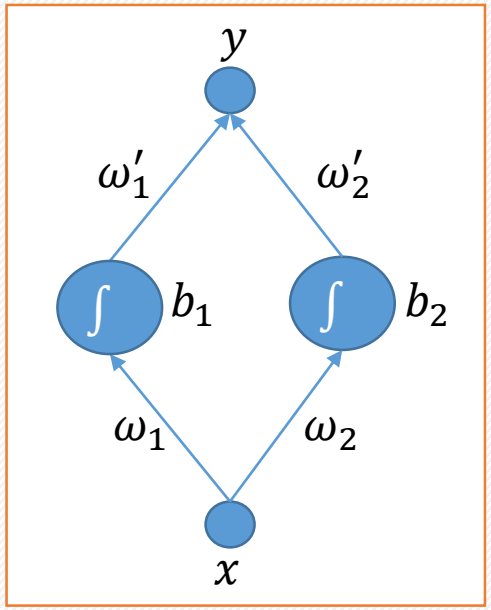
$$s(x) = \frac{\omega'}{1 + e^{-x}}$$



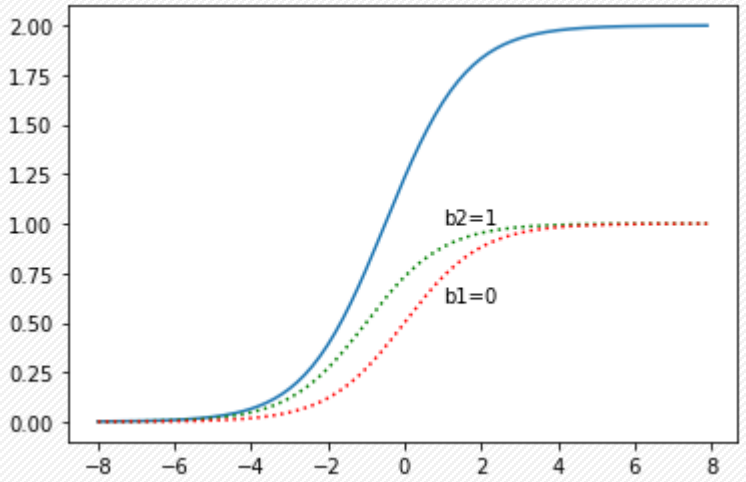
不同参数ω'下的sigmoid函数曲线

双神经元模型

$$y = s(x) = \omega_1'\sigma(\omega_1 x + b_1) + \omega_2'\sigma(\omega_2 x + b_2)$$

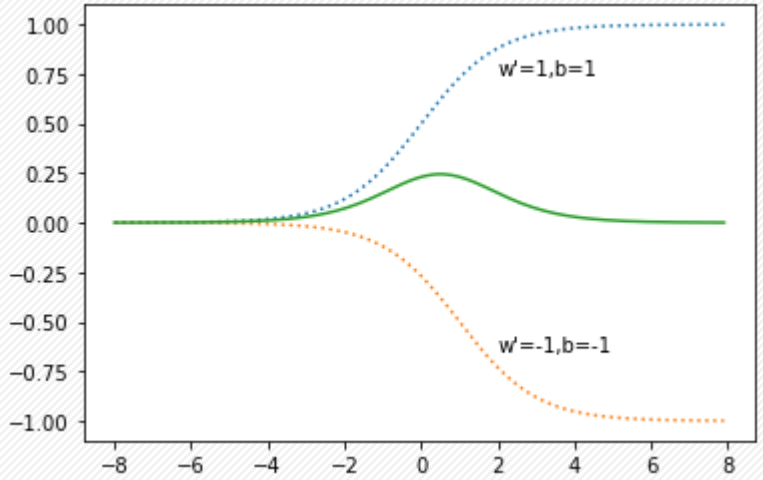


$\omega_1 = \omega_2 = \omega_1' = \omega_2'$=1, $b_1 = 1, b_2 = 0$



双神经元合成的阶梯曲线

$\omega_1 = \omega_2$ =1, $b_1 = 0, b_2 = -1, \omega_1' = 1, \omega_2'$=-1



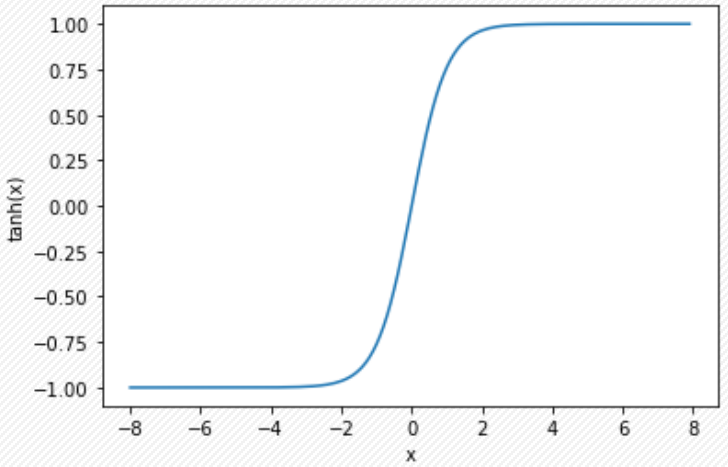
不同参数对双神经元曲线的影响

常见的激活函数

(2) Tanh函数

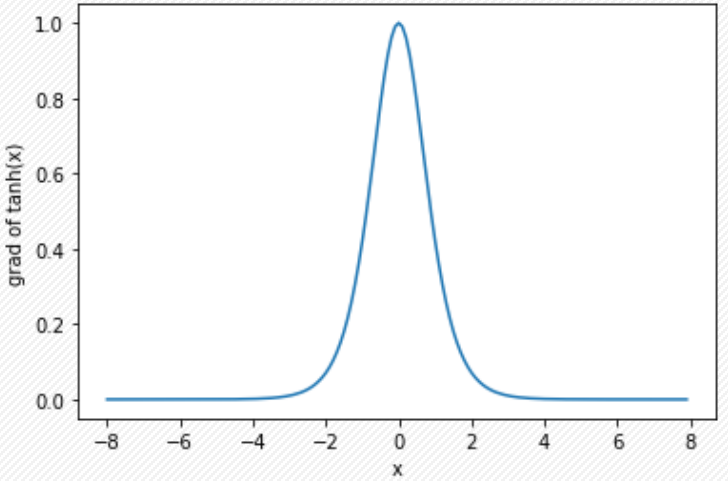
$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$

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



$$t(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{(e^x - e^{-x})e^{-x}}{(e^x + e^{-x})e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} = \frac{2 - (1 + e^{-2x})}{1 + e^{-2x}} = 2s(2x) - 1$$

$$\begin{aligned} t'(x) &= 2s(2x)\big(1 - s(2x)\big) * 2 \\ &= 1 - (4s(2x) * s(2x) - 4s(2x) + 1) \\ &= 1 - (2s(2x) - 1)^2 = 1 - t^2(x) \end{aligned}$$



$$t'(x) = \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2}$$

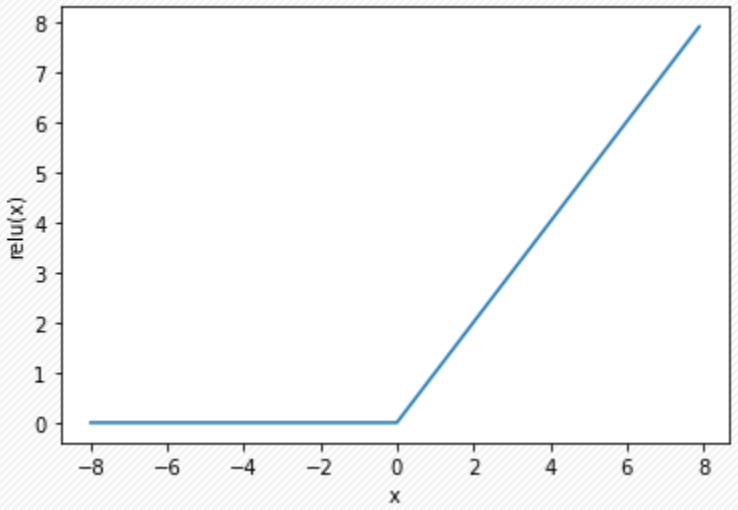
$$= 1 - t(x)^2$$

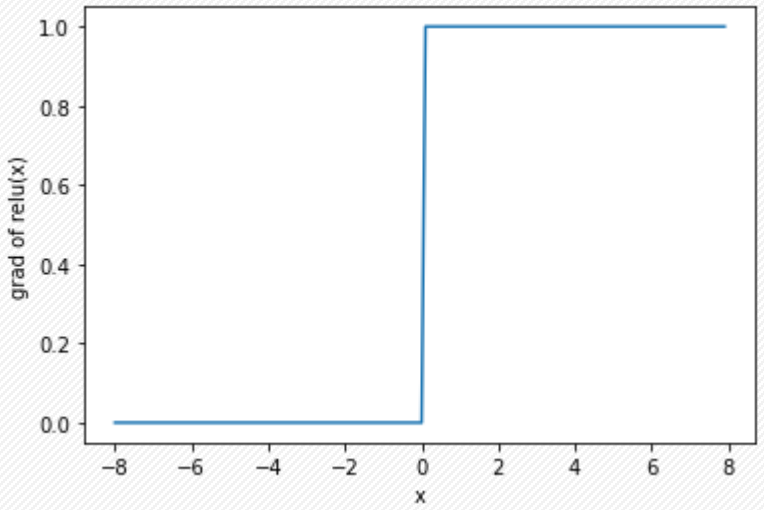
常见的激活函数

(3) Relu(Rectified Linear Unit)函数

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

$$f'(x)=\begin{cases}1 & x\geq 0\\ 0 & x<0\end{cases}$$





优点：

1) 当输入为正数的时候，不存在梯度饱和问题。
2) 计算高效、收敛速度快。

缺点：

1) 输入是负数的时候，完全不被激活，导致特征丢失。
2) 不是以0为中心。

```
m = nn.ReLU()
input = torch.randn(1,1,2,2)
output = m(input)
```

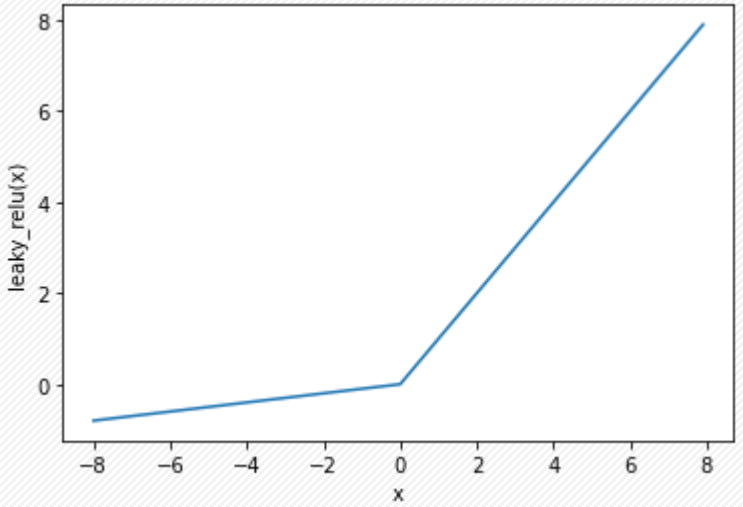
常见的激活函数

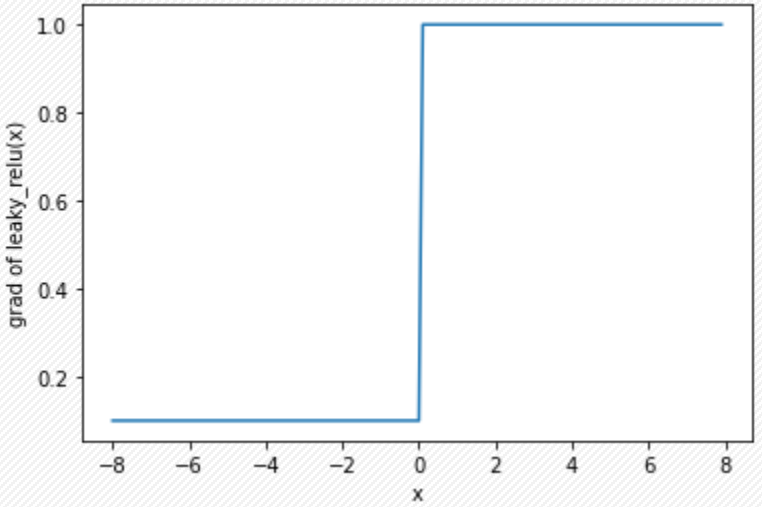
(4) LeakyRelu函数

$$f(x)=\begin{cases}x & x\geq 0\\ \gamma x & x<0\end{cases}$$
$$=\max(0,x)+\gamma\min(0,x)$$
$$=\max(x,\gamma x)$$

$$f'(x)=\begin{cases}1 & x\geq 0\\ \gamma & x<0\end{cases}$$

```
m = nn.LeakyReLU(0.03)
input = torch.randn(1,1,2,2)
output = m(input)
```





常见激活函数

(5) Prelu函数

$$\mathrm{PReLU}(x) = \max(0, x) + \alpha * \min(0, x)$$

$$\mathrm{PReLU}(x) = \begin{cases} x, & if\ x \geq 0 \\ \alpha x, & otherwise \end{cases}$$

Here α is a learnable parameter. When called without arguments, *nn.PReLU()* uses a single parameter α across all input channels. If called with *nn.PReLU(nChannels)*, a separate α is used for each input channel.

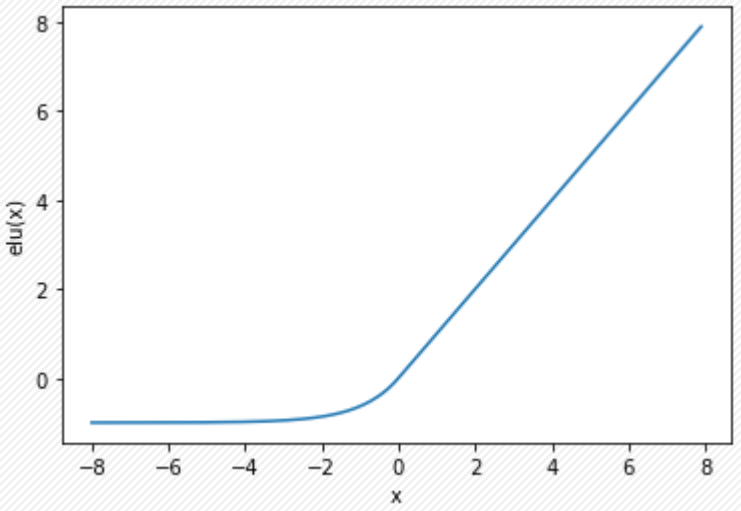
```
af = nn.PReLU()
input = torch.randn(1, 1, 2, 2)
output = af(input)
```

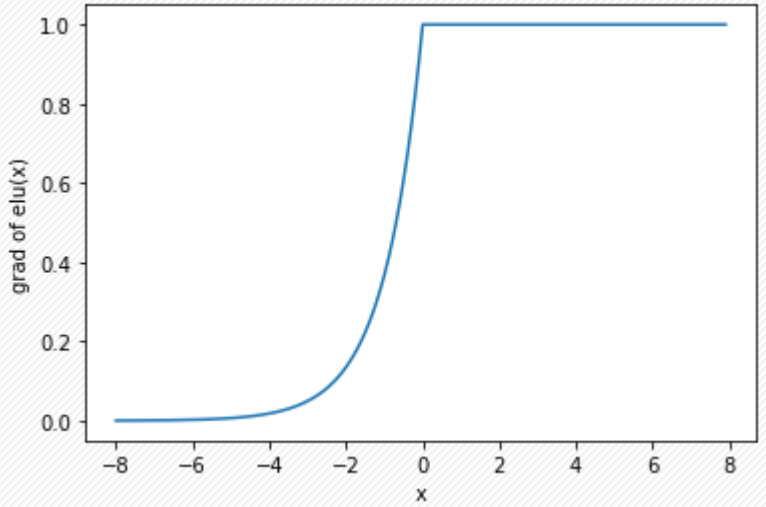
常见激活函数

(6) Elu函数

$$f(x)=\begin{cases}x & x\geq 0\\ \gamma(e^{x}-1) & x<0\end{cases}$$
$$=\max(0,x)+\min(0,e^{x}-1)$$

$$f'(x)=\begin{cases}1 & x\geq 0\\ f(x)+x & x<0\end{cases}$$

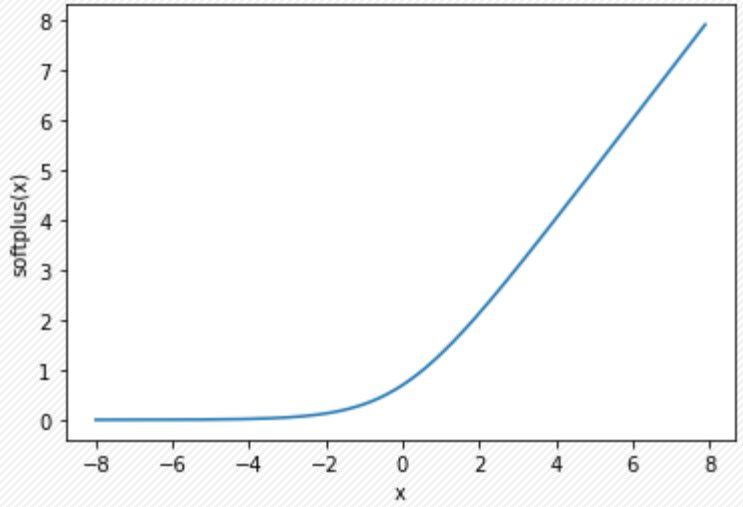


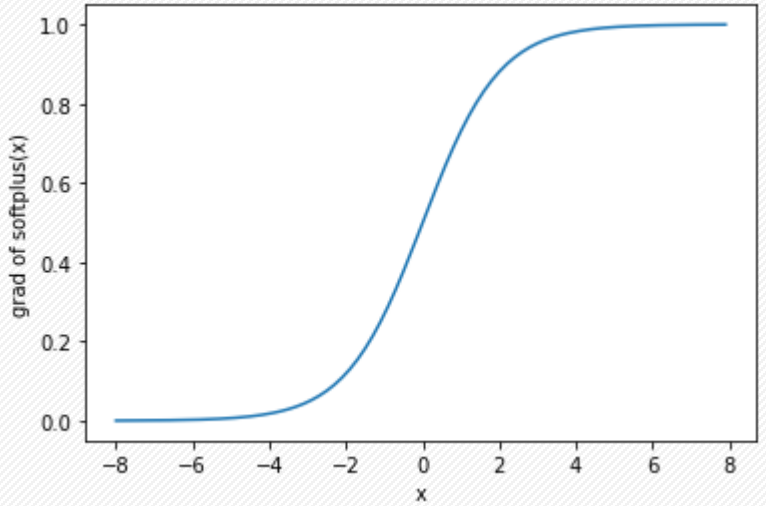


(7) Softplus函数

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

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





常见激活函数

(8) Softmax函数

torch.nn.Softmax(dim=None)

Applies the Softmax function to an n-dimensional input Tensor rescaling them so that the elements of the n-dimensional output Tensor lie in the range [0,1] and sum to 1.

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

当$i = j$时

$$\frac{\partial y_i}{\partial x_i} = \frac{e^{x_i}\left(\sum_{t=1}^{k} e^{x_t}\right) - e^{x_i}e^{x_i}}{\left(\sum_{t=1}^{k} e^{x_t}\right)^2} = y_i - y_i^2$$

当$i \neq j$时

$$\frac{\partial y_i}{\partial x_j} = \frac{0 - e^{x_i}e^{x_j}}{\left(\sum_{t=1}^{k} e^{x_t}\right)^2} = 1 - y_i\, y_j$$

```
from torch import nn
m = nn.Softmax(dim=1)
input = torch.randn(2, 3)
print(input)
output = m(input)
print(output)
```

常见激活函数

(7) Softmax激活函数

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}, \quad i = 1,2,\ldots,k$$

```
import numpy as np

def Softmax(x):
    exp_x=np.exp(x)
    softmax_x=exp_x/np.sum(exp_x)
    return softmax_x

Softmax([1,2,3])
Softmax([1000,2000,3000])
```

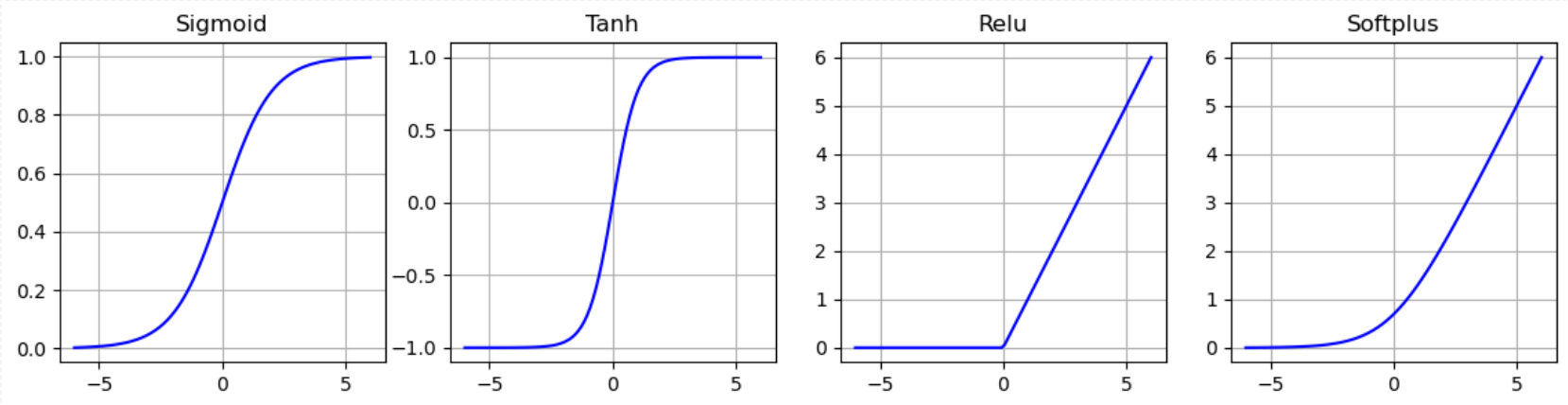
```
array([0.09003057, 0.24472847, 0.66524096])
array([nan, nan, nan])
```

```
import numpy as np

def Softmax(x):
    x-=np.max(x)
    exp_x=np.exp(x)
    softmax_x=exp_x/np.sum(exp_x)
    return softmax_x

Softmax([1,2,3])
Softmax([1000,2000,3000])
```

```
array([0.09003057, 0.24472847, 0.66524096])
array([0., 0., 1.])
```

常见激活函数及其导数

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



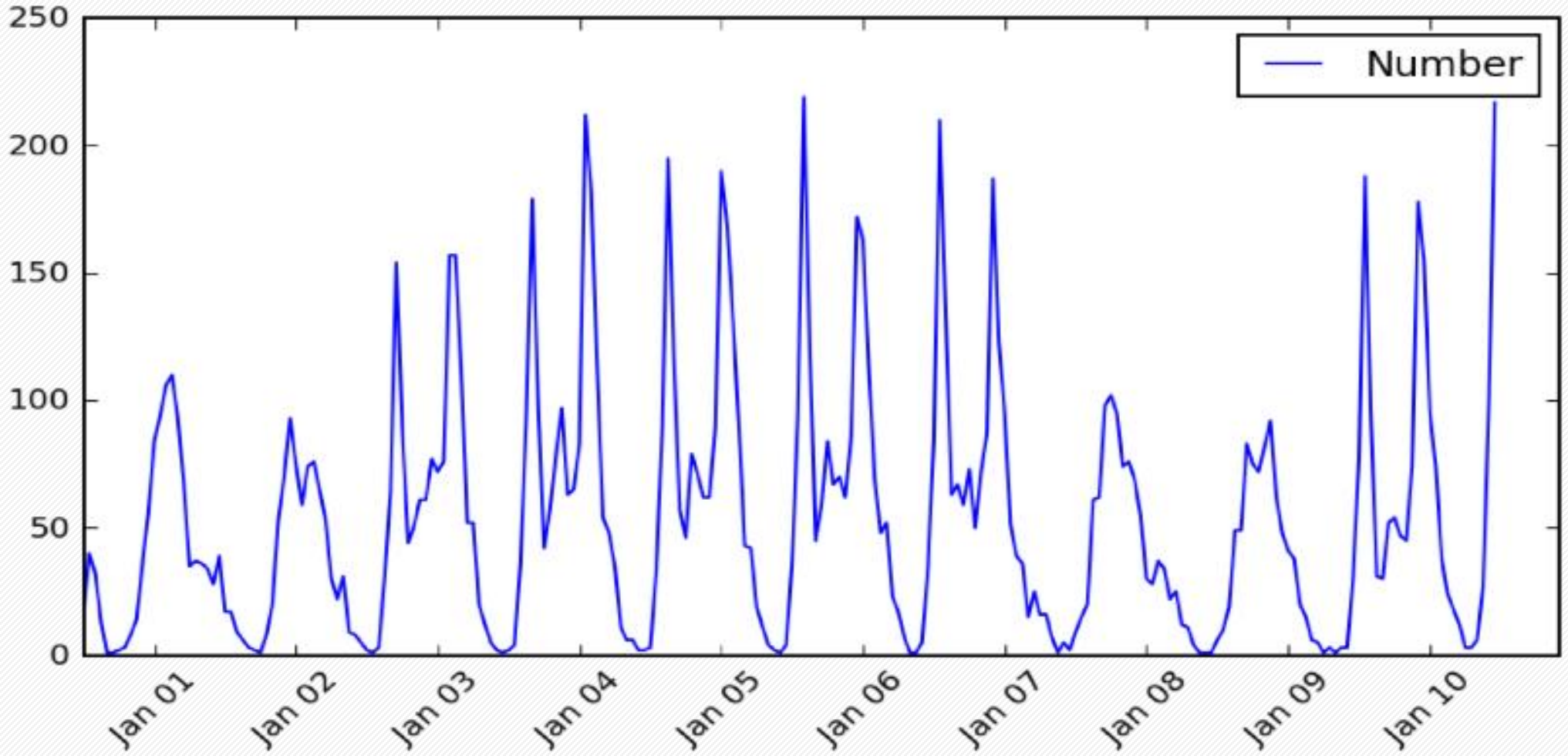
单车预测器

https://www.capitalbikeshare.com/system-data。

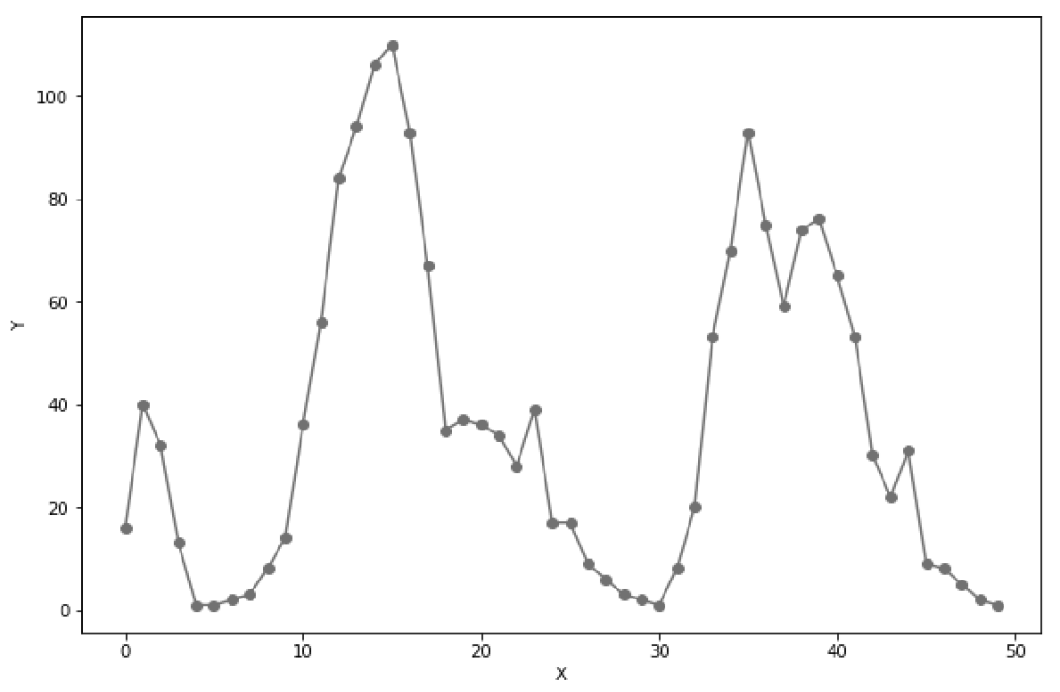
Capital Bikeshare共享单车原始数据

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
instant	dteday	season	yr	mnth	hr	holiday	weekday	workingda	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
1	2011/1/1	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0	3	13	16
2	2011/1/1	1	0	1	1	0	6	0	1	0.22	0.2727	0.8	0	8	32	40
3	2011/1/1	1	0	1	2	0	6	0	1	0.22	0.2727	0.8	0	5	27	32
4	2011/1/1	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0	3	10	13
5	2011/1/1	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0	0	1	1
6	2011/1/1	1	0	1	5	0	6	0	2	0.24	0.2576	0.75	0.0896	0	1	1
7	2011/1/1	1	0	1	6	0	6	0	1	0.22	0.2727	0.8	0	2	0	2
8	2011/1/1	1	0	1	7	0	6	0	1	0.2	0.2576	0.86	0	1	2	3
9	2011/1/1	1	0	1	8	0	6	0	1	0.24	0.2879	0.75	0	1	7	8
10	2011/1/1	1	0	1	9	0	6	0	1	0.32	0.3485	0.76	0	8	6	14
11	2011/1/1	1	0	1	10	0	6	0	1	0.38	0.3939	0.76	0.2537	12	24	36
12	2011/1/1	1	0	1	11	0	6	0	1	0.36	0.3333	0.81	0.2836	26	30	56
13	2011/1/1	1	0	1	12	0	6	0	1	0.42	0.4242	0.77	0.2836	29	55	84
14	2011/1/1	1	0	1	13	0	6	0	2	0.46	0.4545	0.72	0.2985	47	47	94
15	2011/1/1	1	0	1	14	0	6	0	2	0.46	0.4545	0.72	0.2836	35	71	106
16	2011/1/1	1	0	1	15	0	6	0	2	0.44	0.4394	0.77	0.2985	40	70	110
17	2011/1/1	1	0	1	16	0	6	0	2	0.42	0.4242	0.82	0.2985	41	52	93
18	2011/1/1	1	0	1	17	0	6	0	2	0.44	0.4394	0.82	0.2836	15	52	67
19	2011/1/1	1	0	1	18	0	6	0	3	0.42	0.4242	0.88	0.2537	9	26	35
20	2011/1/1	1	0	1	19	0	6	0	3	0.42	0.4242	0.88	0.2537	6	31	37
21	2011/1/1	1	0	1	20	0	6	0	2	0.4	0.4091	0.87	0.2537	11	25	36
22	2011/1/1	1	0	1	21	0	6	0	2	0.4	0.4091	0.87	0.194	3	31	34
23	2011/1/1	1	0	1	22	0	6	0	2	0.4	0.4091	0.94	0.2239	11	17	28
24	2011/1/1	1	0	1	23	0	6	0	2	0.46	0.4545	0.88	0.2985	15	24	39
25	2011/1/2	1	0	1	0	0	0	0	2	0.46	0.4545	0.88	0.2985	4	13	17
26	2011/1/2	1	0	1	1	0	0	0	2	0.44	0.4394	0.94	0.2537	1	16	17
27	2011/1/2	1	0	1	2	0	0	0	2	0.42	0.4242	1	0.2836	1	8	9
28	2011/1/2	1	0	1	3	0	0	0	2	0.46	0.4545	0.94	0.194	2	4	6
29	2011/1/2	1	0	1	4	0	0	0	2	0.46	0.4545	0.94	0.194	2	1	3
30	2011/1/2	1	0	1	6	0	0	0	3	0.42	0.4242	0.77	0.2985	0	2	2
31	2011/1/2	1	0	1	7	0	0	0	2	0.4	0.4091	0.76	0.194	0	1	1
32	2011/1/2	1	0	1	8	0	0	0	3	0.4	0.4091	0.71	0.2239	0	8	8
33	2011/1/2	1	0	1	9	0	0	0	2	0.38	0.3939	0.76	0.2239	1	19	20
34	2011/1/2	1	0	1	10	0	0	0	2	0.36	0.3485	0.81	0.2239	7	46	53
35	2011/1/2	1	0	1	11	0	0	0	2	0.36	0.3333	0.71	0.2537	16	54	70

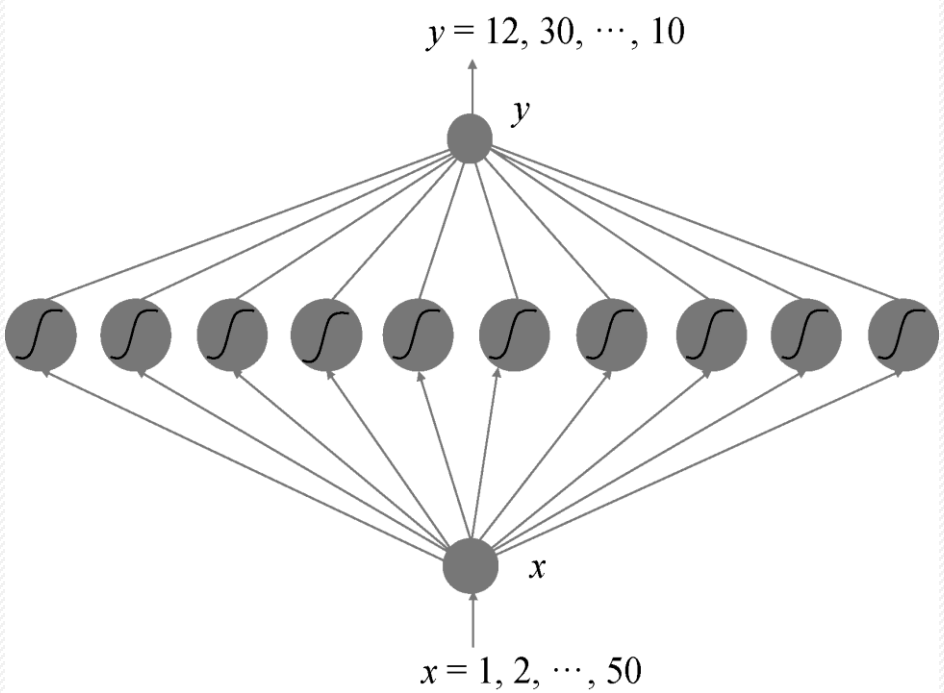
一段时间内单车数量随时间的变化



失败的神经网络预测器1



部分单车数据曲线



神经网络架构

失败的神经网络预测器1

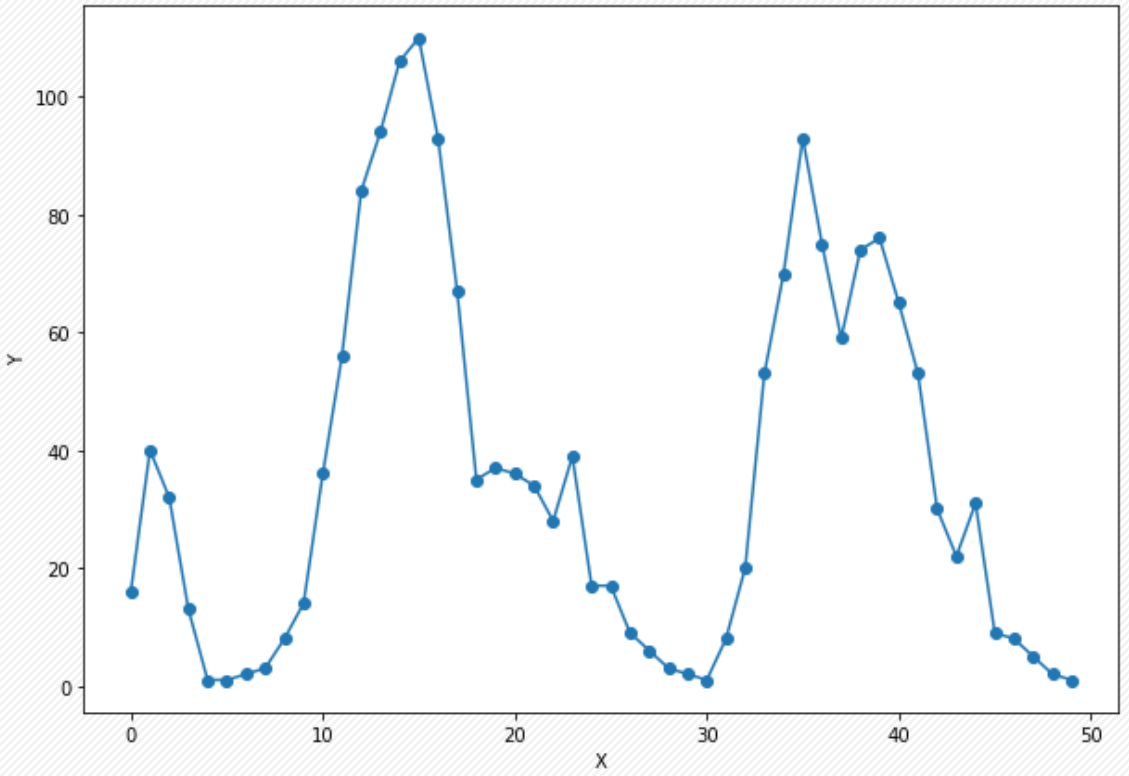
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.optim as optim
%matplotlib inline # 输出的图形直接在Notebook中显示
```

```
#读取数据到内存，rides为一个dataframe对象
data_path = 'hour.csv'
rides = pd.read_csv(data_path)
rides.head()
```

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

失败的神经网络预测器1

```
#取出最后一列的前50条记录
counts = rides['cnt'][:50]
#获得变量x，取值为0，1，……，49
x = np.arange(len(counts))
# 将counts转成标签
y = np.array(counts)

plt.figure(figsize = (10, 7)) #设定绘图窗口大小
plt.plot(x, y, 'o-')  # 绘制原始数据
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



前50条记录的单车数量曲线

失败的神经网络预测器1

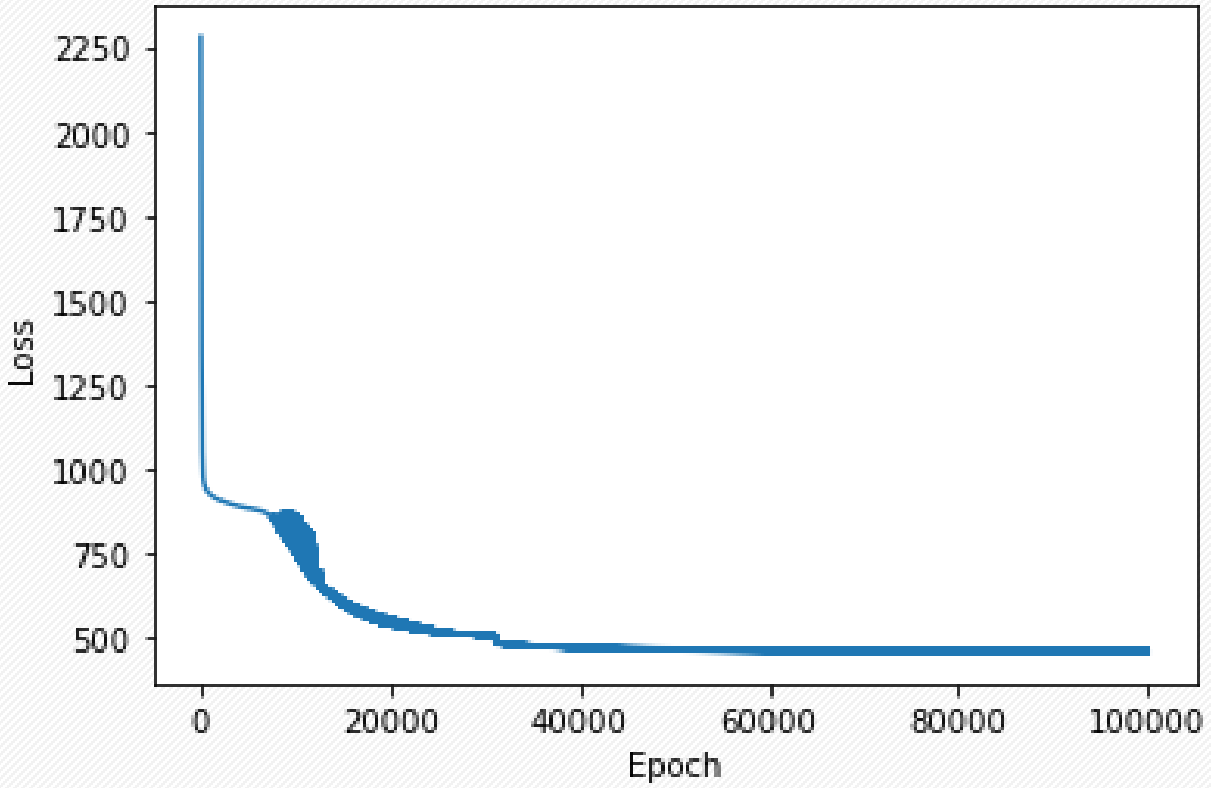
```
#创建变量x，取值1，2，…，50
x = torch.tensor(np.arange(len(counts), dtype = float))
# 将counts转成真值标签
y = torch.tensor(np.array(counts, dtype = float))
# 设置隐层神经元数量
sz = 10
# 初始化神经网络的权重（weights）和阈值（biases）
weights = torch.randn((1, sz), dtype = torch.double, requires_grad = True)  #输入到隐层的权重矩阵
biases = torch.randn(sz, dtype = torch.double, requires_grad = True)  #隐含节点偏置向量
weights2 = torch.randn((sz, 1), dtype = torch.double, requires_grad = True)  #隐层到输出层权重矩阵
learning_rate = 0.001 #学习率
losses = []

# 将 x 转换为(50,1)的维度
x = x.view(50, -1)
# 将 y 转换为(50,1)的维度
y = y.view(50, -1)
```

失败的神经网络预测器1

```
for i in range(100000):
    hidden = x * weights + biases  #输入层到隐层的计算
    hidden = torch.sigmoid(hidden)
    predictions = hidden.mm(weights2)     #隐层输出到输出层
    loss = torch.mean((predictions - y) ** 2)
    losses.append(loss.data.numpy())
    if i % 10000 == 0:
        print('loss:', loss)

    loss.backward()
    weights.data.add_(- learning_rate * weights.grad.data)
    biases.data.add_(- learning_rate * biases.grad.data)
    weights2.data.add_(- learning_rate * weights2.grad.data)

    #清空梯度
    weights.grad.data.zero_()
    biases.grad.data.zero_()
    weights2.grad.data.zero_()
```

失败的神经网络预测器1

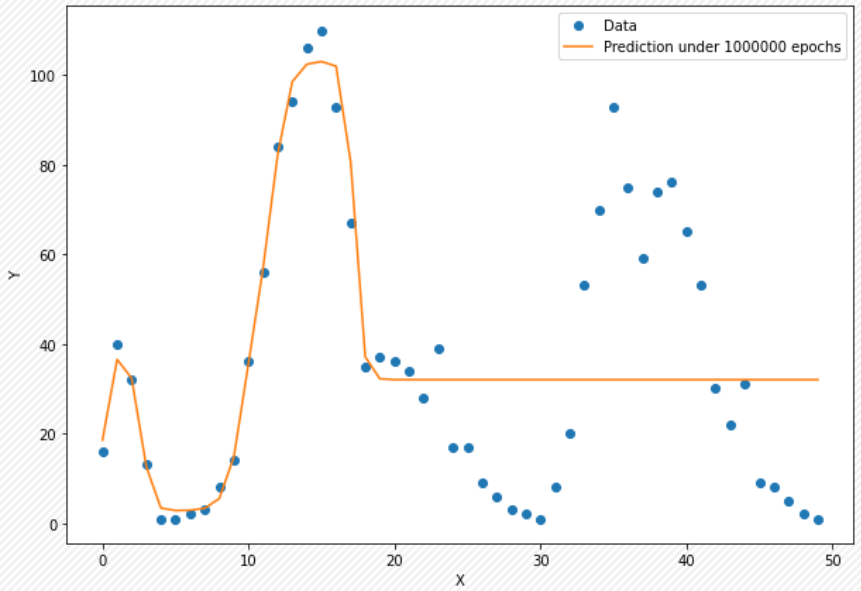
```
plt.plot(losses)
plt.xlabel('Epoch')
plt.ylabel('Loss')
```



模型的Loss曲线

失败的神经网络预测器1

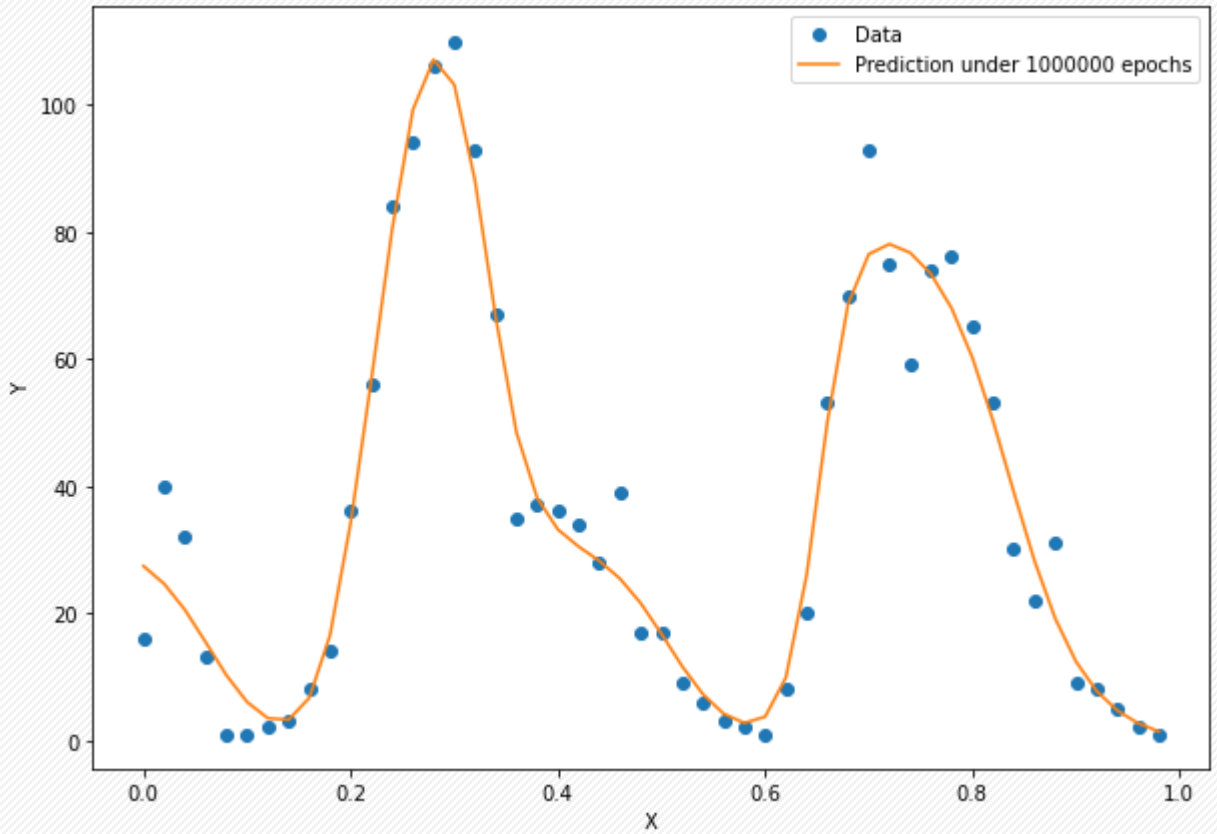
```
x_data = x.data.numpy()
plt.figure(figsize = (10, 7))
xplot, = plt.plot(x_data, y.data.numpy(), 'o')  #绘制原始数据
yplot, = plt.plot(x_data, predictions.data.numpy())  #绘制拟合数据
plt.xlabel('X')
plt.ylabel('Y')
plt.legend([xplot, yplot],['Data', 'Prediction under 1000000 epochs'])  #绘制图例
plt.show()
```



模型拟合训练数据的可视化

失败的神经网络预测器2

```
x=torch.FloatTensor(np.arange(len(counts), dtype=float) / len(counts))
```



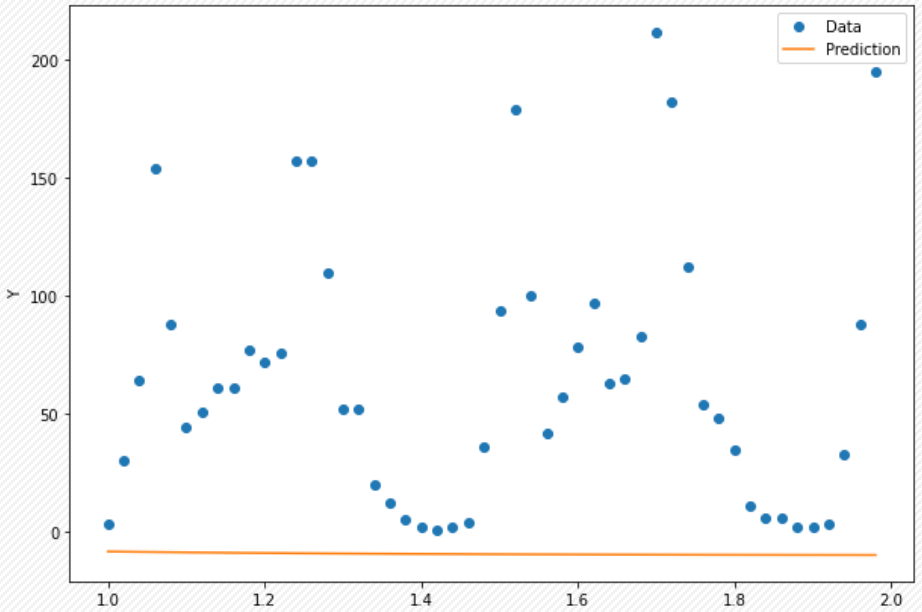
改进的模型拟合训练数据的可视化

开展模型预测

```
counts_predict = rides['cnt'][50:100]  #读取后续50个数据点
#首先对50个数据点进行选取，注意x取50，51，……，99，并归一化
x = torch.tensor((np.arange(50, 100, dtype = float) / len(counts)), requires_grad = True)
#读取对应标签y
y = torch.tensor(np.array(counts_predict, dtype = float), requires_grad = True)
x = x.view(50, -1)
y = y.view(50, -1)

# 从输入层到隐含层的计算
hidden = x * weights + biases
# 将sigmoid函数作用在隐层的每一个神经元上
hidden = torch.sigmoid(hidden)
# 隐含层输出到输出层，计算得到最终预测
predictions = hidden.mm(weights2)
# 计算损失
loss = torch.mean((predictions - y) ** 2)
print(loss)
```

开展模型预测

```
x_data = x.data.numpy()
plt.figure(figsize = (10, 7))  #设定绘图窗口大小
xplot, = plt.plot(x_data, y.data.numpy(), 'o')  # 绘制原始数据
yplot, = plt.plot(x_data, predictions.data.numpy())  #绘制拟合数据
plt.xlabel('X')
plt.ylabel('Y')
plt.legend([xplot, yplot],['Data', 'Prediction'])  #绘制图例
plt.show()
```



模型在测试数据上预测失败的曲线

人工神经网络Neu

数据的预处理过程

① 类型变量：在几种不同的类别中取值。例如星期（week）变量值0, 1, …, 6分别代表星期日、星期一、……星期六。而天气状况（weathersit）变量用1~4表示。其中，1表示晴天，2为多云，3为小雨/雪，4为大雨/雪。

② 数值变量：取值某个数值区间。例如，湿度（humidity）是[0, 1]区间取值的连续变量。此外还有温度、风速等。

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	3	13	16
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	8	32	40
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	5	27	32
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	3	10	13
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	0	1	1

人工神经网络Neu

1. “独热”类型变量

例如季节变量season取值1,2,3,4。不能将其直接输入神经网络，因为数值高低并不表示相应信号强度的大小。解决方案是将类型变量用“独热“（one-hot）编码：

$$seaon = 1 \rightarrow (1, 0, 0, 0)$$
$$seaon = 2 \rightarrow (0, 1\ 0, 0)$$
$$seaon = 3 \rightarrow (0, 0, 1, 0)$$
$$seaon = 4 \rightarrow (0, 0, 0, 1)$$

```
dummy_fields = ['season', 'weathersit', 'mnth', 'hr', 'weekday']
for each in dummy_fields:
    #将一个类型变量属性进行one-hot编码，变成多个属性
    dummies = pd.get_dummies(rides[each], prefix=each, drop_first=False)
    rides = pd.concat([rides, dummies], axis=1)
# 将一些不相关的特征去掉
fields_to_drop = ['instant', 'dteday', 'season', 'weathersit', 'weekday', 'atemp', 'mnth', 'workingday', 'hr']
data = rides.drop(fields_to_drop, axis=1)
```

人工神经网络Neu

2. 标准化数值类型变量

$$temp' = \frac{temp - mean(temp)}{std(temp)}$$

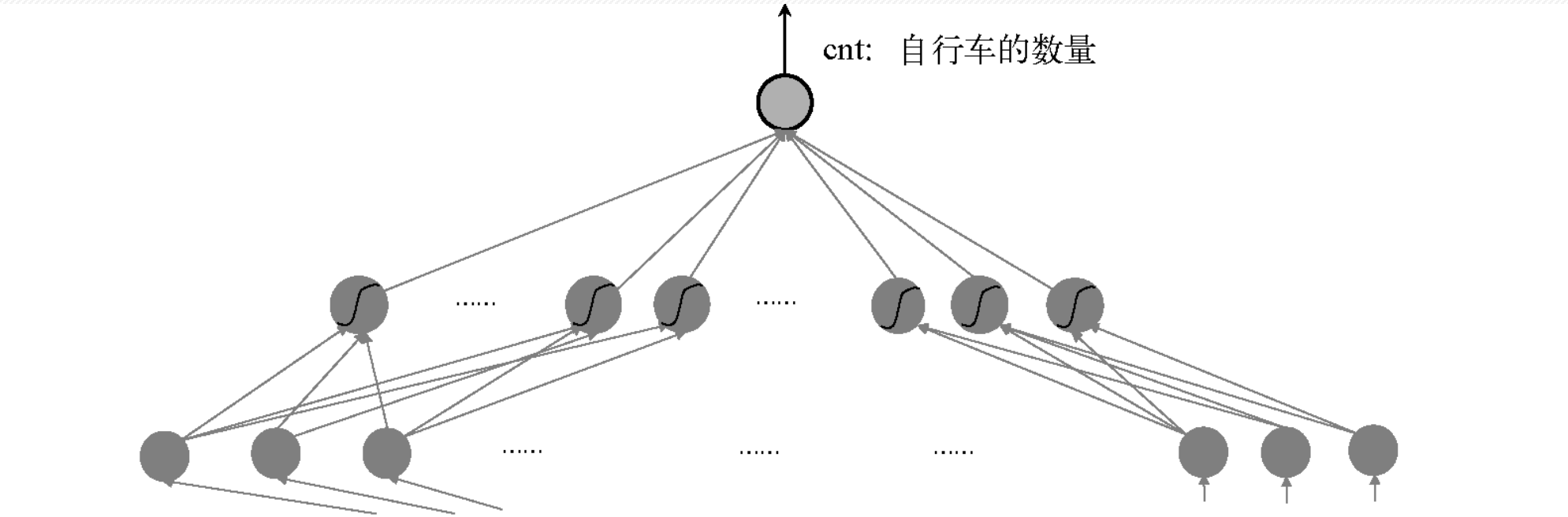
```
quant_features = ['cnt', 'temp', 'hum', 'windspeed']

# 将每个变量的均值和方差存储到scaled_features
scaled_features = {}
for each in quant_features:
    mean, std = data[each].mean(), data[each].std()
    scaled_features[each] = [mean, std]
    data.loc[:, each] = (data[each] - mean)/std
```

人工神经网络Neu

c. 将数据集进行分割

```
#后21天数据一共21*24个样本作为测试集，其它为训练集
test_data = data[-21*24:]
train_data = data[:-21*24]
print( '训练数据：' ,len(train_data), '测试数据：', len(test_data))

#将数据列分为特征列和目标列
target_fields = ['cnt', 'casual', 'registered']
features, targets = train_data.drop(target_fields, axis=1), train_data[target_fields]
test_features, test_targets = test_data.drop(target_fields, axis=1), test_data[target_fields]

# 将数据从dataframe转换为numpy
X = features.values
Y = targets['cnt'].values
Y = Y.astype(float)
Y = np.reshape(Y, [len(Y),1])
losses = []
```

构建神经网络



yr	holiday	temp	hum	windspeed	season_1	season_2	season_3	season_4	weathersit_1	...	weekday_5	weekday_6
0	0	-1.334609	0.947345	-1.553844	1	0	0	0	1	...	0	1
0	0	-1.438475	0.895513	-1.553844	1	0	0	0	1	...	0	1
0	0	-1.438475	0.895513	-1.553844	1	0	0	0	1	...	0	1
0	0	-1.334609	0.636351	-1.553844	1	0	0	0	1	...	0	1
0	0	-1.334609	0.636351	-1.553844	1	0	0	0	1	...	0	1

单车预测神经网络

版本1

```
#定义神经网络架构
input_size = features.shape[1] #输入层单元个数
hidden_size = 10 #隐层单元个数
output_size = 1 #输出层单元个数
batch_size = 128 #每个batch的记录数
weights1 = torch.randn([input_size, hidden_size], dtype = torch.double,  requires_grad = True)
biases1 = torch.randn([hidden_size], dtype = torch.double, requires_grad = True) #隐层偏置
weights2 = torch.randn([hidden_size, output_size], dtype = torch.double, requires_grad = True)
```

```
def neu(x):
    #输出为batch_size * hidden_size矩阵
    hidden = x.mm(weights1) + biases1.expand(x.size()[0], hidden_size)
    hidden = torch.sigmoid(hidden)
    #输出batch_size*output_size矩阵
    output = hidden.mm(weights2)
    return output
```

版本1

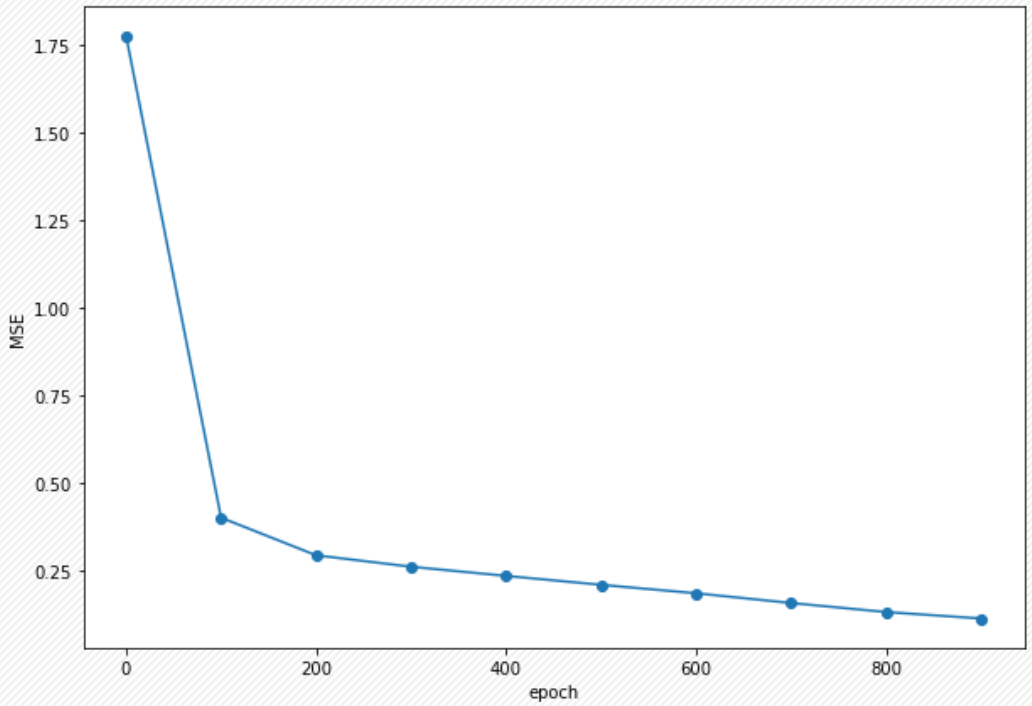
```
def cost(x, y): #损失函数
    error = torch.mean((x - y)**2)
    return error

def zero_grad(): # 清空梯度信息
    if weights1.grad is not None and biases1.grad is not None and weights2.grad is not None:
        weights1.grad.data.zero_()
        weights2.grad.data.zero_()
        biases1.grad.data.zero_()

def optimizer_step(learning_rate):  #更新梯度
    weights1.data.add_(- learning_rate * weights1.grad.data)
    weights2.data.add_(- learning_rate * weights2.grad.data)
    biases1.data.add_(- learning_rate * biases1.grad.data)
```

版本1

```
for i in range(1000):
    batch_loss = []
    # start和end分别是提取一个batch数据的起始和终止下标
    for start in range(0, len(X), batch_size):  # 每128个样本点被为一批
        end = start + batch_size if start + batch_size < len(X) else len(X)
        xx = torch.tensor(X[start:end], dtype = torch.double, requires_grad = True)
        yy = torch.tensor(Y[start:end], dtype = torch.double, requires_grad = True)
        predict = neu(xx)
        loss = cost(predict, yy)
        zero_grad()
        loss.backward()
        optimizer_step(0.01)
        batch_loss.append(loss.data.numpy())

    if i % 100==0:
        losses.append(np.mean(batch_loss))
        print(i, np.mean(batch_loss))
```

版本1

```
# 打印输出损失值
fig = plt.figure(figsize=(10, 7))
plt.plot(np.arange(len(losses))*100,losses, 'o-')
plt.xlabel('epoch')
plt.ylabel('MSE')
```



单车预测器Neu的训练曲线

版本2

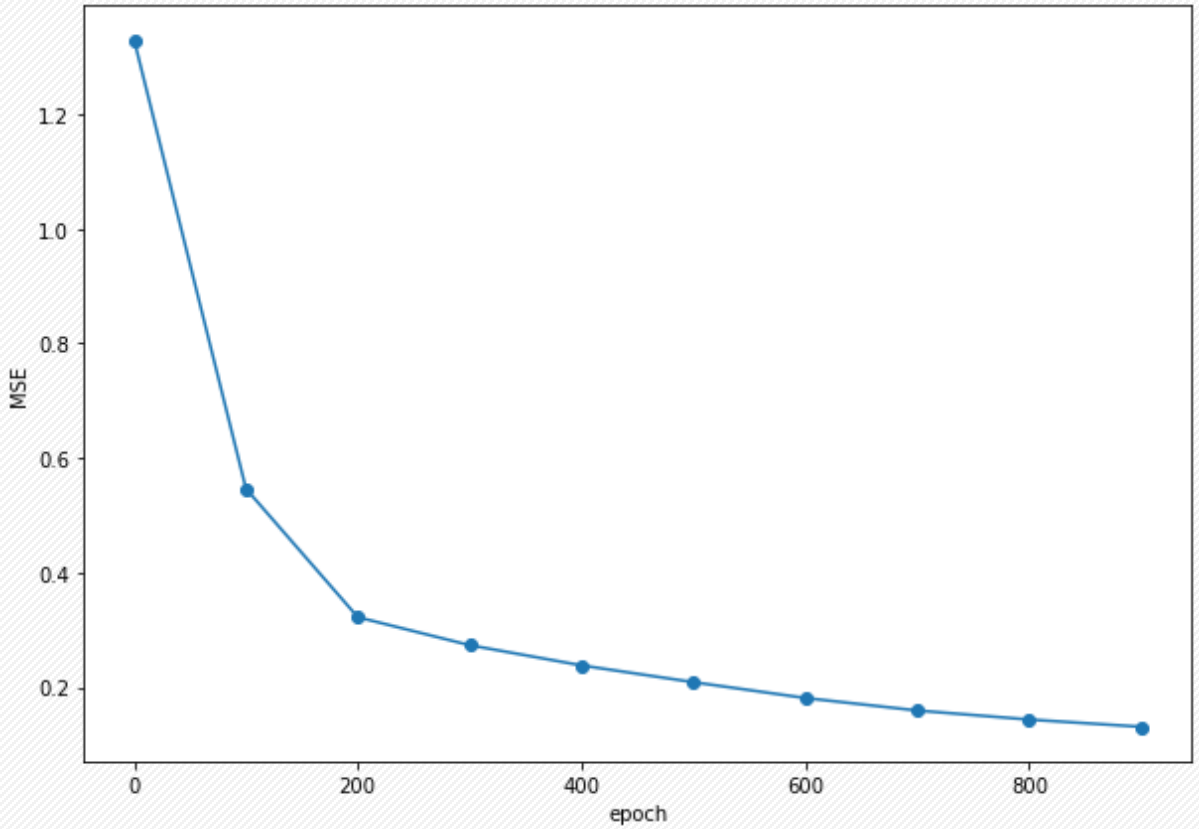
```
#定义网络架构，10个隐层单元， 1个输出层单元
input_size = features.shape[1]
hidden_size = 10
output_size = 1
batch_size = 128
neu = torch.nn.Sequential(
    torch.nn.Linear(input_size, hidden_size),
    torch.nn.Sigmoid(),
    torch.nn.Linear(hidden_size, output_size),
    )
cost = torch.nn.MSELoss()
optimizer = torch.optim.SGD(neu.parameters(), lr=0.01)
```

版本2

```
for i in range(1000):
    batch_loss = []
    for start in range(0, len(X), batch_size):
        end = start + batch_size if start + batch_size < len(X) else len(X)
        xx = torch.tensor(X[start:end], dtype=torch.float, requires_grad=True)
        yy = torch.tensor(Y[start:end], dtype=torch.float, requires_grad=True)
        predict = neu(xx)
        loss = cost(predict, yy)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        batch_loss.append(loss.data.numpy())
    if i%100==0:
        losses.append(np.mean(batch_loss))
        print(i,np.mean(batch_loss))
```

运行结果

```
fig = plt.figure(figsize=(10, 7))
plt.plot(np.arange(len(losses))*100, losses, 'o-')
plt.xlabel('epoch')
plt.yLabel('MSE')
```

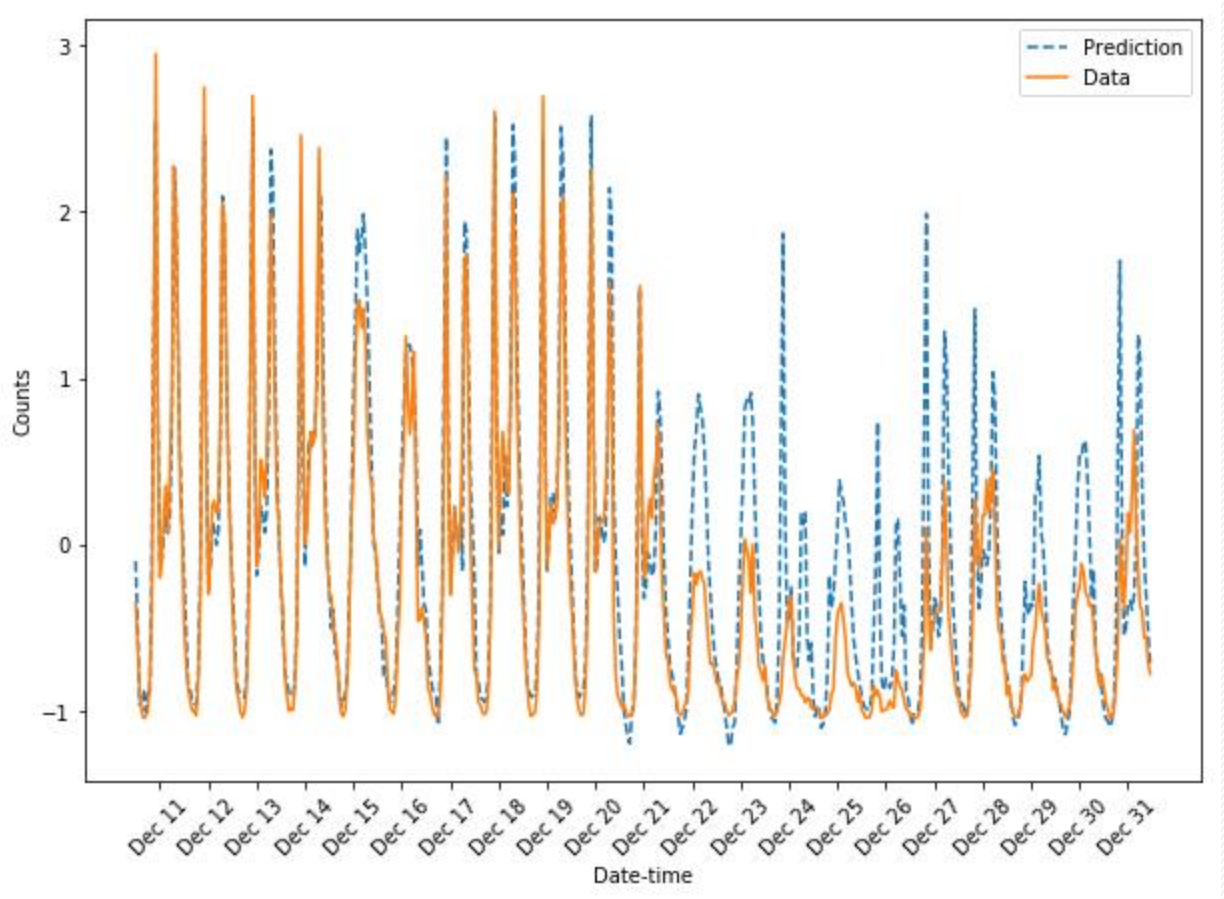


单车预测其Neu的训练曲线

测试神经网络

```
targets = test_targets['cnt'] #读取测试集的cnt数值
targets = targets.values.reshape([len(targets),1])
targets = targets.astype(float)
x = torch.tensor(test_features.values, dtype = torch.float, requires_grad = True)
y = torch.tensor(targets, dtype = torch.float, requires_grad = True)
predict = neu(x) #进行预测
predict = predict.data.numpy()
# 将后21天的预测数据与真实数据画在一起并比较
fig, ax = plt.subplots(figsize = (10, 7))
mean, std = scaled_features['cnt']
ax.plot(predict * std + mean, label='Prediction', linestyle = '--')
ax.plot(targets * std + mean, label='Data', linestyle = '-')
ax.legend()
ax.set_xlabel('Date-time')
ax.set_ylabel('Counts')
dates = pd.to_datetime(rides.loc[test_data.index]['dteday'])
dates = dates.apply(lambda d: d.strftime('%b %d'))
ax.set_xticks(np.arange(len(dates))[12::24])
_ = ax.set_xticklabels(dates[12::24], rotation=45)
```

测试神经网络



实际曲线与预测曲线的对比

常用诀窍 (tricks)

■ 预训练+微调

- 预训练：监督逐层训练，每次训练一层隐结点
- 微调：预训练全部完成后,对全网络进行微调训练，通常使用BP算法

可视为将大量参数分组，对每组先找到较好的局部配置，再全局寻优

权共享 (weight-sharing)　　减少需优化的参数

- 一组神经元使用相同的连接权值

Dropout　　可能：降低 Rademacher 复杂度

- 在每轮训练时随机选择一些隐结点令其权重不被更新(下一轮可能被更新)

ReLU (Rectified Linear Units)　　求导容易；可能：缓解梯度消失现象

- 将 Sigmoid 激活函数修改为修正线性函数　　$f(x) = \max(0, x)$

深度学习

典型的深度学习模型就是深层的神经网络
（例如微软研究院2015年在ImageNet竞赛获胜使用 152层网络）

提升模型复杂度　提升学习能力

- 增加隐层神经元数目 (模型宽度)
- 增加隐层数目 (模型深度)

增加隐层数目比增加隐层神经元数目更有效。不仅增加了拥有激活函数的神经元数，还增加了激活函数嵌套的层数

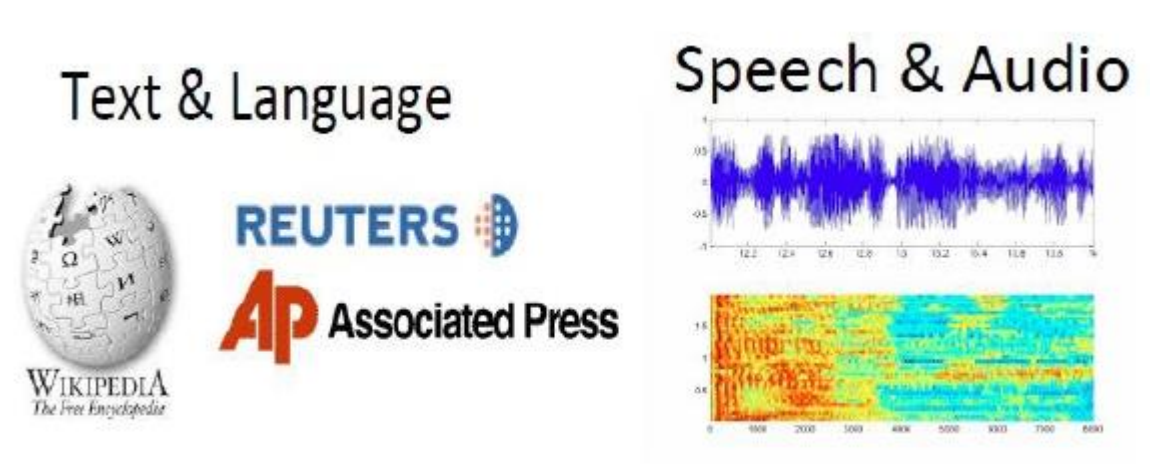
提升模型复杂度　增加过拟合风险；
增加训练难度

- 过拟合风险：使用大量训练数据
- 训练困难：使用若干启发式诀窍

误差梯度在多隐层内传播时，往往会发散而不能收敛到稳定状态，因此难以直接用经典BP算法训练

深度学习的兴起

- 2006年，Hinton发表了深度学习的 Nature 文章
- 2012年，Hinton 组参加 ImageNet 竞赛，使用 CNN 模型以超过第二名10个百分点的成绩夺得当年竞赛的冠军
- 伴随云计算、大数据时代的到来，计算能力的大幅提升，使得深度学习模型在计算机视觉、自然语言处理、语音识别等众多领域都取得了很大的成功。



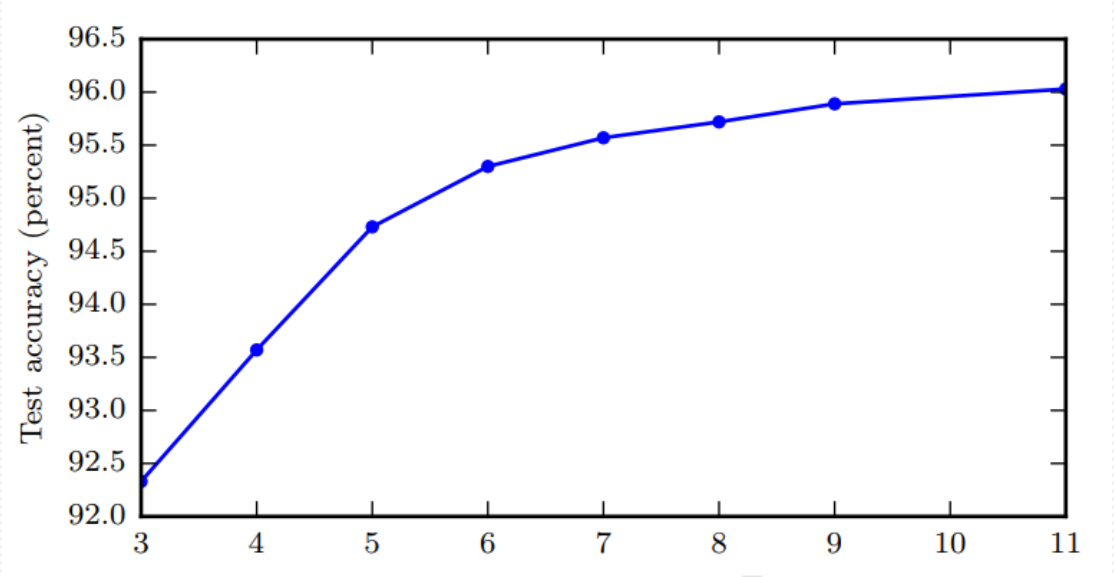
为什么要选择深度模型

通用近似定理(universal approximation theorem)(Hornik et al, 1989;Cybenko, 1989) 表明，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(例如logistic sigmoid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-个有限维空间的Borel可测函数。

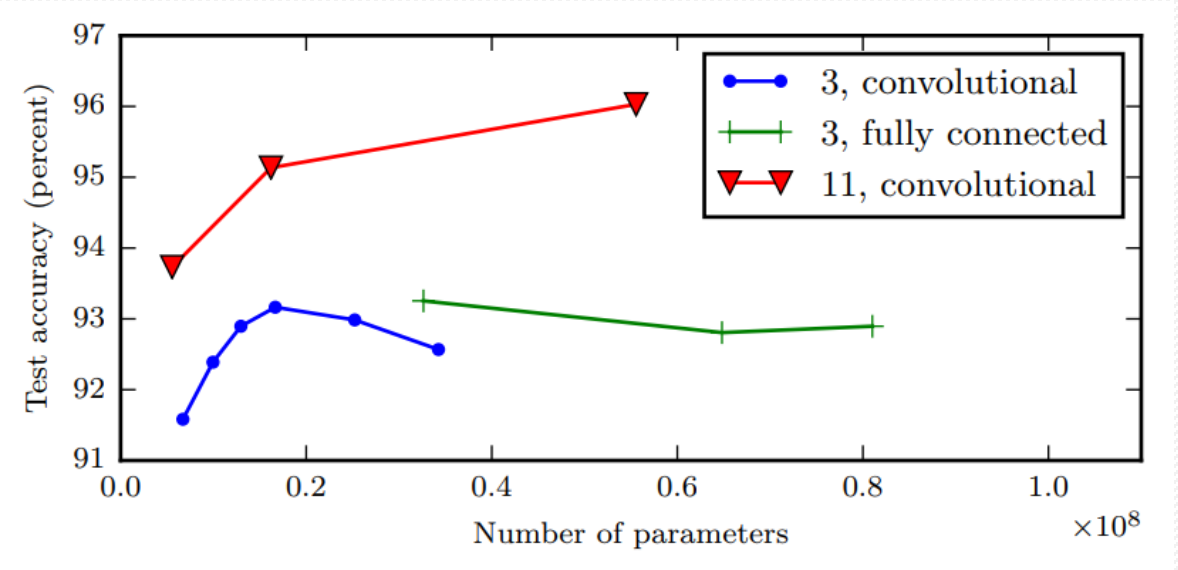
具有单层的前馈网络足以表示任何函数、但是网络层可能不可实现得大并且可能无法正确地学习和泛化。在很多情况下，使用更深的模型能够减少表示期望函数所需的单元的数量，并且可以减少泛化误差。

选择深度模型默许一个非常普遍的信念,那就是我们想要学得的函数应该涉及几个更加简单的函数的组合。这可以从表示学习的观点来解释，我们相信学习的问题包含发现--组潜在的变化因素，它们可以根据其他更简单的潜在的变化因素来描述。或者，我们可以将深度结构的使用解释为另一种信念，那就是我们想要学得的函数是包含多个步骤的计算机程序，其中每个步骤使用前一步骤的输出。这些中间输出不一定是变化的因素，而是可以类似于网络用来组织其内部处理的计数器或指针。

为什么要选择深度模型



深度的影响。实验结果表明，当从地址照片转录多位数字时，更深层的网络能够更好地泛化。 测试集上的准确率随着深度的增加而不断增加。



参数数量的影响。更深的模型往往表现更好。这不仅仅是因为模型更大。数据来自Goodfellow et al. (2014d)。

The end