《神经网络与深度学习》

CCF ADL 65期《知识图谱前沿》

邱锡鹏

xpqiu@fudan.edu.cn

http://nlp.fudan.edu.cn/~xpqiu

微博：http://weibo.com/xpqiu

http://weibo.com/xpqiu?is_hot=1

复旦大学

2015年12月26-27日，北京

大纲

深度学习

- 深度学习是由在计算机上模拟人类神经回路的“神经元网络”技术发展而来。
- 神经元网络是在计算机上把虚拟的神经元排列成层状，模拟真正的神经细胞之间的电信号。借此实现大脑从各式各样的数据中提取本质概念的功能。
- 人工神经网络通过模拟生物神经网络（大脑）的结构和功能，由大量的节点（或称“神经元”，或“单元”）和之间相互联接构成，可以用来对数据之间的复杂关系进行建模。
- 所谓“深度”是指网络层数大于 1。通常是把神经元“深化”到 4-9 层，实现接近于大脑的性能。

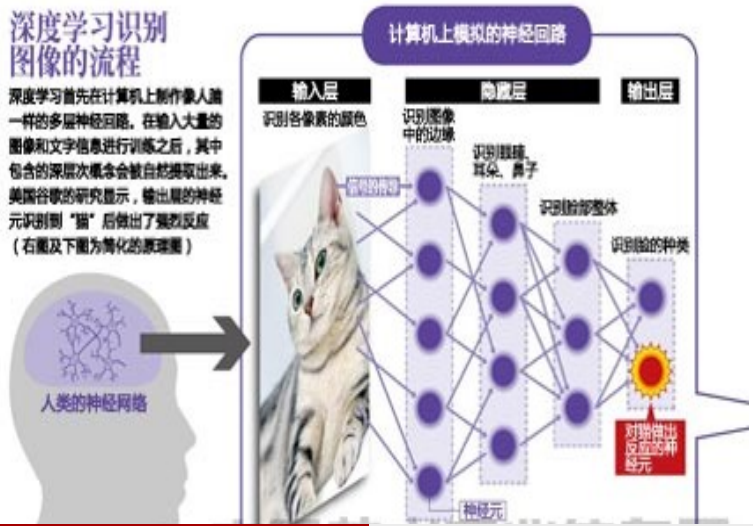
深度学习革命

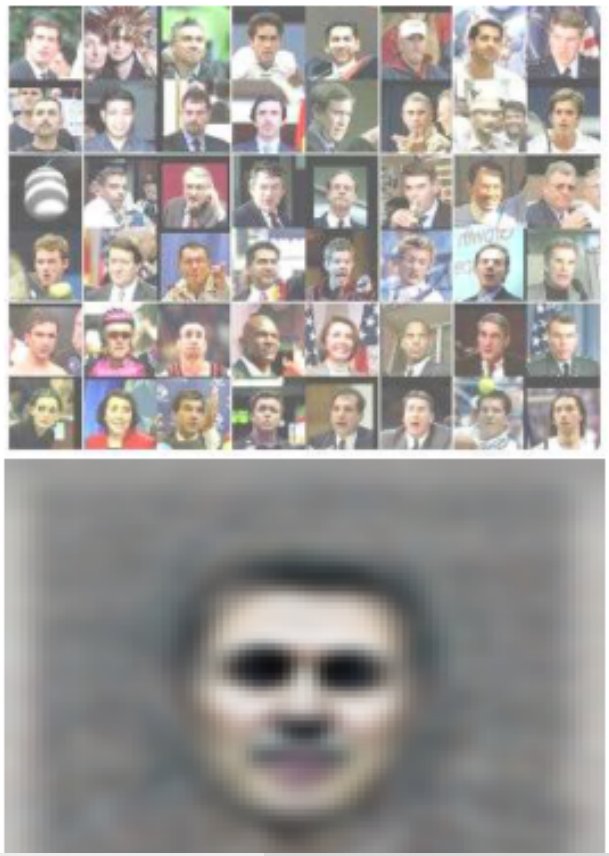
- 语音识别：可以使得词错误率从 $\frac{1}{4}$ 下降到 $\frac{1}{8}$
- 计算机视觉：目标识别、图像分类等
- 自然语言处理：分布式表示、机器翻译、问题回答等

- Deep Blue
- Deep QA
- Deep Learning

深度学习革命-祖母细胞 [Le et al., 2012]

在谷歌的研究之中，参数约为 10 亿个，数量相当庞大。该公司向如此巨大的网络输入了从 1000 万个 YouTube 视频中截取的图像。通过 1.6 万个 CPU（中央运算处理装置）并用的大规模计算，耗费 1 周时间实施了训练。





深度学习历史

- 1958 年 Rosenblatt 感知器
- 1969 年 Minsky XOR
- 1986 年 Hinton、LeCun 人工神经网络（BP 算法）
- 1998 年 LeCun 卷积神经网络
- 2006 年 Hinton 深度网络

学术机构

- Toronto 大学 Hinton 75 年 Edinburgh 大学博士
- NYU Lecun (Now Facebook) 87 年 Hinton 博士后
- Montreal 大学 Bengio 91 年 M. Jordan 博士后
- Stanford 大学 Ng (Now Baidu) 03 年 UC Berkeley 大学 M. Jordan 博士
- IDSIA Jürgen Schmidhuber

公司

- Google (DeepMind, Hinton)
- Microsoft
- Facebook (Lecun)
- Baidu (Andrew Ng)

深度学习难点

- 参数过多，影响训练
- 非凸优化问题：即存在局部最优而非全局最优解，影响迭代
- 下层参数比较难调
- 参数解释起来比较困难

需求

- 计算资源要大
- 数据要多
- 算法效率要好：即收敛快

大纲

向量

在线性代数中，**标量**（Scalar）是一个实数，而**向量**（Vector）是指 n 个实数组成的有序数组，称为 n 维向量。如果没有特别说明，一个 n 维向量一般表示列向量，即大小为 $n \times 1$ 的矩阵。

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, \tag{1}$$

向量符号一般用黑体小写字母 $\mathbf{a}, \mathbf{b}, \mathbf{c}$，或小写希腊字母 α, β, γ 等来表示。

向量的模和范数

向量 $\mathbf{a}$ 的模 $\|\mathbf{a}\|$ 为

$$\|\mathbf{a}\| = \sqrt{\sum_{i=1}^{n} a_i^2}. \tag{2}$$

在线性代数中，**范数**（norm）是一个表示“长度”概念的函数，为向量空间内的所有向量赋予非零的正长度或大小。对于一个 n 维的向量 $\mathbf{x}$，其常见的范数有：

L_1 范数：

$$|\mathbf{x}|_1 = \sum_{i=1}^{n} |x_i|. \tag{3}$$

L_2 范数：

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{\mathbf{x}^T\mathbf{x}}. \tag{4}$$

常见的向量

全1向量指所有值为1的向量。用$\mathbf{1}_n$表示，n表示向量的维数。

$\mathbf{1}_K = [1, \cdots, 1]^\top_{K\times 1}$是$K$维的全1向量。

one-hot 向量表示一个n维向量，其中只有一维为1，其余元素都为0。在数字电路中，one-hot是一种状态编码，指对任意给定的状态，状态寄存器中只有1位为1，其余位都为0。

矩阵

一个大小为 $m \times n$ 的**矩阵**（Matrix）是一个由 m 行 n 列元素排列成的矩形阵列。矩阵里的元素可以是数字、符号或数学式。这里，矩阵我们一般默认指数字矩阵。

一个 n 维向量可以看作是 $n \times 1$ 的矩阵。

矩阵的基本运算

如果 A 和 B 都为 $m \times n$ 的矩阵，则 A 和 B 的加减为

$$(A + B)_{ij} = A_{ij} + B_{ij}, \tag{5}$$

$$(A - B)_{ij} = A_{ij} - B_{ij}. \tag{6}$$

A 和 B 的点乘 $A \odot B \in \mathbb{R}^{m \times n}$ 为

$$(A \odot B)_{ij} = A_{ij} B_{ij}. \tag{7}$$

一个标量 c 与矩阵 A 乘积为

$$(cA)_{ij} = cA_{ij}. \tag{8}$$

若 A 是 $m \times p$ 矩阵和 B 是 $p \times n$ 矩阵，则乘积 AB 是一个 $m \times n$ 的矩阵

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj} \tag{9}$$

矩阵的基本运算

$m \times n$ 矩阵 A 的**转置**（Transposition）是一个 $n \times m$ 的矩阵，记为 $A^\top$，$A^\top$ 第 i 行第 j 列的元素是原矩阵 A 第 j 行第 i 列的元素，

$$(A^\top)_{ij} = A_{ji}. \tag{10}$$

矩阵的**向量化**是将矩阵表示为一个列向量。这里，**vec** 是向量化算子。设 $A = [a_{ij}]_{m \times n}$，则

$$\mathbf{vec}(A) = [a_{11}, a_{21}, \cdots, a_{m1}, a_{12}, a_{22}, \cdots, a_{m2}, \cdots, a_{1n}, a_{2n}, \cdots, a_{mn}]^\top.$$

常见的矩阵

对称矩阵指其转置等于自己的矩阵，即满足 $A = A^\top$。

对角矩阵（Diagonal Matrix）是一个主对角线之外的元素皆为 0 的矩阵。对角线上的元素可以为 0 或其他值。一个 $n \times n$ 的对角矩阵矩阵 A 满足：

$$A_{ij} = 0 \text{ if } i \neq j \qquad \forall i, j \in \{1, \cdots, n\} \tag{11}$$

对角矩阵 A 也可以记为 $\mathbf{diag}(\mathbf{a})$，$\mathbf{a}$ 为一个 n 维向量，并满足

$$A_{ii} = a_i. \tag{12}$$

$n \times n$ 的对角矩阵矩阵 $A = \mathbf{diag}(\mathbf{a})$ 和 n 维向量 $\mathbf{b}$ 的乘积为一个 n 维向量

$$A\mathbf{b} = \mathbf{diag}(\mathbf{a})\mathbf{b} = \mathbf{a} \odot \mathbf{b}, \tag{13}$$

其中 $\odot$ 表示点乘，即 $(\mathbf{a} \odot \mathbf{b})_i = a_i b_i$。

常见的矩阵

单位矩阵是一种特殊的的对角矩阵，其主对角线元素为 1，其余元素为 0。n 阶单位矩阵 I_n，是一个 $n \times n$ 的方形矩阵。可以记为 $I_n = \mathbf{diag}(1, 1, \ldots, 1)$。

一个矩阵和单位矩阵的乘积等于其本身。

$$AI = IA = A \tag{14}$$

导数

对于定义域和值域都是实数域的函数 $y = f(x)$;。若 $f(x)$ 在点 x_0 的某个邻域 Δx 内，极限

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \tag{15}$$

存在，则称函数 $y = f(x)$ 在点 x_0 处可导，并导数为 $f'(x_0)$。

若函数 $f(x)$ 在其定义域包含的某区间内每一个点都可导，那么也可以说函数 $f(x)$ 在这个区间内可导。这样，我们可以定义函数 $f'(x)$ 为函数 $f(x)$ 的**导函数**，通常也成为**导数**。

函数 $f(x)$ 的**导数** $f'(x)$ 也可记作 $\nabla_x f(x)$，$\frac{\partial f(x)}{\partial x}$ 或 $\frac{\partial}{\partial x} f(x)$。

向量导数

对于一个 p 维向量 $\mathbf{x} \in \mathbb{R}^p$，函数 $y = f(\mathbf{x}) = f(x_1, \cdots, x_p) \in \mathbb{R}$，则 y 关于 $\mathbf{x}$ 的导数为

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_p} \end{bmatrix} \in \mathbb{R}^p. \tag{16}$$

对于一个 p 维向量 $\mathbf{x} \in \mathbb{R}^p$，函数 $\mathbf{y} = f(\mathbf{x}) = f(x_1, \cdots, x_p) \in \mathbb{R}^q$，则 $\mathbf{y}$ 关于 $\mathbf{x}$ 的导数为

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_q(\mathbf{x})}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_1(\mathbf{x})}{\partial x_p} & \cdots & \frac{\partial f_q(\mathbf{x})}{\partial x_p} \end{bmatrix} \in \mathbb{R}^{p \times q}. \tag{17}$$

导数法则

加（减）法则

$\mathbf{y} = f(\mathbf{x}), \mathbf{z} = g(\mathbf{x})$ 则

$$\frac{\partial(\mathbf{y} + \mathbf{z})}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} + \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \tag{18}$$

导数法则

乘法法则

（1）若 $\mathbf{x} \in \mathbb{R}^p$，$\mathbf{y} = f(\mathbf{x}) \in \mathbb{R}^q$，$\mathbf{z} = g(\mathbf{x}) \in \mathbb{R}^q$，则

$$\frac{\partial \mathbf{y}^\top \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \mathbf{z} + \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \mathbf{y} \tag{19}$$

（2）若 $\mathbf{x} \in \mathbb{R}^p$，$\mathbf{y} = f(\mathbf{x}) \in \mathbb{R}^s$，$\mathbf{z} = g(\mathbf{x}) \in \mathbb{R}^t$，$A \in \mathbb{R}^{s \times t}$ 和 $\mathbf{x}$ 无关，则

$$\frac{\partial \mathbf{y}^\top A \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} A \mathbf{z} + \frac{\partial \mathbf{z}}{\partial \mathbf{x}} A^\top \mathbf{y} \tag{20}$$

（3）若 $\mathbf{x} \in \mathbb{R}^p$，$y = f(\mathbf{x}) \in \mathbb{R}$，$\mathbf{z} = g(\mathbf{x}) \in \mathbb{R}^p$，则

$$\frac{\partial y\mathbf{z}}{\partial \mathbf{x}} = y \frac{\partial \mathbf{z}}{\partial \mathbf{x}} + \frac{\partial y}{\partial \mathbf{x}} \mathbf{z}^\top \tag{21}$$

导数法则

链式法则

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

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \tag{22}$$

（2）若 $X \in \mathbb{R}^{p \times q}$ 为矩阵，$Y = g(X) \in \mathbb{R}^{s \times t}$，$z = f(Y) \in \mathbb{R}$，

$$\frac{\partial z}{\partial X_{ij}} = \mathbf{tr}\left((\frac{\partial z}{\partial Y})^\top \frac{\partial Y}{\partial X_{ij}} \right) \tag{23}$$

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

$$\frac{\partial z}{\partial X_{ij}} = (\frac{\partial z}{\partial \mathbf{y}})^\top \frac{\partial \mathbf{y}}{\partial X_{ij}} \tag{24}$$

常用函数及其导数

指示函数

指示函数 $I(x = c)$ 为

$$I(x = c) == \begin{cases} 1 & \text{if} \quad x = c, \\ 0 & \text{else} \quad 0. \end{cases} \tag{25}$$

指示函数 $I(x = c)$ 除了在 c 外，其导数为 0。

多项式函数

如果 $f(x) = x^r$，其中 r 是非零实数，那么导数

$$\frac{\partial x^r}{\partial x} = rx^{r-1}. \tag{26}$$

当 $r = 0$ 时，常函数的导数是 0。

常用函数及其导数

指数函数 $\exp(x) = e^x$

$$\frac{\partial \exp(x)}{\partial x} = \exp(x). \tag{27}$$

对数函数 $\log(x)$

$$\frac{\partial \log(x)}{\partial x} = \frac{1}{x}. \tag{28}$$

向量函数及其导数

$$\frac{\partial \mathbf{x}}{\partial \mathbf{x}} = I, \tag{29}$$

$$\frac{\partial A\mathbf{x}}{\partial \mathbf{x}} = A^{\top}, \tag{30}$$

$$\frac{\partial \mathbf{x}^{\top} A}{\partial \mathbf{x}} = A \tag{31}$$

按位计算的向量函数及其导数

我们定义 $\mathbf{x} = [x_1, \cdots, x_K]^\top$，$\mathbf{z} = [z_1, \cdots, z_K]^\top$，

$$\mathbf{z} = f(\mathbf{x}), \tag{32}$$

其中，$f(\mathbf{x})$ 是按位运算的，即 $(f(\mathbf{x}))_i = f(x_i)$。

如果 $f(x)$ 的导数记为 $f'(x)$。当这个函数的输入为 K 维向量 $\mathbf{x} = [x_1, \cdots, x_K]^\top$ 时，其导数为一个对角矩阵。

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = [\frac{\partial f(x_j)}{\partial x_i}]_{K\times K} = \begin{bmatrix} f'(x_1) & 0 & \cdots & 0 \\ 0 & f'(x_2) & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & f'(x_K) \end{bmatrix} \tag{33}$$

$$= \mathbf{diag}(f'(\mathbf{x})). \tag{34}$$

logistic 函数

logistic 函数经常用来将一个实数空间的数映射到 $(0, 1)$ 区间，记为 $\sigma(x)$

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{35}$$

其导数为

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{36}$$

$$\tag{37}$$

当输入为 K 维向量 $\mathbf{x} = [x_1, \cdots, x_K]^\top$ 时，其导数为

$$\sigma'(\mathbf{x}) = \mathbf{diag}\left(\sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x}))\right). \tag{38}$$

softmax 函数

softmax 函数是将多个标量映射为一个概率分布。

对于 K 个标量 $x_1, \cdots, x_K$，**softmax** 函数定义为

$$z_k = \textbf{softmax}(x_k) = \frac{\exp(x_k)}{\sum_{i=1}^{K} \exp(x_i)}, \tag{39}$$

这样，我们可以将 K 个变量 $x_1, \cdots, x_K$ 转换为一个分布：$z_1, \cdots, z_K$，满足

$$z_k \in [0, 1], \forall k, \qquad \sum_{i=1}^{K} z_k = 1. \tag{40}$$

softmax 函数

当 **softmax** 函数的输入为 K 维向量 $\mathbf{x}$ 时，

$$\hat{\mathbf{z}} = \mathbf{softmax}(\mathbf{x}) \tag{41}$$

$$= \frac{1}{\sum_{k=1}^{K} \exp(x_k)} \begin{bmatrix} \exp(x_1) \\ \vdots \\ \exp(x_K) \end{bmatrix}$$

$$= \frac{\exp(\mathbf{x})}{\sum_{k=1}^{K} \exp(x_k)}$$

$$= \frac{\exp(\mathbf{x})}{\mathbf{1}_K^{\top} \exp(\mathbf{x})}, \tag{42}$$

其中，$\mathbf{1}_K = [1, \cdots, 1]_{K \times 1}$ 是 K 维的全 1 向量。

softmax 函数

其导数为

$$\begin{aligned}
\frac{\partial \mathbf{softmax}(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial \left(\frac{\exp(\mathbf{x})}{\mathbf{1}_K^\top \exp(\mathbf{x})} \right)}{\partial \mathbf{x}} \\
&= \frac{1}{\mathbf{1}_K \exp(\mathbf{x})} \frac{\partial \exp(\mathbf{x})}{\partial \mathbf{x}} + \frac{\partial \left(\frac{1}{\mathbf{1}_K^\top \exp(\mathbf{x})} \right)}{\partial \mathbf{x}} (\exp(\mathbf{x}))^\top \\
&= \frac{\mathbf{diag}\left(\exp(\mathbf{x})\right)}{\mathbf{1}_K \exp(\mathbf{x})} - \left(\frac{1}{(\mathbf{1}_K^\top \exp(\mathbf{x}))^2} \right) \frac{\partial \left(\mathbf{1}_K^\top \exp(\mathbf{x}) \right)}{\partial \mathbf{x}} (\exp(\mathbf{x}))^\top \\
&= \frac{\mathbf{diag}\left(\exp(\mathbf{x})\right)}{\mathbf{1}_K \exp(\mathbf{x})} - \left(\frac{1}{(\mathbf{1}_K^\top \exp(\mathbf{x}))^2} \right) \mathbf{diag}\left(\exp(\mathbf{x})\right) \mathbf{1}_K (\exp(\mathbf{x}))^\top \\
&= \frac{\mathbf{diag}\left(\exp(\mathbf{x})\right)}{\mathbf{1}_K \exp(\mathbf{x})} - \left(\frac{1}{(\mathbf{1}_K^\top \exp(\mathbf{x}))^2} \right) \exp(\mathbf{x}) (\exp(\mathbf{x}))^\top
\end{aligned} \tag{43}$$

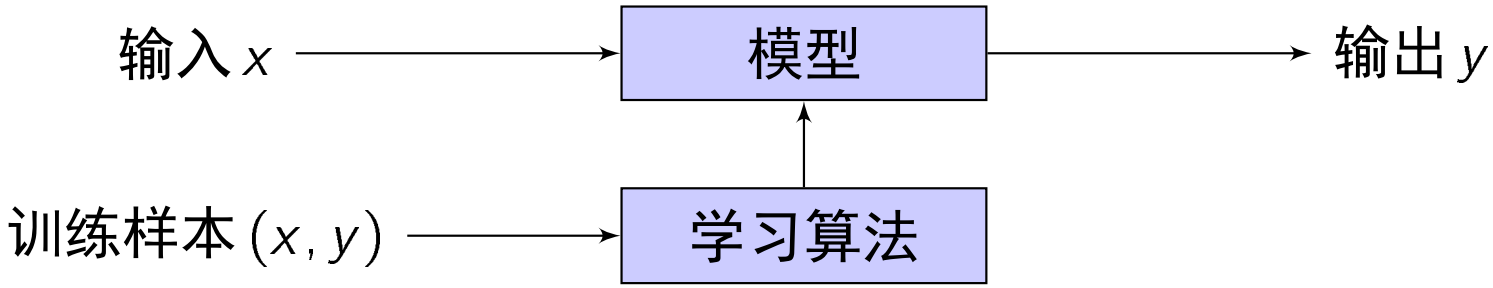
softmax 函数

$$
\begin{aligned}
&= \mathbf{diag}\left(\frac{\exp(\mathbf{x})}{\mathbf{1}_K \exp(\mathbf{x})}\right) - \frac{\exp(\mathbf{x})}{\mathbf{1}_K^\top \exp(\mathbf{x}))} \cdot \frac{(\exp(\mathbf{x}))^\top}{\mathbf{1}_K^\top \exp(\mathbf{x}))} \\
&= \mathbf{diag}\left(\mathbf{softmax}(\mathbf{x})\right) - \mathbf{softmax}(\mathbf{x})\,\mathbf{softmax}(\mathbf{x})^\top, \qquad (44)
\end{aligned}
$$

其中，$\mathbf{diag}\left(\exp(\mathbf{x})\right)\mathbf{1}_K = \exp(\mathbf{x})$。

机器学习

机器学习主要是研究如何使计算机从给定的数据中学习规律，即从观测数据（样本）中寻找规律，并利用学习到的规律（模型）对未知或无法观测的数据进行预测。目前，主流的机器学习算法是基于统计的方法，也叫统计机器学习。



机器学习

狭义地讲，机器学习是给定一些训练样本 $(x_i, y_i), 1 \leq i \leq N$ （其中 x_i 是输入，y_i 是需要预测的目标），让计算机自动寻找一个**决策函数** $f(\cdot)$ 来建立 x 和 y 之间的关系。

$$\hat{y} = f(\phi(x), \theta), \tag{45}$$

这里，$\hat{y}$ 是模型输出，θ 为决策函数的参数，$\phi(x)$ 表示样本 x 对应的特征表示。因为 x 不一定都是数值型的输入，因此需要通过 $\phi(x)$ 将 x 转换为数值型的输入。如果我们假设 x 是已经处理好的标量或向量，公式45也可以直接写为

$$\hat{y} = f(x, \theta). \tag{46}$$

损失函数

我们还要建立一些准则来衡量决策函数的好坏。在很多机器学习算法中，一般是定义一个**损失函数** $L(y, f(x, \theta))$，然后在所有的训练样本上来评价决策函数的风险。

$$R(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(y^{(i)}, f(x^{(i)}, \theta)). \tag{47}$$

这里，风险函数 $R(\theta)$ 是在已知的训练样本（经验数据）上计算得来的，因此被称之为**经验风险**。用对参数求经验风险来逐渐逼近理想的期望风险的最小值，就是我们常说的**经验风险最小化原则**（Empirical Risk Minimization）。这样，我们的目标就是变成了找到一个参数 θ^* 使得经验风险最小。

$$\theta^* = \arg\min_{\theta} R(\theta). \tag{48}$$

过拟合

因为用来训练的样本往往是真实数据的一个很小的子集或者包含一定的噪声数据，不能很好地反映全部数据的真实分布。

经验风险最小化原则很容易导致模型在训练集上错误率很低，但是在未知数据上错误率很高。这就是所谓的**过拟合**。过拟合问题往往是由于训练数据少和噪声等原因造成的。

和过拟合相对应的一个概念是**泛化错误**。

结构风险最小化原则

为了解决过拟合问题，一般在经验风险最小化的原则上上加参数的**正则化**（Regularization），也叫**结构风险最小化原则**（Structure Risk Minimization）。

$$\theta^* = \arg\min_{\theta} R(\theta) + \lambda \|\theta\|_2^2 \tag{49}$$

$$= \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(y^{(i)}, f(x^{(i)}, \theta)) + \lambda \|\theta\|^2. \tag{50}$$

这里，$\|\theta\|_2$ 是 L_2 范数的**正则化项**，用来减少参数空间，避免**过拟合**。λ 用来控制正则化的强度。

正则化项也可以使用其它函数，比如 L_1 范数。L_1 范数的引入通常会使得参数有一定稀疏性，因此在很多算法中也经常使用。在 Bayes 估计的角度来讲，正则化是假设了参数的先验分布，不完全依赖训练数据。

损失函数

给定一个实例(x, y)，真实目标是y，机器学习模型的预测为$f(x, \theta)$。如果预测错误时（$f(x, \theta) \neq y$），我们需要定义一个度量函数来定量地计算错误的程度。常见的损失函数有如下几类：

0-1 损失函数 0-1 损失函数（0-1 loss function）是

$$L(y, f(x, \theta)) = \begin{cases} 1 & \textbf{if } y = f(x, \theta) \\ 0 & \textbf{if } y \neq f(x, \theta) \end{cases} \tag{51}$$

$$= I(y \neq f(x, \theta)), \tag{52}$$

这里 I 是指示函数。

平方损失函数 平方损失函数（quadratic loss function）是

$$L(y, \hat{y}) = (y - f(x, \theta))^2 \tag{53}$$

损失函数

交叉熵损失函数 对于分类问题，预测目标 y 为离散的类别，模型输出 $f(x, \theta)$ 为每个类的条件概率。

假设 $y \in \{1, \cdots C\}$，模型预测的第 i 个类的条件概率 $P(y = i|x) = f_i(x, \theta)$，则 $f(x, \theta)$ 满足

$$f_i(x, \theta) \in [0, 1], \qquad \sum_{i=1}^{C} f_i(x, \theta) = 1 \tag{54}$$

$f_y(x, \theta)$ 可以看作真实类别 y 的似然函数。参数可以直接用最大似然估计来优化。考虑到计算问题，我们经常使用最小化负对数似然，也就是**负对数似然损失函数**（Negative Log Likelihood function）。

$$L(y, f(x, \theta)) = -\log f_y(x, \theta). \tag{55}$$

损失函数

如果我们用 one-hot 向量 $\mathbf{y}$ 来表示目标类别 c，其中只有 $y_c = 1$，其余的向量元素都为 0。

负对数似然函数也可以写为：

$$L(y, f(x, \theta)) = -\sum_{i=1}^{C} y_i \log f_i(x, \theta). \tag{56}$$

y_i 也也可以看成是真实类别的分布，这样公式56恰好是交叉熵的形式。因此，负对数似然损失函数也常叫做**交叉熵损失函数**（Cross Entropy Loss function）。

损失函数

Hinge 损失函数 对于两类分类问题，假设 y 和 $f(x, \theta)$ 的取值为 $\{-1, +1\}$。Hinge 损失函数（Hinge Loss Function）的定义如下：

$$L(y, f(x, \theta)) = \max(0, 1 - yf(x, \theta)) \tag{57}$$

$$= |1 - yf(x, \theta)|_+. \tag{58}$$

机器学习算法类型：有监督学习

有监督学习是利用一组已知输入 x 和输出 y 的数据来学习模型的参数，使得模型预测的输出标记和真实标记尽可能的一致。有监督学习根据输出类型又可以分为**回归**和**分类**两类。

回归（Regression） 如果输出 y 是连续值（实数或连续整数），$f(x)$ 的输出也是连续值。这种类型的问题就是回归问题。对于所有已知或未知的 (x, y)，使得 $f(x, \theta)$ 和 y 尽可能地一致。损失函数通常定义为平方误差。

分类（Classification） 如果输出 y 是离散的类别标记（符号），就是分类问题。损失函数有一般用 0-1 损失函数或负对数似然函数等。在分类问题中，通过学习得到的决策函数 $f(x, \theta)$ 也叫**分类器**。

机器学习算法的类型：无监督学习

无监督学习（Unsupervised Learning） 无监督学习是用来学习的数据不包含输出目标，需要学习算法自动学习到一些有价值的信息。一个典型的无监督学习问题就是**聚类**（Clustering）。

增强学习（Reinforcement Learning） 增强学习也叫强化学习，强调如何基于环境做出一系列的动作，以取得最大化的累积收益。每做出一个动作，并不一定立刻得到收益。增强学习和有监督学习的不同在于增强学习不需要显式地以输入/输出对的方式给出训练样本，是一种在线的学习机制。

参数估计

在机器学习问题中，我们需要学习到参数θ，使得风险函数最小化。

$$\theta^* = \arg\min_{\theta} \mathcal{R}(\theta_t) \tag{59}$$

$$= \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(y^{(i)}, f(x^{(i)}, \theta)\right). \tag{60}$$

如果用梯度下降法进行参数学习，

$$\mathbf{a}_{t+1} = \mathbf{a}_t - \lambda \frac{\partial \mathcal{R}(\theta)}{\partial \theta_t} \tag{61}$$

$$= \mathbf{a}_t - \lambda \sum_{i=1}^{N} \frac{\partial \mathcal{R}\left(\theta_t; x^{(i)}, y^{(i)}\right)}{\partial \theta}, \tag{62}$$

搜索步长λ在机器学习中也叫作**学习率**（Learning Rate）。

梯度下降法

梯度下降是求得所有样本上的风险函数最小值，叫做**批量梯度下降法**。若样本个数 N 很大，输入 $\mathbf{x}$ 的维数也很大时，那么批量梯度下降法每次迭代要处理所有的样本，效率会较低。为此，有一种改进的方法即**随机梯度下降法**。

随机梯度下降法（Stochastic Gradient Descent，SGD）也叫**增量梯度下降**，每个样本都进行更新

$$\mathbf{a}_{t+1} = \mathbf{a}_t - \lambda \frac{\partial \mathcal{R}\left(\theta_t; x^{(t)}, y^{(t)}\right)}{\partial \theta}, \tag{63}$$

$x^{(t)}, y^{(t)}$ 是第 t 次迭代选取的样本。

Early-Stop

在梯度下降训练的过程中，由于过拟合的原因，在训练样本上收敛的参数，并不一定在测试集上最优。因此，我们使用一个**验证集**（Validation Dataset）（也叫**开发集**（Development Dataset））来测试每一次迭代的参数在验证集上是否最优。如果在验证集上的错误率不再下降，就停止迭代。这种策略叫Early-Stop。如果没有验证集，可以在训练集上进行**交叉验证**。

学习率设置：动量法

动量法（Momentum Method）[Rumelhart et al., 1988] 对当前迭代的更新中加入上一次迭代的更新。我们记 $\nabla\theta_t = \theta_t - \theta_{t-1}$。在第 t 迭代时，

$$\theta_t = \theta_{t-1} + (\rho\nabla\theta_t - \lambda g_t), \tag{64}$$

其中，ρ 为动量因子，通常设为 0.9。这样，在迭代初期，使用前一次的梯度进行加速。在迭代后期的收敛值附近，因为两次更新方向基本相反，增加稳定性。

学习率设置：AdaGrad

AdaGrad（Adaptive Gradient）算法 [Duchi et al., 2011] 是借鉴 L2 正则化的思想。在第 t 迭代时，

$$\theta_t = \theta_{t-1} - \frac{\rho}{\sqrt{\sum_{\tau=1}^{t} g_\tau^2}} g_t, \tag{65}$$

其中，ρ 是初始的学习率，$g_\tau \in \mathbb{R}^{|\theta|}$ 是第 τ 次迭代时的梯度。

随着迭代次数的增加，梯度逐渐缩小。

学习率设置：AdaDelta

AdaDelta 算法 [Zeiler, 2012] 用指数衰减的移动平均来累积历史的梯度信息。第 t 次迭代的梯度的期望 $E(g^2)_t$ 为：

$$E(g^2)_t = \rho E(g^2)_{t-1} + (1-\rho) g_t^2, \tag{66}$$

其中，ρ 是衰减常数。

本次迭代的更新为

$$\nabla\theta_t = -\frac{\sqrt{E(\nabla\theta^2)_{t-1} + \epsilon}}{\sqrt{E(g^2)_t + \epsilon}} g_t \tag{67}$$

其中，$E(\nabla\theta^2)_t$ 为前一次迭代时 $\nabla\theta^2$ 的移动平均，ϵ 为常数。

累计更新本次迭代 $\nabla\theta^2$ 的移动平均

$$E(\nabla\theta^2)_t = \rho E(\nabla\theta^2)_{t-1} + (1-\rho)\nabla\theta_t^2. \tag{68}$$

学习率设置：AdaDelta

最后更新参数

$$\theta_t = \theta_{t-1} + \nabla\theta_t. \tag{69}$$

线性回归

如果输入 $\mathbf{x}$ 是列向量，目标 y 是连续值（实数或连续整数），预测函数 $f(\mathbf{x})$ 的输出也是连续值。这种机器学习问题是回归问题。

如果我们定义 $f(\mathbf{x})$ 是线性函数，

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b, \tag{70}$$

这就是线性回归问题（Linear Regression）。

为了简单起见，我们将公式70写为

$$f(\mathbf{x}) = \hat{\mathbf{w}}^{\mathrm{T}}\hat{\mathbf{x}}, \tag{71}$$

其中 $\hat{\mathbf{w}}$ 和 $\hat{\mathbf{x}}$ 分别称为**增广权重向量**和**增广特征向量**。

平方损失函数

线性回归的损失函数通常定义为平方损失函数。

$$L(y, f(x, \mathbf{w})) = \|y - f(x, \mathbf{w})\|^2. \tag{72}$$

给定 N 给样本 $(x^{(i)}, y^{(i)}), 1 \leq i \leq N$，模型的经验风险为

$$R(Y, f(X, \mathbf{w})) = \sum_{i=1}^{N} L(y^{(i)}, f(\mathbf{x}^{(i)}, \mathbf{w})) \tag{73}$$

$$= \sum_{i=1}^{N} \|\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)} - y^{(i)}\|^2 \tag{74}$$

$$= \|X^{\mathrm{T}}\mathbf{w} - \mathbf{y}\|^2, \tag{75}$$

平方损失函数

其中，**y** 是一个目标值 $y^{(i)}, \cdots, y^{(N)}$ 的列向量，X 是所有输入组成的矩阵：

$$X = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(N)} \\ \vdots & \vdots & \ddots & \vdots \\ x_k^{(1)} & x_k^{(2)} & \cdots & x_k^{(N)} \end{pmatrix} \tag{76}$$

最小二乘法估计

要最小化 $R(Y, f(X, \mathbf{w}))$，我们要计算 $R(Y, f(X, \mathbf{w}))$ 对 $\mathbf{w}$ 的导数

$$\frac{\partial \mathcal{R}(Y, f(X, \mathbf{w}))}{\partial \mathbf{w}} = \frac{\partial \|X^{\mathrm{T}}\mathbf{w} - \mathbf{y}\|^2}{\partial \mathbf{w}} \tag{77}$$

$$= X(X^{\mathrm{T}}\mathbf{w} - \mathbf{y}). \tag{78}$$

让 $\frac{\partial \mathcal{R}(Y, f(X, \mathbf{w}))}{\partial \mathbf{w}} = 0$，则可以得到

$$\mathbf{w} = (XX^{\mathrm{T}})^{-1}X\mathbf{y} \tag{79}$$

$$= \Big(\sum_{i=1}^{N} \mathbf{x}^{(i)}(\mathbf{x}^{(i)})^{\mathrm{T}}\Big)^{-1}\Big(\sum_{i=1}^{N} \mathbf{x}^{(i)}y^{(i)}\Big). \tag{80}$$

这里要求 XX^T 是满秩的，存在逆矩阵，也就是要求 $\mathbf{x}$ 的每一维之间是非线性相关的。这样的参数求解方法也叫**最小二乘法估计**。

梯度下降法

就需要用梯度下降法来求解。初始化 $\mathbf{w}_0 = 0$，迭代公式为：

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \lambda \frac{\partial \mathcal{R}\big(Y, f(X, \mathbf{w})\big)}{\partial \mathbf{w}}, \tag{81}$$

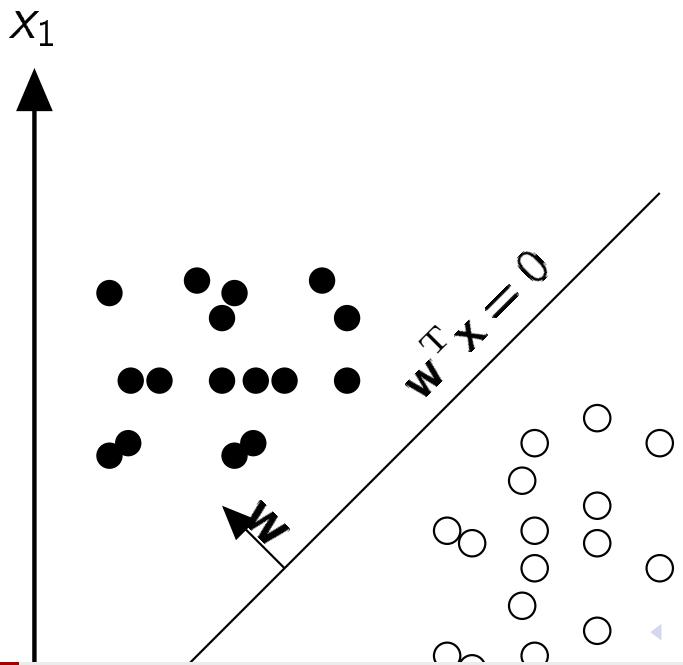
其中 λ 是学习率。

线性分类

线性分类是机器学习中最常见并且应用最广泛的一种分类器。

首先对于两类分类问题，假设类别 $y \in \{0, 1\}$，线性分类函数为

$$\hat{y} = \begin{cases} 1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x} > 0 \\ 0 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x} \leq 0 \end{cases} = I(\mathbf{w}^{\mathrm{T}}\mathbf{x} > 0), \tag{82}$$



Logistic 回归

线性分类函数的参数 $\mathbf{w}$ 有很多种学习方式，比如感知器。这里使用另一种常用的学习算法：**Logistic 回归**。

我们定义目标类别 $y = 1$ 的后验概率为：

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x})}, \tag{83}$$

其中，$\sigma(\cdot)$ 为 logistic 函数，$\mathbf{x}$ 和 $\mathbf{w}$ 为增广的输入向量和权重向量。

$y = 0$ 的后验概率为 $P(y = 0|\mathbf{x}) = 1 - P(y = 1|\mathbf{x})$。

Logistic 回归

给定 N 给样本 $(x^{(i)}, y^{(i)}), 1 \leq i \leq N$，我们使用交叉熵损失函数，模型在训练集的风险函数为：

$$\mathcal{J}(\mathbf{w}) = -\sum_{i=1}^{N} \left(y^{(i)} \log \left(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}) \right) + (1 - y^{(i)}) \log \left(1 - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}) \right) \right) \tag{84}$$

$$= -\sum_{i=1}^{N} \left(y^{(i)} \log \left(\frac{1}{1 + \exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})} \right) + (1 - y^{(i)}) \log \left(1 - \frac{1}{1 + \exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})} \right) \right) \tag{85}$$

$$= -\sum_{i=1}^{N} \left(y^{(i)} \log \left(\frac{\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}{1 + \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})} \right) + (1 - y^{(i)}) \log \left(\frac{1}{1 + \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})} \right) \right) \tag{86}$$

$$= -\sum_{i=1}^{N} \left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)} y^{(i)} - \log \left(1 + \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}) \right) \right) \tag{87}$$

梯度下降法

采样梯度下降法，$\mathcal{J}(\mathbf{w})$ 关于 $\mathbf{w}$ 的梯度为：

$$\frac{\partial \mathcal{J}(\mathbf{w})}{\partial \mathbf{w}} = -\sum_{i=1}^{N}\left(\mathbf{x}^{(i)}y^{(i)} - \mathbf{x}^{(i)} \cdot \exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}) \cdot \frac{1}{1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right) \tag{88}$$

$$= -\sum_{i=1}^{N}\left(\mathbf{x}^{(i)}y^{(i)} - \mathbf{x}^{(i)} \cdot \frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right) \tag{89}$$

$$= -\sum_{i=1}^{N}\left(\mathbf{x}^{(i)} \cdot \left(y^{(i)} - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\right)\right) \tag{90}$$

$$= \sum_{i=1}^{N}\left(\mathbf{x}^{(i)} \cdot \left(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}) - y^{(i)}\right)\right) \tag{91}$$

我们可以初始化 $\mathbf{w}_0 = 0$，然后用梯度下降法进行更新

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \lambda \frac{\partial \mathcal{R}(\mathbf{w})}{\partial \mathbf{w}}, \tag{92}$$

多类线性分类

对于多类分类问题（假设类别数为 $C(C > 2)$），一般有两种多类转两类的转换方式：

1. 把多类分类问题转换为 C 个两类分类问题，构建 C 个一对多的分类器。每个两类分类问题都是把某一类和其他类用一个超平面分开。
2. 把多类分类问题转换为 $C(C-1)/2$ 个两类分类问题，构建 $C(C-1)/2$ 个两两分类器。每个两类分类问题都是把 C 类中某两类用一个超平面分开。

缺陷：空间中的存在一些区域，这些区域中点的类别是不能区分确定的。

多类线性分类

为了避免上述缺陷，可以使用一个更加有效的决策规则，直接建立多类线性分类器。假设 $y = \{1, \cdots, C\}$ 共 C 个类别，首先定义 C 个判别函数：

$$f_c(\mathbf{x}) = \mathbf{w}_c^{\mathrm{T}}\mathbf{x}, \qquad c = 1, \cdots, C, \tag{93}$$

这里 $\mathbf{w}_c$ 为类 c 的权重向量。

这样，对于空间中的一个点 $\mathbf{x}$，如果存在类别 c，对于所有的其他类别 $\tilde{c}(\mathbf{w}_c^T\mathbf{x} \neq c)$ 都满足 $\mathbf{f}_c(\mathbf{x}) > \mathbf{f}_{\tilde{c}}(\mathbf{x})$，那么 $\mathbf{x}$ 属于类别 c。相应的分类函数可以表示为：

$$\hat{y} = \arg\max_{c=1}^{C} \mathbf{w}_c^{\mathrm{T}}\mathbf{x} \tag{94}$$

SoftMax 回归

SoftMax 回归是 Logistic 回归的多类推广。

在 SoftMax 回归中，机器学习模型预测目标为每一个类别的后验概率。

利用 **softmax** 函数，我们定义目标类别 $y = c$ 的后验概率为：

$$P(y = c|\mathbf{x}) = \mathbf{softmax}(\mathbf{w}_c^{\mathrm{T}}\mathbf{x}) = \frac{\mathbf{w}_c^{\mathrm{T}}\mathbf{x}}{\exp(\sum_{i=1}^{C}\mathbf{w}_i^{\mathrm{T}}\mathbf{x})}. \quad (95)$$

对于样本 $(\mathbf{x}, y)$，输出目标 $y = \{1, \cdots, C\}$，我们用 C 维的 one-hot 向量 $\mathbf{y}$ 来表示输出目标。对于类别 c，

$$\mathbf{y} = [I(1 = c), I(2 = c), \cdots, I(C = c)]^{\mathrm{T}}, \quad (96)$$

这里，$I()$ 是指示函数。

SoftMax 回归

我们将公式95重新定义一下，直接输出 k 维向量。

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{softmax}(W^{\mathrm{T}}\mathbf{x}) \\ &= \frac{\exp(W^{\mathrm{T}}\mathbf{x})}{\mathbf{1}^{\mathrm{T}}\exp\left((W^{\mathrm{T}}\mathbf{x})\right)} \\ &= \frac{\exp(\mathbf{z})}{\mathbf{1}^{\mathrm{T}}\exp\left(\mathbf{z}\right)},\end{aligned} \tag{97}$$

其中，$W = [\mathbf{w}_1, \cdots, \mathbf{w}_C]$ 是 C 个类对应权重向量组成的矩阵。$\hat{\mathbf{y}}$ 的第 c 维的值是第 c 类的预测后验概率。其中，$\hat{\mathbf{z}} = W^{\mathrm{T}}\mathbf{x}$，为 **softmax** 函数的输入向量。

SoftMax 回归

给定 N 给样本 $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}), 1 \leq i \leq N$，我们使用交叉熵损失函数，模型在训练集的风险函数为：

$$\begin{aligned}
\mathcal{J}(W) &= -\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbf{y}_c^{(i)}\log\hat{\mathbf{y}}_c^{(i)} \\
&= -\sum_{i=1}^{N}(\mathbf{y}^{(i)})^{\mathrm{T}}\log\hat{\mathbf{y}}^{(i)} \\
&= -\sum_{i=1}^{N}(\mathbf{y}^{(i)})^{\mathrm{T}}\log\left(\mathbf{softmax}(\mathbf{z}^{(i)})\right) \\
&= -\sum_{i=1}^{N}(\mathbf{y}^{(i)})^{\mathrm{T}}\log\left(\mathbf{softmax}(W^{\mathrm{T}}\mathbf{x}^{(i)})\right). \qquad (98)
\end{aligned}$$

梯度下降法

采样梯度下降法，我们要计算 $\mathcal{J}(W)$ 关于 W 的梯度。首先，我们列下要用到的公式。

（1）softmax 函数的导数为

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial\, \mathbf{softmax}(\mathbf{x})}{\partial \mathbf{x}} \tag{99}$$

$$= \mathbf{diag}\left(\mathbf{softmax}(\mathbf{x})\right) - \mathbf{softmax}(\mathbf{x})\,\mathbf{softmax}(\mathbf{x})^{\top} \tag{100}$$

$$= \mathbf{diag}\left(\mathbf{y}\right) - \mathbf{y}\mathbf{y}^{\top}. \tag{101}$$

（2）$\mathbf{z} = W^{\mathrm{T}}\mathbf{x}$，则

$$\frac{\partial \mathbf{z}}{\partial \mathbf{w}_c} = M(\mathbf{x}, c), \tag{102}$$

$M(\mathbf{x}, c)$ 为第 c 列为 $\mathbf{x}$，其余为 0 的矩阵。

（3）$\mathbf{x}^{\top}\,\mathbf{diag}(\mathbf{x})^{-1} = \mathbf{1}_K^{\top}$

梯度下降法

采样梯度下降法，$\mathcal{J}(W)$ 关于 $\mathbf{w}_c$ 的梯度为：

$$\frac{\partial \mathcal{J}(W)}{\partial \mathbf{w}_c} = -\sum_{i=1}^{N} \frac{\left((\mathbf{y}^{(i)})^{\mathrm{T}} \log\left(\mathbf{softmax}(\hat{\mathbf{y}}^{(i)})\right)\right)}{\partial \mathbf{w}_c}$$

$$= -\sum_{i=1}^{N} \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{w}_c} \frac{\partial \mathbf{softmax}(\mathbf{z}^{(i)})}{\partial \mathbf{z}^{(i)}} \frac{\partial \log \hat{\mathbf{y}}^{(i)}}{\partial \hat{\mathbf{y}}^{(i)}} \mathbf{y}^{(i)}$$

$$= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left(\mathbf{diag}\left(\hat{\mathbf{y}}^{(i)}\right) - \hat{\mathbf{y}}^{(i)}(\hat{\mathbf{y}}^{(i)})^{\top}\right) \left(\mathbf{diag}(\hat{\mathbf{y}}^{(i)})\right)^{-1} \mathbf{y}^{(i)}$$

$$= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left(I - \hat{\mathbf{y}}^{(i)} \mathbf{1}_K^{\top}\right) \mathbf{y}^{(i)}$$

$$= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left((\mathbf{y}^{(i)}) - \hat{\mathbf{y}}^{(i)} \mathbf{1}_K^{\top} \mathbf{y}^{(i)}\right)$$

$$= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\right)$$

梯度下降法

如果采用 $y \in \{1, \cdots, C\}$ 的离散表示形式，梯度公式也可以写为：

$$\frac{\partial \mathcal{J}(W)}{\partial \mathbf{w}_c} = -\frac{1}{N} \sum_{i=1}^{C} \mathbf{x}^{(i)} \left(I(y^{(i)} = c) - p(y^{(i)} = c | \mathbf{x}^{(i)}; W) \right), \tag{104}$$

其中，$I()$ 为指示函数。

评价方法

为了衡量一个分类算法好坏，需要给定一个测试集，用分类器对测试集中的每一个样本进行分类，并根据分类结果计算评价分数。常见的评价标准有正确率、准确率、召回率和F 值等。

给定测试集 $T = (x_1, y_1), \cdots, (x_N, y_N)$，对于所有的 $y_i \in \{\omega_1, \cdots, \omega_C\}$。假设分类结果为 $Y = \hat{y_1}, \cdots, \hat{y_N}$。

则**正确率**（Accuracy，Correct Rate）为：

$$Acc = \frac{\sum_{i=1}^{N} |y_i = \hat{y_i}|}{N} \tag{105}$$

其中，$|\cdot|$ 为指示函数

和正确率相对应的就是**错误率**（Error Rate）。

$$\frac{\sum_{i=1}^{N} |y_i \neq \hat{y_i}|}{}$$

评价方法

在很多情况下，我们需要对每个类都进行性能估计，这就需要计算准确率和召回率。正确率和召回率是广泛用于信息检索和统计学分类领域的两个度量值，在机器学习的评价中也被大量使用。

准确率（Precision，P），也叫查准率，精确率或精度，是识别出的个体总数中正确识别的个体总数的比例。对于类 c 来说，

$$P_c = \frac{\sum_{\substack{i=1 \\ \hat{y}_i=c}}^{N} |y_i = \hat{y}_i|}{\sum_{\substack{i=1 \\ \hat{y}_i=c}}^{N} 1} \tag{107}$$

召回率（Recall，R），也叫查全率，是测试集中存在的个体总数中正确识别的个体总数的比例。

$$\sum^{N} |y_i = \hat{y}_i|$$

感知器

在介绍人工神经网络之前，我们先来介绍下最简单的神经网络：感知器。

感知器，也是最简单的神经网络（只有一层）。感知器是由美国计算机科学家 Roseblatt 于 1957 年提出的。

感知器可谓是最简单的人工神经网络，只有一个神经元。

感知器也可以看出是线性分类器的一个经典学习算法。

生物神经细胞

大脑和神经元的结构

生物神经细胞

感知器是对生物神经细胞的简单数学模拟。神经细胞也叫神经元（neuron），结构大致可分为细胞体和细胞突起。

- **细胞体**（Soma）中的神经细胞膜上有各种受体和离子通道，胞膜的受体可与相应的化学物质神经递质结合，引起离子通透性及膜内外电位差发生改变，产生相应的生理活动：兴奋或抑制。
- 细胞突起是由细胞体延伸出来的细长部分，又可分为树突和轴突。
  - **树突**（Dendrite）可以接受刺激并将兴奋传入细胞体。每个神经元可以有一或多个树突。
  - **轴突** (Axons) 可以把兴奋从胞体传送到另一个神经元或其他组织。每个神经元只有一个轴突。

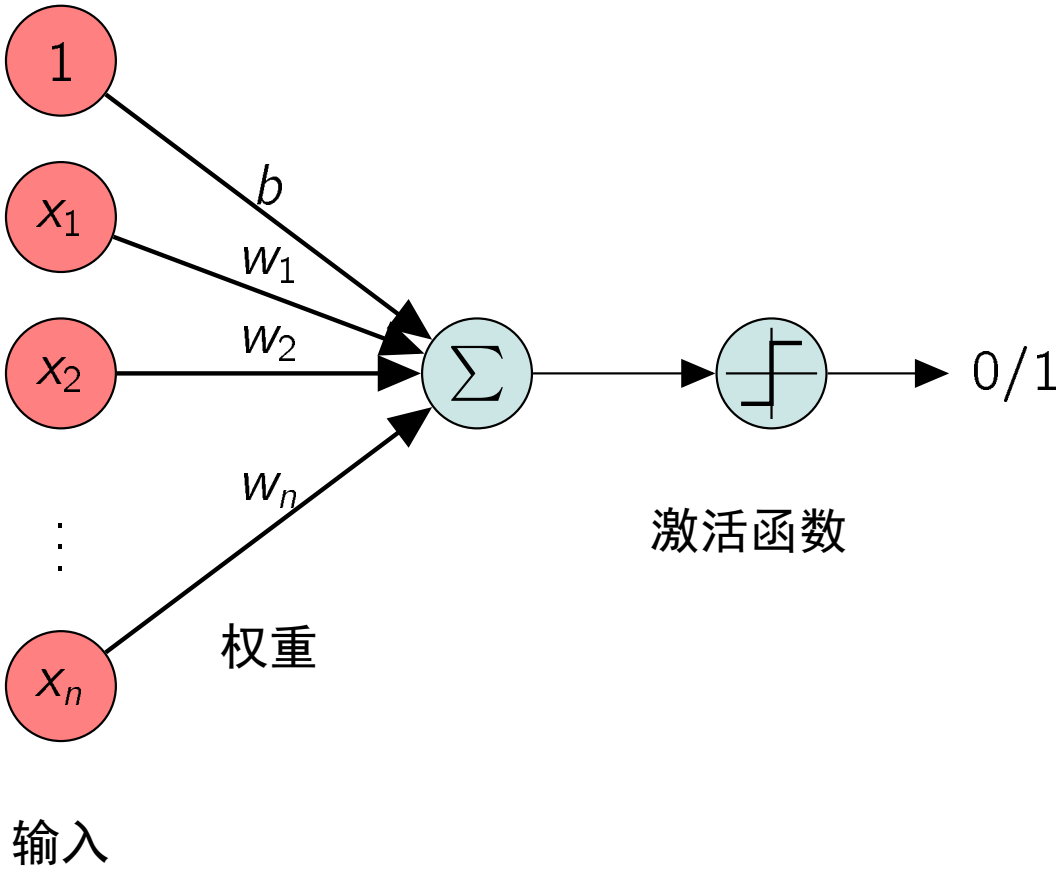
生物神经细胞

两个神经元之间或神经元与效应器细胞之间信息传递靠**突触**（Synapse）完成。突触是一个神经元的冲动传到另一个神经元或传到另一细胞间的相互接触的结构。

单个神经细胞可被视为一种只有两种状态的机器——兴奋和抑制。神经细胞的状态取决于从其它的神经细胞收到的输入信号量，及突触的强度（抑制或加强）。当信号量总和超过了某个阈值时，细胞体就会兴奋，产生电脉冲。电脉冲沿着轴突并通过突触传递到其它神经元。

感知器模型

感知器是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 0 或 1。



感知器

给定一个 n 维的输入 $\mathbf{x} = (x_1, x_2, \cdots, x_n)$，

$$\hat{y} = \begin{cases} +1 & \text{当} \mathbf{w}^T\mathbf{x} + b > 0 \\ -1 & \text{当} \mathbf{w}^T\mathbf{x} + b \leq 0 \end{cases}, \tag{113}$$

其中，$\mathbf{w}$ 是 n 维的权重向量，b 是偏置。$\mathbf{w}$ 和 b 是未知的，需要从给定的训练数据集中学习得到。

不失一般性，我们使用增广的输入和权重向量，公式113可以简写为：

$$\hat{y} = \begin{cases} +1 & \text{当} \mathbf{w}^T\mathbf{x} > 0 \\ -1 & \text{当} \mathbf{w}^T\mathbf{x} \leq 0 \end{cases}, \tag{114}$$

两类感知器算法 Rosenblatt [1958]

输入: 训练集: $(\mathbf{x}_i, y_i), i = 1, \cdots, N$，迭代次数： T

输出: $\mathbf{w}$

初始化： $\mathbf{w}_0 = 0$;

$k = 0$;

for $t = 1 \cdots T$ **do**

  for $i = 1 \cdots N$ **do**

    选取一个样本 $(\mathbf{x}_i, y_i)$， **if** $\mathbf{w}^T(y_i \mathbf{x}_i) < 0$ **then**

      $\mathbf{w}_{k+1} = \mathbf{w}_k + y_i \mathbf{x}_i$， ;

      $k = k + 1$;

    end

  end

end

return $\mathbf{w}_k$;

收敛性

Novikoff [1963] 证明对于两类问题，如果训练集是线性可分的，那么感知器算法可以在有限次迭代后收敛。然而，如果训练集不是线性分隔的，那么这个算法则不能确保会收敛。

感知器收敛性

对于任何线性可分的训练集 $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$，假设 R 是所有样本中输入向量的模的最大值。

$$R = \max_i \|x_i\|$$

那么在感知器学习算法中，总共的预测错误次数 $K < \frac{R^2}{\gamma^2}$。

大纲

人工神经网络

人工神经网络模型主要考虑网络连接的拓扑结构、神经元的特征、学习规则等。目前，已有近 40 种神经网络模型。

- 前馈神经网络：也经常称为**多层感知器**（Multilayer Perceptron，MLP）。
- 反馈神经网络：网络内神经元间有反馈，可以用一个无向的完备图表示。这种神经网络的信息处理是状态的变换，可以用动力学系统理论处理。

神经元

人工神经元使用一个非线性的激活函数，输出一个活性值。假定神经元接受 n 个输入 $\mathbf{x} = (x_1, x_2, \cdots, x_n)$，用状态 z 表示一个神经元所获得的输入信号 x 的加权和，输出为该神经元的活性值 a。具体定于如下：

$$z = \mathbf{w}^\top \mathbf{x} + b \tag{115}$$

$$a = f(z) \tag{116}$$

其中，$\mathbf{w}$ 是 n 维的权重向量，b 是偏置。典型的激活函数 f 有 sigmoid 型函数、非线性斜面函数等。

神经元示例

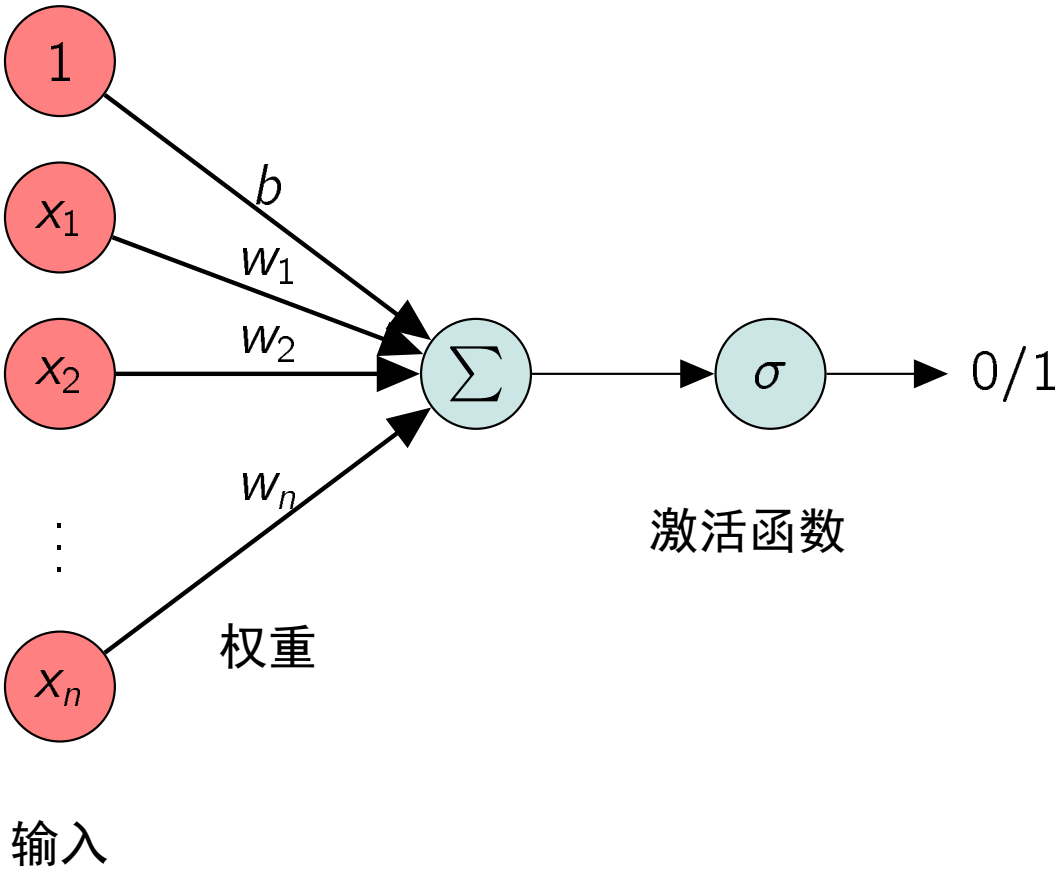


图: 人工神经元模型

激活函数

传统神经网络中最常用的激活函数分别是**sigmoid 型函数**。sigmoid 型函数是指一类 S 型曲线函数，常用的 sigmoid 型函数有 logistic 函数 $\sigma(x)$ 和 tanh 函数。

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{117}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{118}$$

tanh 函数可以看作是放大并平移的 logistic 函数：$\tanh(x) = 2\sigma(2x) - 1$。

激活函数

rectifier 函数 [Glorot et al., 2011]，定义为

$$\textbf{rectifier}(x) = \max(0, x) \tag{119}$$

rectifier 函数被认为有生物上的解释性。神经科学家发现神经元具有单侧抑制、宽兴奋边界、稀疏激活性等特性。

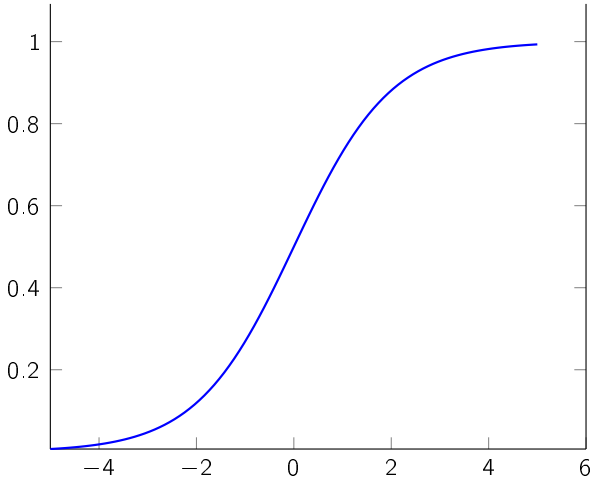
采用 rectifier 函数的单元也叫作**修正线性单元**（rectified linear unit，ReLU）[Nair and Hinton, 2010]。

softplus 函数 [Dugas et al., 2001]，定义为

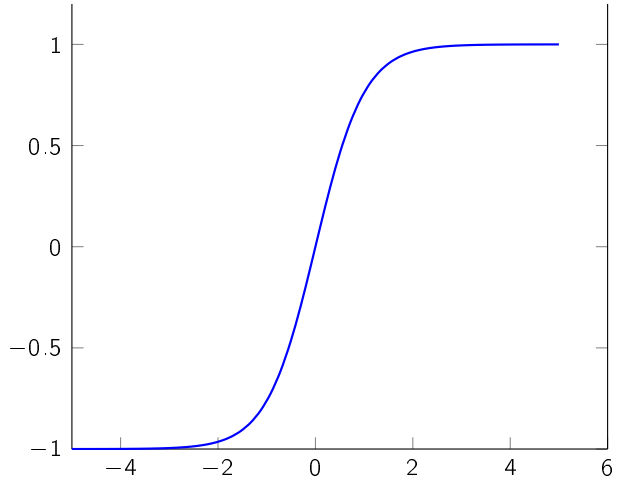
$$\textbf{softplus}(x) = \log(1 + e^x) \tag{120}$$

softplus 函数可以看作是 rectifier 函数的平滑版本，其导数刚好是 logistic 函数。softplus 虽然也有具有单侧抑制、宽兴奋边界的特性，却没有稀疏激活性。

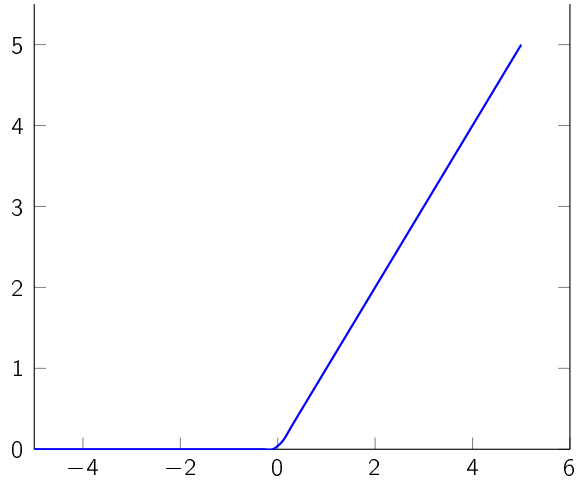
激活函数



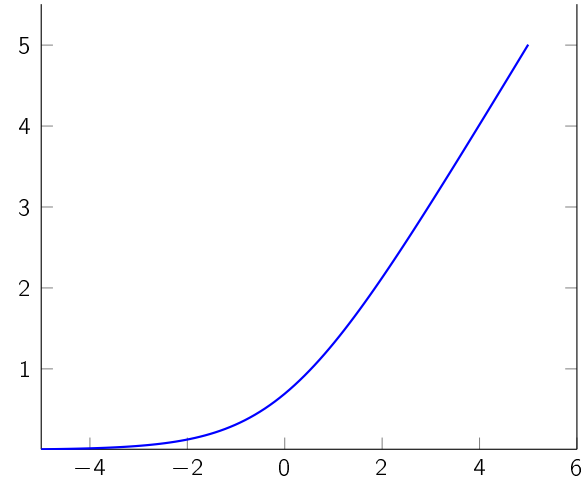
(a) logistic 函数



(b) tanh 函数



(c) rectifier 函数



(d) softplus 函数

前馈神经网络

在前馈神经网络中，各神经元分别属于不同的层。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。

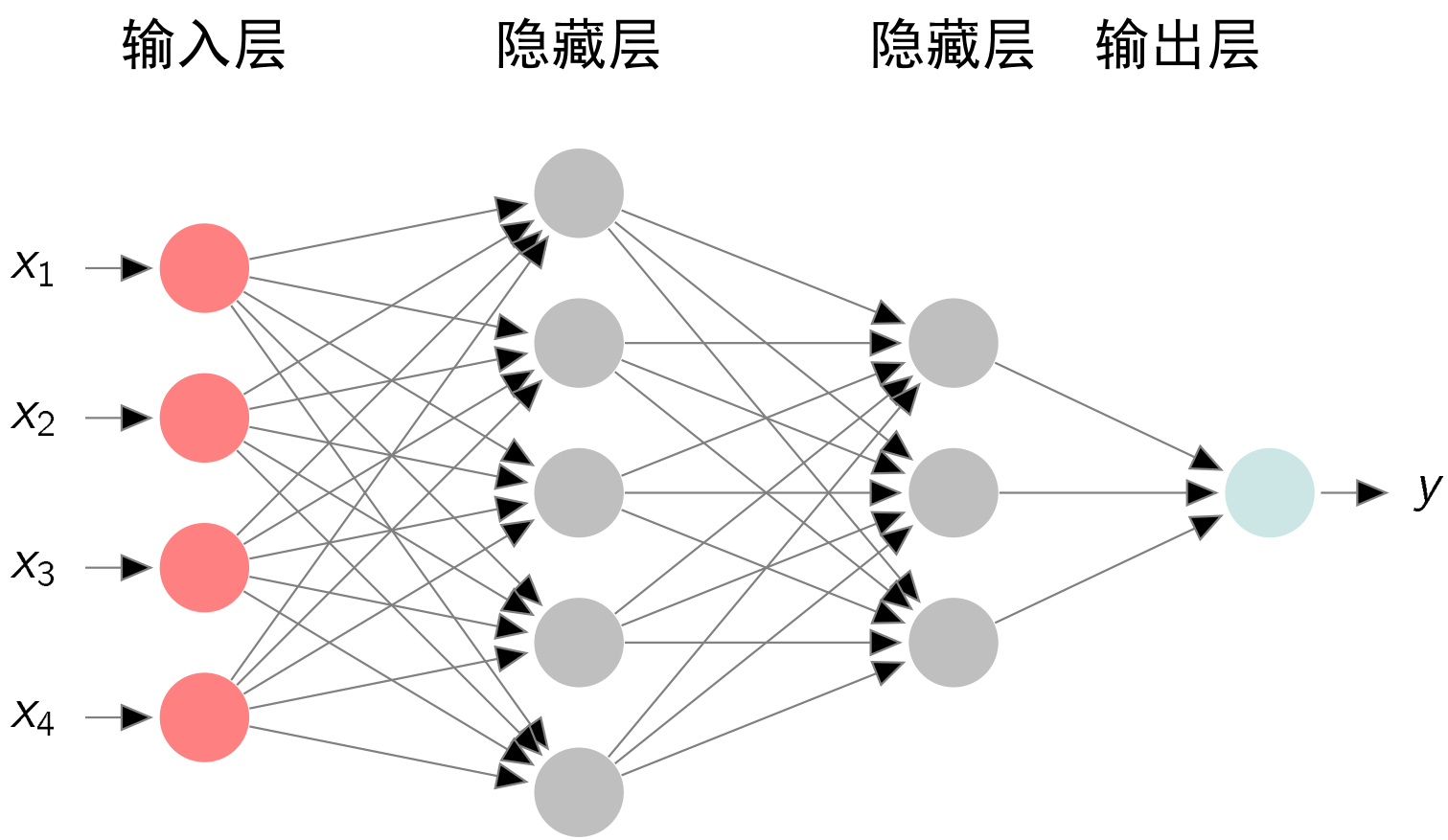


图 多层神经网络

前馈计算

给定一个前馈神经网络，我们用下面的记号来描述这样网络。

- L：表示神经网络的层数；
- n^l：表示第 l 层神经元的个数；
- $f_l(\cdot)$：表示 l 层神经元的激活函数；
- $W^{(l)} \in \mathbb{R}^{n^l \times n^{l-1}}$：表示 $l-1$ 层到第 l 层的权重矩阵；
- $\mathbf{b}^{(l)} \in \mathbb{R}^{n^l}$：表示 $l-1$ 层到第 l 层的偏置；
- $\mathbf{z}^{(l)} \in \mathbb{R}^{n^l}$：表示 l 层神经元的状态；
- $\mathbf{a}^{(l)} \in \mathbb{R}^{n^l}$：表示 l 层神经元的活性值。

前馈神经网络通过下面公式进行信息传播。

$$\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \tag{121}$$

$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)}) \tag{122}$$

前馈计算

公式121和122也可以合并写为：

$$\mathbf{z}^{(l)} = W^{(l)} \cdot f_l(\mathbf{z}^{(l-1)}) + \mathbf{b}^{(l)} \tag{123}$$

这样，前馈神经网络可以通过逐层的信息传递，得到网络最后的输出 $\mathbf{a}^L$。

$$\mathbf{x} = \mathbf{a}^{(0)} \to \mathbf{z}^{(1)} \to \mathbf{a}^{(1)} \to \mathbf{z}^{(2)} \to \cdots \to \mathbf{a}^{(L-1)} \to \mathbf{z}^{(L)} \to \mathbf{a}^{(L)} = y \tag{124}$$

将前馈网络应用于机器学习

给定一组样本 $(\mathbf{x}^{(i)}, y^{(i)}), 1 \leq i \leq N$，用前馈神经网络的输出为 $f(\mathbf{x}|\mathbf{w}, \mathbf{b})$，目标函数为：

$$J(W, \mathbf{b}) = \sum_{i=1}^{N} L(y^{(i)}, f(\mathbf{x}^{(i)}|W, \mathbf{b})) + \frac{1}{2}\lambda \|W\|_F^2, \tag{125}$$

$$= \sum_{i=1}^{N} J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)}) + \frac{1}{2}\lambda \|W\|_F^2, \tag{126}$$

这里，W 和 $\mathbf{b}$ 包含了每一层的权重矩阵和偏置向量，$\|W\|_F^2 = \sum_{l=1}^{L} \sum_{j=1}^{n^{l+1}} \sum_{j=}^{n^l} W_{ij}^{(l)}$。

参数估计

我们的目标是最小化 $J(W, \mathbf{b}; \mathbf{x}, y)$。如果采用梯度下降方法，我们可以用如下方法更新参数：

$$W^{(l)} = W^{(l)} - \alpha \frac{\partial J(W, \mathbf{b})}{\partial W^{(l)}}, \tag{127}$$

$$= W^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial W^{(l)}} \right) - \lambda W, \tag{128}$$

$$\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha \frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial \mathbf{b}^{(l)}}, \tag{129}$$

$$= \mathbf{b}^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial \mathbf{b}^{(l)}} \right), \tag{130}$$

$$\tag{131}$$

这里 α 是参数的更新率。

反向传播算法

我们首先来看下 $\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W^{(l)}}$ 怎么计算。

根据公式24定义的链式法则，$\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W_{ij}^{(l)}}$ 可以写为

$$\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W_{ij}^{(l)}} = \left(\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l)}}\right)^{\top} \frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}}. \tag{132}$$

对于第 l 层，我们定义一个**误差项** $\delta^{(l)} = \frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial \mathbf{z}^{(l)}} \in \mathbb{R}^{n^{(l)}}$ 为目标函数关于第 l 层的神经元 $\mathbf{z}^{(l)}$ 的偏导数，来表示第 l 层的神经元对最终误差的影响。$\delta^{(l)}$ 也反映了最终的输出对第 l 层的神经元对最终误差的敏感程度。

反向传播算法

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

$$\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}} = \frac{\partial (W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})}{\partial W_{ij}^{(l)}} = \begin{bmatrix} 0 \\ \vdots \\ a_j^{(l-1)} \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{第 } i \text{ 行(133)}$$

因此，

$$\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial W_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)} \tag{134}$$

(135)

反向传播算法

公式132可以写为

$$\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^{\top}. \tag{136}$$

同理可得，

$$\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}. \tag{137}$$

误差项

现在我们再来看下第 l 层的**误差项** $\delta^{(l)}$ 怎么计算。

$$\delta^{(l)} \triangleq \frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l)}} \tag{138}$$

$$= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l+1)}} \tag{139}$$

$$= \mathbf{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^{\top} \cdot \delta^{(l+1)} \tag{140}$$

$$= f_l'(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^{\top} \delta^{(l+1)}), \tag{141}$$

其中 $\odot$ 是向量的点积运算符，表示每个元素相乘。

反向传播

从公式141可以看出，第 l 层的误差项可以通过第 $l+1$ 层的误差项计算得到。这就是误差的**反向传播**（Backpropagation，BP）。反向传播算法的含义是：第 l 层的一个神经元的误差项（或敏感性）是所有与该神经元相连的第 $l+1$ 层的神经元的误差项的权重和。然后，在乘上该神经元激活函数的梯度。

在计算出每一层的误差项之后，我们就可以得到每一层参数的梯度。因此，前馈神经网络的训练过程可以分为以下三步：（1）先前馈计算每一层的状态和激活值，直到最后一层；（2）反向传播计算每一层的误差；（3）计算每一层参数的偏导数，并更新参数。具体的训练过程如算法1所示，也叫**反向传播算法**。

反向传播算法

输入: 训练集: $(\mathbf{x}^{(i)}, y^{(i)}), i = 1, \cdots, N$，最大迭代次数：$T$

输出: $W, \mathbf{b}$

初始化 $W, \mathbf{b}$;

for $t = 1 \cdots T$ **do**

  for $i = 1 \cdots N$ **do**

    (1) 前馈计算每一层的状态和激活值，直到最后一层;

    (2) 用公式141反向传播计算每一层的误差 $\delta^{(l)}$;

    (3) 用公式136和137每一层参数的导数;

$$\frac{\partial J(W,\mathbf{b};\mathbf{x}^{(i)},y^{(i)})}{\partial W^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^{\top};$$

$$\frac{\partial J(W,\mathbf{b};\mathbf{x}^{(i)},y^{(i)})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)};$$

    (4) 更新参数;

$$W^{(l)} = W^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W,\mathbf{b};\mathbf{x}^{(i)},y^{(i)})}{\partial W^{(l)}} \right) - \lambda W;$$

$$\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha \sum_{i=1}^{N} \left(\frac{\partial J(W,\mathbf{b};\mathbf{x}^{(i)},y^{(i)})}{\partial \mathbf{b}^{(l)}} \right);$$

梯度消失问题

在神经网络中误差反向传播的迭代公式为

$$\delta^{(l)} = f_l'(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^\top \delta^{(l+1)}, \tag{142}$$

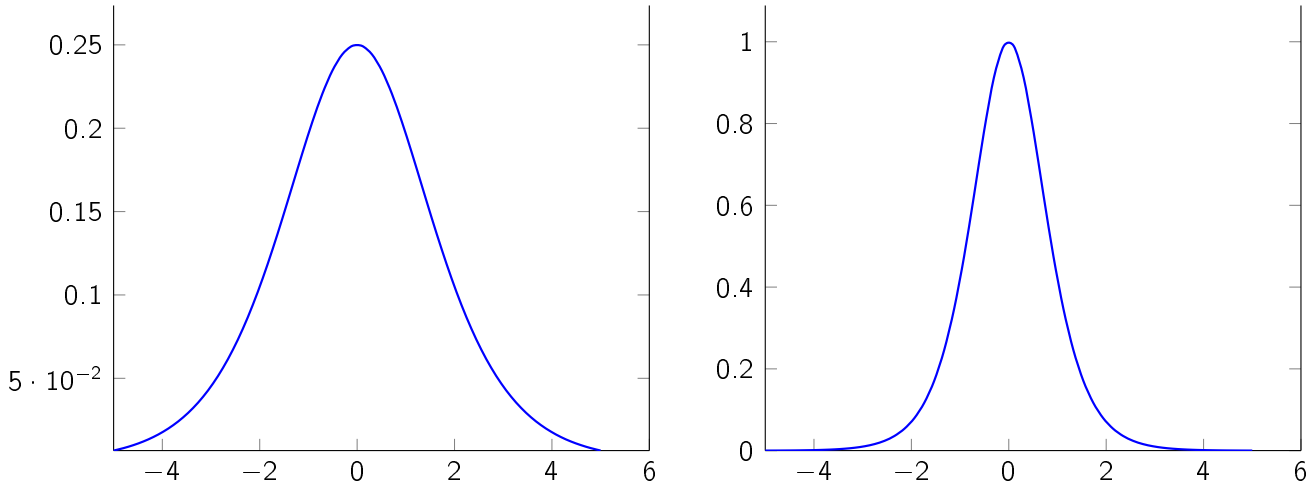
其中需要用到激活函数 f_l 的导数。误差从输出层反向传播时，在每一层都要乘以该层的激活函数的导数。

当我们使用 sigmoid 型函数：logistic 函数 $\sigma(x)$ 或 tanh 函数时，其导数为

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \in [0, 0.25] \tag{143}$$

$$\tanh'(x) = 1 - (\tanh(x))^2 \in [0, 1]. \tag{144}$$

梯度消失问题



(a) logistic 函数的导数 (b) tanh 函数的导数

图: 激活函数的导数

卷积神经网络

卷积神经网络（Convolutional Neural Networks，CNN）是一种前馈神经网络。卷积神经网络是受生物学上**感受野**（Receptive Field）的机制而提出的。感受野主要是指听觉系统、本体感觉系统和视觉系统中神经元的一些性质。比如在视觉神经系统中，一个神经元的感受野是指视网膜上的特定区域，只有这个区域内的刺激才能够激活该神经元 [Hubel and Wiesel, 1968]。

卷积神经网络有三个结构上的特性：局部连接，权重共享以及空间或时间上的次采样。这些特性使得卷积神经网络具有一定程度上的平移、缩放和扭曲不变性 [LeCun et al., 1998]。

卷积

卷积，也叫摺积，是分析数学中一种重要的运算。我们这里只考虑离散序列的情况。一维卷积经常用在信号处理中。给定一个输入信号序列 x_t，$t = 1, \cdots, n$，和滤波器 f_t，$t = 1, \cdots, m$，一般情况下滤波器的长度 m 远小于信号序列长度 n。

卷积的输出为：

$$y_t = \sum_{k=1}^{n} f_k \cdot x_{t-k+1}. \tag{145}$$

当滤波器 $f_t = 1/m$ 时，卷积相当于信号序列的移动平均。

卷积的结果按输出长度不同可以分为两类：一类是**宽卷积**，输出长度 $n + m - 1$，对于不在 $[1, n]$ 范围之外的 x_t 用零补齐（zero-padding）。一类是**窄卷积**，输出长度 $n - m + 1$，不补零。

在这里除了特别声明，我们一般说的卷积默认为**窄卷积**。

两维卷积

两维卷积经常用在图像处理中。给定一个图像 $x_{ij},\ 1 \leq i \leq M,\ 1 \leq j \leq N$，和滤波器 $f_{ij},\ 1 \leq i \leq m,\ 1 \leq j \leq n$，一般 $m << M, n << N$。

卷积的输出为：

$$y_{ij} = \sum_{u=1}^{m} \sum_{v=1}^{n} f_{uv} \cdot x_{i-u+1,j-v+1}. \tag{146}$$

在图像处理中，常用的均值滤波（mean filter）就是当前位置的像素值设为滤波器窗口中所有像素的平均值，也就是 $f_{uv} = \frac{1}{mn}$。

卷积层：用卷积来代替全连接

在全连接前馈神经网络中，如果第 l 层有 n^l 个神经元，第 $l-1$ 层有 $n^{(l-1)}$ 个神经元，连接边有 $n^{(l)} \times n^{(l-1)}$ 个，也就是权重矩阵有 $n^{(l)} \times n^{(l-1)}$ 个参数。当 m 和 n 都很大时，权重矩阵的参数非常多，训练的效率会非常低。

如果采用卷积来代替全连接，第 l 层的每一个神经元都只和第 $l-1$ 层的一个局部窗口内的神经元相连，构成一个局部连接网络。第 l 层的第 i 个神经元的输入定义为：

$$a_i^{(l)} = f(\sum_{j=1}^{m} w_j^{(l)} \cdot a_{i-j+m}^{(l-1)} + b^{(l)}), \tag{147}$$

$$= f(\mathbf{w}^{(l)} \cdot \mathbf{a}_{(i+m-1):i}^{(l-1)} + b_i), \tag{148}$$

其中，$\mathbf{w}^{(l)} \in \mathbb{R}^m$ 为 m 维的滤波器，$\mathbf{a}_{(i+m-1):i}^{(l)} = [a_{(i+m-1)}^{(l)}, \cdots, a_i^{(l)}]^T$。这里，$a^{(l)}$ 的下标从 1 开始，我们这里的卷积公式和原始的公式中 $\mathbf{a}$ 的下标有所不同。

卷积层：用卷积来代替全连接

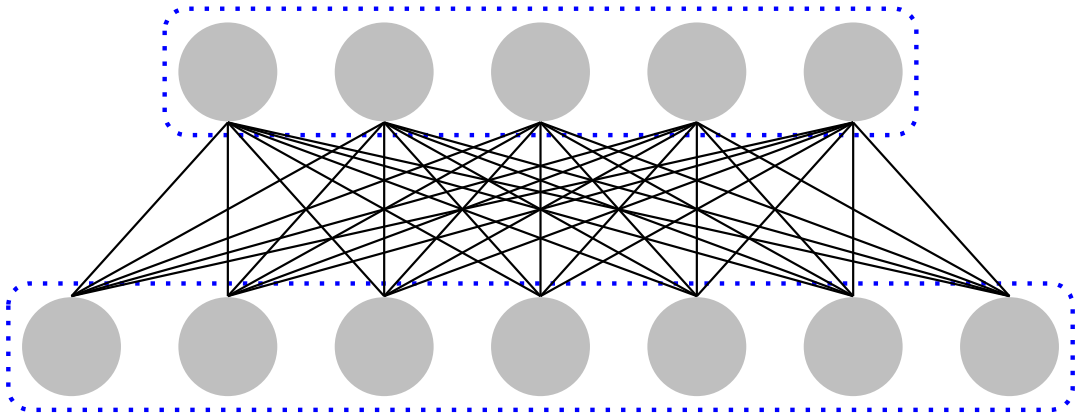
上述公式也可以写为：

$$\mathbf{a}^{(l)} = f(\mathbf{w}^{(l)} \otimes \mathbf{a}^{(l-1)} + b^{(l)}), \tag{149}$$

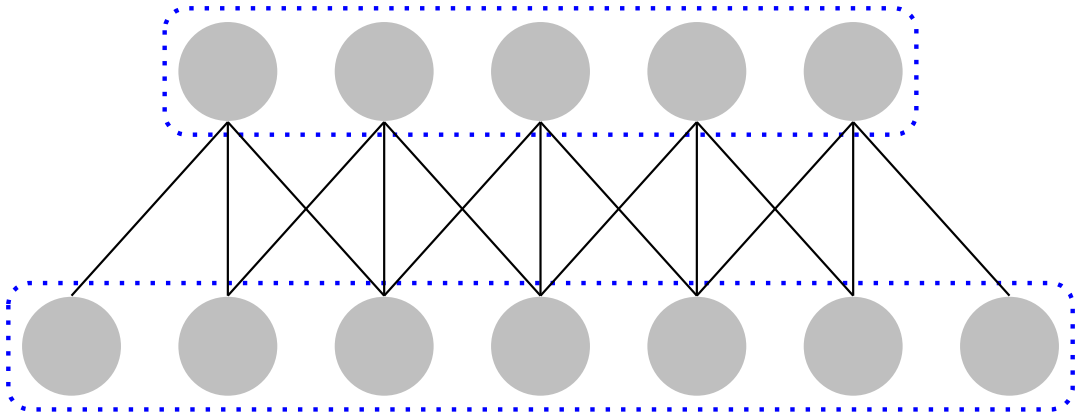
$\otimes$ 表示卷积运算。

从公式149可以看出，$\mathbf{w}^{(l)}$ 对于所有的神经元都是相同的。这也是卷积层的另外一个特性是**权值共享**。这样，在卷积层里，我们只需要 $m+1$ 个参数。另外，第 $l+1$ 层的神经元个数不是任意选择的，而是满足 $n^{(l+1)} = n^{(l)} - m + 1$。

全连接层和卷积层



(a) 全连接层



(b) 卷积层

图: 全连接层和卷积层

二维卷积层

在图像处理中，图像是以二维矩阵的形式输入到神经网络中，因此我们需要二维卷积。假设 $x^{(l)} \in \mathbb{R}^{(w_l \times h_l)}$ 和 $x^{(l-1)} \in \mathbb{R}^{(w_{l-1} \times h_{l-1})}$ 分别是第 l 层和第 $l-1$ 层的神经元活性。$X^{(l)}$ 的每一个元素为：

$$X_{s,t}^{(l)} = f\left(\sum_{i=1}^{u}\sum_{j=1}^{v} W_{i,j}^{(l)} \cdot X_{s-i+u,t-j+v}^{(l-1)} + b^{(l)}\right), \tag{150}$$

其中，$W^{(l)}\mathbb{R}^{u \times v}$ 为两维的滤波器，B 为偏置矩阵。第 $l-1$ 层的神经元个数为 $(w_l \times h_l)$，并且 $w_l = w_{l-1} - u + 1$，$h_l = h_{l-1} - v + 1$。

也可以写为：

$$X^{(l)} = f\left(W^{(l)} \otimes X^{(l-1)} + b^{(l)}\right), \tag{151}$$

特征映射

为了增强卷积层的表示能力，我们可以使用 K 个不同的滤波器来得到 K 组输出。每一组输出都共享一个滤波器。如果我们把滤波器看成一个特征提取器，每一组输出都可以看成是输入图像经过一个特征抽取后得到的特征。因此，在卷积神经网络中每一组输出也叫作一组**特征映射**（Feature Map）。

特征映射

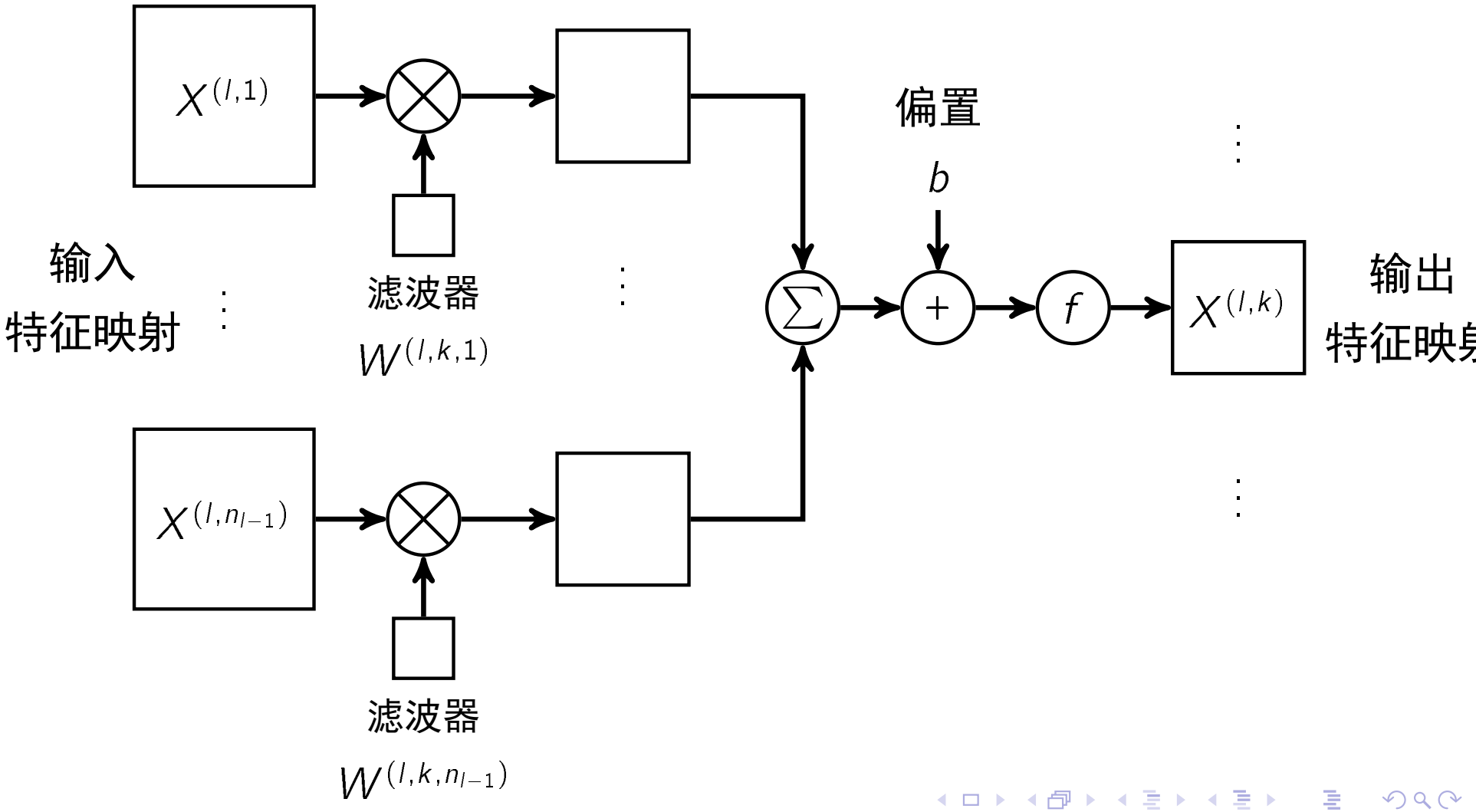
不失一般性，我们假设第 $l-1$ 层的特征映射组数为 n_{l-1}，每组特征映射的大小为 $m_{l-1} = w_{l-1} \times h_{l-1}$。第 $l-1$ 层的总神经元数：$n_{l-1} \times m_{l-1}$。第 l 层的特征映射组数为 n_l。如果假设第 l 层的每一组特征映射 $X^{(l,k)}$ 的输入为第 $l-1$ 层的所有组特征映射。

第 l 层的第 k 组特征映射 $X^{(l,k)}$ 为：

$$X^{(l,k)} = f\left(\sum_{p=1}^{n_{l-1}} \left(W^{(l,k,p)} \otimes X^{(l-1,p)}\right) + b^{(l,k)}\right), \tag{152}$$

其中，$W^{(l,k,p)}$ 表示第 $l-1$ 层的第 p 组特征向量到第 l 层的第 k 组特征映射所需的滤波器。

两维卷积层的映射关系



连接表

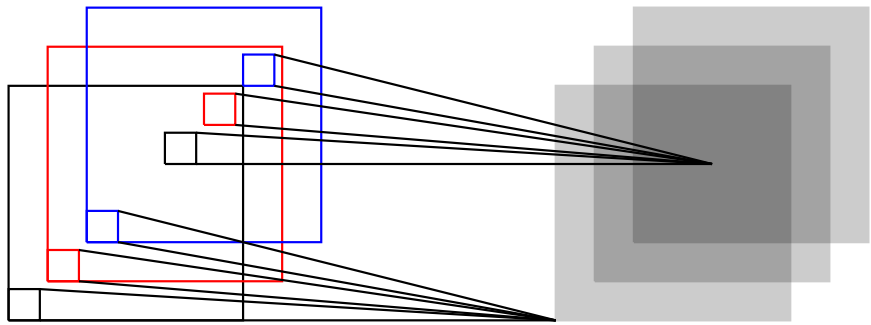
第 l 层的每一组特征映射都依赖于第 l 层的所有特征映射，相当于不同层的特征映射之间是全连接的关系。实际上，这种全连接关系不是必须的。我们可以让第 l 层的每一组特征映射都依赖于前一层的少数几组特征映射。这样，我们定义一个**连接表** T 来描述不同层的特征映射之间的连接关系。如果第 l 层的第 k 组特征映射依赖于前一层的第 p 组特征映射，则 $T_{p,k}=1$，否则为 0。

$$X^{(l,k)}=f\left(\sum_{\substack{p,\\ T_{p,k}=1}}\left(W^{(l,k,p)}\otimes X^{(l,p)}\right)+b^{(l,k)}\right) \tag{153}$$

这样，假如连接表 T 的非零个数为 K，每个滤波器的大小为 $u\times v$，那么共需要 $K\times(u\times v)+n_l$ 参数。

两维卷积层示例

卷积层的作用是提取一个局部区域的特征，每一个滤波器相当于一个特征提取器。



子采样层

卷积层虽然可以显著减少连接的个数，但是每一个特征映射的神经元个数并没有显著减少。这样，如果后面接一个分类器，分类器的输入维数依然很高，很容易出现过拟合。为了解决这个问题，在卷积神经网络一般会在卷积层之后再加上一个池化（Pooling）操作，也就是子采样（Subsampling），构成一个子采样层。子采样层可以来大大降低特征的维数，避免过拟合。

子采样层

对于卷积层得到的一个特征映射 $X^{(l)}$，我们可以将 $X^{(l)}$ 划分为很多区域 $R_k, k = 1, \cdots, K$，这些区域可以重叠，也可以不重叠。一个子采样函数 $\mathbf{down}(\cdots)$ 定义为：

$$X_k^{(l+1)} = f(Z_k^{(l+1)}), \tag{154}$$

$$= f\left(w^{(l+1)} \cdot \mathbf{down}(R_k) + b^{(l+1)}\right), \tag{155}$$

其中，$w^{(l+1)}$ 和 $b^{(l+1)}$ 分别是可训练的权重和偏置参数。

子采样层

$$X^{(l+1)} = f(Z^{(l+1)}), \tag{156}$$

$$= f\left(w^{(l+1)} \cdot \mathbf{down}(X^l) + b^{(l+1)}\right), \tag{157}$$

$\mathbf{down}(X^l)$ 是指子采样后的特征映射。

子采样函数 $\mathbf{down}(\cdot)$ 一般是取区域内所有神经元的最大值（Maximum Pooling）或平均值（Average Pooling）。

$$pool_{max}(R_k) = \max_{i \in R_k} a_i \tag{158}$$

$$pool_{avg}(R_k) = \frac{1}{|R_k|} \sum_{i \in R_k} a_i. \tag{159}$$

子采样的作用还在于可以使得下一层的神经元对一些小的形态改变保持不变性，并拥有更大的感受野。

卷积神经网络示例：LeNet-5

LeNet-5[LeCun et al., 1998]。LeNet-5 虽然提出时间比较早，但是是一个非常成功的神经网络模型。基于 LeNet-5 的手写数字识别系统在 90 年代被美国很多银行使用，用来识别支票上面的手写数字。LeNet-5 的网络结构如图6所示。

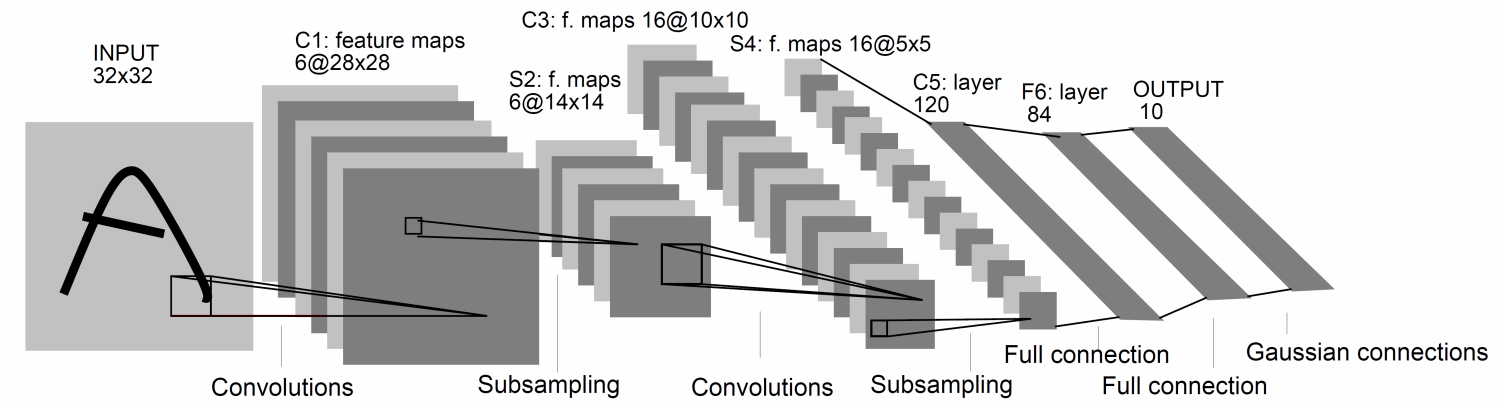


图: LeNet-5 网络结构。图片来源：[LeCun et al., 1998]

LeNet 示例

不计输入层，LeNet-5 共有 7 层，每一层的结构为：

1. 输入层：输入图像大小为 $32 \times 32 = 1024$。
2. C1 层：这一层是卷积层。滤波器的大小是 $5 \times 5 = 25$，共有 6 个滤波器。得到 6 组大小为 $28 \times 28 = 784$ 的特征映射。因此，C1 层的神经元个数为 $6 \times 784 = 4,704$。可训练参数个数为 $6 \times 25 + 6 = 156$。连接数为 $156 \times 784 = 122,304$（包括偏置在内，下同）。
3. S2 层：这一层为子采样层。由 C1 层每组特征映射中的 2×2 邻域点次采样为 1 个点，也就是 4 个数的平均。这一层的神经元个数为 $14 \times 14 = 196$。可训练参数个数为 $6 \times (1 + 1) = 12$。连接数为 $6 \times 196 \times (4 + 1) = 122,304$ （包括偏置的连接）

LeNet 示例

1. C3 层：这一层是卷积层。由于 S2 层也有多组特征映射，需要一个连接表来定义不同层特征映射之间的依赖关系。LeNet-5 的连接表如图7所示。这样的连接机制的基本假设是：C3 层的最开始的 6 个特征映射依赖于 S2 层的特征映射的每 3 个连续子集。接下来的 6 个特征映射依赖于 S2 层的特征映射的每 4 个连续子集。再接下来的 3 个特征映射依赖于 S2 层的特征映射的每 4 个不连续子集。最后一个特征映射依赖于 S2 层的所有特征映射。这样共有 60 个滤波器，大小是 $5 \times 5 = 25$。得到 16 组大小为 $10 \times 10 = 100$ 的特征映射。C3 层的神经元个数为 $16 \times 100 = 1,600$。可训练参数个数为 $(60 \times 25 + 16 = 1,516$。连接数为 $1,516 \times 100 = 151,600$。
2. S4 层：这一层是一个子采样层，由 2×2 邻域点次采样为 1 个点，得到 16 组 5×5 大小的特征映射。可训练参数个数为 $16 \times 2 = 32$。连接数为 $16 \times (4 + 1) = 2000$。

LeNet 示例

1. C5 层：是一个卷积层，得到 120 组大小为 1×1 的特征映射。每个特征映射与 S4 层的全部特征映射相连。有 $120 \times 16 = 1,920$ 个滤波器，大小是 $5 \times 5 = 25$。C5 层的神经元个数为 120，可训练参数个数为 $1,920 \times 25 + 120 = 48,120$。连接数为 $120 \times (16 \times 25 + 1) = 48,120$。
2. F6 层：是一个全连接层，有 84 个神经元，可训练参数个数为 $84 \times (120 + 1) = 10,164$。连接数和可训练参数个数相同，为 $10,164$。
3. 输出层：输出层由 10 个欧氏径向基函数（Radial Basis Function，RBF）函数组成。这里不再详述。

连接表

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

图: LeNet-5 中 C3 层的连接表。图片来源：[LeCun et al., 1998]

梯度计算

在全连接前馈神经网络中，目标函数关于第 l 层的神经元 $\mathbf{z}^{(l)}$ 的梯度为

$$\delta^{(l)} \triangleq \frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l)}} \tag{160}$$

$$= f_l'(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^T \delta^{(l+1)} \tag{161}$$

在卷积神经网络中，每一个卷积层后都接着一个子采样层，然后不断重复。因为我们需要分别来看下卷积层和子采样层的梯度。

卷积层的梯度

我们假定卷积层为 l 层，子采样层为 $l+1$ 层。因为子采样层是下采样操作，$l+1$ 层的一个神经元的误差项 δ 对应于卷积层（上一层）的相应特征映射的一个区域。l 层的第 k 个特征映射中的每个神经元都有一条边和 $l+1$ 层的第 k 个特征映射中的一个神经元相连。根据链式法则，第 l 层的一个特征映射的误差项 $\delta^{(l,k)}$，只需要将 $l+1$ 层对应特征映射的误差项 $\delta^{(l+1,k)}$ 进行上采样操作（和第 l 层的大小一样），再和 l 层特征映射的激活值偏导数逐元素相乘，再乘上权重 $w^{(l+1,k)}$），就得到了 $\delta^{(l,k)}$。

卷积层的梯度

第 l 层的第 k 个特征映射的误差项 $\delta^{(l,k)}$ 的具体推导过程如下：

$$\delta^{(l,k)} \triangleq \frac{\partial J(W, \mathbf{b}; X, y)}{\partial Z^{(l,k)}} \tag{162}$$

$$= \frac{\partial X^{(l,k)}}{\partial Z^{(l,k)}} \cdot \frac{\partial Z^{(l+1,k)}}{\partial X^{(l,k)}} \cdot \frac{\partial J(W, \mathbf{b}; X, y)}{\partial Z^{(l+1,k)}} \tag{163}$$

$$= f_l'(Z^{(l)}) \odot \left(\mathbf{up} \left(w^{(l+1,k)} \delta^{(l+1)} \right) \right) \tag{164}$$

$$= w^{(l+1,k)} \left(f_l'(Z^{(l)}) \odot \mathbf{up}(\delta^{(l+1)}) \right), \tag{165}$$

其中，**up** 为上采用函数（Upsampling）。

卷积层的梯度

在得到第 l 层的第 k 个特征映射的误差项 $\delta^{(l,k)}$，目标函数关于第 l 层的第 k 个特征映射神经元滤波器 $W_{i,j}^{(l,k,p)}$ 的梯度

$$\frac{\partial J(W,\mathbf{b};X,y)}{\partial W_{i,j}^{(l,k,p)}} = \sum_{s=1}^{w_l}\sum_{t=1}^{h_l}\left(X_{s-i+u,t-j+v}^{(l-1,p)} \cdot (\delta^{(l,k)})_{s,t}\right) \tag{166}$$

$$= \sum_{s=1}^{w_l}\sum_{t=1}^{h_l}\left(X_{(u-i)-s,(v-j)-t}^{(l-1,p)} \cdot \left(\mathbf{rot180}(\delta^{(l,k)})\right)_{s,t}\right). \tag{167}$$

卷积层的梯度

公式167也刚好是卷积形式，因此目标函数关于第 l 层的第 k 个特征映射神经元滤波器 $W^{(l,k,p)}$ 的梯度可以写为：

$$\frac{\partial J(W,\mathbf{b};X,y)}{\partial W^{(l,k,p)}} = \mathbf{rot180}\left(X^{(l-1,p)} \otimes \mathbf{rot180}(\delta^{(l,k)})\right). \tag{168}$$

目标函数关于第 l 层的第 k 个特征映射的偏置 $b^{(l)}$ 的梯度可以写为：

$$\frac{\partial J(W,\mathbf{b};X,y)}{\partial b^{(l,k)}} = \sum_{i,j}(\delta^{(l,k)})_{i,j}. \tag{169}$$

子采样层的梯度

我们假定子采样层为 l 层，$l+1$ 层为卷积层。因为子采样层是下采样操作，$l+1$ 层的一个神经元的误差项 δ 对应于卷积层（上一层）的相应特征映射的一个区域。

$$Z^{(l+1,k)} = \sum_{\substack{p, \\ T_{p,k}=1}} \left(W^{(l+1,k,p)} \otimes X^{(l,p)} \right) + b^{(l+1,k)} \tag{170}$$

第 l 层的第 k 个特征映射的误差项 $\delta^{(l,k)}$ 的具体推导过程如下：

$$\delta^{(l,k)} \triangleq \frac{\partial J(W, \mathbf{b}; X, y)}{\partial Z^{(l,k)}} \tag{171}$$

$$= \frac{\partial X^{(l,k)}}{\partial Z^{(l,k)}} \cdot \frac{\partial Z^{(l+1,k)}}{\partial X^{(l,k)}} \cdot \frac{\partial J(W, \mathbf{b}; X, y)}{\partial Z^{(l+1,k)}} \tag{172}$$

$$= f_l'(Z^{(l)}) \odot \left(\sum \left(\delta^{(l+1,p)} \tilde{\otimes} \mathbf{rot180}(W^{(l,k,p)}) \right) \right) \tag{173}$$

子采样层的梯度

公式167也刚好是卷积形式，因此目标函数关于第 l 层的第 k 个特征映射神经元滤波器 $W^{(l,k,p)}$ 的梯度可以写为：

$$\frac{\partial J(W,\mathbf{b};X,y)}{\partial w^{(l,k)}} = \sum_{i,j}\left(\mathbf{down}(X^{(l-1,k)})\cdot\delta^{(l,k)}\right)_{i,j}. \tag{174}$$

目标函数关于第 l 层的第 k 个特征映射的偏置 $b^{(l)}$ 的梯度可以写为：

$$\frac{\partial J(W,\mathbf{b};X,y)}{\partial b^{(l,k)}} = \sum_{i,j}(\delta^{(l,k)})_{i,j}. \tag{175}$$

循环神经网络

前馈神经网络的输入和输出的维数都是固定的，不能任意改变。当处理序列数据时，前馈神经网络就无能力为了。因为序列数据是变长的。

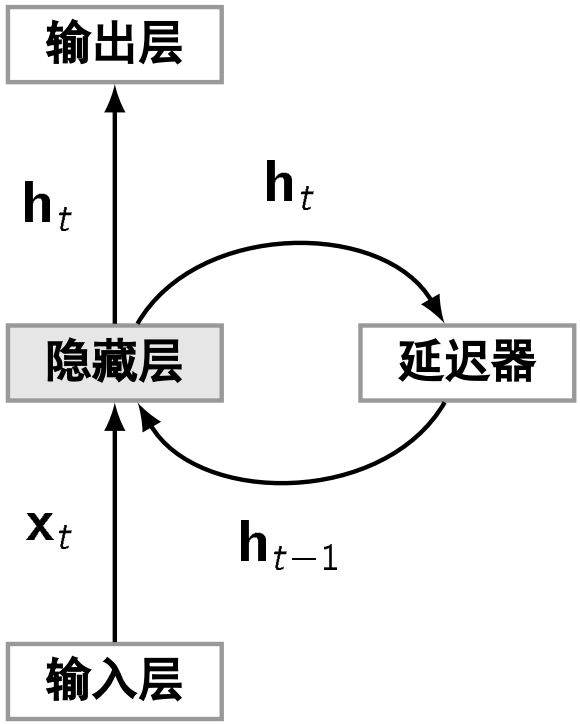
循环神经网络通过使用带自反馈的神经元，能够处理任意长度的序列。循环神经网络比前馈神经网络更加符合生物神经网络的结构。循环神经网络已经被广泛应用在语音识别、语言模型以及自然语言生成等任务上。

循环神经网络

给定一个输入序列$\mathbf{x}^{(1:n)} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(t)}, \ldots, \mathbf{x}^{(n)})$，循环神经网络通过下面公式更新带反馈边的隐藏层的**活性值**$\mathbf{h}^{(t)}$：

$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otherwise} \end{cases} \tag{176}$$

循环神经网络的示例



简单循环网络

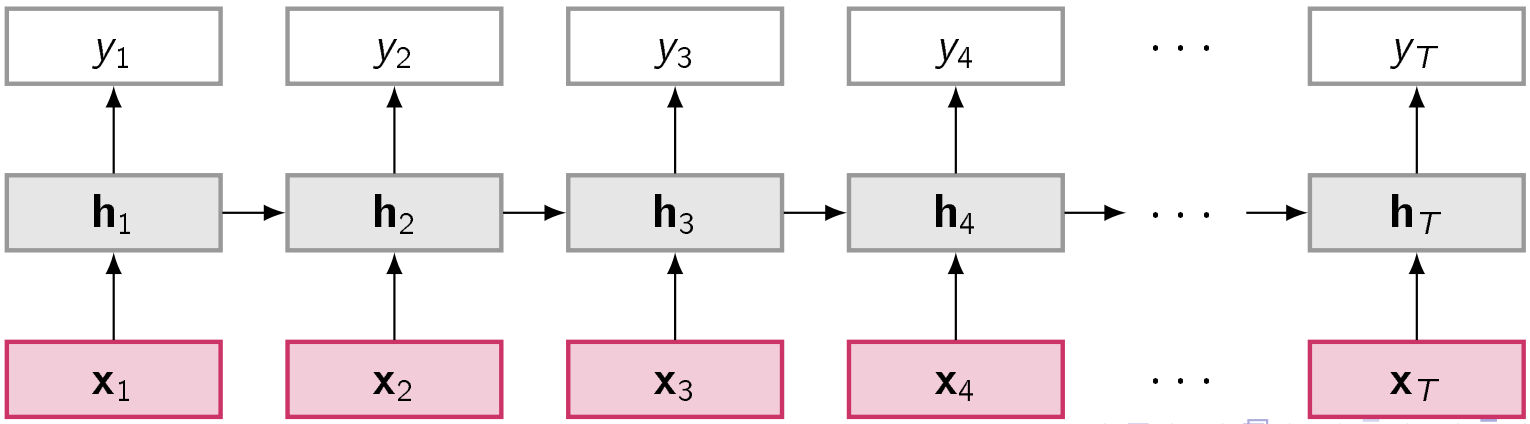
假设时刻 t 时，输入为 $\mathbf{x}_t$，隐层状态（隐层神经元活性）为 $\mathbf{h}_t$。$\mathbf{h}_t$ 不仅和当前时刻的输入相关，也和上一个时刻的隐层状态相关。

一般我们使用如下函数：

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b}), \tag{177}$$

这里，f 是非线性函数，通常为 logistic 函数或 tanh 函数。

图8给出了按时间展开的循环神经网络。



梯度

循环神经网络的参数训练可以通过**随时间进行反向传播**（Backpropagation Through Time，BPTT）算法 [Werbos, 1990]。

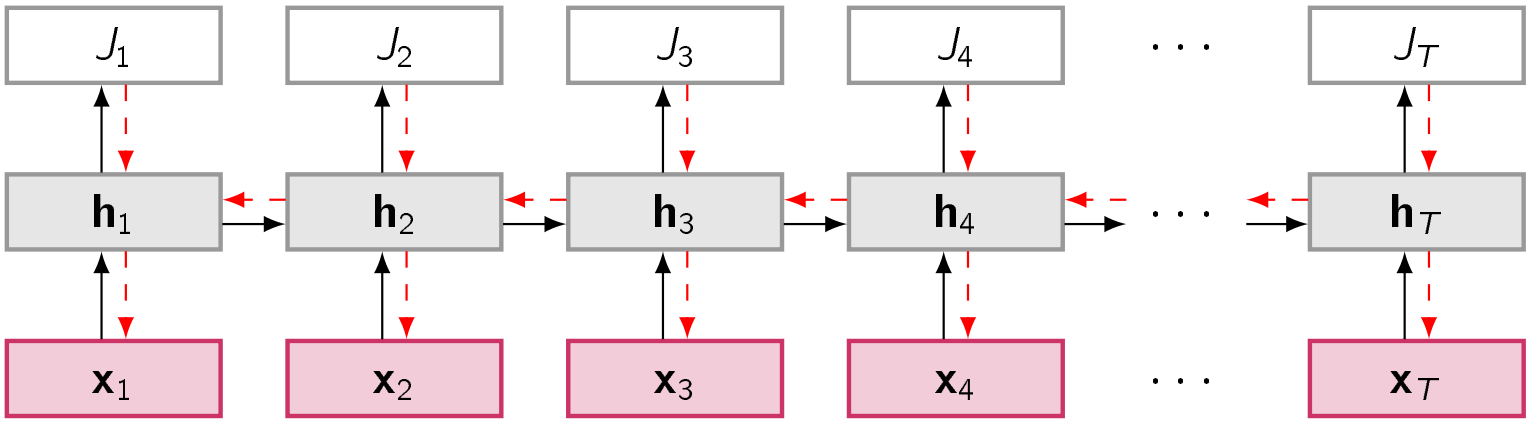


图: 按时间展开的循环神经网络

梯度

假设循环神经网络在每个时刻 t 都有一个监督信息，损失为 J_t。则整个序列的损失为 $J = \sum_{t=1}^{T} J_t$。

损失 J 关于 U 的梯度为：

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T} \frac{\partial J_t}{\partial U} \tag{178}$$

$$= \sum_{t=1}^{T} \frac{\partial \mathbf{h}_t}{\partial U} \frac{\partial J_t}{\partial \mathbf{h}_t}, \tag{179}$$

其中，$\mathbf{h}_t$ 是关于 U 和 $\mathbf{h}_{t-1}$ 的函数，而 $\mathbf{h}_{t-1}$ 又是关于 U 和 $\mathbf{h}_{t-2}$ 的函数。

梯度

用链式法则得到

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T}\sum_{k=1}^{t} \frac{\partial \mathbf{h}_k}{\partial U}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t}\frac{\partial J_t}{\partial \mathbf{y}_t}, \tag{180}$$

其中，

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \tag{181}$$

$$= \prod_{i=k+1}^{t} U^T \mathbf{diag}[f'(h_{i-1})]. \tag{182}$$

梯度

因此，

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathbf{h}_k}{\partial U} \left(\prod_{i=k+1}^{t} U^T \mathbf{diag}(f'(h_{i-1})) \right) \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial J_t}{\partial \mathbf{y}_t}. \tag{183}$$

长期依赖问题

我们定义$\gamma = \|U^T \mathbf{diag}(f'(h_{i-1}))\|$，则在上面公式中的括号里面为$\gamma^{t-k}$。如果$\gamma > 1$，当$t-k \to \infty$时，$\gamma^{t-k} \to \infty$，会造成系统不稳定，也就是所谓的**梯度爆炸**问题；相反，如果$\gamma < 1$，当$t-k \to \infty$时，$\gamma^{t-k} \to 0$，会出现和深度前馈神经网络类似的**梯度消失**问题。

因此，虽然简单循环网络从理论上可以建立长时间间隔的状态之间的依赖关系（Long-Term Dependencies），但是由于梯度爆炸或消失问题，实际上只能学习到短周期的依赖关系。这就是所谓的**长期依赖问题**。

改进方案

为了避免梯度爆炸或消失问题，关键是使得 $U^T \mathbf{diag}(f'(h_{i-1})) = 1$。一种方式就是选取合适的参数，同时使用非饱和的激活函数。但这样的方式需要很多人工经验，同时限制了模型的广泛应用。

还有一种方式就是改变模型，比如让 $U = 1$，同时使用 $f'(h_{i-1}) = 1$。

$$\mathbf{h}_t = \mathbf{h}_{t-1} + \mathbf{W}\mathbf{g}(\mathbf{x}_t), \tag{184}$$

$\mathbf{g}$ 是非线性激活函数。

但这样的形式，丢失了神经元在反馈边上的非线性激活的性质。

改进方案

一个更加有效的改进是引入一个新的状态 $\mathbf{c}_t$ 专门来进行线性的反馈传递，同时在 $\mathbf{c}_t$ 的信息非线性传递给 $\mathbf{h}_t$。

$$\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{W}\mathbf{g}(\mathbf{x}_t), \tag{185}$$

$$\mathbf{h}_t = \tanh(\mathbf{c}_t). \tag{186}$$

但是，这样依然存在一定的问题。因为 $\mathbf{c}_t$ 和 $\mathbf{c}_{t-1}$ 是线性关系，同时不断累积 $\mathbf{x}_t$ 的信息，会使得 $\mathbf{c}_t$ 变得越来越大。

长短时记忆神经网络：LSTM

长短时记忆神经网络（Long Short-Term Memory Neural Network，LSTM）[Hochreiter and Schmidhuber, 1997] 是循环神经网络的一个变体，可以有效地解决简单循环神经网络的梯度爆炸或消失问题。

LSTM 模型的关键是引入了一组记忆单元（Memory Units），允许网络可以学习何时遗忘历史信息，何时用新信息更新记忆单元。在时刻 t 时，记忆单元 $\mathbf{c}_t$ 记录了到当前时刻为止的所有历史信息，并受三个“门”控制：输入门 $\mathbf{i}_t$，遗忘门 $\mathbf{f}_t$ 和输出门 $\mathbf{o}_t$。三个门的元素的值在 $[0, 1]$ 之间。

LSTM

在时刻 t 时LSTM 的更新方式如下：

$$\mathbf{i}_t = \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + V_i\mathbf{c}_{t-1}), \tag{187}$$

$$\mathbf{f}_t = \sigma(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + V_f\mathbf{c}_{t-1}), \tag{188}$$

$$\mathbf{o}_t = \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + V_o\mathbf{c}_t), \tag{189}$$

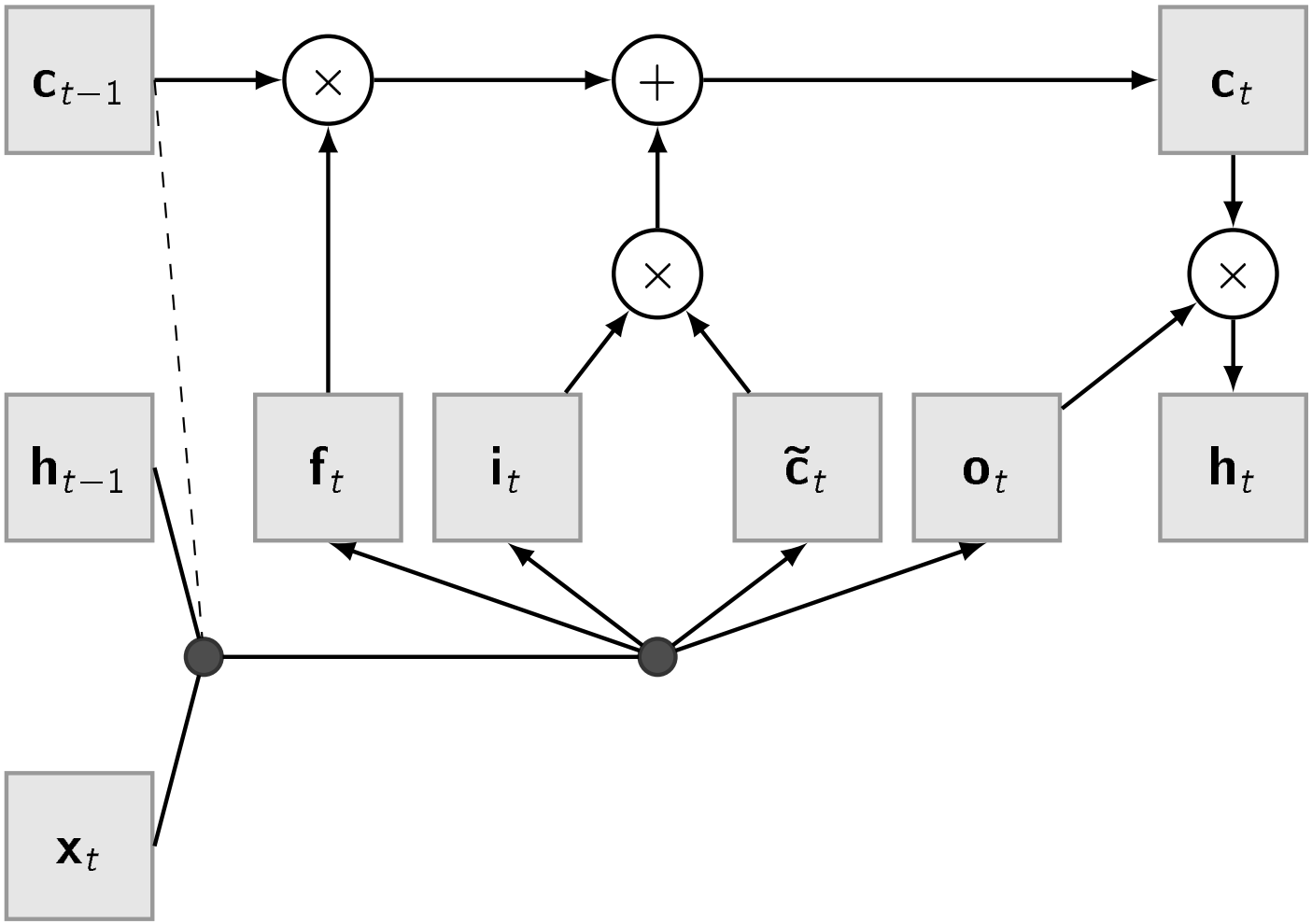
$$\tilde{\mathbf{c}}_t = \tanh(W_c\mathbf{x}_t + U_c\mathbf{h}_{t-1}), \tag{190}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \tag{191}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{192}$$

这里，$\mathbf{x}_t$ 是当前时刻的输入，σ 是logistic 函数，V_i，V_f，V_o 是对角矩阵。遗忘门$\mathbf{f}_t$ 控制每一个内存单元需要遗忘多少信息，输入门$\mathbf{i}_t$ 控制每一个内存单元加入多少新的信息，输出门$\mathbf{o}_t$ 控制每一个内存单元输出多少信息。

LSTM 结构示例


$\mathbf{c}_{t-1}$
$\mathbf{c}_t$
$\mathbf{h}_{t-1}$
$\mathbf{f}_t$
$\mathbf{i}_t$
$\tilde{\mathbf{c}}_t$
$\mathbf{o}_t$
$\mathbf{h}_t$
$\mathbf{x}_t$

门限循环单元：GRU

门限循环单元（Gated Recurrent Unit，GRU）[Cho et al., 2014, Chung et al., 2014]是一种比LSTM更加简化的版本。在LSTM中，输入门和遗忘门是互补关系，因为同时用两个门比较冗余。GRU将输入门与和遗忘门合并成一个门：更新门（Update Gate），同时还合并了记忆单元和神经元活性。

GRU模型中有两个门：更新门$\mathbf{z}$和重置门$\mathbf{r}$。更新门$\mathbf{z}$用来控制当前的状态需要遗忘多少历史信息和接受多少新信息。重置门$\mathbf{r}$用来控制候选状态中有多少信息是从历史信息中得到。

GRU模型的更新方式如下：

$$\mathbf{r}_t = \sigma(\mathbf{W}_r\mathbf{x}_t + \mathbf{U}_r\mathbf{h}_{t-1}) \tag{193}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z\mathbf{x}_t + \mathbf{U}_z\mathbf{h}_{t-1}) \tag{194}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1})), \tag{195}$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t, \tag{196}$$

(197)

大纲

最新进展

- 控制 + 计算 + 记忆
  - 传统神经网络
- 记忆 $\leftrightsquigarrow$ 控制 + 计算
  - 神经图灵机 Neural Turing Machines [Graves et al., 2014]
  - 记忆神经网络 Memory Network [Sukhbaatar et al., 2015]
  - 动态记忆神经网络 Dynamic Memory Network [Kumar et al., 2015]
- 记忆 $\leftrightsquigarrow$ 控制 $\leftrightsquigarrow$ 计算
  - 神经随机访问机 Neural Random-Access Machines [Kurach et al., 2015]
- 结构记忆（栈、队列）
  - Learning to Transduce with Unbounded Memory [Grefenstette et al., 2015]

开发工具

- Torch 7 （Facebook）
- Theano （Bengio）
- Keras （Recommended, Based on Theano/TensorFlow）

详细材料

《神经网络与深度学习》最新讲义：

http://vdisk.weibo.com/s/A_pmE4ilotGf/1450319894

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. Incorporating second-order functional knowledge for better option pricing. *Advances in Neural Information Processing Systems*, pages 472–478, 2001.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. *CoRR*, abs/1506.02516, 2015. URL http://arxiv.org/abs/1506.02516.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *arXiv preprint arXiv:1506.07285*, 2015.

Karol Kurach, Marcin Andrychowicz, and Ilya Sutskever. Neural random-access machines. *CoRR*, abs/1511.06392, 2015. URL http://arxiv.org/abs/1511.06392.

Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg

Corrado, Jeff Dean, and Andrew Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning*, 2012.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

Albert BJ Novikoff. On convergence proofs for perceptrons. Technical report, DTIC Document, 1963.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386–408, 1958.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5:3, 1988.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, pages 2431–2439, 2015.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Matthew D Zeiler. Adadelta: An adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.