《神经网络与深度学习》讲义

Notes on Artificial Neural Networks and Deep Learning

邱锡鹏
xpqiu@fudan.edu.cn

2015年12月11日

目录

第一章　绪论

> 一个人在不接触对方的情况下，通过一种特殊的方式，和对方进行一系列的问答。如果在相当长时间内，他无法根据这些问题判断对方是人还是计算机，那么就可以认为这个计算机是智能的。
>
> — Alan Turing [1950],《机器能思维吗?》

让机器具备智能是人们长期追求的目标，但是关于智能的定义也十分模糊。Alan Turing在1950年提出了著名的图灵测试：“一个人在不接触对方的情况下，通过一种特殊的方式，和对方进行一系列的问答。如果在相当长时间内，他无法根据这些问题判断对方是人还是计算机，那么就可以认为这个计算机是智能的”。

要通过真正地通过图灵测试，计算机必须具备理解语言、学习、记忆、推理、决策等能力。这也延伸出很多不同的学科，比如机器感知（计算机视觉、自然语言处理），学习（模式识别、机器学习、增强学习），记忆（知识表示）、决策（规划、数据挖掘）等。所有这些分支学科都可以看成是**人工智能**（Artificial Intelligence，AI）的研究范畴。其中，**机器学习**（Machine Learning，ML）因其在很多领域的出色表现逐渐成为热门学科。机器学习的主要目的是设计和分析一些**学习算法**，让计算机从数据中获得一些决策函数，从而可以帮助人们解决一些特定任务，提高效率。对于人工智能来说，机器学习从一开始就是一个重要的研究方向，并涉及了概率论、统计学、逼近论、凸分析、计算复杂性理论等多门学科。

人工神经网络（Artificial Neural Network，ANN），也简称**神经网络**，是众多机器学习算法中比较接近生物神经网络特性的数学模型。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。

Rosenblatt [1958]最早提出可以模拟人类感知能力的数学模型，并称之为**感知器**（Perceptron），并提出了一种接近于人类学习过程（迭代、试错）的学习算法。但感知器因其

结构过于简单，不能解决简单的异或（XOR）等线性不可分问题，造成了人工神经领域发展的长年停滞及低潮。直到1980年以后，Geoffrey Hinton、Yann LeCun等人将**反向传播算法**（Backpropagation，BP）引入到多层感知器[Williams and Hinton, 1986]，人工神经网络才又重新引起人们的注意，并开始成为新的研究热点。但是，2000年以后，因为当时计算机的计算能力不足以支持训练大规模的神经网络，并且随着支持向量机（Support Vector Machines，SVM）等方法的兴起，人工神经网络又一次陷入低潮。

直到2006年，Hinton and Salakhutdinov [2006]发现多层前馈神经网络可以先通过逐层预训练，再用反向传播算法进行精调的方式进行有效学习。并且近年来计算机计算能力的提高（大规模并行计算，GPU），计算机已经可以训练大规模的人工神经网络。随着深度的人工神经网络在语音识别[Hinton et al., 2012]和图像分类[Krizhevsky et al., 2012]等任务上的巨大成功，越来越多的人开始关注这一个“崭新”的研究领域：深度学习。目前，深度学习技术在学术界和工业界取得了广泛的成功，并逐渐受到了高度重视。

深度学习（Deep Learning，DL）是从机器学习中的人工神经网络发展出来的新领域。早期所谓的“深度”是指超过一层的神经网络。但随着深度学习的快速发展，其内涵已经超出了传统的多层神经网络，甚至机器学习的范畴，逐渐朝着人工智能的方向快速发展。

本书主要介绍人工神经网络与深度学习中的基础知识、主要模型（卷积神经网络、递归神经网络等）以及在计算机视觉、自然语言处理等领域的应用。

1.1 总结和深入阅读

若希望全面了解人工神经网络和深度学习的知识，可以参考如下材料：

1. Ian Goodfellow, Aaron Courville, and Yoshua Bengio. Deep learning. Book in preparation for MIT Press, 2015. URL http://goodfeli.github.io/dlbook/
2. Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009

另外，网站也给出很好的教程，比如http://deeplearning.net/等。

第二章　数学基础

2.1　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。

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}, \tag{2.1}$$

其中，a_i称为向量$\mathbf{a}$的第i个分量，或第i维。

为简化书写、方便排版起见，有时会以加上**转置**符号T的行向量（大小为$1 \times n$的矩阵）表示列向量。

$$\mathbf{a} = [a_1, a_2, \cdots, a_n]^{\mathrm{T}} \tag{2.2}$$

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$\alpha, \beta, \gamma$等来表示。

2.1.1　向量的模

向量$\mathbf{a}$的模$\|\mathbf{a}\|$为

$$\|\mathbf{a}\| = \sqrt{\sum_{i=1}^{n} a_i^2}. \tag{2.3}$$

2.1.2　向量的范数

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：

$$|\mathbf{x}|_1 = \sum_{i=1}^{n} |x_i|. \tag{2.4}$$

L_2 范数：

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{\mathbf{x}^T\mathbf{x}}. \tag{2.5}$$

2.2 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。

一个矩阵A从左上角数起的第i行第j列上的元素称为第i,j项，通常记为$A_{i,j}$或A_{ij}。

一个n维向量可以看作是$n \times 1$的矩阵。

2.3 矩阵的基本运算

如果A和B都为$m \times n$的矩阵，则的$A+B$和$A-B$也是$m \times n$的矩阵，其每个元素是A和B相应元素相加或相减。

$$(A+B)_{ij} = A_{ij} + B_{ij}, \tag{2.6}$$

$$(A-B)_{ij} = A_{ij} - B_{ij}. \tag{2.7}$$

一个标量b与矩阵A乘积为A的每个元素是A的相应元素与c的乘积

$$(cA)_{ij} = cA_{ij}. \tag{2.8}$$

一个标量b与矩阵A乘积为A的每个元素是A的相应元素与c的乘积

$$(cA)_{ij} = cA_{ij}. \tag{2.9}$$

两个矩阵的乘积仅当第一个矩阵A的列数和另一个矩阵B的行数相等时才能定义。如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

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{p} A_{ik}B_{kj} \tag{2.10}$$

矩阵的乘法满足**结合律**和**分配律**：

- 结合律：$(AB)C = A(BC)$,
- 分配律：$(A+B)C = AC + BC$，$C(A+B) = CA + CB$.

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

$$(A^{\mathrm{T}})_{ij} = A_{ji}. \tag{2.11}$$

2.3.1 常见的矩阵

对称矩阵指其转置等于自己的矩阵，即满足$A = A^{\mathrm{T}}$。

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：

$$A_{ij} = 0 \text{ if } i \neq j \qquad \forall i, j \in \{1, \cdots, n\} \tag{2.12}$$

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

$$A_{ii} = a_i. \tag{2.13}$$

$n \times n$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

$$A\mathbf{b} = \mathbf{diag}(\mathbf{a})\mathbf{b} = \mathbf{a} \odot \mathbf{b}, \tag{2.14}$$

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

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$I_n = \mathbf{diag}(1, 1, ..., 1)$。

一个矩阵和单位矩阵的乘积等于其本身。

$$AI = IA = A \tag{2.15}$$

2.3.2 矩阵的范数

矩阵的范数有很多种形式，这里我们定义其p-范数为

$$\|A\|_p = \Big(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^p \Big)^{1/p}. \tag{2.16}$$

2.4 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

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_p} \end{bmatrix} \in \mathbb{R}^p. \tag{2.17}$$

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

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_q(\mathbf{x})}{\partial x_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_q(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_q(\mathbf{x})}{\partial x_p} \end{bmatrix} \in \mathbb{R}^{p \times q}. \tag{2.18}$$

2.4.1 常见的向量导数

$$\frac{\partial A\mathbf{x}}{\partial \mathbf{x}} = A^{\mathrm{T}}, \tag{2.19}$$

$$\frac{\partial \mathbf{x}^{\mathrm{T}} A}{\partial \mathbf{x}} = A \tag{2.20}$$

2.4.2 导数法则

导数满足如下法则：

- 加（减）法则：$\mathbf{y} = f(\mathbf{x})$,$\mathbf{z} = g(\mathbf{x})$ 则

$$\frac{\partial(\mathbf{y} + \mathbf{z})}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} + \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \tag{2.21}$$

- 乘法法则：$\mathbf{y} = f(\mathbf{x})$,$\mathbf{z} = g(\mathbf{x})$ 则

$$\frac{\partial \mathbf{y}^T \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \mathbf{z} + \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \mathbf{y} \tag{2.22}$$

- 链式法则：$\mathbf{z} = f(\mathbf{y})$,$\mathbf{y} = g(\mathbf{x})$ 则

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{y}} \tag{2.23}$$

如果$\mathbf{z} = f(\mathbf{y})$,$\mathbf{y} = g(X)$，$X$为矩阵，则

$$\frac{\partial \mathbf{z}}{\partial X_{ij}} = \mathbf{tr}\left((\frac{\partial \mathbf{z}}{\partial \mathbf{y}})^T \frac{\partial \mathbf{y}}{\partial X_{ij}} \right) \tag{2.24}$$

2.5 常用函数

这里我们介绍几个本书中常用的函数。

假设一个函数 $f(x)$ 的输入是标量 x。对于一组 K 个标量 $x_1, \cdots, x_K$，我们可以通过 $f(x)$ 得到另外一组 K 个标量 $z_1, \cdots, z_K$，

$$z_k = f(x_k), \forall k = 1, \cdots, K \tag{2.25}$$

为了简便起见，我们定义 $\mathbf{x} = [x_1, \cdots, x_K]^{\mathrm{T}}$，$\mathbf{z} = [z_1, \cdots, z_K]^{\mathrm{T}}$，

$$\mathbf{z} = f(\mathbf{x}), \tag{2.26}$$

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

如果 $f(x)$ 的导数记为 $f'(x)$。当这个函数的输入为 K 维向量 $\mathbf{x} = [x_1, \cdots, x_K]^{\mathrm{T}}$ 时，其导数为一个对角矩阵。

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = [\frac{\partial f(x_j)}{\partial x_i}]_{K \times K} = \begin{bmatrix} f'(x_1) & 0 & \cdots & 0 \\ 0 & f'(x_2) & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & f'(x_K) \end{bmatrix} \tag{2.27}$$

$$= \mathbf{diag}(f'(\mathbf{x})). \tag{2.28}$$

2.5.1 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 $\sigma(x)$

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{2.29}$$

其导数为

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{2.30}$$

$$\tag{2.31}$$

2.5.2 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

$$z_k = \mathbf{softmax}(x_k) = \frac{\exp(x_k)}{\sum_{i=1}^{K} \exp(x_i)}, \tag{2.32}$$

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

$$z_k \in [0, 1], \forall k, \qquad \sum_{i=1}^{K} z_k = 1. \tag{2.33}$$

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

$$\begin{aligned}
\hat{\mathbf{z}} &= \mathbf{softmax}(\mathbf{x}) && (2.34)\\
&= \frac{1}{\sum_{k=1}^{K} \exp(x_k)} \begin{bmatrix} \exp(\mathbf{x}_1) \\ \vdots \\ \exp(\mathbf{x}_K) \end{bmatrix} \\
&= \frac{\exp(\mathbf{x})}{\sum_{k=1}^{K} \exp(x_k)} \\
&= \frac{\exp(\mathbf{x})}{\underline{1}_K^{\mathrm{T}} \exp(\mathbf{x})}, && (2.35)
\end{aligned}$$

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

其导数为

$$\begin{aligned}
\frac{\partial \mathbf{softmax}(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial \left(\frac{\exp(\mathbf{x})}{\underline{1}_K^{\mathrm{T}} \exp(\mathbf{x})} \right)}{\partial \mathbf{x}} \\
&= \frac{1}{\underline{1}_K \exp(\mathbf{x})} \frac{\partial \exp(\mathbf{x})}{\partial \mathbf{x}} + \frac{\partial \left(\frac{1}{\underline{1}_K^{\mathrm{T}} \exp(\mathbf{x})} \right)}{\partial \mathbf{x}} (\exp(\mathbf{x}))^{\mathrm{T}} \\
&= \frac{\mathbf{diag}(\exp(\mathbf{x}))}{\underline{1}_K \exp(\mathbf{x})} - \left(\frac{1}{(\underline{1}_K^{\mathrm{T}} \exp(\mathbf{x}))^2} \right) \frac{\partial \left(\underline{1}_K^{\mathrm{T}} \exp(\mathbf{x}) \right)}{\partial \mathbf{x}} (\exp(\mathbf{x}))^{\mathrm{T}} \\
&= \frac{\mathbf{diag}(\exp(\mathbf{x}))}{\underline{1}_K \exp(\mathbf{x})} - \left(\frac{1}{(\underline{1}_K^{\mathrm{T}} \exp(\mathbf{x}))^2} \right) \mathbf{diag}(\exp(\mathbf{x})) \underline{1}_K (\exp(\mathbf{x}))^{\mathrm{T}} \\
&= \frac{\mathbf{diag}(\exp(\mathbf{x}))}{\underline{1}_K \exp(\mathbf{x})} - \left(\frac{1}{(\underline{1}_K^{\mathrm{T}} \exp(\mathbf{x}))^2} \right) \exp(\mathbf{x}) (\exp(\mathbf{x}))^{\mathrm{T}} \\
&= \mathbf{diag}\left(\frac{\exp(\mathbf{x})}{\underline{1}_K \exp(\mathbf{x})} \right) - \frac{\exp(\mathbf{x})}{\underline{1}_K^{\mathrm{T}} \exp(\mathbf{x}))} \cdot \frac{(\exp(\mathbf{x}))^{\mathrm{T}}}{\underline{1}_K^{\mathrm{T}} \exp(\mathbf{x}))} \\
&= \mathbf{diag}(\mathbf{softmax}(\mathbf{x})) - \mathbf{softmax}(\mathbf{x}) \mathbf{softmax}(\mathbf{x})^{\mathrm{T}}, && (2.36)
\end{aligned}$$

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

2.6 总结和深入阅读

详细的矩阵偏导数参考https://en.wikipedia.org/wiki/Matrix_calculus。

第三章　机器学习概述

机器学习是对能通过经验自动改进的计算机算法的研究。

— Mitchell [1997]

在介绍人工神经网络之前，我们先来了解下机器学习的基本概念。然后再介绍下最简单的神经网络：感知器。

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。

机器学习系统的示例见图3.1。

3.1　机器学习概述

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。

$$\hat{y} = f(\boldsymbol{\phi}(x), \theta), \tag{3.1}$$

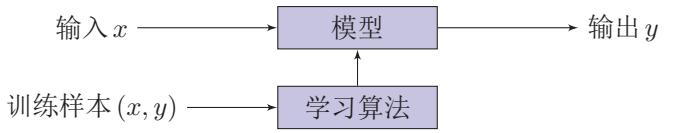


图 3.1: 机器学习系统示例

这里，$\hat{y}$是模型输出，θ为决策函数的参数，$\boldsymbol{\phi}(x)$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$\boldsymbol{\phi}(x)$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3.1也可以直接写为

$$\hat{y} = f(x, \theta). \tag{3.2}$$

此外，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。

$$R(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(y^{(i)}, f(x^{(i)}, \theta)). \tag{3.3}$$

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。

$$\theta^* = \arg\min_{\theta} R(\theta). \tag{3.4}$$

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** index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。过拟合的标准定义为：给定一个假设空间H，一个假设h属于H，如果存在其他的假设h属于H,使得在训练样例上h的损失比h小，但在整个实例分布上h比h 的损失小，那么就说假设h过度拟合训练数据 [Mitchell, 1997]。

和过拟合相对应的一个概念是**泛化错误**。泛化错误是衡量一个机器学习模型是否可以很好地泛化到未知数据。泛化错误一般表现为一个模型在训练集和测试集上错误率的差距。

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）。

$$\theta^* = \arg\min_{\theta} R(\theta) + \lambda \|\theta\|_2^2 \tag{3.5}$$

$$= \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(y^{(i)}, f(x^{(i)}, \theta)) + \lambda \|\theta\|^2. \tag{3.6}$$

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。

3.1.1 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

$$L(y, f(x, \theta)) = \begin{cases} 1 & \textbf{if } y = f(x, \theta) \\ 0 & \textbf{if } y \neq f(x, \theta) \end{cases} \tag{3.7}$$

$$= I(y \neq f(x, \theta)), \tag{3.8}$$

这里I是指示函数。

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

$$L(y, \hat{y}) = (y - f(x, \theta))^2 \tag{3.9}$$

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

$$f_i(x, \theta) \in [0, 1], \qquad \sum_{i=1}^{C} f_i(x, \theta) = 1 \tag{3.10}$$

$f_y(x, \theta)$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）。

$$L(y, f(x, \theta)) = -\log f_y(x, \theta). \tag{3.11}$$

如果我们用one-hot向量[1]$\mathbf{y}$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：

$$L(y, f(x,\theta)) = -\sum_{i=1}^{C} y_i \log f_i(x,\theta). \tag{3.12}$$

y_i也也可以看成是真实类别的分布，这样公式3.12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）是负对数似然函数的一种改进。

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：

$$L(y, f(x,\theta)) = \max(0, 1 - yf(x,\theta)) \tag{3.13}$$

$$= |1 - yf(x,\theta)|_+. \tag{3.14}$$

3.1.2 机器学习算法的类型

根据训练数据提供的信息以及反馈方式的不同，机器学习算法一般可以分为以下几类：

有监督学习（Supervised Learning） 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。

$$L(y, f(x,\theta)) = \|y - f(x,\theta)\|^2$$

分类（Classification） 如果输出y是离散的类别标记（符号），就是分类问题。损失函数有很多种定义方式。一种常用的方式0-1损失函数。

$$L(\hat{y}, y) = I(f(x,\theta) \neq y),$$

[1] 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。

这里 $f(x,\theta)$ 的输出也是离散值，$I(\cdot)$ 是指示函数，若条件为真，$I(\cdot)=1$；否则 $I(\cdot)=0$。另一种常用的方式是让 $f_i(x,\theta)$ 去估计给定 x 的情况下第 i 个类别的条件概率 $P(y=i|x)$。损失函数定义为负对数似然函数。

$$L(y,f(x,\theta)) = -\log f_y(x,\theta).$$

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**。

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。

有监督的学习方法需要每个数据记录都有类标号，而无监督的学习方法则不考虑任何指导性信息。一般而言，一个监督学习模型需要大量的有标记数据集，而这些数据集是需要人工标注的。因此，也出现了很多**弱监督学习**和**半监督学习**的方法，希望从大规模的未标记数据中充分挖掘有用的信息，降低对标记数据数量的要求。

3.1.3 机器学习中的一些概念

上述的关于机器学习的介绍中，提及了一些基本概念，比如“数据”，“样本”，“特征”，“数据集”等。我们首先来解释下这些概念。

数据

在计算机科学中，**数据**是指所有能计算机程序处理的对象的总称，可以是数字、字母和符号等。在不同的任务中，表现形式不一样，比如图像、声音、文字、传感器数据等。

特征

机器学习中很多算法的输入要求是数学上可计算的。而在现实世界中，原始数据通常是并不都以连续变量或离散变量的形式存在的。我们首先需要将抽取出一些可以表征这些数据的数值型特征。这些数值型特征一般可以表示为向量形式，也称为**特征向量**。

特征学习

数据的原始表示转换为。原始数据的特征有很多，但是并不是所有的特征都是有用的。并且，很多特征通常是冗余并且易变的。我们需要抽取有效的、稳定的特征。传统的特征提取是通过人工方式进行的，这需要大量的人工和专家知识。即使这样，人工总结的特征在很多任务上也不能满足需要。因此，如何自动地学习有效的特征也成为机器学习中一个重要的研究内容，也就是**特征学习**，也叫**表示学习**。特征学习分成两种，一种是**特征选择**，是在很多特征集合选取有效的子集；另一种是**特征提取**，是构造一个新的特征空间，并将原始特征投影在新的空间中。

样本

样本是按照一定的抽样规则从全部数据中取出的一部分数据，是实际观测得到的数据。在有监督学习中，需要提供一组有输出目标的样本用来学习模型以及检验模型的好坏。

训练集和测试集

一组样本集合就称为**数据集**。在很多领域，数据集也经常称为**语料库**。为了检验机器学习算法的好坏，一般将数据集分为两部分：训练集和测试集。训练集用来进行模型学习，测试集用来进行模型验证。通过学习算法，在训练集得到一个模型，这个模型可以对测试集上样本x预测一个类别标签$\hat{y}$。假设测试集为T,模型的正确率为：

$$Acc = \frac{1}{|T|} \sum_{(x_i, y_i) \in T} |\hat{y}_i = y_i|, \tag{3.15}$$

其中$|T|$为测试集的大小。第**??**节中会介绍更多的评价方法。

正例和负例

对于两类分类问题，类别可以表示为$\{+1, -1\}$，或者直接用正负号表示。因此，常用**正例**和**负例**来分别表示属于不同类别的样本。

判别函数

经过特征抽取后，一个样本可以表示为k维特征空间中的一个点。为了对这个特征空间中的点进行区分，就需要寻找一些超平面来将这个特征空间分为一些互不重叠的子区域，使得不同类别的点分布在不同的子区域中，这些超平面就成为判别界面。

为了定义这些用来进行空间分割的超平面，就需要引入判别函数的概念。假设变量$\mathbf{z} \in \mathbb{R}^m$为特征空间中的点，这个超平面由所有满足函数$f(\mathbf{z}) = 0$的点组成。这里的$f(\mathbf{z})$就称为**判别函数**。

有了判别函数，分类就变得很简单，就是看一个样本在特征空间中位于哪个区域，从而确定这个样本的类别。

判别函数的形式多种多样，在自然语言处理中，最为常用的判别函数为线性函数。

3.1.4 参数学习算法

学习算法就是如何从训练集的样本中，自动学习决策函数的参数。不同机器学习算法的区别在于决策函数和学习算法的差异。相同的决策函数可以有不同的学习算法。比如线性分类器，其参数的学习算法可以是感知器、支持向量机以及梯度下降法等。通过一个学习算法进行自动学习参数的过程也叫作**训练过程**。

这里我们介绍一种常用的参数学习算法：**梯度下降法**（Gradient Descent Method）。

梯度下降法也叫最速下降法（Steepest Descend Method）。如果一个实值函数$f(\mathbf{x})$在点$\mathbf{a}$处可微且有定义，那么函数$f(\mathbf{x})$在$\mathbf{a}$点沿着梯度相反的方向$-\nabla f(\mathbf{a})$下降最快。梯度下降法经常用来求解无约束优化的极值问题。梯度下降法的迭代公式为：

$$\mathbf{a}_{t+1} = \mathbf{a}_t - \lambda \nabla f(\mathbf{a_t}), \tag{3.16}$$

其中$\lambda > 0$是梯度方向上的搜索步长。

对于λ为一个够小数值时，那么 $f(\mathbf{a}_{k+1}) \leq f(\mathbf{a}_1)$。因此，我们可以从一个初始值$\mathbf{x}_0$开始，并通过迭代公式得到$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n$，并满足

$$f(\mathbf{x}_0) \geq f(\mathbf{x}_1) \geq f(\mathbf{x}_2) \geq \cdots \geq f(\mathbf{x}_n),$$

最终$\mathbf{x}_n$收敛到期望的极值。

搜索步长的取值必须合适，如果过大就不会收敛，如果过小则收敛速度太慢。一般步长可以由线性搜索算法来确定。

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。

$$\theta^* = \arg\min_{\theta} \mathcal{R}(\theta_t) \tag{3.17}$$

$$= \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\big(y^{(i)}, f(x^{(i)}, \theta)\big). \tag{3.18}$$

如果用梯度下降法进行参数学习，

$$\mathbf{a}_{t+1} = \mathbf{a}_t - \lambda \frac{\partial \mathcal{R}(\theta)}{\partial \theta_t} \tag{3.19}$$

$$= \mathbf{a}_t - \lambda \sum_{i=1}^{N} \frac{\partial \mathcal{R}\big(\theta_t; x^{(i)}, y^{(i)}\big)}{\partial \theta}, \tag{3.20}$$

λ在机器学习中也叫作**学习率**（Learning Rate）。

这里，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

$$\mathbf{a}_{t+1} = \mathbf{a}_t - \lambda \frac{\partial \mathcal{R}\big(\theta_t; x^{(t)}, y^{(t)}\big)}{\partial \theta}, \tag{3.21}$$

$x^{(t)}, y^{(t)}$是第t次迭代选取的样本。

批量梯度下降和随机梯度下降之间的区别在于每次迭代的风险是对所有样本汇总的风险还是单个样本的风险。随机梯度下降因为实现简单，收敛速度也非常快，因此使用非常广泛。

还有一种折中的方法就是**mini-batch随机梯度下降**，每次迭代时，只采用一小部分的训练样本，兼顾了批量梯度下降和随机梯度下降的优点。

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

在梯度下降中，学习率的取值非常关键，如果过大就不会收敛，如果过小则收敛速度太慢。一般步长可以由线性搜索算法来确定。在机器学习中，经常使用自适应调整学习率的方法。

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，

$$\theta_t = \theta_{t-1} + (\rho\nabla\theta_t - \lambda g_t), \tag{3.22}$$

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。

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，

$$\theta_t = \theta_{t-1} - \frac{\rho}{\sqrt{\sum_{\tau=1}^{t} g_\tau^2}} g_t, \tag{3.23}$$

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。

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：

$$E(g^2)_t = \rho E(g^2)_{t-1} + (1-\rho) g_t^2, \tag{3.24}$$

其中，ρ 是衰减常数。

本次迭代的更新为

$$\nabla\theta_t = -\frac{\sqrt{E(\nabla\theta^2)_{t-1} + \epsilon}}{\sqrt{E(g^2)_t + \epsilon}} g_t \tag{3.25}$$

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

$$E(\nabla\theta^2)_t = \rho E(\nabla\theta^2)_{t-1} + (1-\rho)\nabla\theta_t^2. \tag{3.26}$$

最后更新参数

$$\theta_t = \theta_{t-1} + \nabla\theta_t. \tag{3.27}$$

3.2 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，

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b, \tag{3.28}$$

这就是线性回归问题（Linear Regression）。

为了简单起见，我们将公式3.28写为

$$f(\mathbf{x}) = \hat{\mathbf{w}}^{\mathrm{T}}\hat{\mathbf{x}}, \tag{3.29}$$

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**。

$$\hat{\mathbf{x}} = 1 \oplus \mathbf{x} \triangleq \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} 1 \\ \\ \mathbf{x} \\ \\ \end{bmatrix}, \tag{3.30}$$

$$\hat{\mathbf{w}} = 1 \oplus \mathbf{w} \triangleq \begin{bmatrix} b \\ w_1 \\ \vdots \\ w_k \end{bmatrix} = \begin{bmatrix} b \\ \\ \mathbf{w} \\ \\ \end{bmatrix}, \tag{3.31}$$

这里 $\oplus$ 只两个向量的拼接。

在后面的描述中，我们一般采用简化的表示方法，直接用 $\mathbf{w}$ 和 $\mathbf{x}$ 来表示增广权重向量和增广特征向量。

线性回归的损失函数通常定义为平方损失函数。

$$L(y, f(x, \mathbf{w})) = \|y - f(x, \mathbf{w})\|^2. \tag{3.32}$$

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

$$R(Y, f(X, \mathbf{w})) = \sum_{i=1}^{N} L(y^{(i)}, f(\mathbf{x}^{(i)}, \mathbf{w})) \tag{3.33}$$

$$= \sum_{i=1}^{N} \|\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)} - y^{(i)}\|^2 \tag{3.34}$$

$$= \|X^{\mathrm{T}}\mathbf{w} - \mathbf{y}\|^2, \tag{3.35}$$

其中，$\mathbf{y}$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：

$$X = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(N)} \\ \vdots & \vdots & \ddots & \vdots \\ x_k^{(1)} & x_k^{(2)} & \cdots & x_k^{(N)} \end{pmatrix} \tag{3.36}$$

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

$$\frac{\partial \mathcal{R}\big(Y, f(X, \mathbf{w})\big)}{\partial \mathbf{w}} = \frac{\partial \|X^{\mathrm{T}}\mathbf{w} - \mathbf{y}\|^2}{\partial \mathbf{w}} \tag{3.37}$$

$$= X(X^{\mathrm{T}}\mathbf{w} - \mathbf{y}) \tag{3.38}$$

$$\tag{3.39}$$

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

$$\mathbf{w} = (XX^{\mathrm{T}})^{-1}X\mathbf{y} \tag{3.40}$$

$$= \big(\sum_{i=1}^{N} \mathbf{x}^{(i)}(\mathbf{x}^{(i)})^{\mathrm{T}}\big)^{-1}\big(\sum_{i=1}^{N} \mathbf{x}^{(i)}y^{(i)}\big). \tag{3.41}$$

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**。

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \lambda \frac{\partial \mathcal{R}\big(Y, f(X, \mathbf{w})\big)}{\partial \mathbf{w}}, \tag{3.42}$$

其中 λ 是学习率。

3.3 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。

3.3.1 两类分类

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

$$\hat{y} = \begin{cases} 1 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x} > 0 \\ 0 & \text{if } \mathbf{w}^{\mathrm{T}}\mathbf{x} \leq 0 \end{cases} \tag{3.43}$$

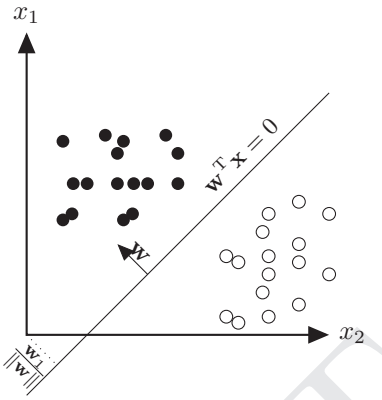


图 3.2: 两类分类线性判别函数

$$= I(\mathbf{w}^{\mathrm{T}}\mathbf{x} > 0), \tag{3.44}$$

其中，$I()$ 是指示函数。

公式3.60定义了一个两类分类问题的线性判别函数。在高维的特征空间中，所有满足 $\mathbf{f}(z) = 0$ 的点组成用一个超平面，这个超平面将特征空间一分为二，划分成两个区域。这两个区域分别对应两个类别。

图3.2中给了一个两维数据的判别函数以及对应的判别界面。在二维空间中，分类界面为一个直线。在三维空间中，分类界面为一个平面。在高维空间中，分类界面为一个超平面。对于线性函数来说，权重向量在线性空间中垂直于分类界面的向量。

Logistic 回归

线性分类函数的参数 $\mathbf{w}$ 有很多种学习方式，比如第四章中介绍的感知器。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：

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x})}, \tag{3.45}$$

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 $P(y = 0|\mathbf{x}) = 1 - P(y = 1|\mathbf{x})$。

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：

$$\mathcal{J}(\mathbf{w}) = -\sum_{i=1}^{N}\Big(y^{(i)}\log\Big(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\Big) + (1-y^{(i)})\log\Big(1-\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\Big)\Big) \tag{3.46}$$

$$= -\sum_{i=1}^{N}\left(y^{(i)}\log\left(\frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right) + (1-y^{(i)})\log\left(1-\frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right)\right) \tag{3.47}$$

$$= -\sum_{i=1}^{N}\left(y^{(i)}\log\left(\frac{\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}{1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right) + (1-y^{(i)})\log\left(\frac{1}{1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right)\right) \tag{3.48}$$

$$= -\sum_{i=1}^{N}\Big(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}y^{(i)} - \log\Big(1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\Big)\Big) \tag{3.49}$$

$$\tag{3.50}$$

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

$$\frac{\partial\mathcal{J}(\mathbf{w})}{\partial\mathbf{w}} = -\sum_{i=1}^{N}\left(\mathbf{x}^{(i)}y^{(i)} - \mathbf{x}^{(i)}\cdot\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\cdot\frac{1}{1+\exp(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right) \tag{3.51}$$

$$= -\sum_{i=1}^{N}\left(\mathbf{x}^{(i)}y^{(i)} - \mathbf{x}^{(i)}\cdot\frac{1}{1+\exp(-\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})}\right) \tag{3.52}$$

$$= -\sum_{i=1}^{N}\Big(\mathbf{x}^{(i)}\cdot\Big(y^{(i)} - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})\Big)\Big) \tag{3.53}$$

$$= \sum_{i=1}^{N}\Big(\mathbf{x}^{(i)}\cdot\Big(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}) - y^{(i)}\Big)\Big) \tag{3.54}$$

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

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \lambda\frac{\partial\mathcal{R}(\mathbf{w})}{\partial\mathbf{w}}, \tag{3.55}$$

其中 λ 是学习率。

3.3.2 多类线性分类

对于多类分类问题（假设类别数为 $C(C>2)$），有很多种利用判别函数的方法。比较常用的是把多类分类问题转换为两类分类问题，在得到两类分类结果后，还需要通过投票方法进一步确定多类的分类结果。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。

当对一个样本进行分类时，首先用多个两个分类器进行分类，然后进行投票，选择一个得分最高的类别。

但是上面两种转换方法都存在一个缺陷：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。因为如果用多个两个分类器对这些点进行分类，然后进行投票时，会发现有两个或更多个类别的得分是一样的。

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：

$$f_c(\mathbf{x})=\mathbf{w}_c^{\mathrm{T}}\mathbf{x}, \qquad c=1,\cdots,C, \tag{3.56}$$

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：

$$\hat{y}=\arg\max_{c=1}^{C}\mathbf{w}_c^{\mathrm{T}}\mathbf{x} \tag{3.57}$$

当$C=2$时，

$$\begin{aligned}\hat{y} &= \arg\max_{y\in\{0,1\}} f_y(\mathbf{x}) \\ &= I(f_1(\mathbf{x})-f_0(\mathbf{x})>0) \\ &= I(\mathbf{w}_1^{\mathrm{T}}\mathbf{x}-\mathbf{w}_0^{\mathrm{T}}\mathbf{x}>0) \\ &= I((\mathbf{w}_1-\mathbf{w}_0)^{\mathrm{T}}\mathbf{x}>0)\end{aligned} \tag{3.58}$$

这里，$I()$是指示函数。对比公式3.60中的两类分类判别函数，可以发现$\mathbf{w}=\mathbf{w}_1-\mathbf{w}_0$。

SoftMax 回归

SoftMax 回归是 Logistic 回归的多类推广。

多类线性分类函数的参数$\mathbf{w}$也有很多种学习方式。这里我们介绍一种常用的学习算法：**SoftMax 回归**。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。这就需要用到**softmax**函数。

利用**softmax**函数，我们定义目标类别$y=c$的后验概率为：

$$P(y=c|\mathbf{x})=\mathbf{softmax}(\mathbf{w}_c^{\mathrm{T}}\mathbf{x})=\frac{\mathbf{w}_c^{\mathrm{T}}\mathbf{x}}{\exp(\sum_{i=1}^{C}\mathbf{w}_i^{\mathrm{T}}\mathbf{x})}. \tag{3.59}$$

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，

$$\mathbf{y}=[I(1=c),I(2=c),\cdots,I(C=c)]^{\mathrm{T}}, \tag{3.60}$$

这里，$I()$是指示函数。

同时，我们将公式3.59重新定义一下，直接输出k维向量。

$$\begin{aligned}\hat{\mathbf{y}}&=\mathbf{softmax}(W^{\mathrm{T}}\mathbf{x})\\&=\frac{\exp(W^{\mathrm{T}}\mathbf{x})}{\underline{1}^{\mathrm{T}}\exp\left((W^{\mathrm{T}}\mathbf{x})\right)}\\&=\frac{\exp(\mathbf{z})}{\underline{1}^{\mathrm{T}}\exp\left(\mathbf{z}\right)},\end{aligned} \tag{3.61}$$

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。

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：

$$\begin{aligned}\mathcal{J}(W)&=-\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbf{y}_c^{(i)}\log\hat{\mathbf{y}}_c^{(i)}\\&=-\sum_{i=1}^{N}\sum_{c=1}^{C}(\mathbf{y}^{(i)})^{\mathrm{T}}\log\hat{\mathbf{y}}^{(i)}\\&=-\sum_{i=1}^{N}(\mathbf{y}^{(i)})^{\mathrm{T}}\log\left(\mathbf{softmax}(\mathbf{z}^{(i)})\right)\\&=-\sum_{i=1}^{N}(\mathbf{y}^{(i)})^{\mathrm{T}}\log\left(\mathbf{softmax}(W^{\mathrm{T}}\mathbf{x}^{(i)})\right).\end{aligned} \tag{3.62}$$

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

$$\frac{\partial\mathbf{y}}{\partial\mathbf{x}}=\frac{\partial\,\mathbf{softmax}(\mathbf{x})}{\partial\mathbf{x}} \tag{3.63}$$

$$=\mathbf{diag}\left(\mathbf{softmax}(\mathbf{x})\right)-\mathbf{softmax}(\mathbf{x})\,\mathbf{softmax}(\mathbf{x})^{\mathrm{T}} \tag{3.64}$$

$$=\mathbf{diag}\left(\mathbf{y}\right)-\mathbf{y}\mathbf{y}^{\mathrm{T}}. \tag{3.65}$$

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

$$\frac{\partial \mathbf{z}}{\partial \mathbf{w}_c} = M(\mathbf{x}, c), \tag{3.66}$$

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

$\underset{\sim}{1}_K^{\mathrm{T}}\mathbf{y}^{(i)} = 1$

$\mathbf{x}^{\mathrm{T}}\,\mathbf{diag}(\mathbf{x})^{-1} = \underset{\sim}{1}_K^{\mathrm{T}}$

$\underset{\sim}{1}_K^{\mathrm{T}}\mathbf{y} = 1$

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

$$\begin{aligned}
\frac{\partial \mathcal{J}(W)}{\partial \mathbf{w}_c} &= -\sum_{i=1}^{N} \frac{\left((\mathbf{y}^{(i)})^{\mathrm{T}} \log\left(\mathbf{softmax}(\hat{\mathbf{y}}^{(i)})\right)\right)}{\partial \mathbf{w}_c} \\
&= -\sum_{i=1}^{N} \frac{\partial \mathbf{z}^{(i)}}{\partial \mathbf{w}_c} \frac{\partial\, \mathbf{softmax}(\mathbf{z}^{(i)})}{\partial \mathbf{z}^{(i)}} \frac{\partial \log \hat{\mathbf{y}}^{(i)}}{\partial \hat{\mathbf{y}}^{(i)}} \mathbf{y}^{(i)} \\
&= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left(\mathbf{diag}\left(\hat{\mathbf{y}}^{(i)}\right) - \hat{\mathbf{y}}^{(i)}(\hat{\mathbf{y}}^{(i)})^{\mathrm{T}}\right) \left(\mathbf{diag}(\hat{\mathbf{y}}^{(i)})\right)^{-1} \mathbf{y}^{(i)} \\
&= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left(I - \hat{\mathbf{y}}^{(i)}\underset{\sim}{1}_K^{\mathrm{T}}\right) \mathbf{y}^{(i)} \\
&= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left((\mathbf{y}^{(i)}) - \hat{\mathbf{y}}^{(i)}\underset{\sim}{1}_K^{\mathrm{T}}\mathbf{y}^{(i)}\right) \\
&= -\sum_{i=1}^{N} M(\mathbf{x}^{(i)}, c) \left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\right) \\
&= -\sum_{i=1}^{N} \mathbf{x}^{(i)} \left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\right)_c
\end{aligned} \tag{3.67}$$

如果采用$y \in \{1, \cdots, C\}$的形式，梯度公式也可以写为：

$$\frac{\partial \mathcal{J}(W)}{\partial \mathbf{w}_c} = -\frac{1}{N}\sum_{i=1}^{C} \mathbf{x}^{(i)} \left(1\{y^{(i)} = c\} - p(y^{(i)} = c|\mathbf{x}^{(i)}; W)\right) \tag{3.68}$$

根据公式3.3.2，我们可以得到

$$\frac{\partial \mathcal{J}(W)}{\partial W} = -\sum_{i=1}^{N} \mathbf{x}^{(i)} \left(\mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)}\right)^{\mathrm{T}} \tag{3.69}$$

我们可以初始化$W=0$，然后用梯度下降法进行更新

$$W_{t+1}=W_t+\lambda-\sum_{i=1}^{N}\mathbf{x}^{(i)}\left(\mathbf{y}^{(i)}-\hat{\mathbf{y}}^{(i)}\right)^{\mathrm{T}}, \tag{3.70}$$

其中λ是学习率。

3.4 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：

$$Acc=\frac{\sum_{i=1}^{N}|y_i=\hat{y_i}|}{N} \tag{3.71}$$

其中，$|\cdot|$为指示函数

和正确率相对应的就是**错误率**（Error Rate）。

$$Err=\frac{\sum_{i=1}^{N}|y_i\neq\hat{y_i}|}{N} \tag{3.72}$$

正确率是平均的整体性能。

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，

$$P_c=\frac{\sum_{\substack{i=1\\ \hat{y_i}=c}}^{N}|y_i=\hat{y_i}|}{\sum_{\substack{i=1\\ \hat{y_i}=c}}^{N}1} \tag{3.73}$$

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。

$$R_c = \frac{\sum\limits_{\substack{i=1 \\ y_i=c}}^{N} |y_i = \hat{y}_i|}{\sum\limits_{\substack{i=1 \\ y_i=c}}^{N} 1} \tag{3.74}$$

F1 值是根据正确率和召回率二者给出的一个综合的评价指标，具体定义如下：

$$F1_c = \frac{P_c * R_c * 2}{(P_c + R_c)} \tag{3.75}$$

为了计算分类算法在整个数据集上的总体准确率、召回率和F1值，经常使用两种平均方法，分别称为**宏平均**（macro average）和**微平均**（micro average）。

宏平均是每一个类的性能指标的算术平均值，

$$R_{macro} = \sum_{i=1}^{C} R_c/C, \tag{3.76}$$

$$P_{macro} = \sum_{i=1}^{C} P_c/C, \tag{3.77}$$

$$F1_{macro} = \frac{P_{macro} * R_{macro} * 2}{(P_{macro} + R_{macro})}. \tag{3.78}$$

而微平均是每一个样本的性能指标的算术平均。对于单个样本而言，它的准确率和召回率是相同的（要么都是1，要么都是0）因此准确率和召回率的微平均是相同的，根据F1 值公式，对于同一个数据集它的准确率、召回率和F1的微平均指标是相同的。

3.5 总结和深入阅读

本章简单地介绍了机器学习的理论知识，主要为后面讲解人工神经网络铺垫一些基础知识。如果需要快速全面地了解机器学习的基本概念可以阅读《Pattern Classification》[Duda et al., 2001]和《Pattern Recognition and Machine Learning》[Bishop, 2006]，进一步深入了解可以阅读《The Elements of Statistical Learning》[Hastie et al., 2001] 以及《Learning in Graphical Models》[Jordan, 1998]。

第四章　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。

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。

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。

下面我们来介绍下感知器模型和学习算法。

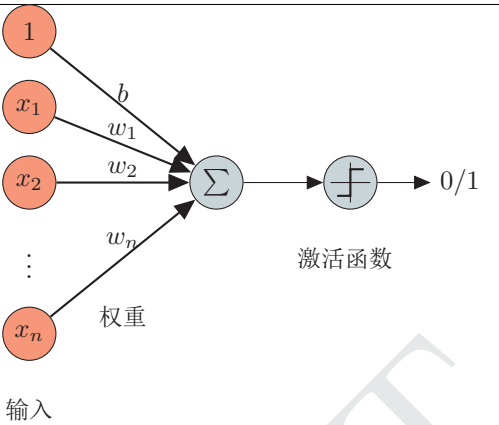


图 4.1: 感知器模型

4.1 两类感知器

图5.1给出了感知器模型的结构。

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

$$\hat{y} = \begin{cases} +1 & \text{当} \mathbf{w}^T\mathbf{x} + b > 0 \\ -1 & \text{当} \mathbf{w}^T\mathbf{x} + b \leq 0 \end{cases}, \tag{4.1}$$

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4.1可以简写为：

$$\hat{y} = \begin{cases} +1 & \text{当} \mathbf{w}^T\mathbf{x} > 0 \\ -1 & \text{当} \mathbf{w}^T\mathbf{x} \leq 0 \end{cases}, \tag{4.2}$$

接下来我们看下感知器是如何学习的。

4.1.1 感知器学习算法

给定 N 个样本的训练集: $(\mathbf{x}_i, y_i), i = 1, \cdots, N$。我们希望学习到参数 $\mathbf{w}*$，使得

$$\begin{aligned} &\mathbf{w}^{*T}\mathbf{x}_i > 0 \quad \text{当} y_i > 0, \\ &\mathbf{w}^{*T}\mathbf{x}_i < 0 \quad \text{当} y_i < 0. \end{aligned} \tag{4.3}$$

公式4.3等价于 $\mathbf{w}^{*T}(y_i\mathbf{x}_i) > 0$。

Rosenblatt [1958] 首次提出了感知器的学习算法。这个算法是错误驱动的在线学习算法。先初始化一个权重向量 $\mathbf{w}_0$（通常是全零向量），然后每次分错一个样本时，就用这个样本来更新权重。具体的学习过程如算法4.1所示。

算法 4.1: 两类感知器算法

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 $\mathbf{w}_k$;

4.1.2 收敛性证明

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。

定义 4.1 – 两类线性可分： 对于训练集 $\mathcal{D} = \{(x_i, y_i) \mid y_i \in \{-1, 1\}\}_{i=1}^n$，如果存在一个正的常数 $\gamma(\gamma > 0)$ 和权重向量 $\mathbf{w}^*$，并且 $\|\mathbf{w}^*\| = 1$，对所有 i 都满足 $(\mathbf{w}^*)^T(y_i\mathbf{x}_i) > \gamma$ （$\mathbf{x}_i \in \mathbb{R}^m$ 为样本 x_i 的增广特征向量），那么训练集 $\mathcal{D}$ 是线性可分的。

在数据集是两类线性可分的条件下，我们可以证明如下定理。

定理 4.1–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4.1中，总共的预测错误次数 $K < \frac{R^2}{\gamma^2}$。

证明如下：

权重向量的更新公式为：

$$\mathbf{w}_k = \mathbf{w}_{k-1} + y_k \mathbf{x}_k, \tag{4.4}$$

这里，$\mathbf{x}_k, y_k$ 为第 k 个错误分类的样本。

因为初始权重向量为0，因此在第 K 次更新时，

$$\mathbf{w}_K = \sum_{k=1}^{K} y_k \mathbf{x}_k. \tag{4.5}$$

那么，

（1）$\|\mathbf{w}_K\|^2$ 的上界为：

$$\begin{aligned} \|\mathbf{w}_K\|^2 &= \|\sum_{k=1}^{K} y_k \mathbf{x}_k\|^2 \\ &\leq \sum_{k=1}^{K} \|y_k \mathbf{x}_k\|^2 \\ &\leq \sum_{k=1}^{K} \max_{k=1}^{K} \|y_k \mathbf{x}_k\|^2 \\ &\leq K \cdot R^2 \end{aligned} \tag{4.6}$$

（2）我们再来看下 $\|\mathbf{w}_K\|^2$ 的下界。

首先，因为两个向量内积的平方一定小于等于这两个向量的模的乘积。

$$\|\mathbf{w}^{*T} \mathbf{w}_K\|^2 \leq \|\mathbf{w}^*\|^2 \cdot \|\mathbf{w}_K\|^2 = \|\mathbf{w}_K\|^2. \tag{4.7}$$

$\|\mathbf{w}^*\| = 1$

因此，

$$
\begin{aligned}
\|\mathbf{w}_K\|^2 &\geq \|\mathbf{w}^{*T}\mathbf{w}_K\|^2 \\
&= \|\mathbf{w}^{*T}\sum_{k=1}^{K}(y_k\mathbf{x}_k)\|^2 \\
&= \|\sum_{k=1}^{K}\mathbf{w}^{*T}(y_k\mathbf{x}_k)\|^2 \\
&\geq K^2\gamma^2 \quad \leftarrow \boxed{\mathbf{w}^{*T}(y_i\mathbf{x}_i) > \gamma, \forall i}
\end{aligned} \tag{4.8}
$$

由公式4.6和4.8，得到

$$K^2\gamma^2 \leq ||\mathbf{w}_K||^2 \leq K \times R^2 \tag{4.9}$$

取最左和最右的两项，进一步得到，$K^2\gamma^2 \leq K \cdot R^2$。然后两边都除$K$，最终得到

$$K \leq \frac{R^2}{\gamma^2}. \tag{4.10}$$

因此，在线性可分的条件下，算法4.1会在$\frac{R^2}{\gamma^2}$步内收敛。如果训练集不s是线性可分的，就永远不会收敛[Freund and Schapire, 1999]。

4.2 多类感知器

原始的感知器的输出是0或1，不能提供概率形式的输出，因此只能处理两类问题。为了使得感知器能处理多类问题以及更复杂的结构化学习任务，我们引入一个特征函数$\boldsymbol{\phi}(x, y)$将输入输出对映射到一个向量空间中[Collins, 2002]。这样，我们可以得到一个更为泛化的感知器：

$$\hat{y} = \arg\max_{y \in \texttt{Gen}(\mathbf{x})} \mathbf{w}^T\boldsymbol{\phi}(x, y), \tag{4.11}$$

这里$\texttt{Gen}(x)$表示输入x所有的输出目标集合。当处理C类分类问题时，$\texttt{Gen}(x) = \{1, \cdots, C\}$。

在上面几节中，分类函数都是在输入x的向量空间上。通过引入特征函数$\boldsymbol{\phi}(x, y)$，感知器不但可以用于多类分类问题，也可以用于结构化学习问题，比如输出是序列或来其它结构化的形式。

当y为离散变量时（$y \in \{1, \cdots, C\}$），类别也可以表示为向量：

$$\boldsymbol{\phi}(y=c) = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \boxed{1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{第}\, c \,\text{行}$$

如果每个样本可以有多个类别，标签分类$y = \{c, k\}$时，类别可以表示为向量：

$$\boldsymbol{\phi}(y=\{c,k\}) = \begin{bmatrix} 0 \\ \vdots \\ \boxed{1} \\ \vdots \\ \boxed{1} \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \\ \leftarrow \text{第}\, c \,\text{行} \\ \\ \leftarrow \text{第}\, k \,\text{行} \\ \\ \end{matrix}$$

$\boldsymbol{\phi}(x, y)$可以看成是$\boldsymbol{\phi}(x)$和$\boldsymbol{\phi}(y)^T$的乘积得到矩阵的向量化。

$$\boldsymbol{\phi}(x, y=c) = \mathbf{vec}(\boldsymbol{\phi}(x)\boldsymbol{\phi}(y)^T), \tag{4.12}$$

这里，**vec**是向量化算子。设$A = [a_{ij}]_{m \times n}$，则

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

$$\boldsymbol{\phi}(x, y=c) = \begin{bmatrix} \vdots \\ 0 \\ \boxed{\phi(x)} \\ 0 \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ 0 \\ \boxed{\begin{matrix} \phi_1(x) \\ \vdots \\ \phi_m(x) \end{matrix}} \\ 0 \\ \vdots \end{bmatrix} \tag{4.13}$$

多类感知器算法的训练过程如算法4.2所示。

算法 4.2: 多类感知器算法

输入: 训练集: $(x_i, y_i), i = 1, \cdots, N$，最大迭代次数：$T$

输出: $\mathbf{w}_k$

$\mathbf{w}_0 = 0$;
$k = 0$;
for $t = 1 \cdots T$ **do**
  for $i = 1 \cdots N$ **do**
    选取一个样本 (x_i, y_i);
    用公式4.11计算预测类别 $\hat{y}_i$;
    if $\hat{y}_i \neq y_i$ **then**
      $\mathbf{w}_{k+1} = \mathbf{w}_k + (\boldsymbol{\phi}(x_i, y_i) - \boldsymbol{\phi}(x_i, \hat{y}_i))$;
      $k = k + 1$;
    end
  end
end
return $\mathbf{w}_k$;

4.2.1 多类感知器的收敛性

多类分类时，感知器的收敛情况。Collins [2002] 给出了多类感知器在多类线性可分的收敛性证明，具体推导过程和两类分类器比较类似。

定义 4.2–多类线性可分： 对于训练集 $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$，如果存在一个正的常数 $\gamma(\gamma > 0)$ 和权重向量 $\mathbf{w}^*$，并且 $\|\mathbf{w}^*\| = 1$，对所有 i 都满足 $\langle \mathbf{w}^*, \boldsymbol{\phi}(x_i, y_i)\rangle - \langle \mathbf{w}^*, \boldsymbol{\phi}(x_i, y)\rangle > \gamma, y \neq y_i$ （$\boldsymbol{\phi}(x_i, y_i) \in \mathbb{R}^m$ 为样本 x_i, y_i 的特征向量），那么训练集 $\mathcal{D}$ 是线性可分的。

定理 4.2–多类感知器收敛性： 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**错误类别**和**真实类别**在特征空间 $\boldsymbol{\phi}(x, y)$ 最远的距离。

$$R = \max_i \max_{z \neq y_i} \|\boldsymbol{\phi}(x_i, y_i) - \boldsymbol{\phi}(x_i, z)\|.$$

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

4.3 投票感知器

从定理4.2可以看出，如果训练数据是线性可分的，那么感知器可以找到一个判别函数来分割不同类的数据。并且如果边际距离越大，收敛越快。但是，感知器并不能保证找到的判别函数是最优的（比如泛化能力高），这样可能导致过拟合。

此外，从感知器的迭代算法可以看出：在迭代次序上排在后面的错误点，比前面的错误点对最终的权重向量影响更大。比如有1,000个训练样本，在迭代100个样本后，感知器已经学习到一个很好的权重向量。在接下来的899个样本上都预测正确，也没有更新权重向量。但是在最后第1,000个样本时预测错误，并更新了权重。这次更新可能反而使得权重向量变差了。

为了这种情况，可以使用“参数平均”的策略来提高感知器的鲁棒性，也叫**投票感知器**（Voted Perceptron）[Freund and Schapire, 1999]。

投票感知器记录第 k 次更新后得到的权重 $\mathbf{w}_k$ 在其后分类中正确分类样本的次数 c_k。这样最后的分类器形式为（假设输出为0或1）：

$$\hat{y} = \mathbf{sign}(\sum_{k=1}^{K} c_k \, \mathbf{sign}(\mathbf{w}_k^T x)). \tag{4.14}$$

投票感知器虽然提高了感知器的泛化能力，但是需要保存 K 个权重向量。在实际操作中会带来额外的开销。因此，人们经常会使用一个简化的版本，也叫做**平均感知器**（Averaged Perceptron）[Collins, 2002]。

$$\begin{aligned}\hat{y} &= \mathbf{sign}(\sum_{k=1}^{K} c_k(\mathbf{w}_k^T x)) \\ &= \mathbf{sign}(\sum_{k=1}^{K} c_k \mathbf{w}_k)^T x) \\ &= \mathbf{sign}(\bar{\mathbf{w}}^T x),\end{aligned} \tag{4.15}$$

其中，$\bar{\mathbf{w}}$ 为平均的权重向量。

假设 $\mathbf{w}^{t,i}$ 是在第 t 轮更新到第 i 个样本时权重向量的值，平均的权重向量 $\bar{\mathbf{w}}$ 也可以写为

$$\bar{\mathbf{w}} = \frac{\sum_{t=1}^{T}\sum_{i=1}^{n} \mathbf{w}^{t,i}}{nT} \tag{4.16}$$

这个方法非常简单，只需要在算法4.2中增加一个$\bar{\mathbf{w}}$，并且在处理每一个样本后，更新

$$\bar{\mathbf{w}} = \bar{\mathbf{w}} + \mathbf{w}^{t,i} \tag{4.17}$$

这里要注意的是，$\bar{\mathbf{w}}$需要在处理每一个样本时都需要更新，并且$\bar{\mathbf{w}}$和$\mathbf{w}^{t,i}$都是稠密向量。因此，这个操作比较费时。为了提高迭代速度，有很多改进的方法，让这个更新只需要在错误预测发生时才进行更新。

算法4.3给出了一个改进的平均感知器算法的训练过程 [Daumé III]。

算法 4.3: 平均感知器算法

输入: 训练集: $(x_i, y_i), i = 1, \cdots, N$，最大迭代次数：$T$
输出: $\bar{\mathbf{w}}$

$\mathbf{w} = 0$;
$\mathbf{u} = 0$;
$c = 0$;
for $t = 1 \cdots T$ **do**
  for $i = 1 \cdots N$ **do**
    选取一个样本 (x_i, y_i);
    用公式4.11计算预测类别 $\hat{y}_t$;
    if $\hat{y}_t \neq y_t$ **then**
      $\mathbf{w} = \mathbf{w} + ((x_t, y_t) - (x_t, \hat{y}_t))$;
      $\mathbf{u} = \mathbf{u} + c \cdot ((x_t, y_t) - (x_t, \hat{y}_t))$;
    end
    $c = c + 1$;
  end
end
$\bar{\mathbf{w}} = \mathbf{w}_T - \frac{1}{c}\mathbf{u}$;
return $\bar{\mathbf{w}}$;

4.4 总结和深入阅读

Rosenblatt [1958] 最早提出了两类感知器算法，并随后给出了感知机收敛定理。但是感知器的输出是离散的以及学习算法比较简单，不能解决线性不可分问题，限制了其应用范围。Minsky and Seymour [1969] 分析了感知机的局限性，证明感知机不能解决非常

简单的异或（XOR）问题。虽然他也认为多层的网络可以解决非线性问题，但是遗憾的是，在当时这个问题还不可解。直到1980年以后，Geoffrey Hinton、Yann LeCun等人用连续输出代替离散的输出，并将**反向传播算法**（Backpropagation，BP）[Werbos, 1974]引入到多层感知器[Williams and Hinton, 1986]，人工神经网络才又重新引起人们的注意。Minsky and Papert [1987]也修正之前的看法。

另外一方面，人们对感知器本身的认识也在不断发展。Freund and Schapire [1999]提出了使用核技巧改进感知器学习算法，并用投票感知器来提高泛化能力。Collins [2002]将感知器算法扩展到结构化学习，给出了相应的收敛性证明，并且提出一种更加有效并且实用的参数平均化策略。[McDonald et al., 2010]有扩展了平均感知器算法，使得感知器可以在分布式计算环境中并行计算，这样感知器可以用在大规模机器学习问题上。

第五章　人工神经网络

人工神经网络 1943 年，心理学家 W.S.McCulloch 和数理逻辑学家 W.Pitts 建立了神经网络和数学模型，称为 MP 模型。他们通过 MP 模型提出了神经元的形式化数学描述和网络结构方法，证明了单个神经元能执行逻辑功能，从而开创了人工神经网络研究的时代。1949 年，心理学家提出了突触联系强度可变的设想。60 年代，人工神经网络得到了进一步发展，更完善的神经网络模型被提出人工神经网络人工神经网络，其中包括感知器和自适应线性元件等。

前馈神经网络也经常称为**多层感知器**（Multilayer Perceptron，MLP）。但多层感知器的叫法并不是否合理，因为前馈神经网络其实是由多层的 logistic 回归模型（连续的非线性函数）组成，而不是有多层的感知器（不连续的非线性函数）组成 [Bishop, 2006]。

人工神经网络（Artificial Neural Network，即 ANN ），是 20 世纪 80 年代以来人工智能领域兴起的研究热点。它从信息处理角度对人脑神经元网络进行抽象，建立某种简单模型，按不同的连接方式组成不同的网络。在工程与学术界也常直接简称为神经网络或类神经网络。神经网络是一种运算模型，由大量的节点（或称神经元）之间相互联接构成。每个节点代表一种特定的输出函数，称为激励函数（activation function）。每两个节点间的连接都代表一个对于通过该连接信号的加权值，称之为权重，这相当于人工神经网络的记忆。网络的输出则依网络的连接方式，权重值和激励函数的不同而不同。而网络自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。最近十多年来，人工神经网络的研究工作不断深入，已经取得了很大的进展，其在模式识别、智能机器人、自动控制、预测估计、生物、医学、经济等领域已成功地解决了许多现代计算机难以解决的实际问题，表现出了良好的智能特性。

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，其中有反传网络、感知器、自组织映射、Hopfield 网络、波耳兹曼机、适应谐振理论等。根据连接的拓扑结构，神经网络模型可以分为：[1]

前向网络

网络中各个神经元接受前一级的输入，并输出到下一级，网络中没有反馈，可以用一

个有向无环路图表示。这种网络实现信号从输入空间到输出空间的变换，它的信息处理能力来自于简单非线性函数的多次复合。网络结构简单，易于实现。反传网络是一种典型的前向网络。[2] 反馈网络

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。系统的稳定性与联想记忆功能有密切关系。Hopfield网络、波耳兹曼机均属于这种类型。

5.1 神经元

人工神经元（Neuron）是构成人工神经网络的基本单元。人工神经元和感知器非常类似，也是模拟生物神经元特性，接受一组输入信号并产出输出。生物神经元有一个阀值，当神经元所获得的输入信号的积累效果超过阀值时，它就处于兴奋状态；否则，应该处于抑制状态。

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：

$$z = \mathbf{w}^T\mathbf{x} + b \tag{5.1}$$

$$a = f(z) \tag{5.2}$$

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。

人工神经元的结构如图**??**所示。如果我们设激活函数f为0或1的阶跃函数，人工神经元就是感知器。

5.1.1 激活函数

为了增强网络的表达能力，我们需要引入连续的非线性**激活函数**（Activation Function）。因为连续非线性激活函数可以可导的，所以可以用最优化的方法来求解。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。

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.3}$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5.4}$$

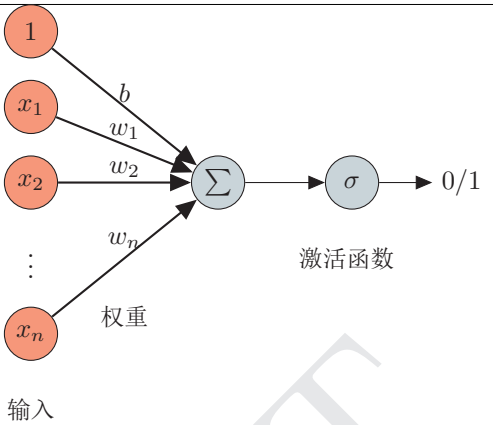


图 5.1: 感知器模型

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

sigmoid 型函数对中间区域的信号有增益，对两侧区的信号有抑制。这样的特点也和生物神经元类似，对一些输入有兴奋作用，另一些输入（两侧区）有抑制作用。和感知器的阶跃激活函数 $(-1/1, 0/1)$ 相比，sigmoid 型函数更符合生物神经元的特性，同时也有更好的数学性质很好。

除了 sigmoid 型函数，在神经网络中还有一些其它的非线性激活函数。

一种是 rectifier 函数 [Glorot et al., 2011]，定义为

$$\textbf{rectifier}(x) = \max(0, x) \tag{5.5}$$

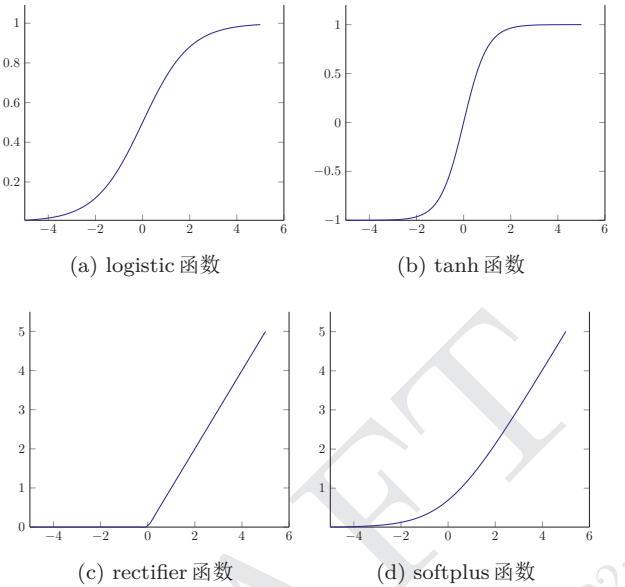
rectifier 函数被认为有生物上的解释性。神经科学家发现神经元具有单侧抑制、宽兴奋边界、稀疏激活性等特性。神经元只对少数输入信号选择性响应，大量信号被刻意的屏蔽了。因为随机初始化的原因，sigmoid 系函数同时近乎有 50% 的神经元被激活，这不符合神经科学的研究，而且会给深度网络训练带来巨大问题。而 rectifier 函数却具备很好的稀疏性，并且整个网络只有比较、加、乘操作，计算上也更加高效。使用 rectifier 函数的深度神经网络可以不需要非监督的预训练过程。

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]。

另一种是 softplus 函数 [Dugas et al., 2001]，定义为

$$\textbf{softplus}(x) = \log(1 + e^x) \tag{5.6}$$

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。



(a) logistic 函数 (b) tanh 函数

(c) rectifier 函数 (d) softplus 函数

图 5.2: 激活函数

图5.2给出了 $\sigma(x)$ 和 tanh 两种函数的形状。

5.2 前馈神经网络

给定一组神经元，我们可以以神经元为节点来构建一个网络。不同的神经网络模型有着不同网络连接的拓扑结构。一种比较直接的拓扑结构是前馈网络。**前馈神经网络**（Feed-forward Neural Network）是最早发明的简单人工神经网络。

各神经元分别属于不同的层。每一层的神经元可以接收前一层神经元的信号，并产生信号输出到下一层。第一层叫**输入层**，最后一层叫**输出层**，其它中间层叫做**隐藏层**。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5.3给出了前馈神经网络的示例。

5.2.1 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；

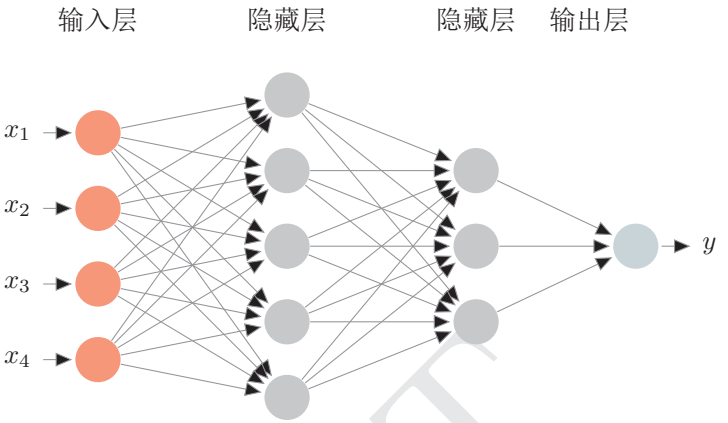


图 5.3: 多层神经网络

- 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。

$$\mathbf{z}^{(l)} = W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} \tag{5.7}$$

$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)}) \tag{5.8}$$

公式5.7和5.8也可以合并写为：

$$\mathbf{z}^{(l)} = W^{(l)} \cdot f_l(\mathbf{z}^{(l-1)}) + \mathbf{b}^{(l)} \tag{5.9}$$

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 $\mathbf{a}^L$。

$$\mathbf{x} = \mathbf{a}^{(0)} \rightarrow \mathbf{z}^{(1)} \rightarrow \mathbf{a}^{(1)} \rightarrow \mathbf{z}^{(2)} \rightarrow \cdots \rightarrow \mathbf{a}^{(L-1)} \rightarrow \mathbf{z}^{(L)} \rightarrow \mathbf{a}^{(L)} = y \tag{5.10}$$

5.3 反向传播算法

在了解前馈网络的结构之后，我们可以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：

$$J(W, \mathbf{b}) = \sum_{i=1}^{N} L(y^{(i)}, f(\mathbf{x}^{(i)}|W, \mathbf{b})) + \frac{1}{2}\lambda\|W\|_F^2, \tag{5.11}$$

$$= \sum_{i=1}^{N} J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)}) + \frac{1}{2}\lambda\|W\|_F^2, \tag{5.12}$$

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

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：

$$W^{(l)} = W^{(l)} - \alpha\frac{\partial J(W, \mathbf{b})}{\partial W^{(l)}}, \tag{5.13}$$

$$= W^{(l)} - \alpha\sum_{i=1}^{N}(\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial W^{(l)}}) - \lambda W, \tag{5.14}$$

$$\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial \mathbf{b}^{(l)}}, \tag{5.15}$$

$$= \mathbf{b}^{(l)} - \alpha\sum_{i=1}^{N}(\frac{\partial J(W, \mathbf{b}; \mathbf{x}^{(i)}, y^{(i)})}{\partial \mathbf{b}^{(l)}}), \tag{5.16}$$

$$\tag{5.17}$$

这里 α 是参数的更新率。

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

根据链式法则，$\frac{\partial J(W,\mathbf{b};\mathbf{x},y)}{\partial W^{(l)}}$ 可以写为

$$\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial W_{ij}^{(l)}} = \mathbf{tr}\left(\left(\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l)}}\right)^T \frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}}\right). \tag{5.18}$$

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。

公式5.18的第一项可以暂记为$\delta^{(l)}$，稍后再分析。我们先来看第二项。因为$\mathbf{z}^{(l)} = W^{(l)}$.

$\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$，所以

$$\frac{\partial \mathbf{z}^{(l)}}{\partial W_{ij}^{(l)}} = \frac{\partial (W^{(l)} \cdot \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)})}{\partial W_{ij}^{(l)}} = \begin{bmatrix} 0 \\ \vdots \\ a_j^{(l-1)} \\ \vdots \\ 0 \end{bmatrix} \leftarrow \text{第 } i \text{ 行} \tag{5.19}$$

因此，

$$\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial W_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)} \tag{5.20}$$

$$\tag{5.21}$$

公式5.18可以写为

$$\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial W^{(l)}} = \delta^{(l)} (\mathbf{a}^{(l-1)})^T. \tag{5.22}$$

同理可得，

$$\frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}. \tag{5.23}$$

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

$$\delta^{(l)} \triangleq \frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l)}} \tag{5.24}$$

$$= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l+1)}} \tag{5.25}$$

$$= \mathbf{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^T \cdot \delta^{(l+1)} \tag{5.26}$$

$$= f_l'(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^T \delta^{(l+1)}), \tag{5.27}$$

其中 $\odot$ 是向量的点积运算符，表示每个元素相乘。

上述推导中，关键是公式5.25到公式5.26的推导。公式5.25中有三项，第三项根据定义为 $\delta^{(l+1)}$。

第二项因为 $\mathbf{z}^{(l+1)} = W^{(l+1)} \cdot \mathbf{a}^{(l)} + \mathbf{b}^{(l)}$，所以

$$\frac{\partial \mathbf{z}^{(l+1)}}{\partial a^{(l)}} = (W^{(l+1)})^T. \tag{5.28}$$

第一项因为$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)})$，而$f_l(\cdot)$为按位计算的函数。因此

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \frac{\partial f_l(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}} \tag{5.29}$$

$$= \mathbf{diag}(f_l'(\mathbf{z}^{(l)})). \tag{5.30}$$

从公式5.27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5.1所示，也叫**反向传播算法**。

算法 5.1: 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：$T$

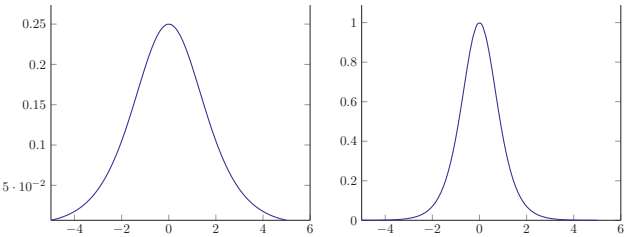
输出: $W, \mathbf{b}$

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5.27反向传播计算每一层的误差$\delta^{(l)}$;
    (3) 用公式5.22和5.23每一层参数的导数;
      $\frac{\partial J(W,\mathbf{b};\mathbf{x}^{(i)},y^{(i)})}{\partial W^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^T$;
      $\frac{\partial J(W,\mathbf{b};\mathbf{x}^{(i)},y^{(i)})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}$;
    (4) 更新参数;
      $W^{(l)} = W^{(l)} - \alpha \sum_{i=1}^{N} (\frac{\partial J(W,\mathbf{b};\mathbf{x}^{(i)},y^{(i)})}{\partial W^{(l)}}) - \lambda W$;
      $\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha \sum_{i=1}^{N} (\frac{\partial J(W,\mathbf{b};\mathbf{x}^{(i)},y^{(i)})}{\partial \mathbf{b}^{(l)}})$;
  end
end

5.4 梯度消失问题

在神经网络中误差反向传播的迭代公式为

$$\delta^{(l)} = f_l'(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^T \delta^{(l+1)}, \tag{5.31}$$



(a) logistic 函数的导数 (b) tanh 函数的导数

图 5.4: 激活函数的导数

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

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \in [0, 0.25] \tag{5.32}$$

$$\tanh'(x) = 1 - (\tanh(x))^2 \in [0, 1]. \tag{5.33}$$

我们可以看到，sigmoid 型函数导数的值域都小于 1。这样误差经过每一层传递都会不断衰减。当网络层数很深时，梯度就会不停的衰减，甚至消失，使得整个网络很难训练。这就是所谓的**梯度消失问题**（Vanishing Gradient Problem），也叫**梯度弥散**。

减轻梯度消失问题的一个方法是使用线性激活函数（比如 rectifier 函数）或近似线性函数（比如 softplus 函数）。这样，激活函数的导数为 1，误差可以很好地传播，训练速度得到了很大的提高。

5.5 训练方法

批量

随机梯度下降

minibatch

5.6 经验

超参

初始化

5.7 总结和深入阅读

Anderson and Rosenfeld [2000]

第六章　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]。

6.1　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。

6.1.1　一维场合

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：

$$y_t = \sum_{k=1}^{n} f_k \cdot x_{t-k+1}. \tag{6.1}$$

当滤波器$f_t = 1/n$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**。

6.1.2 两维场合

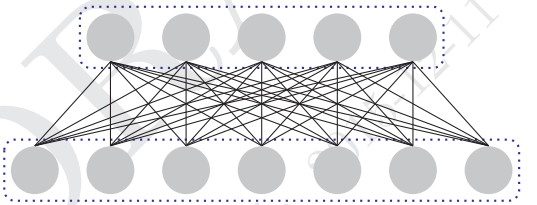
一维卷积经常用在图像处理中。给定一个图像 $x_{ij}, 1 \leq i \leq M, 1 \leq j \leq N$，和滤波器 $f_{ij}, 1 \leq i \leq m, 1 \leq j \leq n$，一般 $m << M, n << N$。

卷积的输出为：

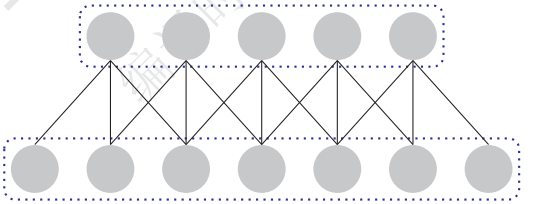
$$y_{ij} = \sum_{u=1}^{m} \sum_{v=1}^{n} f_{uv} \cdot x_{i-u+1, j-v+1}. \tag{6.2}$$

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 $f_{uv} = \frac{1}{mn}$。

6.2 卷积层：用卷积来代替全连接



(a) 全连接层



(b) 卷积层

图 6.1: 全连接层和卷积层

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：

$$a_i^{(l)} = f(\sum_{j=1}^{m} w_j^{(l)} \cdot a_{i-j+m}^{(l-1)} + b^{(l)}), \tag{6.3}$$

$$= f(\mathbf{w}^{(l)} \cdot \mathbf{a}_{(i+m-1):i}^{(l-1)} + b_i), \tag{6.4}$$

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。

上述公式也可以写为：

$$\mathbf{a}^{(l)} = f(\mathbf{w}^{(l)} \otimes \mathbf{a}^{(l-1)} + b^{(l)}), \tag{6.5}$$

$\otimes$表示卷积运算。

从公式6.5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$n^{(l+1)} = n^{(l)} - m + 1$。

上面是一维的卷积层，下面我们来看下两维的情况。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：

$$X_{s,t}^{(l)} = f\left(\sum_{i=1}^{u}\sum_{j=1}^{v} W_{i,j}^{(l)} \cdot X_{s-i+u,t-j+v}^{(l-1)} + b^{(l)}\right), \tag{6.6}$$

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：

$$X^{(l)} = f\left(W^{(l)} \otimes X^{(l-1)} + b^{(l)}\right), \tag{6.7}$$

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）。

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。

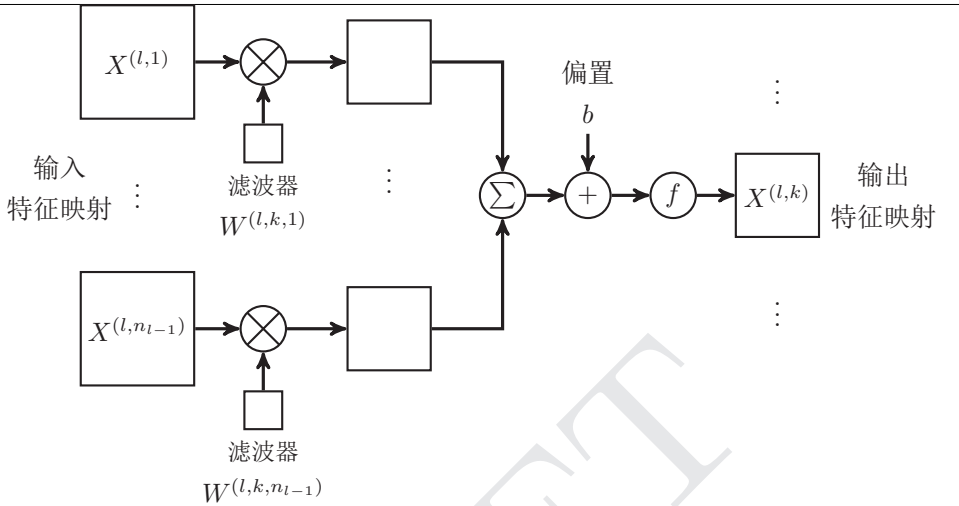


图 6.2: 两维卷积层的映射关系

第l层的第k组特征映射$X^{(l,k)}$为：

$$X^{(l,k)} = f\left(\sum_{p=1}^{n_{l-1}} \left(W^{(l,k,p)} \otimes X^{(l-1,p)}\right) + b^{(l,k)}\right), \tag{6.8}$$

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。

第l层的每一组特征映射都需要n_{l-1}个滤波器以及一个偏置b。假设每个滤波器的大小为$u \times v$，那么共需要$n_l \times n_{l-1} \times (u \times v) + n_l$。

这样，我们在第$l+1$层就得到n_l组特征映射，每一组特征映射的大小为$m_l = w_{l-1} - u + 1 \times h_{l-1} - v + 1$，总的神经元个数为$n_l \times m_l$。图6.2给出了公式6.8的可视化映射关系。

连接表 公式6.8中，第$l-1$层的所有特征映射都经过滤波器得到一个第l层的一组特征映射$X^{(l,k)}$。也就是说，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。

$$X^{(l,k)} = f\left(\sum_{\substack{p, \\ T_{p,k}=1}} \left(W^{(l,k,p)} \otimes X^{(l,p)}\right) + b^{(l,k)}\right) \tag{6.9}$$

每一张feature map的生成，都对应着特定的上一层的feature map（连接表决定），以及过滤器（W_lkp）。

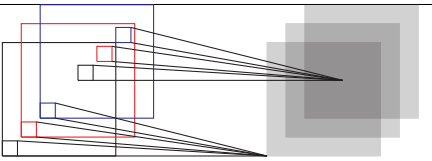


图 6.3: 两维卷积层

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。

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。图6.3给出了两维卷积层示例。

6.3 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。

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：

$$X_k^{(l+1)} = f(Z_k^{(l+1)}), \tag{6.10}$$

$$= f\left(w^{(l+1)} \cdot \mathbf{down}(R_k) + b^{(l+1)}\right), \tag{6.11}$$

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

$$X^{(l+1)} = f(Z^{(l+1)}), \tag{6.12}$$

$$= f\left(w^{(l+1)} \cdot \mathbf{down}(X^l) + b^{(l+1)}\right), \tag{6.13}$$

$\mathbf{down}(X^l)$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）。

$$pool_{max}(R_k) = \max_{i \in R_k} a_i \tag{6.14}$$

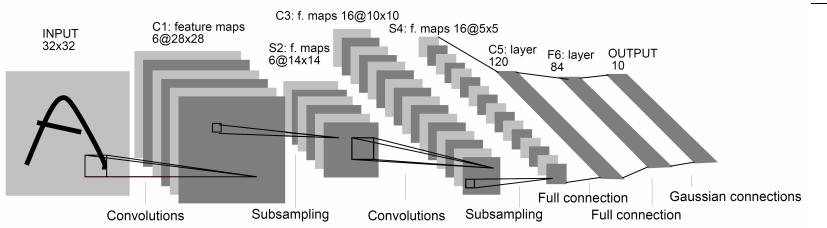


图 6.4: LeNet-5 网络结构。图片来源：[LeCun et al., 1998]

$$pool_{avg}(R_k) = \frac{1}{|R_k|} \sum_{i \in R_k} a_i. \tag{6.15}$$

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。

6.4 卷积神经网络示例：LeNet-5

下面我们来看一个具体的深层卷积神经网络：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6.4所示。

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）
4. 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6.5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

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

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 $1,516 \times 100 = 151,600$。

5. 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 $16 \times (4 + 1) = 2000$。

6. 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 $120 \times (16 \times 25 + 1) = 48,120$。

7. 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 $10,164$。

8. 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。

6.5 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

$$\delta^{(l)} \triangleq \frac{\partial J(W, \mathbf{b}; \mathbf{x}, y)}{\partial \mathbf{z}^{(l)}} \tag{6.16}$$

$$= f_l'(\mathbf{z}^{(l)}) \odot ((W^{(l+1)})^T \delta^{(l+1)}) \tag{6.17}$$

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。

6.5.1 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$\delta^{(l,k)}$。

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：

$$\begin{aligned}
\delta^{(l,k)} &\triangleq \frac{\partial J(W,\mathbf{b};X,y)}{\partial Z^{(l,k)}} && (6.18)\\
&= \frac{\partial X^{(l,k)}}{\partial Z^{(l,k)}} \cdot \frac{\partial Z^{(l+1,k)}}{\partial X^{(l,k)}} \cdot \frac{\partial J(W,\mathbf{b};X,y)}{\partial Z^{(l+1,k)}} && (6.19)\\
&= f_l'(Z^{(l)}) \odot \left(\mathbf{up}\left(w^{(l+1,k)}\delta^{(l+1)}\right)\right) && (6.20)\\
&= w^{(l+1,k)}\left(f_l'(Z^{(l)}) \odot \mathbf{up}(\delta^{(l+1)})\right), && (6.21)
\end{aligned}$$

其中，**up**为上采用函数（Upsampling）。

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

$$\begin{aligned}
\frac{\partial J(W,\mathbf{b};X,y)}{\partial W_{i,j}^{(l,k,p)}} &= \sum_{s=1}^{w_l}\sum_{t=1}^{h_l}\left(X_{s-i+u,t-j+v}^{(l-1,p)} \cdot (\delta^{(l,k)})_{s,t}\right) && (6.22)\\
&= \sum_{s=1}^{w_l}\sum_{t=1}^{h_l}\left(X_{(u-i)-s,(v-j)-t}^{(l-1,p)} \cdot \left(\mathbf{rot180}(\delta^{(l,k)})\right)_{s,t}\right). && (6.23)
\end{aligned}$$

公式6.23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：

$$\frac{\partial J(W,\mathbf{b};X,y)}{\partial W^{(l,k,p)}} = \mathbf{rot180}\left(X^{(l-1,p)} \otimes \mathbf{rot180}(\delta^{(l,k)})\right). \quad (6.24)$$

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：

$$\frac{\partial J(W,\mathbf{b};X,y)}{\partial b^{(l,k)}} = \sum_{i,j}(\delta^{(l,k)})_{i,j}. \quad (6.25)$$

6.5.2 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。

$$Z^{(l+1,k)} = \sum_{\substack{p,\\ T_{p,k}=1}} \left(W^{(l+1,k,p)} \otimes X^{(l,p)}\right) + b^{(l+1,k)} \tag{6.26}$$

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：

$$\delta^{(l,k)} \triangleq \frac{\partial J(W,\mathbf{b};X,y)}{\partial Z^{(l,k)}} \tag{6.27}$$

$$= \frac{\partial X^{(l,k)}}{\partial Z^{(l,k)}} \cdot \frac{\partial Z^{(l+1,k)}}{\partial X^{(l,k)}} \cdot \frac{\partial J(W,\mathbf{b};X,y)}{\partial Z^{(l+1,k)}} \tag{6.28}$$

$$= f_l'(Z^{(l)}) \odot \left(\sum_{\substack{p,\\ T_{p,k}=1}} \left(\delta^{(l+1,p)} \tilde{\otimes} \mathbf{rot180}(W^{(l,k,p)})\right) \right). \tag{6.29}$$

其中，$\tilde{\otimes}$为宽卷积。

公式6.23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：

$$\frac{\partial J(W,\mathbf{b};X,y)}{\partial w^{(l,k)}} = \sum_{i,j} \left(\mathbf{down}(X^{(l-1,k)}) \cdot \delta^{(l,k)}\right)_{i,j}. \tag{6.30}$$

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：

$$\frac{\partial J(W,\mathbf{b};X,y)}{\partial b^{(l,k)}} = \sum_{i,j} (\delta^{(l,k)})_{i,j}. \tag{6.31}$$

6.6 总结和深入阅读

第七章　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。为了使得前馈神经网络能处理变长的序列数据，一种方法是使用延时神经网络（Time-Delay Neural Networks，TDNN）[Waibel et al., 1989]。

循环神经网络（Recurrent Neural Network，RNN），也叫**递归神经网络**。这里为了区别与另外一种递归神经网络（Recursive Neural Network），我们称为**循环神经网络**。在前馈神经网络模型中，连接存在层与层之间，每层的节点之间是无连接的。

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。

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**$\mathbf{h}^{(t)}$：

$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otherwise} \end{cases} \tag{7.1}$$

从数学上讲，公式7.1可以看成一个动态系统。**动态系统**是指系统的状态按照一定的规律随时间变化的系统。因此，活性值$\mathbf{h}^{(t)}$在很多文献上也称为**状态**。但这里的状态是数学上的概念，区别与我们在前馈网络中定义的神经元的状态。理论上循环神经网络可以近似任意的动态系统。图7.1给出了循环神经网络的示例。

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]。但循环神经网络的一个最大问题是训练时梯度需要随着时间进行反向传播。当输入序列比较长时，会存在梯度爆炸和消失问题Bengio et al. [1994], Hochreiter and Schmidhuber [1997], Hochreiter et al. [2001]。长短时记忆神经网络

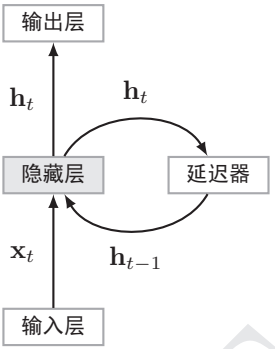


图 7.1: 循环神经网络

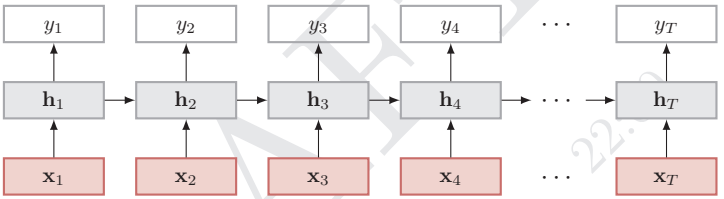


图 7.2: 按时间展开的循环神经网络

（long short term memory neural network，LSTM）[Hochreiter and Schmidhuber, 1997]是训练神经网络的一个扩展。

7.1 简单循环网络

我们先来看一个非常简单的循环神经网络，叫**简单循环网络**（Simple Recurrent Network，SRN）[Elman, 1990]。

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：

$$\mathbf{h}_t = f(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t + \mathbf{b}), \tag{7.2}$$

这里，f是非线性函数，通常为logistic函数或tanh函数。

图7.2给出了按时间展开的循环神经网络。

7.1.1 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7.3给出了随时间进行反向传播算法的示例。

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：

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T} \frac{\partial J_t}{\partial U} \tag{7.3}$$

$$= \sum_{t=1}^{T} \frac{\partial \mathbf{h}_t}{\partial U} \frac{\partial J_t}{\partial \mathbf{h}_t}, \tag{7.4}$$

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。因此，我们可以用链式法则得到

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathbf{h}_k}{\partial U} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial J_t}{\partial \mathbf{y}_t}, \tag{7.5}$$

其中，

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} = \prod_{i=k+1}^{t} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} \tag{7.6}$$

$$= \prod_{i=k+1}^{t} U^T \, \mathbf{diag}[f'(h_{i-1})]. \tag{7.7}$$

因此，

$$\frac{\partial J}{\partial U} = \sum_{t=1}^{T} \sum_{k=1}^{t} \frac{\partial \mathbf{h}_k}{\partial U} \left(\prod_{i=k+1}^{t} U^T \, \mathbf{diag}(f'(h_{i-1})) \right) \frac{\partial \mathbf{y}_t}{\partial \mathbf{h}_t} \frac{\partial J_t}{\partial \mathbf{y}_t}. \tag{7.8}$$

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。

在训练循环神经网络时，更经常出现的是梯度消失问题。因为我们一般情况下使用的非线性激活函数为logistic函数或tanh函数，其导数值都小于1。而权重矩阵$\|U^T\|$也不会太大。我们定义$\|U^T\| \leq \gamma_u \leq 1$，$\|\mathbf{diag}[f'(h_{i-1})]\| \leq \gamma_f \leq 1$，则有

$$\|\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}\| \leq \|U^T\| \|\mathbf{diag}[f'(h_{i-1})]\| \leq \gamma_u \gamma_f \leq 1. \tag{7.9}$$

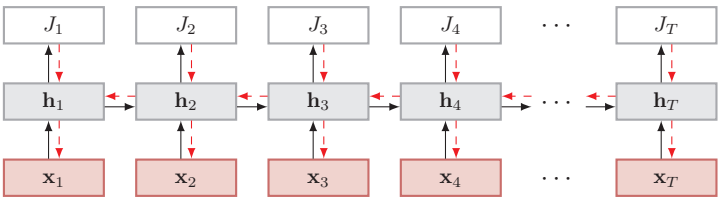


图 7.3: 按时间展开的循环神经网络

经过 $t-k$ 次传播之后，

$$\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\| \leq (\gamma_u \gamma_f)^{t-k}. \tag{7.10}$$

如果时间间隔 $t-k$ 过大，$\|\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k}\|$ 会趋向于 0。

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**。

7.1.2 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 $f'(h_{i-1})=1$。

$$\mathbf{h}_t = \mathbf{h}_{t-1} + \mathbf{W}\mathbf{g}(\mathbf{x}_t), \tag{7.11}$$

$\mathbf{g}$ 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。因此，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 $\mathbf{h}_t$。

$$\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{W}\mathbf{g}(\mathbf{x}_t), \tag{7.12}$$

$$\mathbf{h}_t = \tanh(\mathbf{c}_t). \tag{7.13}$$

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。为了解决这个问题，Hochreiter and Schmidhuber [1997] 提

出一个非常好的解决方案，就是引入**门机制**（Gating Mechanism）来控制信息的累积速度，并可以选择遗忘之前累积的信息。这就是下面要介绍的**长短时记忆神经网络**。

7.2 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。

在时刻 t 时 LSTM 的更新方式如下：

$$\mathbf{i}_t = \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + V_i\mathbf{c}_{t-1}), \tag{7.14}$$

$$\mathbf{f}_t = \sigma(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + V_f\mathbf{c}_{t-1}), \tag{7.15}$$

$$\mathbf{o}_t = \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + V_o\mathbf{c}_t), \tag{7.16}$$

$$\tilde{\mathbf{c}}_t = \tanh(W_c\mathbf{x}_t + U_c\mathbf{h}_{t-1}), \tag{7.17}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \tag{7.18}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{7.19}$$

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。

图7.4给出了 LSTM 模型的计算结构。

这样，LSTM 可以学习到长周期的历史信息。

LSTM 已经被应用到很多的任务中，比如机器翻译 [Sutskever et al., 2014] 等。

7.3 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

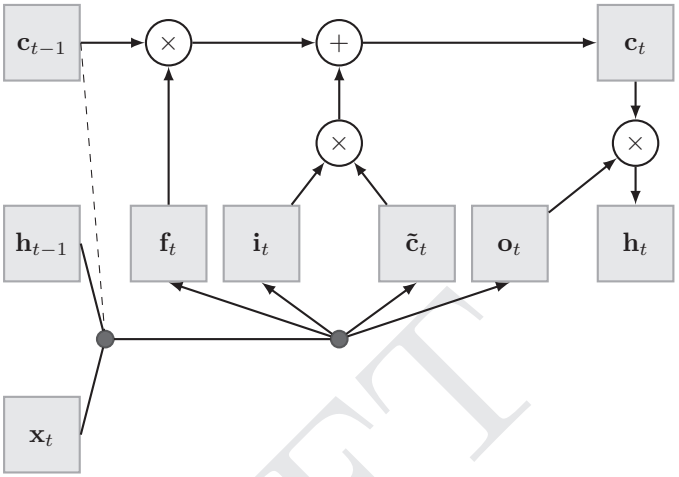


图 7.4: LSTM 结构示例

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：

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1}) \tag{7.20}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1}) \tag{7.21}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}(\mathbf{r}_t \odot \mathbf{h}_{t-1})), \tag{7.22}$$

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t, \tag{7.23}$$

$$\tag{7.24}$$

这里选择 tanh 函数也是因为其导数有更大的值域。

7.4 总结和深入阅读

参考文献

James A Anderson and Edward Rosenfeld. *Talking nets: An oral history of neural networks*. MiT Press, 2000.

Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

Yoshua Bengio, Jean-Sébastien Senécal, et al. Quick training of probabilistic neural nets by importance sampling. In *AISTATS Conference*, 2003.

C.M. Bishop. *Pattern recognition and machine learning*. Springer New York., 2006.

P.F. Brown, P.V. Desouza, R.L. Mercer, V.J.D. Pietra, and J.C. Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

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.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, 2002.

Hal Daumé III. A course in machine learning. http://ciml.info/. [Online].

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.

R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley, New York, 2nd edition, 2001. ISBN 0471056693.

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.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

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

Ian Goodfellow, Aaron Courville, and Yoshua Bengio. Deep learning. Book in preparation for MIT Press, 2015. URL http://goodfeli.github.io/dlbook/.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, New York, 2001.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

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

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

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.

M.I. Jordan. *Learning in Graphical Models*. Kluwer Academic Publishers, 1998.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 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.

Ryan McDonald, Keith Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 456–464. Association for Computational Linguistics, 2010.

Marvin Minsky and Papert Seymour. Perceptrons. 1969.

Marvin L Minsky and Seymour A Papert. *Perceptrons - Expanded Edition: An Introduction to Computational Geometry*. MIT press Boston, MA:, 1987.

T.M. Mitchell. *Machine learning*. Burr Ridge, IL: McGraw Hill, 1997.

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

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.

Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37(3):328–339, 1989.

Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974.

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

DE Rumelhart GE Hinton RJ Williams and GE Hinton. Learning representations by back-propagating errors. *Nature*, pages 323–533, 1986.

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

索引