



神经表示模型

人工智能建模

谭忠

厦门大学

2025 年 8 月 3 日



厦門大學 数学科学学院
SCHOOL OF MATHEMATICAL SCIENCES XIAMEN UNIVERSITY

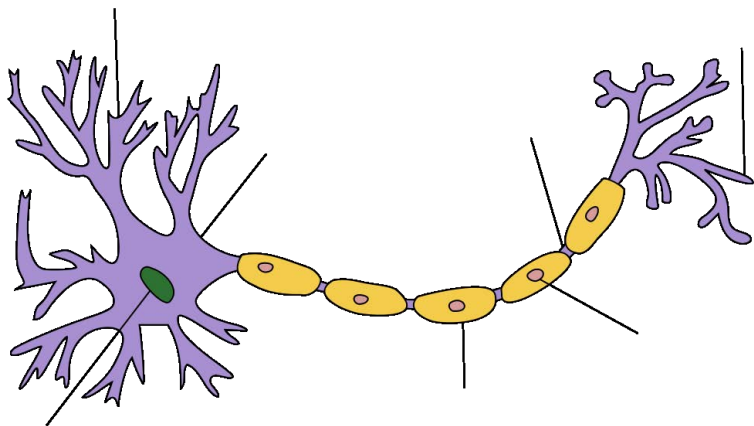
课程大纲

- 前馈神经网络
- 反向传播算法
- 卷积神经网络
- 模型的优化



廈門大學 数学科学学院
SCHOOL OF MATHEMATICAL SCIENCES XIAMEN UNIVERSITY

第一章：前馈神经网络



第一章：前馈神经网络

人类大脑是人体最复杂的器官，由神经元、神经胶质细胞、神经干细胞和血管组成。其中，神经元 (Neuron)，也叫神经细胞，是携带和传输信息的细胞，是人脑神经系统中最基本的单元。人脑神经系统是一个非常复杂的组织，包含近 860 亿个神经元 [Azevedo et al., 2009]，每个神经元有上千个突触和其他神经元相连接。这些神经元和它们之间的连接形成巨大的复杂网络，其中神经连接的总长度可达数千公里。我们人造的复杂网络，比如全球的计算机网络，和大脑神经网络相比要“简单”得多。



第一章：前馈神经网络

随着神经科学、认知科学的发展，我们逐渐知道人类的智能行为都和大脑活动有关。人类大脑是一个可以产生意识、思想和情感的器官。受到人脑神经系统的启发，早期的神经科学家构造了一种模仿人脑神经系统的数学模型，称为人工神经网络，简称神经网络。在机器学习领域，神经网络是指由很多人工神经元构成的网络结构模型，这些人工神经元之间的连接强度是可学习的参数。

第一章：前馈神经网络

神经网络的发展大致经过五个阶段.

第一阶段：模型提出第一阶段为 1943 年 ~ 1969 年，是神经网络发展的第一个高潮期. 在此期间，科学家们提出了许多神经元模型和学习规则.

第二阶段：冰河期第二阶段为 1969 年 ~ 1983 年，是神经网络发展的第一个低谷期. 在此期间，神经网络的研究处于长年停滞及低潮状态.

1969 年，Marvin Minsky 出版《感知器》一书，指出了神经网络的两个关键缺陷：一是感知器无法处理“异或”回路问题；二是当时的计算机无法支持处理大型神经网络所需要的计算能力. 这些论断使得人们对以感知器为代表的神经网络产生质疑，并导致神经网络的研究进入了十多年的“冰河期”.

第一章：前馈神经网络

第三阶段：反向传播算法引起的复兴第三阶段为 1983 年 ~ 1995 年，是神经网络发展的第二个高潮期。这个时期中，反向传播算法重新激发了人们对神经网络的兴趣。然而，梯度消失问题（Vanishing Gradient Problem）阻碍神经网络的进一步发展，特别是循环神经网络。为了解决这个问题，[Schmidhuber, 1992] 采用两步来训练一个多层的循环神经网络：1) 通过无监督学习的方式来逐层训练每一层循环神经网络，即预测下一个输入；2) 通过反向传播算法进行精调。

第一章：前馈神经网络

第四阶段：流行度降低第四阶段为 1995 年 ~ 2006 年，在此期间，支持向量机和其他更简单的方法（例如线性分类器）在机器学习领域的流行度逐渐超过了神经网络。

第五阶段：深度学习的崛起第五阶段为从 2006 年开始至今，在这一时期研究者逐渐掌握了训练深层神经网络的方法，使得神经网络重新崛起。



第一章：前馈神经网络

假设一个神经元接收 D 个输入 $x_1, x_2, \dots, x_D$ ，令向量 $\mathbf{x} = [x_1; x_2; \dots; x_D]$ 来表示这组输入，并用净输入 (Net Input) $z \in \mathbb{R}$ 表示一个神经元所获得的输入信号 $\mathbf{x}$ 的加权和，

$$z = \sum_{d=1}^D w_d x_d + b \quad (1)$$

$$= \mathbf{w}^T \mathbf{x} + b, \quad (2)$$

其中 $\mathbf{w} = [w_1; w_2; \dots; w_D] \in \mathbb{R}^D$ 是 D 维的权重向量， $b \in \mathbb{R}$ 是偏置。净输入 z 在经过一个非线性函数 $f(\cdot)$ 后，得到神经元的活性值 (Activation) a ,

$$a = f(z), \quad (3)$$

其中非线性函数 $f(\cdot)$ 称为激活函数 (Activation Function)。

第一章：前馈神经网络

激活函数在神经元中非常重要的。为了增强网络的表示能力和学习能力，激活函数需要具备以下几点性质：

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

第一章：前馈神经网络

Sigmoid 型函数：一类 S 型曲线函数，为两端饱和函数。常用的 Sigmoid 型函数有 Logistic 函数和 Tanh 函数。

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

第一章：前馈神经网络

下面介绍几种常见的激活函数

Logistic 函数

定义为

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \quad (4)$$

Logistic 函数可以看成是一个“挤压”函数，把一个实数域的输入“挤压”到 $(0, 1)$ 。

Tanh 函数

定义为

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}. \quad (5)$$

Tanh 函数可以看作放大并平移的 Logistic 函数，其值域是 $(-1, 1)$ 。

$$\tanh(x) = 2\sigma(2x) - 1. \quad (6)$$

第一章：前馈神经网络

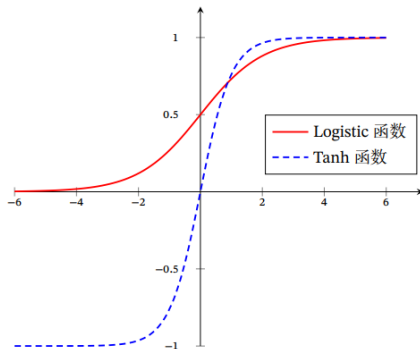


图 4.2 Logistic 函数和 Tanh 函数

图: Enter Caption



廈門大學 数学科学学院
SCHOOL OF MATHEMATICAL SCIENCES XIAMEN UNIVERSITY

第一章：前馈神经网络

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

$$\text{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \quad (7)$$

$$= \max(0, x). \quad (8)$$

在优化方面, 相比于 Sigmoid 型函数的两端饱和, ReLU 函数为左饱和函数, 且在 $x > 0$ 时导数为 1, 在一定程度上缓解了神经网络的梯度消失问题, 加速梯度下降的收敛速度。



第一章：前馈神经网络

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



第一章：前馈神经网络

在实际使用中，为了避免上述情况，有几种 ReLU 的变种也会被广泛使用。

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

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{if } x \leq 0 \end{cases} \quad (9)$$

$$= \max(0, x) + \gamma \min(0, x), \quad (10)$$

其中 γ 是一个很小的常数，比如 0.01。当 $\gamma < 1$ 时，带泄露的 ReLU 也可以写为

$$\text{LeakyReLU}(x) = \max(x, \gamma x), \quad (11)$$



第一章：前馈神经网络

带参数的 ReLU (Parametric ReLU, PReLU) 引入一个可学习的参数, 不同神经元可以有不同的参数 [He et al., 2015]. 对于第 i 个神经元, 其 PReLU 的定义为

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

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

第一章：前馈神经网络

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

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

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

第一章：前馈神经网络

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

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

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

第一章：前馈神经网络

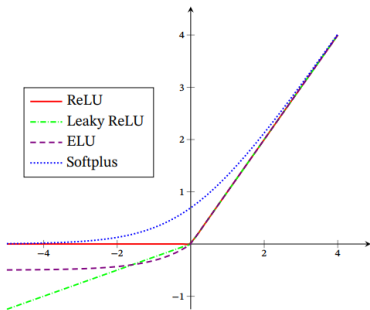


图 4.4 ReLU、Leaky ReLU、ELU 以及 Softplus 函数

图: Enter Caption



第一章：前馈神经网络

Swish 函数 [Ramachandran et al., 2017] 是一种自门控 (Self-Gated) 激活函数，定义为

$$\text{swish}(x) = x\sigma(\beta x), \quad (12)$$

其中 $\sigma(\cdot)$ 为 Logistic 函数， β 为可学习的参数或一个固定超参数。 $\sigma(\cdot) \in (0, 1)$ 可以看作一种软性的门控机制。当 $\sigma(\beta x)$ 接近于 1 时，门处于“开”状态，激活函数的输出近似于 x 本身；当 $\sigma(\beta x)$ 接近于 0 时，门的状态为“关”，激活函数的输出近似于 0。



第一章：前馈神经网络

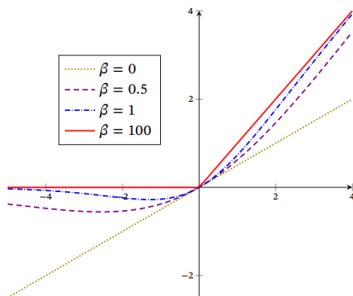


图 4.5 Swish 函数

图: Enter Caption



第一章：前馈神经网络

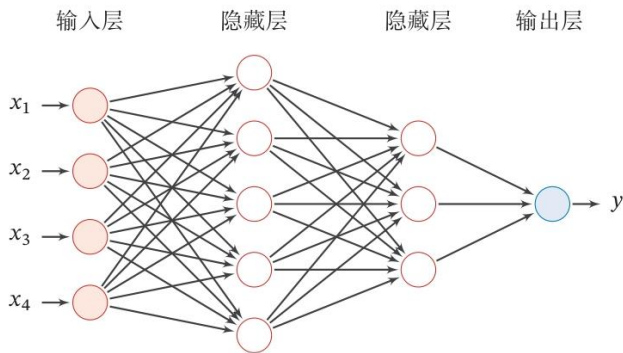


图: Enter Caption



第一章：前馈神经网络

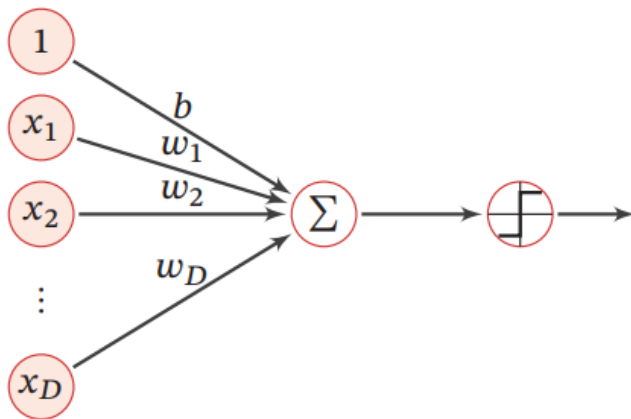


图: Enter Caption



第一章：前馈神经网络

记号

L : 网络层数

$f_l(\cdot)$: 第 l 层的激活函数

$\mathbf{W}^{(l)} \in \mathbb{R}^{M_l \times M_{l-1}}$: 第 $l-1$ 层到第 l 层的权重矩阵

$\mathbf{b}^{(l)} \in \mathbb{R}^{M_l}$: 第 $l-1$ 层到第 l 层的偏置

$\mathbf{z}^{(l)} \in \mathbb{R}^{M_l}$: 第 l 层神经元的净输入 (净活性值)

$\mathbf{a}^{(l)} \in \mathbb{R}^{M_l}$: 第 l 层神经元的输出 (活性值)

第一章：前馈神经网络

令 $\mathbf{a}^{(0)} = \mathbf{x}$ ，前馈神经网络通过不断迭代下面公式进行信息传播：

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}, \quad (13)$$

$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)}). \quad (14)$$

首先根据第 $l-1$ 层神经元的活性值 (Activation) $\mathbf{a}^{(l-1)}$ 计算出第 l 层神经元的净活性值 (Net Activation) $\mathbf{z}^{(l)}$ ，然后经过一个激活函数得到第 l 层神经元的活性值。因此，我们也可以把每个神经层看作一个仿射变换 (Affine Transformation) 和一个非线性变换。



第一章：前馈神经网络

公式 (13) 和公式 (14) 也可以合并写为：

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)} f_{l-1}(\mathbf{z}^{(l-1)}) + \mathbf{b}^{(l)}, \quad (15)$$

或者

$$\mathbf{a}^{(l)} = f_l(\mathbf{W}^{(l)} \mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}). \quad (16)$$

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 $\mathbf{a}^{(L)}$ 。整个网络可以看作一个复合函数 $\phi(\mathbf{x}; \mathbf{W}, \mathbf{b})$ ，将向量 $\mathbf{x}$ 作为第 1 层的输入 $\mathbf{a}^{(0)}$ ，将第 L 层的输出 $\mathbf{a}^{(L)}$ 作为整个函数的输出。

$$\mathbf{x} = \mathbf{a}^{(0)} \rightarrow \mathbf{z}^{(1)} \rightarrow \mathbf{a}^{(1)} \rightarrow \mathbf{z}^{(2)} \rightarrow \dots \rightarrow \mathbf{a}^{(L-1)} \rightarrow \mathbf{z}^{(L)} \rightarrow \mathbf{a}^{(L)} = \phi(\mathbf{x}; \mathbf{W}, \mathbf{b}), \quad (17)$$

其中 $\mathbf{W}, \mathbf{b}$ 表示网络中所有层的连接权重和偏置。

第一章：前馈神经网络

定理 (通用近似定理 (Universal Approximation Theorem) [?, ?])

$\phi(\cdot)$ 是一个非常数、有界、单调递增的连续函数, J_D 是一个 D 维的单位超立方体 $[0, 1]^D$, $C(J_D)$ 是定义在 J_D 上的连续函数集合。对于任意给定的一个函数 $f \in C(J_D)$, 存在一个整数 M , 和一组实数 $v_m, b_m \in \mathbb{R}$ 以及实数向量 $\mathbf{w}_m \in \mathbb{R}^D$, $m = 1, \dots, M$, 以至于我们可以定义函数

$$F(\mathbf{x}) = \sum_{m=1}^M v_m \phi(\mathbf{w}_m^T \mathbf{x} + b_m), \quad (18)$$

作为函数 f 的近似实现, 即

$$|F(\mathbf{x}) - f(\mathbf{x})| < \epsilon, \forall \mathbf{x} \in \mathcal{I}_D, \quad (19)$$

其中 $\epsilon > 0$ 是一个很小的正数。



第一章：前馈神经网络

通用近似定理在实数空间 $\mathbb{R}^D$ 中的有界闭集上依然成立。根据通用近似定理，对于具有线性输出层和至少一个使用“挤压”性质的激活函数的隐藏层组成的前馈神经网络，只要其隐藏层神经元的数量足够，它可以以任意的精度来近似任何一个定义在实数空间 $\mathbb{R}^D$ 中的有界闭集函数 [Funahashi et al., 1993; Hornik et al., 1989]。所谓“挤压”性质的函数是指像 Sigmoid 函数的有界函数，但神经网络的通用近似性质也被证明对于其他类型的激活函数，比如 ReLU，也都是适用的。通用近似定理只是说明了神经网络的计算能力可以去近似一个给定的连续函数，但并没有给出如何找到这样一个网络，以及是否是最优的。此外，当应用到机器学习时，真实的映射函数并不知道，一般是通过经验风险最小化和正则化来进行参数学习。因为神经网络的强大能力，反而容易在训练集上过拟合。

第二章：反向传播算法

假设采用随机梯度下降进行神经网络参数学习，给定一个样本 $(\mathbf{x}, y)$ ，将其输入到神经网络模型中，得到网络输出为 $\hat{y}$ 。假设损失函数为 $\mathcal{L}(y, \hat{y})$ ，要进行参数学习就需要计算损失函数关于每个参数的导数。不失一般性，对第 l 层中的参数 $\mathbf{W}^{(l)}$ 和 $\mathbf{b}^{(l)}$ 计算偏导数。因为 $\frac{\partial \mathcal{L}(y, \hat{y})}{\partial \mathbf{W}^{(l)}}$ 的计算涉及向量对矩阵的微分，十分繁琐，因此我们先计算 $\mathcal{L}(y, \hat{y})$ 关于参数矩阵中每个元素的偏导数 $\frac{\partial \mathcal{L}(y, \hat{y})}{\partial w_{ij}^{(l)}}$ 。根据链式法则

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(y, \hat{y})}{\partial \mathbf{z}^{(l)}}, \quad (20)$$

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial b_i^{(l)}} = \frac{\partial z_i^{(l)}}{\partial b_i^{(l)}} \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z_i^{(l)}}. \quad (21)$$



第二章：反向传播算法

公式 (20) 和公式 (21) 中的第二项都是目标函数关于第 l 层的神经元 $z^{(l)}$ 的偏导数，称为误差项，可以一次计算得到。这样我们只需要计算三个偏导数，分别为 $\frac{\partial z^{(l)}}{\partial w_{ij}^{(l)}}$ 、 $\frac{\partial z^{(l)}}{\partial b^{(l)}}$ 和 $\frac{\partial \mathcal{L}(y, \hat{y})}{\partial z^{(l)}}$ 。

下面分别来计算这三个偏导数。



第二章：反向传播算法

(1) 计算偏导数 $\frac{\partial z^{(l)}}{\partial w_{ij}^{(l)}}$ ：由于 $z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$ ，其偏导数为

$$\begin{aligned}\frac{\partial z^{(l)}}{\partial w_{ij}^{(l)}} &= \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \dots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \dots, \frac{\partial z_{M_l}^{(l)}}{\partial w_{ij}^{(l)}} \right] \\ &= \left[0, \dots, \underbrace{\frac{\partial (w_i^{(l)} a^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}}_i, \dots, 0 \right] \\ &= \left[0, \dots, a_j^{(l-1)}, \dots, 0 \right] \\ &\triangleq \mathbb{I}_i(a_j^{(l-1)}) \in \mathbb{R}^{1 \times M_l}\end{aligned}$$

其中， $w_i^{(l)}$ 为权重矩阵 $W^{(l)}$ 的第 i 行， $\mathbb{I}_i(a_j^{(l-1)})$ 表示第 i 个元素为 $a_j^{(l-1)}$ 、其余元素为 0 的行向量。

第二章：反向传播算法

(2) 计算偏导数 $\frac{\partial z^{(l)}}{\partial b^{(l)}}$ ：由于 $z^{(l)}$ 与 $b^{(l)}$ 的函数关系为 $z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$ ，因此偏导数为

$$\frac{\partial z^{(l)}}{\partial b^{(l)}} = I_{M_l} \in \mathbb{R}^{M_l \times M_l}$$

其中， I_{M_l} 是 $M_l \times M_l$ 的单位矩阵。

第二章：反向传播算法

(3) 计算偏导数 $\frac{\partial L(y, \hat{y})}{\partial z^{(l)}}$ ：偏导数 $\frac{\partial L(y, \hat{y})}{\partial z^{(l)}}$ 表示第 l 层神经元对最终损失的影响，也反映了最终损失对第 l 层神经元的敏感程度，因此通常称为第 l 层神经元的误差项，用 $\delta^{(l)}$ 来表示

$$\delta^{(l)} \triangleq \frac{\partial \mathcal{L}(y, \hat{y})}{\partial z^{(l)}} \in \mathbb{R}^{M_l}.$$

误差项 $\delta^{(l)}$ 也间接反映了不同神经元对网络能力的贡献程度，从而较好地解决了贡献度分配问题（Credit Assignment Problem, CAP）。

$$z = Wa + b$$

第二章：反向传播算法

根据 $z^{(l+1)} = W^{(l+1)}a^{(l)} + b^{(l+1)}$, 有

$$\frac{\partial z^{(l+1)}}{\partial a^{(l)}} = \left(W^{(l+1)}\right)^{\top} \in \mathbb{R}^{M_l \times M_{l+1}}$$

根据 $a^{(l)} = f_l(z^{(l)})$, 其中 $f_l(\cdot)$ 为按位计算的函数, 因此有

$$\begin{aligned} \frac{\partial a^{(l)}}{\partial z^{(l)}} &= \frac{\partial f_l(z^{(l)})}{\partial z^{(l)}} \\ &= \text{diag}(f'_l(z^{(l)})) \in \mathbb{R}^{M_l \times M_l}. \end{aligned}$$

第二章：反向传播算法

因此，根据链式法则，第 l 层的误差项为

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

其中 $\odot$ 是向量的点积运算符，表示每个元素相乘。



第二章：反向传播算法

从以上公式可以看出，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，然后再乘上该神经元激活函数的梯度。



第二章：反向传播算法

在计算出上述三个偏导数之后，公式可以写成：

$$\begin{aligned}\frac{\partial \mathcal{L}(y, \hat{y})}{\partial w_{ij}^{(l)}} &= \mathbb{I}_i(a_j^{(l-1)}) \delta_i^{(l)} \\ &= \left[0, \dots, a_j^{(l-1)}, \dots, 0\right] \left[\delta_1^{(l)}, \dots, \delta_i^{(l)}, \dots, \delta_{M_l}^{(l)}\right]^\top \\ &= \delta_i^{(l)} a_j^{(l-1)},\end{aligned}$$

其中， $\delta_i^{(l)} a_j^{(l-1)}$ 相当于向量 $\delta^{(l)}$ 和向量 $a^{(l-1)}$ 的外积的第 i, j 个元素。
上式可以进一步写成：

$$\left[\frac{\partial \mathcal{L}(y, \hat{y})}{\partial \mathcal{W}^{(l)}}\right]_{ij} = \left[\delta^{(l)} (a^{(l-1)})^\top\right]_{ij}.$$



第二章：反向传播算法

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

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial W^{(l)}} = \delta^{(l)} \left(a^{(l-1)} \right)^T \in \mathbb{R}^{M_l \times M_{l-1}}.$$

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

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial b^{(l)}} = \delta^{(l)} \in \mathbb{R}^{M_l}.$$

第二章：反向传播算法

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，使用误差反向传播算法的前馈神经网络训练过程可以分为以下三步：

- (1) 前馈计算每一层的净输入 $z^{(l)}$ 和激活值 $a^{(l)}$ ，直到最后一层；
- (2) 反向传播计算每一层的误差项 $\delta^{(l)}$ ；
- (3) 计算每一层参数的偏导数，并更新参数。



第三章：卷积神经网络

卷积神经网络 (Convolutional Neural Network, CNN)

CNN 和之前介绍的神经网络一样，可以像乐高积木一样通过组装层来构建。不过，CNN 中新增了卷积层 (Convolution 层) 和池化层 (Pooling 层)。



第三章：卷积神经网络

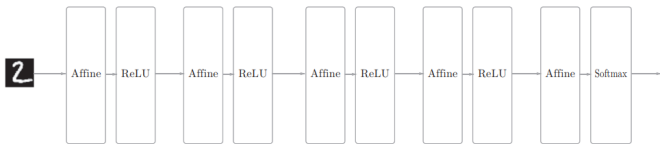


图: Enter Caption

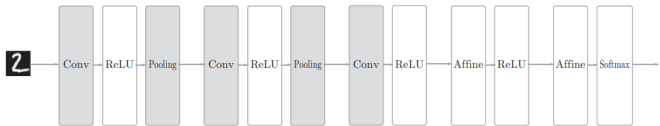


图: Enter Caption



第三章：卷积神经网络

CNN 中新增了卷积层 (Convolution 层) 和池化层 (Pooling 层)。CNN 各层的连接顺序是 “Convolution - ReLU - (Pooling)” (池化层有时会被省略)。这可以理解为之前的 “Affine - ReLU” 连接被替换成了 “Convolution - ReLU - (Pooling)” 连接。

还需要注意的是，在 CNN 中，靠近输出的层使用了之前的 “Affine - ReLU” 组合。此外，最后的输出层使用了之前的 “Affine - Softmax” 组合。这些都是一般 CNN 中比较常见的结构。

第三章：卷积神经网络

图像是 3 维形状, 这个形状中应该含有重要的空间信息。比如, 空间上邻近的像素为相似的值。RGB 的各个通道之间分别有密切的关联性。相距较远的像素之间没有什么关联等, 3 维形状中可能隐藏有值得提取的本质模式。但是, 因为全连接层会忽视形状, 将全部的输入数据作为相同的神经元 (同一维度的神经元) 处理, 所以无法利用与形状相关的信息。而卷积层可以保持形状不变。当输入数据是图像时, 卷积层会以 3 维数据的形式接收输入数据, 并同样以 3 维数据的形式输出至下一层。因此, 在 CNN 中, 可以 (有可能) 正确理解图像等具有形状的数据。



第三章：卷积神经网络

一维卷积常用于信号处理中，用于计算信号的延迟累积。假设一个信号发生器每个时刻 t 产生一个信号 x_t ，其信息的衰减率为 w_k ，即在 $k-1$ 个时间步长后，信息为原来的 w_k 倍。假设 $w_1 = 1$ ， $w_2 = 1/2$ ， $w_3 = 1/4$ ，那么在时刻 t 收到的信号 y_t 为当前时刻产生的信息和以前时刻延迟信息的叠加，

$$\begin{aligned}y_t &= 1 \times x_t + 1/2 \times x_{t-1} + 1/4 \times x_{t-2} \\&= w_1 \times x_t + w_2 \times x_{t-1} + w_3 \times x_{t-2} \\&= \sum_{k=1}^3 w_k x_{t-k+1}.\end{aligned}$$



第三章：卷积神经网络

我们把 $w_1, w_2, \dots$ 称为滤波器 (Filter) 或卷积核 (Convolution Kernel)。假设滤波器长度为 K ，它和一个信号序列 $x_1, x_2, \dots$ 的卷积为

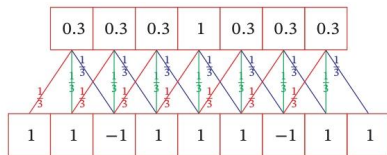
$$y_t = \sum_{k=1}^K w_k x_{t-k+1}.$$

为了简单起见，这里假设卷积的输出 y_t 的下标 t 从 K 开始。信号序列 x 和滤波器 w 的卷积定义为

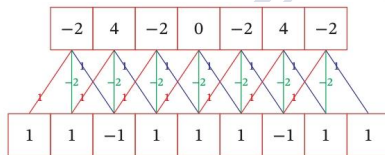
$$y = w * x,$$

其中 $*$ 表示卷积运算。一般情况下，滤波器的长度 K 远小于信号序列 x 的长度。我们可以设计不同的滤波器来提取信号序列的不同特征。

第三章：卷积神经网络



(a) 滤波器 $[1/3, 1/3, 1/3]$



(b) 滤波器 $[1, -2, 1]$

图: Enter Caption



第三章：卷积神经网络

二维卷积

卷积也常用于图像处理中。由于图像是二维结构，因此需要对一维卷积进行扩展。给定一幅图像 $X \in \mathbb{R}^{M \times N}$ 和一个滤波器 $W \in \mathbb{R}^{U \times V}$ ，通常 $U \ll M$ ， $V \ll N$ ，它们的卷积为

$$y_{ij} = \sum_{u=1}^U \sum_{v=1}^V w_{uv} x_{i-u+1, j-v+1}$$

为简单起见，这里假设卷积输出 y_{ij} 的下标 (i, j) 从 (U, V) 开始。

第三章：卷积神经网络

1	1	1	1	1
-1	0	-3	0	1
2	1	1	-1	0
0	-1	1	2	1
1	2	1	1	1

 $*$

1	0	0
0	0	0
0	0	-1

 $=$

0	-2	-1
2	2	4
-1	0	0

图 5.2 二维卷积示例

图: Enter Caption



第三章：卷积神经网络

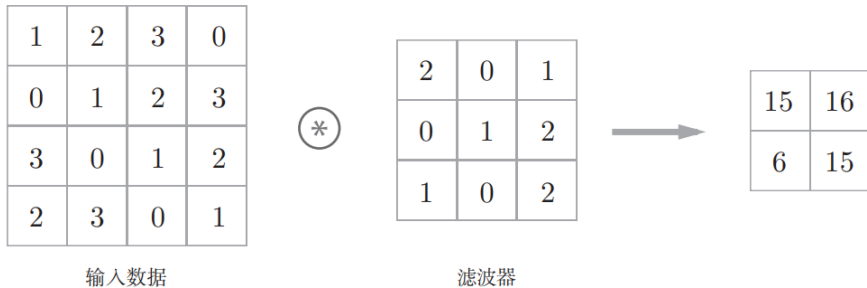


图: Enter Caption



第三章：卷积神经网络

填充

在进行卷积层的处理之前，有时要向输入数据的周围填入固定的数据（比如 0 等），这称为填充（padding），是卷积运算中经常会用到的处理。比如，在下图例子中，对大小为 $(4, 4)$ 的输入数据应用了幅度为 1 的填充。“幅度为 1 的填充”是指用幅度为 1 像素的 0 填充周围。



第三章：卷积神经网络

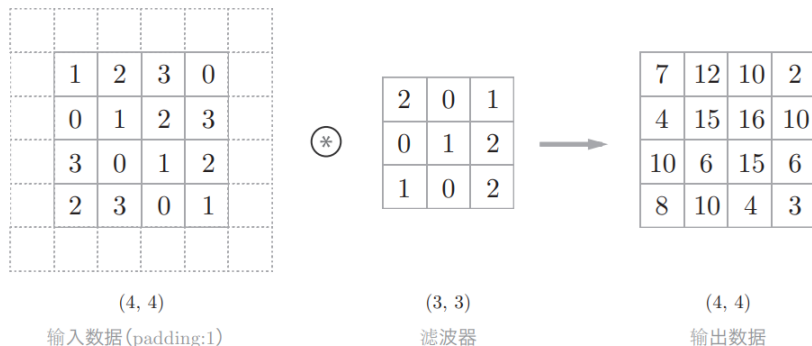


图: Enter Caption



第三章：卷积神经网络

步幅

应用滤波器的位置间隔称为步幅 (stride)。之前的例子中步幅都是 1，如果将步幅设为 2，则如下图所示，应用滤波器的窗口的间隔变为 2 个元素。

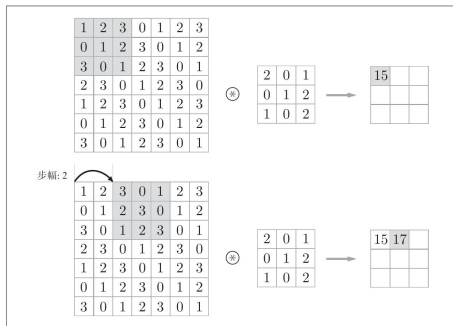


图: Enter Caption



第三章：卷积神经网络

以下是 3 维卷积运算的示例和计算顺序。这里以 3 通道的数据为例，展示了卷积运算的结果。与二维数据相比，可以发现纵深方向（通道方向）上的特征图增多了。当通道方向上有多个特征图时，会按通道对输入数据和滤波器进行卷积运算，并将结果相加，进而得到输出

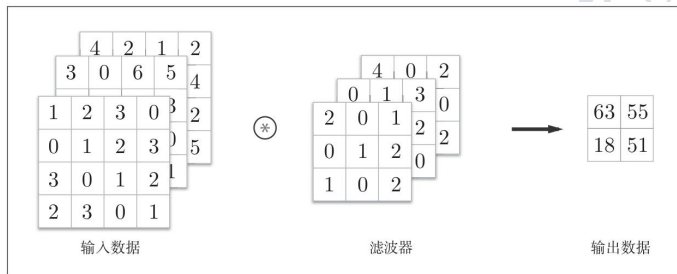


图: Enter Caption



第三章：卷积神经网络

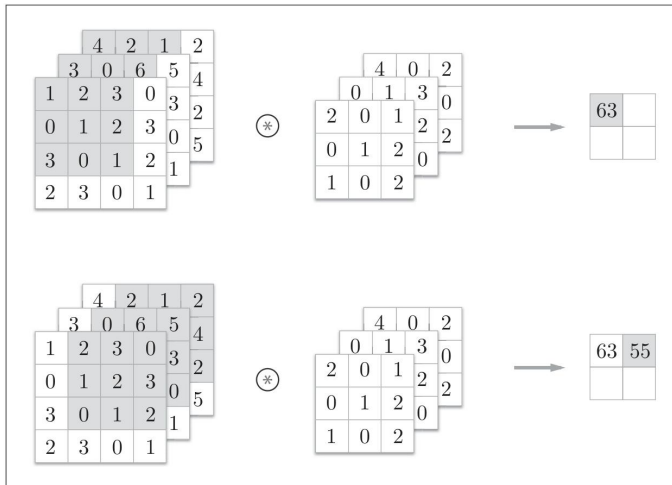
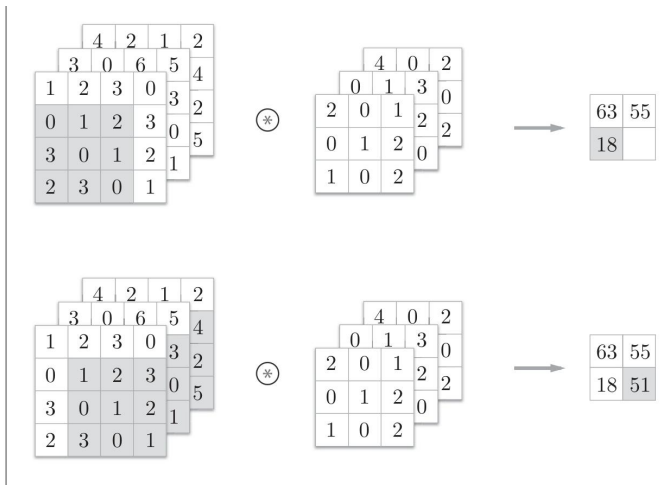


图: Enter Caption



第三章：卷积神经网络



第三章：卷积神经网络

在图像处理中，卷积经常作为特征提取的有效方法。一幅图像在经过卷积操作后得到的结果称为特征映射（Feature Map）。下图给出了图像处理中几种常用的滤波器，以及其对应的特征映射。图最上面的滤波器是常用的高斯滤波器，可用于对图像进行平滑去噪；中间和最下面的滤波器则可用来提取边缘特征。

第三章：卷积神经网络

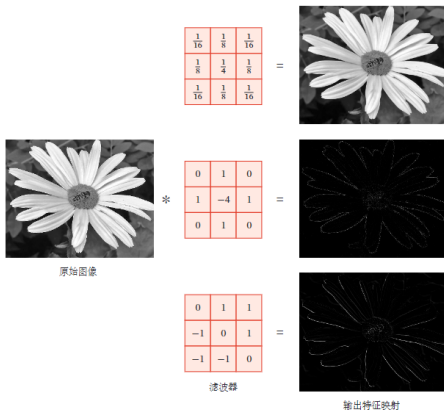


图: Enter Caption



第三章：卷积神经网络

池化层

池化是缩小高、长方向上的空间的运算。比如, 如图所示, 进行将 2×2 的区域集约成 1 个元素的处理, 缩小空间大小。

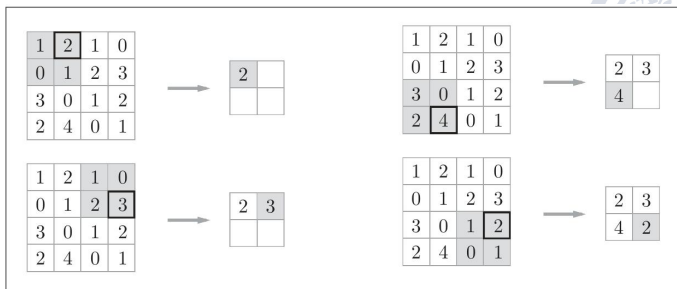


图: Enter Caption



第三章：卷积神经网络

上图的示例展示了步幅为 2 时进行 2×2 最大池化的处理顺序。“最大池化”是一种获取最大值的运算，“ 2×2 ”表示目标区域的大小。如图所示，从 2×2 的区域中提取最大的元素。此外，在这个例子中，步幅被设为 2，因此 2×2 的窗口以 2 个元素为间隔进行移动。另外，一般来说，池化的窗口大小和步幅会设置为相同的值。例如， 3×3 的窗口会搭配步幅 3， 4×4 的窗口会搭配步幅 4，以此类推。

除了最大池化之外，还有平均池化等。与最大池化从目标区域中提取最大值不同，平均池化计算的是目标区域的平均值。在图像识别领域，主要使用最大池化。

第三章：卷积神经网络

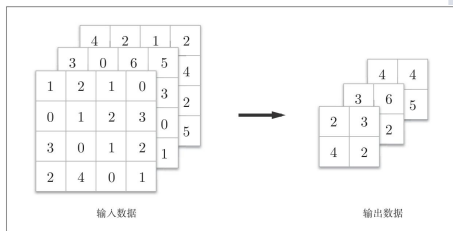
池化层的特征

没有要学习的参数

池化层和卷积层不同，它没有需要学习的参数。池化只是从目标区域中选取最大值（或者计算平均值），因此不存在需要学习的参数。

通道数不发生变化

经过池化运算后，输入数据和输出数据的通道数不会发生变化。如图所示，计算是按通道独立进行的。



第三章：卷积神经网络

对微小的位置变化具有鲁棒性

当输入数据发生微小偏差时，池化操作仍然会返回相同的结果。因此，池化对输入数据的微小偏差具有稳健性（抗干扰能力）。例如，在 3×3 的池化情况下，如图所示，池化能够吸收输入数据的偏差（不过，根据具体数据的不同，结果也可能不一致）。

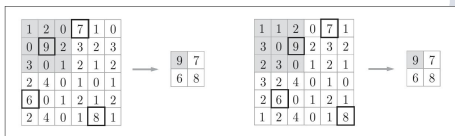


图: Enter Caption



第四章：模型的优化

深度学习的参数规模一般比较大，参数学习往往需要投入大量的计算资源和时间。因此，高效的优化算法对加快训练和拓展深度学习模型的应用具有重要的意义。以下将介绍几种经典的优化算法。

深度学习的目标是最小化以下损失函数：

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n L(f(x_i; \theta), y_i). \quad (22)$$

其中 n 为样本数量, $L(f(x_i; \theta), y_i)$ 为样本 i 上的损失函数。解决函数 $J(\theta)$ 最小化的典型学习算法叫做梯度下降算法。

第四章：模型的优化

$$\begin{aligned} f(x+x_0, y+y_0) - f(x, y) &= f_x(x, y)x_0 + f_y(x, y)y_0 + o(\sqrt{x_0^2 + y_0^2}) \\ &= \nabla f(x, y) \cdot (x_0, y_0) + o(\sqrt{x_0^2 + y_0^2}) \end{aligned} \quad (23)$$

其中 $\nabla f(x, y) = (f_x(x, y), f_y(x, y))$, 显然当 (x_0, y_0) 与 $\nabla f(x, y)$ 同方向时, $\nabla f(x, y) \cdot (x_0, y_0)$ 取最大值, 而 $o(\sqrt{x_0^2 + y_0^2})$ 相对可以忽略不计, 所以 (x, y) 沿着 $\nabla f(x, y)$ 方向增长最快. 沿着 $-\nabla f(x, y)$ 方向下降最快.

第四章：模型的优化

梯度下降法 (gradient descent) 是求解无约束最优化问题的一阶最优化算法，又称最速下降法 (steepest descent)。梯度下降法基于以下的观察：若实值函数 $f(\mathbf{x})$ 在点 a 处可微且有定义，那么函数 $f(\mathbf{x})$ 在 a 点沿着梯度相反的方向 $-\nabla f(\mathbf{x})$ 下降最快。

假设 $f(\mathbf{x})$ 是具有一阶连续偏导数的函数，梯度下降法求解的无约束最优化问题是：

$$\min_{\mathbf{x}} f(\mathbf{x})$$

梯度下降法是一种迭代算法。从初始值 $\mathbf{x}^{(0)}$ 开始，通过下列迭代公式更新：

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \lambda_t \nabla f(\mathbf{x}^{(t)}),$$

其中 λ_t 为学习率。 t 是迭代步数，通过不断迭代，我们能够不断降低目标函数值，直到收敛。



第四章：模型的优化

由于负梯度方向是使得函数值下降最快的方向，因此在每一步迭代中以负梯度方向更新 x 的值。当目标函数是凸函数时，梯度下降法的解是全局最优的。

在损失函数中，梯度为

$$\nabla_{\theta} J(\theta^{(t)}) = \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} L(f(x_i; \theta), y_i). \quad (24)$$

每轮迭代都需要计算 n 个样本的梯度，然后计算其平均值。当数据集规模很大 ($n > 10^6$) 时，计算代价是非常高的。为了解决这个问题，我们希望简化梯度的计算，通常我们不直接计算 $\nabla_{\theta} J(\theta)$ ，而是对其进行近似。



第四章：模型的优化

随机近似是一种被广泛采用的方法。在每轮迭代中，随机采样数据集的一个大小为 m 的子集（一般为几十到几百），这个子集通常称为 mini-batch。然后用子集中的样本梯度 $\nabla_{\theta} \hat{J}(\theta)$ 去近似 $\nabla_{\theta} J(\theta)$ 。 $\nabla_{\theta} \hat{J}(\theta)$ 的计算公式为：

$$\nabla_{\theta} \hat{J}(\theta) = \frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}_i; \theta_t), y_i)$$

可以证明 $\nabla_{\theta} \hat{J}(\theta)$ 是 $\nabla_{\theta} J(\theta)$ 的无偏估计，即满足 $\nabla_{\theta} J(\theta) = \mathbb{E} [\nabla_{\theta} \hat{J}(\theta)]$ 。现在，每轮迭代中参数更新公式为：

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla_{\theta} \hat{J}(\theta^{(t)}),$$

其中 η_t 为学习率。



廈門大學 数学科学学院
SCHOOL OF MATHEMATICAL SCIENCES XIAMEN UNIVERSITY

第四章：模型的优化

这种依赖于随机梯度的学习方法被称为随机梯度下降 (Stochastic Gradient Descent, SGD)。SGD 是深度学习模型中最经典的优化方法，大多数其他的算法都是基于 SGD 的思想开发出来的。学习率的设定是 SGD 算法中的一个关键。Bottou 给出了理论上保证 SGD 收敛的充分条件为：

$$\sum_{t=1}^{\infty} \eta_t = \infty, \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty.$$

这个条件告诉我们在实际中有必要随着迭代次数的增加逐渐降低学习率。具体地，学习率可通过目标函数值随迭代次数的变化来确定是否减少。如果观察到目标函数较长时间内没有明显变化，可以考虑将学习率降低至原来的 $\frac{1}{10}$ 到 $\frac{1}{2}$ 。

我们下面介绍几种具有代表性的 SGD 的变种。

第四章：模型的优化

动量法

动量法 (momentum) 的主要想法是在参数更新时考虑历史梯度的信息。
动量法的参数更新规则如下：

$$\mathbf{v}^{(t+1)} = \alpha \mathbf{v}^{(t)} + \eta \nabla_{\theta} \hat{J}(\theta^{(t)}),$$

$$\theta^{(t+1)} = \theta^{(t)} - \mathbf{v}^{(t)}.$$

其中超参数 $\alpha \in [0, 1)$ 决定了历史梯度的贡献的衰减速率， α 越大，表明历史梯度对现在的更新方向的影响越大。实践中 α 的一般设置为 0.5、0.9 或者 0.99。

第四章：模型的优化

与学习率相似， α 有时也需要随着迭代次数进行调整。一般初始为一个较小的值，然后随着迭代次数增多而慢慢变大。随着时间推移调整 α 比收缩 η 更重要。

动量法克服了 SGD 的两个问题：黑塞矩阵的病态（poor conditioning）和随机梯度的方差带来的不稳定。它比 SGD 多了一个超参数 α ，增加了调参的难度。但如果调整恰当，动量法往往能取得比 SGD 更佳的效果。

第四章：模型的优化

Nesterov 动量法

Sutskever 等人提出了动量法的一个变种。它将 SGD 和凸优化中著名的 Nesterov 加速梯度算法结合，其更新规则如下：

$$\mathbf{v}^{(t+1)} = \alpha \mathbf{v}^{(t)} + \eta \nabla_{\theta} \hat{J}(\theta^{(t)} - \alpha \mathbf{v}^{(t)}),$$

$$\theta^{(t+1)} = \theta^{(t)} - \mathbf{v}^{(t)}.$$

Nesterov 动量法和标准动量法的不同体现在梯度计算上。在 Nesterov 动量法中，梯度计算在更新参数之后（即加了当前速度之后），而标准动量法的梯度计算在更新参数之前。因此，Nesterov 动量法可以理解为在进行参数更新之前，对更新的方向 $\mathbf{v}$ 提前探测和进行校正。

第四章：模型的优化

自适应学习率的优化方法

在神经网络的训练过程中，学习率对模型的表现有着显著的影响，但是却很难设置。另一方面，神经网络的黑塞矩阵的病态特征，导致目标函数对不同参数的敏感度差异巨大。如果对所有参数都采取同样大小的学习率，显然是个效率比较低的做法。因此我们希望优化算法在整个迭代过程中自动设置学习率，且对不同参数设置不同的学习率，从而提高学习效率。在本小节中，我们将介绍几种自适应学习率的优化算法。

第四章：模型的优化

1. AdaGrad

在 AdaGrad 算法中，每一个模型参数的学习率都单独设定。具体地，它利用梯度历史平方和信息来自动调整学习率大小。

参数 θ 在第 t 步迭代中的随机梯度为 $g^{(t)} = \nabla_{\theta} \hat{J}(\theta^{(t)})$ 。对于参数 θ 的第 i 个分量，记其到第 t 步迭代的历史梯度平方和为 $r_i^{(t)}$ ，即：

$$r_i^{(t)} = \sum_{j=0}^t g_i^{(j)} \cdot g_i^{(j)}.$$

假设全局学习率为 η ，AdaGrad 算法的参数更新公式如下：

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{r_i^{(t)} + \delta}} g_i^{(t)}.$$

其中 δ 是一个为了保持数值计算稳定性的常数，一般设置为 10^{-8} 。

第四章：模型的优化

可见，在累积梯度平方和较大的方向上，学习率相对小一些；累积梯度平方和较小的方向上，学习率相对大一些。AdaGrad 算法可以在参数空间中更为平缓的方向上取得更大的更新。

AdaGrad 算法在参数的不同分量上使用不同的学习率，因此不太容易停留在目标函数的鞍点 (saddle point) 上。虽然 AdaGrad 算法能够自动调整学习率，全局学习率 η 仍然需要人工设置。由于在迭代的开始就计算梯度平方和，AdaGrad 算法容易使得学习率过早过快衰减。这样的结果是会使得学习率过小，参数的更新趋近于零，从而提前终止模型训练。



第四章：模型的优化

RMSprop

RMSprop 算法是由 Geoffrey Hinton 提出的。与 AdaGrad 算法一样，RMSprop 算法也是由梯度的历史信息来自动调整学习率大小。在 AdaGrad 算法中，学习率根据历史梯度平方和来调整。而在 RMSprop 算法中，则将历史梯度的平均值和当前梯度平方和同时进行考虑。参数 θ 在第 t 步迭代中的随机梯度为 $g^{(t)} = \nabla_{\theta} \hat{J}(\theta^{(t)})$ 。对于参数 θ 的第 i 个分量，我们使用 $r_i^{(t)}$ 表示其在第 t 步的学习率调节因子，其计算公式为：

$$r_i^{(t)} = \rho \cdot \frac{1}{t-1} \sum_{j=0}^{t-1} g_i^{(j)} \cdot g_i^{(j)} + (1 - \rho) g_i^{(t)} \cdot g_i^{(t)}.$$

其中 $\rho \in [0, 1]$ 是一个超参数，来调节历史梯度的平均值和当前梯度平方和的相对重要性。

第四章：模型的优化

基于学习率调节因子 $r_i^{(t)}$ ，RMSprop 的参数更新公式为：

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{r_i^{(t)} + \delta}} g_i^{(t)},$$

其中 η 为全局学习率。

在 RMSprop 算法中，当前迭代步的学习率主要受最近迭代步的梯度影响，而大大降低了对早期历史梯度的依赖。这使得它不会产生像 AdaGrad 过早过大地减少某些方向学习率的问题。然而，与 AdaGrad 算法相比，RMSprop 算法多了一个超参数 ρ 需要人工设置。一个建议的设置是将 ρ 设置为 0.9，全局学习率 η 设置为 0.001。

第四章：模型的优化

AdaDelta

AdaDelta 算法进一步使用每步迭代中参数的更新值信息来调整学习率。
假设 $r_i^{(t)}$ 的定义如上所示，第 t 步参数的更新值为 $\Delta\theta_i^{(t)}$ ：

$$\Delta\theta_i^{(t)} = -\frac{\eta}{\sqrt{r_i^{(t)} + \delta}} g_i^{(t)},$$

则截止到第 t 步，参数更新值因子为

$$s_i^{(t)} = \gamma \cdot \frac{1}{t-1} \sum_{j=1}^{t-1} \Delta\theta_i^{(j)} \cdot \Delta\theta_i^{(j)} + (1 - \gamma) \Delta\theta_i^{(t)} \cdot \Delta\theta_i^{(t)}.$$

其中 γ 为超参数。

第四章：模型的优化

AdaDelta 的参数更新公式为

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\sqrt{s_i^{(t-1)} + \delta}}{\sqrt{r_i^{(t-1)} + \delta}} g_i^{(t)}.$$

可见，在 AdaDelta 算法中，我们不再需要人工设置学习率。然而在计算 r 和 s 时，需要额外设置两个超参数 ρ 和 γ 。

第四章：模型的优化

Adam[193] 是最近提出的一种能自适应选择学习率的算法。Adam 综合使用当前梯度和历史梯度来决定参数更新方向。假设第 $t-1$ 步更新方向为 $\mathbf{s}^{(t-1)}$ ，则 $\mathbf{s}^{(0)} = \mathbf{0}$ ，则有

$$\mathbf{s}^{(t)} = \rho_1 \mathbf{s}^{(t-1)} + (1 - \rho_1) \mathbf{g}^{(t)}.$$

同时，使用梯度历史平方和 $\mathbf{r}^{(t-1)}$ 来调整学习率， $\mathbf{r}^{(0)} = \mathbf{0}$ 且

$$r_i^{(t)} = \rho_2 r_i^{(t-1)} + (1 - \rho_2) g_i^{(t)} \cdot g_i^{(t)},$$

其中 $\mathbf{g}$ 为随机梯度， $\rho_1, \rho_2 \in [0, 1]$ ，推荐默认值分别为 0.9 和 0.999。

第四章：模型的优化

进一步地，可以对 $s^{(t)}$ 和 $r^{(t)}$ 进行偏差修正

$$\hat{s}^{(t)} = \frac{s^{(t)}}{1 - \rho_1^t},$$

$$\hat{r}^{(t)} = \frac{r^{(t)}}{1 - \rho_2^t},$$

其中 t 为迭代步。在以上步骤完成之后才进行参数的更新：

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{\hat{r}_i^{(t)} + \delta}} \hat{s}_i^{(t)}.$$

Adam 对超参数不太敏感，且在实践中能取得较好效果。