理解深度学习

许志钦

2023 年 9 月 13 日

目录

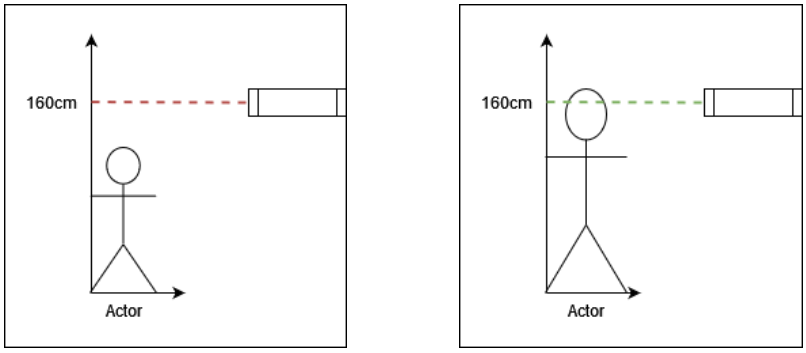
Chapter 1

深度学习介绍

1.1 神经网络简介

1.1.1 单个神经元如何感知信息

感知信息的基本条件是对不同的输入能够区分。为了区分不同的信息，首先要能够从信号中提取有意义的内容。举个例子，假若我们需要征兵，身高超过 160cm 才能满足初试的条件。于是我们设计一个身高信息的提取器。



(a) 未达到 160cm　　(b) 达到 160cm

图 1.1: 身高提取器

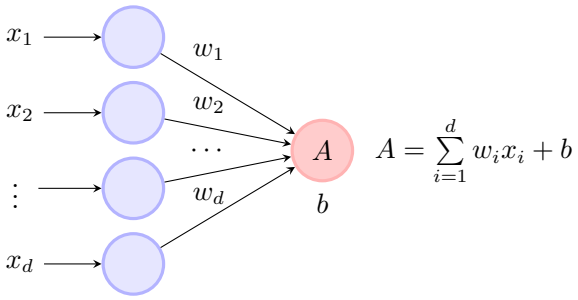
如图 1.1，我们在 160cm 高的位置放一个激光器和接收器。它不断地发出信号，若应征者的身高达到 160cm，则信号会被反射回来，接收器在接收到信号以后，会显示合格，否则不合格。注意到，尽管身高的完整信息是一个具体数字，但我们实际上不需要完整的信息，而是

只需要知道是否大于 160cm，只需要输出 0 或者 1，一个字节的信息。因此这套装备的作用可以描述为如下的阀值函数

$$\sigma(x) = \begin{cases} 1 & x \geq T \\ 0 & x < T \end{cases}, \tag{1.1}$$

一个神经元所执行的基本功能也通常简化成类似的函数，也就是，当外界的信息的强度大于某个阀值 T 后，神经元会输出一个脉冲信号，即 1，否则为 0。这里的关键是信息怎么被传递给神经元的。

我们来举一个例子。假设 A 有两个朋友，B 和 C。这两个朋友每天会给 A 关于股票的信息，该信息是介于 0 和 1 之间的数，表示他们认为股票上升的概率。他们给的信息分别记为 b 和 c。即使 B 和 C 没有给任何信息，A 自己也有一个判断值 a_0。如果 B 是一个经济学家，C 是一个普通人，A 会认为 B 更可信，于是 A 给 B 和 C 各分配了一个权重 λ_1 和 λ_2，其中 $\lambda_1 > \lambda_2 > 0$。最终，A 接收的信息为 $a = \lambda_1 b + \lambda_2 c + a_0$。我们可以把这类收集信息的方式推广到一般的形式。假设有 d 个信息源，$x_1, \cdots, x_d$，神经元对各个信息的权重为 $w_1, \cdots, w_d$，神经元本身的偏置信息为 b，



最终神经元得到的总输入为

$$w_1 x_1 + \cdots + w_d x_d + b \tag{1.2}$$

$$= (w_1, \cdots, w_d) \cdot (x_1, \cdots, x_d) + b \tag{1.3}$$

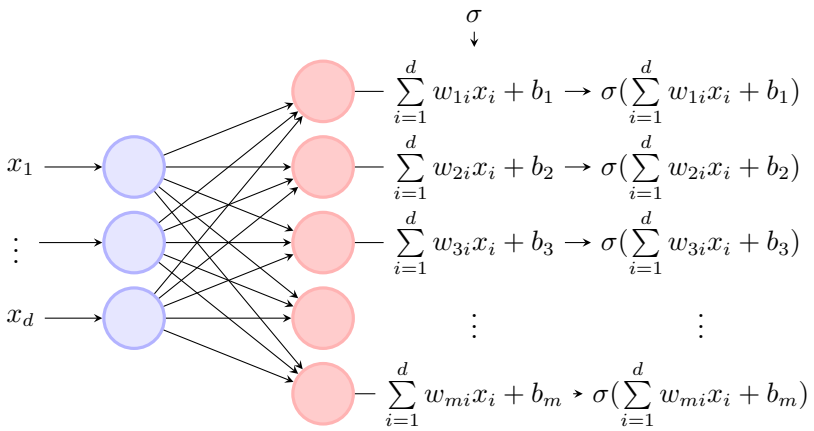
$$= \boldsymbol{w}^T \boldsymbol{x} + b. \tag{1.4}$$

$(\boldsymbol{w}, b)$ 是神经元的参数，体现神经元的性质。我们再从投影的角度来理解一下神经元处理信息的方式。$\boldsymbol{w}^T \boldsymbol{x}$ 是两个向量的内积，可以理解为信息 $\boldsymbol{x}$ 在神经元特征 $\boldsymbol{w}$ 上的投影。

单个神经元的输出为 $\sigma(\boldsymbol{w}^T \boldsymbol{x} + b)$，$\sigma(\cdot)$ 通常叫做激活函数。

1.1.2 单层神经网络

简单地把多个神经元堆在一起，我们就可以得到单层神经网络。



数学上，我们可以把参数写成一般的形式

$$W = \begin{pmatrix} w_{11}, & \cdots, w_{1d} \\ \vdots, & \cdots, \vdots \\ w_{m1}, & \cdots, w_{md} \end{pmatrix}, \tag{1.5}$$

$$\boldsymbol{b} = (b_1, b_2, \cdots, b_m)^T. \tag{1.6}$$

网络的输出为

$$y = \sigma \circ (W\boldsymbol{x} + \boldsymbol{b}), \tag{1.7}$$

其中 $\circ$ 表示对每个元素作用上激活函数。例如，$\sigma \circ ([0.1, 0.3, 0.5]) = [\sigma(0.1), \sigma(0.3), \sigma(0.5)]$。在神经网络发展的初期，由于计算量不足，主要是单层神经网络起到重要作用。

从纯设计的角度，所有神经元的参数都是人为给定，单个神经元可以完成很多逻辑运算。比如，AND 运算，考虑输入是两维的，$x_1, x_2 \in \{0, 1\}^2$，也就是每个维度只取 0 或者 1。取阀值 $T = 2$，神经元的输出

$$\sigma(x) = \begin{cases} 1 & x_1 + x_2 \geq T \\ 0 & x_1 + x_2 < T \end{cases}, \tag{1.8}$$

只有两个维度同时为 1 时，输出才为 1，否则为 0，此即为 AND 运算。若取 $T = 1$，则该神经元在执行 OR 运算。联系到，现代的计算机的基础运算单元正是逻辑运算，所以尽管尚未考虑神经元参数的学习规则，人们已经意识到神经网络可以完成很多任务。

随着使用的深入，人们发现单层神经元只能完成线性可分的问题，对于如图 1.2 所示的 XOR 问题，单层神经网络是无法做好的，至少需要两层的网络才能做到。而多层网络难以训练的情况在当时也劝退了一大批研究人员，带来了神经网络的第一次低潮。

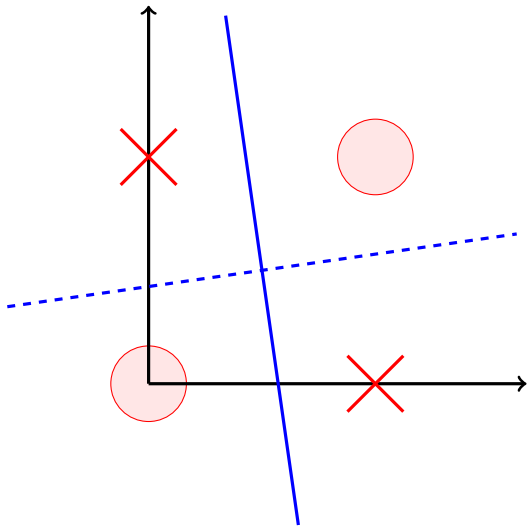


图 1.2: XOR 问题

1.1.3 多层神经网络

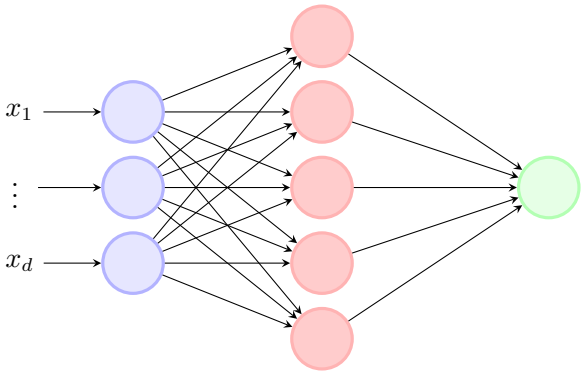


图 1.3: 两层神经网络

多层神经网络是在一层神经网络的基础上堆叠起来的。以两层网络为例，如图1.3，从输入层到第一层的运算和单层网络完全一样，然后以第一层为输入，从第一层到第二层是另一个单层网络。这里的第一层对于使用者来说是看不见的，它是用在内部处理信息的，因此也被称为隐藏层。

我们这里构造一个两层神经网络解 XOR 问题的例子1.4。

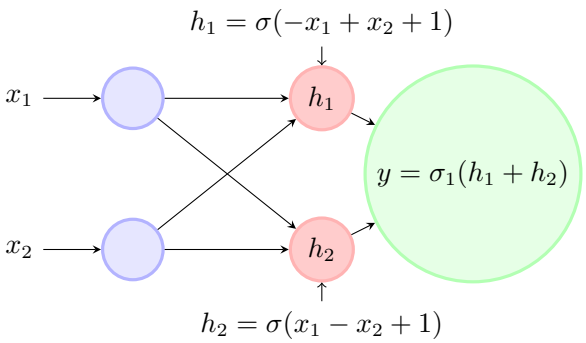


图 1.4: 两层网络解 XOR 问题

如图 1.4所示，当 $x_1 = x_2 = 0$ 或者 $x_1 = x_2 = 1$ 时，$h_1 = h_2 = \sigma(1)$, 而当 $x_1 = 1, x_2 = 0$ 或 $x_1 = 0, x_2 = 1$ 时，$h_{\{1,2\}} = \sigma(0)$,$h_{\{2,1\}} = \sigma(2)$，这样就完成 XOR 问题的分类了。

进一步，在 1980 年代末，一系列的工作证明当激活函数是非线性且非多项式的时候，对任意连续函数，当神经网络的隐藏层足够宽时，两层神经网络可以以任意精度逼近该连续函数，这也就是万有逼近定理 (Cybenko 1989)。

作为思考，大家考虑一下为什么这里要求激活函数不能是多项式函数？

万有逼近定理展现了神经网络具有很强的逼近能力，给神经网络的使用提供了一个重要的理论支撑。但这个理论也与实际使用神经网络有很大的区别，比如尽管两层网络的表达能力已经很强，但实际使用中一般用的是更多层的网络，我们需要多少神经元足够？神经元的数目会不会多到我们实际训练无法承受？定理证明中构造的解是否在实际训练中能被找到？我们需要用什么样的训练方法，需要多少数据？很多类似的问题是万有逼近定理无法回答的。

通过调节多层网络的参数（包括权重和偏置）去学习一组数据是现代常用的机器学习方法。机器学习的方法很多，神经网络只是其中一种。简单地说，凡是通过数据调整模型参数的方法可以称为机器学习。我们也经常听到统计学习，我们下面用线性拟合来简单讨论这两类说法。

1.2 线性拟合

统计和机器学习 (Machine learning, ML) 在数据挖掘中都有应用，用一个直观但并不是很严谨的方法去解释它们区别就是：统计是基于模型假设的，而机器学习是不需要模型基础的。本节中我们会用一个线性模型去理解它们的区别。

1.2.1 统计的角度

假设我们有一组训练集 $S = \{(x_i, y_i)\}_{i=1}^n, (x_i, y_i) \in \mathbb{R}^2$。可能会有一个真实的函数 f^* 可以生成这些数据，但是我们并不知道它具体是哪个函数。所以用一个最简单的方法，我们会想到用线性模型拟合这些训练数据，

$$f_\theta(x) = ax + b = \begin{bmatrix} x & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}. \tag{1.9}$$

把系数记为 $\beta = (a, b)^T$。一个特殊的模型是令

$$\hat{\beta} = (X^T X)^{-1} X^T Y, \tag{1.10}$$

其中

$$X = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}. \tag{1.11}$$

这个 $\hat{\beta}$ 叫做 β 的最小二乘估计，因为它满足最小二乘方法

$$\hat{\beta} = \arg\min_\theta (X\theta - Y)^T (X\theta - Y), \tag{1.12}$$

从统计的角度看，我们会尝试着理解什么样的原始数据可以被这个模型很好的预测。下面我们会介绍一个高斯-马尔可夫定理 (Gauss-Markov) 以更好的理解统计的 “基于模型”。在线性回归模型中，如果误差满足零均值、同方差且互不相关，则回归系数的最佳线性无偏估计 (BLUE, Best Linear unbiased estimator) 就是普通最小二乘估计法。这里“最佳”的意思是：与其他的无偏线性估计相比，最佳线性无偏估计具有最小的方差。

Theorem 1 (Gauss-Markov). *假设数据 (X, Y) 是从下面的模型中采样的*

$$Y = X\beta + \epsilon \tag{1.13}$$

其中 $X \in \mathbb{R}^{n\times d}, Y, \epsilon \in \mathbb{R}^{n\times 1}$。假设 $X^T X$ 是可逆的，$E(\epsilon) = 0, Var(\epsilon) = \sigma^2 I_{n\times n}$。那么，
(i) $\hat{\beta}$ 是无偏的，i.e.

$$E(\hat{\beta}) = \beta. \tag{1.14}$$

(ii) 对于任何的 $\tilde{\beta} = CY$，

$$Var(\hat{\beta}) \le Var(\tilde{\beta}) \tag{1.15}$$

1.2.2 机器学习的角度

从机器学习的角度看，我们会更关心什么样的假设函数空间和训练算法可以用来拟合数据。我们通常不会基于特定的数据生成模型来设计一个特定的假设空间。因此一个好的方法应该是能从数据中提取有意义的特征的方法。这种数据驱动的方法被用于复杂的数据或者是大数据，它们的数据特征一般很难分析。

通过一个线性模型，我们会介绍机器学习的概念，包括训练误差、测试误差、泛化差距、梯度下降、随机梯度下降等。

同样假设我们有一组数据 $S = \{(x_i, y_i)\}_{i=1}^n, (x_i, y_i) \in \mathbb{R}^2$。用一个最简单的方法来学习这组数据，我们会想到用线性模型拟合，

$$h_i := f_\theta(x_i) = ax_i + b, \tag{1.16}$$

其中 $\theta = (a, b)$。当 a 和 b 变化时，$f_\theta(x_i)$ 表示不同的函数。这些函数形成了一个函数空间。我们希望当给定一组数据的时候，我们能从这个函数空间找到一个某种意义上最佳的函数来逼近这组数据所代表的背后的真实函数。为了刻画模型的效果，我们需要定义一种损失函数 (loss function) 来刻画模型与真实函数的距离。因为真实函数并不是已知的，只有训练数据是可以获得的，因此，我们只能退而求其次，把模型在训练集上逼近得有多好用数学公式表达出来，但我们需要记得的是，我们关心的其实是泛化误差 (generalization error)，也就是那些没有被用来训练的测试集上的误差。一种用的比较广泛的损失函数是均方差 (Mean Squared Error, MSE):

$$L_S = \frac{1}{2n} \sum_{i=1}^{n} (y_i - h_i)^2. \tag{1.17}$$

注意这里分母上的 2 只是为了计算方便。既然 L_S 是在训练集上定义的，所以它也被称为训练误差（training error）或者经验风险（empirical risk）。我们的目标是最小化训练误差，即

$$\min_{a,b} L_S.$$

这就是广为人知的经验风险最小化问题 (ERM, Empirical Risk Minimization)。

在经过一些训练方法优化以后（如下节介绍的梯度下降），模型 (1.16) 可以比较准确地预测 x_i 对应的 y_i。在线性模型，均方差和梯度下降的作用下，我们可以得到和 (1.10) 一样的解。

对于一个一般的机器学习方法，比如神经网络方法，我们并不完全理解这些方法，要刻画这些方法的性能，我们要计算那些我们更关心的没有被训练过的数据的预测准确率。理论上，我们关心从分布 $\mathcal{D}$ 中抽出的数据，它的群体风险 (population risk) 或者泛化误差 (generalization error) 是

$$L_{\mathcal{D}} = \mathbb{E}_{x \sim \mathcal{D}} \frac{1}{2} (y(x) - f_\theta(x))^2.$$

实际上，我们只能采集另外一组测试数据 $\{x_j^{test}, y_j^{test}\}_{i=1}^{n}$。那么测试误差被定义为：

$$L_{test} = \frac{1}{2n'} \sum_{j=1}^{n'} (y_j^{test} - f_\theta(x_j^{test}))^2.$$

通常我们把训练误差和测试误差的差值定义为泛化差距 (generalization gap)，即是

$$L_{gap} = L_{\mathcal{D}} - L_S.$$

在一般过程中，我们可以做到将 L_S 训练的比较小，而对于实际使用的模型来说，同时要求泛化误差 L_D 比较小。

1.2.3 区别与联系

统计学习不需要训练过程，它是根据理论上对数据模型的假设，通过观测数据将算法的参数估计出来，并且对误差有一定的保证。但是它有一个明显的缺陷，也就是迁移性比较差。对不同的数据模型，需要发展不同的算法。而对于大量的真实数据，其底层模型本身是很难刻画的。

机器学习的优点是它不需要对数据模型有太多的假设，它有一套标准的流程可以通过数据去估计算法的参数。它重要的方面在于选择一个合适的假设函数空间，这个函数空间要足够大以便能够拟合复杂的函数，并且训练参数的方法要能够适应不同的数据，在尽量多类型的数据中都取得好的性能。

可见，从统计的角度，我们是从一个数据模型中去找一个合适的学习方法，而从机器学习的角度，我们是用一个学习方法去尽量的学习更多的数据模型。要针对每种数据模型去发展对应的学习方法，可能过于繁琐。而暴力地把一个普通的学习方法用在所有数据似乎又显得不太可能。对于线性模型，显然它是做不到的。但对于一个足够复杂的学习方法，其方法的特征可以被多种超参数所调节，比如神经网络的结构，宽度或者深度等，可以从不同的数据中提取合适的特征。为了理解为什么一个机器学习方法有效，我们需要类似统计学习的方式去证明这类机器学习方法在某些超参数下能学习好什么样的数据。我们希望能有一个强大的机器学习方法，同时又有解释性。因此，区分“机器学习角度”或者“统计学习角度”的意义并不很大。

1.3 梯度下降与随机梯度下降

在这节中，我们以线性模型来介绍梯度下降这个常用的优化方法。梯度下降是最小化 L_S 的一种比较常用的方法：在每一个训练步 (training step)，以 η 为学习步长（step size 或叫学习率，learning rate），按如下方式更新参数：

$$a_{t+1} = a_t - \eta \frac{\partial L_S}{\partial a}, \quad b_{t+1} = b_t - \eta \frac{\partial L_S}{\partial b}. \tag{1.18}$$

L_S 对 a 求导可知，

$$\frac{\partial L_S}{\partial a} = \frac{1}{n}\sum_{i=1}^{n}(h_i - y_i)\frac{\partial h_i}{\partial a} = \frac{1}{n}\sum_{i=1}^{n}(h_i - y_i)x_i. \tag{1.19}$$

注意1.19用到了所有的样本，它被称为梯度下降。如果我们随机的从所有样本中选取 $|B|$ 个样本来估计损失函数，即

$$L_B = \frac{1}{n}\sum_{i=i_1}^{i_{|B|}}(y_i - h_i)^2, \tag{1.20}$$

$$\frac{\partial L_B}{\partial a} = \frac{1}{n}\sum_{i=i_1}^{i_{|B|}}(h_i - y_i)x_i, \tag{1.21}$$

其中 $\{i_i, i_2, \cdots, i_{|B|}\}$ 在每一步可以是随机选取，也可以是提前固定分配好的。既然没有用到所有的样本，那么计算出来的梯度并不是精确的，所以被称为随机梯度下降法（stochastic gradient descent, SGD）。实际训练中，通常会把样本分成多份，每份的数据点数量为 $|B|$，每一份称为一个 batch，在每一个 epoch 中，按顺序一步取一个 batch 进行梯度下降，直到所有的 batch 都被使用过一次。然后再将数据打乱后再进行下一个 epoch。有的算法并不打乱数据而是直接进行下一个 epoch。SGD 的引入最初的想法只是为了解决计算量和内存的问题，当数据量太大时，一次性完全导入到内存进行计算的情况下，一般计算机的内存无法承受，计算的时间也会特别长。后来的实践发现，SGD 在泛化性上比普通的梯度下降会有优势。因此，对 SGD 的研究也是重要且热门的方向。

1.4 两层神经网络

神经网络是一个有许多参数的机器学习方法。它的基本单元是神经元，它从前面一层的神经元或者是输入 x 接收输入信息，然后通过激活函数 σ 输出到下一层：$\sigma(wx + b)$。常见的 $\sigma(x)$ 为 $\mathrm{ReLU}(x) = \max\{0, x\}$、$\tanh(x)$ 等。

假设数据集为 $\{(x_i, y_i)\}_{i=1}^n$，考虑一个只有一层隐藏层的神经网络，即两层神经网络，

$$f_\theta(x) = \sum_{j=1}^{m} a_j \sigma(w_j x + b_j).$$

为了方便起见，定义

$$h_i = f_\theta(x_i). \tag{1.22}$$

我们这里假设 x, w, b, a 都是一维的标量，对于高维的情形，后面会有介绍。与线性模型类似，我们可以通过（随机）梯度下降来调节参数。当梯度下降用在神经网络模型中时，它有一种特殊的名字：向后传播法。

1.5 向后传播

下面我们要用两层神经网络的训练来解释为什么叫向后传播法 (back propagation)。我们同样考虑均方差:

$$L_S = \frac{1}{2n}\sum_{i=1}^{n}(y_i - h_i)^2. \tag{1.23}$$

a_j 是最外层的参数，它的求导比较简单：

$$\frac{\partial L_S}{\partial a_j} = \frac{1}{n}\sum_{i=1}^{n}(h_i - y_i)\sigma(w_j x + b_j).$$

对于 w_j，由于它不是最外层的参数，我们需要多次用到链式求导法则：

$$\begin{aligned}
\frac{\partial L_S}{\partial w_j} &= \frac{1}{n}\sum_{i=1}^{n}(h_i - y_i)\frac{\partial h_i}{\partial w_j} \\
&= \frac{1}{n}\sum_{i=1}^{n}(h_i - y_i)\frac{\partial(\sum_{l=1}^{m} a_l\sigma(w_l x_i + b_l))}{\partial w_j} \\
&= \frac{1}{n}\sum_{i=1}^{n}(h_i - y_i)a_j\frac{\partial(\sigma(w_j x_i + b_j))}{\partial w_j} \\
&= \frac{1}{n}\sum_{i=1}^{n}(h_i - y_i)a_j\frac{\partial\sigma(x)}{\partial x}|_{x=w_j x_i+b_j}\frac{\partial(w_j x_i)}{\partial w_j} \\
&= \frac{1}{n}\sum_{i=1}^{n}(h_i - y_i)a_j\frac{\partial\sigma(x)}{\partial x}|_{x=w_j x_i+b_j}x_i.
\end{aligned}$$

因为链式法则，梯度下降的计算顺序是 $h_i \to a_j \to \sigma \to x_i$，而信息流 (information flow) 的顺序是 $x_i \to \sigma \to a_j \to h_i$，这两者的顺序相反，因此称其为向后传播。

下面用矩阵的形式写 $f_\theta(\boldsymbol{x})$。

$$f_\theta(\boldsymbol{x}) = \boldsymbol{W}^{[2]}\sigma(\boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}),$$

其中 $\boldsymbol{x} \in \mathbb{R}^{d\times 1}, \boldsymbol{W}^{[1]} \in \mathbb{R}^{m\times d}, \boldsymbol{b}^{[1]} \in \mathbb{R}^{m\times 1}, \boldsymbol{W}^{[2]} \in \mathbb{R}^{d_o\times m}$。这里 d 是输入数据的维度，m 是隐藏层神经元的个数，d_o 是输出维度。有时我们会同时计算 n 个样本的值，比如在实际编程计算中，我们把 $\boldsymbol{x}$ 写成矩阵形式：

$$\boldsymbol{Y} = h(\boldsymbol{X}) = \boldsymbol{W}^{[2]}\sigma(\boldsymbol{W}^{[1]}\boldsymbol{X} + \boldsymbol{B}^{[1]}),$$

其中 $\boldsymbol{X} \in \mathbb{R}^{d\times n}, \boldsymbol{Y} \in \mathbb{R}^{[d_o\times n]}, \boldsymbol{B}^{[1]} \in \mathbb{R}^{m\times n}$。$\boldsymbol{B}^{[1]}$ 的每一列都为 $b^{[1]}$，即 $\boldsymbol{B}^{[1]} = [b^{[1]}, b^{[1]}, \cdots, b^{[1]}]$。用这些符号可以类似得定义一个深度神经网络。

1.6 机器学习的基本概念

机器学习主要包含了三个类别：

(1) 监督学习 (supervised learning)：带标签的学习

(2) 无监督学习 (unsupervised learning)：不带标签的学习（聚类，维度下降等等）

(3) 强化学习 (reinforcement learning)：在一个动力系统里学习做最优的决策

本书我们主要关注监督学习。

回归问题与分类问题 在机器学习中，我们一般可以研究两类问题，分别为分类问题和回归问题。

分类问题和回归问题的本质是一样的，都是对输出的一种预测。而他们也有不相同的地方，比如分类问题的输出一般为离散的变量，而回归问题的输出一般为连续的变量。当然在这里我们也可以对连续变量取整，这样就可以对连续变量按照分类问题的模式进行训练。

对于回归问题，我们可以举以下例子:

$$\hat{Y}_i = f_\theta(X_i),\ \text{Loss}= \frac{1}{2}\sum \left(Y_i - \hat{Y}_i\right)^2 \tag{1.24}$$

其中 $\{X_i, Y_i\}$ 为数据集，损失函数为均方差函数。

而对于分类问题，比如猫狗的二分类问题，若给猫的标签为 1，狗的标签为 2，那自然的问题是为什么狗比猫大。为了保证所有类别具有同等地位，我们一般以 one-hot 向量来表示标签，

- 第一类: [1,0,0,...]
- 第二类: [0,1,0,...]

如果属于某一类，那么就将其对应的独热 (one-hot) 向量分量记为 1。同时，我们一般使用交叉熵来定义损失函数，

$$H(p,q) = -\sum_i p_i \log q_i = -y\log\hat{y} - (1-y)\log(1-\hat{y}) \tag{1.25}$$

其中 $i = 0, 1$ 表示两种状态，同样 $y \in \{0, 1\}$ 表示真实值，$\hat{y} \in (0, 1)$。我们可以看到，当 y 与 $\hat{y}$ 十分相近的时候，其交叉熵小，而当 y 与 $\hat{y}$ 相差较大的时候，交叉熵会很大。

对 n 个样本，C 个类别，$\boldsymbol{y}_i \in \mathbb{R}^C$, 我们就可以用交叉熵来定义损失函数：

$$H(p,q) = -\sum_{i=1}^{n}\sum_{j=1}^{C} y_{ij}\log\hat{y}_{ij} - (1-y_{ij})\log(1-\hat{y}_{ij}) \tag{1.26}$$

在这里 i 代表样本指标，j 代表类别指标。

Remark 1. 回归问题与分类问题最大的不同为标签是否可交换，对于分类问题，比如对于猫狗的二分类问题，将 $[1,0]$ 分配给猫或者狗应该不影响问题的定义和学习，也就是标签可以打乱顺序。而对于回归问题则不可以，因为打乱顺序会影响数据本身的逻辑意义，比如相同体积的物质，密度越大，质量越大，若我们随意分配密度对应的质量，那整个问题的学习就毫无意义。

1.7 一些常用的概念及符号

数据集 (dataset) 数据集 $S = \{\boldsymbol{z}_i\}_{i=1}^n = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ 是从分布 $\mathcal{D}$ 中采样的，$\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$。$\mathcal{X}$ 被称为样本集 (instance set)，$\mathcal{Y}$ 被称为标签集 (label set)。通常 $\mathcal{X} \in \mathbb{R}^d, \mathcal{Y} \in \mathbb{R}^{d_o}$，这里的 d 是输入维数，d_o 是输出维数，$n = \#S$ 是样本的数量。一般没有特殊说明时，S 和 n 都是训练集的指标。

函数 (function) 如果存在一个真实（目标）函数，把它记做 f^* 或 $f : \mathcal{X} \to \mathcal{Y}$，它满足 $\boldsymbol{y}_i = f^*(\boldsymbol{x}_i), i = 1, \cdots, n.$ 定义假设空间 (hypothesis space) 为 $\mathcal{H}$，假设函数 (hypothesis function) 记为 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) \in \mathcal{H}$ 或 $f(\boldsymbol{x}; \boldsymbol{\theta}) \in \mathcal{H}, f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$。$\boldsymbol{\theta}$ 为 $f_{\boldsymbol{\theta}}$ 中参数的集合。

损失函数 (loss function) 损失函数记为 $\ell : \mathcal{H} \times \mathcal{Z} \to \mathbb{R}_+ := [0, +\infty)$，它测量预测标签与真实标签之间的差距。譬如说 L^2 损失：

$$\ell(f_{\boldsymbol{\theta}}, z) = \frac{1}{2}(f_{\boldsymbol{\theta}}(\boldsymbol{x}) - \boldsymbol{y})^2,$$

其中 $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{y})$。$\ell(f_{\boldsymbol{\theta}}, \boldsymbol{z})$ 也可以写成 $\ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y})$。对于数据集 $S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$，训练误差 (empirical risk or training loss) 被定义为 $L_S(\boldsymbol{\theta})$ 或 $L_n(\boldsymbol{\theta})$ 或 $R_n(\boldsymbol{\theta})$ 或 $R_S(\boldsymbol{\theta})$，

$$L_S(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i). \tag{1.27}$$

期望误差 (population risk or expected loss) 被定义为 $L_{\mathcal{D}}(\boldsymbol{\theta})$ 或 $R_{\mathcal{D}}(\boldsymbol{\theta})$

$$L_{\mathcal{D}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathcal{D}} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}), \boldsymbol{y}). \tag{1.28}$$

激活函数 (activation function) 激活函数记为 $\sigma(x)$

Example 1. 列举一些常用的激活函数：

1. $\sigma(x) = \text{sigmoid}(x) = \frac{1}{1+e^{-x}}$;

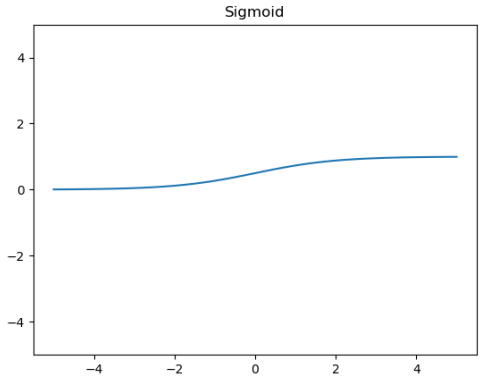


图 1.5: Sigmoid(x)

2. $\sigma(x) = \tanh(x)$*;*

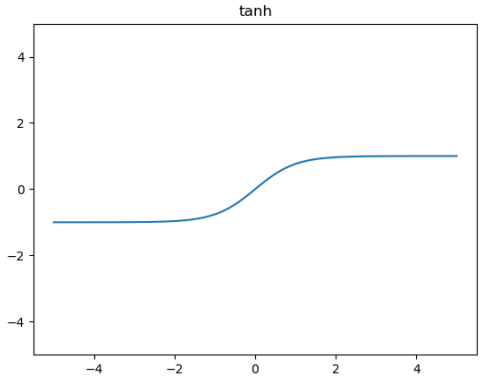


图 1.6: tanh(x)

3. $\sigma(x) = \cos x, \sin x$*;*

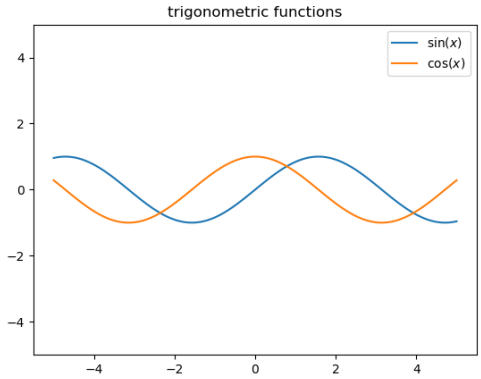


图 1.7: $\sin(x)$ and $\cos(x)$

4. $\sigma(x) = \text{ReLU}(x) = \max(0, x)$*;*

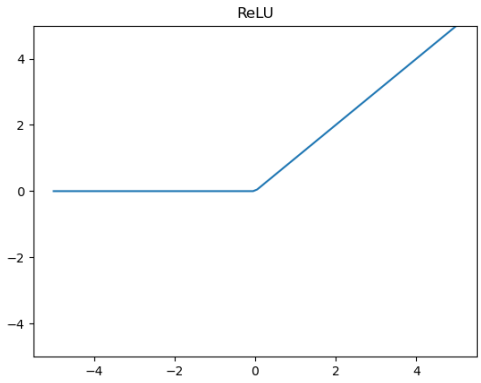


图 1.8: $\text{ReLU}(x)$

5. $\sigma(x) = \text{SiLU}(x) = x \times \text{sigmoid}(x)$*;*

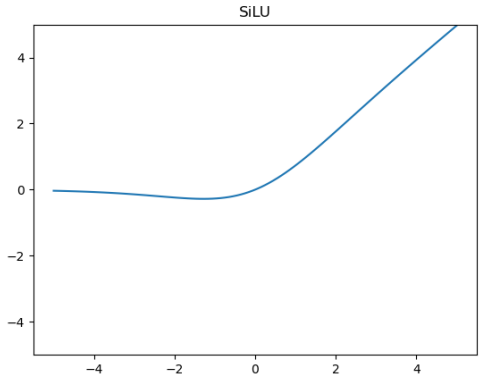


图 1.9: SiLU(x)

6. $\sigma(x) = \text{LeakyReLU}(x) = \max(0, x) + a * \min(0, x)$ *where* a *is a negative slope to control the angle;*

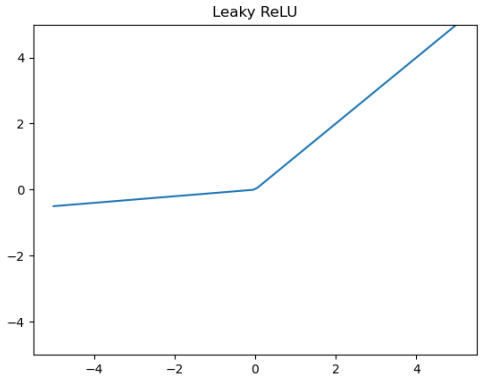


图 1.10: LeakyReLU(x)

7. $\sigma(x) = \text{GELU}(x) = x \times \Phi(x)$ *where* $\Phi(x)$ *is the Cumulative Distribution Function for Gaussian Distribution.*

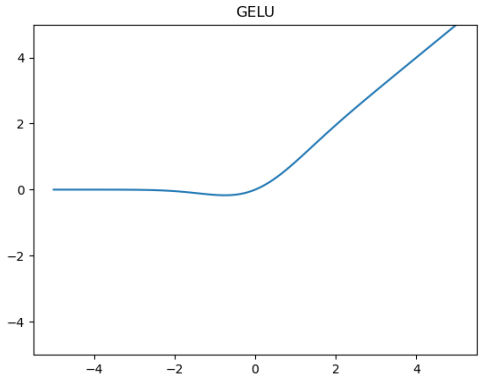


图 1.11: GELU(x)

Remark 2. $\sigma(x)$ 一般不是多项式，因为 $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{m} a_j \sigma(\boldsymbol{w}_j \boldsymbol{x} + b_j)$，如果 $\sigma(x)$ 是二次多项式，则 $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ 也为二次多项式，它无法表示更复杂的式子。

两层神经网络 (two-layer neural network) 隐藏层 (hidden layer) 的神经元数量被记为 m。两层神经网络为：

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=1}^{m} a_j \sigma(\boldsymbol{w}_j \cdot \boldsymbol{x} + b_j), \tag{1.29}$$

其中 σ 为激活函数 (activation function)，$\boldsymbol{w}_j$ 为输入权重 (input weight)，a_j 为输出权重 (output weight)，b_j 为偏置项 (bias)。我们把参数集记为 $\boldsymbol{\theta} = (a_1, \cdots, a_m, \boldsymbol{w}_q, \cdots, \boldsymbol{w}_m, b_1, \cdots, b_m)$.

一般神经网络 (general deep neural network) 一个 L 层神经网络记为

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{W}^{[L-1]} \sigma \circ (\boldsymbol{W}^{L-2} \sigma \circ (\cdots (\boldsymbol{W}^{[1]} \sigma \circ (\boldsymbol{W}^{[0]} \boldsymbol{x} + \boldsymbol{b}^{[0]}) + \boldsymbol{b}^{[1]}) \cdots) + \boldsymbol{b}^{[L-2]}) + \boldsymbol{b}^{[L-1]}, \tag{1.30}$$

其中 $\boldsymbol{W}^{[l]} \in \mathbb{R}^{m_{l+1} \times m_l}, \boldsymbol{b}^{[l]} \in \mathbb{R}^{m_{l+1}}, m_0 = d_{in} = d, m_L = d_o$，$\sigma$ 是一个向量函数，“$\circ$”的意思是对应元素的运算 (entry-wise operation)（例如对应元素相乘）。注意一下在计算网络层数时，输入层不计入（即层数为隐藏层 + 输出层的层数）我们把参数集记为

$$\boldsymbol{\theta} = (\boldsymbol{W}^{[0]}, \boldsymbol{W}^{[1]}, \cdots, \boldsymbol{W}^{[L-1]}, \boldsymbol{b}^{[0]}, \boldsymbol{b}^{[1]}, \cdots, \boldsymbol{b}^{[L-1]}),$$

把 $\boldsymbol{W}^{[l]}$ 中的元素记为 $\boldsymbol{W}_{ij}^{[l]}$。也可以用递归的方法定义这个网络：

$$f_{\boldsymbol{\theta}}^{[0]}(\boldsymbol{x}) = \boldsymbol{x} \tag{1.31}$$

$$f_{\boldsymbol{\theta}}^{[l]}(\boldsymbol{x}) = \sigma \circ (\boldsymbol{W}^{[l-1]} f_{\boldsymbol{\theta}}^{[l-1]}(\boldsymbol{x}) + \boldsymbol{b}^{[l-1]}), \quad 1 \leq l \leq L-1 \tag{1.32}$$

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = f_{\boldsymbol{\theta}}^{[L]}(\boldsymbol{x}) = \boldsymbol{W}^{[L-1]} f_{\boldsymbol{\theta}}^{[L-1]}(\boldsymbol{x}) + \boldsymbol{b}^{[L-1]} \tag{1.33}$$

Remark 3. *万有逼近定理 (Universal Approximation Theorem)*：如果 $\sigma(\boldsymbol{x})$ 是非线性函数且不是多项式，那么对于任意的函数 $f(\boldsymbol{x})$，只要神经元数目 m 足够大，那么 $f_\theta(\boldsymbol{x})$ 可以以任意的精度逼近 $f(\boldsymbol{x})$。这个定理说明了神经网络是一个非常强大的表示器。*[加参考文献?? ??]* 但同时，我们也应该注意逼近定理所构造的解和实际训练中找到的解一般是不一样的。因此，一个神经网络有能力逼近只是一个好模型的必要条件，是否能真正拟合好一个给定函数还需要看训练过程的很多因素。

1.8 常用的损失函数

神经网络训练用到的损失函数根据任务不同而往往不同。一般来说，设计符合目标且容易训练的损失函数是一个机器学习任务中最重要的子任务之一。对于监督学习的任务，最简单的和最常用的损失函数是样本标签和神经网络输出之间的均方差，但对于类似对抗生成网络、解微分方程和强化学习等没有显式标签的问题，如何设计损失函数并不直接。我们将在后面的介绍中针对具体的问题讨论损失函数的设计。本节中，我们主要介绍均方误差损失、绝对误差损失和交叉熵。

1.8.1 均方误差损失 (mean squared error (MSE))

均方误差损失 (Mean Squared Error (MSE)) 损失是回归问题中最常用的一种损失函数，也被称为 L2 Loss。考虑数据集 $S = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^n$，以及模型 $f_{\boldsymbol{\theta}}(\boldsymbol{x})$, 其中 $\boldsymbol{\theta}$ 是可学习的参数。MSE 可以表达成：

$$R_{MSE}(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} (f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{y}_i)^2. \tag{1.34}$$

若数据有随机性，则 MSE 是高斯分布下，在最大似然角度下最合理的损失函数的形式。考虑一维的情况，我们可以假设模型预测与真实值之间的误差服从标准高斯分布（$\mu = 0, \sigma = 1$），则给定一个 x_i 模型输出真实值 y_i 的概率为

$$p_i = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(y_i - f_{\boldsymbol{\theta}}(x_i))^2}{2} \right).$$

我们进一步假设数据是独立采样的，因此，所有样本的联合概率为

$$L(S) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(y_i - f_{\boldsymbol{\theta}}(x_i))^2}{2}\right).$$

考虑对数似然函数 Log-Likelihood

$$LL(S) = \log(L(S)) = -\frac{n}{2}\log 2\pi - \sum_{i=1}^{n}\left(-\frac{(y_i - f_{\boldsymbol{\theta}}(x_i))^2}{2}\right).$$

最大化对数似然函数等价于最小化 MSE。

1.8.2 绝对误差损失

绝对误差损失 (Mean Absolute Error (MAE)) 损失是回归问题中最常用的一种损失函数，也被称为 L1 Loss。考虑数据集 $S = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{n}$，以及模型 $f_{\boldsymbol{\theta}}(\boldsymbol{x})$, 其中 $\boldsymbol{\theta}$ 是可学习的参数。MAE 可以表达成：

$$R_{MAE}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} |f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - \boldsymbol{y}_i|. \tag{1.35}$$

若数据有随机性，则 MAE 是拉普拉斯分布下，在最大似然角度下最合理的损失函数的形式。考虑一维的情况，我们可以假设模型预测与真实值之间的误差服从标准拉普拉斯分布，则给定一个 x_i 模型输出真实值 y_i 的概率为

$$p_i \sim \exp\left(-|y_i - f_{\boldsymbol{\theta}}(x_i)|\right).$$

我们进一步假设数据是独立采样的，因此，所有样本的联合概率为

$$L(S) \sim \prod_{i=1}^{n} \exp\left(-|y_i - f_{\boldsymbol{\theta}}(x_i)|\right).$$

考虑对数似然函数 Log-Likelihood

$$LL(S) = \log(L(S)) \sim \sum_{i=1}^{n}\left(-|y_i - f_{\boldsymbol{\theta}}(x_i)|\right).$$

最大化对数似然函数等价于最小化 MAE。

1.8.3 交叉熵 (Cross-Entropy)

在分类问题中，交叉熵 (Cross-Entropy) 比 MSE 或者 MAE 都更加常用，主要的原因是在分类问题中，模型的输出往往具有概率的解释。分类问题的标签一般是 0 或者 1，表示是或

者否。而模型的输出一般是连接的，并且很难只在 0 或者 1 这两个数之间切换。一般会通过加 softmax 使得每一个维度的输出在 0 到 1 之间，并且这个数可以解释成模型认为该样本属于该类别的概率。我们可以继续借用最大似然的想法来构造损失函数。考虑数据集 $S = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$，以及模型 $f_{\boldsymbol{\theta}}(\boldsymbol{x})$, 其中 $\boldsymbol{\theta}$ 是可学习的参数。所有独立采样的样本的对数似然为

$$LL(\boldsymbol{\theta}) = \log \prod_{i=1}^{n} (f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^{y_i} (1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i))^{(1-y_i)}, \tag{1.36}$$

$$LL(\boldsymbol{\theta}) = \sum_{i=1}^{n} y_i \log f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) + (1 - y_i) \log(1 - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)). \tag{1.37}$$

这就是常用的交叉熵。如果输出是高维的，则将每一维的损失加起来，一般还要取个平均。这个形式也是常用的 Kullback–Leibler Divergence，常用来刻画两个分布的距离，但注意这个函数在两个分布的位置交换以后值并不相等，因此，它并不是真正的距离。

1.9 常用的优化方法

接下来，将介绍一些目前常用的优化方法 (Ruder 2016).

1.9.1 梯度下降法（GD）

$$g_t = \nabla_\theta R(\theta_{t-1}), \tag{1.38}$$

$$\theta_t = \theta_{t-1} - \eta g_t. \tag{1.39}$$

η 是学习率。GD 的学习比较慢，对于各个方向曲率差异过大的损失函数，容易产生振荡。

1.9.2 动量（momentum）

$$g_t = \nabla_\theta R(\theta_{t-1}), \tag{1.40}$$

$$m_t = \mu m_{t-1} + g_t, \tag{1.41}$$

$$\theta_t = \theta_{t-1} - \eta m_t. \tag{1.42}$$

μ 是一个超参数, m_0 可以设为 0. 动量的想法类似于保留上一步的速度的一部分。

1.9.3 Nesterov

Nesterov 直接考虑在动量作用下跑到的位置后，计算新的梯度，再与动量共同组合作为下降的方向。

$$g_t = \nabla_\theta R(\theta_{t-1} - \eta\mu m_{t-1}), \tag{1.43}$$

$$m_t = \mu m_{t-1} + g_t, \tag{1.44}$$

$$\theta_t = \theta_{t-1} - \eta m_t. \tag{1.45}$$

1.9.4 Adagrad

学习率过大会使训练末期发生振荡，现在有很多研究讨论了这个研究。为了解决这个问题，可以采用自适应的学习率。

$$g_t = \nabla_\theta R(\theta_{t-1}), \tag{1.46}$$

$$n_t = n_{t-1} + g_t^2, \tag{1.47}$$

$$\theta_t = \theta_{t-1} - \eta g_t / \sqrt{\epsilon + n_t}. \tag{1.48}$$

这个方法会带来新的问题，也就是在训练末期，由于 n_t 太大，导致学习太慢。ϵ 目的是让分母不为零。

1.9.5 Adadelta

Adadelta 采用一种归一化的方式，在每个维度上类似于只考虑其方向，用统一学习率来控制变化大小。

$$g_t = \nabla_\theta R(\theta_{t-1}), \tag{1.49}$$

$$n_t = \nu n_{t-1} + (1-\nu) g_t^2, \tag{1.50}$$

$$\theta_t = \theta_{t-1} - \eta g_t / \sqrt{\epsilon + n_t}. \tag{1.51}$$

ν 是一个超参。

1.9.6 RMSProp

RMSProp 是 Adadelta 中 $\nu = 0.5$ 的情况。

1.9.7 Adam

Adam 是综合了动量和自适应学习率的方法。

$$g_t = \nabla_\theta R(\theta_{t-1}), \tag{1.52}$$

$$n_t = \nu n_{t-1} + (1-\nu) g_t^2, \tag{1.53}$$

$$m_t = \mu m_{t-1} + (1-\mu) g_t, \tag{1.54}$$

$$\hat{m}_t = \frac{m_t}{1-(\beta_1)^t}, \tag{1.55}$$

$$\hat{n}_t = \frac{n_t}{1-(\beta_2)^t}, \tag{1.56}$$

$$\theta_t = \theta_{t-1} - \eta \hat{m}_t / \sqrt{\epsilon + \hat{n}_t}. \tag{1.57}$$

ν, μ, β_1, β_2, η 为超参数。

1.10 训练中的一些问题

1.10.1 参数的初始化

我们以全连接网络来举例，m_{in} 和 m_{out} 分别表示输入当前层的的参数维度以及当前层的宽度，即一个隐藏层对应的参数为 $\boldsymbol{\theta} \in \mathbb{R}^{m_{out} \times m_{in}}$，则有下列一些比较常见的初始化：

Lecun Initializatioin LeCun et al. (2012)

- 满足正态分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{N}(0, \frac{1}{m_{in}})$。
- 满足均匀分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{U}(-\sqrt{\frac{3}{m_{in}}}, \sqrt{\frac{3}{m_{in}}})$。

Xavier Initialization Glorot & Bengio (2010)

- 满足正态分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{N}(0, \frac{2}{m_{in}+m_{out}})$。
- 满足均匀分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{U}(-\sqrt{\frac{6}{m_{in}+m_{out}}}, \sqrt{\frac{6}{m_{in}+m_{out}}})$。

Kaiming Initialization He et al. (2015)

- 满足正态分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{N}(0, \frac{2}{m_{in}})$。
- 满足均匀分布的初始化 $\theta \in \boldsymbol{\theta} \sim \mathcal{U}(-\sqrt{\frac{6}{m_{in}}}, \sqrt{\frac{6}{m_{in}}})$。

注意到以上三种常见的初始化不论是正态分布还是均匀分布的形式，都存在随着网络宽度增加而减小的性质，这主要是因为在设计这些初始化时需要满足计算神经网络的信号 $\boldsymbol{x}$ 在 $l-1$ 和 l 层之间流动时强度不变，

$$\boldsymbol{x}^{[l]} = \boldsymbol{W}^{[l]}\sigma(\boldsymbol{x}^{[l-1]}) + \boldsymbol{b}^{[l]}, \tag{1.58}$$

即如果我们用(1.58)表示信号在两层中流动，则我们需要满足 $\mathrm{Var}(\boldsymbol{x}_j^{[l]}) = \mathrm{Var}(\boldsymbol{x}_i^{[l-1]})$，这就意味着神经网络的参数需随网络宽度增加而减小。

如果想要了解更多的初始化，可以参考Narkhede et al. (2022)。此外不同的初始化往往会导致神经网络拥有不同的训练动力学，这将在后续的章节中详细说明。

1.10.2 梯度消失及残差连接

由于链式法则，在求损失函数关于参数的导数时，会得到一串表达式相乘的情况。对于很深的网络，可能会出现这些相乘后的数很大导致训练不稳定，或者数很小导致无法训练。在 He et al. (2016) 的文章中，发现 34 层的网络的效果比 18 层的网络的效果要差。具体来说，34 层的网络同样可以达到 28.54% 的错误率，而 18 层的网络达到 27.94%。显然，34 层网络能够达到还不错的结果，说明其并没有完全梯度消失，只是训练速度比较慢。为了解决深层网络训练困难的问题，He et al. (2016) 提出了一种残差连接网络。最简单的一种残差连接为

$$F_{l+1} = \sigma(W_l F_l + b_l) + F_l,$$

其中，F_l 是第 l 层的输出。这种情况在 F_l 和 F_{l+1} 的维度一样的时候适用。当它们的维度不一样时，可以用下面的形式

$$F_{l+1} = \sigma(W_l F_l + b_l) + W_s F_l,$$

其中 W_s 是一个可训练的矩阵，其目的是将维度转化为 $l+1$ 层的维度。通过残差连接，34 层的错误率可以降到 25.03%，而 18 层可以降到 27.88%。这个例子的训练过程可以参见图1.12。

1.10.3 学习率与训练不稳定性

在训练神经网络的过程中，学习率是一个至关重要的超参数，只有选取合适的学习率才能顺利达到比较好的训练效果。以梯度下降为例，如果学习率过小，则会如图1.13左图所示，神经

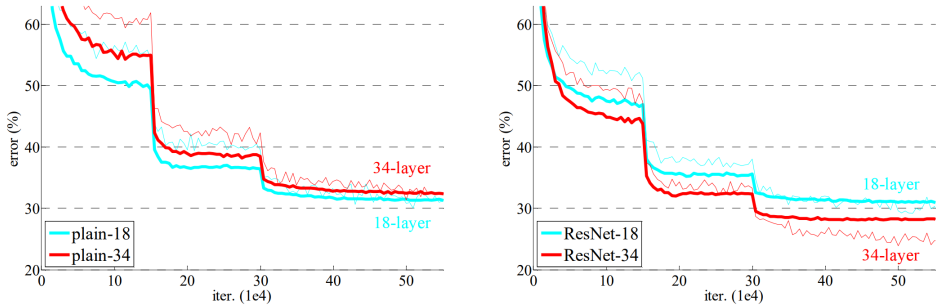


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

图 1.12: 残差连接提升深层网络的训练。图来源于He et al. (2016)。

网络需要很多个 epoch 才能够达到极小点，这会造成计算资源的浪费；而当学习率过大，则会如图1.13右图所示，神经网络会震荡的离开这个局部极小，给神经网络训练带来不稳定性。我们从数学的角度来理解这种不稳定性，以二次函数

$$L = \frac{1}{2}\lambda\theta^2$$

为例，在高维空间中 λ 为损失函数 hessian 矩阵的特征值，η 为学习率，根据梯度下降更新参数

$$\theta_{t+1} = \theta_t - \eta\frac{\partial L}{\partial \theta},$$

线性稳定要求

$$|\theta_{t+1}| < |\theta_t|,$$

即

$$|1 - \lambda\eta| < 1,$$

则我们有学习率 η 与 λ 的关系：

$$\eta < \frac{2}{\lambda} \tag{1.59}$$

这说明只有学习率小于 $\frac{2}{\lambda}$ 时，才能保证神经网络的训练过程在梯度下降的方法下稳定，关于随机梯度下降下的稳定性分析，可以参考Wu et al. (2018)。

1.11 神经网络研究的一些问题

神经网络目前有很多不清楚的地方，这节中，我们将列举四个问题。

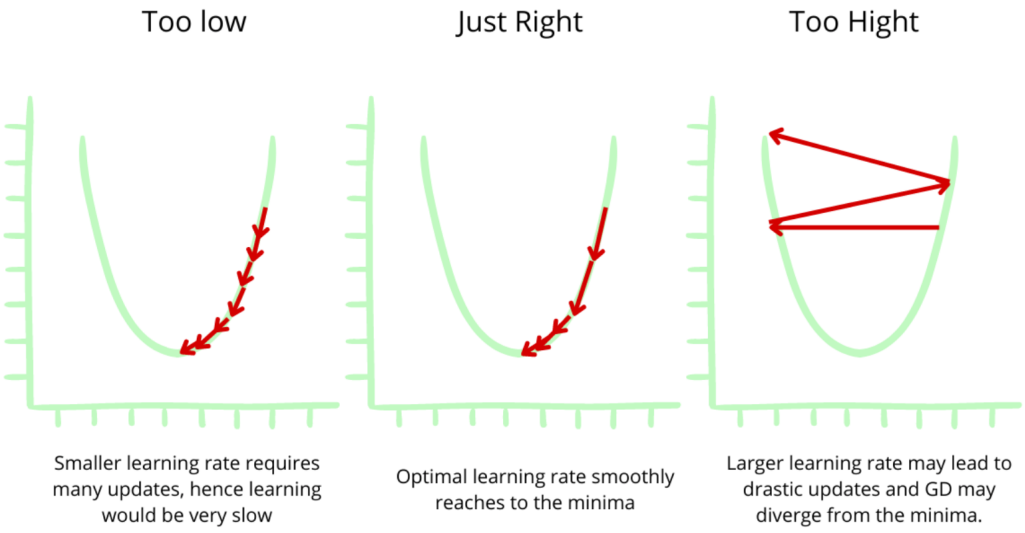


图 1.13: 梯度下降下不同学习率对于神经网络在局部极小的动力学演化示意图。图源：https://www.enjoyalgorithms.com/blog/loss-and-cost-functions-in-machine-learning

误差分析 神经网络的误差分析一直是研究的重点，包括逼近误差、泛化误差、训练误差。逼近误差指的是神经网络所形成的函数空间每一类误差都仍然是现在研究的热点问题。

多层网络的优势 1、多层网络有什么好处？实验上多层网络的泛化性较好，训练速度变快。2、为什么深层神经网络的效果会比浅层的网络好很多？（目前还没有定量的解释）从逼近论的角度看，深层网络可以使用较少的神经元达到浅层网络同样的效果。现在大部分理论研究都集中于两层网络。对于多层网络，它相对于浅层网络的优势如何定量刻画，以及在理论上如何解释都是非常困难的问题。

训练 理解为什么简单优化算法，例如 SGD，可以在很大的参数空间中找到泛化好的解以及如何理解随机优化算法的隐式正则效应对实际训练具有重要意义。

高维问题 (high-dimensional problems) 为什么 DNN 可以克服维数灾难？

下面我们针对维数灾难做更多描述。首先，我们以数值积分的例子来介绍传统算法经常会

遇到的维数灾难。最简单直观的是数值积分问题。令

$$I = \int_0^1 f(x)\mathrm{d}x. \tag{1.60}$$

如果我们用梯形公式计算每一个长度为 h 的均匀格子面积，则

$$I_n = \frac{h}{2}(f(0) + f(1)) + h\sum_{i=1}^{n-1} f(x_i), \tag{1.61}$$

其中 $h = \frac{1}{n}, x_i = ih$。误差的上界则为：

$$|I - I_n| \leq \frac{h^2}{8}\|f''\|_\infty = \frac{n^{-2}}{8}\|f''\|_\infty. \tag{1.62}$$

对于 d 维，也有类似结论：

$$|I - I_n| \leq \frac{h^{\frac{2}{d}}}{8}\|f''\|_\infty = \frac{n^{-\frac{2}{d}}}{8}\|f''\|_\infty. \tag{1.63}$$

结论中的 $\frac{1}{d}$ 可以通过 d 维空间的等距网格划分理解：假设一维时填充 $[0,1]$ 区间的等距划分需要 n 个点（假设划分数为 n），那么对于 d 维空间，同样距离为 $\frac{1}{n}$ 的等距划分需要 n^d 个点。

Remark 4. *以 MNIST 数据集中的样本为例，它的每一张图都是由 $28 \times 28 = 784$ 个像素点组成（每个像素点都是 0 或 1），那么如果要填满这个空间，至少需要 2^{784} 个样本，显然不可能提供那么多的样本用于训练。*

蒙特卡洛插值。这个方法很好的解决了维数灾难的问题。令 $\{x_i\}_{i=1}^n$ 为均匀分布在 $[0,1]^d$ 上的随机独立采样点，得到 $\{x_i, f(x_i)\}_{i=1}^n$。接着我们用这些采样点上的值来估计积分：

$$I = \int_{[0,1]^d} f(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \approx \frac{1}{n}\sum_{i=1}^n f(x_i). \tag{1.64}$$

用期望的定义知 $\mathbb{E}I_n = I$. 下面计算方差：

$$\mathbb{E}(I - I_n)^2 = \mathbb{E}(1 - \frac{1}{n}\sum_{i=1}^{n} f(x_i))(1 - \frac{1}{n}\sum_{i=1}^{n} f(x_i)) \tag{1.65}$$

$$= \mathbb{E}\frac{1}{n}\sum_{i=1}^{n}(I - f(x_i)) \cdot \frac{1}{n}\sum_{j=1}^{n}(I - f(x_j)) \tag{1.66}$$

$$= \frac{1}{n^2}\sum_{i,j=1}^{n}\mathbb{E}(I - f(x_i))(I - f(x_j)) \tag{1.67}$$

$$= \frac{1}{n^2}\sum_{i=j}\mathbb{E}(I - f(x_i))(I - f(x_j)) + \frac{1}{n^2}\sum_{i\neq j}\mathbb{E}(I - f(x_i))(I - f(x_j)) \tag{1.68}$$

$$= \frac{1}{n^2}\sum_{i=1}^{n}\mathbb{E}(I - f(x_i))^2 + 0 \tag{1.69}$$

$$= \frac{Var(f)}{n}, \tag{1.70}$$

其中 $Var(f)$ 是 f 的方差。所以它的误差大约为 $n^{-\frac{1}{2}}$（方差的精度为 $\frac{1}{n}$，所以误差即标准差的精度为 $n^{-\frac{1}{2}}$），这与维数 d 无关。蒙特卡洛插值法现在已经得到了很好的发展，在统计物理领域，人们经常用这个方法处理高维数值积分问题。深度神经网络的形式与蒙特卡洛采样类似，比如

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{m}\sum_{j=1}^{m} a_j\sigma(\boldsymbol{w}_j\boldsymbol{x} + b_j),$$

我们可以认为是从函数空间去采样神经元来逼近目标函数，这个形式和蒙特卡洛一样，所以我们认为它的精度可能与神经元数量 m 有关，所以不会遇到维数灾难的问题。

Remark 5. 前面给出的积分问题的例子之所以有维数灾难，与均匀的（有规律的）划分有关，而蒙特卡洛采样则用了采样的随机性很好的避免了维数灾难。

1.12 深度学习的历史简介

深度学习的应用很广，逐渐渗透到社会生活、工业生产、国防、科学研究等等。比如说下棋、玩游戏、人脸识别、图像分割、工业设计、解方程等。它如此之强大，以至于我们开始担心它所代表的人工智能会不会对人类造成威胁。更加理性地考虑关于深度学习的问题可以让我们更好的利用深度学习以及应对深度学习带来的变化。一种自然的考虑方式是理清以下三个问题，它是谁，它从哪里来，它将到哪里去。带着这些问题我们来快速浏览一次深度学习的历史。

单个生物神经元的组成可以简化成三个组成部分，树突部分将其它与之连接的神经元的信息收集到胞体部分。如果胞体的信号强度大于某个阈值，胞体将产生一个脉冲信号经由轴突传

递给所有它连接的神经元。人脑大约有 10^{11} 个神经元，平均每个神经元与 10^3 个其它神经元相互连接，构成极其复杂而富有结构的神经网络。我们从接受输入到做出反应，极少是由单个神经元就能完成的，通常都是部分神经元构成的子网络的共同结果。单个神经元似乎很简单，网络的群体效应又极难理解，是否这些神经元简单的相互作用，就足以学习到复杂的功能，最终形成了意识？

神经生理学家 Warren MuCulloch (1898-1969) 和逻辑学家 Walter Pitts (1923-1969) 在 1943 年设计了第一个 M-P 仿生神经元或叫感知器 (perceptron)。这个器件的功能是对输入 x 做一个仿射变换成 $wx+b$，类似胞体整合信号。取一个阈值 T，如果 $wx+b>T$，那器件输出 1，否则输出 0。给定合适的参数，这个器件可以完成基本的逻辑运算，比如 AND，OR，NOT 等。

1956 年，Marvin Minsky 和 John McCarthy 和 Claude Shannon 以及 Nathan Rochester 在 Dartmouth 组织召开了一次会议，也被指为人工智能诞生的标志。这个会议指出：“学习的每一个方面或智能的任何其他特征都可以被如此精确地描述，以至于可以制造一台机器来模拟它”并第一次明确提出了人工智能 (artificial intelligence) 这个词。然而，我们要注意，此到的所谓智能是一种设计的智能，而非学习的智能，此时的 M-P 神经元完全不能从数据中学习任何东西！

1958 年，Frank Rosenblatt 设计了一个针对 M-P 感知器的学习规则，其基本思路来源于心理学家 Donald Hebb 基于现象观察提出的学习法则，两个神经元的连接强度遵循越同时发放，连接越强 (Firing together, wiring together)。我们来细看一下过程。第 j 个神经元通过权重 w_{ji} 加权整合其它神经元对它的输入，然后通过跳跃函数 $f(x)$ 来确定输出。

$$f(x)=\begin{cases}1 & x\geq T\\ 0 & x<T\end{cases}, \tag{1.71}$$

$$y_j=f\left(\sum_{i=1}^{m}w_{ji}x_{ji}\right), \tag{1.72}$$

若神经元的期望输出是 d_j, Rosenblatt 提出的权重的变化如下

$$w_{ji}(t+1)=w_{ji}(t)-\eta(d_j-y_j)x_{ji} \tag{1.73}$$

其中 η 是学习率。实际上，若考虑均方误差，再用梯度下降，其变化应该如下

$$w_{ji}(t+1)=w_{ji}(t)-\eta\frac{df}{dx}(d_j-y_j)x_{ji} \tag{1.74}$$

但注意到函数 $f(x)$ 是跳跃的，不可导，因此 Rosenblatt 直接丢掉了不可计算的一项。因此 Rosenblatt 的更新法则实际上是一种阉割的梯度下降法。但无论如何，此时的神经网络已经是

一个可学习的网络了。这足以让人以其为基点，在想象中撬动地球。在 1958 年，基于 Rosenblatt 的表述，纽约时报将 perceptron 描述为“电子计算机的胚胎，期望它能走路、说话、看、写、自我复制并意识到自己的存在。”一言以蔽之，这是典型的“见得风，是个雨”。既然前面由于跳跃带来的不可导并没有实际体现在学习规则中，为什么不直接把不可导的阈值函数丢掉呢？1960 年 Bernard Widrow 和 Tedd Hoff 直接考虑没有 f(x) 的线性感知器，把它叫做“ADALINE”(Adaptive linear neuron)，并在物理器件上实现这类线性神经元。一方面 ADALINE 开启了一个可学习智能的时代，使得二十世纪六十年代是人工智能发展的第一个黄金年代。但另一方面它也带来沉重的代价，它丢掉了最重要的非线性部分，使得整个学习无非就是一个线性模型。1969 年，Minsky 和 Papert 发表了一本名为《Perceptron》的书，该书中有两个关键点直接改变了人工智能的发展历史。第一是，对于基本的 XOR 问题，单层神经网络（没有隐藏层）没办法拟合好。所谓的 XOR 问题是一个分类问题，有四个样本，对角的两个样本是一类。对这个问题，单用一条线是无法准备分割的。而前面提到的 ADALINE 由于是一个线性模型，只能做到线性分割，因此，第一个点可以说成当前用的神经网络能力很有限，连简单的问题都很难做好，大家不要想太多。第二个点是虽然当前的简单网络不行，但只要加上隐藏层，变成多层感知机（Multi-layer perceptron, MLP）或者叫深度网络就可以做到功能强大，但遗憾的是当前地球上没有可行的办法训练这么复杂的工具。因此，这两个点的一个核心就是当前神经网络或者能力不行或者不能训练。

Minsky 和 Papert 开启了第一次人工智能的寒冬。资本和官方基金都把人工智能相关的经费砍掉了。许多研究人员为了使得相关的研究发表，也将神经网络换了名字。尽管是寒冬，但仍然有不少研究人员坚持研究神经网络相关的工作。Minsky 和 Papert 虽然指出了当前的不足，但也指了一条明路，寻找一种可行的办法训练多层神经网络。今天我们用来训练神经网络的向后传播方法早在二十世纪六十年代就由多个研究人员发展起来。本质上，这个方法就是简单的梯度下降，所谓的向后传播也只不过是几百年前就发展出来的链式求导法则。但在 1970 年 Seppo Linnainmaa 就将向后传播方法在计算机中实现。由于算力有限，那只是一种尝试并没有带来本质的影响。在美国，Paul Werbos 在他的博士论文中提出用向后传播来学习多层神经网络，他还是 MIT 访问了 Minsky。他回忆说：“但当时的悲观情绪已经走到尽头。在 1970 年代初期，我确实访问了麻省理工学院的明斯基。我提议我们联合发表一篇论文，证明 MLP 实际上可以克服早期的问题……但明斯基并不感兴趣。事实上，当时麻省理工学院、哈佛大学或我能找到的任何地方都没有人感兴趣。”直到八十年代，Werbos 才发表了他关于应用向后传播到神经网络的工作。对于很多人来说，他们的命运，固然和他们的努力密不可分，但也受到了历史进程的影响，当时的计算能力限制了他们的发挥。

在八十年代，人工智能慢慢又重回到人们的视线。一个很重要的工作是物理学家 John Hopfield 关于联想记忆的工作。大致的想法是假如你记住某张图片，当这张图片的部分呈现给你的时候，你会回想起这个图片的全部。若某个函数有很多极值点，每个极值点对应了一张记

住的图片。当呈现部分图片时，可以理解为在相应的极值点附近，通过简单的优化，可以很容易找到极值点的位置，就能重现整张图片。Hofield 用统计物理的模型实现了这个想法。这不仅是工程上有意思，在科学上对于解释联想记忆也是非常有意思，因此吸引了大量的关注。

1986 年，David E. Rumelhart，Geoffrey E. Hinton 和 Ronald J. Williams 在 Nature 发表了一篇用向后传播方法训练神经网络的文章，这篇文章成为深度学习发展历史上的一个里程碑，它的贡献可能不在于它有什么非常原创的想法，而是某种“胆大心细”。首先，这篇文章明确指出向后传播的方法已经有人在之前发现了，包括 David Parker 和 Yann LeCun，可见它并不是提出这个算法的第一人。其次，这篇文章描述向后传播方法的方式和我们今天描述的方式差异不大，是一种简洁又清晰的描述方式，这是文章的亮点之一，它虽然不是第一个发现，但它可能是第一个把这个方法讲得清楚，大部分读者容易读懂。最后，作者们比前人在实验上走得更远，他们在相对复杂的问题上验证了向后传播算法可以用来训练好多层神经网络。清晰有用的做法把神经网络再一次带到大众眼前。

此时，大家也意识到神经网络虽然发展了很长一段时间，但其相关的理论极少。从 1989 年起，Cybenko，Hornik 和 Barron 等独立证明了包含一个隐藏层的神经网络具有万有逼近能力，也就是说，只要足够宽，它能以很小的误差几乎拟合所有的函数。这当然很令人兴奋，但这些数学证明都有一个致命的弱点，也就是它们都假设了无穷的数据和无穷的算力等，理论构造的解往往也不是实际中能找到的解。另外，1986 年的文章中相对复杂的任务和实际问题比，仍然是非常简单的任务。一个关键的问题再次摆在研究人员的面前：向后传播算法训练的神经网络能完成有实际价值的任务吗？

1989 年，Yann LeCun 用卷积神经网络展示了深度学习可以高效准确地识别手写数据，并在支票和邮政编码的读取上产生了实际应用。自此，深度学习像六十年代一样，掀起了研究热潮。研究包括无监督学习，生成模型，强化学习，AI 玩游戏、自动驾驶等。狂欢之下，人们对人工智能的期望再一次过分的高起来，使得期望与现实产生越来越大的差距。在国际象棋上，AI 并没有很智能，自动驾驶的故事只能存在于实验室中等。其原因也很快被认识到，简单向后传播算法应用到深度网络中，会梯度消失而导致浅层网络反而比深层网络好。处理带有时间关联的循环神经网络极难训练，尽管 Schmidhuber 和 Hochreiter 在 1997 发展了今天应用极为广泛的长短程记忆网络（LSTM），但仍然未能挽回人们的信心。计算机虽然有一定的发展，但仍然非常慢，数据量也非常少，没有理论支撑的算法也难以吸引很多科学家的注意。这些因素累积在一起，使得人们再一次对人工智能丢失了兴趣。

这一次寒冬真的很冷。神经网络相关的研究人员的论文常常因为研究方向直接被拒。但有一小掇人坚持下来了，这包括 2018 年图灵奖的三位得主 LeCun，Hinton 和 Bengio。他们在极其困难的时候找到了一个金主，那就是加拿大高等研究院 (CIFAR) 愿意资助他们的自由研究，而不带有应用落地的要求。

时间来到 2006 年，Hinton 的一篇文章，通过结合无监督学习和监督学习的训练方法，再

次证明了很深的网络可以被训练好。这种训练方法今天已经几乎没有人使用了，但它作为敲门砖，敲开了深度学习的第三次发展浪潮。这个网络也被叫做深度信念网络。Hinton 曾这样回忆：“从历史上看，这对于克服深层神经网络不好且无法训练的信念是非常重要的。这是一个非常强烈的信念。不久前，我的一个朋友向 ICML[国际机器学习会议] 发送了一篇论文，仲裁人说 ICML 不应该接受它，因为它是关于神经网络的，不适合 ICML。事实上，如果你看一下去年的 ICML，你会发现没有一篇论文的题目是“neural”，所以 ICML 不应该接受关于神经网络的论文。那是几年前的事了。实际上，有一份 IEEE 期刊的官方政策是 [不接受你的论文]。所以，这是一种强烈的信念。”

大约在 2010 年，Hinton 把他的三位学生派到谷歌，微软和 IBM 实习，结合 GPU，他们让这三家公司意识到了深度学习巨大潜力，并在后面的时光里大力发展深度学习。2006 年开始，李飞飞开始建立一个超大的监督图片数据集 Imagenet，大约有 320 万张带有标签的图片，分成 5247 个类别。2012 年，在大量数据、强算力的计算机以及大量地深度学习经验下，Hinton 组参加 Imagenet 的识别竞赛，将误差 26.2% 直接降到 15.3%，吊打其它所有算法，第二年的竞赛中，排名靠前的算法均是依赖神经网络。这一次，在学术界，奠定了深度学习作为人工智能代表的基础。让深度学习完全走进普通人眼里的则是 2016 年 AlphaGo 战胜围棋高手李世石。阶段性的高潮是 LeCun，Hinton 和 Bengio 获得 2018 年的图灵奖。再一次，深度学习掀起了社会的狂欢，而这种狂欢之下难免开始有人思考是不是又有另一场危机即将到来。

自动驾驶带来的安全隐患，生成图片产生的奇怪图片，神经网络易于被攻击等让人不禁怀疑越深的学习是不是带来越深的麻烦。在重温深度学习发展的历史和放眼当前深度学习的发展，我有几点看法。首先，和历史上深度学习引发的高潮不一样的是，今天的深度学习已经走出实验，在实际工业和生活中产生了应用，它不会就这样简单的消失，更可能是成为人们生活中必不可能但又不会特别关注的一个领域。正如计算机的发展一样，没有人在今天会去谈论计算机有没有用，因为每个行业都离不开他。当计算机领域发展出现问题时，人们关注的是如何解决问题，而不是先有悲观情绪。第二，当前深度学习的网络设计离不开模仿、调参和试错，颇有暴力美学的味道。究其原因是我们对其的理解太少，基础的研究太缺乏。现在深度学习带来的巨大价值使得我们有资本去支持看似无用或者短期难以实用的基础研究，这将为深度学习的长远发展提供重要保障。

深度学习从 2012 年起就开始吸引了学术界很多注意力，我的导师在 2014 年左右还在组里组织过一次讨论关于 Hinton2006 的文章，但并没有深入。数学界对深度学习的重视也是相对滞后，为此，鄂维南院士 2020 年发表了《The Dawning of a New Era in Applied Mathematics》，鄂老师让我帮助组织人翻译了这篇文章《应用数学新时代的曙光》，呼吁数学家，特别是应用数学应该重视机器学习的发展。我也曾和一些颇有声望的数学家聊过，问他们为什么不研究深度学习，有一个数学家曾说，这只是线性和非线性变换的不断耦合，我总结他的意思就是这个东西 too simple, sometimes navie。事实上，深度学习相关的理论是比较复杂的。很多研究人

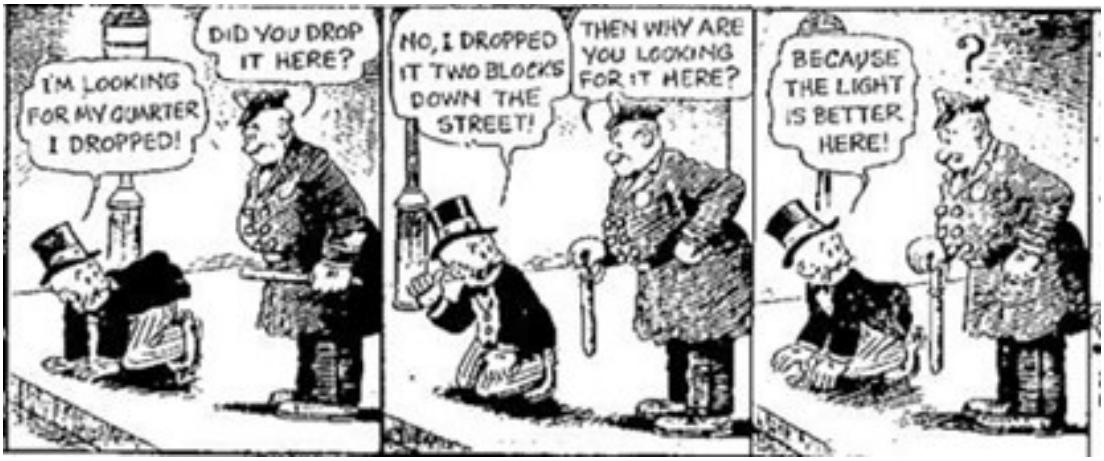


图 1.14: A simple cartoon.

员不得不做很多简化，也带来了很多批评。我们来看一个对话：

A: I am looking for my quarter I dropped.

B: Did you drop it here?

A: No, I dropped it two blocks down the street.

B: Then why are you looking for it here?

A: Because the light is better here.

一些人用这个小故事来批评很多研究人员做是可做的问题，而不是真正需要被研究的问题。

深度学习的研究是一个非常交叉的学科，研究人员经常不知道要用什么样的工具才适合研究。应用数学家 David Donoho 曾经把深度学习比作一个魔镜，每一个理论学家看到深度学习的时候，看的是他们想看的（图1.15）。比如物理学家可能会说深度学习就是某个物理系统，数学家可能会说它就是一组微分方程，生物学家可能会它只是大脑的一个仿生品，优化学家会说这一切都是优化问题。一方面，大家说的都没有错，另一方面，仅从一个角度来看可能不足够。我们应该把深度学习看作是一个交叉多学科的新学科，立足深度学习本身，而不是将它化作某个领域的子问题，然后继续研究其领域，等研究完，再套上深度学习的壳。

研究方法的思考 正如前面提到的，不同领域的人看深度学习或者机器学习的时候，颇有盲人摸象的感觉。每个人看的角度不一样，看到的东西也不一样。但这并不是一件坏事！回想物理发展，我们可能正处于安培和法拉第的时代，不同的人提出不同的定律，看似有关联，却尚未发展一套完整的理论。我们需要发展各种定律，然后类似麦克斯韦一般把这些方程统一起来，不仅仅是统一了已经有的认识，还看到了诸如电磁波等新的现象。

物理发展还有一个重要的过程可以用来借鉴。深度学习也类似于第谷和开普勒的时代，我们累积了大量的数据，慢慢发现一些类似开普勒三定律的定律，并用它们来解释很多现象。待数据和规律越来越丰富后，我们也许会来到另一种牛顿时代，用简单的几条法则构建一套可以指导深度学习设计和训练以及理解的框架。

回顾物理学的发展，有一个很重要的研究思路是从简单到复杂，比如前面提到的卡通故事里，找硬币的人可以这么回答:“Because I need to get familiar with the road structure first!”。在一个复杂的系统中，不断的把不重要的因素扔掉，研究重要因素之间的关系，比如伽利略的理想实验。在深度学习中，尽管我们从大量的实践中累积了大量数据，但它们不一定能给我们很多启发。比如人类历史上已经累积了很多生活经验相关的数据，但没有整理，没有设计相应的实验，很难从中得到关键的定律。因此在深度学习的研究中，如何设计实验，使其简单到可以分析，并且复杂到可以产生有意义的现象，是非常重要的！

Deep Learning as a Magic Mirror

Figure : Every theorist who looks at it see what they wish

图 1.15: 图片来源于 David Donoho 的报告。

1.13 课后思考

机器学习有哪些类别？
机器学习与统计学习有什么关联和差异？
为什么机器学习越来越重要？
什么是回归问题？
什么是分类问题？

回归问题和分类问题有什么区别？
什么是损失函数？
交叉熵 H(p,q) 关于 p,q 是对称的吗？为什么？
什么是神经网络？
神经网络的层数怎么算？
一层网络有什么限制？
向后传播和链式法则有什么区别？
误差可以分成哪几类？
什么样的网络具有万有逼近能力？
没有偏置项两层 ReLU 的神经网络的输出长什么样？
非线性激活函数有什么好处？
有哪些常用的激活函数？
多项式激活函数合适吗？
为什么要研究两层神经网络？
什么叫维数灾难？
机器学习常用的概念一般用什么符号表示？

参考文献

Cybenko, G. (1989), 'Approximation by superpositions of a sigmoidal function', *Mathematics of control, signals and systems* **2**(4), 303–314.

Glorot, X. & Bengio, Y. (2010), Understanding the difficulty of training deep feedforward neural networks, *in* 'Proceedings of the thirteenth international conference on artificial intelligence and statistics', pp. 249–256.

He, K., Zhang, X., Ren, S. & Sun, J. (2015), Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 1026–1034.

He, K., Zhang, X., Ren, S. & Sun, J. (2016), Deep residual learning for image recognition, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 770–778.

LeCun, Y. A., Bottou, L., Orr, G. B. & Müller, K.-R. (2012), Efficient backprop, *in* 'Neural networks: Tricks of the trade', Springer, pp. 9–48.

Narkhede, M. V., Bartakke, P. P. & Sutaone, M. S. (2022), 'A review on weight initialization strategies for neural networks', *Artificial intelligence review* **55**(1), 291–322.

Ruder, S. (2016), 'An overview of gradient descent optimization algorithms', *arXiv preprint arXiv:1609.04747* .

Wu, L., Ma, C. et al. (2018), 'How sgd selects the global minima in over-parameterized learning: A dynamical stability perspective', *Advances in Neural Information Processing Systems* **31**.