第一章　引言及浅层神经网络

1.1　引言

1.1.1　参考书目及课程资源

参考书目如下：

- 《动手学深度学习》
- 《深度学习》(花书)

视频资源：

- 《动手学深度学习》
- 深度学习—— 李宏毅

代码资源：

- 《动手学深度学习》

其他优质资源：

- 深度学习论文精读 (李沐) 持续更新中

1.1.2　深度学习与机器学习的区别与联系

机器学习是什么？　并不是所有的解决方案都可以通过直接编写程序来解决，比如，编写一个识别猫的图片的程序。机器学习尝试利用**数据**，并结合**模型假设**，再基于一定的**优化方案**获得相对最优的模型。这一过程中，通常需要借用人类的认知知识，比如，标注好的猫的图片，因此，“学习”的

过程就有了，所以称为机器学习 (machine learning)。机器学习是现阶段实现人工智能的有效方式。

从程序设计的角度而言，这一区别于直接编写程序，而是通过数据学习的方式来构建解决方案，也因此被称为“**用数据编程**”。

从数学的角度而言，机器学习与统计学有着密切联系，即，基于数据的分析技术，比如，最小二乘线性回归。

深度学习中的模型假设。 通常人们所说的深度学习是基于深度神经网络模型的学习框架。这种形式灵活多变的模型构建方法，使得模型假设也变得特别一般化 (general)，随着网络模型宽度与深度的增加，模型的表达能力 (capacity) 可以很大很大。从网络模型这样的模型假设来看，机器学习中的一些经典模型都可以看做是浅层神经网络，比如，线性回归，逻辑斯蒂回归，支持向量机。。。

表示学习 (representation learning)。 深度学习成功引起人们的重视，是源自于2012年计算机视觉领域中的 ImageNet 竞赛，AlexNet 模型取得了远超传统机器学习模型的性能优势。视觉领域中，特征表示学习起着至关重要的作用，回顾深度模型之前的视觉领域，大量的科研关注点是设计更加有效的特征抽取方法，经典的工作包括适用于行人识别的 HOG 特征，适用于纹理细节描述的 LBP、SIFT、Gabor 等特征，以及为解决多尺度目标识别的特征金字塔构建方案。

深度网络模型中从原始图像像素输入层层到分类决策层的层层堆叠过程中，可以看成是在逐级构建不同抽象语义级别的特征表示，比如，底层在抽取图像中的边、角信息，中层在抽取部件、边框等信息，高层在抽取对象级上的语义信息。在终端任务驱动下，自动学习出的特征表示如下图 1.1 所示。正是因为这种分层逐级抽象的特征表示方式，也被称为“分布式表示学习(distributed representation learning)”。

1.1.3 深度学习能够兴起的时代背景

优化技术。 从浅层神经网络到深度神经网络的想法并不是多么石破天惊，早有人尝试过，但是难以学习到有效的模型，主要原因在于基于梯度下降算法的优化过程中存在严重的梯度消失/梯度爆炸问题，以及权重矩阵退化问题，即，多数权重趋于 0。聚焦于权重初始化，数据预处理及归一化，以

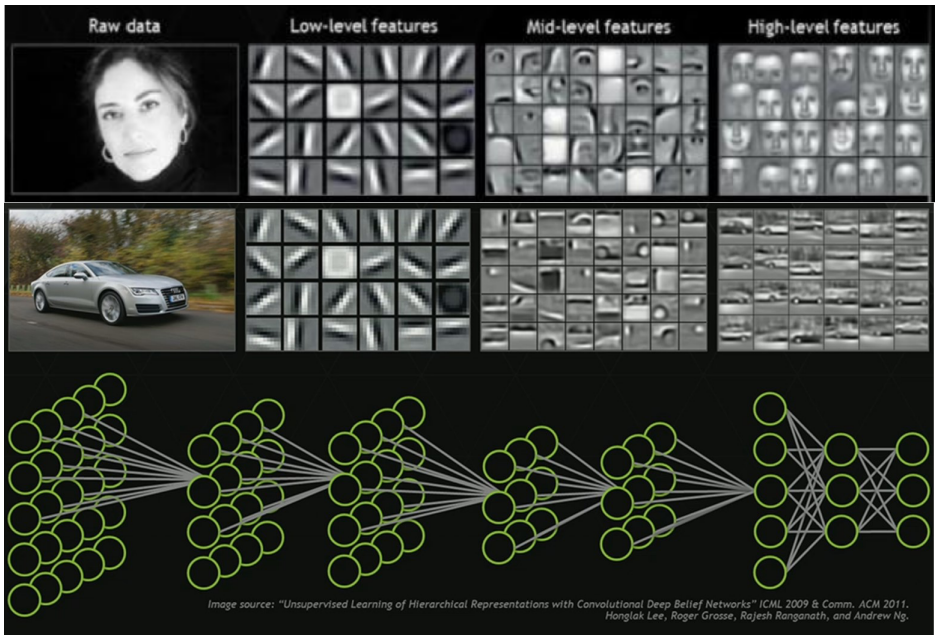

图 1.1: 表示学习学到了什么？

及 dropout 以及动量梯度优化技术，训练深层神经网络成为可能。

大数据。 从 ORL 人脸数据集，到 CIFAR，到 ImageNet，大规模精细标注的数据为模型及算法的研究提供了学习基础。

高算力。 NVIDIA 推出了用于 GPU 并行计算的 CUDA 工具包为深度神经网络模型的并行训练提供了便利。

1.2 浅层神经网络模型

到目前为止，已经了解了最基本的线性模型的建模方法，比如，线性回归模型，即，$f(x) = w^T x + b$，或者对线性组合值再施加一个非线性映射，即，$f(x) = \phi(w^T x + b)$，这种**计算单元**在神经网络模型中被称为一个“**M-P 神经元**”（*M-P 是两名提出者的名字首字母缩写，心理学家 W.S.McCulloch 和数理逻辑学家 W.Pitts*），示意图如图 1.2 所示。

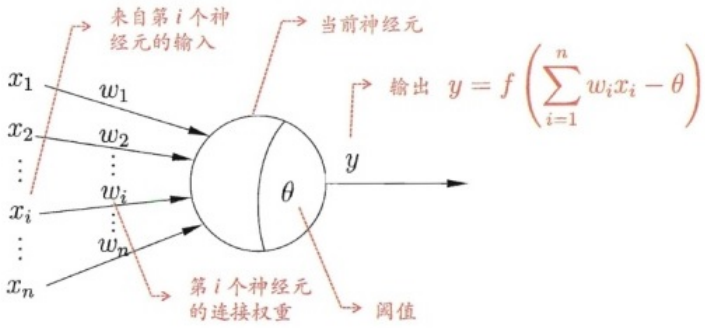

图 1.2: M-P 神经元示意图。

现在考虑一种拓展，即，对原始的输入特征 x 实行多级线性变换 $(W^T x)$，但是一定要融入非线性映射 $(\phi(W^T x))$，否则就会退化为单层线性映射 (比如，$W_1^T W_2^T W_3^T x = W^T x$，矩阵相乘仍然为矩阵)。由此，可以建模为

$$f(x) = \phi(W_{l_k}^T(\cdots, \phi(W_{l_2}^T \phi(W_{l_1}^T x + b_{l_1}) + b_{l_2})) + b_{l_k}) \tag{1.1}$$

其中，W 在较低的层上通常为矩阵，每一列对应一个神经元连接权重，最后一层为输出层，如果只有一个输出，则 W 为向量，否则也是一个矩阵 (比如多类分类问题)。由此，就得到了多层神经网络的形式化描述，对应的图形展示如图 1.3 所示。

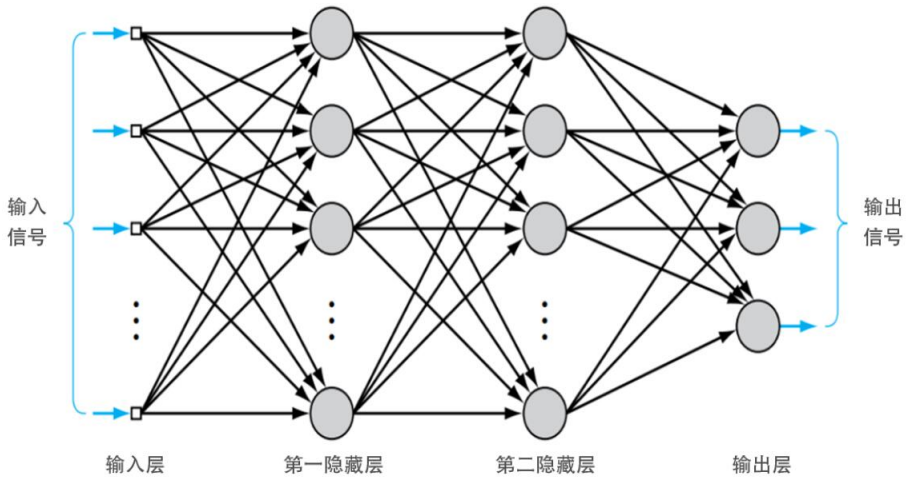

图 1.3: 多层神经网络示意图。

对神经网络模型架构的理解 神经网络模型是通过多层级嵌套 M-P 神经元的方法来构建复杂模型的一种建模方法。先不考虑神经网络的宽度 (*单层上神经元的数目*) 和深度 (*层数*) 的设置问题，先来直观上理解神经网络在干什么事。神经网络必然包含输出层和输出层这两层，输入层神经元数目由样本的特征维度来决定，输出层神经元数目由任务来确定，中间层 (*也称为隐含层*) 可以看做是**特征空间的转换**，可以进行**特征的升维或降维**变换。确定了神经网络的模型架构之后，就需要解决模型参数的学习问题了。好的模型参数能够达到预期的输出，这种输出上的预期被满足的程度是通过一个所谓的**损失函数**来实现的，其负责衡量模型输出与预期输出之间的差异。从而，模型参数调优可以通过追求在训练样本上的损失函数最小化来实现，从而实现包括**特征变换**和**任务实现**一体化的学习框架，被称为端到端的学习 (*end-to-end learning*，*针对分类任务可以理解为从特征 x 到任务 $p(y|x)$*)。

1.2.1 常见的损失 (误差) 函数

损失 (误差) 函数是机器学习中的一个常见概念，绝对不局限于神经网络模型，这里列举几个常见损失函数，便于构建起神经网络模型学习的全流程，同时也深入了解一下神经网络可以完成哪些任务，比如，回归、分类，等等。

损失函数设计的概率框架 损失函数设计的一个基本思想可以为最大似然，对于分类模型而言，即，最大化 (最小化) 训练集上的 (负) 类后验概率，形式化描述如下：

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{N} -\log p_{model}(y_i|x_i;\theta) \tag{1.2}$$

最小均方误差 在公式 1.2 所述的损失函数概率统一框架下，对于回归问题，可以假设输出变量 y 的后验概率分布为高斯分布，即

$$p(y|x) = \mathcal{N}(\hat{y}, I)$$

由此，可以得到训练集上的所有样本的预期输出 y 与模型实际输出 $\hat{y}$ 之间的差异程度的累计统计指标

$$J(\theta) = \sum_{i=1}^{N} ||\hat{y}_i - y_i||^2$$

这就是最小均方误差 (Mean Squared Error) 损失函数，最终的优化问题为：

$$\theta^* = \arg\min_{\theta} J(\theta) = \arg\min_{\theta} \sum_{i=1}^{N} ||\hat{y}_i - y_i||^2$$

交叉熵损失 在公式 1.2 所述的损失函数概率统一框架下，对于两类分类问题可以假设类后验概率分布为伯努利分布，即，

$$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

由此，可以得到训练集上的所有样本的预期输出 y 与模型实际输出 $\hat{y}$ 之间的差异程度的累计统计指标为

$$J(\theta) = \sum_{i=1}^{N} -\Big(y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)\Big)$$

这就是交叉熵 (cross entropy) 损失函数，最终的优化问题为：

$$\theta^* = \arg\min_{\theta} J(\theta) = \arg\min_{\theta} \sum_{i=1}^{N} -\Big(y_i \ln \hat{y}_i + (1-y_i)\ln(1-\hat{y}_i)\Big)$$

推广到多类分类情况下，则假设类后验分布为多项式分布，在公式 1.2 所述的损失函数概率统一框架下，可以得出最终的优化问题为：

$$\theta^* = \arg\min_{\theta} \sum_{i=1}^{N} \sum_{k=1}^{K} -y_{ik} \ln \hat{y}_{ik}$$

1.2.2 后向传播算法

多层神经网络模型参数如何优化呢？不得不提一下梯度下降算法，一旦知道了参数的梯度，就可以沿着负梯度方向走一小步，来修正当前参数的值。所以问题的关键转换为如何高效地计算梯度？多层神经网络模型参数的梯度计算存在什么问题呢？通过前面的学习，了解了神经网络模型的参数分布在不同的层之间，层与层之间可以看做如公式 1.1 所示的嵌套复合关系。神经网络模型参数的梯度的求解对象是损失函数，靠近输出层的参数的梯度可以很便捷的计算，但是远离输出层 (靠近输入层) 的参数的梯度则需要通过复合函数的层层递推求导来获得。这有点类似于动态规划中的最优问题求解，于是希望通过一次运算来实现不同层上的模型参数的梯度计算。相应的一套方法，称为后向传播算法，因此，**后向传播是一种高**

效的梯度计算方法，特别面向多层神经网络模型参数的梯度计算，客观地讲，与具体的梯度算法无关。

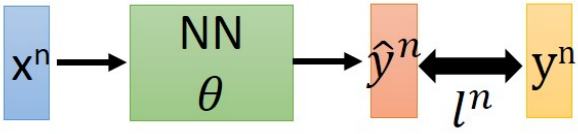

图 1.4: 神经网络模型学习的流程示意图。

神经网络的训练过程的全流程如图 1.4 所示，训练集上的样本累计损失为：

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} l^n(\theta)$$

其中，模型参数 θ 指代神经元之间的连接权重 w 和偏置量 b，从而对模型参数的求导为：

$$\frac{\partial \mathcal{L}(\theta)}{\partial w} = \sum_{i=1}^{N} \frac{\partial l^n(\theta)}{\partial w}$$

即，样本损失函数针对模型参数的梯度计算可以在样本上独立的计算，最后再累计起来。

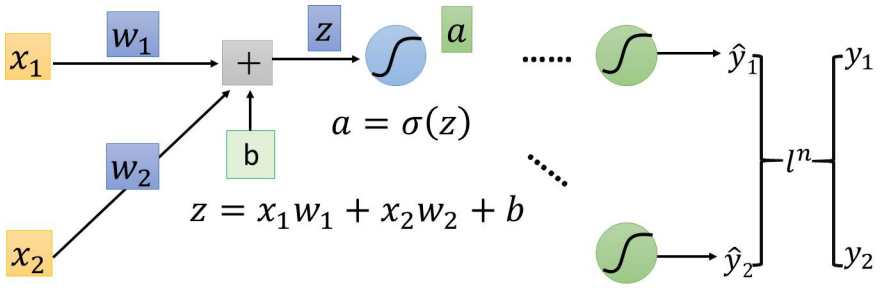

图 1.5: 后向传播算法中的数据流示意图。这里以单个神经元上的参数的梯度计算为例，展示了其梯度计算的两步走方案：前向数据流和后向数据流。

以下，分析单个样本上的梯度计算。先分析神经网络中的一个神经元的参数的梯度，如图 1.5 所示。该神经元的**预激活值**记作 z，即，$z = w^T x + b = w_1 x_1 + w_2 x_2 + b$，**激活值**记作 a，即，$a = \sigma(z)$，其中，$\sigma(\cdot)$ 为激活函数。从损失函数开始计算针对当前神经元的参数的梯度为：

$$\frac{\partial l}{\partial w} = \frac{\partial l}{\partial z}\frac{\partial z}{\partial w} \tag{1.3}$$

该公式推导用到了求导中的链式法则，这里主要介绍两种类型的复合函数求导的链式法则，如图 1.6 所示。

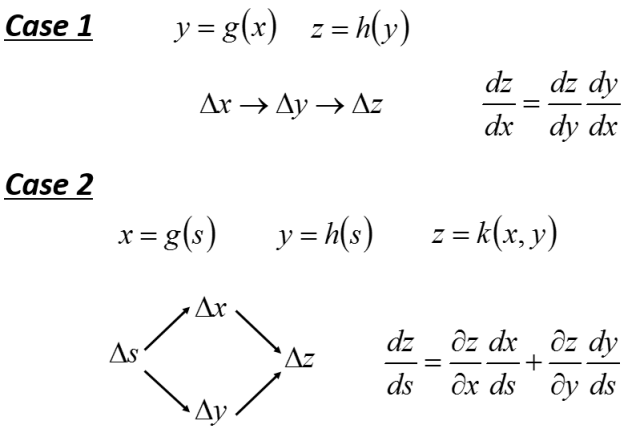

图 1.6: 两种类型的复合函数求导的链式法则。

继续分析单个神经元的参数的梯度的计算公式 1.3，其中第二项的计算结果为前一层神经元的激活值 a，这里恰好是输入层，则为输入值 x，即，$\frac{\partial z}{\partial w_1} = x_1$，$\frac{\partial z}{\partial w_2} = x_2$，该项的值可以看做是从前一层传递过来的信息，即从输入层向输出层进行前向传播而来的信息。

第一项的值 $\frac{\partial l}{\partial z}$ 的计算为损失函数针对当前神经元的预激活值求导，如图 1.5 中所示的函数复合关系，第一项的计算通过链式法则转换为：

$$\frac{\partial l}{\partial z} = \frac{\partial l}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial l}{\partial a}\sigma'(z)$$

继续向后推导，如图 1.7 给出后续层的符号表示。 则后向梯度的计算可以进一步推导为：

$$\begin{aligned}
\frac{\partial l}{\partial z} &= \frac{\partial l}{\partial a}\frac{\partial a}{\partial z} \\
&= \frac{\partial l}{\partial a}\sigma'(z) \\
&= \sigma'(z)\left(\frac{\partial l}{\partial z'}\frac{\partial z'}{\partial a} + \frac{\partial l}{\partial z''}\frac{\partial z''}{\partial a}\right) \\
&= \sigma'(z)\left(\frac{\partial l}{\partial z'}w_3 + \frac{\partial l}{\partial z''}w_4\right)
\end{aligned} \tag{1.4}$$

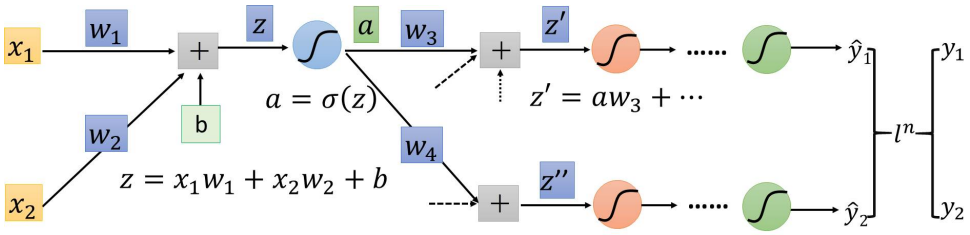

图 1.7: 后向传播算法中的数据流示意图。梯度计算公式中的后向数据流的进一步计算。

从公式 1.4 可以看出，求导公式中出现了递归关系，即，计算 $\frac{\partial l}{\partial z}$ 用到了后一层中的 $\frac{\partial l}{\partial z'}$ 和 $\frac{\partial l}{\partial z''}$。由此，一直递归到输出层，即可。通常把**后向梯度流**记作 δ_l，即，$\delta_l = \frac{\partial l}{\partial z_l}$。仔细观察公式 1.4，后向梯度流的递归计算过程，可以看做是依据当前神经网络反向运算的过程。

这里反向计算的细节为：反向计算的输入数据流为 δ_l，连接权重仍然为对应连接边的值，由此进行线性组合得到反向预激活值 $z_b = \frac{\partial l}{\partial z'}w_3 + \frac{\partial l}{\partial z''}w_4$，然后计算神经元的非线性映射函数的导函数 $\sigma'(z)$ 作为一个乘积项施加到反向预激活值 z_b 上 (注意：导函数作用于前向过程中计算的预激活值 z)。这就是梯度流的后向传播过程，直观地展示如图 1.8 所示。

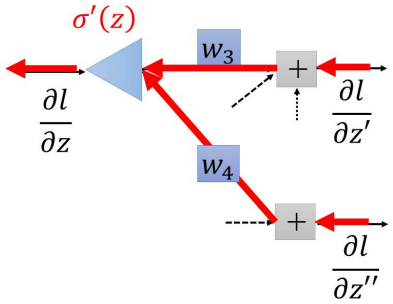

图 1.8: 后向传播算法中的梯度后向传播示意图。神经网络前后颠倒看待，权值不变，输入数据流为靠近输出层的梯度流反向流，激活函数的导函数代替原来的激活函数，扮演缩放器的作用。

由于是递归计算，因此，只需要具体分析最后一层上的梯度计算，即，计算损失函数对输出层上的预激活值 z_o 的导数。不失一般性地假设图 1.7

中的预激活值 z'、z'' 为最后一层神经元的预激活值，如图 1.9 所示。 则，

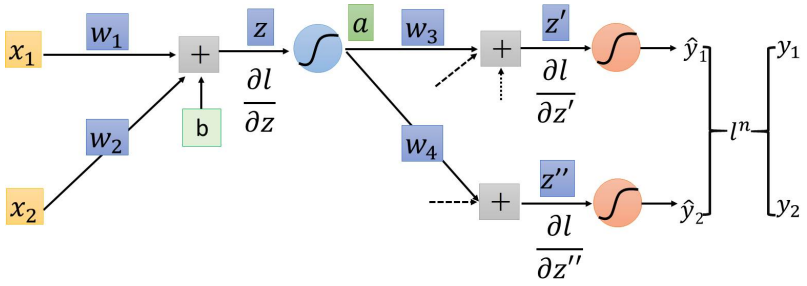

图 1.9: 后向传播算法中的梯度后向传播示意图。输出层上的后向梯度作为后向传播的梯度源头。

可以得出后向梯度流为：

$$\frac{\partial l}{z'} = \frac{\partial l}{\hat{y}_1}\frac{\partial \hat{y}_1}{z'}, \quad \frac{\partial l}{z''} = \frac{\partial l}{\hat{y}_2}\frac{\partial \hat{y}_2}{z''}$$

其中，$\frac{\partial l}{\hat{y}_1}$ 为损失函数对模型实际输出的偏导数，$\frac{\partial \hat{y}_1}{z'}$ 为输出层神经元的非线性激活函数的导函数。这些都是可以计算的，由此，为梯度后向传播提供了最原始的梯度流。

总结： 公式 1.3 展示了每一层上的参数的梯度计算为两部分的乘积，前向数据流与后向梯度流的乘积。前向数据流的计算比较简单，只需要整个网络对于给定的输入数据前向计算一次即可，存储下来每一层上的预激活值 z_l，以备独立计算每一层梯度时使用。后向梯度流的计算，则需要从损失函数层开始计算，然后像输入层反向传播，传播过程中，网络的连接权重与前向一致，非线性函数的导函数作用到前向预激活值，并作为缩放尺度作用到反向传播中的线性组合值上，如此，反向计算一次即可，存储下来每一层上的反向梯度流 $\delta_l = \frac{\partial l}{\partial z_l}$，以备独立计算每一层梯度时使用。最终，依据公式 1.3计算每一层网络上的参数的梯度计算。BP 算法的整体示意图如图 1.10 所示。 另外，可以看出，BP 算法是以存储空间来换取时间，即，存储下来每一步的数据 (*包括前向数据流传播和后向梯度流传播*) 来减少递归计算中的重复计算。

理解 BP 算法的意义： 在现有的深度学习平台上，BP 算法都被集成了，不需要开发者再去独立的实现了。理解 BP 算法的具体过程，有助于理解

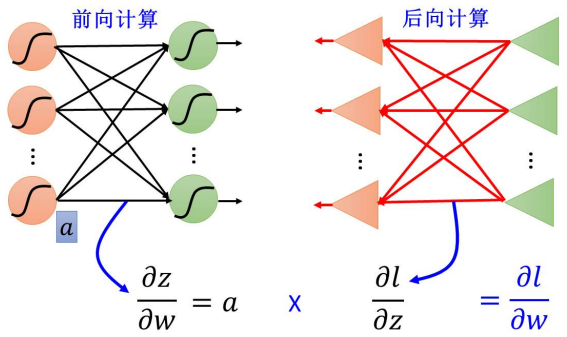

图 1.10: 后向传播算法中梯度计算的两个关键过程的示意图。

模型实际的训练过程，比如，批处理样本上的梯度计算。以及理解梯度消失问题产生的原因：在于非线性激活函数的导函数扮演梯度缩放器的角色。当激活函数为 sigmoid 函数时，如图 1.11 所示，如果预激活值较大或较小，则导函数的值趋于 0，由此导致梯度消失。

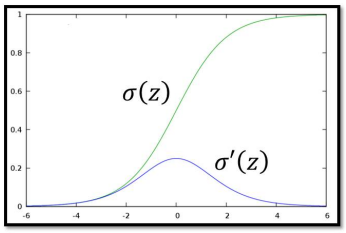

图 1.11: sigmoid 函数及其导函数的曲线图。