Chapter 1

神经网络的复习

1.1 神经网络的学习

1.1.1 损失函数

我们来介绍一下 Softmax 函数和交叉熵误差。首先，Softmax 函数可由下式表示：

$$y_k = \frac{\exp s_k}{\sum_{i=1}^{n} \exp s_i} \tag{1.1}$$

此时，交叉熵误差可由下式表示：

$$L = -\sum_k t_k \log y_k \tag{1.2}$$

这里，t_k 是对应于第 k 个类别的监督标签。监督标签以 one-hot 向量的形式表示，比如 $\boldsymbol{t} = (0, 0, 1)$。

另外，在考虑了 mini-batch 处理的情况下，交叉熵误差可以由下式表示：

$$L = -\frac{1}{N} \sum_n \sum_k t_{nk} \log y_{nk} \tag{1.3}$$

这里假设数据有 N 笔，t_{nk} 表示第 n 笔数据的第 k 维元素的值，y_{nk} 表示神经网络的输出，t_{nk} 表示监督标签。通过这样的平均化，无论 mini-batch 的大小如何，都始终可以获得一致的指标。

1.1.2 梯度与导数

> 严格地说，这里使用的“梯度”一词与数学中的“梯度”是不同的。数学中的梯度仅限于关于向量的导数。而在深度学习领域，一般也会定义关于矩阵和张量的导数，称为“梯度”。

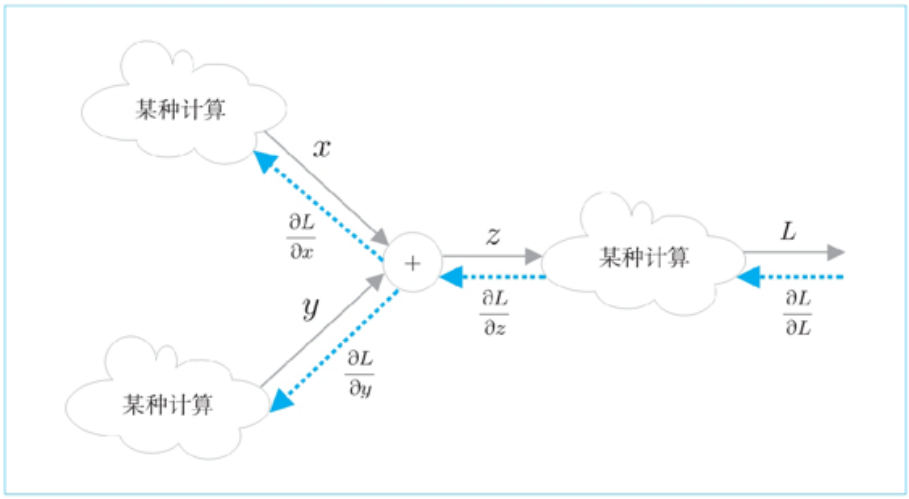

图 1.1: 计算图的反向传播

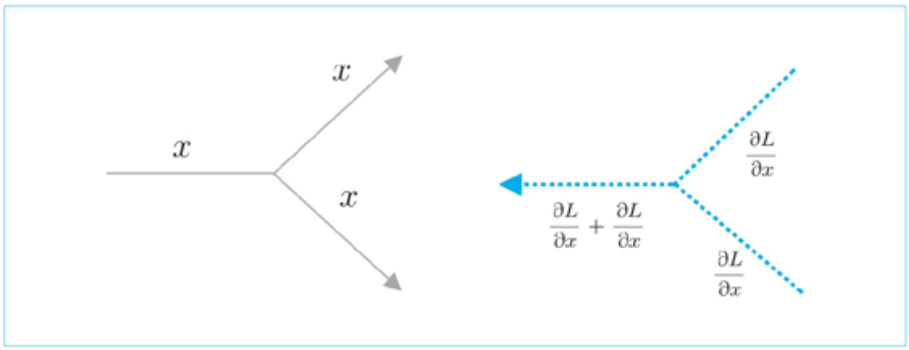

图 1.2: 分支节点的正向传播（左图）和反向传播（右图）

1.1.3 链式法则

链式法则的重要之处在于，无论我们要处理的函数有多复杂（无论复合了多少个函数），都可以根据它们各自的导数来求复合函数的导数。也就是说，只要能够计算各个函数的局部的导数，就能基于它们的积计算最终的整体的导数。

分支节点

严格来说，分支节点并没有节点，只有两根分开的线。此时，相同的值被复制并分叉。因此，分支节点也称为复制节点。如 Figure 1.2 所示，它的反向传播是上游传来的梯度之和。

Repeat 节点

分支节点有两个分支，但也可以扩展为 N 个分支（副本），这里称为 Repeat 节点。现在，我们尝试用计算图绘制一个 Repeat 节点（图 Figure 1.3）。

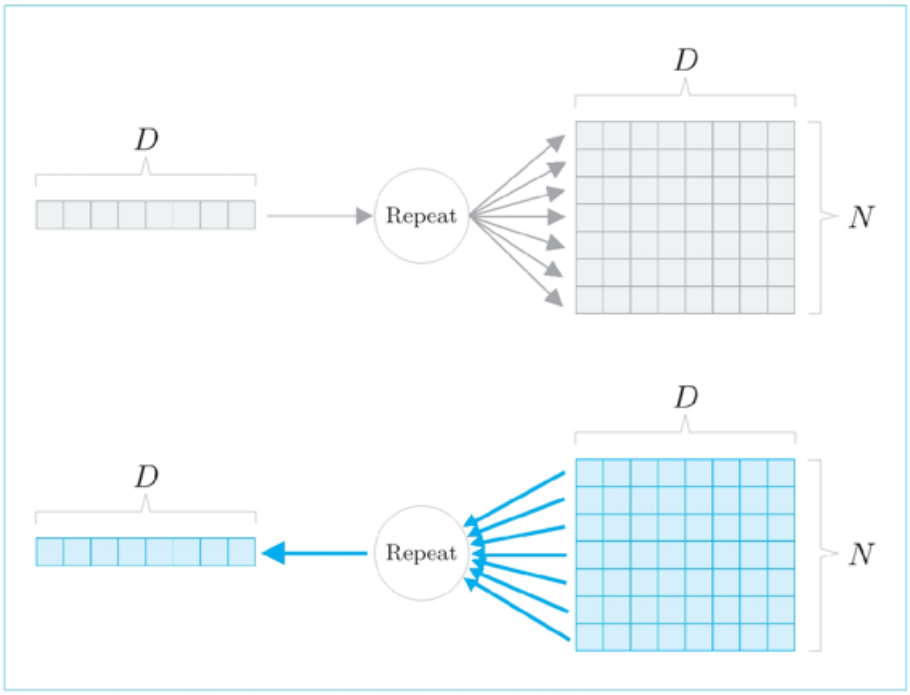

图 1.3: Repeat 节点的正向传播（上图）和反向传播（下图）

Sum 节点

Sum 节点是通用的加法节点。这里考虑对一个 $N \times D$ 的数组沿第 0 个轴求和。此时，Sum 节点的正向传播和反向传播如图 Figure 1.4 所示。有趣的是，Sum 节点和 Repeat 节点存在逆向关系。所谓逆向关系，是指 Sum 节点的正向传播相当于 Repeat 节点的反向传播，Sum 节点的反向传播相当于 Repeat 节点的正向传播。

MatMul 节点

考虑 $\boldsymbol{y} = \boldsymbol{x}\boldsymbol{W}$ 这个计算，这里，$\boldsymbol{x}$、 $\boldsymbol{W}$、$\boldsymbol{y}$ 的形状分别是 $1 \times D$、$D \times H$、$1 \times H$。此时，可以按如下方式求得关于 $\boldsymbol{x}$ 的第 i 个元素的导数 $\frac{\partial L}{\partial x_i}$。

$$\frac{\partial L}{\partial x_i} = \sum_j \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{1.4}$$

利用 $\frac{\partial y_j}{\partial x_i} = W_{ij}$，将其代入 Equation 1.4：

$$\frac{\partial L}{\partial x_i} = \sum_j \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial x_i} = \sum_j \frac{\partial L}{\partial y_j} W_{ij} \tag{1.5}$$

$\frac{\partial L}{\partial x_i}$ 由向量 $\frac{\partial L}{\partial \boldsymbol{y}}$ 和 $\boldsymbol{W}$ 的第 i 行向量的内积求得。从这个关系可以导出下式：

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial L}{\partial \boldsymbol{y}}\boldsymbol{W}^T \tag{1.6}$$

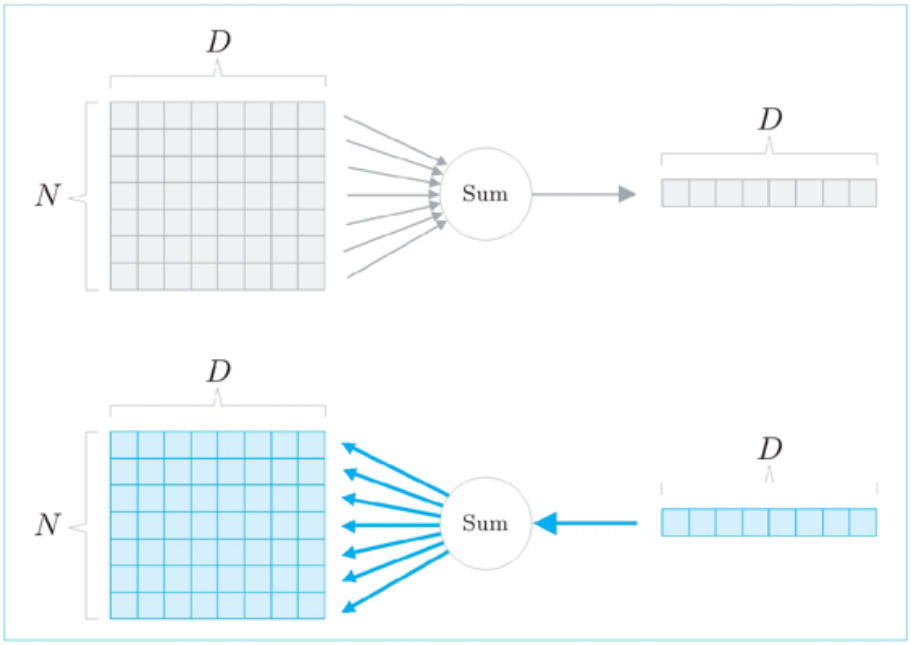

图 1.4: Sum 节点的正向传播（上图）和反向传播（下图）

和省略号一样，这里也可以进行基于 `grads[0] = dW` 的赋值。不同的是，在使用省略号的情况下会覆盖掉 NumPy 数组。这是浅复制（shallow copy）和深复制（deep copy）的差异。`grads[0] = dW` 的赋值相当于浅复制，`grads[0][...] = dW` 的覆盖相当于深复制。

浅复制中，`a` 的指向发生改变（指向的内存地址改变），而深复制中 `a` 的指向没有发生改变，只是指向的内存地址内的值改变（Figure 1.6）。

1.2 计算高速化

1.2.1 位精度

我们已经知道，如果只是神经网络的推理，则即使使用 16 位浮点数进行计算，精度也基本上不会下降。不过，虽然 NumPy 中准备有16 位浮点数，但是普通 CPU 或 GPU 中的运算是用 32 位执行的。因此，即便变换为 16 位浮点数，因为计算本身还是用 32 位浮点数执行的，所以处理速度方面并不能获得什么好处。但是，如果是要（在外部文件中）保存学习好的权重，则 16 位浮点数是有用的。具体地说，将权重数据用 16 位精度保存时，只需要 32 位时的一半容量。

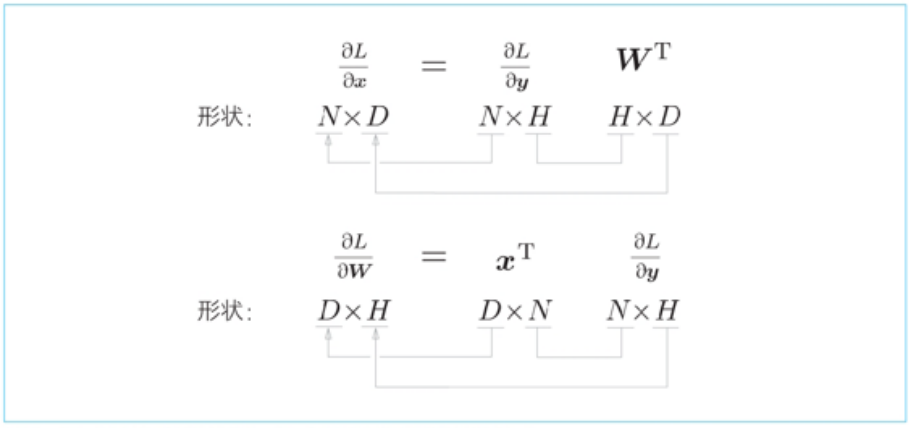

图 1.5: 通过确认矩阵形状，推导反向传播的数学式

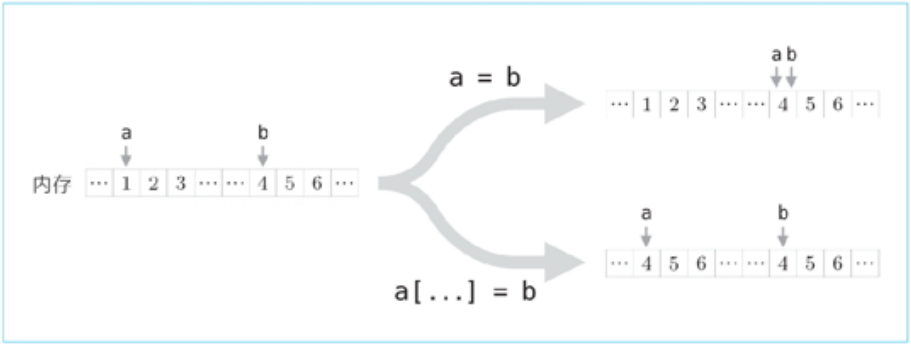

图 1.6: `a = b` 和 `a[...] = b` 的区别：使用省略号时数据被覆盖，变量指向的内存地址不变

Chapter 2

自然语言和单词的分布式表示

2.1 同义词词典

回顾自然语言处理的历史，人们已经尝试过很多次类似这样的人工定义单词含义的活动。但是，目前被广泛使用的并不是《新华字典》那样的常规词典，而是一种被称为同义词词典（thesaurus）的词典。在同义词词典中，具有相同含义的单词（同义词）或含义类似的单词（近义词）被归类到同一个组中。比如，使用，我们可以知道 car 的同义词有 automobile、motorcar 等

2.1.1 WordNet

在自然语言处理领域，最著名的同义词词典是 WordNet。使用 WordNet，可以获得单词的近义词，或者利用单词网络。使用单词网络，可以计算单词之间的相似度。

2.1.2 同义词词典的问题

- 难以顺应时代变化
- 人力成本高
- 无法表示单词的微妙差异

2.2 基于计数的方法

从介绍基于计数的方法开始，我们将使用语料库（corpus）。简而言之，语料库就是大量的文本数据。不过，语料库并不是胡乱收集数据，一般收集的都是用于自然语言处理研究和应用的文本数据。

自然语言处理领域中使用的语料库有时会给文本数据添加额外的信息。比如，可以给文本数据的各个单词标记词性。在这种情况下，为了方便计算机处理，语料库通常会被结构化（比

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
i	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

图 2.1: 用表格汇总各个单词的上下文中包含的单词的频数

如，采用树结构等数据形式）。这里，假定我们使用的语料库没有添加标签，而是作为一个大的文本文件，只包含简单的文本数据。

能不能将类似于颜色的向量表示方法运用到单词上呢？更准确地说，可否在单词领域构建紧凑合理的向量表示呢？接下来，我们将关注能准确把握单词含义的向量表示。在自然语言处理领域，这称为分布式表示。

单词的分布式表示将单词表示为固定长度的向量。这种向量的特征在于它是用密集向量表示的。密集向量的意思是，向量的各个元素（大多数）是由非0实数表示的。例如，三维分布式表示是 $[0.21, -0.45, 0.83]$。

2.2.1 分布式假设

“某个单词的含义由它周围的单词形成”，称为分布式假设（distributional hypothesis）。分布式假设所表达的理念非常简单。单词本身没有含义，单词含义由它所在的上下文（语境）形成。上下文是指某个居中单词的周围词汇。这里，我们将上下文的大小（即周围的单词有多少个）称为**窗口大小**（window size）。窗口大小为 1，上下文包含左右各 1 个单词；窗口大小为 2，上下文包含左右各 2 个单词，以此类推。

2.2.2 共现矩阵

2.2.3 向量间的相似度

测量向量间的相似度有很多方法，其中具有代表性的方法有向量内积或欧式距离等。虽然除此之外还有很多方法，但是在测量单词的向量表示的相似度方面，**余弦相似度**（cosine similarity）是

很常用的。设有 $\boldsymbol{x} = (x_1, x_2, x_3, \cdots, x_n)$ 和 $\boldsymbol{y} = (y_1, y_2, y_3, \cdots, y_n)$ 两个向量，它们之间的余弦相似度的定义如下式所示：

$$similarity(\boldsymbol{x}, \boldsymbol{y}) = \frac{x_1 y_1 + \cdots + x_n y_n}{\sqrt{x_1^2 + \cdots + x_n^2}\sqrt{y_1^2 + \cdots + y_n^2}} \tag{2.1}$$

余弦相似度直观地表示了“两个向量在多大程度上指向同一方向”。两个向量完全指向相同的方向时，余弦相似度为 1；完全指向相反的方向时，余弦相似度为 -1。