深度学习入门

基于Python的理论与实现

Stephen CUI

March 14, 2023

Chapter 1

误差反向传播法

通过数值微分计算了神经网络的权重参数的梯度（严格来说，是损失函数关于权重参数的梯度）。数值微分虽然简单，也容易实现，但缺点是计算上比较费时间。本章我们将学习一个能够高效计算权重参数的梯度的方法——误差反向传播法。

1.1 计算图

计算图将计算过程用图形表示出来。这里说的图形是数据结构图，通过多个节点和边表示（连接节点的直线称为“边”）。

计算图通过节点和箭头表示计算过程。节点用◯表示，◯中是计算的内容。将计算的中间结果写在箭头的上方，表示各个节点的计算结果从左向右传递。

虽然Figure 1.1中把“×2”“×1.1”等作为一个运算整体用〇括起来了，不过只用◯表示乘法运算“×”也是可行的。此时，如Figure 1.2所示，可以将“2”和“1.1”分别作为变量“苹果的个数”和“消费税”标在◯外面。

用计算图解题的情况下，需要按如下流程进行。

1. 构建计算图。
2. 在计算图上，从左向右进行计算。

这里的第2步“从左向右进行计算”是一种正方向上的传播，简称为**正向传播**（forward propagation）。正向传播是从计算图出发点到结束点的传播。既然有正向传播这个名称，当然也可以考虑反向（从图上看的话，就是从右向左）的传播。实际上，这种传播称为**反向传播**（backward propagation）。反向传播将在接下来的导数计算中发挥重要作用。

1.1.1 局部计算

计算图的特征是可以通过传递“局部计算”获得最终结果。“局部”这个词的意思是“与自己相关的某个小范围”。局部计算是指，无论全局发生了什么，都能只根据与自己相关的信息输出接下来的结果。

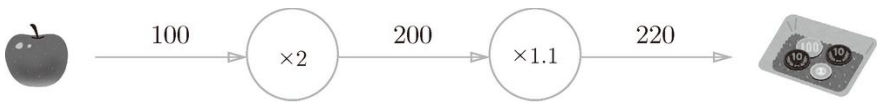

Figure 1.1: Based on the calculation graph to solve the answer

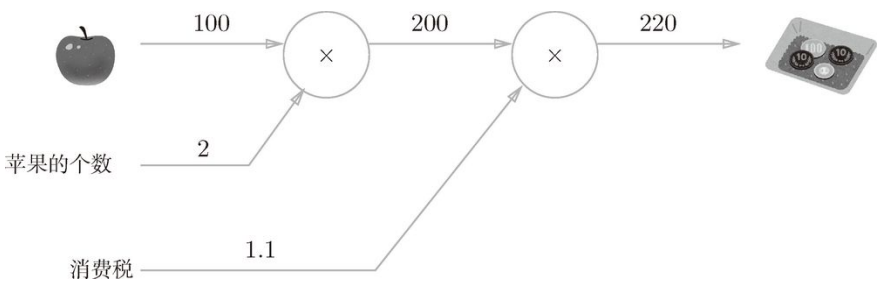

Figure 1.2: Based on the calculation graph to solve the answer2

1.2 链式法则

前面介绍的计算图的正向传播将计算结果正向（从左到右）传递，其计算过程是我们日常接触的计算过程，所以感觉上可能比较自然。而反向传播将局部导数向正方向的反方向（从右到左）传递，一开始可能会让人感到困惑。传递这个局部导数的原理，是基于链式法则（chain rule）的。

1.2.1 计算图的反向传播

1.2.2 什么是链式法则

介绍链式法则时，我们需要先从复合函数说起。复合函数是由多个函数构成的函数。比如，$z=(x+y)^2$是下面所示的两个式子构成的：

$$\begin{aligned} z &= t^2 \\ t &= x+y \end{aligned} \tag{1.1}$$

链式法则是关于复合函数的导数的性质，定义如下：

> 如果某个函数由复合函数表示，则该复合函数的导数可以用构成复合函数的各个函数的导数的乘积表示。

以Equation 1.1为例，$\frac{\partial z}{\partial x}$可以用$\frac{\partial z}{\partial t}$和$\frac{\partial t}{\partial x}$的乘积表示，可以写成下式：

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} \tag{1.2}$$

对于$z=(x+y)^2$，那么就有

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = 2t*1 = 2(x+y)$$

1.2.3 链式法则和计算图

如图所示，计算图的反向传播从右到左传播信号。反向传播的计算顺序是，先将节点的输入信号乘以节点的局部导数（偏导数），然后再传递给下一个节点。比如，反向传播时，“$**2$”节点的输入是$\frac{\partial z}{\partial z}$，将其乘以局部导数$\frac{\partial z}{\partial t}$（因为正向传播时输入是$t$、输出是$z$，所以这个节点的局部导数是$\frac{\partial z}{\partial t}$），然后传递给下一个节点。另外，图5-7中反向传播最开始的信号$\frac{\partial z}{\partial t}$在前面的数学式中没有出现，这是因为$\frac{\partial z}{\partial t}=1$，所以在刚才的式子中被省略了。

1.3 反向传播

上一节介绍了计算图的反向传播是基于链式法则成立的。本节将以“$+$”和“$\times$”等运算为例，介绍反向传播的结构。

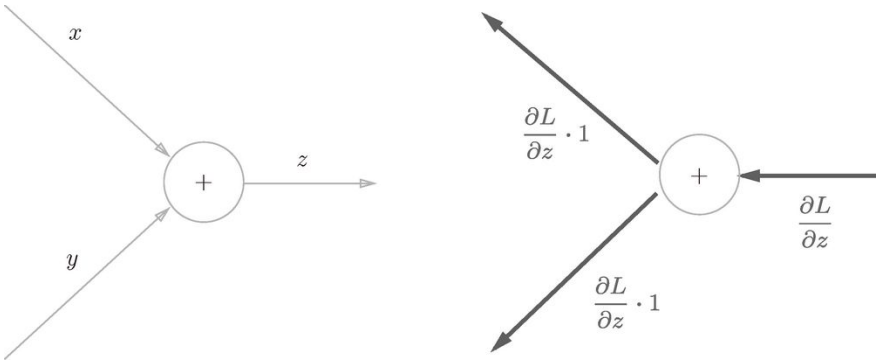

Figure 1.3: Backpropagation for Adder nodes

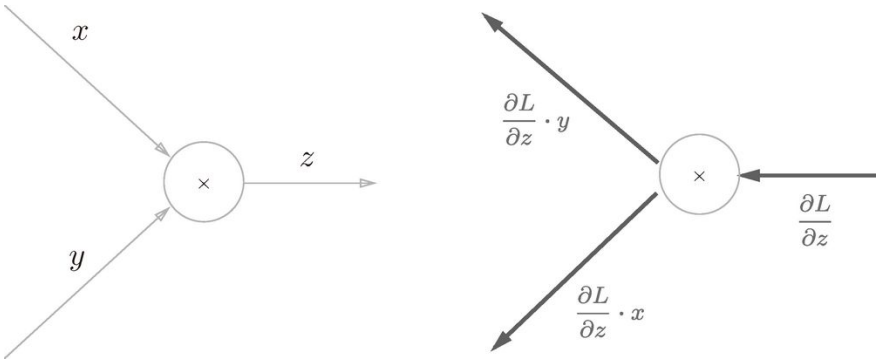

Figure 1.4: Backpropagation of multiply nodes

1.3.1 加法节点的反向传播

考虑加法节点的反向传播。这里以$z = x + y$为对象，观察它的反向传播。$z = x + y$的导数可由下式（解析性地）计算出来：

$$\frac{\partial z}{\partial x} = 1$$
$$\frac{\partial z}{\partial y} = 1$$

Figure 1.3, 反向传播将从上游传过来的导数乘以1，然后传向下游。也就是说，因为加法节点的反向传播只乘以1，所以输入的值会原封不动地流向下一个节点。

1.3.2 乘法节点的反向传播

看一下乘法节点的反向传播。这里我们考虑$z = xy$。这个式子的导数用下式表示：

$$\frac{\partial z}{\partial x} = y$$
$$\frac{\partial z}{\partial y} = x$$

乘法的反向传播会将上游的值乘以正向传播时的输入信号的“翻转值”后传递给下游。

加法的反向传播只是将上游的值传给下游，并不需要正向传播的输入信号。但是，乘法的反向传播需要正向传播时的输入信号值。因此，实现乘法节点的反向传播时，要保存正向传播的输入信号。

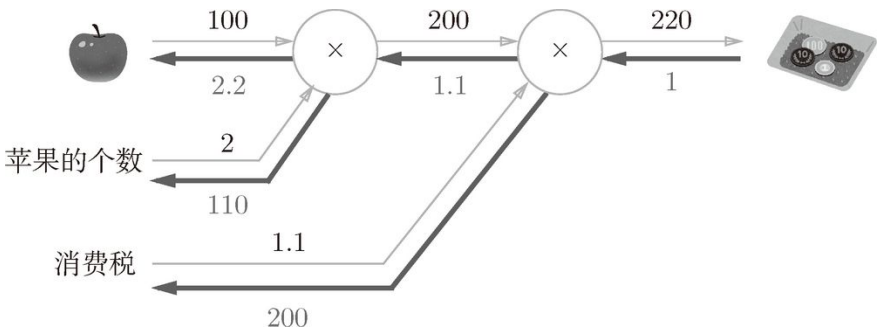

Figure 1.5: Example of backpropagation for buying apples

1.4 简单层的实现

我们把要实现的计算图的乘法节点称为“乘法层”（MulLayer），加法节点称为“加法层”（AddLayer）。

1.4.1 乘法层的实现

层的实现中有两个共通的方法（接口）forward()和backward()。forward() 对应正向传播，backward()对应反向传播。

1.5 激活函数层的实现

现在，我们将计算图的思路应用到神经网络中。这里，我们把构成神经网络的层实现为一个类。先来实现激活函数的 ReLU层和 Sigmoid层。

1.5.1 ReLU层

激活函数ReLU（Rectified Linear Unit）由下式表示：

$$y = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

可以求出y关于x的导数，如下式所示：

$$\frac{\partial y}{\partial x} = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \tag{1.3}$$

Equation 1.3中，如果正向传播时的输入x大于0，则反向传播会将上游的值原封不动地传给下游。反过来，如果正向传播时的x小于等于0，则反向传播中传给下游的信号将停在此处。

> ReLU层的作用就像电路中的开关一样。正向传播时，有电流通过的话，就将开关设为ON；没有电流通过的话，就将开关设为OFF。反向传播时，开关为ON的话，电流会直接通过；开关为OFF的话，则不会有电流通过。

1.5.2 Sigmoid层

sigmoid函数由下式表示：

$$y = \frac{1}{1 + \exp(-x)}$$

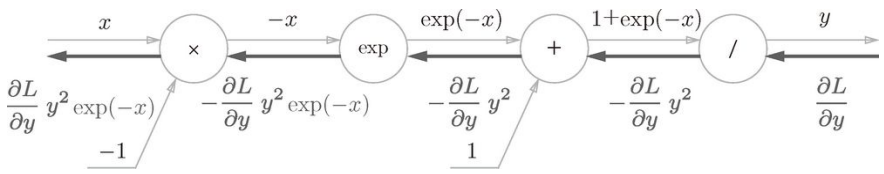

Figure 1.6: Computational graph of the Sigmoid layer

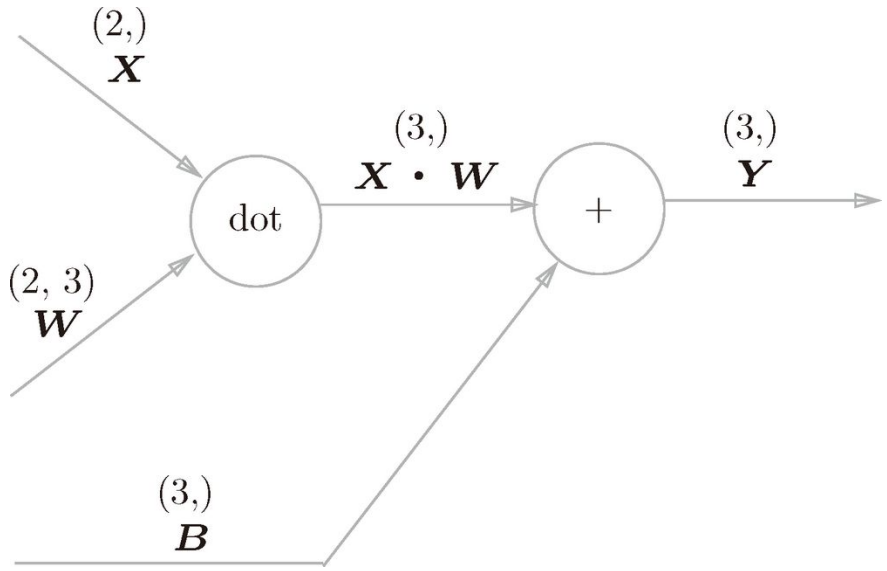

Figure 1.7: Computational graph of the Affine layer

另外，$\frac{\partial L}{\partial y}y^2\exp(-x)$可以进一步整理如下：

$$\begin{aligned}\frac{\partial L}{\partial y}y^2\exp(-x) &= \frac{\partial L}{\partial y}\frac{1}{(1+\exp(-x))^2}\exp(-x)\\ &= \frac{\partial L}{\partial y}\frac{1}{1+\exp(-x)}\frac{\exp(-x)}{1+\exp(-x)}\\ &= \frac{\partial L}{\partial y}y(1-y)\end{aligned}$$

1.6 Affine/Softmax层的实现

1.6.1 Affine层

神经元的加权和可以用`Y = np.dot(X, W) + B`计算出来。然后，`Y`经过激活函数转换后，传递给下一层。这就是神经网络正向传播的流程。

神经网络的正向传播中进行的矩阵的乘积运算在几何学领域被称为“仿射变换”[a]。因此，这里将进行仿射变换的处理实现为“Affine层”。

[a]几何中，仿射变换包括一次线性变换和一次平移，分别对应神经网络的加权和运算与加偏置运算。

Figure 1.7展示了Affine层的计算图（注意变量是矩阵，各个变量的上方标记了该变量的形状）

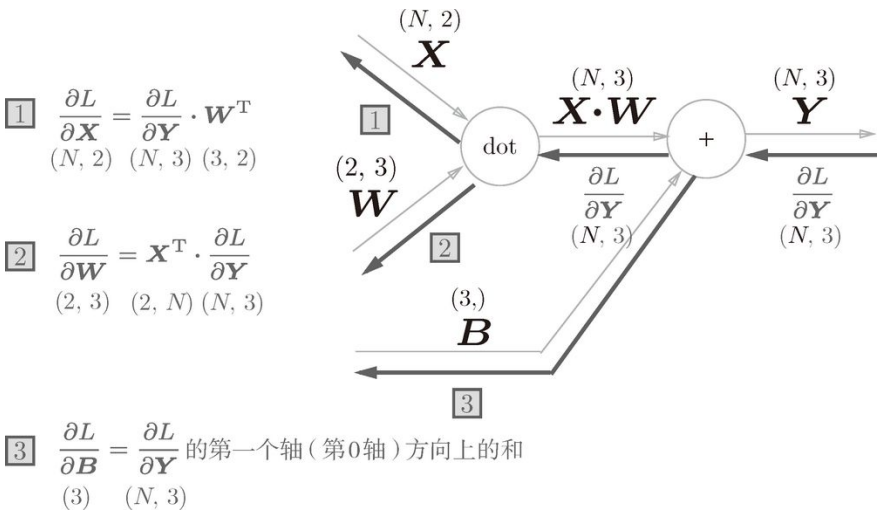

Figure 1.8: Computational graph of the batch version of the Affine layer

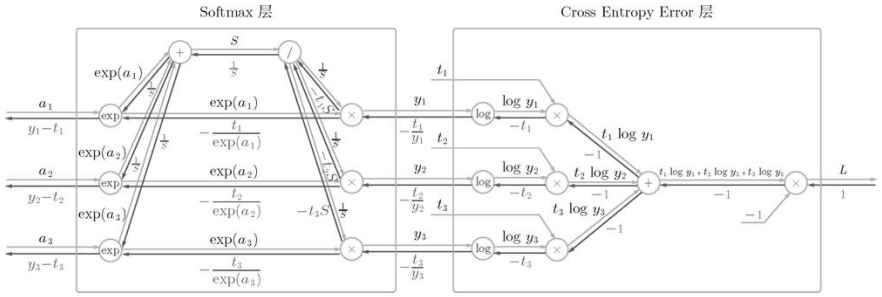

Figure 1.9: Computational graph of the Softmax-with-Loss layer

以矩阵为对象的反向传播，按矩阵的各个元素进行计算时，步骤和以标量为对象的计算图相同。实际写一下的话，可以得到下式

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{X}} &= \frac{\partial L}{\partial \boldsymbol{Y}}\boldsymbol{W}^T \\ \frac{\partial L}{\partial \boldsymbol{W}} &= \boldsymbol{X}^T\frac{\partial L}{\partial \boldsymbol{Y}}\end{aligned} \tag{1.4}$$

1.6.2 批版本的Affine层

前面介绍的Affine层的输入$\boldsymbol{X}$是以单个数据为对象的。现在我们考虑N 个数据一起进行正向传播的情况，也就是批版本的Affine层。

正向传播时，偏置会被加到每一个数据（第1个、第2个……）上。因此，反向传播时，各个数据的反向传播的值需要汇总为偏置的元素。

1.6.3 Softmax-with-Loss 层

softmax函数会将输入值正规化之后再输出。

考虑到这里也包含作为损失函数的交叉熵误差（cross entropy error），所以称为“Softmax-with-Loss层”。Softmax-with- Loss层（Softmax函数和交叉熵误差）的计算图如Figure 1.9所示。

反向传播的具体的推导参见Appendix A

Figure 1.10计算图中，softmax 函数记为 Softmax 层，交叉熵误差记为 Cross Entropy Error层。这里假设要进行3类分类，从前面的层接收3个输入（得分）。如Figure 1.10 所示，Softmax 层将输入$(a1, a2, a3)$正

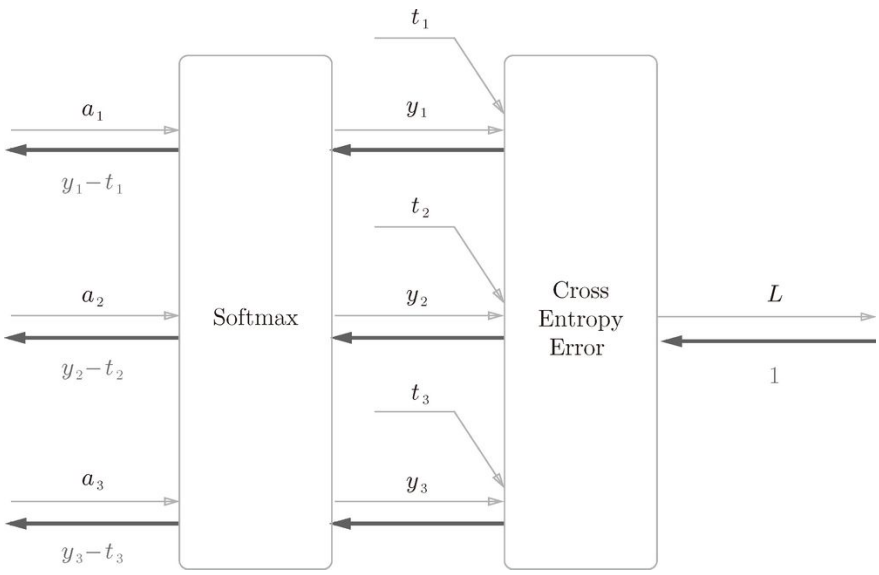

Figure 1.10: Simple version of the calculation graph of the Softmax-with-Loss layer

规化，输出(y_1,y_2,y_3)。Cross Entropy Error层接收Softmax的输出(y_1,y_2,y_3)和训练标签(t_1,t_2,t_3)，从这些数据中输出损失L。

Softmax层的反向传播得到了$(y_1-t_1,y_2-t_2,y_3-t_3)$这样“漂亮”的结果。由于$y_1,y_2,y_3$是Softmax层的输出，$(t_1,t_2,t_3)$是监督数据，所以$(y_1-t_1,y_2-t_2,y_3-t_3)$是Softmax层的输出和训练标签的差分。神经网络的反向传播会把这个差分表示的误差传递给前面的层，这是神经网络学习中的重要性质。

神经网络学习的目的就是通过调整权重参数，使神经网络的输出（Softmax 的输出）接近训练标签。因此，必须将神经网络的输出与训练标签的误差高效地传递给前面的层。刚刚的$(y_1-t_1,y_2-t_2,y_3-t_3)$正是Softmax层的输出与训练标签的差，直截了当地表示了当前神经网络的输出与训练标签的误差。

好的损失函数的意义

使用交叉熵误差作为 softmax 函数的损失函数后，反向传播得到 $(y_1-t_1,y_2-t_2,y_3-t_3)$这样“漂亮”的结果。实际上，这样“漂亮” 的结果并不是偶然的，而是为了得到这样的结果，特意设计了交叉熵误差函数。回归问题中输出层使用“恒等函数”，损失函数使用“平方和误差”，也是出于同样的理由。也就是说，使用“平方和误差”作为“恒等函数”的损失函数，反向传播才能得到$(y_1-t_1,y_2-t_2,y_3-t_3)$这样“漂亮”的结果。

Appendix A

Softmax-with-Loss层的计算图

A.1 反向传播

A.1.1 Cross Entropy Error层

A.1.2 Softmax层

正向传播时若有分支流出，则反向传播时它们的反向传播的值会相加。

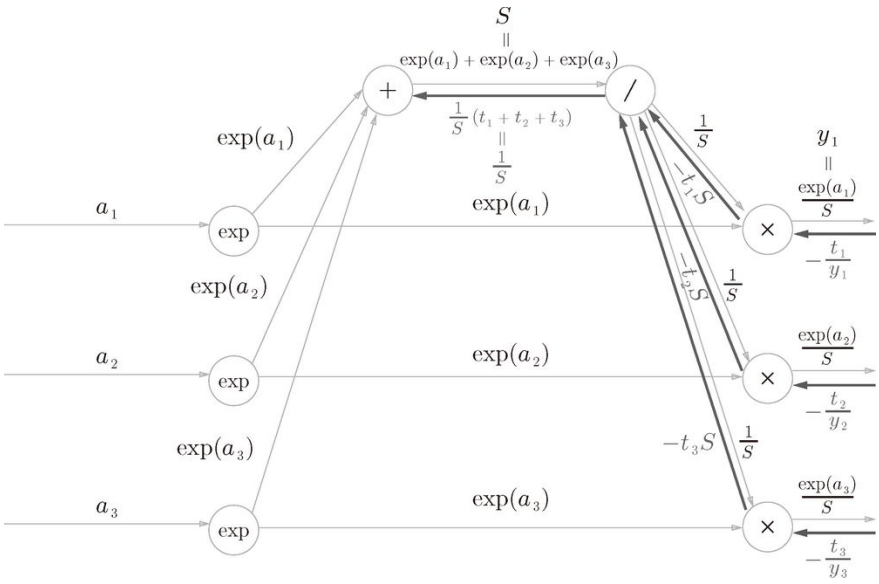

Figure A.1: Step 3