> 本文由 简悦 SimpRead 转码， 原文地址 zhuanlan.zhihu.com

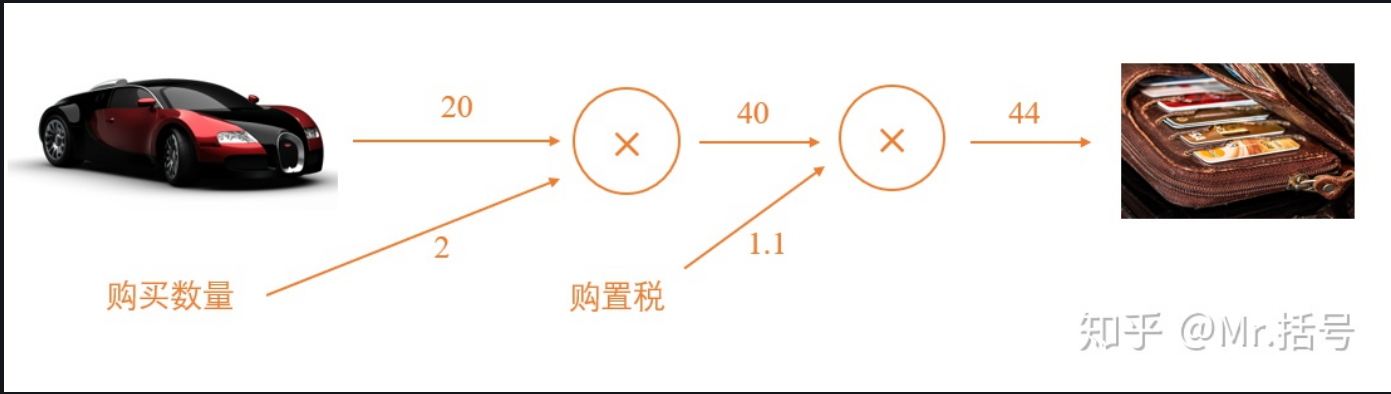
上一篇 Mr. 括号：神经网络 15 分钟入门！足够通俗易懂了吧文章中对两层神经网络进行了描述，从中我们知道神经网络的过程就是正向传播得到 Loss 值，再把 Loss 值反向传播，并对神经网络的参数进行更新。其中反向传播正是神经网络的要点所在。

本篇将对反向传播的内容进行讲解，力求通俗，毕竟只有 15 分钟时间~

链式法则

在讲反向传播之前先讲一下链式法则。

假设一个场景，一辆汽车 20 万元，要收 10% 的购置税，如果要买 2 辆，则正向传播的过程可以画成：



汽车单价 20 万，最终需要支付 44 万，我现在想知道汽车单价每波动 1 万，对最终支付价格的影响是多少。参看下图：我们从右向左依次求导，得到的值分别为

①44/44=1

②44/40=1.1

③40/20=2

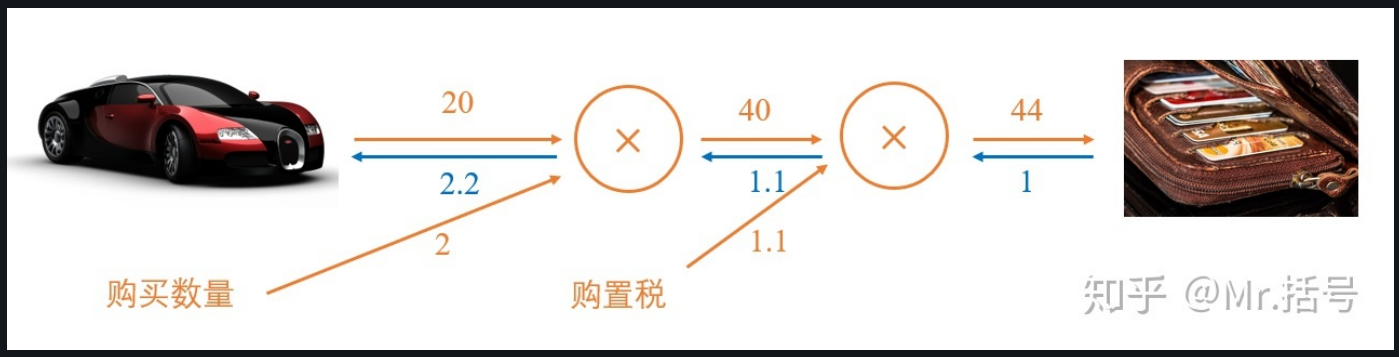
那么最终价格相对于汽车单价的导数就是①×②×③=2.2

这就是链式法则。我们只需要知道每个节点导数值，然后求乘积就可以了。

链式法则的一种定义 * 是：

> 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。

所以我们只需要关注每个节点的导数值即可。

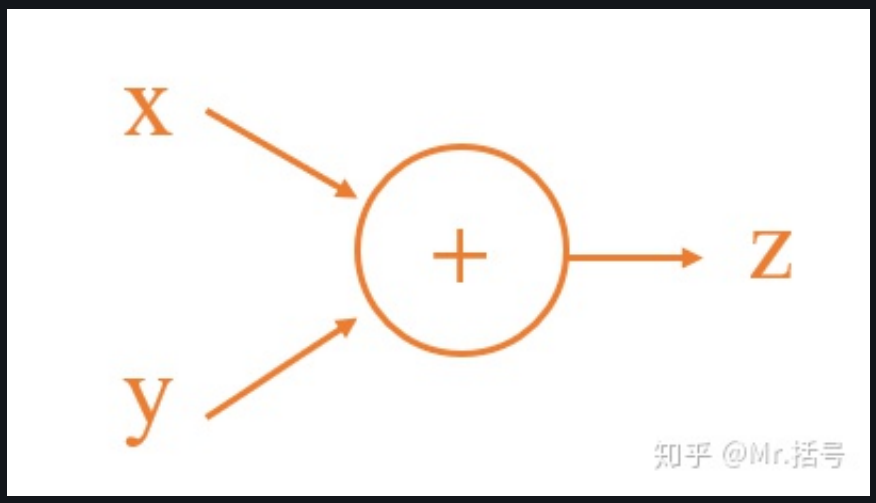


反向传播

下边介绍几种典型节点的反向传播算法。

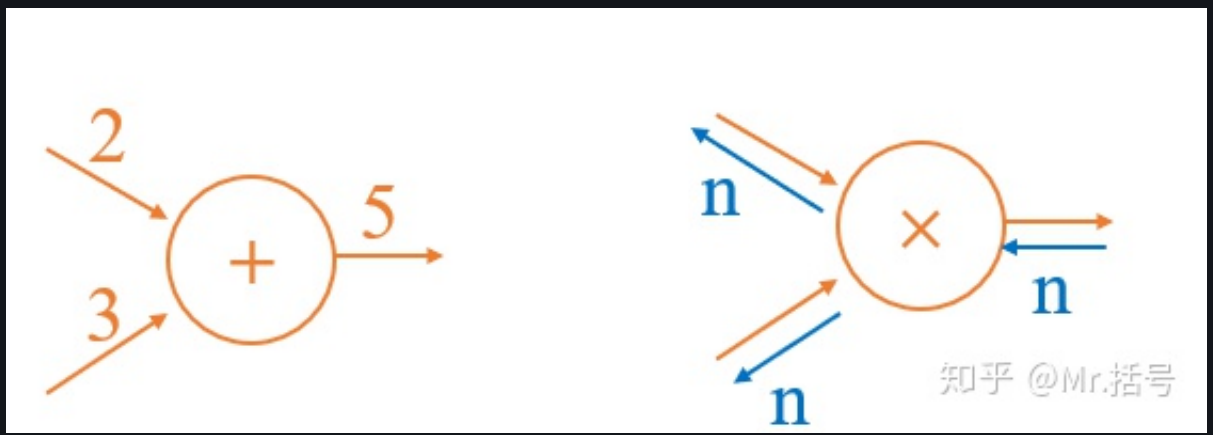
加法节点

如下图：该节点可以写作 z=x+y



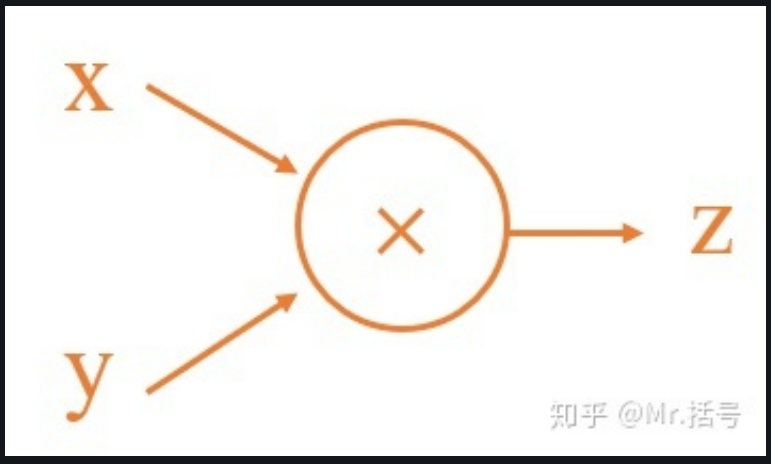
很容易知道，z 对 x 求导等于 1，对 y 求导也等于 1，所以在加法节点反向传递时，输入的值会**原封不动**地流入下一个节点。

比如：



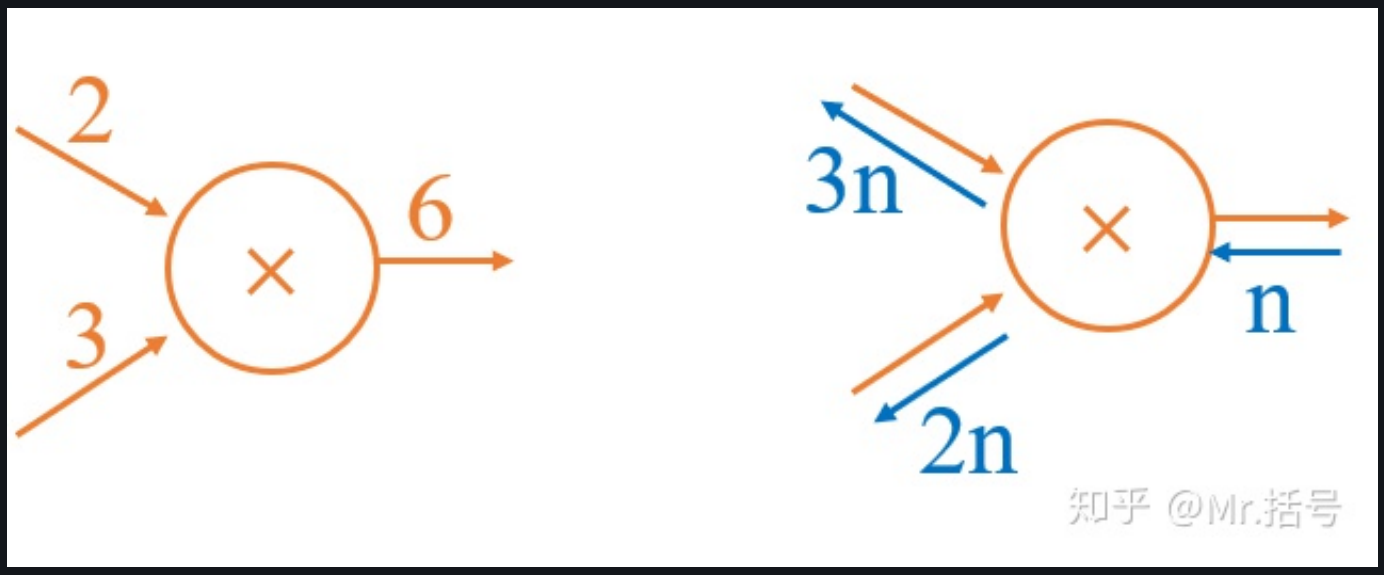
乘法节点

如下图，该节点可以写作 z=x*y



同样很容易知道，z 对 x 求导等于 y，对 y 求导等于 x，所以在加法节点反向传递时，输入的值**交叉相乘**然后流入下一个节点。

比如：

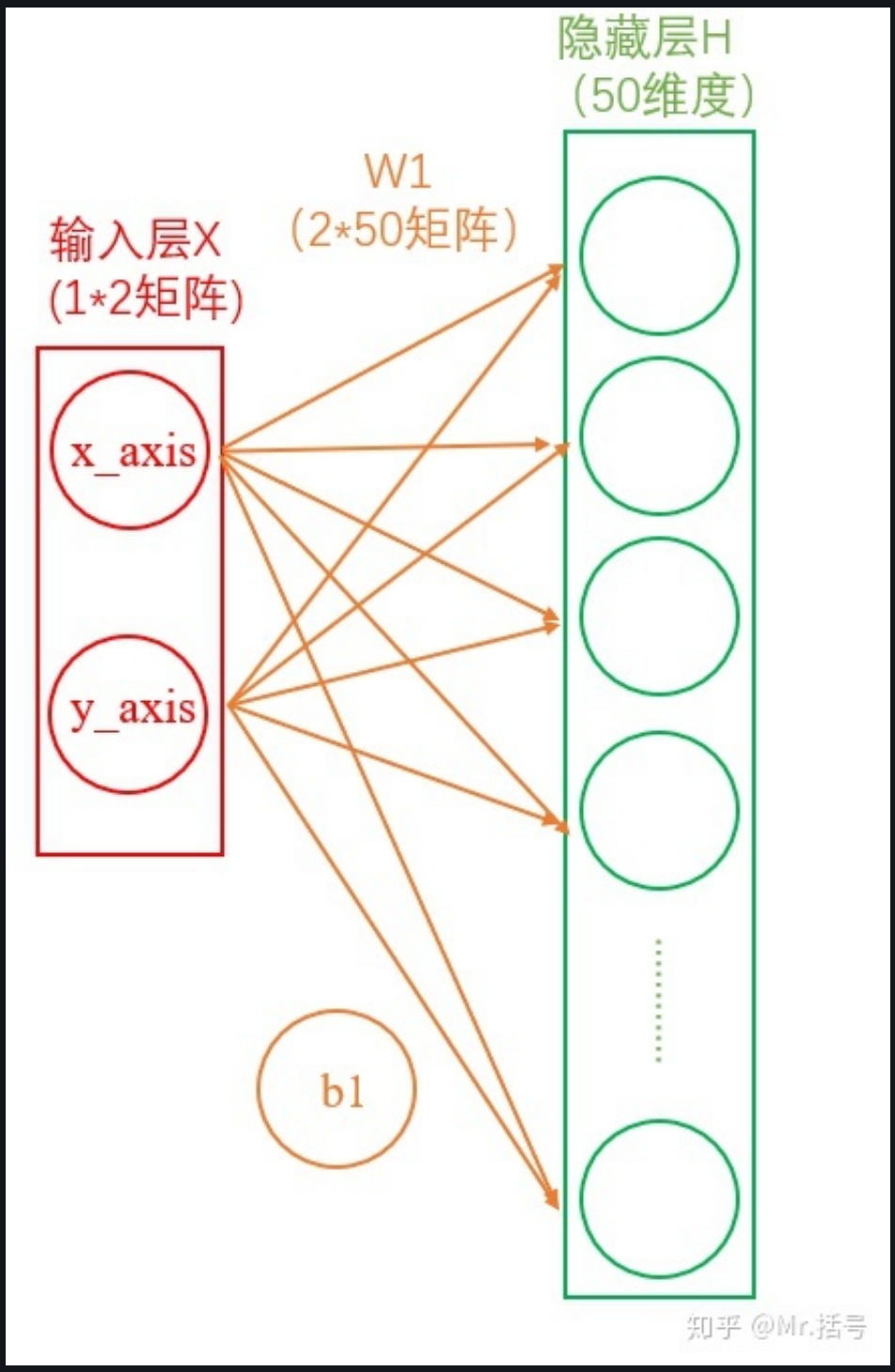


仿射变换

所谓仿射变换就是这个式子，如果觉得眼生就去看上一篇文章。

$$H=X*W1+b1$$

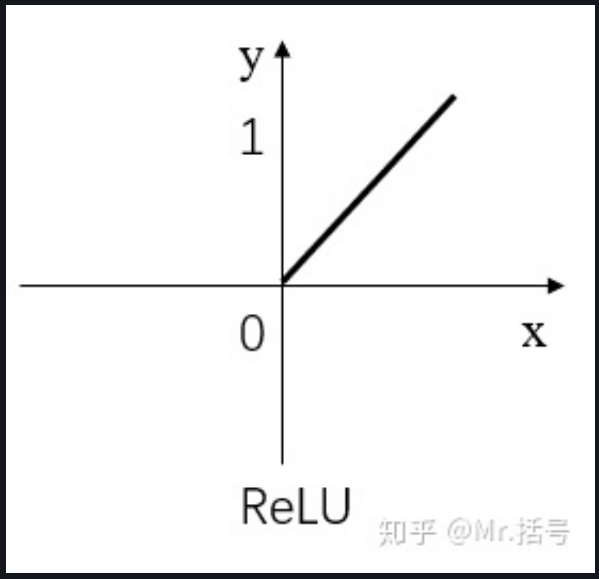
画成图的话就是：



这是神经网络里的一个重要形式单元。这个图片看起来虽然复杂，但其实和乘法节点是类似的，我们对 X 求导，结果就是 W1；对 W1 求导，结果就是 X，到这里和乘法节点是一样的；对 b1 求导，结果为 1，**原封不动**地流入即可。不过需要注意的一点是，这里的相乘是向量之间的乘法。

ReLU 层

激活层我们就以 ReLU 为例。回忆一下，ReLU 层的形式是这样的：

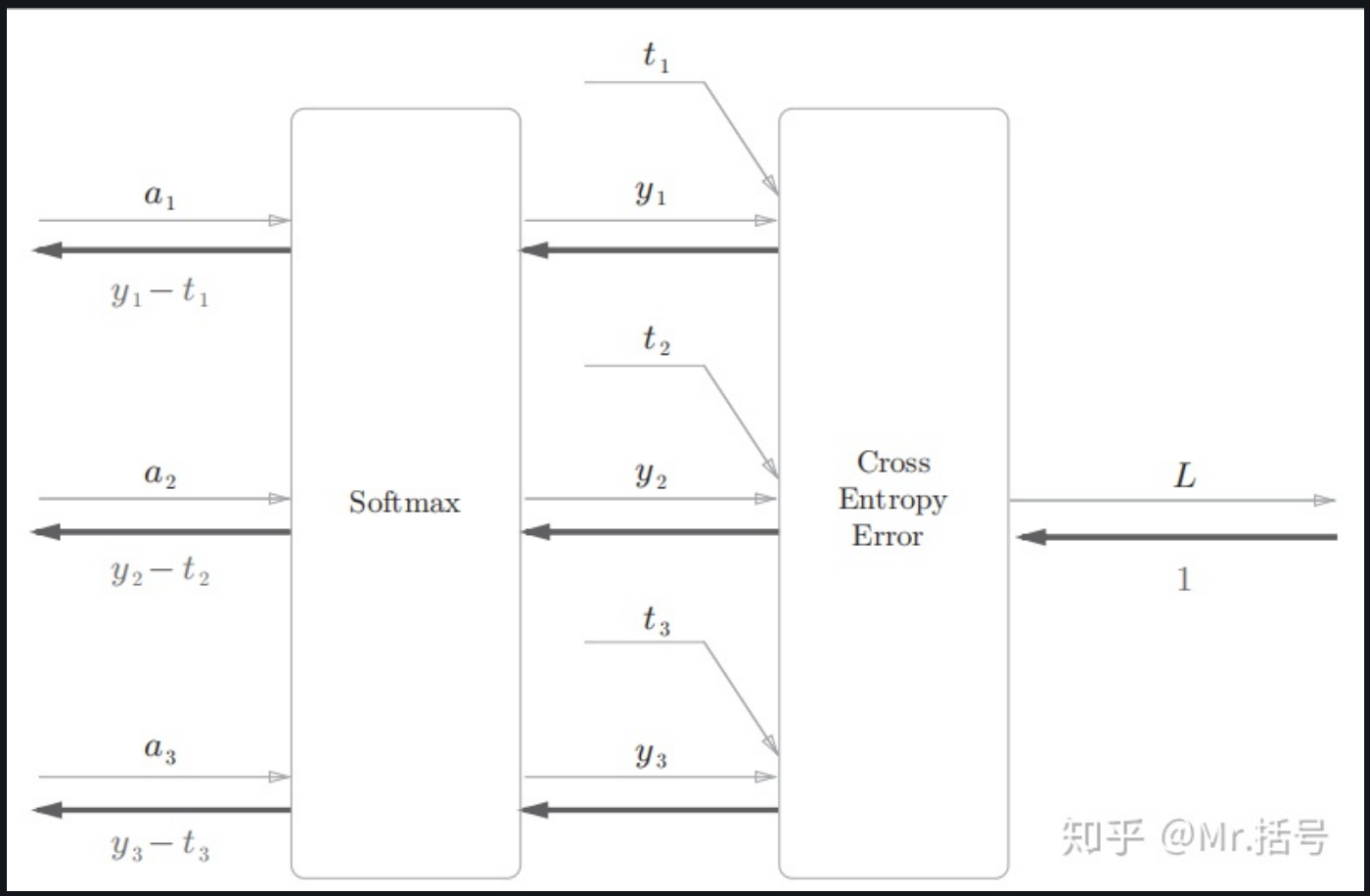


因为当 x>0 时，y=x，求导为 1，也就是原封不动传递。

当 x<=0 时，y=0，求导为 0，也就是传递值为 0。

Softmax-with-Loss

Softmax-with-Loss 指的就是 Softmax 和交叉熵损失的合称。这是我们之前提到的神经网络的最后一个环节。这部分的反向传播推导过程比较复杂，这里直接上结论吧（对推导过程感兴趣的话可以看文末参考文献 * 的附录 A）：



其中

> 从前面的层输入的是 (a1, a2, a3)，softmax 层输出 (y1, y2, y3)。此外，教师标签是 (t1, t2, t3)，Cross Entropy Error 层输出损失 L。

所谓教师标签，就是表示是否分类正确的标签，比如正确分类应该是第一行的结果时，(t1, t2, t3) 就是 (1,0,0)。

从上图可以看出，Softmax-with-Loss 的反向传播的结果为 (y1 − t1, y2 − t2, y3 − t3)。

参数更新

参数的更新对象其实就是 W 和 b，具体的在 2.3 中对其更新方法进行了描述，简单来说，dW 就是输入值乘以 X，db 就等于输入值。这里用 dW 和 db 表示反向传播到 W 和 b 节点时的计算结果。

那现在该怎样更新 W 和 b 呢?

直接用 W=W-dW；b=b-db 么?

可以，但不太好。

其一，需要引入**正则化惩罚项**。这是为了避免最后求出的 W 过于集中所设置的项，比如 [1/3,1/3,1/3] 和[1,0,0]，这两个结果明显前一个结果更为分散，也是我们更想要的。为了衡量分散度，我们用$1/2W^2$ 来表示。对该式求导，结果就是 W。设正则化惩罚项的系数值为 reg，那么修正后的 dW 可以写为：

$$dW = dW + reg * W$$

其二，是步子迈的有点大。直接反向传播回来的量值可能会比较大，在寻找最优解的过程中可能会直接将最优解越过去，所以在这里设置一个参数：**学习率**。这个数通常很小，比如设学习率为 0.0001 这样。我们将学习率用 epsilon 表示，那么最终更新后的 W 和 b 写为：

$$W = W - epsilon * dW$$

$$b = b - epsilon * db$$

至此，一次反向传播的流程就走完了。

总结

链式法则是反向传播的基本传递方式，它大大简化了反向传播计算的复杂程度。在本例中可能还不太明显，在有些非常复杂的网络中，它的好处会更加显而易见。

另外反向传播的各个节点的算法也是比较重要的内容，本文介绍了常用的节点的反向传播计算结果，实际应用中可能会有更多的形式。不过不用担心，google 一下，你就知道。

参数更新是反向传播的目的，结合例子来看可能会更容易理解。下一篇文章会不使用任何框架，纯手写一个我们之前提到的神经网络，并实现象限分类的问题。

欢迎持续关注我的专栏与信号处理有关的那些东东

欢迎关注我的公众号 “括号的城堡”，微信号为 “khscience”，会有更多有趣的东西分享。

参考：

*《深度学习入门：基于 Python 的理论与实现》