> 本文阅读书籍与参考代码：
>
> https://github.com/MichalDanielDobrzanski/DeepLearningPython

1 使用神经网络识别手写数字

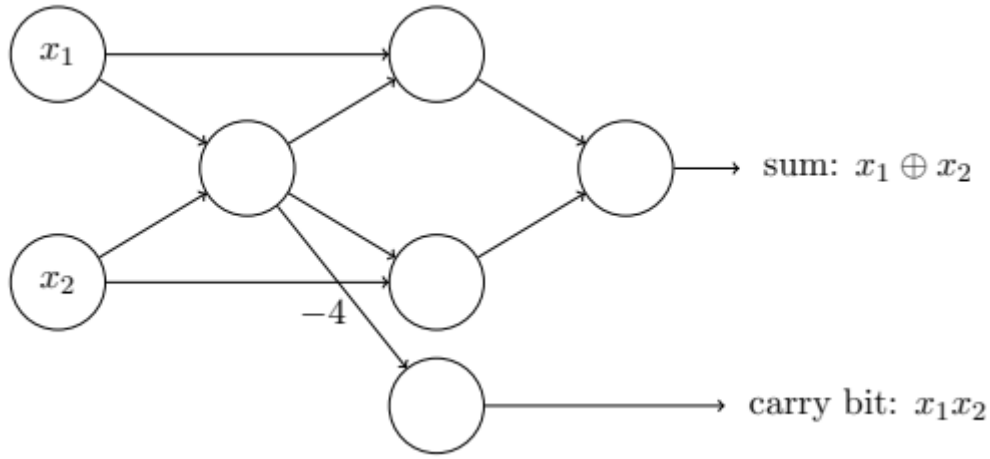
神经网络使用样本来自动推断出识别手写数字的规则。另外，通过增加训练样本的数量，网络可以学到更多关于手写数字的知识，这样就能够提升自身的准确性。

1.1 感知器

一个感知器接受几个二进制输入，x1, x2, . . .，并产生一个二进制输出。你可以将感知器看作依据权重来作出决定的设备。感知器的规则可以写为：

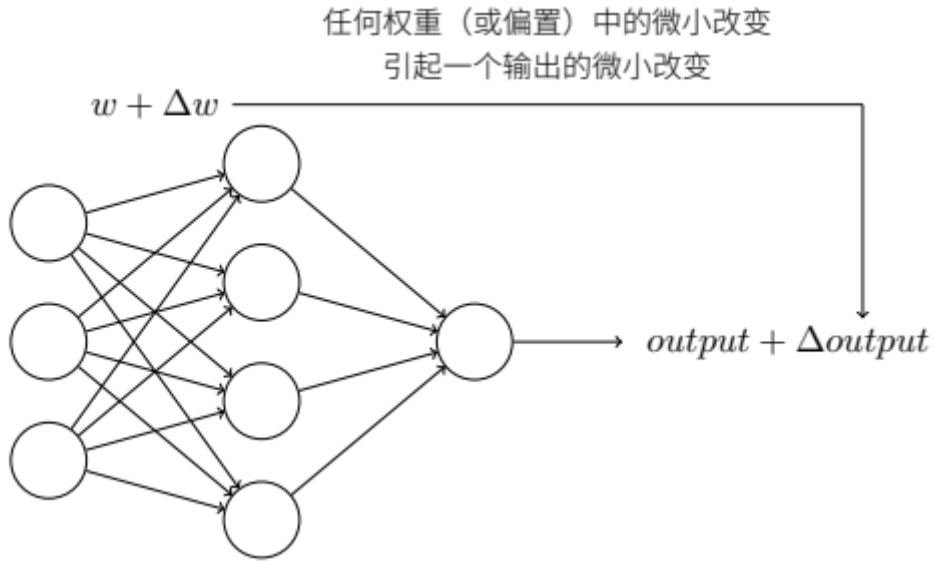
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

感知器网络可以表示为如下，并且我们完全能用感知器网络来计算任何逻辑功能，例如“与”，“或”和“与非”：



1.2 S型神经元

假设我们把网络中的权重（或者偏置）做些微小的改动。就像我们马上会看到的，这一属性会让学习变得可能：



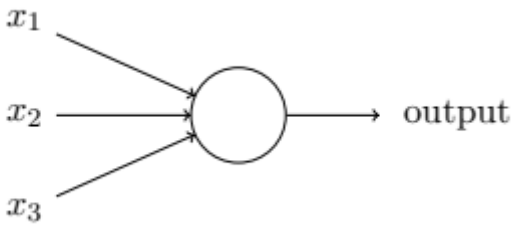
如果对权重（或者偏置）的微小的改动真的能够仅仅引起输出的微小变化，那我们可以利用这一事实来修改权重和偏置，让我们的网络能够表现得像我们想要的那样。

例如，假设网络错误地把一个“9”的图像分类为“8”。我们能够计算出怎么对权重和偏置做些小的改动，这样网络能够接近于把图像分类为“9”。

然后我们要重复这个工作，反复改动权重和偏置来产生更好的输出。这时网络就在学习。

实际上，网络中单个感知器上一个权重或偏置的微小改动有时候会引起那个感知器的输出完全翻转，如 0 变到 1。因此，虽然你的“9”可能被正确分类，网络在其它图像上的行为很可能以一些很难控制的方式被完全改变。

我们引入一种称为 S 型神经元的新的人工神经元来克服这个问题。S 型神经元和感知器类似，但是被修改为权重和偏置的微小改动只引起输出的微小变化。这对于让神经元网络学习起来是很关键的。

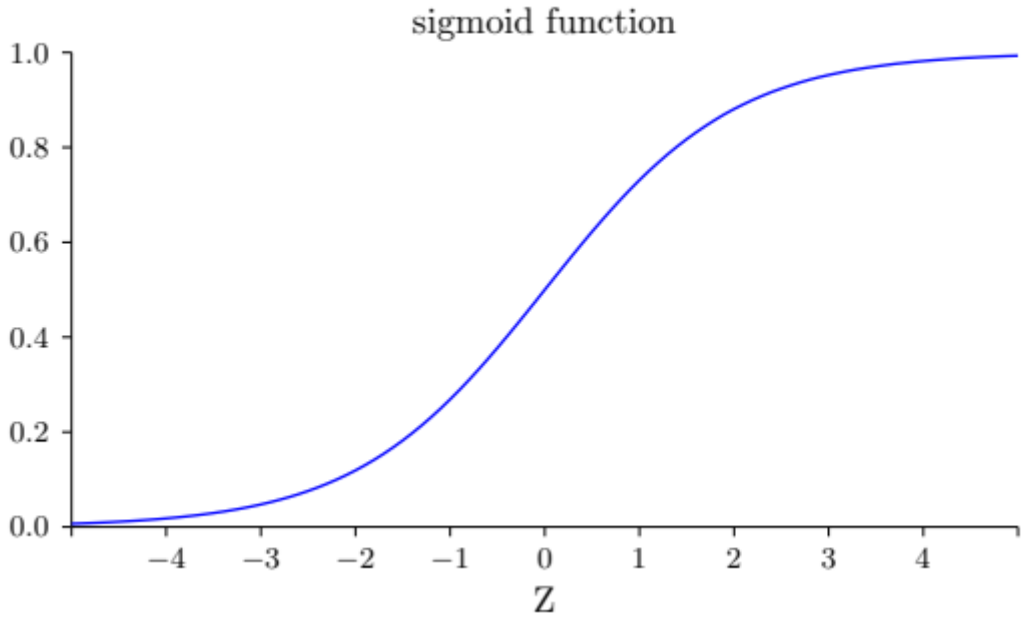


S 型神经元有多个输入，x1, x2, . . .。但是这些输入可以取 0 和 1 中的任意值，而不仅仅是 0 或 1。例如，0.638 。

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$

其中z ≡ w · x + b，σ 有时被称为逻辑函数。

形状如下：



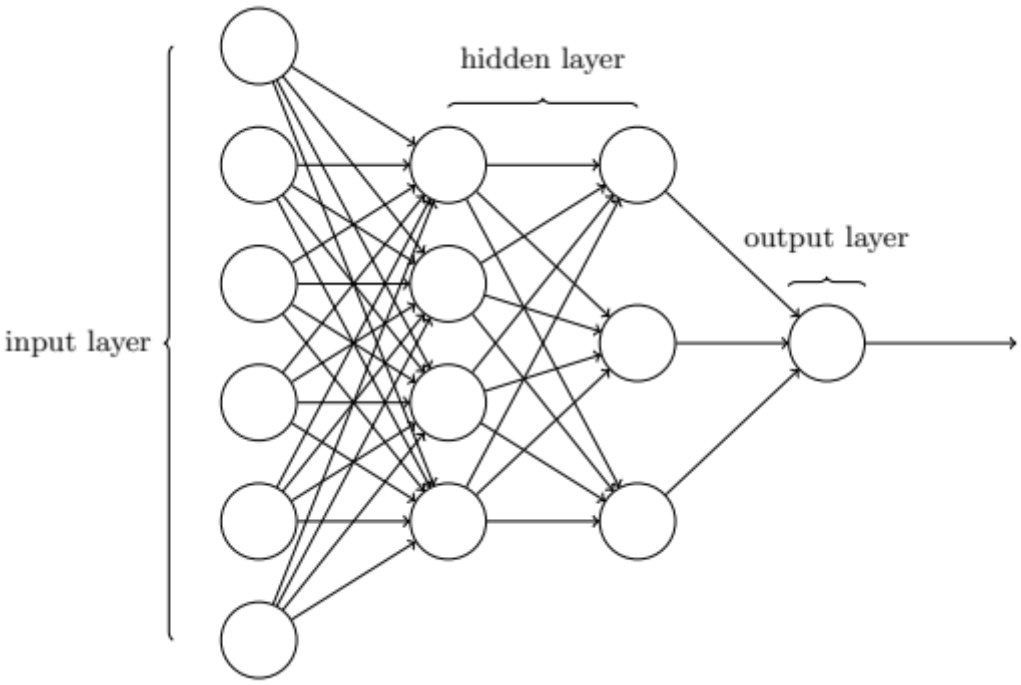
σ 函数的平滑特性，正是关键因素，而不是其细部形式。σ 的平滑意味着权重和偏置的微小变化，即 Δwj 和 Δb，会从神经元产生一个微小的输出变化 Δoutput。实际上它的意思非常简单（这可是个好消息）：Δoutput 是一个反映权重和偏置变化 —— 即 Δwj 和 Δb —— 的线性函数。这一线性使得选择权重和偏置的微小变化来达到输出的微小变化的运算变得容易。

$$\Delta \text{output} \approx \sum_j \frac{\partial \,\text{output}}{\partial w_j} \Delta w_j + \frac{\partial \,\text{output}}{\partial b} \Delta b$$

`S 型神经元不仅仅输出 0 或 1。它可以输出 0 和 1 之间的任何实数，所以诸如 0.173。`

1.3 神经网络的架构

有这样一个网络：



这个网络中，最左边的称为输入层，其中的神经元称为输入神经元。

最右边的，即输出层，包含有输出神经元，在本例中，输出层只有一个神经元。

中间层，既然这层中的神经元既不是输入也不是输出，则被称为隐藏层。“隐藏”这一术语也许听上去有些神秘——但它实际上仅仅意味着“既非输入也非输出”。

1.3.1 网络层的设计

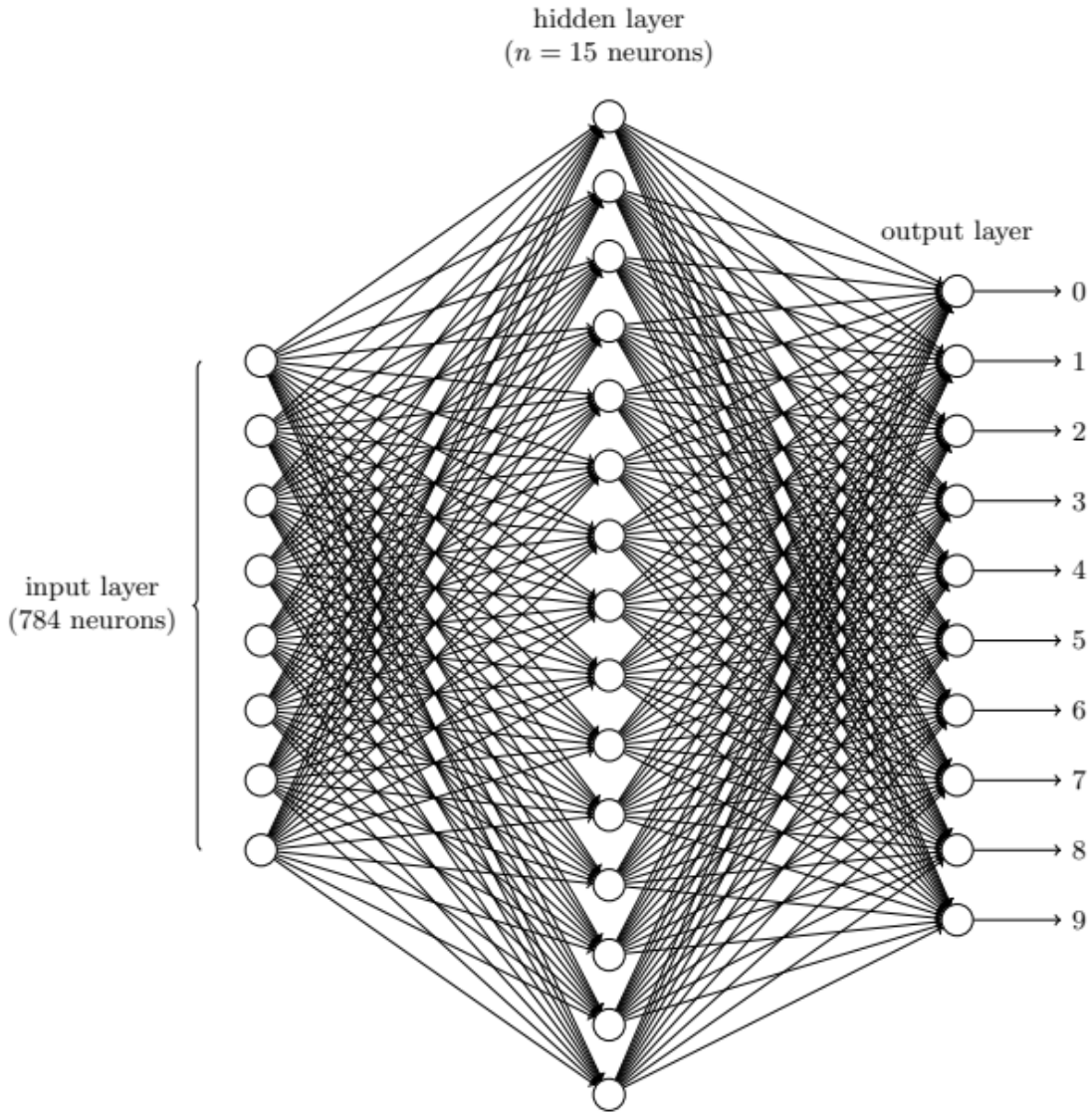
设计网络的输入输出层通常是比较直接的。例如，假设我们尝试确定一张手写数字的图像上是否写的是“9”。很自然地，我们可以将图片像素的强度进行编码作为输入神经元来设计网络。

如果图像是一个 64 × 64 的灰度图像，那么我们会需要 4096 = 64 × 64 个输入神经元，每个强度取 0 和 1 之间合适的值。

输出层只需要包含一个神经元，当输出值小于 0.5 时表示“输入图像不是一个 9”，大于 0.5 的值表示“输入图像是一个 9”。

目前为止，我们讨论的神经网络，都是以上一层的输出作为下一层的输入。这种网络被称为 `前馈神经网络`。

1.4 一个简单的分类手写数字的网络



我们给网络的训练数据会有很多扫描得到的 28 × 28 的手写数字的图像组成，所有输入层包含有 784 = 28 × 28个神经元。输入像素是灰度级的，值为 0.0 表示白色，值为 1.0 表示黑色，中间数值表示逐渐暗淡的灰色。

网络的第二层是一个隐藏层。我们用 n 来表示神经元的数量，我们将给 n 实验不同的数值。示例中用一个小的隐藏层来说明，仅仅包含 n = 15 个神经元。

网络的输出层包含有 10 个神经元。如果第一个神经元激活，即输出 ≈ 1，那么表明网络认为数字是一个 0。

1.4.1 隐藏层的神经元在做什么

假设隐藏层的第一个神经元只是用于检测如下的图像是否存在：

为了达到这个目的，它通过对此图像对应部分的像素赋予较大权重，对其它部分赋予较小的权重。同理，我们可以假设隐藏层的第二，第三，第四个神经元是为检测下列图片是否存在：

这四幅图像组合在一起构成了前面显示的一行数字图像中的 0：

1.4.2 练习——十进制转二进制

通过在上述的三层神经网络加一个额外的一层就可以实现按位表示数字。额外的一层把原来的输出层转化为一个二进制表示，如下图所示。为新的输出层寻找一些合适的权重和偏置。假定原先的 3 层神经网络在第三层得到正确输出（即原来的输出层）的激活值至少是0.99，得到错误的输出的激活值至多是0.01。

原数字	二进制表示
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

四位二进制用abcd表示，当十进制输入为8，9的时候，a=1。

所以可以设置权重为[0，0，0，0，0，0，0，0，1，1]T

例如，计算第一个神经元a的w·x，当第三层十进制输出8时：

1. 激活值：当该位激活时=0.99，未激活时=0.01。第三层输出8时，8那个位置激活了，所以激活值=0.99
2. 9的权重为1，但是未激活，所以激活值为0.01
3. 计算`各位的权重·激活值并进行相加`，结果=1.00

原数字	权重	激活值	计算w·激活值
0	0	0.01	0
1	0	0.01	0
2	0	0.01	0
3	0	0.01	0
4	0	0.01	0
5	0	0.01	0
6	0	0.01	0
7	0	0.01	0
8	1	0.99	0.99
9	1	0.01	0.01
			1.00

又例如，计算第三个神经元c的w·x，当第三层十进制输出7时：

1. 激活值：当该位激活时=0.99，未激活时=0.01。第三层输出7时，7那个位置激活了，所以激活值=0.99
2. 2、3、6的权重为1，但是未激活，所以激活值为0.01
3. 计算`各位的权重·激活值并进行相加`，结果=1.02

原数字	权重	激活值	计算w·激活值
0	0	0.01	0
1	0	0.01	0
2	1	0.01	0.01
3	1	0.01	0.01
4	0	0.01	0
5	0	0.01	0
6	1	0.01	0.01
7	1	0.99	0.99
8	0	0.01	0
9	0	0.01	0
			1.02

最后可以得出总表：

数字	w·x（第一个神经元）	第二个神经元	第三个神经元	第四个神经元
0	0.02	0.04	0.04	0.05
1	0.02	0.04	0.04	1.03
2	0.02	0.04	1.02	0.05
3	0.02	0.04	1.02	1.03
4	0.02	1.02	0.04	0.05
5	0.02	1.02	0.04	0.05
6	0.02	1.02	1.02	0.05
7	0.02	1.02	1.02	1.03
8	1.00	0.04	0.04	0.05
9	1.00	0.04	0.04	0.05

可以看出最大误差为0.05，那么我们可以设置偏置b为-0.06，就可以消去误差，然后规定至少大于0激活。比如，当第三层输出的数字为7，得出4个神经元adcd的输出w·x分别为[0.02，1.02，1.02，1.03]，这时候去掉误差-0.06，得到[-0.04，0.96，0.96，0.97]，-0.04未激活，结果=0。此时adcd=0111，也就是十进制的7。

1.5 使用梯度下降算法进行学习

把每个训练输入 x 看作一个 28 × 28 = 784维的向量。每个向量中的项目代表图像中单个像素的灰度值。我们用 y = y(x) 表示对应的期望输出，这里 y 是一个 10 维的向量。

我们希望有一个算法，能让我们找到权重和偏置，以至于网络的输出 y(x) 能够拟合所有的训练输入 x。为了量化我们如何实现这个目标，我们定义一个损失函数：

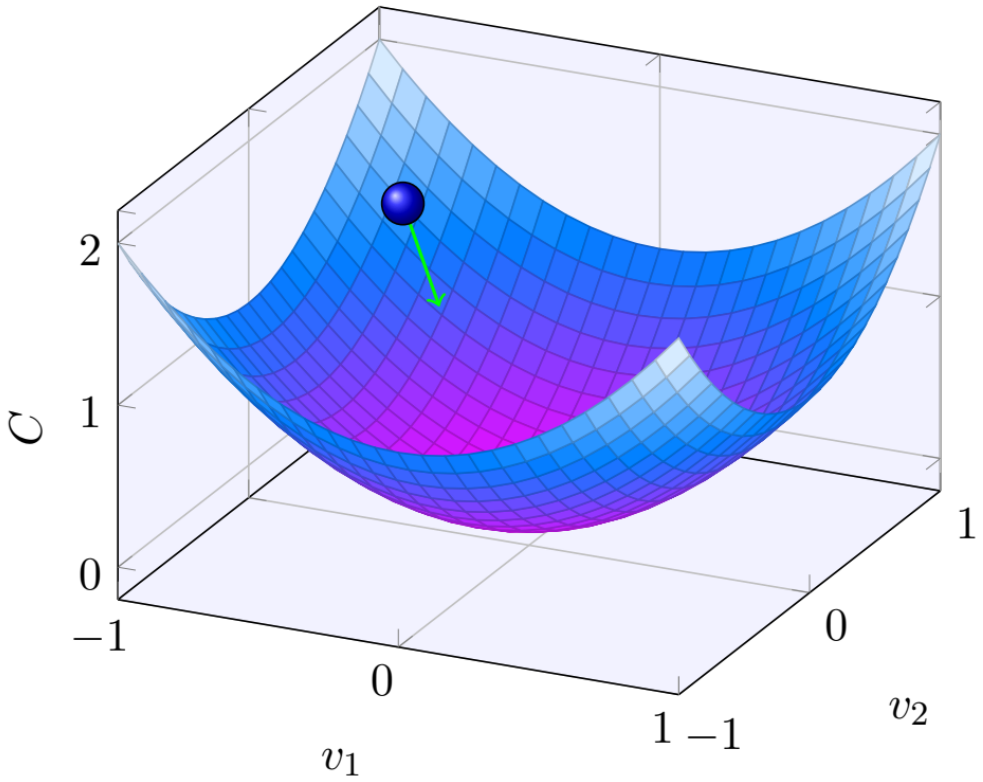
$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$

这里 w 表示所有的网络中权重的集合，b 是所有的偏置，n 是训练输入数据的个数，a 是表示当输入为 x 时输出的向量，求和则是在总的训练输入 x 上进行的。

我们训练算法的目的，是最小化权重和偏置的代价函数 C(w, b)。换句话说，`我们想要找到一系列能让代价尽可能小的权重和偏置。`

我们将采用称为梯度下降的算法来达到这个目的。

梯度下降算法工作的方式就是重复计算梯度 ∇C，然后沿着相反的方向移动，沿着山谷"滚落"。我们可以想象它像这样：



梯度下降法可以被视为一种在 C 下降最快的方向上做微小变化的方法。

1.6 实现我们的网络来分类数字

一旦我们给一个网络学会了一组好的权重集和偏置集，它能很容易地被移植到网络浏览器中以 Javascript 运行，或者如在移动设备上的本地应用。

建立一个`test1.py`文件，写入以下代码：

```
import mnist_loader
import network

training_data, validation_data, test_data = mnist_loader.load_data_wrapper()
net = network.Network([784, 30, 10])
net.SGD(training_data, 30, 10, 3.0, test_data=test_data)
```

代码输出如下，最高准确率为95.08%：

```
Epoch 0 : 8937 / 10000
...
Epoch 19 : 9508 / 10000
...
Epoch 29 : 9488 / 10000
```

将隐藏层神经元数量改到 100，准确率提升到了96%以上，至少在这种情况下，使用更多的隐藏神经元帮助我们得到了更好的结果。

当然，为了获得这些准确性，我不得不对训练的迭代期数量，小批量数据大小和学习速率η做特别的选择。正如我上面所提到的，这些在我们的神经网络中被称为超参数，以区别于通过我们的学习算法所学到的参数（权重和偏置）。如果我们选择了糟糕的超参数，我们会得到较差的结果。