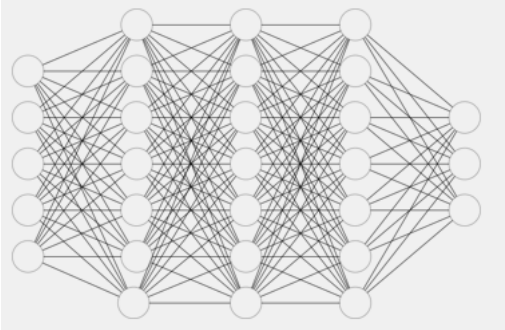
训练深层神经网络

机器学习（入门） DC学院

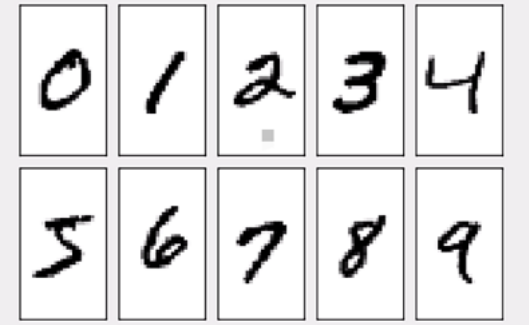
梯度消失

神经网络通过非线性隐藏层的堆叠可以拟合复杂的函数，但是随着层数的增加神经网络会出现学习缓慢甚至无法学习的情况，这样的问题被称为梯度消失问题（vanishing gradient problem）



MNIST

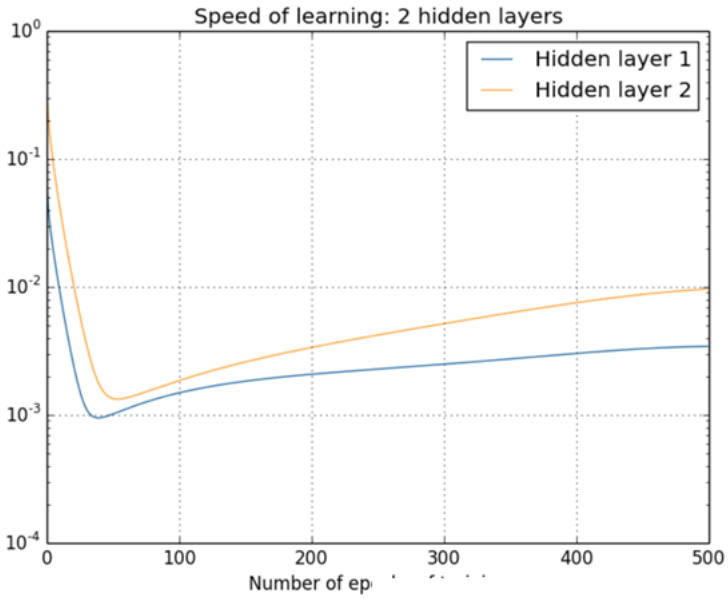
MNIST数据集是来自美国国家标准与技术研究所的手写体数据集，包括从0到9总共10个数字的手写体图像，常被用来做教程或者作为benchmark用于对比模型优劣。

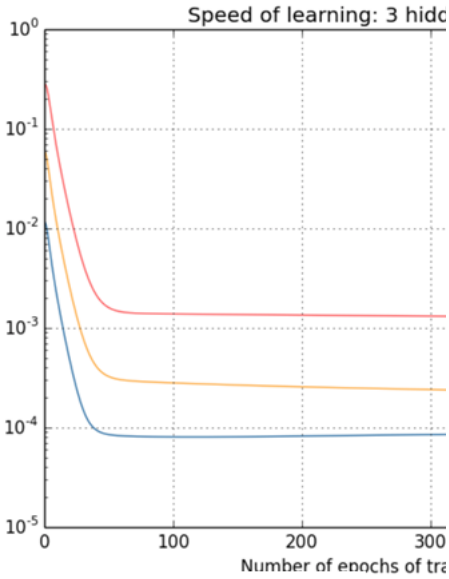


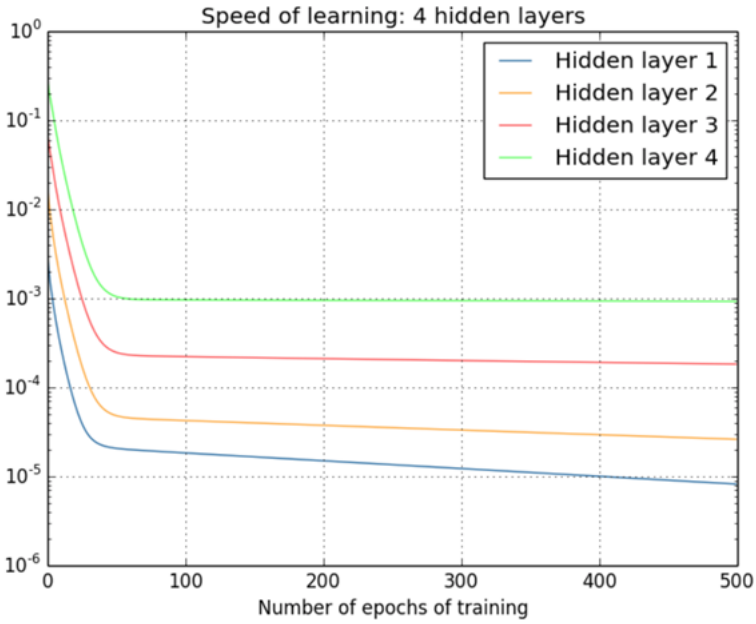
我们用MNIST数据集训练神经网络，其中输入层有784个神经元，每一个隐藏层都包含30个神经元，输出神经元为10个，对应于MNIST数据集的10个分类。我们通过增加隐藏层来比较模型表现：

隐藏层数	准确率
1	96.48
2	96.90
3	96.57
4	96.53

为了更直观的表示深层神经网络中存在的问题，我们一直训练了500轮，同时对数据做平滑处理，将训练过程中每层的学习速率画了出来：



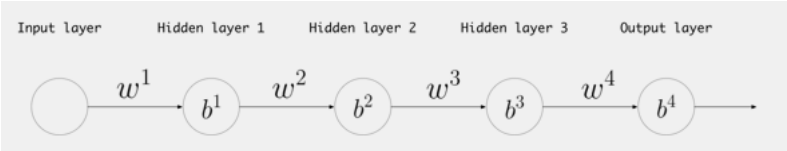




随着网络模型不断的训练，我们发现靠前的层比靠后的层梯度要小，并且随着网络隐藏层的增加，这种问题越发明显。深度神经网络中，隐藏层无法学习的情况被称为**梯度消失问题**。

深度神经网络的学习梯度是不稳定的，这种不稳定性是深度神经网络基于梯度下降学习网络参数的一个最基本的问题。

为了探究为什么存在梯度消失的问题，我们以一个**三层隐藏层的神经网络**为例：



第一个隐藏层偏差变化率：$\frac{\partial J}{\partial b^1} = \sigma'(z^1) \times w^2 \times \sigma'(z^2) \times w^3 \times \sigma'(z^3) \times w^4 \times \sigma'(z^4) \times \frac{\partial J}{\partial a^4}$

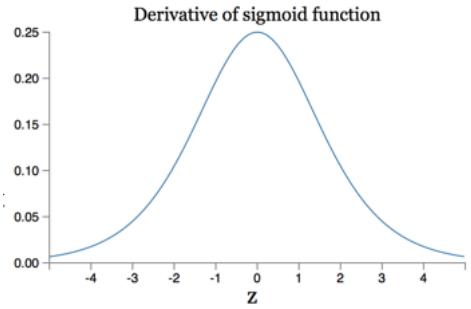
对于b^1的微小变化会引起损失函数J的变化，我们可以假设：$\frac{\partial J}{\partial b^1} \approx \frac{\Delta J}{\Delta b^1}$

对于$a^1 = \sigma(z^1) = \sigma(w^1 a^0 + b^1)$可以得到$\Delta a^1 \approx \frac{\partial \sigma(w^1 a^0 + b^1)}{\partial b^1} \Delta b^1 = \sigma'(z^1)\Delta b^1$

Δa_1会引起$z^2 = w^2 a^1 + b^2$的变化$\Delta z^2 \approx \frac{\partial z^2}{\partial a^1} \Delta a^1 = w^2 \Delta a^1 = \sigma'(z^1) w^2 \Delta b^1$

第一个隐藏层偏差变化率：$\frac{\partial J}{\partial b^1} = \sigma'(z^1) \times w^2 \times \sigma'(z^2) \times w^3 \times \sigma'(z^3) \times w^4 \times \sigma'(z^4) \times \frac{\partial J}{\partial a^4}$

$\sigma'(z)$是sigmoid函数的导数，我们将sigmoid函数画出来如右图所示，可以看出在z=0的时候sigmoid函数取得最大梯度为0.25



在由链式法则求出来的梯度公式中，越靠前层的梯度受到后面层的梯度影响越大，加上sigmoid函数的梯度始终小于1，所以靠前的神经网络层容易出现梯度变得极小的情况，没有学习效果，这样的现象被称为梯度消失。

$$\frac{\partial J}{\partial b^1} = \sigma'(z^1) \times w^2 \times \underbrace{\sigma'(z^2)}_{<\frac{1}{4}} \times w^3 \times \underbrace{\sigma'(z^3)}_{<\frac{1}{4}} \times w^4 \times \sigma'(z^4) \times \frac{\partial J}{\partial a^4}$$

梯度爆炸

对应于梯度消失的情况，同样的还存在梯度爆炸的情况，虽然相对比较少见。比如令$w^2 = w^3 = w^4 = 100$，通过链式法则权重不断相乘使得前面层的变化率变得非常大。这样的现象被称为梯度爆炸。

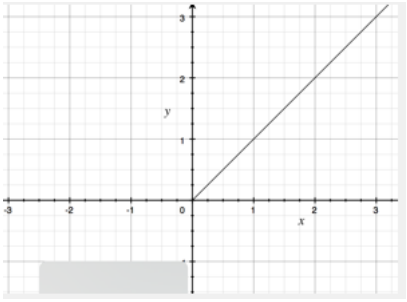
$$\frac{\partial J}{\partial b^1} = \sigma'(z^1) \times w^2 \times \sigma'(z^2) \times w^3 \times \sigma'(z^3) \times w^4 \times \sigma'(z^4) \times \frac{\partial J}{\partial a^4}$$

100

ReLU

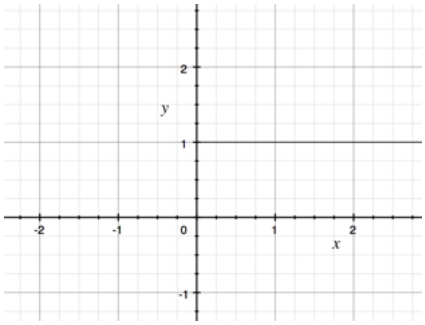
前面提到深度神经网络中存在的梯度问题一部分是由于sigmoid函数本身而导致，通过选择其他的激活函数可以很好的减少这种问题的发生。于是我们引入了矫正线性单元（ReLU，即rectified linear units）：

$A(x)$=max(0, x)



对激活函数ReLU求导的结果如下所示：

$$\frac{dA(x)}{dx} = \begin{cases} 1, if\ x > 0 \\ 0, if\ x \le 0 \end{cases}$$



在x大于0的时候，ReLU的导数始终为1。在将ReLU作为神经网络每层的激活函数时，即使经过层层网络的传播也可以保证前面层的变化率可以相对保持稳定，这也就是为什么ReLU可以使得深层神经网络更加容易训练的原因：

$$\frac{dA(x)}{dx} = \begin{cases} 1, if\ x > 0 \\ 0, if\ x \leq 0 \end{cases}$$

$$\frac{\partial J}{\partial b^1} = A'(z^1) \times w^2 \times A'(z^2) \times w^3 \times A'(z^3) \times w^4 \times A'(z^4) \times \frac{\partial J}{\partial a^4}$$

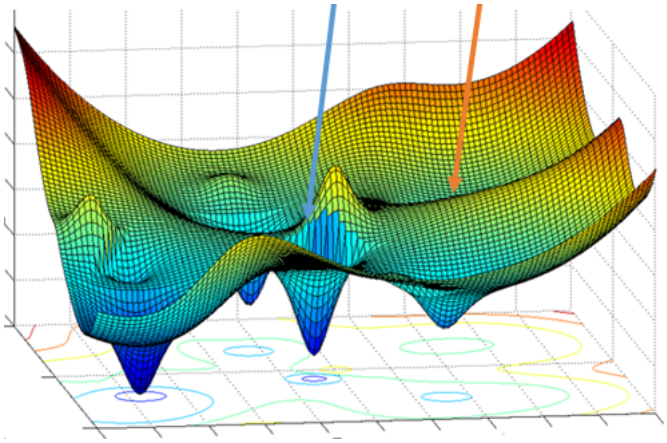
ReLU函数还有其他的许多**好处**：

- ReLU函数通过分段使得函数本身仍然是非线性的，不会减弱神经网络的表现能力
- 小于0时梯度为0使得部分神经元不会被激活，使得神经网络更加稀疏和高效
- ReLU运算简单，比sigmoid函数更节约计算开销

ReLU函数同样也存在**缺点**，主要是由小于0的分段梯度同样为0造成，这意味着输出结果为负的神经元将停止对误差的响应，可能会导致许多神经元的"**死亡**"，通过使用ReLU函数的变体PReLU（Parametric Rectified Linear Unit）来解决。

参数初始化

除了深度网络存在的梯度不稳定的情况之外，还有许多其他的问题影响着网络的训练。比如参数初始化使得网络处于不同的起点，往往带来截然不同的训练结果。如右图的蓝色箭头的起点比黄色箭头指向的起点更容易陷入局部最优。选择合适的初始化方法可以减少这种情况。



戳这里训练深度神经网络总结及建议，更多的还是善用搜索，通过谷歌，github等开源网站去寻找相关学习资料。

机器学习（入门） 袁烨 主讲

更多数据科学课程，上DC学院

关注DC，获取更多学习资源