深度学习

反向传播算法推导

Author:WangQi
Email:wangqikaixin@gmail.com

反向传播算法(Back propagation, BP)是目前训练ANN最常用最有效的算法。主要思想是利用ANN的输出结果与实际结果（标记）的误差从输出层向隐藏层反向传播，直至传播到第二层（即除了输入层）为止。在反向传播的过程中，根据误差调整参数的值；不断迭代，直至收敛。

Author:WangQi
Email:wangqikaixin@gmail.com

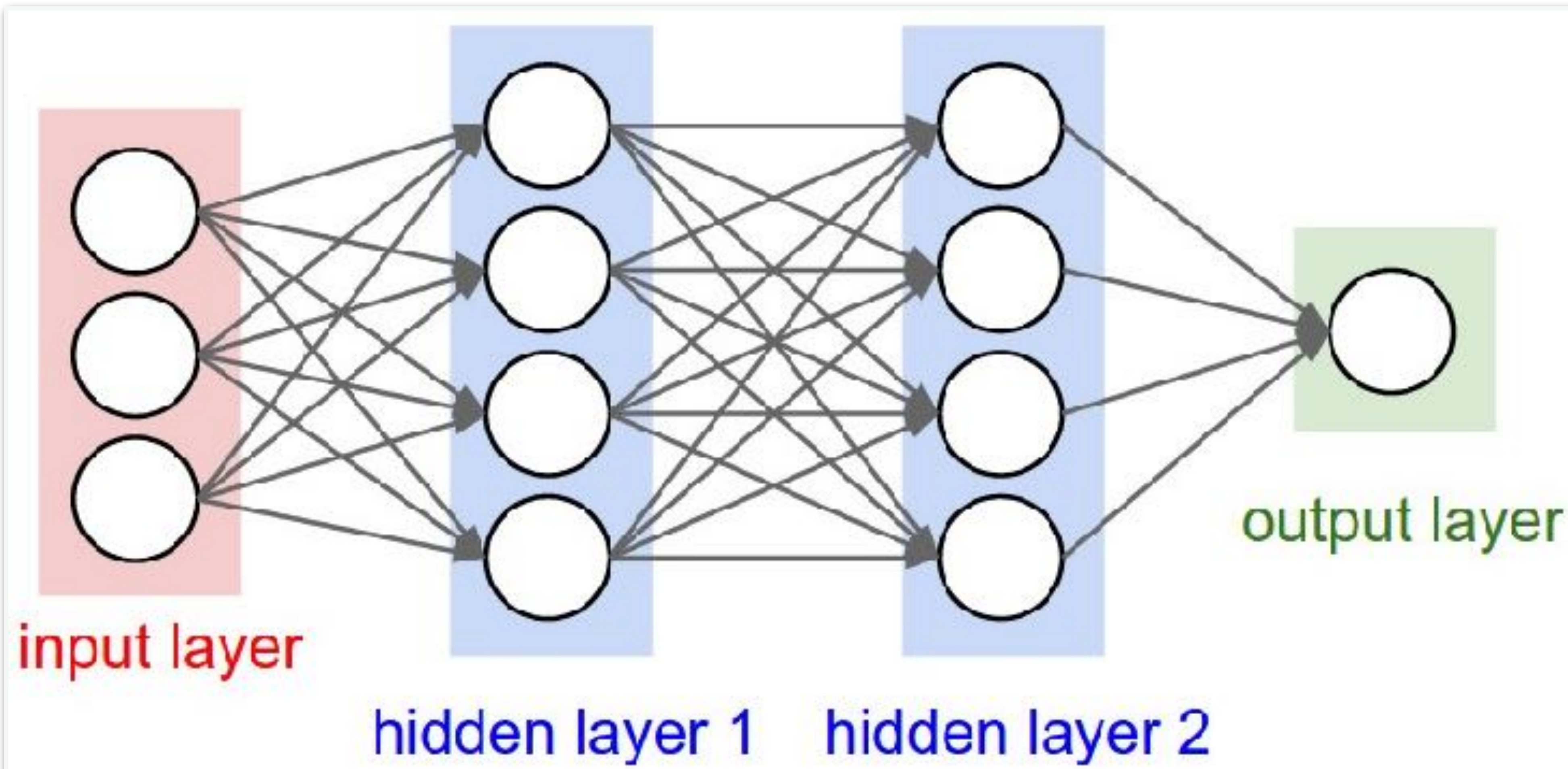
反向传播算法的公式推导

我们以均方误差代价函数为例：

$$J = \frac{1}{2m} \sum_{i=1}^{m} ||y^{(i)} - h_{w,b}^{(i)}(x)||^2$$

Author:WangQi
Email:wangqikaixin@gmail.com

变量定义



我们假设神经网络的神经元均为sigmoid神经元。

Author:WangQi
Email:wangqikaixin@gmail.com

l表示层索引，L表示最后一层的索引。j、k均表示某一层神经元的索引。N表示神经网络每一层的神经元数量。w表示连接权重，b表示偏置值。

$w_{jk}^{(l)}$表示第l层的第j个神经元与第$(l-1)$层的第k个神经元的连接权重。

$b_j^{(l)}$表示第l层的第j个神经元的偏置值。

$N^{(l)}$表示第l层神经元的数量。其中$N^{(L)}$表示最后一层神经元的数量。

$$z_j^{(l)} = \sum_{k=1}^{N^{(l-1)}} w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}$$表示第l层第j个神经元的输入。

$a_j^l = \sigma(z_j^{(l)})$表示第$l$层第$j$个神经元的输出。其中$\sigma$表示$sigmoid$激活函数。

样本数量说明

为避免多个样本带来的上标复杂难记，我们以单样本为例进行推导。

此时的代价函数为：

$$J=\frac{1}{2}||y-h(x)||^2=\frac{1}{2}||y-a^{(L)}||^2=\frac{1}{2}\sum_{j=1}^{N^{(L)}}(y_j-a_j^{(L)})^2$$

Author:WangQi
Email:wangqikaixin@gmail.com

残差定义

我们将第l层第j个神经元产生的残差定义为：

$$\delta_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}}$$

*注意：这里所说的“残差”是指观测值与预测值之间的差，其中隐藏层“残差”表示“输入错误”或“对输入求导”。计算残差的意义在于决定每一层神经元的状态的参数*w*，*b*均在输入*z*中。在深度学习中，还有一个地方会提到“残差”——残差神经网络。残差神经网络的残差与此处的残差并不是一种概念。

Author:WangQi
Email:wangqikaixin@gmail.com

残差与梯度公式推导

Author:WangQi
Email:wangqikaixin@gmail.com

输出层残差

$$\because \delta_j^{(L)} = \frac{\partial J}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}$$

$$\therefore \delta^{(L)} = \frac{\partial J}{\partial a^{(L)}} \odot \frac{\partial a^{(L)}}{\partial z^{(L)}}$$

$$\therefore = \nabla_a J \odot \sigma'(z^{(L)})$$

Author:WangQi
Email:wangqikaixin@gmail.com

隐藏层残差

$$\because \delta_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}} = \sum_{k=1}^{N^{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}$$

$$= \sum_{k=1}^{N^{(l+1)}} \delta_k^{(l+1)} \cdot \frac{\partial(w_{kj}^{(l+1)} a_j^{(l)} + b_k^{(l+1)})}{\partial a_j^{(l)}} \cdot \sigma'(z_j^{(l)})$$

$$= \sum_{k=1}^{N^{(L+1)}} \delta_k^{(l+1)} \cdot w_{kj}^{(l+1)} \cdot \sigma'(z_j^{(l)})$$

$$\therefore \delta^{(l)} = ((w^{(l+1)})^T \delta^{(l+1)}) \odot \sigma'(z_j^{(l)})$$

Author:WangQi
Email:wangqikaixin@gmail.com

连接权重的梯度

$$\frac{\partial J}{\partial w_{jk}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}}$$

$$= \delta_j^{(l)} \cdot \frac{\partial (w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)})}{\partial w_{jk}^{(l)}}$$

$$= a_k^{(l-1)} \delta_j^{(l)}$$

Author:WangQi
Email:wangqikaixin@gmail.com

偏置梯度

$$\frac{\partial J}{\partial b_j^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} \cdot \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}}$$

$$= \delta_j^{(l)} \cdot \frac{\partial (w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)})}{\partial b_j^{(l)}}$$

$$= \delta_j^{(l)}$$

Author:WangQi
Email:wangqikaixin@gmail.com

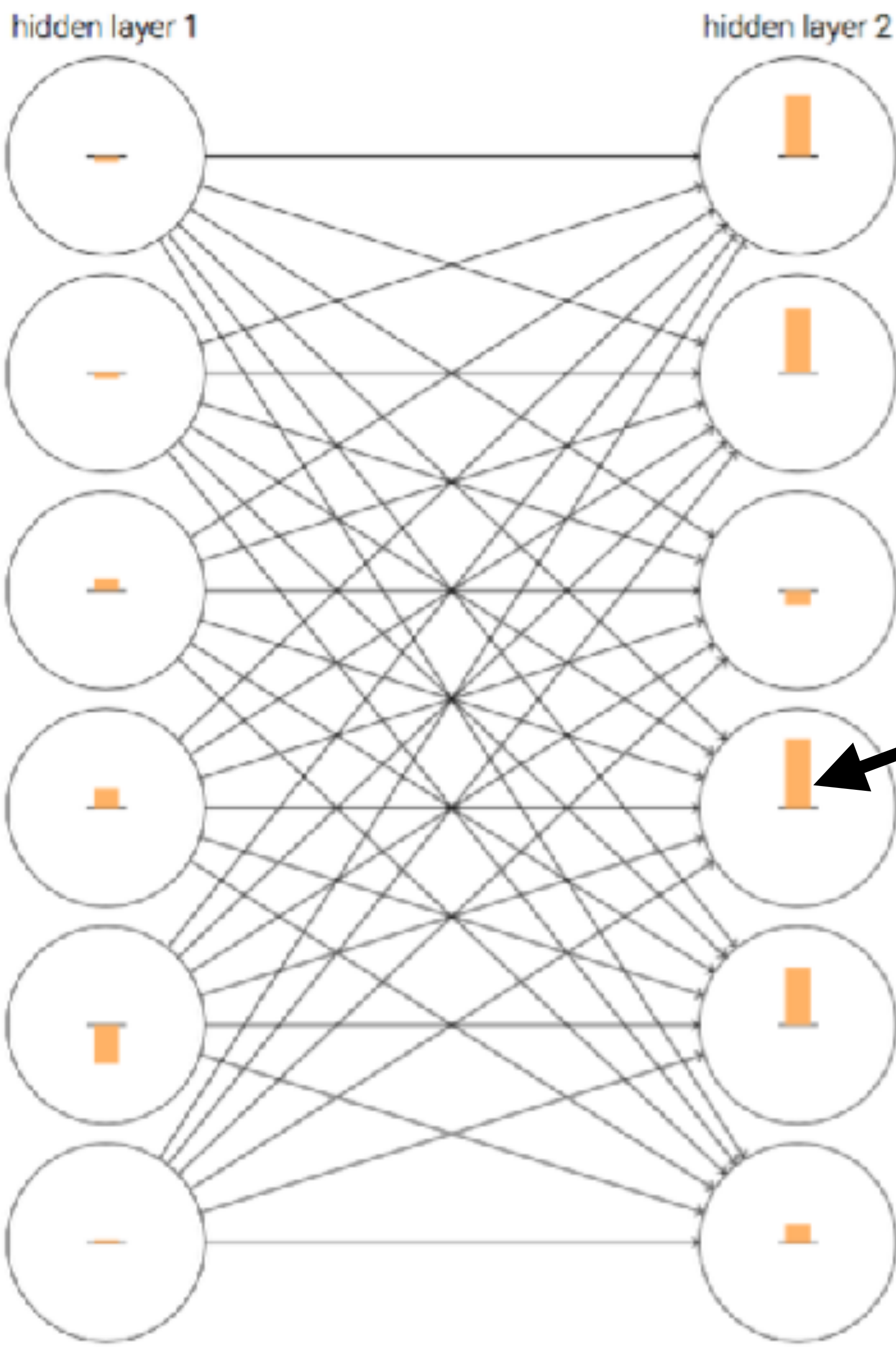
参数更新规则

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$

$$b_{i}^{(l)} = b_{i}^{(l)} - \alpha \frac{\partial}{\partial b_{i}^{(l)}} J(W, b)$$

Author:WangQi
Email:wangqikaixin@gmail.com

梯度消失



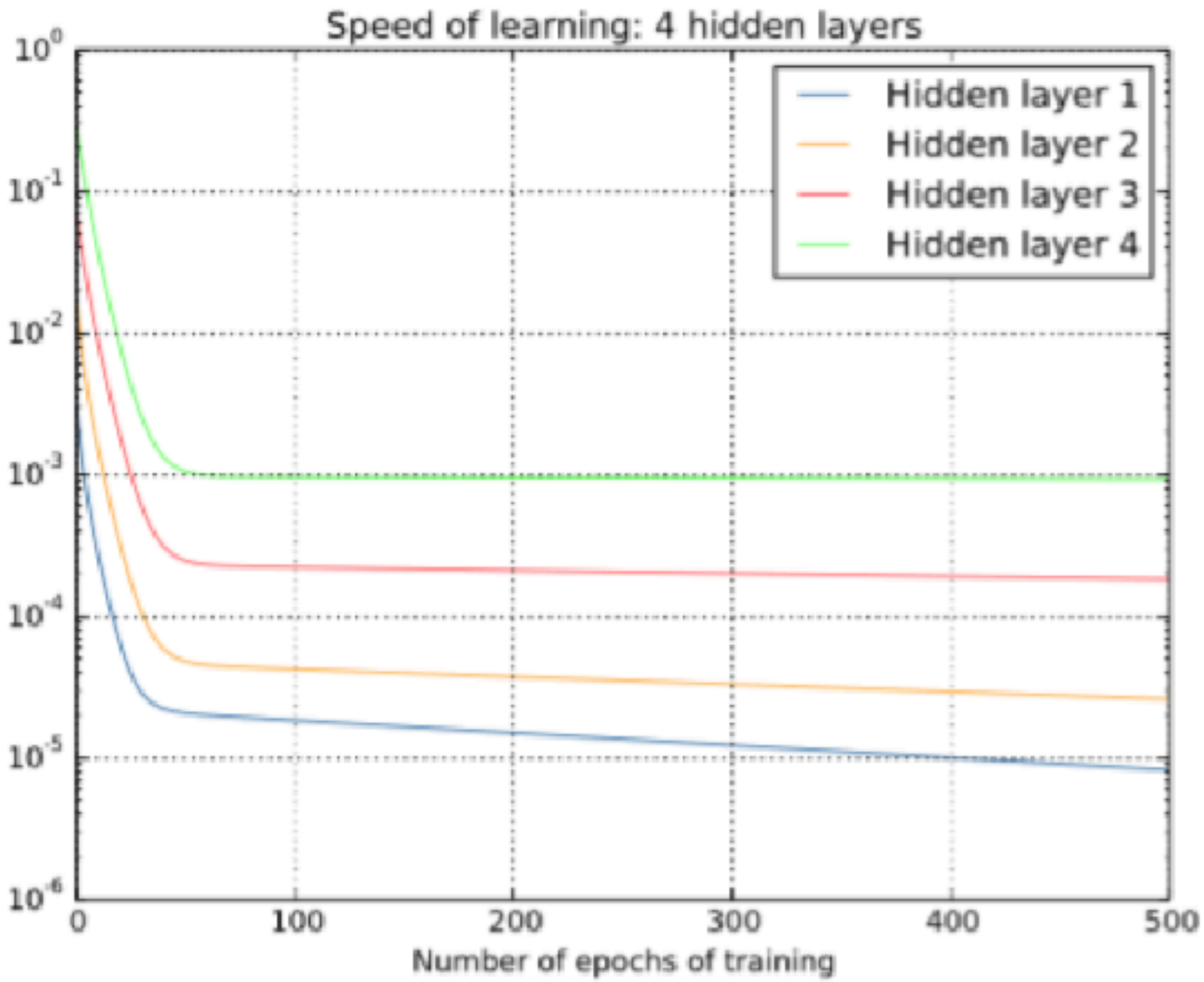
一个网络在初始化之后
在训练初期的可视化如图

黄色的柱状图可以
理解为学习速率

可以看到：第一个隐藏层的学
习速率远低于第二个隐藏层

Author:WangQi
Email:wangqikaixin@gmail.com

梯度消失与梯度爆炸



可以看到：离输出层越远的层，学习速率越小。而且这并非是一个特例，是普遍存在于神经网络中的现象。梯度消失导致了在运行梯度下降时，离输出层较远的层几乎无法更新参数，这也是神经网络一度无人问津的原因之一。

如果神经网络的初识学习率设置过高，还会带来梯度爆炸的问题，即越接近输入层，梯度越大。这也会导致神经网络无法训练。

一个具有四个隐层的神经网络，各隐藏层的学习速率曲线。

Author:WangQi
Email:wangqikaixin@gmail.com

梯度不稳定的缘由

举例：

$$0.1\times0.1\times0.1\times0.1\times0.1=0.00001$$

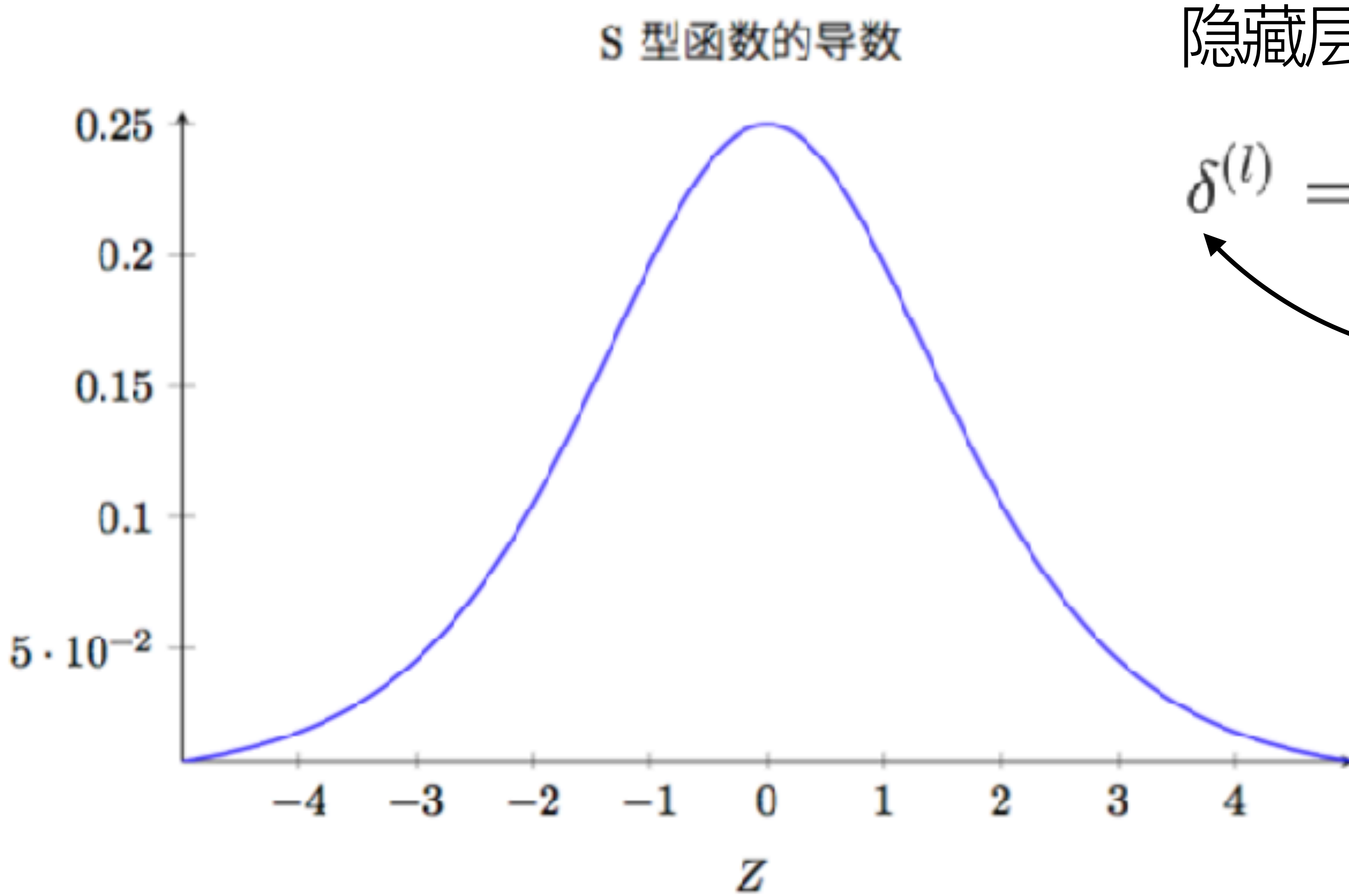
$$9\times9\times9\times9\times9=59049$$

小于1的数累乘会导致数越来越小。
大于1的数累乘会导致数越来越大。

梯度不稳定缘由：当前层的梯度由后面所以层的梯度累乘而来，当存在过多层，就出现了梯度不稳定问题。

Author:WangQi
Email:wangqikaixin@gmail.com

sigmoid导函数



隐藏层残差:

$$\delta^{(l)} = ((w^{(l+1)})^T \delta^{(l+1)}) \odot \sigma'(z_j^{(l)})$$

最大值为0.4

sigmoid导函数的极大值为1/4，随着神经网络层数加深，每一层的残差都会乘以1个小于等于1/4的值，最终导致梯度度呈指数级下降（前提是w的值≤1）。同样的如果我们设置的学习速率与w过高，可能导致梯度爆炸。

其最大值为0.25。累乘会导致梯度消失。

Author:WangQi
Email:wangqikaixin@gmail.com

梯度不稳定的解决方案

- 初始化参数时，设置合理的权重。不要设置过大或过小的权重。
- 使用Tanh或者ReLU激活函数。ReLU的梯度不是0，就是1，是最常用的激活函数之一。
- 梯度裁剪。例如：新梯度=梯度 * 阈值 /max(阈值, 梯度L2范数)，可以在一定程度上避免梯度爆炸。

Author:WangQi
Email:wangqikaixin@gmail.com

均方误差代价函数的缺点

$$J = \frac{1}{2m}\sum_{i=1}^{m}||y^{(i)} - h_{w,b}^{(i)}(x)||^2$$

- 可能导致（输出层）梯度不稳定。
- 可能会使ANN的代价函数成为非凸函数。
- 不太适合分类问题。不能很好地表现与标签概率分布的差异。

Author:WangQi
Email:wangqikaixin@gmail.com

小节

- 反向传播算法需要分别对隐藏层与输出层求残差。但不需要对输入层求残差。
- 利用反向传播算法在求偏导时是首先对全部参数求偏导，最好再执行参数更新。
- 梯度消失与梯度爆炸本质上是由求梯度导致的，我们可以通过改变激活函数或限制梯度等方式缓解。
- 均方误差代价函数并不一定是合适的代价函数。

Author:WangQi
Email:wangqikaixin@gmail.com

THANKS

Author:WangQi
Email:wangqikaixin@gmail.com