深度学习

反向传播算法推导

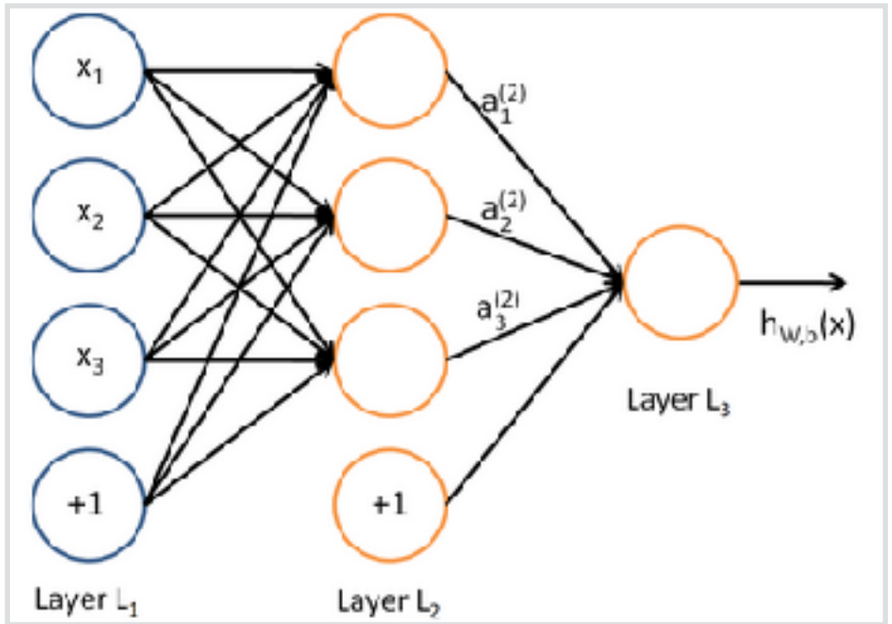
Author:WangQi
Email:wangqikaixin@gmail.com

概览

1. 复合函数求导。
2. 反向传播算法。
3. 梯度消失与梯度爆炸问题。
4. Python实现反向传播算法。

1. 复合函数求导

ANN与复合函数



$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

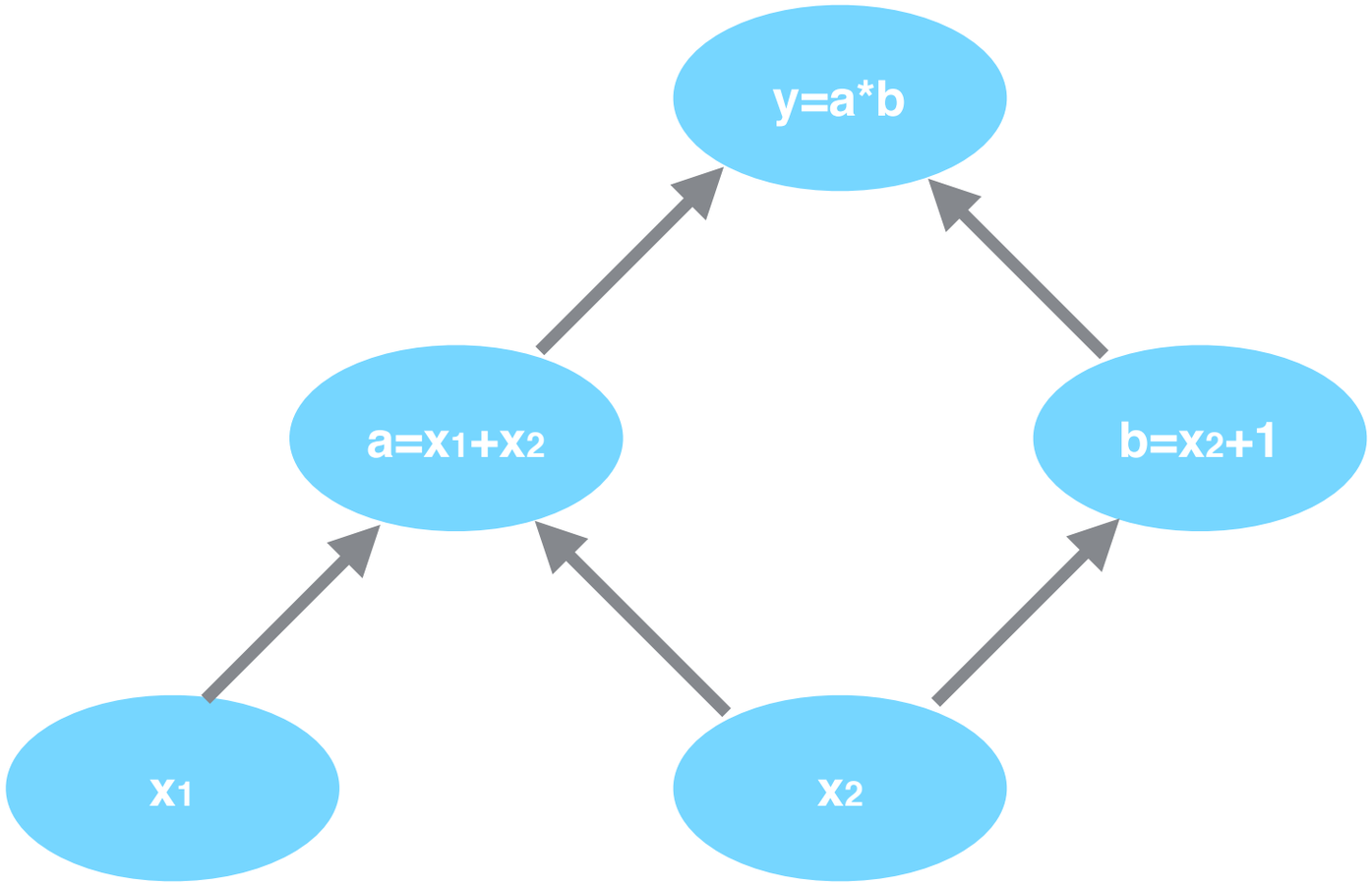
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

$$h_{w,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

复合函数求导

以求 $y=(x_1+x_2)\times(x_2+1)$ 的偏导数为例



对每条通路求偏导

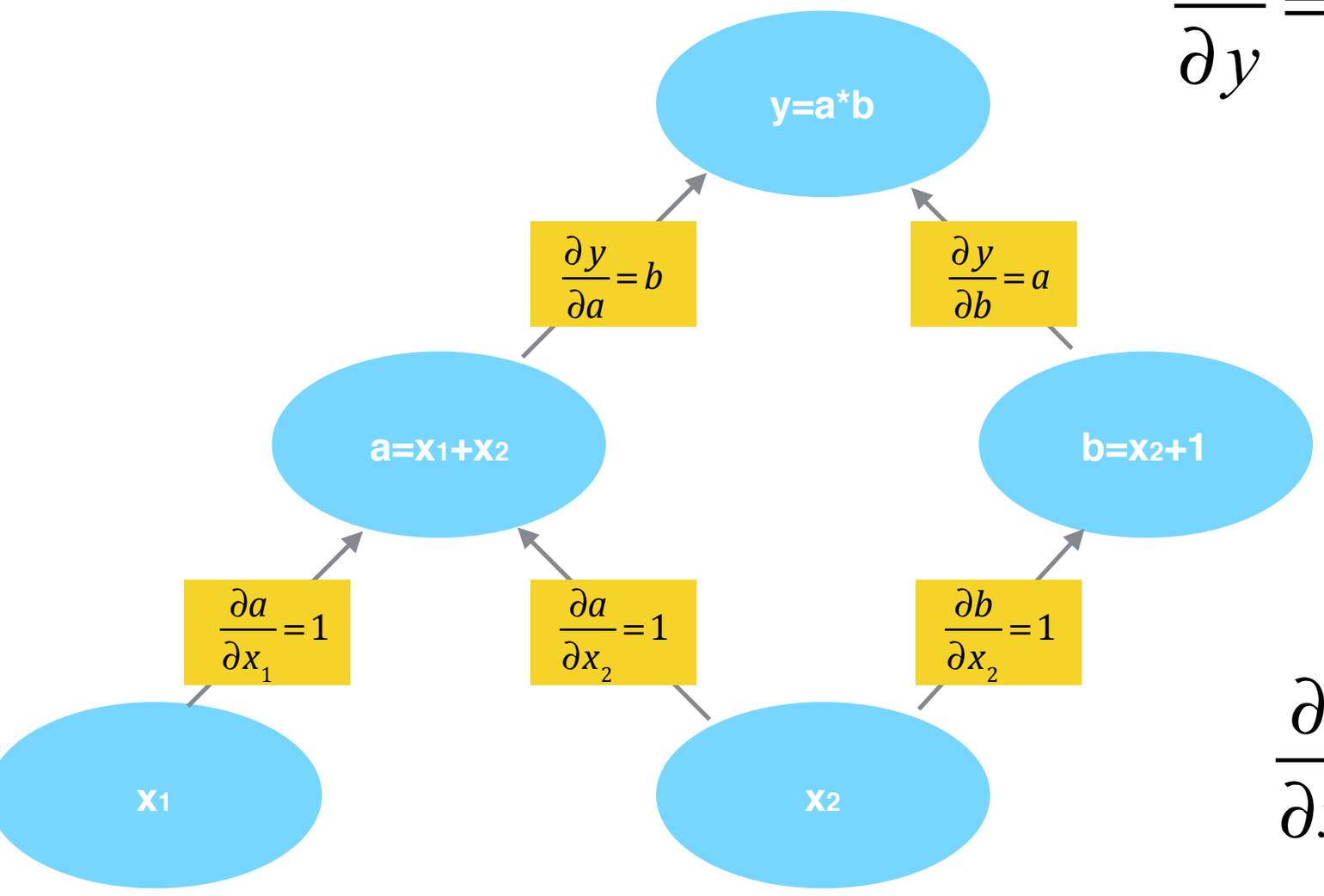
$$\frac{\partial y}{\partial x_1}=\frac{\partial y}{\partial a}\cdot\frac{\partial a}{\partial x_1}$$

$$\frac{\partial y}{\partial x_2}=\frac{\partial y}{\partial a}\cdot\frac{\partial a}{\partial x_2}+\frac{\partial y}{\partial b}\cdot\frac{\partial b}{\partial x_2}$$

重复计算了对a的偏导

Author:WangQi
Email:wangqikaixin@gmail.com

复合函数求导——技巧



$$\frac{\partial y}{\partial y}=1 \qquad \frac{\partial y}{\partial a}=b*1=b$$

$$\frac{\partial y}{\partial b}=a*1=a$$

$$\frac{\partial y}{\partial x_1}=b\cdot 1=b$$

$$\frac{\partial y}{\partial x_2}=b\cdot 1+a*1=b+a$$

Author:WangQi
Email:wangqikaixin@gmail.com

复合函数求导——技巧

1. 根据计算流程画出结构图并保存中间变量的值。

2. 从顶点开始每一层对下一层节点求偏导（顶点的偏导为1）。

3. 从顶点之下开始，每一层每一个节点的偏导等于其与之直接相连的上一层节点的乘积的和。

反向传播算法正是使用了此技巧。

Author:WangQi
Email:wangqikaixin@gmail.com

题：画出下列复合函数的结构图，并利用“技巧”求偏导。

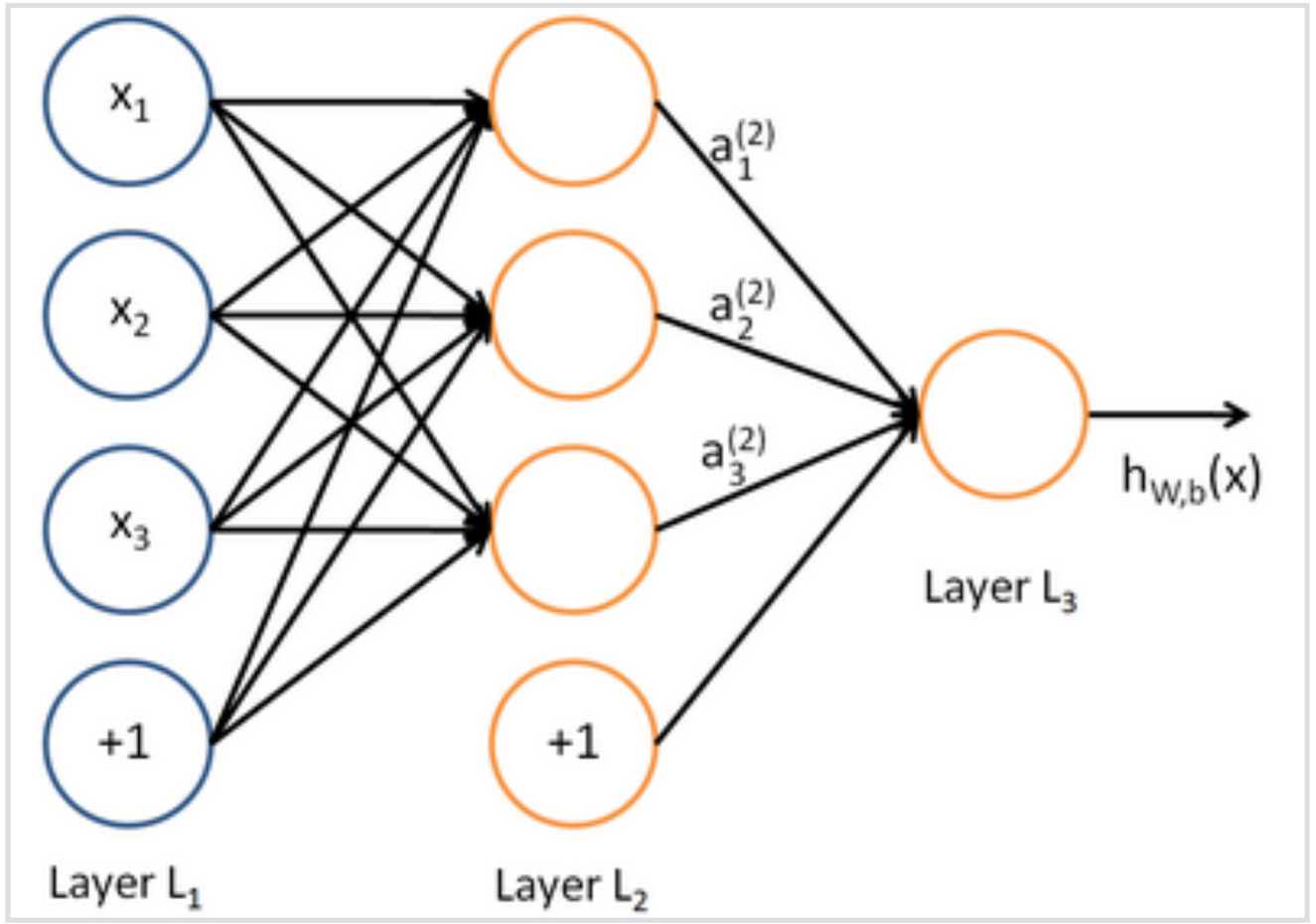
$$y = x_1^2 + (x_1 + x_2)^2$$

$$y = (x_1 + x_2)^2 x_2^2 + x_2$$

了解图与自动微分法的关系

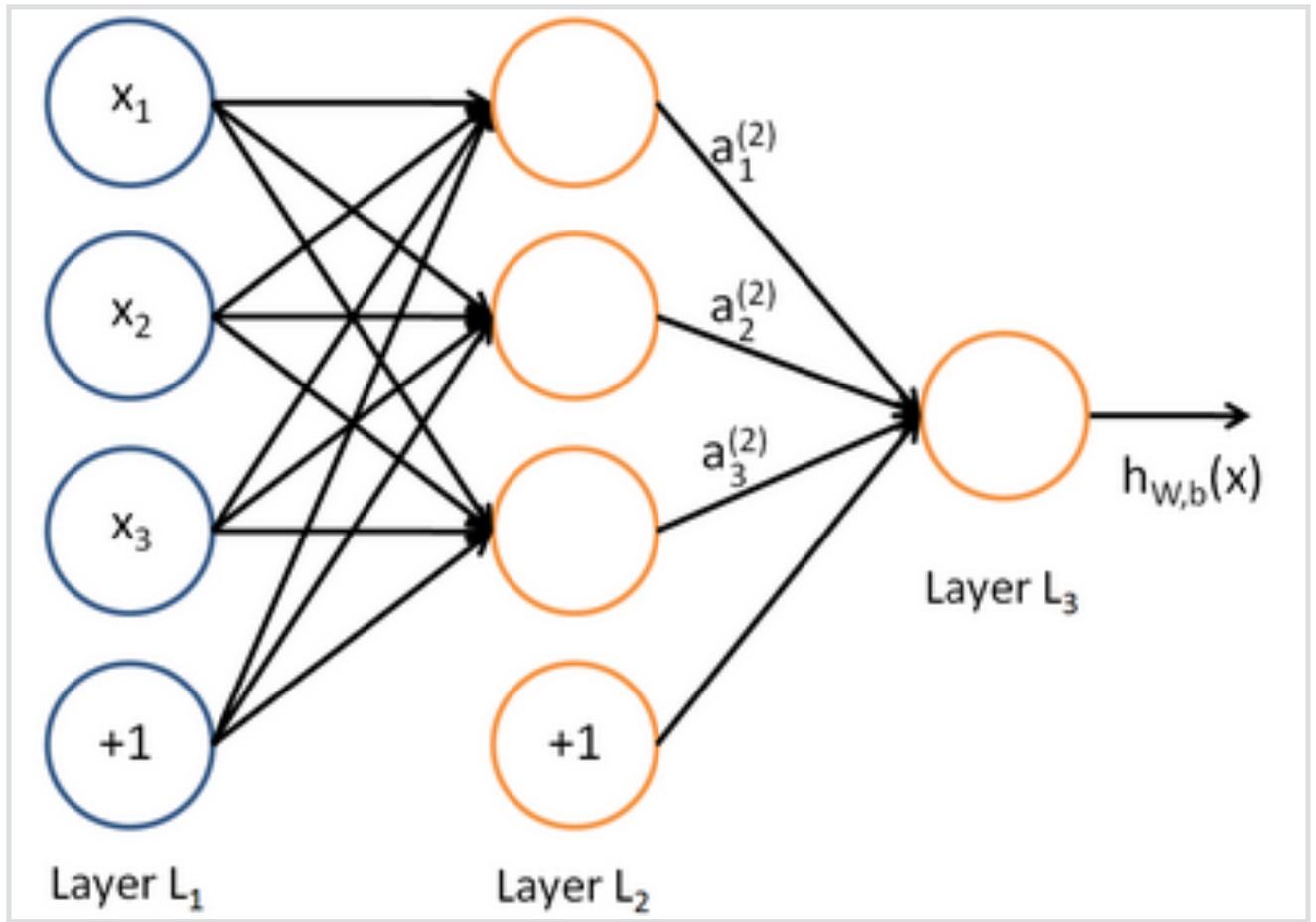
2. 反向传播算法公式推导

利用符合函数求导技巧



从输出层到第一个隐藏层的每一层节点依次求偏导。

ANN的特殊结构



从隐藏层到输出层，每一层结构均是类似的。

ANN的特殊结构

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

隐藏层表达式

$$h_{w,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

输出层表达式

层内表达式形式一致，层与层间表达式形式一致。

变量定义

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。

残差与优化目标

我们将第l层第j个神经元产生的残差定义为：$\delta_j^{(l)} = \frac{\partial J}{\partial z_j^{(l)}}$

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

为避免多个样本带来的上标复杂难记，以单样本为例进行推导。

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

Author:WangQi
Email:wangqikaixin@gmail.com

梯度公式推导

Author:WangQi
Email:wangqikaixin@gmail.com

输出层残差

输出层残差： 说明：激活函数为logistic

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

推导过程如下：

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

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

⊙表示Hadamard乘积，用于矩阵或向量之间点对点的乘法运算

Author:WangQi
Email:wangqikaixin@gmail.com

隐藏层残差

推导过程如下：

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

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

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

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

Author:WangQi
Email:wangqikaixin@gmail.com

连接权重的梯度

推导过程如下：

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

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

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

Author:WangQi
Email:wangqikaixin@gmail.com

偏置梯度

推导过程如下：

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

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

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

Author:WangQi
Email:wangqikaixin@gmail.com

参数更新规则

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

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

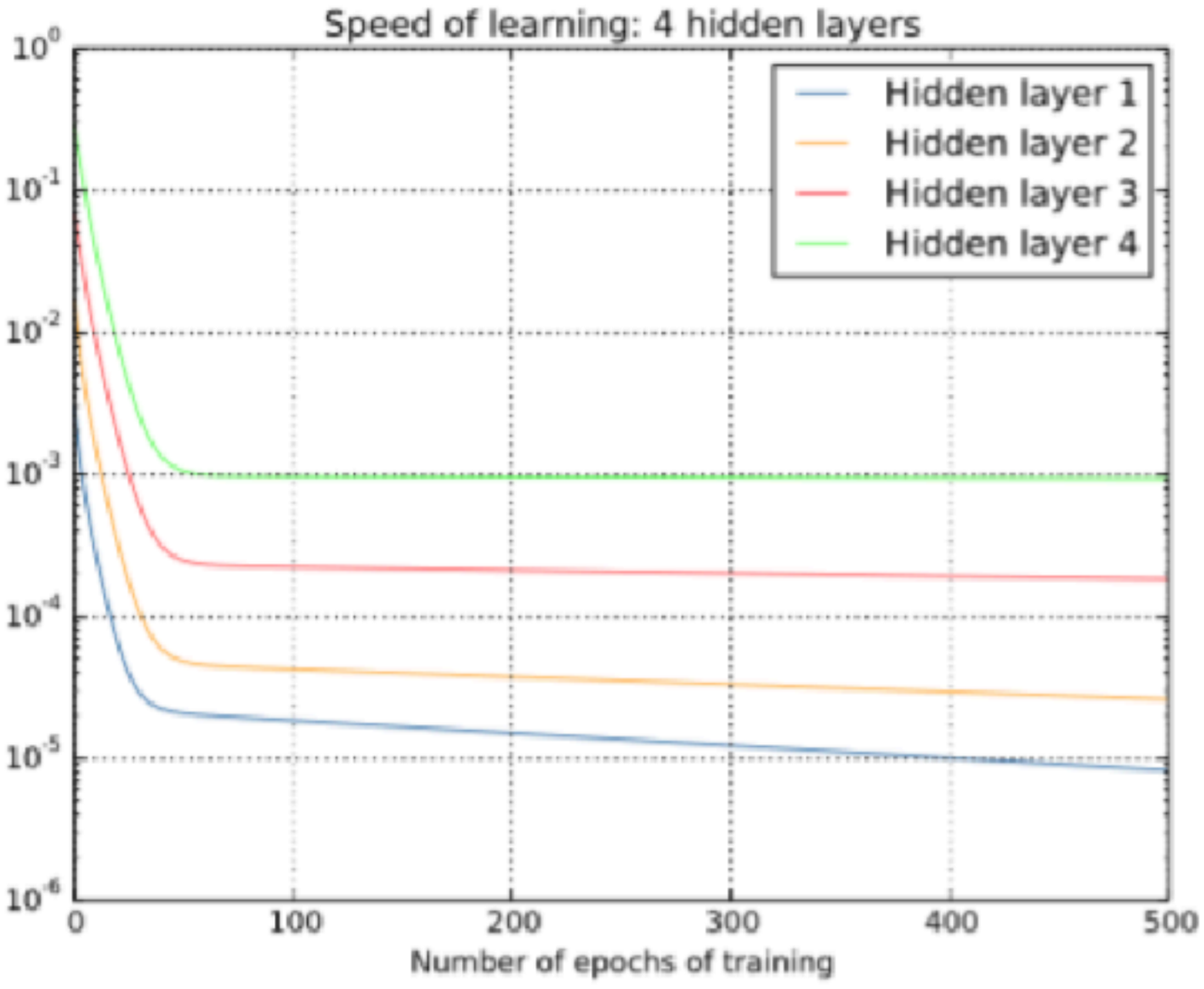
Author:WangQi
Email:wangqikaixin@gmail.com

3. 梯度消失与梯度爆炸

梯度消失

梯度消失问题（vanishing gradient problem），即在使用梯度更新参数时，随着神经网络层数的加深，前面的层的梯度的值越来越小，甚至趋于0，使得前面的层的参数难以得到更新。

梯度消失



一个具有四个隐层的神经网络，各隐藏层的学习速率曲线。可以看到：离输出层越远，梯度值越小。

Author:WangQi
Email:wangqikaixin@gmail.com

梯度爆炸

梯度爆炸问题（exploding gradient problem），即在使用梯度更新参数时，随着神经网络层数的加深，前面的层的梯度的值越来越大，甚至趋于无穷大，使得前面的层的参数无法收敛。

梯度不稳定的原因

隐藏层残差公式： $\delta^{(1)} = (w^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(z^{(1)})$

可以看到：当前层残差由其后所有层残差“累乘”所得。

例如：第一隐藏层的残差为：

$$\begin{aligned}\delta^{(1)} &= (w^{(2)})^T \delta^{(2)} \odot \sigma'(z^{(1)}) \\ &= (w^{(2)})^T ((w^{(3)})^T \delta^{(3)} \odot \sigma'(z^{(2)})) \odot \sigma'(z^{(1)}) \\ &= (w^{(2)})^T ((w^{(3)})^T \ldots \odot \sigma'(z^{(2)})) \odot \sigma'(z^{(1)})\end{aligned}$$

梯度不稳定的原因

隐藏层残差公式： $\delta^{(1)} = (w^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(z^{(1)})$

当w与 σ'的值均小于1时，累乘的值趋于0。

当w与 σ'的值均大于1时，累乘的值趋于无穷。

例如：

$$0.1 \times 0.1 \times 0.1 \times 0.1 \times 0.1 = 0.00001$$

$$9 \times 9 \times 9 \times 9 \times 9 = 59049$$

梯度不稳定的原因

隐藏层残差公式： $\delta^{(1)} = (w^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(z^{(1)})$

初始化w时，通常会将w的值初始化在 $|w| = 1$的附近。

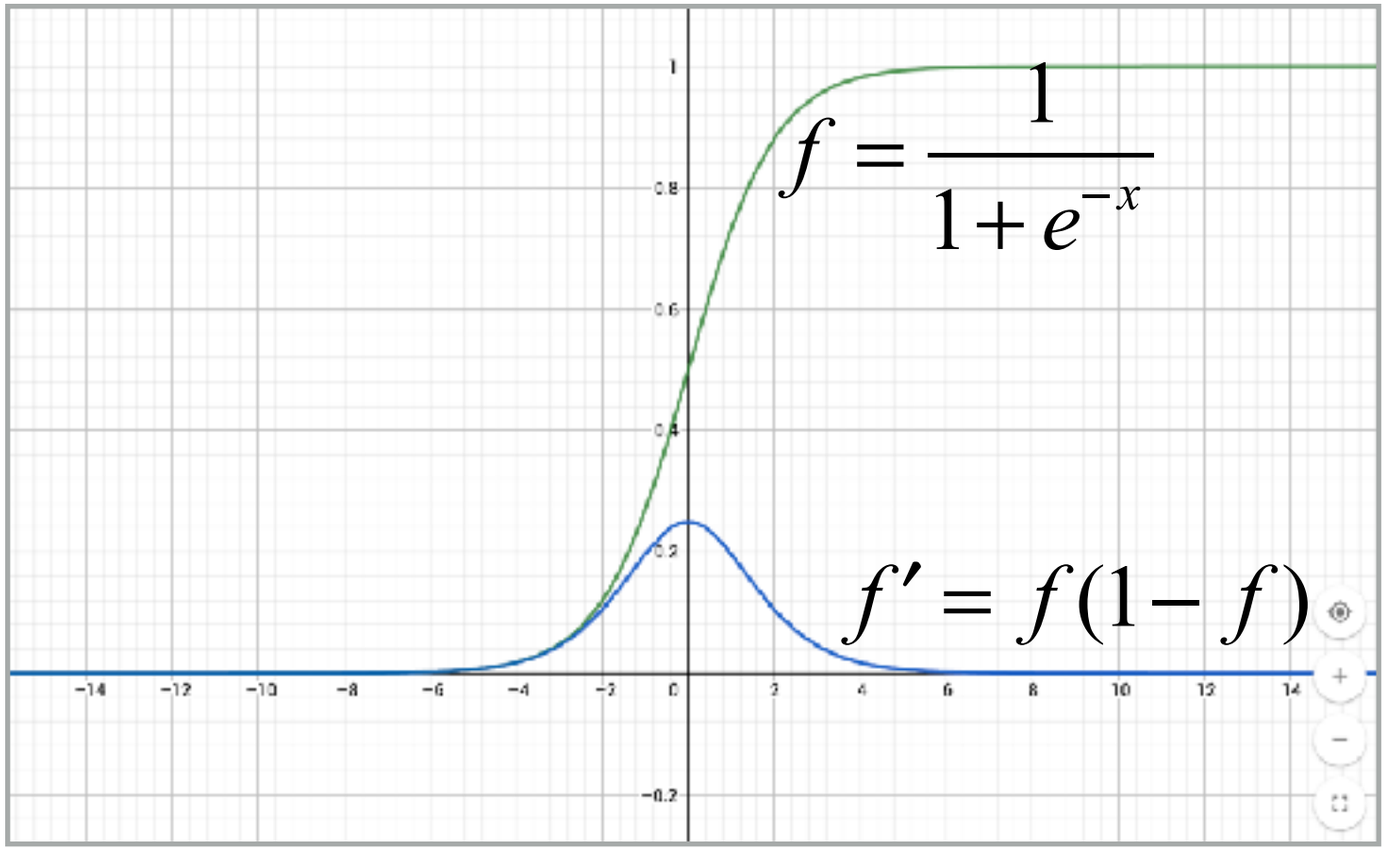
当w初始化的值太大时，会导致梯度爆炸。

当σ'初始化的值太小时，会导致梯度消失。

注意：学习率过大也会导致梯度爆炸，学习率过小也会导致梯度消失。

梯度不稳定的原因

logistic函数与其导函数图像



logistic导函数性质

值域：(0, 0.25]

值域大部分范围趋于0

根本原因：求解梯度时多项式累乘的值不稳定。

梯度不稳定的解决方法

1. 使用合适的参数初始化方法。例如使用服从标准正态分布的随机数初始化。

2. 使用合适的学习率。

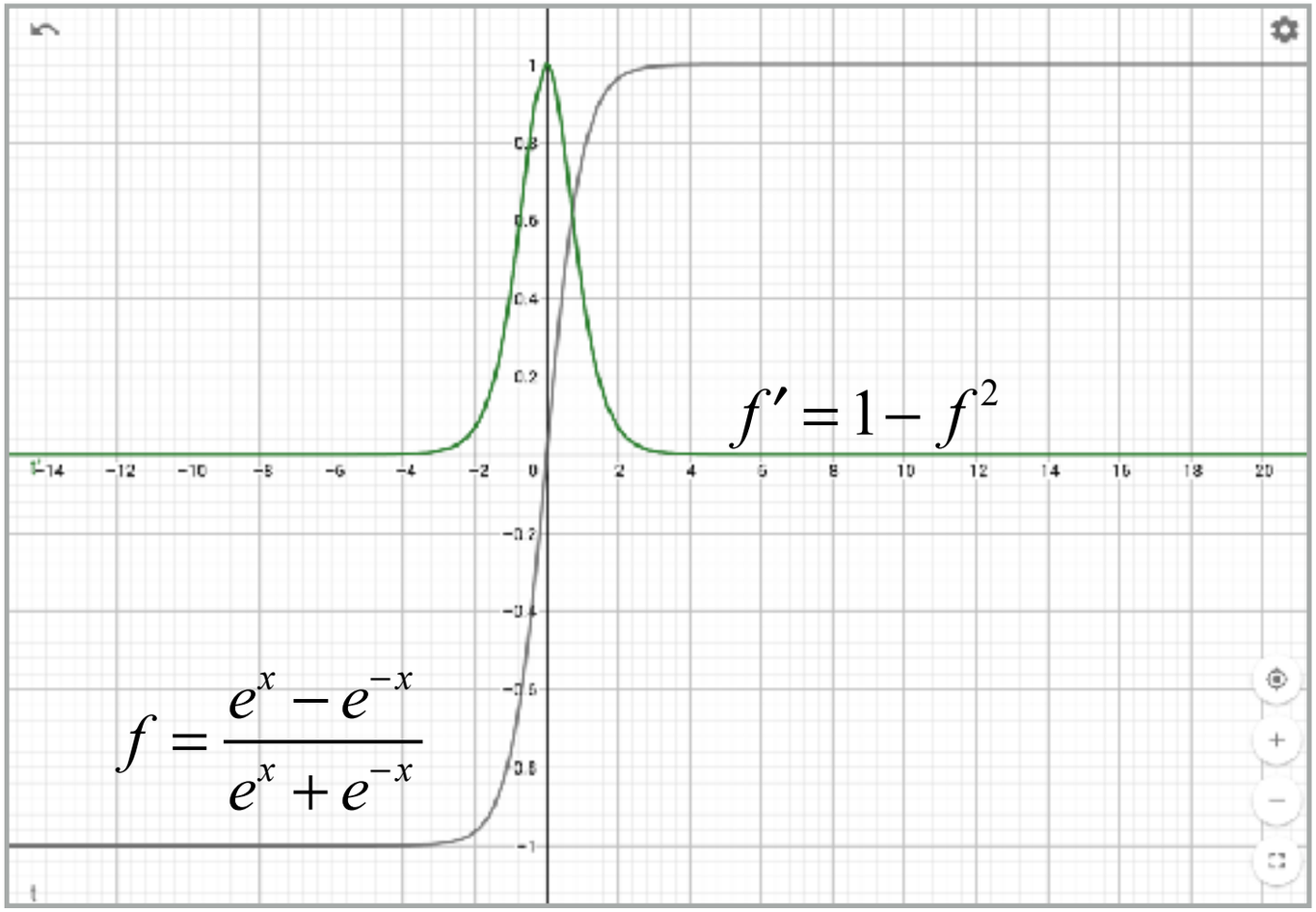
3. 使用更好的激活函数。

4. 梯度裁剪，对过大的梯度进行限制。

5. 使用批规范化（Batch Normalization, BN）技术。

更好的激活函数

tanh函数与其导函数图像

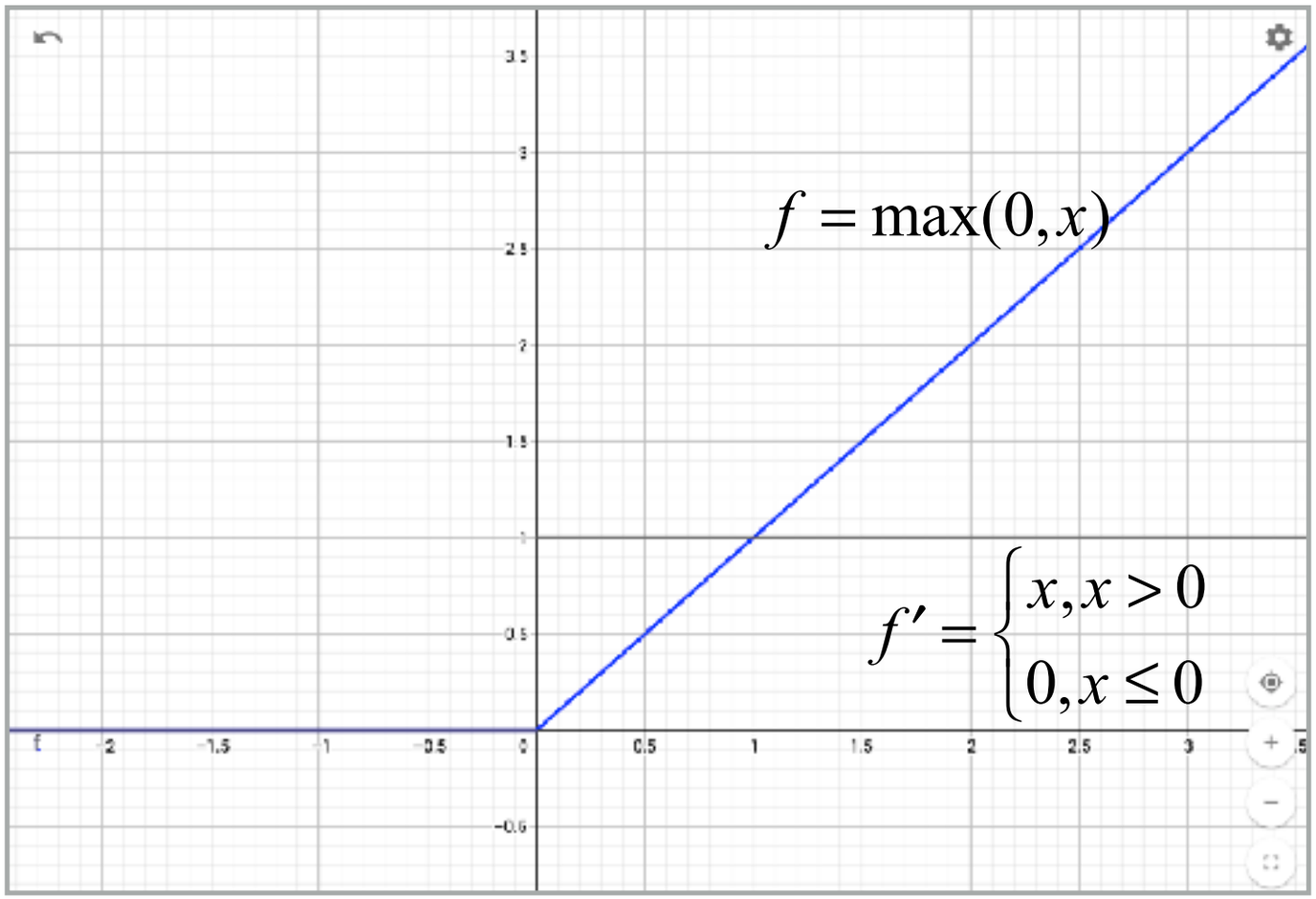


tanh导函数性质

值域：(0, 1]

更好的激活函数

ReLU函数与其导函数图像



当输入大于0时，导数为1。
否则导数为0。

4. python实现反向传播算法

小节

- 复合函数求导法则。
- 反向传播算法的原理与公式推导过程。
- 不同激活函数的导数公式。
- 梯度不稳定的原因与解决方法。
- Python实现反向传播算法。

Author:WangQi
Email:wangqikaixin@gmail.com

THANKS

Author:WangQi
Email:wangqikaixin@gmail.com