



深度学习技术与应用 (2)

Deep Learning: Techniques and Applications (2)

李 戈

北京大学 信息科学技术学院

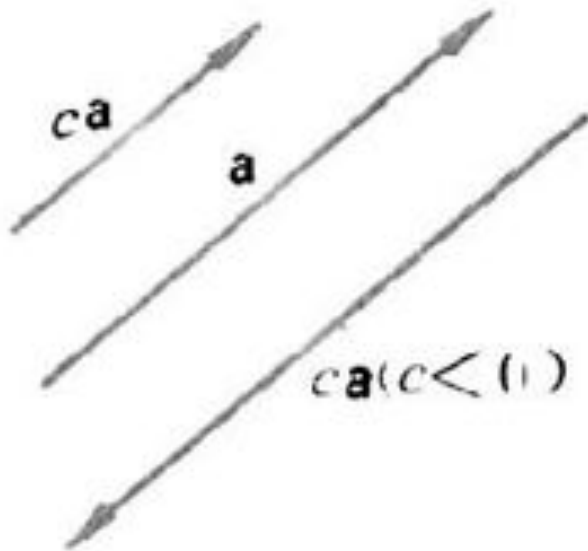
lige@pku.edu.cn

关于向量



■ 向量数乘的意义

◆ 在原向量的直线上向量长度的伸长或缩短;

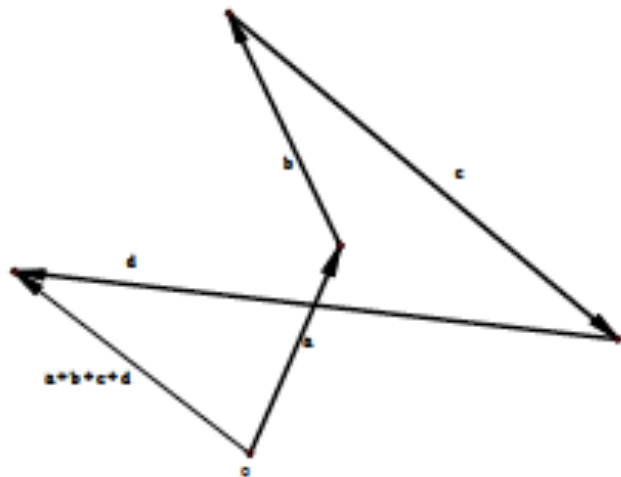
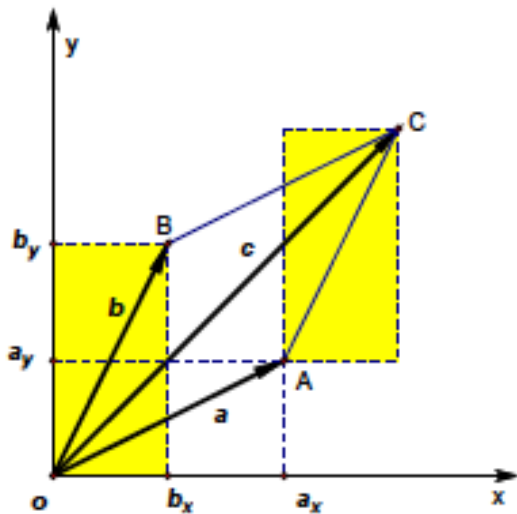


关于向量



■ 向量加法的几何意义

- ◆ 两向量相加的几何解释就是进行依照平行四边形法则对向量合并;
- ◆ 多向量加法的数学本质就是这些向量在坐标轴上的投影的合成结果。

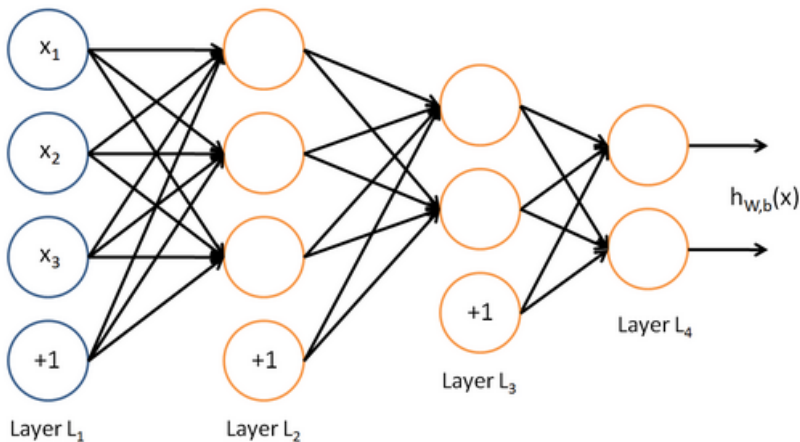


关于向量



■ 向量的线性组合的几何意义

- ◆ 向量的“线性组合”的几何意义就是对向量组内的向量长度进行缩放后依照平行四边形法则进行合并加；
- ◆ “线性”表示的几何意义就是可以把一个向量依照平行四边形法则分解(或投影)为向量组上的和；



关于向量



■ 向量的点积的几何意义

- ◆ 定义有两个，它们是一样的含义（用余玄定理可以证明）

$$\mathbf{a} \cdot \mathbf{b} = ab \cos \theta \dots\dots\dots 1$$

$$\mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z \dots\dots\dots 2$$

- ◆ 向量内积的几何含义：一个向量在另一个向量上的投影的积，它反应了向量之间在方向上的“接近程度”
- ◆ 可知：两个向量内积为正，说明方向相近（夹角小于90度），内积为负，说明方向相反（夹角大于90度），内积为零，说明方向垂直。

关于向量



■ 向量的叉积的几何意义

- ◆ 定义有两个，它们是一样的含义（用余玄定理可以证明）

$$\mathbf{a} \times \mathbf{b} = (a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x) \dots\dots\dots 1$$

$$\mathbf{a} \times \mathbf{b} = ab \sin \theta \mathbf{N} \text{ (其中 } \mathbf{N} \text{ 是垂直于 } \mathbf{a} \text{ 和 } \mathbf{b} \text{ 展成的平面的单位向量) } \dots\dots 2$$

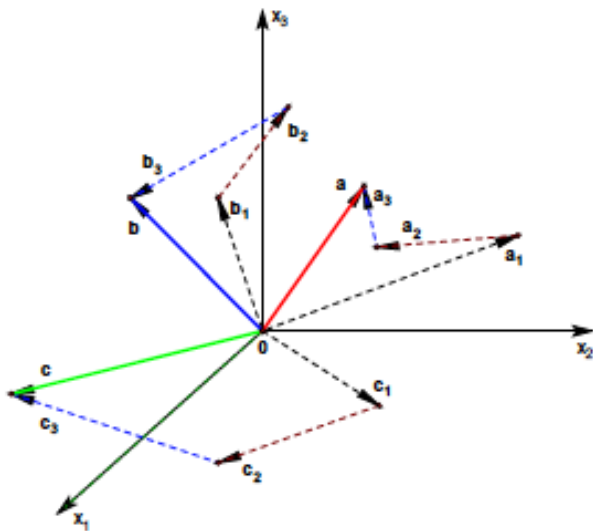
- ◆ 向量叉积的物理意义：两个向量的叉积是把前一个向量旋转至后一个向量所需要的力矩。
- ◆ 力矩在物理学里是指作用力使物体绕着转动轴或支点转动的趋向。力矩能够使物体改变其旋转运动。推挤或拖拉涉及到作用力，而扭转则涉及到力矩。

关于矩阵



■ 矩阵加法的几何意义

- ◆ 矩阵可以看作多个向量的集合
- ◆ 矩阵加法可以看作相对应的行向量或列向量的加法。



关于矩阵



■ 矩阵与向量的乘法

◆ 形式上的定义：

$$\mathbf{A} \cdot \mathbf{c} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} (a_1 \ a_2 \ a_3) \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \\ (b_1 \ b_2 \ b_3) \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a_1 c_1 + a_2 c_2 + a_3 c_3 \\ b_1 c_1 + b_2 c_2 + b_3 c_3 \end{pmatrix}$$

$$\mathbf{A} \cdot \mathbf{c} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} c_1 + \begin{pmatrix} a_2 \\ b_2 \end{pmatrix} c_2 + \begin{pmatrix} a_3 \\ b_3 \end{pmatrix} c_3 = \begin{pmatrix} a_1 c_1 + a_2 c_2 + a_3 c_3 \\ b_1 c_1 + b_2 c_2 + b_3 c_3 \end{pmatrix}$$

关于矩阵



■ 矩阵与向量的乘法

$$\mathbf{A} \cdot \mathbf{c} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} a_1 \\ b_1 \end{pmatrix} c_1 + \begin{pmatrix} a_2 \\ b_2 \end{pmatrix} c_2 + \begin{pmatrix} a_3 \\ b_3 \end{pmatrix} c_3 = \begin{pmatrix} a_1 c_1 + a_2 c_2 + a_3 c_3 \\ b_1 c_1 + b_2 c_2 + b_3 c_3 \end{pmatrix}$$

- ◆ 看做向量对矩阵的操作：
- ◆ 把矩阵A看做两个列向量，Ac的乘积可以理解为矩阵A的列向量的线性组合，组合系数是向量c的三个分量。
- ◆ 几何解释就是：把矩阵 **A** 的列向量进行伸缩变换(比例变换,也可能改变方向)后首尾相连(即向量的和)得到了一个新向量,这个向量就是 **A · d**

关于矩阵



■ 矩阵与向量的乘法

$$\mathbf{A} \cdot \mathbf{c} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} (a_1 \ a_2 \ a_3) \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \\ (b_1 \ b_2 \ b_3) \cdot \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a_1 c_1 + a_2 c_2 + a_3 c_3 \\ b_1 c_1 + b_2 c_2 + b_3 c_3 \end{pmatrix}$$

- ◆ 看做矩阵对向量的操作：
- ◆ 把矩阵A看作两个行向量，矩阵与向量乘积比如 $\mathbf{A}\mathbf{x}$ 表现为矩阵 A 对一个向量 \mathbf{x} 作用的结果。其作用的主要过程是对一个向量 进行旋转和缩放的综合过程(即线性变换的过程),一个向量就变换为另外一个向量。
- ◆ 一个 m 行 n 列的实矩阵 $\mathbf{A}_{m \times n}$ 就是一个 $\mathbb{R}^n \rightarrow \mathbb{R}^m$ 上的线性变换，把一个 n 维空间的 n 维向量变换为一个 m 维空间的 m 维向量。

关于矩阵



■ 矩阵与矩阵之间的相乘

◆ 矩阵与矩阵的乘法可以从矩阵与向量的乘法得到,因为一个矩阵与多个向量相乘,这多个向量就可以组成一个矩阵。

◆ $\mathbf{AB} = \mathbf{C}$ 的几何意义：

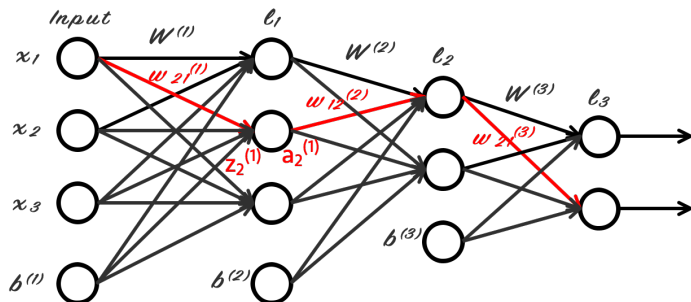
\mathbf{A} 的作用：把矩阵 \mathbf{B} 的数个行向量或列向量构成的几何图形进行旋转、缩放、镜像等变换；

矩阵 \mathbf{C} ：构成的新的几何图形；

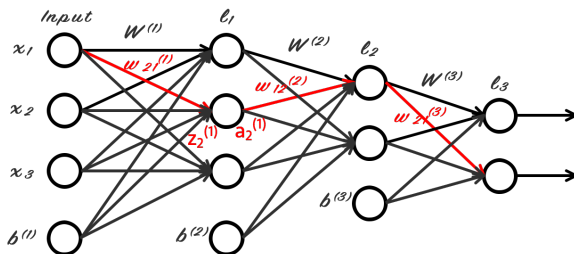
Table of contents

- 1 前向传播的计算
- 2 反向传播的符号体系
- 3 预备知识-多元复合函数求导
- 4 反向传播算法的推导
- 5 反向传播算法总结

前向传播的符号体系



前向传播计算



$$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 + b_1^{(1)}$$

$$z_2^{(1)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3 + b_2^{(1)}$$

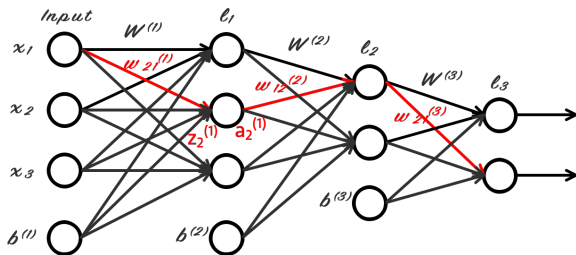
$$z_3^{(1)} = w_{31}^{(1)} x_1 + w_{32}^{(1)} x_2 + w_{33}^{(1)} x_3 + b_3^{(1)}$$

$$a_1^{(1)} = f(z_1^{(1)})$$

$$a_2^{(1)} = f(z_2^{(1)})$$

$$a_3^{(1)} = f(z_3^{(1)})$$

前向传播计算

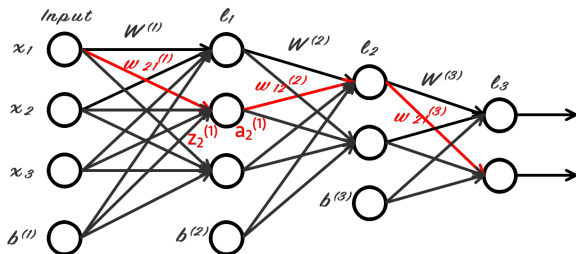


$$z^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix} = W^{(1)}X + b^{(1)}$$

$$a^{(1)} = f(z^{(1)}) = f(W^{(1)}X + b^{(1)})$$

$$a^{(2)} = f(z^{(2)}) = f(W^{(2)}a^{(1)} + b^{(2)})$$

前向传播计算



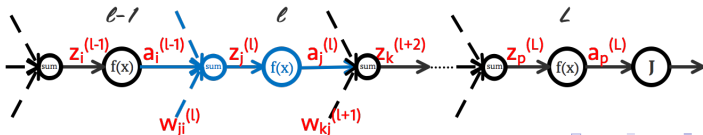
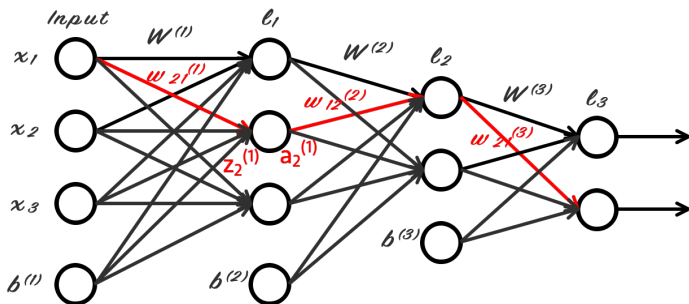
$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \end{bmatrix} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} \end{bmatrix} \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \end{bmatrix} = W^{(2)} a^{(1)} + b^{(2)}$$

总之：

$$z^l = W^{(l)} a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = f(z^{(l)})$$

反向传播符号体系



预备知识-多元复合函数求导

由于接下来的计算中要用到多元符合函数的求导，下面我们先来回顾一下“多元复合函数的求导”的方法：

多元复合函数求导-1

设: $z = f(y_1, y_2, \dots, y_n)$, 其中: $(y_1, y_2, \dots, y_n) \in D_f$ 为区域 $D_f \subset R^m$ 上的 m 元函数。又设:

$$\begin{aligned} g : D_g &\rightarrow R^m, \\ (x_1, x_2, \dots, x_n) &\mapsto (y_1, y_2, \dots, y_m) \end{aligned} \tag{1}$$

为区域 $D_g \subset R^n$ 上的 n 元 m 维向量值函数, 那么, 对于复合函数:

$$\begin{aligned} z = f \circ g &= f[y_1(x_1, x_2, \dots, x_n), y_2(x_1, x_2, \dots, x_n), \dots, y_m(x_1, x_2, \dots, x_n)] \\ &\text{其中: } (x_1, x_2, \dots, x_n) \in D_g \end{aligned}$$

若 g 在 $x^0 \in D_g$ 点可导, 即 y_1, y_2, \dots, y_n 在 x^0 点可偏导, 且 f 在 $y^0 = g(x^0)$ 点可微, 则:

多元复合函数求导-2

$$\frac{\partial z}{\partial x}(x^0) = \left(\frac{\partial z}{\partial x_1}, \frac{\partial z}{\partial x_2}, \dots, \frac{\partial z}{\partial x_i}, \dots, \frac{\partial z}{\partial x_n} \right)_{x=x^0}$$

其中：

$$\frac{\partial z}{\partial x_i}(x^0) = \sum_{j=1}^m \frac{\partial z}{\partial y_j}(y^0) \frac{\partial y_j}{\partial x_i}(x^0)$$

即：

$$\frac{\partial z}{\partial x}(x^0) = \left(\frac{\partial z}{\partial y_1}, \frac{\partial z}{\partial y_2}, \dots, \frac{\partial z}{\partial y_m} \right)_{y=y^0} \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}_{x=x^0}$$

多元复合函数求导-3

若 g 处处可导，即 y_1, y_2, \dots, y_n 处处可偏导，且 f 处处可微，则：

$$\frac{\partial z}{\partial x} = \left(\frac{\partial z}{\partial x_1}, \frac{\partial z}{\partial x_2}, \dots, \frac{\partial z}{\partial x_i}, \dots, \frac{\partial z}{\partial x_n} \right)$$

其中：

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^m \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

即：

$$\frac{\partial z}{\partial x} = \left(\frac{\partial z}{\partial y_1}, \frac{\partial z}{\partial y_2}, \dots, \frac{\partial z}{\partial y_m} \right) \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \quad (\text{此矩阵即 Jacobian 矩阵})$$

一介全微分的形式不变性

对于多元函数 $z = f(y)$, 其中 $y = (y_1, y_2, \dots, y_m)^\top$ 。当 y 为自变量时, 一介全微分形式为:

$$dz = f'(y) dy$$

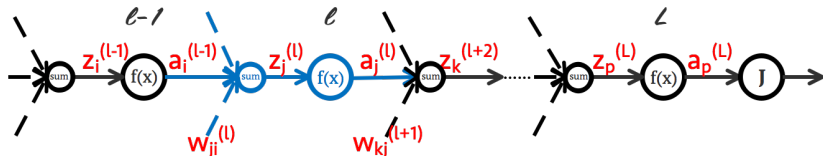
而当 y 为中间变量 $y = g(x) (x = (x_1, x_2, \dots, x_n)^\top)$ 时, $dy = g'(x) dx$ 。由链式规则, 得:

$$dz = (f \circ g)'(x) dx = f'(y)g'(x) dx = f'(y)(g'(x) dx) = f'(y) dy$$

注意符号:

$$\frac{dz}{dy} = f'(y); \quad \frac{dz}{dx} = (f \circ g)'(x) = f'(y)g'(x)$$

反向传播算法



$$z^l = W^{(l)} a^{(l-1)} + b^{(l)}$$

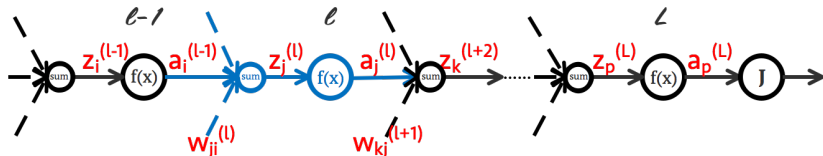
$$a^{(l)} = f(z^{(l)})$$

由梯度下降方法，可知，需要对每个权重权值 $w_{ij}^{(l)}$ ，求取：

$$w_{ji}^{(l)} = w_{ji}^{(l)} - \alpha \frac{\partial J(W, b)}{\partial w_{ji}^{(l)}} \quad b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial J(W, b)}{\partial b_i^{(l)}}$$

其中，关键是如何求取： $\frac{\partial J(W, b)}{\partial w_{ji}^{(l)}}$ 和 $\frac{\partial J(W, b)}{\partial b_i^{(l)}}$

反向传播算法



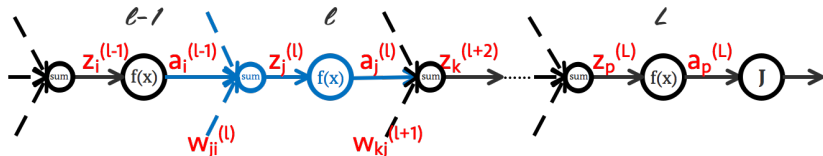
由前向传播过程可知： $z_j^{(l)} = \sum_{i=1}^{n_l} w_{ji}^{(l)} a_i^{(l-1)} + b_i^{(l)}$ 可知：

$$\frac{\partial J(W, b)}{\partial w_{ji}^{(l)}} = \frac{\partial J(W, b)}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}} = \frac{\partial J(W, b)}{\partial z_j^{(l)}} a_i^{(l-1)}$$

$$\frac{\partial J(W, b)}{\partial b_i^{(l)}} = \frac{\partial J(W, b)}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_i^{(l)}} = \frac{\partial J(W, b)}{\partial z_j^{(l)}}$$

到此为止，关键是如何求取 $\frac{\partial J(W, b)}{\partial z_j^{(l)}}$

反向传播算法

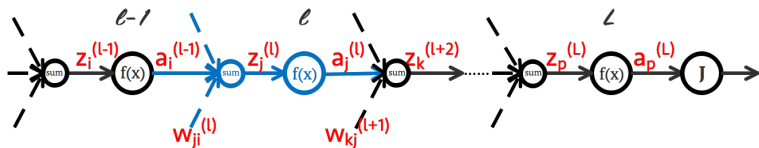


设: $\delta_j^{(l)} = \frac{\partial J(W, b)}{\partial z_j^{(l)}}$

因为: $z_k^{(l+1)} = \sum_{j=1}^{n_{l+1}} w_{kj}^{(l+1)} a_j^{(l)} + b^{(l+1)}$

所以, 可以选择从 $z_k^{(l+1)}$ 开始进行对 $z_j^{(l)}$ 进行求导计算:

反向传播算法推导



$$\begin{aligned}
 \delta_j^{(l)} &= \frac{\partial J(W, b)}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial J(W, b)}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \\
 &= \sum_{k=1}^{n_{l+1}} \frac{\partial J(W, b)}{\partial z_k^{(l+1)}} w_{kj}^{(l+1)} f'(z_j^{(l)}) \\
 &= \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} w_{kj}^{(l+1)} f'(z_j^{(l)})
 \end{aligned}$$

(2)

反向传播算法推导

对于最后一层：

$$\delta_p^{(L)} = \frac{\partial J(W, b)}{\partial z_p^{(L)}} = \frac{\partial J(W, b)}{\partial a_p^{(L)}} * \frac{\partial a_p^{(L)}}{\partial z_p^{(L)}} = \frac{\partial J(W, b)}{\partial a_p^{(L)}} * f'(z_p^{(L)})$$

并且：

$$\frac{\partial J(W, b)}{\partial w_{pq}^{(L)}} = \frac{\partial J(W, b)}{\partial z_p^{(L)}} a_p^{(L-1)} = \delta_p^{(L)} a_p^{(L-1)}$$

$$\frac{\partial J(W, b)}{\partial b_q^{(L)}} = \frac{\partial J(W, b)}{\partial z_p^{(L)}} = \delta_p^{(L)}$$

反向传播算法推导

小结一下，因为：

$$\frac{\partial J(W, b)}{\partial w_{ji}^{(l)}} = \frac{\partial J(W, b)}{\partial z_j^{(l)}} a_i^{(l-1)} \quad \frac{\partial J(W, b)}{\partial b_i^{(l)}} = \frac{\partial J(W, b)}{\partial z_j^{(l)}}$$

又因为 (上文推导结果)：

$$\frac{\partial J(W, b)}{\partial z_j^{(l)}} = \delta_j^{(l)} = \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} w_{kj}^{(l+1)} f'(z_j^{(l)})$$

从而得到：

$$\frac{\partial J(W, b)}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)} \quad \frac{\partial J(W, b)}{\partial b_i^{(l)}} = \delta_j^{(l)}$$

反向传播算法总结

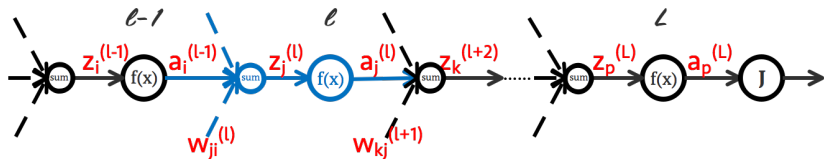
总结一下：

$$\frac{\partial J(W, b)}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)} = \left(\sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} w_{kj}^{(l+1)} f'(z_j^{(l)}) \right) a_i^{(l-1)}$$

$$\frac{\partial J(W, b)}{\partial b_i^{(l)}} = \delta_j^{(l)} = \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} w_{kj}^{(l+1)} f'(z_j^{(l)})$$

反向传播计算流程

Step-1: 依据前向传播算法求解每一层的激活值:

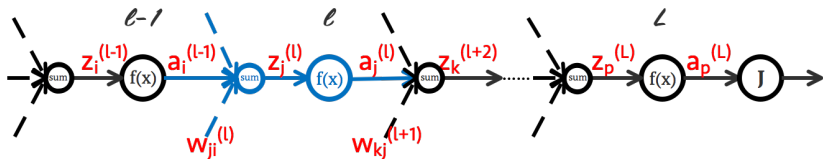


$$z^l = W^{(l)} a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = f(z^{(l)})$$

反向传播计算流程

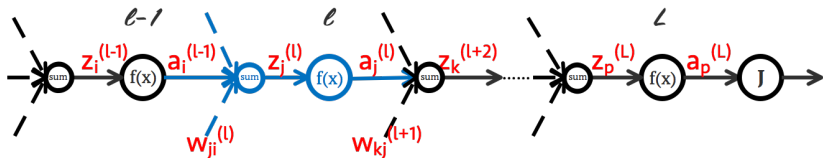
Step-2: 计算出最后一层 (L 层) 的每个神经元的 $\delta_p^{(L)}$:



$$\delta_p^{(L)} = \frac{\partial J(W, b)}{\partial a_p^{(L)}} * f'(z_p^{(L)})$$

反向传播计算流程

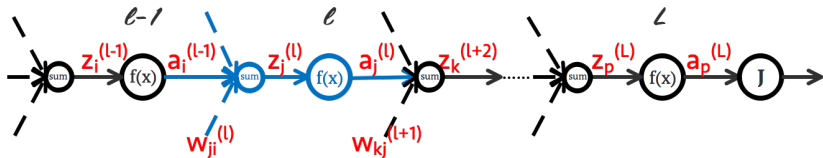
Step-3: 由后向前, 依次计算出各层 (l 层) 各个神经元的 $\delta_j^{(l)}$



$$\delta_j^{(l)} = \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} w_{kj}^{(l+1)} f'(z_j^{(l)})$$

反向传播计算流程

Step-4: 计算出各层 (l 层) 的各个权重 ($w_{ji}^{(l)}$) 的梯度 $\frac{\partial J(W,b)}{\partial w_{ji}^{(l)}}$ 及各个偏置 ($b_i^{(l)}$) 的梯度 $\frac{\partial J(W,b)}{\partial b_i^{(l)}}$:

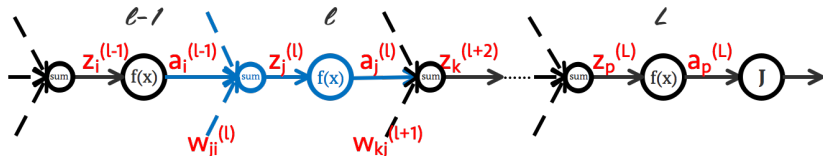


$$\frac{\partial J(W,b)}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}$$

$$\frac{\partial J(W,b)}{\partial b_i^{(l)}} = \delta_j^{(l)}$$

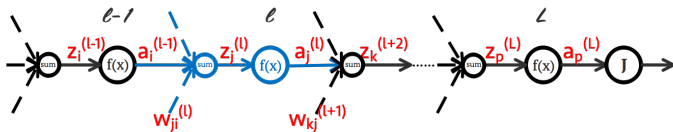
反向传播计算流程

Step-5: 对各层 (l 层) 的各个权重 ($w_{ji}^{(l)}$) 及各个偏置 ($b_i^{(l)}$) 进行更新, 直到代价函数 $J(W, b)$ 足够小:



$$w_{ji}^{(l)} = w_{ji}^{(l)} - \alpha \frac{\partial J(W, b)}{\partial w_{ji}^{(l)}} \quad b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial J(W, b)}{\partial b_i^{(l)}}$$

反向传播核心算式



$$\begin{aligned}
 w_{ji}^{(l)} &= w_{ji}^{(l)} - \alpha \frac{\partial J(W, b)}{\partial w_{ji}^{(l)}} \\
 &= w_{ji}^{(l)} - \alpha \frac{\partial J(W, b)}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ji}^{(l)}} = w_{ji}^{(l)} - \alpha \frac{\partial J(W, b)}{\partial z_j^{(l)}} a_i^{(l-1)} \\
 &= w_{ji}^{(l)} - \alpha \delta_j^{(l)} a_i^{(l-1)} \\
 &= w_{ji}^{(l)} - \alpha \left(\sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} w_{kj}^{(l+1)} f'(z_j^{(l)}) \right) a_i^{(l-1)}
 \end{aligned}$$

矢量化编程



- 提高算法速度的一种有效方法。
- 矢量化编程的思想就是尽量使用这些被高度优化的数值运算操作来实现我们的学习算法。

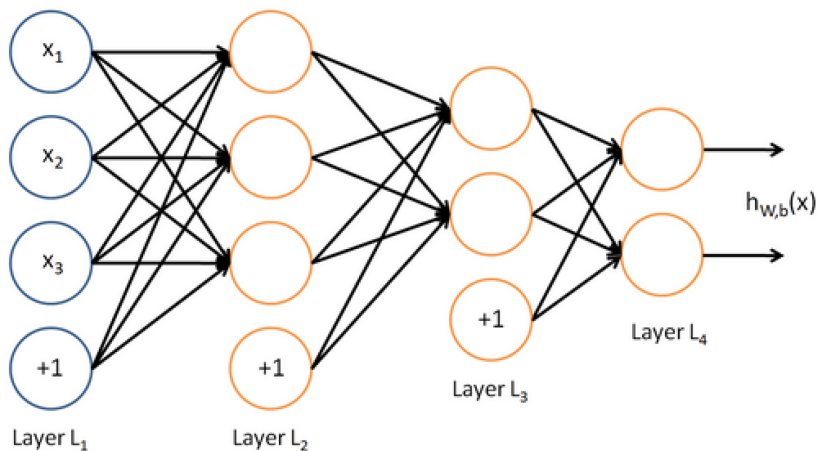
假设 $x \in \mathbb{R}^{n+1}$ 和 $\theta \in \mathbb{R}^{n+1}$ 为向量，需要计算 $z = \theta^T x$

```
z = 0;  
for i=1:(n+1),  
    z = z + theta(i) * x(i);  
end;
```

```
z = theta' * x;
```

代码中尽可能避免显式的for循环

神经网络的矢量编程



$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

■ 每个样本对应一个列向量，把这些列向量组成一个矩阵；

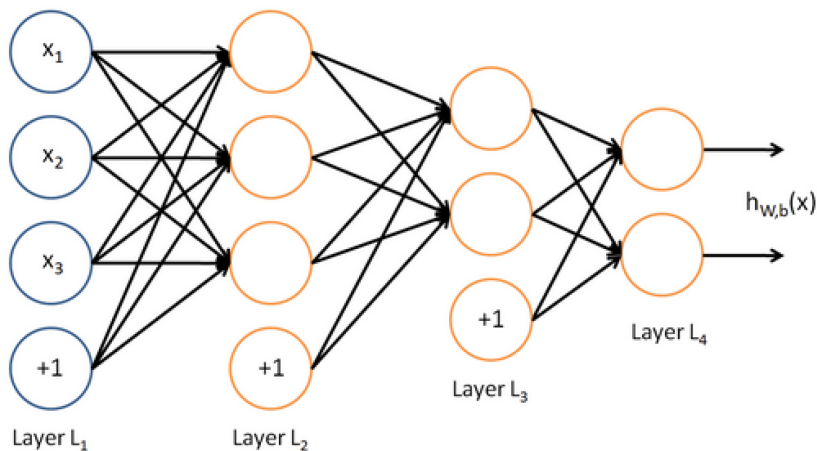
◆ $a^{(2)}$ 就成了一个 $s^{(2)} * m$ 的矩阵。

◆ 第 i 列表示：当第 i 个训练样本 $x(:,i)$ 输入到网络中时，该输入对隐神经元网络第二层的激励结果。

% 非向量化实现

```
for i=1:m,  
    z2 = W1 * x(:,i) + b1;  
    a2 = f(z2);  
    z3 = W2 * a2 + b2;  
    h(:,i) = f(z3);  
end;
```

神经网络的矢量编程



$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$

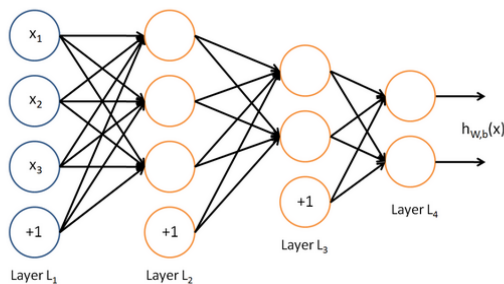
% 正向传播的向量化实现

```
z2 = W1 * x + repmat(b1,1,m);  
a2 = f(z2);  
z3 = W2 * a2 + repmat(b2,1,m);  
h = f(z3)
```

% 非向量化实现

```
for i=1:m,  
    z2 = W1 * x(:,i) + b1;  
    a2 = f(z2);  
    z3 = W2 * a2 + b2;  
    h(:,i) = f(z3);  
end;
```

神经网络的矢量编程



$$\delta_i^{(n_l)} = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$$

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

$$\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \bullet f'(z^{(l)})$$

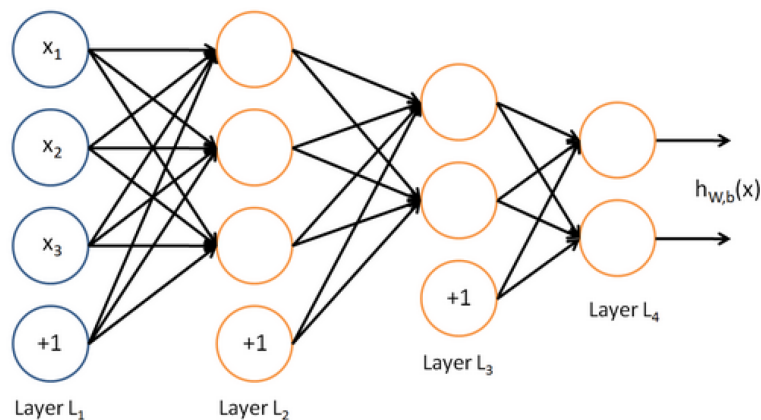
$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}$$

神经网络



1. 进行前馈传导计算, 利用前向传导公式, 得到 L_2, L_3, \dots 直到输出层 L_{n_l} 的激活值。

2. 对输出层 (第 n_l 层), 计算:

$$\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$$

3. 对于 $l = n_l - 1, n_l - 2, n_l - 3, \dots, 2$ 的各层, 计算:

$$\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) \bullet f'(z^{(l)})$$

4. 计算最终需要的偏导数值:

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T,$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}.$$

```
gradW1 = zeros(size(W1));
gradW2 = zeros(size(W2));
for i=1:m,
    delta3 = -(y(:,i) - h(:,i)) .* fprime(z3(:,i));
    delta2 = W2'*delta3(:,i) .* fprime(z2(:,i));

    gradW2 = gradW2 + delta3*a2(:,i)';
    gradW1 = gradW1 + delta2*a1(:,i)';
end;
```



```
from tensorflow.examples.tutorials.mnist import input_data
```

```
import tensorflow as tf
```

```
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
sess = tf.InteractiveSession()
```

```
in_units = 784
```

```
h1_units = 300
```

```
W1 = tf.Variable(tf.truncated_normal([in_units, h1_units], stddev=0.1))
```

```
b1 = tf.Variable(tf.zeros([h1_units]))
```

```
W2 = tf.Variable(tf.zeros([h1_units, 10]))
```

```
b2 = tf.Variable(tf.zeros([10]))
```

```
x = tf.placeholder(tf.float32, [None, in_units])
```

```
keep_prob = tf.placeholder(tf.float32)
```

```
hidden1 = tf.nn.relu(tf.matmul(x, W1)+b1)
```

```
hidden1_drop = tf.nn.dropout(hidden1, keep_prob)
```

```
y = tf.nn.softmax(tf.matmul(hidden1_drop, W2)+b2)
```

```
y_ = tf.placeholder(tf.float32, [None, 10])
```

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),  
                                              reduction_indices=[1]))
```

```
train_step = tf.train.AdamOptimizer(0.3).minimize(cross_entropy)
```

```
tf.global_variables_initializer().run()
```

```
for i in range(3000):
```

```
    batch_xs, batch_ys = mnist.train.next_batch(100)
```

```
    train_step.run({x: batch_xs, y_: batch_ys, keep_prob: 0.75})
```

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
print(accuracy.eval({x: mnist.test.images, y_:mnist.test.labels,  
                    keep_prob: 1.0}))
```



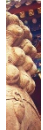
```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
sess = tf.InteractiveSession()

in_units = 784
h1_units = 300
W1 = tf.Variable(tf.truncated_normal([in_units, h1_units], stddev=0.1))
b1 = tf.Variable(tf.zeros([h1_units]))
W2 = tf.Variable(tf.zeros([h1_units, 10]))
b2 = tf.Variable(tf.zeros([10]))

x = tf.placeholder(tf.float32, [None, in_units])
keep_prob = tf.placeholder(tf.float32)

hidden1 = tf.nn.relu(tf.matmul(x, W1)+b1)
hidden1_drop = tf.nn.dropout(hidden1, keep_prob)
y = tf.nn.softmax(tf.matmul(hidden1_drop, W2)+b2)

y = tf.placeholder(tf.float32, [None, 10])
```



```
hidden1 = tf.nn.relu(tf.matmul(x, W1)+b1)
hidden1_drop = tf.nn.dropout(hidden1, keep_prob)
y = tf.nn.softmax(tf.matmul(hidden1_drop, W2)+b2)

y_ = tf.placeholder(tf.float32,[None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
                                                reduction_indices=[1]))
train_step = tf.train.AdamOptimizer(0.3).minimize(cross_entropy)

tf.global_variables_initializer().run()

for i in range(3000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    train_step.run({x: batch_xs, y_: batch_ys, keep_prob: 0.75})

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_:mnist.test.labels,
                    keep_prob: 1.0}))
```



Thanks.