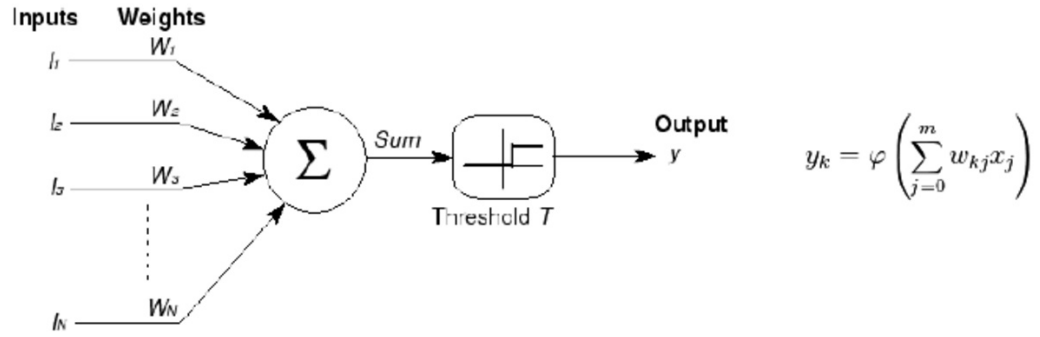
深度学习基础知识

深度学习发展历史

1943 年
由神经科学家麦卡洛克(W.S.McCilloch) 和数学家皮兹（W.Pitts）在《数学生物物理学公告》上发表论文《神经活动中内在思想的逻辑演算》（A Logical Calculus of the Ideas Immanent in Nervous Activity)。建立了神经网络和数学模型，称为 MCP 模型。所谓 MCP 模型，其实是按照生物神经元的结构和工作原理构造出来的一个抽象和简化了的模型，也就诞生了所谓的“模拟大脑”，人工神经网络的大门由此开启。

MCP 当时是希望能够用计算机来模拟人的神经元反应的过程，该模型将神经元简化为了三个过程：输入信号线性加权，求和，非线性激活（阈值法)。如下图所示



$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj}x_j\right)$$

1958 年
计算机科学家罗森布拉特（ Rosenblatt）提出了两层神经元组成的神经网络，称之为“感知器”(Perceptrons)。第一次将 MCP 用于机器学习（machine learning）分类(classification)。“感知器”算法算法使用 MCP 模型对输入的多维数据进行二分类，且能够使用梯度下降法从训练样本中自动学习更新权值。1962 年,该方法被证明为能够收敛，理论与实践效果引起第一次神经网络的浪潮。

1969 年
纵观科学发展史，无疑都是充满曲折的，深度学习也毫不例外。 1969 年，美国数学家及人工智能先驱 Marvin Minsky 在其著作中证明了感知器本质上是一种线性模型（linear model)，只能处理线性分类问题，就连最简单的 XOR（亦或）问题都无法正确分类。这等于直接宣判了感知器的死刑，神经网络的研究也陷入了将近 20 年的停滞。

1986 年
由神经网络之父 Geoffrey Hinton 在 1986 年发明了适用于多层感知器（MLP）

的 BP（Backpropagation）算法，并采用 Sigmoid 进行非线性映射，有效解决了非线性分类和学习的问题。该方法引起了神经网络的第二次热潮。

Sigmoid 函数是一个在生物学中常见的 S 型的函数，也称为 S 型生长曲线。在信息科学中，由于其单增以及反函数单增等性质，Sigmoid 函数常被用作神经网络的阈值函数，将变量映射到 0,1 之间。

$$S(x) = \frac{1}{1 + e^{-x}}$$

90 年代时期
1991 年 BP 算法被指出存在梯度消失问题，也就是说在误差梯度后项传递的过程中，后层梯度以乘性方式叠加到前层，由于 Sigmoid 函数的饱和特性，后层梯度本来就小，误差梯度传到前层时几乎为 0，因此无法对前层进行有效的学习，该问题直接阻碍了深度学习的进一步发展。

此外 90 年代中期，支持向量机算法诞生（SVM 算法）等各种浅层机器学习模型被提出，SVM 也是一种有监督的学习模型，应用于模式识别，分类以及回归分析等。支持向量机以统计学为基础，和神经网络有明显的差异，支持向量机等算法的提出再次阻碍了深度学习的发展。

发展期 2006 年 - 2012 年
2006 年，加拿大多伦多大学教授、机器学习领域泰斗、神经网络之父——Geoffrey Hinton 和他的学生 Ruslan Salakhutdinov 在顶尖学术刊物《科学》上发表了一篇文章，该文章提出了深层网络训练中梯度消失问题的解决方案：无监督预训练对权值进行初始化+有监督训练微调。斯坦福大学、纽约大学、加拿大蒙特利尔大学等成为研究深度学习的重镇，至此开启了深度学习在学术界和工业界的浪潮。

2011 年，ReLU 激活函数被提出，该激活函数能够有效的抑制梯度消失问题。2011 年以来，微软首次将 DL 应用在语音识别上，取得了重大突破。微软研究院和 Google 的语音识别研究人员先后采用 DNN 技术降低语音识别错误率 20%~30%，是语音识别领域十多年来最大的突破性进展。2012 年，DNN 技术在图像识别领域取得惊人的效果，在 ImageNet 评测上将错误率从 26%降低到 15%。在这一年，DNN 还被应用于制药公司的 DrugeActivity 预测问题，并获得世界最好成绩。

爆发期 2012 - 2017

2012 年，Hinton 课题组为了证明深度学习的潜力，首次参加 ImageNet 图像识别比赛，其通过构建的 CNN 网络 AlexNet 一举夺得冠军，且碾压第二名（SVM 方法）的分类性能。也正是由于该比赛，CNN 吸引到了众多研究者的注意。

AlexNet 的创新点在于:
(1)首次采用 ReLU 激活函数，极大增大收敛速度且从根本上解决了梯度消失问题。
(2)由于 ReLU 方法可以很好抑制梯度消失问题，AlexNet 抛弃了“预训练+微调”的方法，完全采用有监督训练。也正因为如此，DL 的主流学习方法也因此变为了纯粹的有监督学习。
(3)扩展了 LeNet5 结构，添加 Dropout 层减小过拟合，LRN 层增强泛化能力/减小过拟合。
(4)第一次使用 GPU 加速模型计算。
2013、2014、2015、2016 年，通过 ImageNet 图像识别比赛，DL 的网络结构，训练方法，GPU 硬件的不断进步，促使其在其他领域也在不断的征服战场。

深度学习目前还处于发展阶段，不管是理论方面还是实践方面都还有许多问题待解决，不过由于我们处在了一个“大数据”时代，以及计算资源的大大提升，新模型、新理论的验证周期会大大缩短。人工智能时代的开启必然会很大程度的改变这个世界。

人工智能，机器学习，深度学习有什么区别和联系？

人工智能（Artificial intelligence）简称 AI。人工智能是计算机科学的一个分支，它企图了解智能的本质，并生产出一种新的能以人类智能相似的方式做出反应的智能机器，是研究、开发用于模拟、延伸和扩展人的智能的理论、方法、技术及应用系统的一门新的技术科学。

人工智能目前分为弱人工智能和强人工智能和超人工智能。

1）弱人工智能：弱人工智能（ArtificialNarrow Intelligence /ANI),只专注于完成某个特定的任务，例如语音识别、图象识别和翻译等，是擅长于单个方面的人工智能。它们只是用于解决特定的具体类的任务问题而存在，大都是统计数据，以此从中归纳出模型。由于弱人工智能智能处理较为单一的问题，且发展程度并没有达到模拟人脑思维的程度，所以弱人工智能仍然属于“工具”的范畴，与传统的“产品”在本质上并无区别。

2) 强人工智能：强人工智能（Artificial Generallnteligence /AGI),属于人类级别的人工智能，在各方面都能和人类比肩，它能够进行思考、计划、解决问题、抽象思维、理解复杂理念、快速学习和从经验中学习等操作，并且和人类一样得心应手。

3）超人工智能：超人工智能（Artificial Superintelligence/ASI），在几乎所有领域都比最聪明的人类大脑都聪明许多，包括科学创新、通识和社交技能。在超人工智能阶段，人工智能已经跨过“奇点”，其计算和思维能力已经远超人

脑。此时的人工智能已经不是人类可以理解和想象。人工智能将打破人脑受到的维度限制，其所观察和思考的内容，人脑已经无法理解，人工智能将形成一个新的社会。

目前我们仍处于弱人工智能阶段。

2、机器学习

机器学习（MachineLearning）简称 ML。机器学习属于人工智能的一个分支，也是人工智能的和核心。机器学习理论主要是设计和分析一些让计算机可以自动”学习“的算法。

3、深度学习

深度学习（DeepLearning）简称 DL。最初的深度学习是利用深度神经网络来解决特征表达的一种学习过程。深度神经网络本身并不是一个全新的概念，可大致理解为包含多个隐含层的神经网络结构。为了提高深层神经网络的训练效果，人们对神经元的连接方法和激活函数等方面做出相应的调整。深度学习是机器学习研究中的一个新的领域，其动机在于建立、模拟人脑进行分析学习的神经网络，它模仿人脑的机制来解释数据，如图象、声音、文本。

1）监督学习：用一部分已知分类、有标记的样本来训练机器后，让它用学到的特征，对没有还分类、无标记的样本进行分类、贴标签。多用于分类。

2）非监督学习：所有的数据没有标记，类别未知，让它自己学习样本之间的相似性来进行分类。多用于聚类。

3）半监督学习：有两个样本集，一个有标记，一个没有标记。综合利用有类标的样本（ labeled sample）和没有类标的样本（ unlabeled sample），来生成合适的分类。

区别于联系

机器学习是一种实现人工智能的方法，深度学习是一种实现机器学习的技术

神经元，单层感知机，多层感知机

人工神经元（Artificial Neuron），简称神经元（Neuron），是构成神经网络的基本单元，其主要是模拟生物神经元的结构和特性，接受一组输入信号并产出输出。一个生物神经元通常具有多个树突和一条轴突。树突用来接受信息，轴突用来发送信息。当神经元所获得的输入信号的积累超过某个阈值时，它就处于兴奋状态，产生电脉冲。轴突尾端有许多末梢可以给其他个神经元的树突产生连接（突触），并将电脉冲信号传递给其它神经元。

MP 神经元[McCulloch and Pitts, 1943]。现代神经网络中的神经元和 M-P 神经元的结构并无太多变化。不同的是，MP 神经元中的激活函数 f 为 0 或 1 的阶跃函数，而现代神经元中的激活函数通常要求是连续可导的函数。

假设一个神经元接受d个输入$x_1, x_2, \cdots, x_d$，令向量$\mathbf{x} = [x_1; x_2; \cdots; x_d]$来表示这组输入，并用**净输入**（Net Input）$z \in \mathbb{R}$表示一个神经元所获得的输入信号$\mathbf{x}$的加权和，

$$z = \sum_{i=1}^{d} w_i x_i + b \tag{4.1}$$

$$= \mathbf{w}^{\mathrm{T}}\mathbf{x} + b, \tag{4.2}$$

其中$\mathbf{w} = [w_1; w_2; \cdots; w_d] \in \mathbb{R}^d$是$d$维的权重向量，$b \in \mathbb{R}$是偏置。

净输入z在经过一个非线性函数$f(\cdot)$后，得到神经元的**活性值**（Activation）a，

$$a = f(z), \tag{4.3}$$

单层感知机和多层感知机(MLP)是最基础的神经网络结构。将卷积操作创新的加入到神经网络结构形成了卷积神经网络，卷积神经网络给现代人工智能注入了活力。感知机网络和卷积网络(CNN)都属于前馈型网络(FeedForward Network)。

单层感知机是二分类的线性分类模型，输入是被感知数据集的特征向量，输出时数据集的类别{+1,-1}。单层感知机的函数近似非常有限，其决策边界必须是一个超平面，严格要求数据是线性可分的。支持向量机，用核函数修正了感知器的不足，将特征向量有效的映射到更高维的空间使得样本成为线性可分的数据集。

本节尝试揭开单层感知机的神秘面纱，尝试用数据给出模型和解释。

1 单层感知机的模型

单层感知机目标是将被感知数据集划分为两类的分离超平面，并计算出该超平面。单层感知机是二分类的线性分类模型，输入是被感知数据集的特征向量，输出时数据集的类别{+1,-1}。感知器的模型可以简单表示为：

$$f(x) = sign(w.x + b)$$

该函数称为单层感知机，其中 w 是网络的 N 维权重向量，b 是网络的 N 维偏置向量，w.x 是 w 和 x 的内积，w 和 b 的 N 维向量取值要求在实数域。

sign 函数是感知机的早期激活函数，后面又演化出一系列的激活函数。激活函数一般采用非线性激活函数，以增强网络的表达能力。常见的激活函数有：sign, sigmoid, tanh, ReLU 等。

$$sign(x) = \begin{cases} +1(x > 0) \\ -1(x < 0) \end{cases}$$

为单层感知机与逻辑回归的差别就是感知机激活函数是 sign，逻辑回归的激活函数是 sigmoid。sign(x)将大于 0 的分为 1，小于 0 的分为-1；sigmoid 将大于 0.5 的分为 1，小于 0.5 的分为 0。因此 sign 又被称为单位阶跃函数，逻辑回归也被看作是一种概率估计。

单层感知机的训练

如果数据集可以被一个超平面完全划分，则称该数据集是线性可分的数据集，否则称为线性不可分的数据集。对于线性可分的数据集，单层感知机基本任务是寻找一个线性可分的超平面

$$S = wx + b = 0$$

该超平面能够将所有的正类和负类完全划分到超平面的两侧。对于线性不可分的数据集，单层感知机由于模型无法稳定收敛，而无法处理。

给出 N 个线性可分的数据集

$$(x_1, y_1), (x_2, y_2), ...(x_n, y_n),$$

其中输出向量

$$y_i = -1, 1$$

输入特征向量

$$x_i = x_1, x_2, x_3, ...x_n$$

学习率 η(0<η<1)，模型选择

$$f(x) = sign(w.x + b)$$

使用 Loss 函数和梯度下降法求解 w 和 b 向量;

对所有数据训练完成后，如果至少有一个数据训练错误，则要对权值进行重新训练，直到对所有数据训练正确，结束训练。

3 单层感知机的使用

外部系统将输入数据变换成单层感知机接受的值域;外部系统将合法的输入数据输入到单层感知机；单层感知机利用已有模型计算结果；单层感知机输出数据;外部系统接收输出数据并使用。

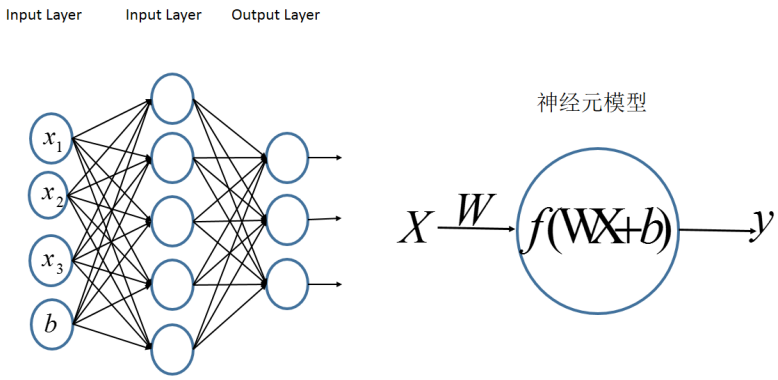
4 单层感知机的缺陷

4.1 XOR 问题

1969 年马文·明斯基将感知机兴奋推到最高顶峰。他提出了著名的 XOR 问题和感知器数据线性不可分的情形。此后,神经网络的研究将处于休眠状态,直到上世纪 80 年代。

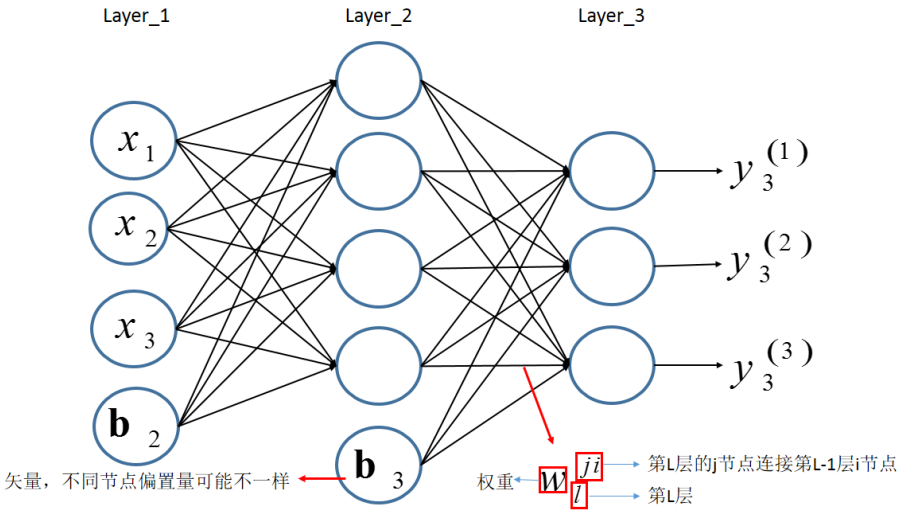
多层感知机

多层感知机的一个重要特点就是多层，我们将第一层称之为输入层，最后一层称之有输出层，中间的层称之为隐层。MLP 并没有规定隐层的数量，因此可以根据各自的需求选择合适的隐层层数。且对于输出层神经元的个数也没有限制。
MLP 神经网络结构模型如下，本文中只涉及了一个隐层，输入只有三个变量[x1,x2,x3][x1,x2,x3]和一个偏置量 bb，输出层有三个神经元。相比于感知机算法中的神经元模型对其进行了集成。



前向传播
前向传播指的是信息从第一层逐渐地向高层进行传递的过程。以下图为例来进行前向传播的过程的分析。

假设第一层为输入层，输入的信息为[x1,x2,x3][x1,x2,x3]。对于层l，用L_l表示该层的所有神经元，其输出为y_l，其中第 j 个节点的输出为$y_l^{(j)}$，该节点的输入为$u_l^{(j)}$，连接第l层与第$(l-1)$的权重矩阵为W_l，上一层（第$l-1$层）的第 i 个节点到第l层第 j 个节点的权重为$w_l^{(ji)}$



结合之前定义的字母标记，对于第二层的三个神经元的输出则有

$$y_2^{(1)} = f(u_2^{(1)}) = f(\sum_{i=1}^{n} w_2^{1i} x_i + b_2^{(1)}) = f(w_2^{(11)} x_1 + w_2^{(12)} x_2 + w_2^{(13)} x_3 + b_2^{(1)})$$

$$y_2^{(2)} = f(u_2^{(2)}) = f(\sum_{i=1}^{n} w_2^{2i} x_i + b_2^{(2)}) = f(w_2^{(21)} x_1 + w_2^{(22)} x_2 + w_2^{(23)} x_3 + b_2^{(2)})$$

$$y_2^{(3)} = f(u_2^{(3)}) = f(\sum_{i=1}^{n} w_2^{3i} x_i + b_2^{(3)}) = f(w_2^{(31)} x_1 + w_2^{(32)} x_2 + w_2^{(33)} x_3 + b_2^{(3)})$$

将第二层的前向传播计算过程推广到网络中的任意一层，则：

$$\begin{cases} y_l^{(j)} = f\left(u_l^{(j)}\right) \\ u_l^{(j)} = \sum\limits_{i \in L_{l-1}} w_l^{(ji)} y_{l-1}^{(i)} + b_l^{(j)} \\ \mathbf{y}_l = f(\mathbf{u}_l) = f(\mathbf{W}_l \mathbf{y}_{l-1} + \mathbf{b}_l) \end{cases}$$

反向传播

基本的模型搭建完成后的，训练的时候所做的就是完成模型参数的更新。由于存在多层的网络结构，因此无法直接对中间的隐层利用损失来进行参数更新，但可以利用损失从顶层到底层的反向传播来进行参数的估计。

假设多层感知机用于分类，在输出层有多个神经元，每个神经元对应一个标签。输入样本为$\mathbf{x} = [x_1, x_2, \cdots, x_n]$,其标签为$\mathbf{t}$;

对于层l，用L_l表示该层的所有神经元，其输出为$\mathbf{y}_l$,其中第j个节点的输出为$y_l^{(j)}$,该节点的输入为$u_l^{(j)}$，连接第l层与第$(l-1)$层的权重矩阵为W_l，上一层（第$l-1$层）的第i个节点到第l层第j个节点的权重为$w_l^{(ji)}$。

对于网络的最后一层第k层——输出层，现在定义损失函数：

$$E = \frac{1}{2} \sum_{j \in L_k} (t^{(j)} - y_k^{(j)})^2$$

为了极小化损失函数，通过梯度下降来进行推导：

$$\begin{cases} \dfrac{\partial E}{\partial w_l^{(ji)}} = \dfrac{\partial E}{\partial y_l^{(j)}} \dfrac{\partial y_l^{(j)}}{\partial w_l^{(ji)}} = \dfrac{\partial E}{\partial y_l^{(j)}} \dfrac{\partial y_l^{(j)}}{\partial u_l^{(j)}} \dfrac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} \\ \dfrac{\partial E}{\partial b_l^{(j)}} = \dfrac{\partial E}{\partial y_l^{(j)}} \dfrac{\partial y_l^{(j)}}{\partial b_l^{(j)}} = \dfrac{\partial E}{\partial y_l^{(j)}} \dfrac{\partial y_l^{(j)}}{\partial u_l^{(j)}} \dfrac{\partial u_l^{(j)}}{\partial b_l^{(j)}} \end{cases}$$

在上式子中，根据之前的定义，很容易得到：

$$\begin{cases} \dfrac{\partial y_l^{(j)}}{\partial u_l^{(j)}} = f'(u_l^{(j)}) \\ \dfrac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} = y_{l-1}^{(i)} \\ \dfrac{\partial u_l^{(j)}}{\partial b_l^{(j)}} = 1 \end{cases}$$

那么则有：

$$\begin{cases} \dfrac{\partial E}{\partial w_l^{(ji)}} = \dfrac{\partial E}{\partial y_l^{(j)}} \dfrac{\partial y_l^{(j)}}{\partial u_l^{(j)}} \dfrac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} = \dfrac{\partial E}{\partial y_l^{(j)}} f'(u_l^{(j)}) y_{l-1}^{(i)} \\ \dfrac{\partial E}{\partial b_l^{(j)}} = \dfrac{\partial E}{\partial y_l^{(j)}} \dfrac{\partial y_l^{(j)}}{\partial u_l^{(j)}} \dfrac{\partial u_l^{(j)}}{\partial b_l^{(j)}} = \dfrac{\partial E}{\partial y_l^{(j)}} f'(u_l^{(j)}) \end{cases}$$

另有，下一层所有结点的输入都与前一层的每个结点输出有关，因此损失函数可以认为是下一层的每个神经元结点输入的函数。那么：

$$\begin{aligned} \frac{\partial E}{\partial y_l^{(j)}} &= \frac{\partial E(u_{l+1}^{(1)}, u_{l+1}^{(2)}, \ldots, u_{l+1}^{(k)}, \ldots, u_{l+1}^{(K)})}{\partial y_l^{(j)}} \\ &= \sum_{k \in L_{l+1}} \frac{\partial E}{\partial u_{l+1}^{(k)}} \frac{\partial u_{l+1}^{(k)}}{\partial y_l^{(j)}} \\ &= \sum_{k \in L_{l+1}} \frac{\partial E}{\partial y_{l+1}^{(k)}} \frac{\partial y_{l+1}^{(k)}}{\partial u_{l+1}^{(k)}} \frac{\partial u_{l+1}^{(k)}}{\partial y_l^{(j)}} \\ &= \sum_{k \in L_{l+1}} \frac{\partial E}{\partial y_{l+1}^{(k)}} \frac{\partial y_{l+1}^{(k)}}{\partial u_{l+1}^{(k)}} w_{l+1}^{(kj)} \end{aligned}$$

此处定义节点的灵敏度为误差对输入的变化率，即：

$$\delta = \frac{\partial E}{\partial u}$$

那么第 l层第j个节点的灵敏度为：

$$\delta_l^{(j)} = \frac{\partial E}{\partial u_l^{(j)}} = \frac{\partial E}{\partial y_l^{(j)}} \frac{\partial y_l^{(j)}}{\partial u_l^{(j)}} = \frac{\partial E}{\partial y_l^{(j)}} f'(u_l^{(j)})$$

结合灵敏度的定义，则有：

$$\begin{aligned} \frac{\partial E}{\partial y_l^{(j)}} &= \sum_{k \in L_{l+1}} \frac{\partial E}{\partial y_{l+1}^{(k)}} \frac{\partial y_{l+1}^{(k)}}{\partial u_{l+1}^{(k)}} w_{l+1}^{(kj)} \\ &= \sum_{k \in L_{l+1}} \delta_{l+1}^k w_{l+1}^{(kj)} \end{aligned}$$

上式两边同时乘上 $f'(u_l^{(j)})$，则有

$$\delta_l^{(j)} = \frac{\partial E}{\partial y_l^{(j)}} f'(u_l^{(j)}) = f'(u_l^{(j)}) \sum_{k \in L_{l+1}} \delta_{l+1}^k w_{l+1}^{(kj)}$$

注意到上式中表达的是前后两层的灵敏度关系，而对于最后一层，也就是输出层来说，并不存在后续的一层，因此并不满足上式。但输出层的输出是直接和误差联系的，因此可以用损失函数的定义来直接求取偏导数。那么：

$$\delta_l^{(j)} = \frac{\partial E}{\partial y_l^{(j)}} f'(u_l^{(j)}) = \begin{cases} f'(u_l^{(j)}) \sum_{k \in L_{l+1}} \delta_{l+1}^k w_{l+1}^{(kj)} & l\text{层为隐层} \\ f'(u_l^{(j)})(y_l^{(j)} - t^{(j)}) & l\text{层为输出层} \end{cases}$$

至此，损失函数对各参数的梯度为：

$$\begin{cases} \frac{\partial E}{\partial w_l^{(ji)}} = \frac{\partial E}{\partial u_l^{(j)}} \frac{\partial u_l^{(j)}}{\partial w_l^{(ji)}} = \delta_l^{(j)} y_{l-1}^{(i)} \\ \frac{\partial E}{\partial b_l^{(j)}} = \frac{\partial E}{\partial u_l^{(j)}} \frac{\partial u_l^{(j)}}{\partial b_l^{(j)}} = \delta_l^{(j)} \end{cases}$$

上述的推到都是建立在单个节点的基础上，对于各层所有节点，采用矩阵的方式表示，则上述公式可以写成：

$$\begin{aligned} \frac{\partial E}{\partial \mathbf{W}_l} &= \boldsymbol{\delta}_l \mathbf{y}_{l-1}^{\mathrm{T}} \\ \frac{\partial E}{\partial \mathbf{b}_l} &= \boldsymbol{\delta}_l \\ \boldsymbol{\delta}_l &= \begin{cases} (\mathbf{W}_{l+1}^T \boldsymbol{\delta}_{l+1}) \circ f'(\mathbf{u}_l), & l\text{层为隐层} \\ (\mathbf{y}_l - \mathbf{t}) \circ f'(\mathbf{u}_l), & l\text{层为输出层} \end{cases} \end{aligned}$$

其中运算符 ○表示矩阵或者向量中的对应元素相乘。
常见的几个激活函数的导数为：

$$
\begin{aligned}
f'(\mathbf{u}_l) &= sigmoid'(\mathbf{u}_l) = sigmoid(\mathbf{u}_l)(1 - sigmoid(\mathbf{u}_l)) = \mathbf{y}_l(1 - \mathbf{y}_l) \\
f'(\mathbf{u}_l) &= tanh'(\mathbf{u}_l) = 1 - tanh^2(\mathbf{u}_l) = 1 - \mathbf{y}_l^2 \\
f'(\mathbf{u}_l) &= softmax'(\mathbf{u}_l) = softmax(\mathbf{u}_l) - softmax^2(\mathbf{u}_l) = \mathbf{y}_l - \mathbf{y}_l^2
\end{aligned}
$$

根据上述公式，可以得到各层参数的更新公式为：

$$
\begin{aligned}
\mathbf{W}_l &:= \mathbf{W}_l - \eta \frac{\partial E}{\partial \mathbf{W}_l} = \mathbf{W}_l - \eta \boldsymbol{\delta}_l \mathbf{y}_{l-1}^{\mathrm{T}} \\
\mathbf{b}_l &:= \mathbf{b}_l - \eta \frac{\partial E}{\partial b} = \mathbf{b}_l - \eta \boldsymbol{\delta}_l
\end{aligned}
$$