

# 深度学习

王璐

从 2012 年开始，深度学习 (deep learning) 让计算机识别图形的能力变得无比强大。在此之前的图形识别算法总是基于一些明确的规则，让计算机根据规则判断，效果总是不好。深度学习是一种没有规则的学习，就像人虽然能一眼区分猫和狗，却很难用语言明确地定义猫和狗。

## 1 神经网络

深度学习的基础——神经网络 (neural network)，几十年前就提出来了，但是一开始并不被看好。神经网络的思想来自脑神经科学，大脑里有数百亿个神经元，人脑的识别能力不都是按照明确的逻辑规则进行的，而是通过专门的神经网络“长”出来的。正如一个法官所说“什么叫色情图片？我没办法给你一个明文规定，但如果我看见了，我就能识别出来。”

图1代表了一个最简单的计算机神经网络，它从左到右分为三层：第一层代表输入数据 (input)，第二层和第三层的每个圆圈代表一个神经元。数据输入进来，经过第二层“隐藏层” (hidden layer) 各个神经元的处理，把信号传递给第三层“输出层” (output layer)，输出层神经元再经过一番处理后做出判断。

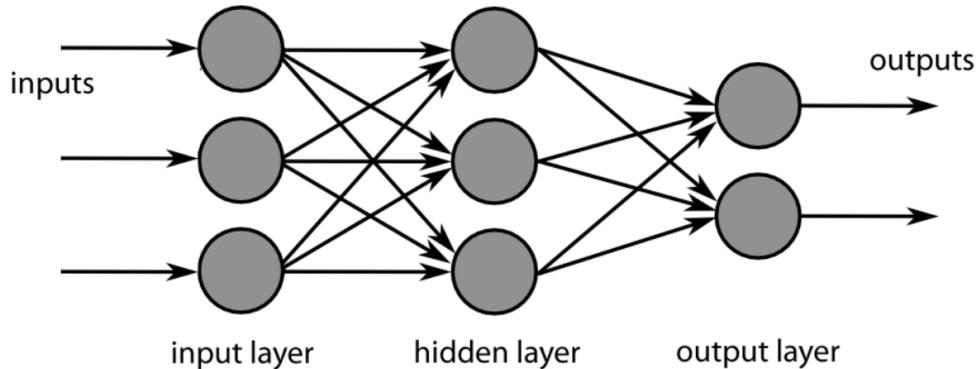


Figure 1: 一个简单的神经网络。Picture source: [coderoncode.com](http://coderoncode.com)

深度学习最简单的理解就是中间有不止一层隐藏层的神经网络，如图2所示，左图是一个简单

的神经网络，右图是深度学习神经网络。

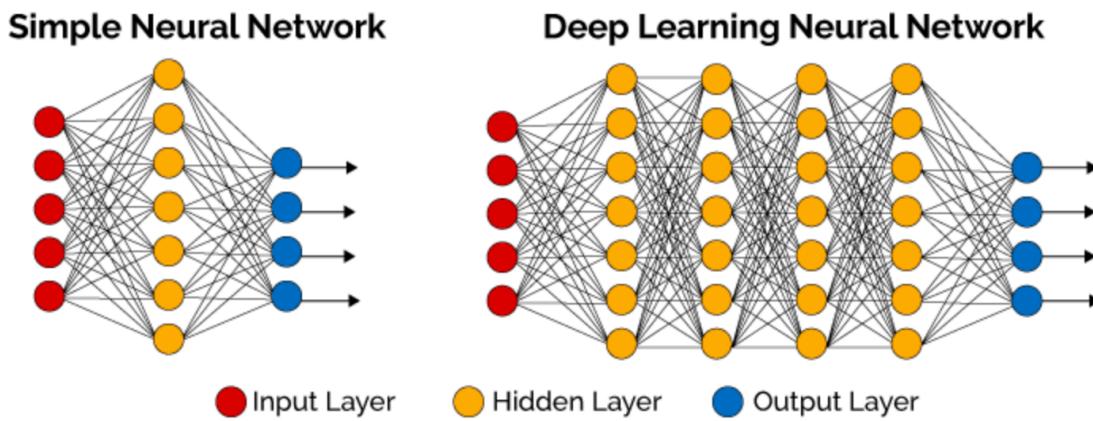


Figure 2: 神经网络 (左) 和深度学习神经网络 (右)。Picture source: [hackernoon.com](https://hackernoon.com)

神经网络中的神经元一般由三部分组成：输入、内部参数和输出。图3展示了一个根据交通信号灯判断是否前进的神经元。神经元的输入是红灯、黄灯和绿灯的亮灯情况，1 代表亮，0 代表不亮， $(1, 0, 0)$  这组输入代表红灯亮，黄灯和绿灯不亮。神经元的内部参数包括“权重” (weight)，它给每个输入值都赋一个权重。图3中的神经元给红灯的权重是 -1，黄灯的权重是 0，绿灯的权重是 1。神经元还有一个内部参数 bias，图3中的 bias = -0.5. 神经元做的计算就是把输入乘以各自权重、相加、再加上 bias. 比如当输入是  $(1, 0, 0)$  时，神经元的计算结果为  $1 \times (-1) + 0 \times 0 + 0 \times 1 - 0.5 = -1.5$ . 输出是做判断，比如判断标准可以是：计算结果  $> 0$  就前进， $\leq 0$  就停止。此时计算结果  $-1.5 < 0$ ，所以神经元会输出“停止”，这就是神经元的工作原理。在实际应用中，神经元会在计算过程中加入非线性函数，比如 Sigmoid 函数  $\phi(x) = 1/(1 + e^{-x})$ ，确保输出值在 0 和 1 之间。

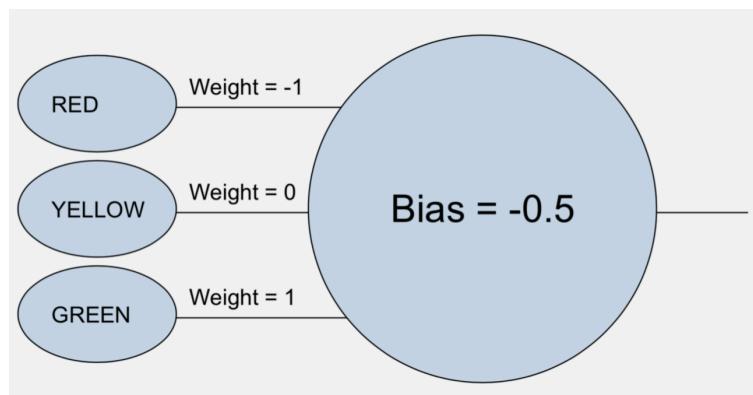


Figure 3: 一个根据交通信号灯判断是否前进的神经元。Picture source: [arstechnica.com](https://arstechnica.com)

## 1.1 误差反向传播 (Backpropagation)

神经元的内部参数，包括权重和 bias 都是可调的。用数据训练神经网络的过程，就是调整各个神经元内部参数的过程。神经网络的结构在训练中不变，其内部神经元的参数决定了神经网络的功能。深度学习给神经元调参的方法叫“误差反向传播”(Rumelhart et al., 1988)，主要使用的是求导中的“链式法则”(chain rule)。

图4展示了一个深度学习网络最后一层(输出层)编号为100的神经元的工作原理：100号神经元将来自其它神经元的输入通过加权求和得到net<sub>100</sub>，再使用Sigmoid函数 $\phi(x)$ 将net<sub>100</sub>映射到(0, 1)区间，输出值 $o_{100} = \phi(\text{net}_{100})$ 。

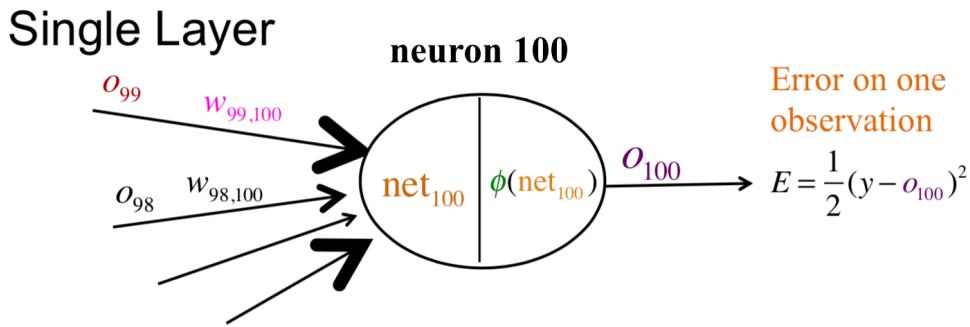


Figure 4: 输出层 100 号神经元的工作原理及观测误差。Picture source: Cynthia Rudin

假设只有一个输入数据且真实的结果为 $y$ ，定义深度学习网络的误差：

$$E = \frac{1}{2}(y - o_{100})^2$$

现在根据误差 $E$ 调整99号神经元的输入权重 $w_{99,100}$ 。使用链式法则求导

$$\begin{aligned} \frac{\partial E}{\partial w_{99,100}} &= \frac{dE}{do_{100}} \cdot \frac{do_{100}}{d\text{net}_{100}} \cdot \frac{\partial \text{net}_{100}}{\partial w_{99,100}} \\ &= -(y - o_{100}) \cdot \frac{d\phi(\text{net}_{100})}{d\text{net}_{100}} \cdot o_{99} \\ &= -(y - o_{100})\phi(\text{net}_{100})[1 - \phi(\text{net}_{100})]o_{99} \\ &= \underbrace{-(y - o_{100})o_{100}(1 - o_{100})}_{\triangleq \delta_{100}} o_{99}. \end{aligned}$$

此处新定义的

$$\delta_{100} = \frac{\partial E}{\partial \text{net}_{100}} = \frac{dE}{do_{100}} \cdot \frac{do_{100}}{d\text{net}_{100}} = -(y - o_{100})o_{100}(1 - o_{100})$$

只与输出 $o_{100}$ 有关。

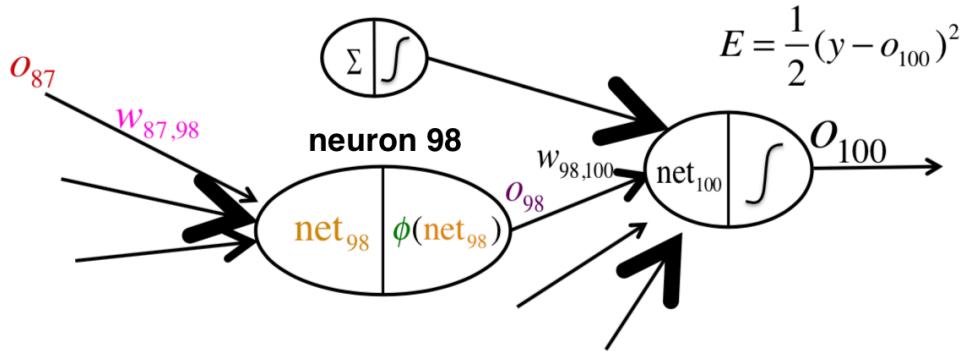


Figure 5: 倒数第二层 98 号神经元的工作原理。Picture source: Cynthia Rudin

后退一层，考察倒数第二层编号 98 的神经元，如图5所示。如果想根据误差  $E$  调整 87 号神经元的输入权重  $w_{87,98}$ ，继续使用链式法则求导：

$$\begin{aligned}\frac{\partial E}{\partial w_{87,98}} &= \frac{\partial E}{\partial o_{98}} \cdot \frac{do_{98}}{d\text{net}_{98}} \cdot \frac{\partial \text{net}_{98}}{\partial w_{87,98}} \\ &= \frac{\partial E}{\partial \text{net}_{100}} \cdot \frac{\partial \text{net}_{100}}{\partial o_{98}} \phi(\text{net}_{98}) [1 - \phi(\text{net}_{98})] o_{87} \\ &= \underbrace{\delta_{100} w_{98,100} o_{98} (1 - o_{98})}_{\triangleq \delta_{98}} o_{87}\end{aligned}$$

其中定义

$$\delta_{98} = \frac{\partial E}{\partial \text{net}_{98}} = \frac{\partial E}{\partial o_{98}} \cdot \frac{do_{98}}{d\text{net}_{98}} = \delta_{100} w_{98,100} o_{98} (1 - o_{98}).$$

可以看到  $\delta_{98}$  的计算需要先知道下游神经元 100 对应的  $\delta_{100}$  的值。

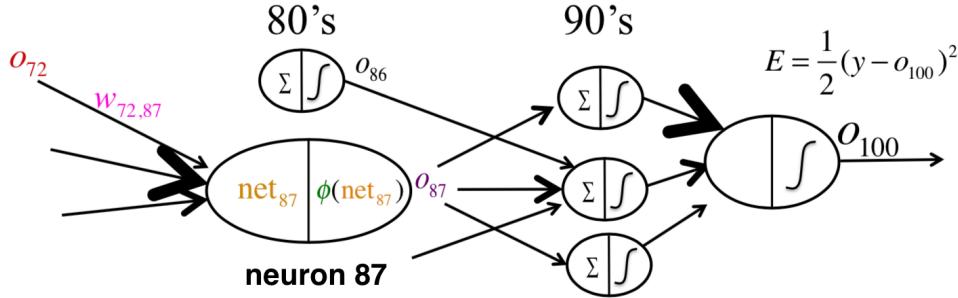


Figure 6: 倒数第三层 87 号神经元的工作原理。Picture source: Cynthia Rudin

再后退一层，考虑倒数第三层编号 87 的神经元，如图6所示。为了调整 72 号神经元的输入权重  $w_{72,87}$ ，对  $E$  关于  $w_{72,87}$  求导：

$$\frac{\partial E}{\partial w_{72,87}} = \frac{\partial E}{\partial o_{87}} \cdot \frac{do_{87}}{d\text{net}_{87}} \cdot \frac{\partial \text{net}_{87}}{\partial w_{72,87}}$$

$$\begin{aligned}
&= \left( \sum_{j=90}^{99} \frac{\partial E}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial o_{87}} \right) o_{87}(1 - o_{87})o_{72} \\
&= \left( \sum_{j=90}^{99} \delta_j w_{87,j} \right) o_{87}(1 - o_{87})o_{72}
\end{aligned}$$

可以看到为了计算该导数的值，需要提前计算出 87 号神经元所有下游神经元的  $\delta_j$ 's.

误差反向传播即按上述过程从深度学习网络的最后一层开始，逐层计算出误差函数关于每个神经元内部参数的导数  $\frac{\partial E}{\partial w_{a,b}}$ ，然后使用梯度下降法 (gradient descent) 更新参数：

$$w_{a,b} \leftarrow w_{a,b} - \lambda \frac{\partial E}{\partial w_{a,b}}$$

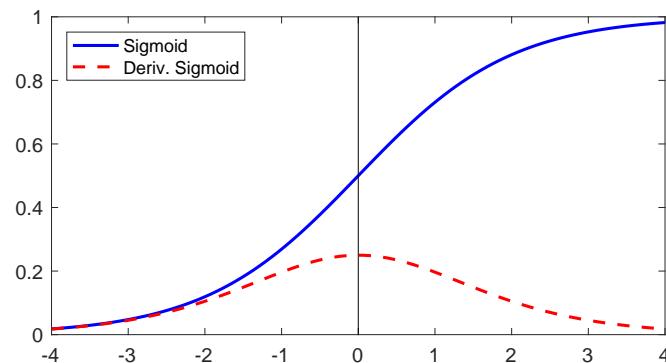
所有参数更新完毕后，再正向计算出深度学习网络的输出，得到新的误差，再通过误差反向传播更新参数，……不断重复上述过程直到输出几乎不变，所有参数收敛到稳定值。

## 1.2 几种常用的激活函数

激活函数是将神经元输入值的加权和进行变化产生输出的过程。最简单的激活函数是线性激活函数，比如在 Section 1 的交通灯例子中，激活函数就是输入值加权和本身，不做任何变化。使用线性激活函数的神经网络很容易训练，但是不能学出较复杂的结构，实际效果差。实践中一般使用非线性激活函数，它可以使神经元学出数据中非常复杂的结构。几种常用的激活函数有：

- Sigmoid 函数

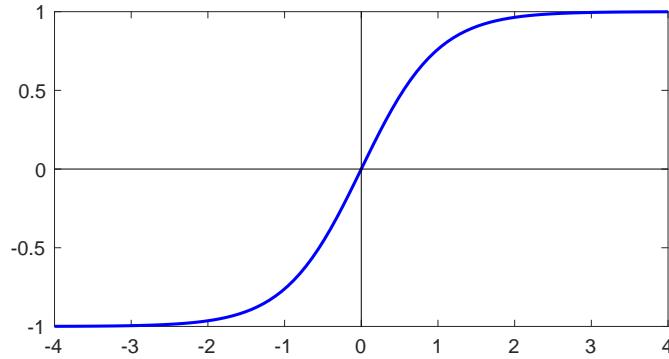
$$\begin{aligned}
\phi(x) &= \frac{1}{1 + e^{-x}} \\
\phi'(x) &= \phi(x)(1 - \phi(x))
\end{aligned}$$



Sigmoid 在 1990 年前是很流行的激活函数，它可以将很大的输入值转化到  $(0, 1)$  区间，但是它的导数在远离  $x = 0$  处迅速下降到 0。随着神经网络层数的增加，利用误差反向传播更新各个权重时，会发现远离输出层的神经元权重的导数会很快下降到 0，这种现象被称为“梯度消失” (vanishing gradients problem)。梯度消失导致神经元的权重无法根据损失函数做出调整，不利于权重的优化。

- tanh 函数

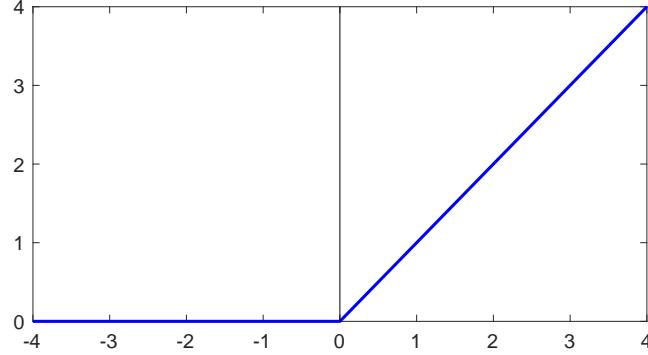
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



双曲正切函数  $\tanh$  是 1990 至 2000 年左右很流行的激活函数，它的取值范围是  $(-1, 1)$ ，且输出以 0 为中心，对权重的更新效率比 sigmoid 高。但是  $\tanh$  的导数也只在  $x = 0$  附近较大，在远离  $x = 0$  处迅速下降到 0，因此使用  $\tanh$  激活函数也会面临梯度消失的问题。

- **ReLU 函数**

$$f(x) = \max(x, 0)$$



ReLU 是整流线性单位 (rectified linear unit) 的缩写，它使神经网络的层数可以极大增长，促进了深度学习的发展 (Glorot et al., 2011)，是目前最流行的激活函数。ReLU 函数是分段线性的，对输入的敏感性比 sigmoid 和  $\tanh$  好，几乎不会出现梯度消失。同时又因为避免了指数运算，计算速度更快。

ReLU 函数的导数很容易计算， $x > 0$  时为 1， $x < 0$  时为 0. 虽然 ReLU 在  $x = 0$  处不可导，但并不影响它在实际应用中的优异表现 (可以规定  $f'(0) = 0$ )，后面提到的 AlexNet 使用的就是 ReLU 激活函数。

### 1.3 深度学习的误差理论

在机器学习 (machine learning) 理论中，人们一般假设数据的真实生成过程 (generating process) 来自一个光滑的函数 (Tsybakov, 2008)，即

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad \varepsilon_i \stackrel{iid}{\sim} N(0, \sigma^2), \quad i = 1, \dots, n. \quad (1)$$

其中  $f : \mathbb{R}^D \rightarrow \mathbb{R}$  是  $\beta$  次可导的函数。在  $f$  的光滑假设下，很多非参数方法如 kernel methods, Gaussian processes, 样条方法等，以及深度神经网络 (deep neural network, DNN) 都可以达到 minmax 最优的泛化误差界 (generalization error bound)(Stone, 1982):

$$O\left(n^{-2\beta/(2\beta+D)}\right), n \rightarrow 0.$$

Imaizumi and Fukumizu (2018) 考虑了当  $f$  是分段光滑 (piecewise smooth) 函数的情况，此时  $f$  在分段区域的边界不可导甚至不连续。Imaizumi and Fukumizu (2018) 给出了这种情况下使用 ReLU 激活函数的 DNN 的泛化误差上界：

$$O\left(\max\left\{n^{-2\beta/(2\beta+D)}, n^{-\alpha/(\alpha+D-1)}\right\} (\log n)^2\right), n \rightarrow 0 \quad (2)$$

其中  $\alpha$  和  $\beta$  分别为  $f$  在边界和内部的可导次数 (smoothness degree),  $D$  是数据  $x_i$  的维度。Imaizumi and Fukumizu (2018) 证明了误差界(2)与分段光滑  $f$  下的 minimax 最优上界只相差了因子  $(\log n)^2$ . 因此当数据真实生成过程(1)中的  $f$  是分段光滑且满足一些假设条件下，其它方法很难达到比 DNN 更小的误差界 (Imaizumi and Fukumizu, 2018). 下表是 Imaizumi and Fukumizu (2018) 给出的使 DNN 达到误差界(2)所需的层数和参数个数：

ELEMENT	NUMBER
# OF LAYERS	$O(1 + \max\{\beta/D, \alpha/2(D-1)\})$
# OF PARAMETERS	$\Theta(n^{\max\{D/(2\beta+D), (D-1)/(\alpha+D-1)\}})$

## 1.4 使用神经网络识别手写数字

本节我们介绍神经网络的一个实战应用 — 识别手写的阿拉伯数字。这是一个非常成熟的项目，网上有现成的数据库 MNIST (包含 6 万个手写的数字图像) 和神经网络的程序代码 (<http://neuralnetworksanddeeplearning.com/chap1>). 下面这组手写的阿拉伯数字，可能是信封上的邮政编码也可能是支票上的钱数，如何教会计算机识别这些数字呢？

504192

想要让计算机进行处理，需要先把问题“数学化”。人们首先用几个正方形把各个数字分开，问题简化为：对包含一个手写数字的正方形区域，如下图所示，能不能让计算机识别是什么数字？



更进一步，忽略字的颜色，降低正方形的分辨率，只考虑一个  $28 \times 28 = 784$  个像素的图像。规定每一个像素值都在 0 到 1 之间，代表灰度的深浅，0 表示纯白，1 表示纯黑。此时每幅图片的输入就是 784 个 0-1 之间的数。

人们书写数字肯定有一定规律，但又说不清具体是什么规律，这种问题特别适合神经网络学习。建立一个三层的神经网络，如图7所示，第一层是输入层，对应要输入的 784 个像素值；第二层是隐藏层，有 15 个神经元；第三层是输出层，有 10 个神经元，对应 0-9 这 10 个数字。每个神经元都由输入、权重、bias 和输出组成。隐藏层中每一个神经元都要接收全部 784 个像素的输入，总共有  $784 \times 15 = 11760$  个权重和 15 个 bias 的值。第三层的每一个神经元都要跟第二层的所有 15 个神经元连接，总共有 150 个权重和 10 个 bias 的值。因此整个神经网络一共有 11935 个可调参数。

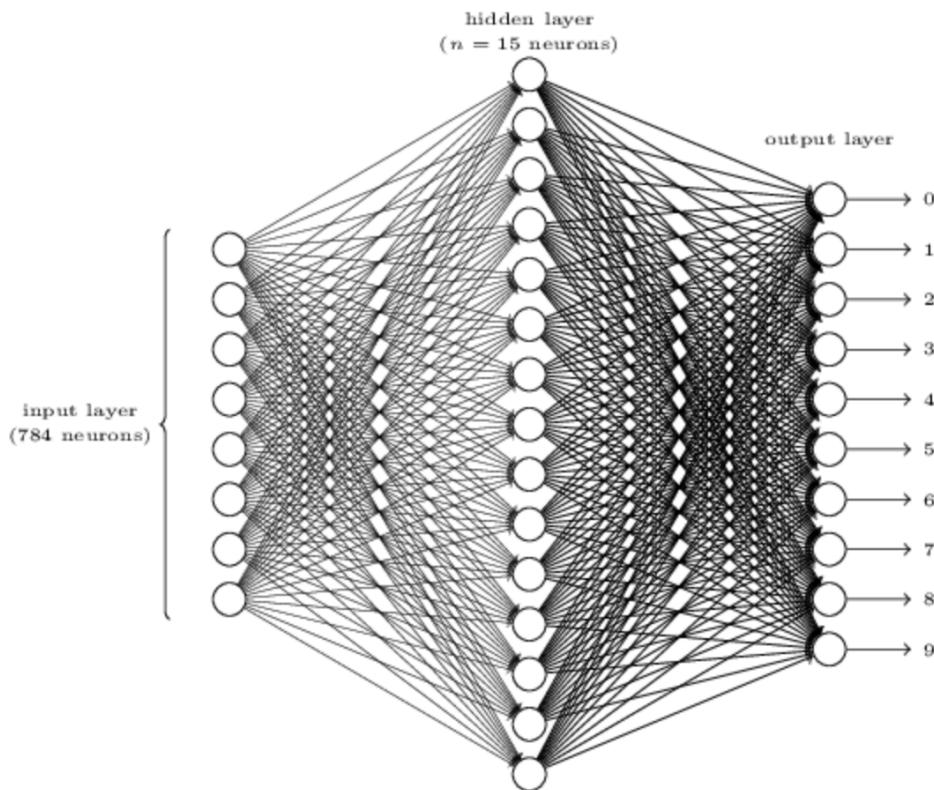


Figure 7: 一个三层的神经网络。Picture source: Michael Nielson

图7中神经网络的工作原理是：784 个像素值经过隐藏层和输出层两层神经元的处理后，输出层的那个神经元的输出结果最接近 1，就判断这是哪一个手写数字。为调整好神经网络的 11935 个参数，用 MNIST 中的 3 万个图像训练，训练的方法是之前介绍的误差反向传播，再用剩下 3 万个图像测试训练效果。测试结果表明这个简单的神经网络的识别准确率达到 95%.

### Remarks

1. 虽然神经网络能够做出出色的判断，但整个神经网络就像一个黑箱，我们能看到其中 11935 个参数的值，但是不知道这些数值体现了哪些规律。
2. 深度学习必须用大量的专门数据训练，且非常消耗资源。在上述例子中，只用两层 25 个神经元和  $28 \times 28$  的低分辨率图像，每次训练就要调整 11935 个参数。如果再多加一层，每层多用几个神经元，运算量就会大大增加，可能几万个图像就够了，应用级别的深度学习是海量的计算。其次，这样的神经网络训练好之后只能用于判断阿拉伯数字，不能判断英文字母或别的东西。

## 2 卷积神经网络

卷积神经网络 (Convolutional Neural Networks, CNN) 的成功使深度学习真正流行起来。下面这张彩色照片的分辨率是  $350 \times 263$ ，总共有 92050 个像素点，每个像素点用三个数来代表颜色，因此这张图需要用 27 万个数来描述。使用之前的神经网络识别这样的图，它的第二层每一个神经元都要有 27 万个权重参数。要想识别包括猫、狗、蓝天、草地等常见物体，它的输出层必须有上千个神经元才行。但是巨大的计算量还不是最大的难点，难点是神经网络中的参数越多，它需要的训练素材就越多，并不是任何照片都能用作训练素材，必须事先靠“人工”标记照片上的内容作为标准答案，这样才能给神经网络提供有效反馈。



斯坦福大学的李飞飞教授组织了一个叫 ImageNet 的机器学习图形识别比赛，从 2010 年开始每年举行一次，该比赛给参赛者提供一百万张图作为训练素材，其中每张图都经过人工标记，如下图所示。

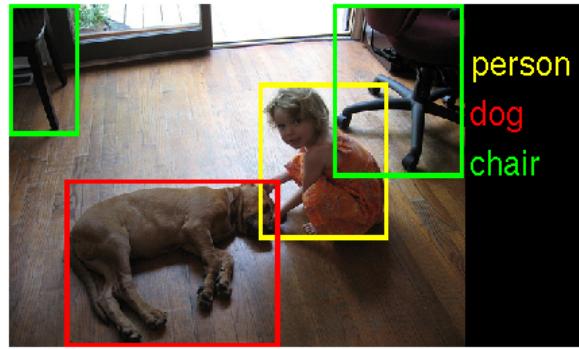


图8是 ImageNet 历届比赛冠军的成绩，可以看到，2010 年和 2011 年最好成绩的判断错误率 (classification error rate) 都在 26% 以上，但是 2012 年，错误率突然降到 16%，从此之后就是直线下降。2012 年的冠军是多伦多大学的一个研究组，使用的方法就是卷积神经网络 (Krizhevsky et al., 2012). 现在这篇论文被称为“AlexNet”.

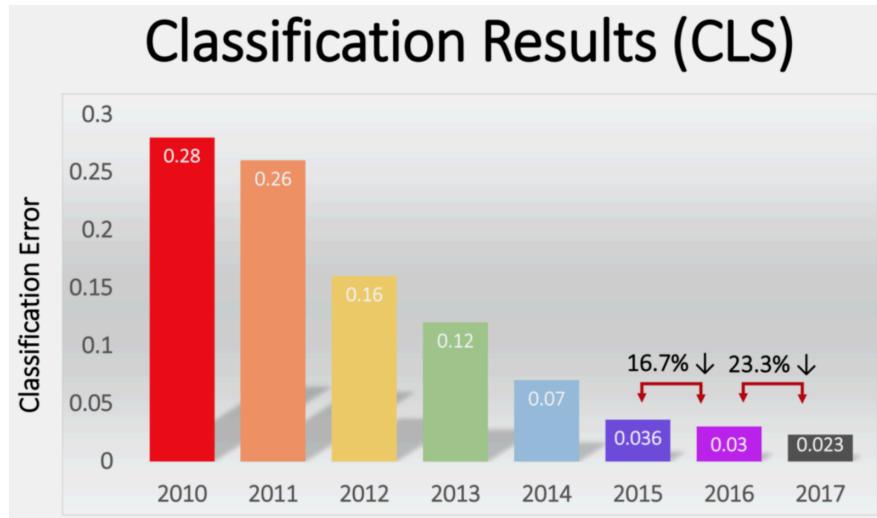


Figure 8: ImageNet 历届比赛冠军的成绩。Picture source: [image-net.org](http://image-net.org)

AlexNet 有一个“往哪看”和“看什么”的思路，这就与人脑的图像识别过程更相似，因为大脑并不是每次都把一张图中所有的像素放在一起考虑，比如让你找猫，你会从一张大图上一块一块地找，没必要同时考虑图片的左上角和右下角。AlexNet 的具体做法是在输入的像素层和最终识别物体的输出层之间加入几个“卷积层”。“卷积”是一种数学操作，在图像识别中可以理解成“过滤”或“提取特征”。考虑一个  $5 \times 5$  的 0-1 像素矩阵，和一个  $3 \times 3$  的“窗口”矩阵 (kernel)，如图9的第一行所示；对像素矩阵做卷积操作就是将窗口矩阵依次划过像素矩阵，在每个位置记录被窗口矩阵覆盖的元素的加权和，如图9的第二行所示。为防止图形结构被窗口矩阵拆散，还要保证窗口每次划过的区域有足够的重叠。AlexNet 的每一个卷积层只识别一种特定规模的图形模式，且后面一层只

在前面一层的基础上进行识别，这就解决了“往哪看”和“看什么”的问题。

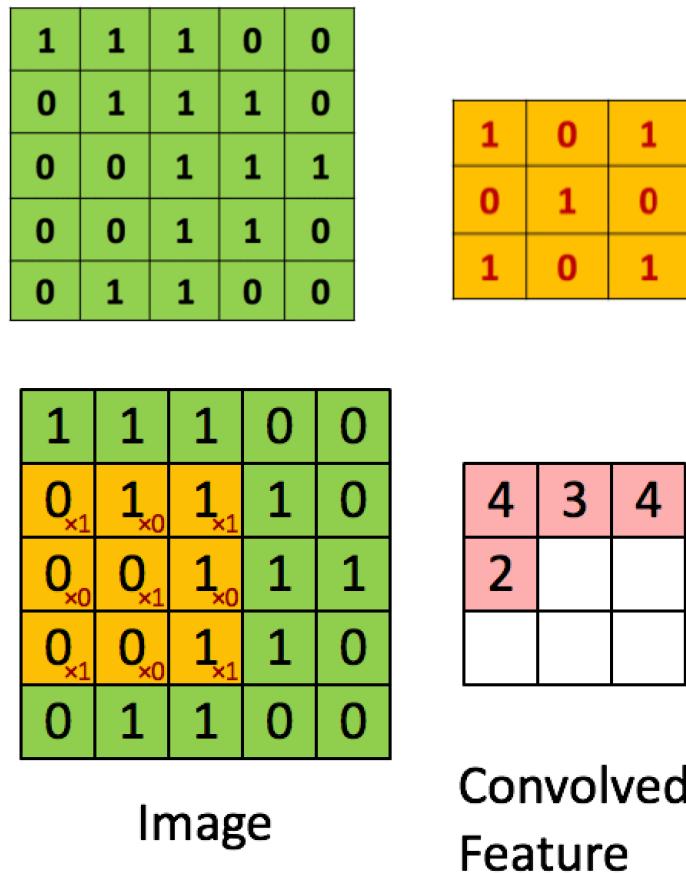


Figure 9: 对图像做卷积操作的过程。Picture source: [hackernoon.com](https://hackernoon.com/introduction-to-convolutional-neural-networks-1d-convolution-1f333a3a)

图10展示了一个人脸识别的卷积神经网络，它有三个卷积层，基本过程是：第一层先从像素点中识别一些小尺度的线条结构；第二层根据第一层识别出来的小尺度结构识别像眼睛、耳朵、嘴之类的局部器官；第三层根据这些局部器官识别人脸。

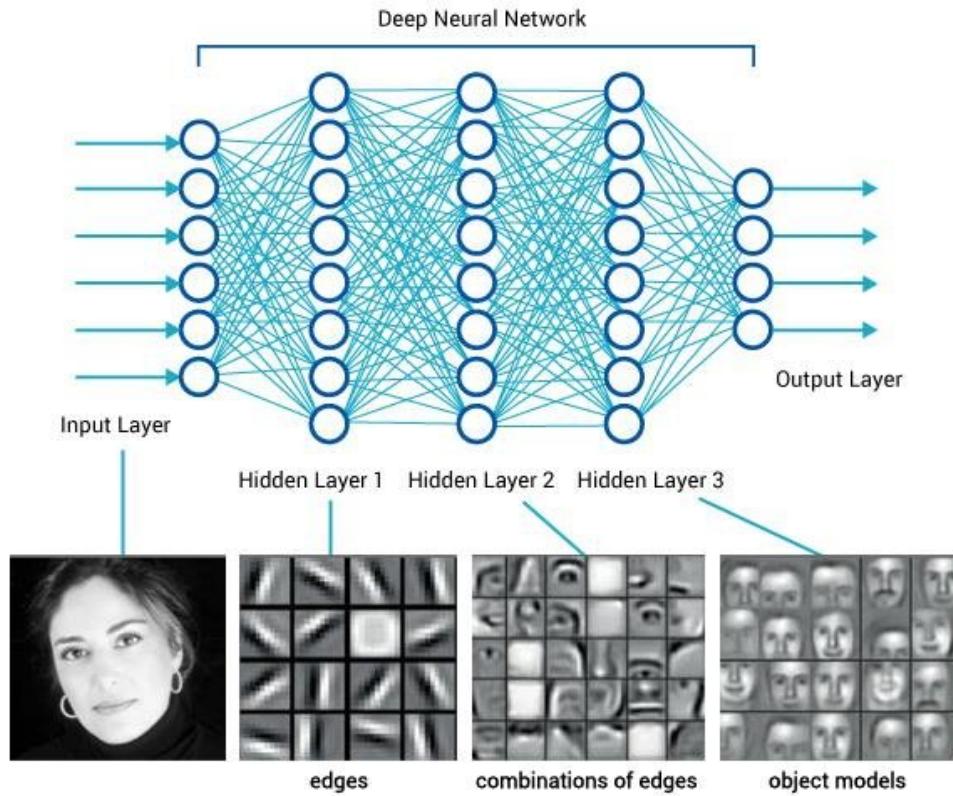


Figure 10: 一个人脸识别的卷积神经网络。Picture source: researchgate.net

AlexNet 的原始论文使用了 5 个卷积层，每一层由很多个“卷积核”(convolutional kernels)组成。第一层有 96 个卷积核，训练的结果如图11所示，每个卷积核有自己的神经网络，里面的每个神经元只负责原始图像中一个  $11 \times 11$  的小区块，考虑到 3 个颜色通道，每个神经元的权重参数有  $11 \times 11 \times 3 = 363$  个，且这些权重参数对所有神经元都是一样的，这就大大降低了运算量。



Figure 11: AlexNet 第一个卷积层训练出的 96 个  $11 \times 11 \times 3$  的卷积核。Picture source: Krizhevsky et al. (2012)

经过第一卷积层的过滤，我们看到的就不再是一个个的像素点，而是从一个个像素区块上提取的线条、颜色等特征。第二卷积层再从这些小特征上学习更复杂的结构，以此类推，一直到第五层。图12展示了 AlexNet 的完整结构，可以看到，卷积网络在第三层已经能找到一些纹理结构，到第五层已经能找出车轮、钟表等物体。在五个卷积层之外，AlexNet 还设置了三个完全连接层 (fully-connected layers)，每层有 4096 个神经元，用于识别更复杂的结构。

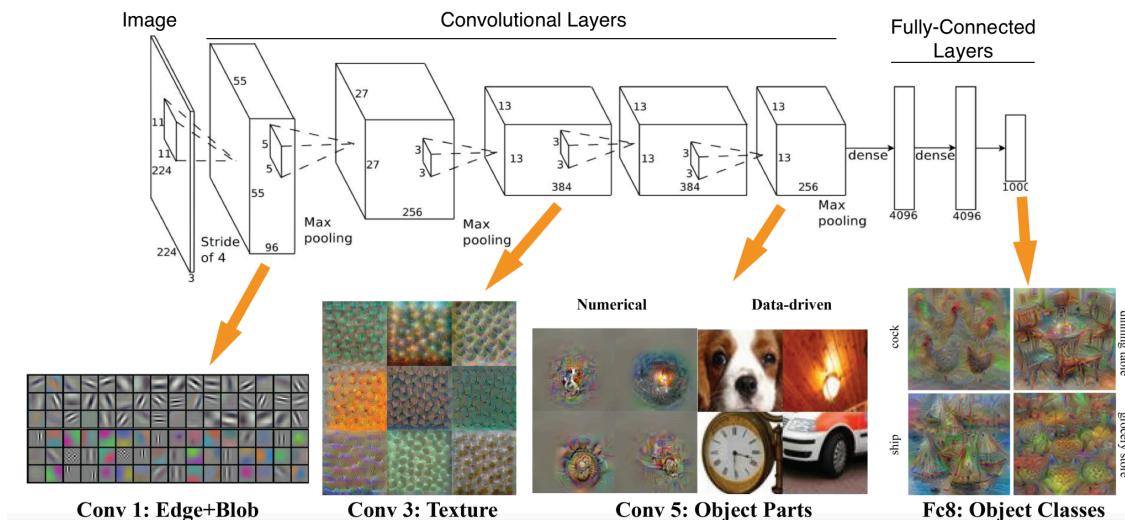


Figure 12: AlexNet CNN 的完整结构。Picture source: Daniel Jeffries

最后来看一下 AlexNet 识别图形的水平。ImageNet 的比赛规则是，对测试集中的每个图片，参赛者的算法可以给出五个答案，只要有一个判断跟标准答案一致就算判断正确。图13展示了 AlexNet 对三张比赛测试图片的识别结果，左图是 AlexNet 判断错误的一个例子，标准答案是“樱桃”，AlexNet 首先注意到了后面是一只达尔马提亚狗；中间图的标准答案是“蘑菇”，AlexNet 给出的第一判断是更精确的“伞菌”；右图的边缘有个红色的螨虫，它也被 AlexNet 正确识别出来了。

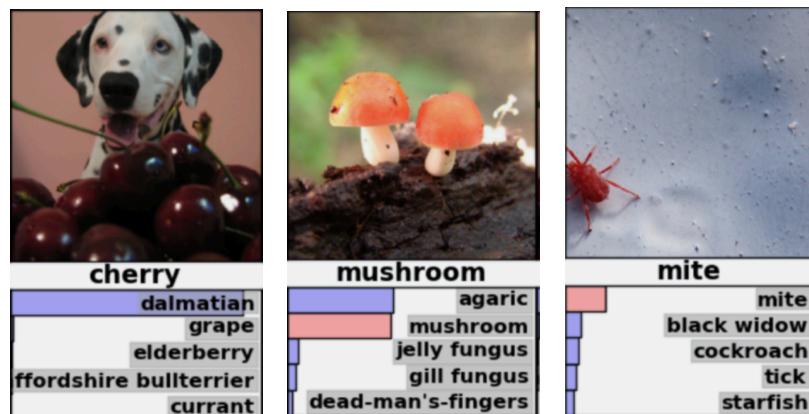


Figure 13: AlexNet 对 3 张测试图片的识别结果。Picture source: Krizhevsky et al. (2012)

意识到图形识别有多难，就能体会到 AlexNet 的识别水平有多神奇。2012 年之前深度学习还是机器学习中的“非主流”，现在是绝对的主流。但是深度学习也有局限性，它是完全基于经验的学习，这样的算法会有创造力吗？它能发现图像中从来没有被人命名过的物体吗？事实上，如果把图片中的物体放大一点，或者旋转一个角度、调整一下光线，卷积网络就不知道那是同一个东西，它必须得重新判断，深度学习还没有任何逻辑推理能力。

想要更详细地了解深度学习的概念和研究前沿，可以阅读复旦大学邱锡鹏教授的《神经网络与深度学习》(<https://nndl.github.io/>)。目前最流行的深度学习框架，即构建和训练深度学习模型的工具是 TensorFlow 和 PyTorch，前者由 Google 开发，后者广泛应用在 Facebook 研发中。*R* package `keras` 也可以构建和训练深度学习模型。此外，数据分析的竞赛平台 Kaggle 提供了很多公开数据库 (<https://www.kaggle.com/datasets>)，使数据科学爱好者能够接触到很多实际问题。

## References

- Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.
- Imaizumi, M. and Fukumizu, K. (2018). Deep neural networks learn non-smooth functions effectively. *arXiv preprint arXiv:1802.04474*.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.
- Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.
- Stone, C. J. (1982). Optimal global rates of convergence for nonparametric regression. *The annals of statistics*, pages 1040–1053.
- Tsybakov, A. B. (2008). *Introduction to nonparametric estimation*. Springer Science & Business Media.