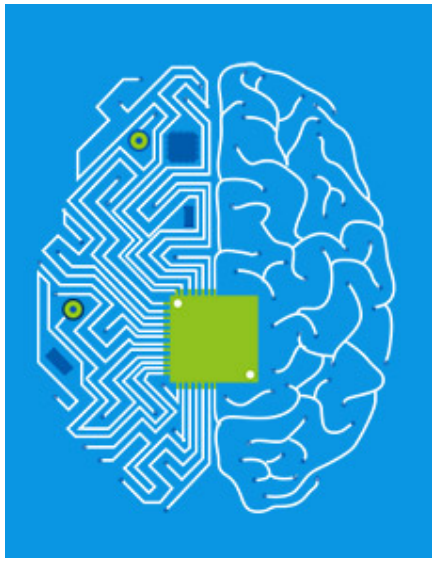
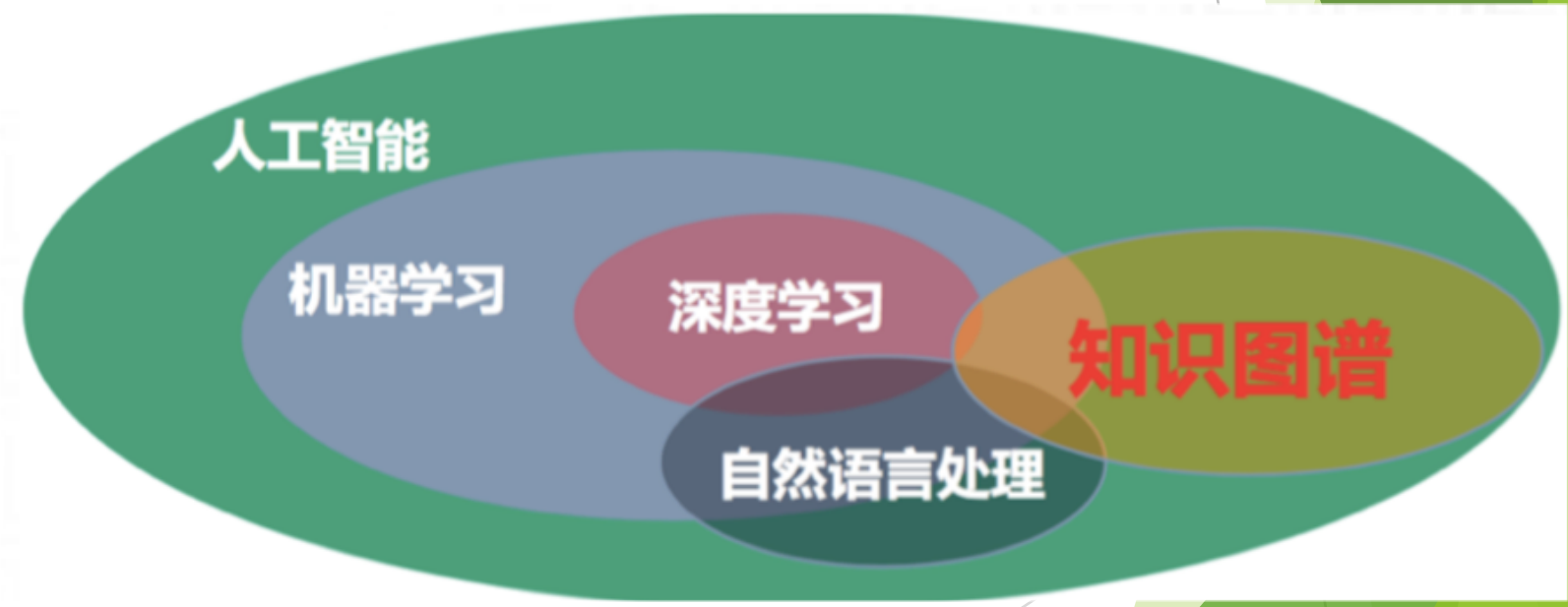
深度学习-Introduction

Guangrui Qian

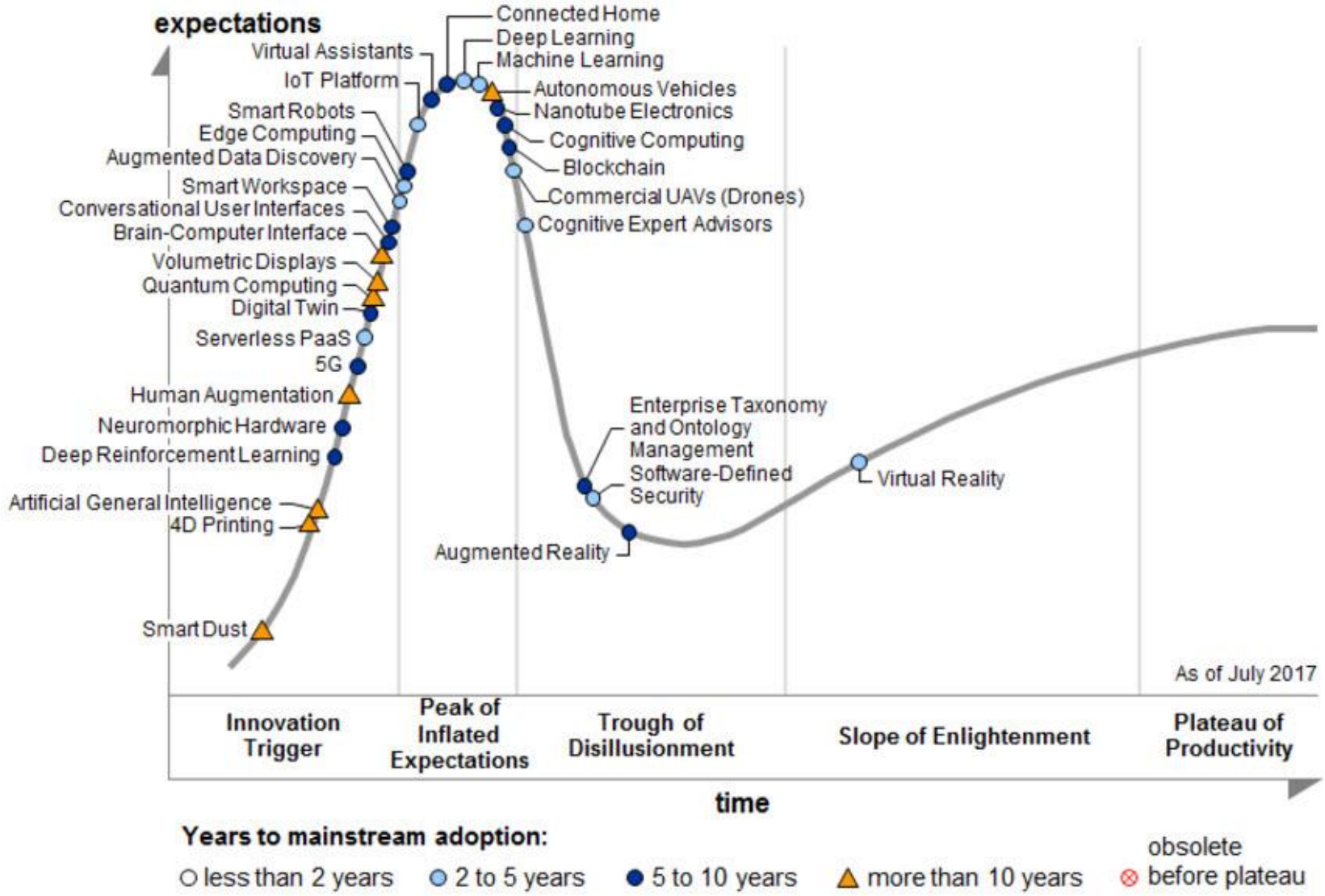
人工智能学 & 深度学习

人工智能学(Artificial Intelligence，简称AI)，是计算机科学的一个分支，是对人的意识、思维的信息过程的模拟。垃圾邮件过滤、谷歌翻译、地图导航等，都属于经典的人工智能应用。

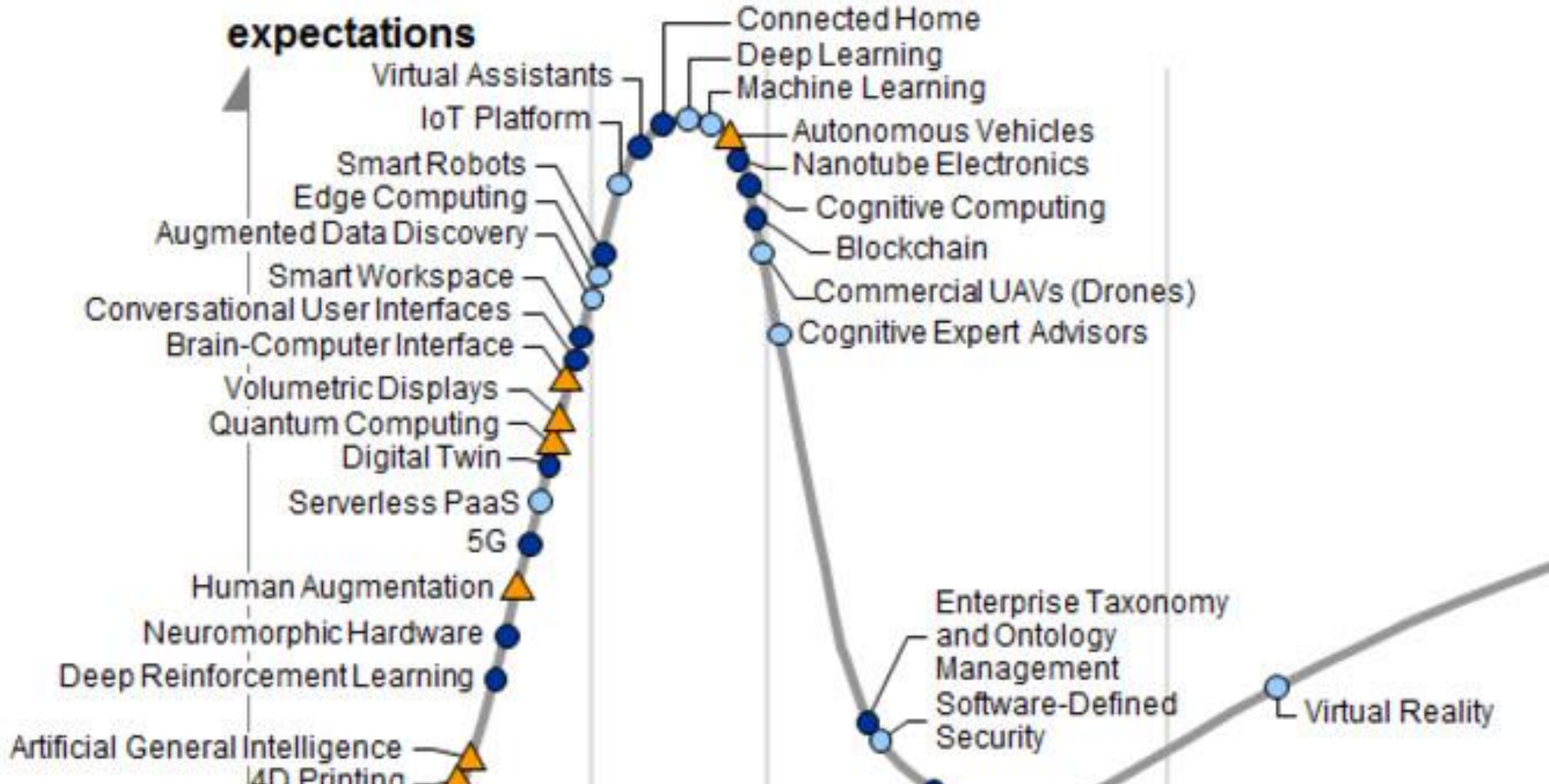




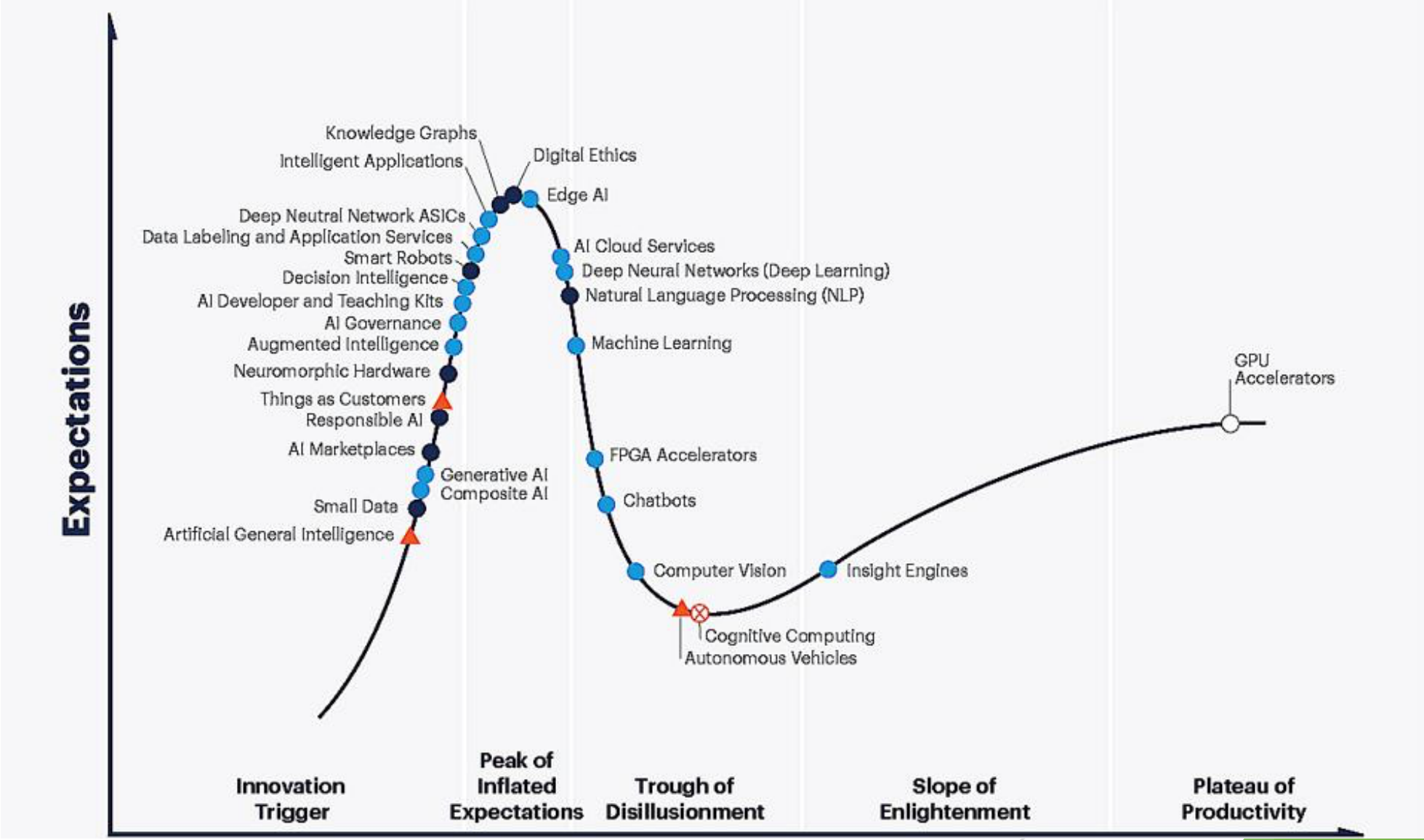
Hyper Cycle for Emerging Technologies 2017



Hyper Cycle for Emerging Technologies 2017



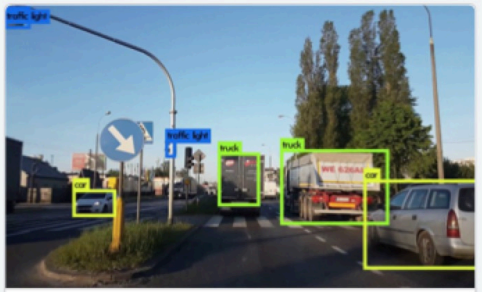
Hyper Cycle for Emerging Technologies 2020



深度学习常见场景

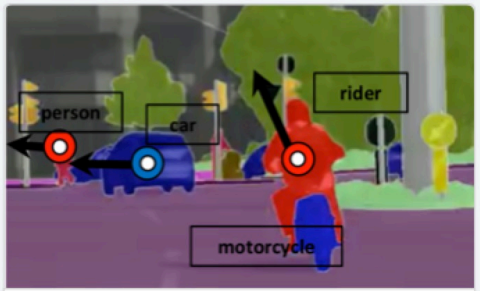
图像分类

根据图像的语义信息将不同的类型的图像区分开来。



图像目标检测

检测出图像中指定的目标区域，支持多目标检测。



图像分割

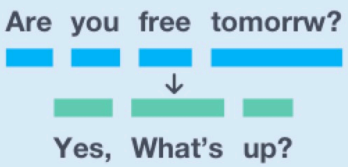
将一副图像分成若干互不重叠的子区域，使得每个子区域相似，不同子区域相异。

图像检索

对给定查询图像，搜索与之在视觉或语义上相似的图像，即以图搜图。

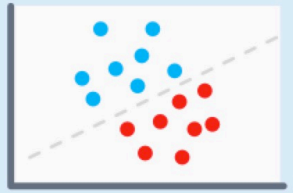
文本分类

根据文本的语义信息将多文本划分类别，应用于情感分析等。



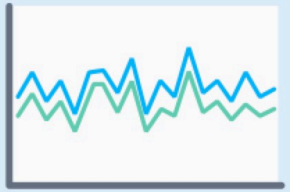
文本序列分析

解决序列到序列的有监督文本任务，如机器翻译、对话机器人、命名实体识别等。



数值分类

根据特征化的数值信息，对多条样本记录进行分类。



数值预测

根据有时间序列性且特征化的数值信息，预测未来时点或时段的目标数值。

人工智能市场产业链场景

人工智能深度学习主要方法

深度学习主要元素

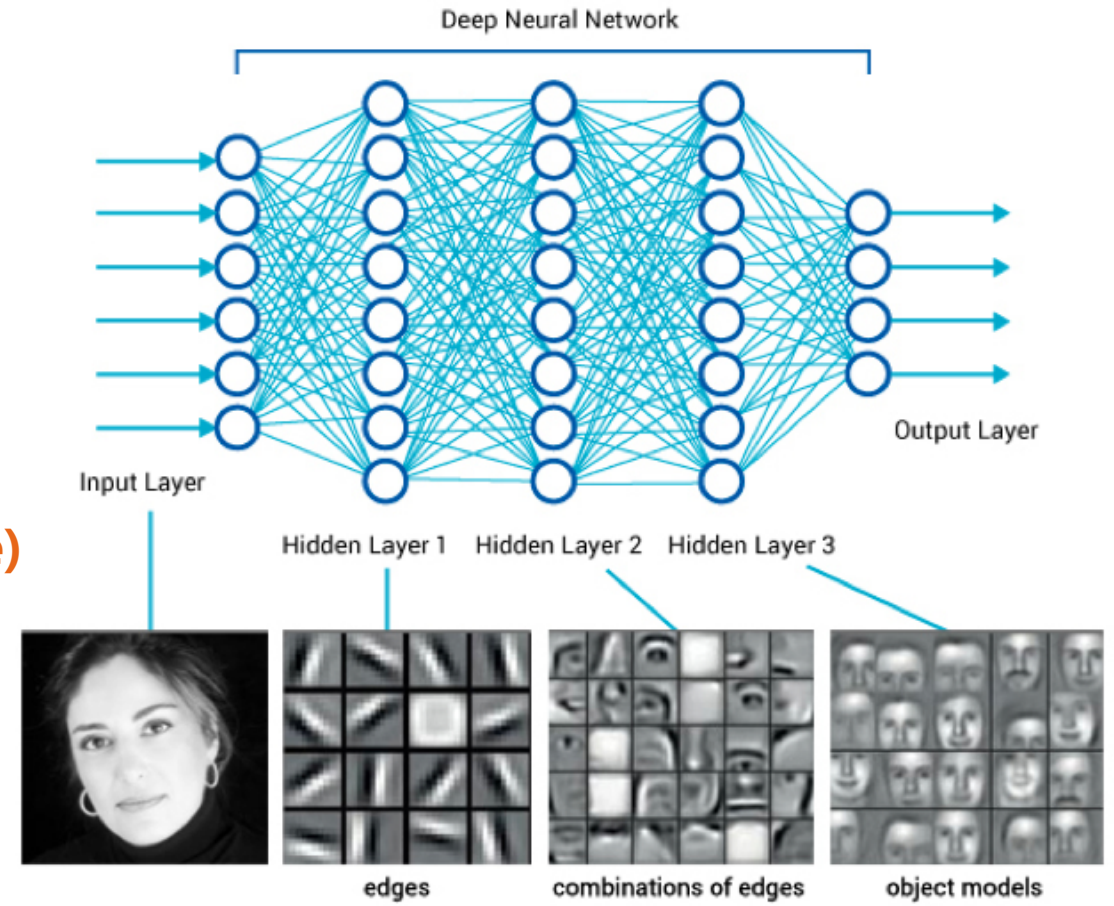
- 数据
- 算力
- 模型

- ❖ **数据收集(Collect)**
- ❖ **模型训练(Training)**
- ❖ **模型推理(Inference)**

深度学习主要方向

- 监督学习
- 迁移学习
- 非监督学习
- 强化学习

经济效益递减



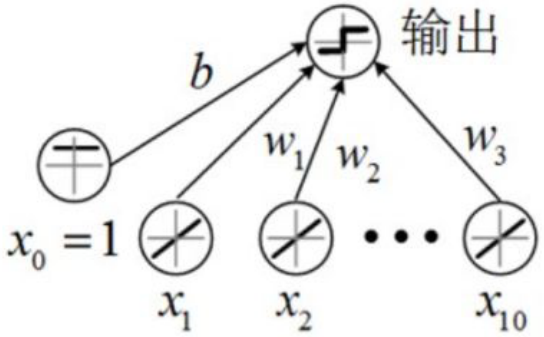
人工智能深度常见框架

国产深度学习框架

业界深度学习框架

神经网络发展历程 - 1

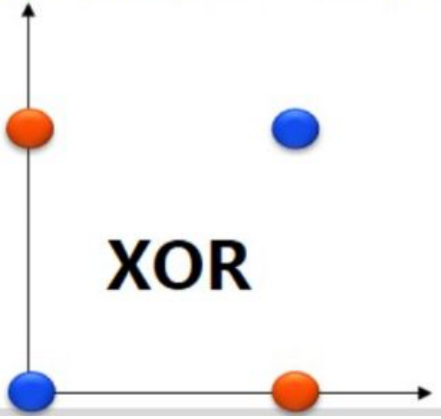
- **第一代神经网络：感知器(1950s)**

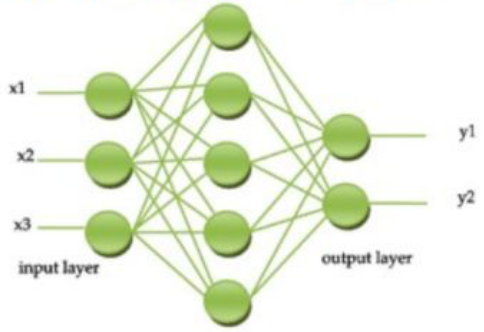


$$f(x)=WX+b$$
$$=w1x1+w2x2+...xnxn+b$$

未能解决线性不可分问题

- **第二代神经网络：多层感知器(MLP)(1980s)**



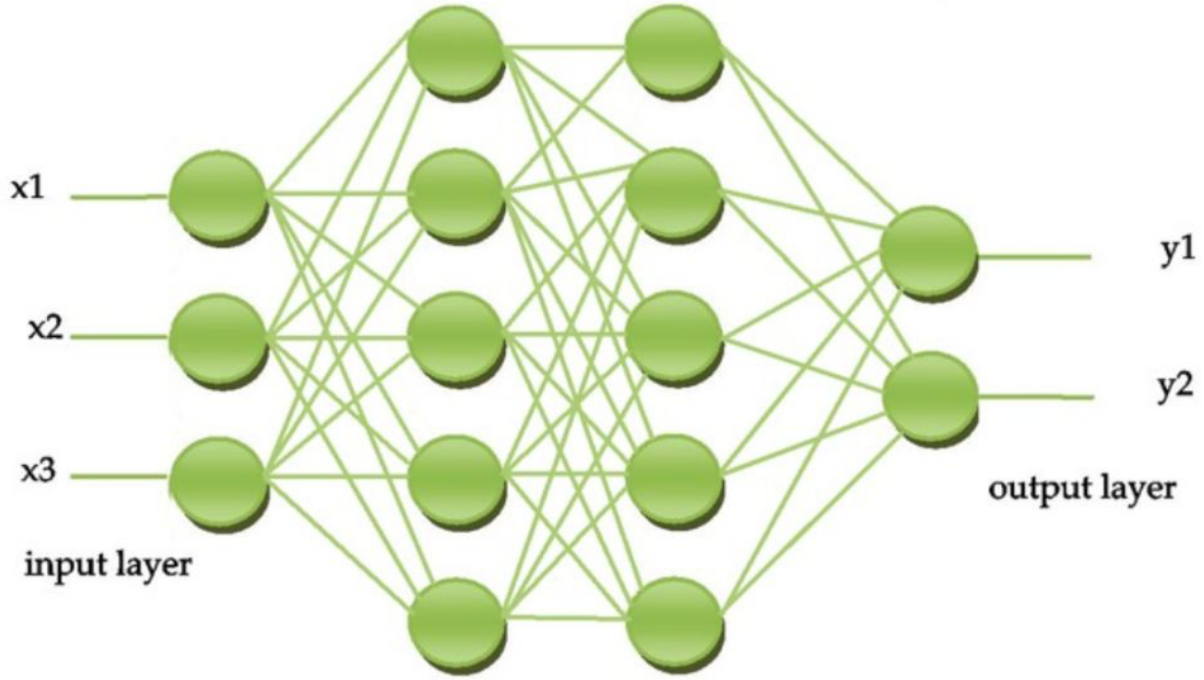


多层感知器能够解决线性不可分问题

神经网络发展历程 - 2

- 第三代神经网络：深度神经网络（2010s）

DNN

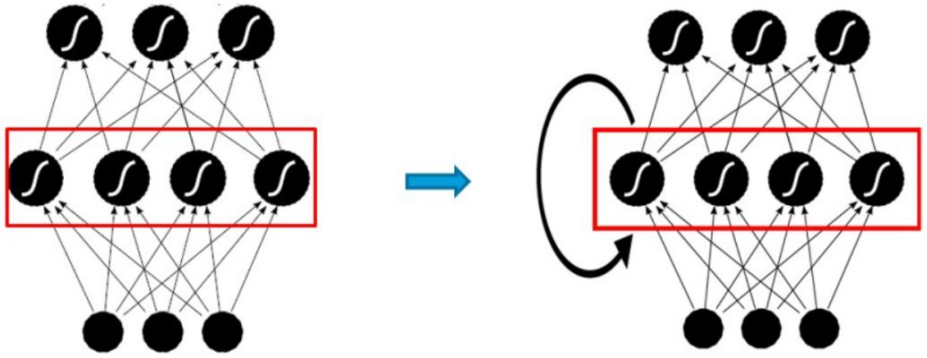


仅仅是多了一些层，仅此而已

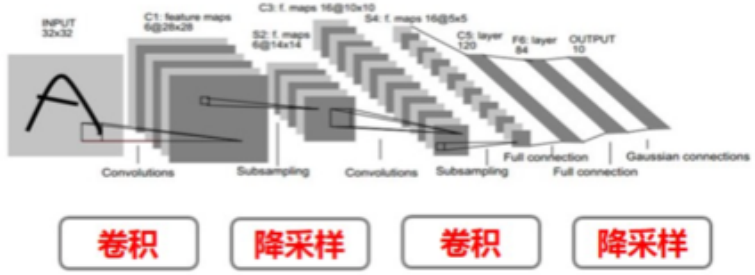
但效果确实很神奇

神经网络发展历程 - RNN & CNN

RNN



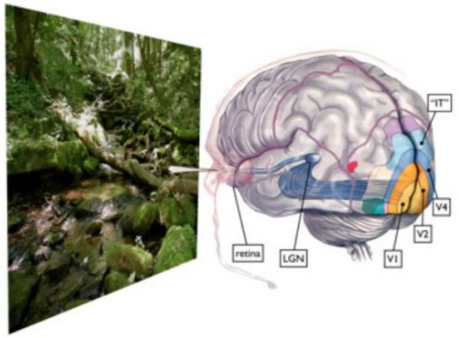
仅仅是多了一些“反馈环”
(带记忆功能的内部状态，特别适用于文本)
使得效果更加神奇（静态→动态）

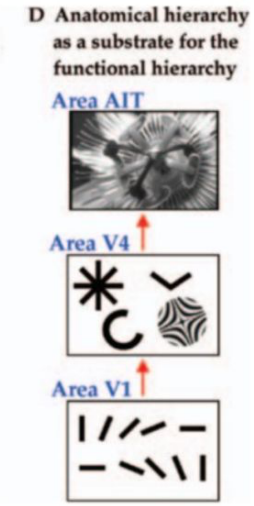


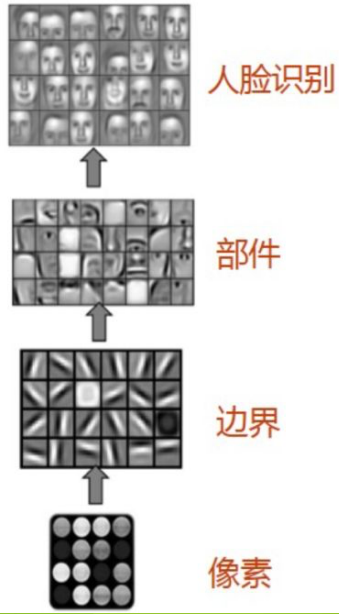
模拟人类视觉系统
多维空间信息的综合利用

CNN

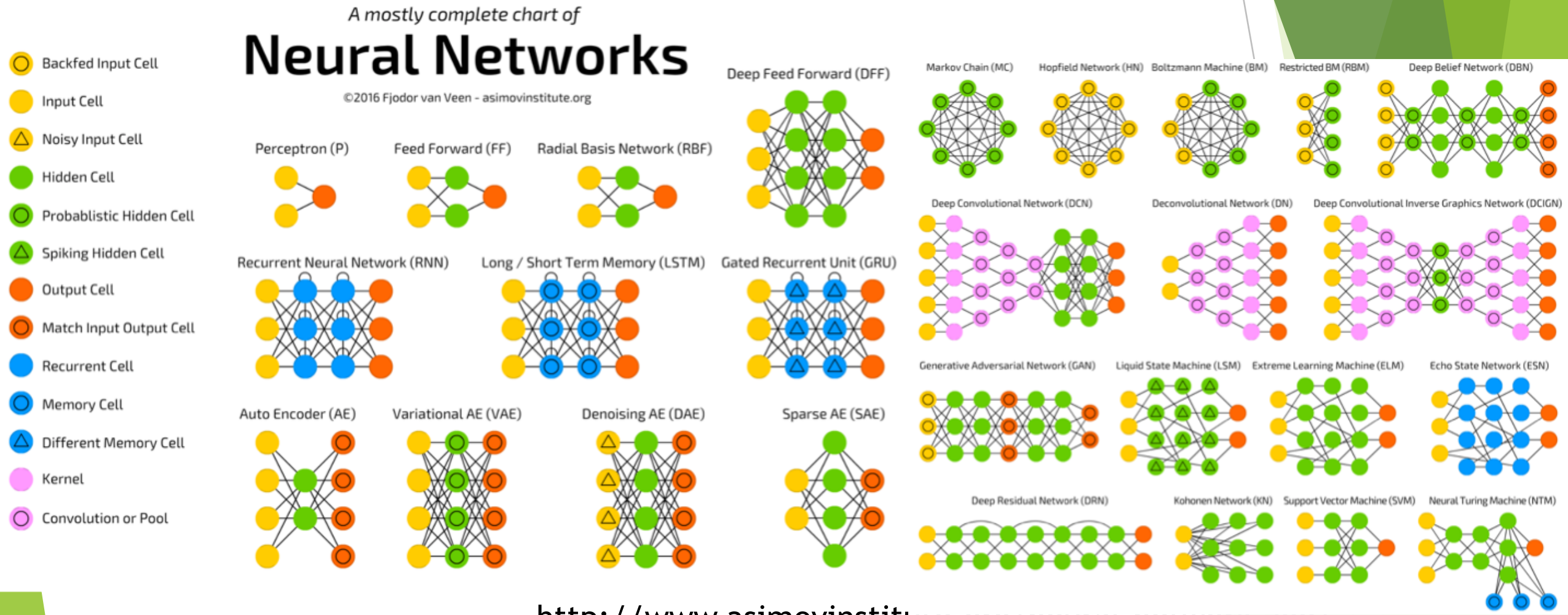
各层次所提取特征
具有明确意义





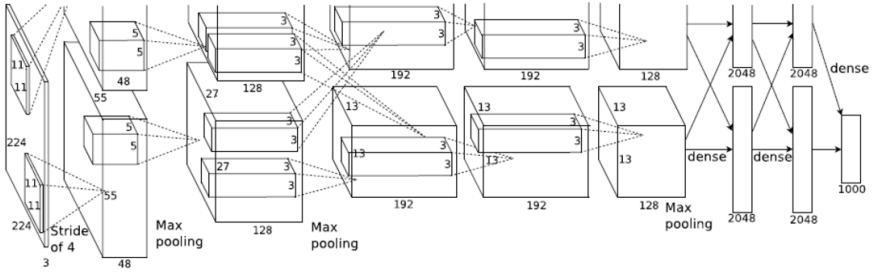


深度学习模型



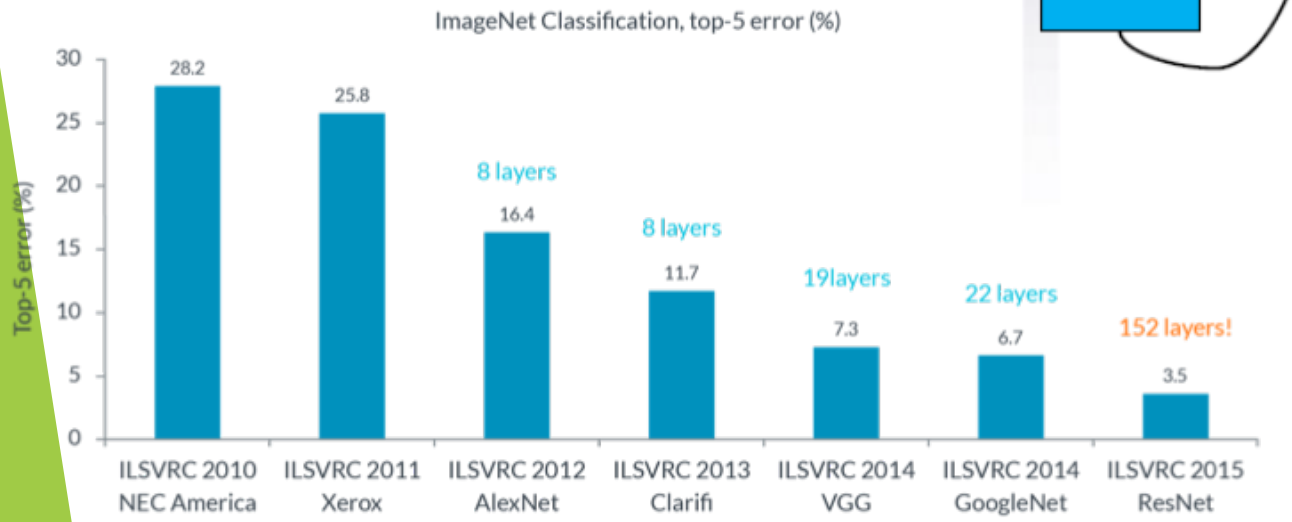
http://www.asimovinstitute.org/neural-network-2007

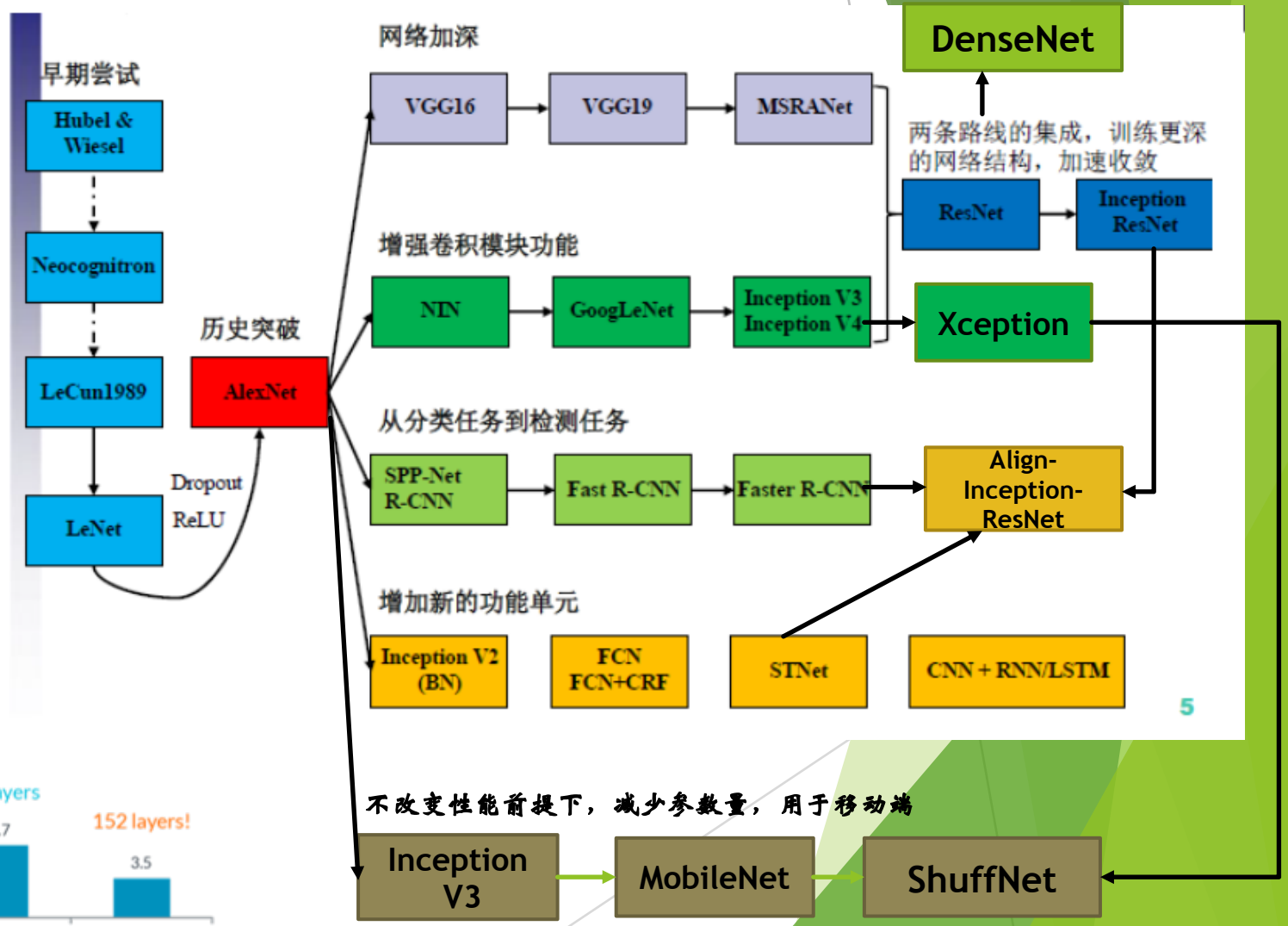
深度学习网络发展-1



AlexNet Framework

- 6亿3000万个连接，6000万个参数
- 5个卷积层，3个池化层和3个全连接层，11层网络





深度学习网络发展-2

- 卷积神经网络（CNN）
- 循环神经网络（RNN）
- 注意力机制网络（Attention in DL）
- 生成对抗网络（GAN）
- 深度强化学习（DQN）
- 图神经网络（GNN）

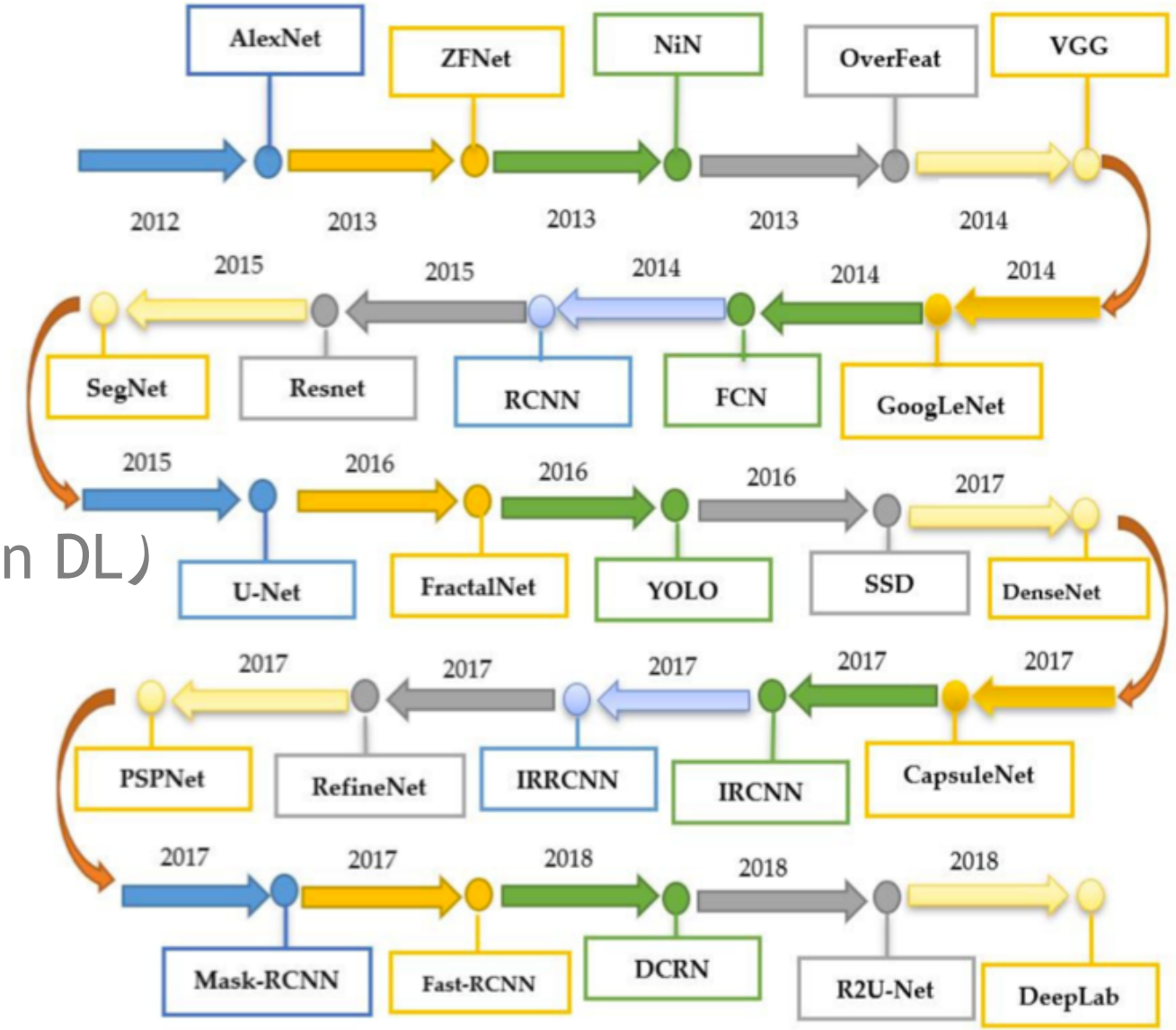
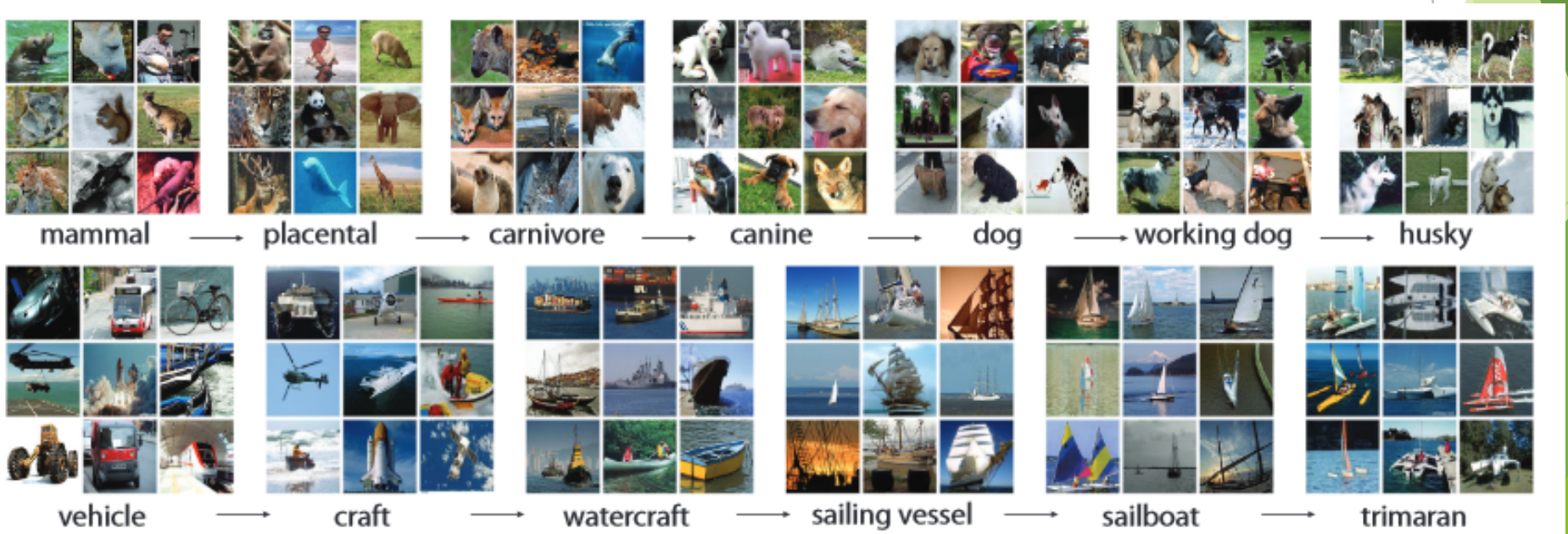


Figure 3. Summary of the evolution of various deep learning models from 2012 until now.

Plant Disease Detection and Classification by Deep Learning, October 2019 DOI: 10.3390/plants8110468

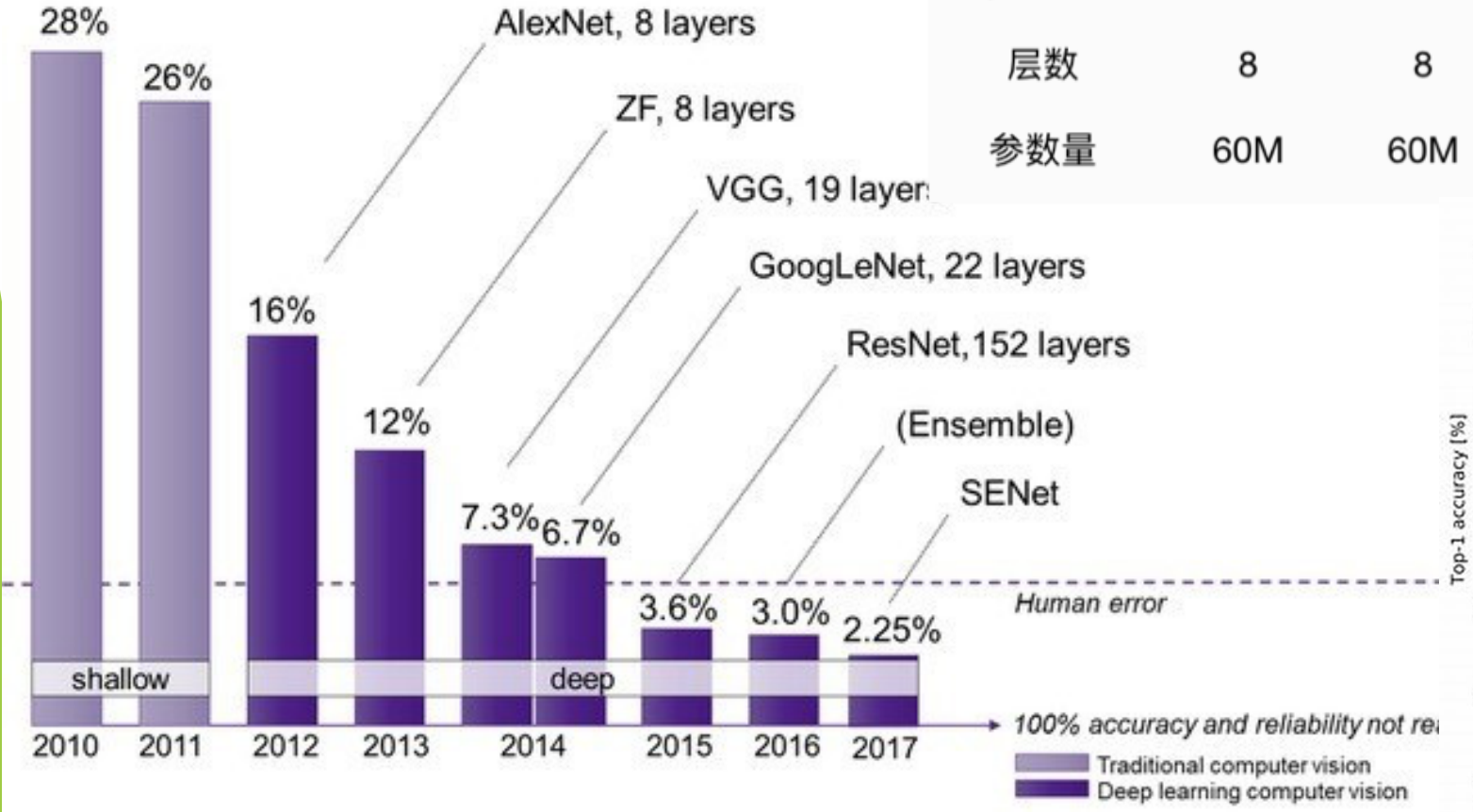
ImageNet数据集

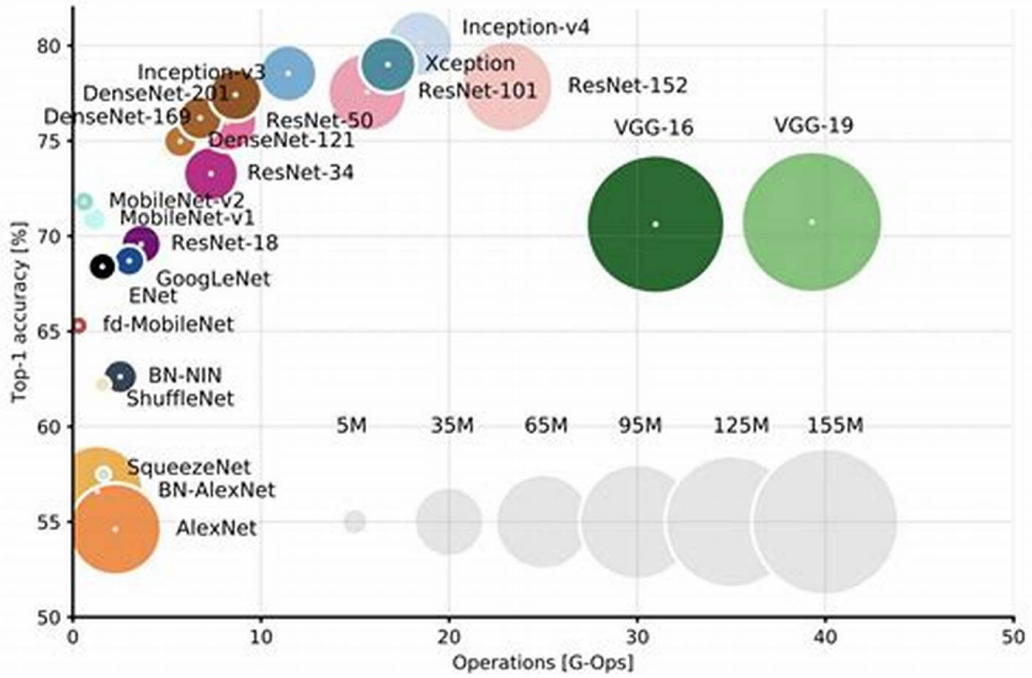
- ImageNet: http://www.image-net.org/
  - World's largest open-source image database
  - More than 10 million images in 2010
  - 20000+ synsets
  - images available with independent URL



ImageNet深度学习网络的性能发展

年份	2012	2013	2014	2014	2015	2016	2017
网络	AlexNet	ZFNet	VGGNet	GoogLeNet	ResNet	ResNeXt	SENet
Top5错误率	15.32%	13.51%	7.32%	6.67%	3.57%	3.03%	2.25%
层数	8	8	16	22	152	152	154
参数量	60M	60M	138M	7M	60M	44M	67M





机器智能学习实践应用所面临的挑战

环境框架搭建

TensorFlow™ Caffe jupyter mxnet ubuntu CentOS redhat CUDA NVIDIA

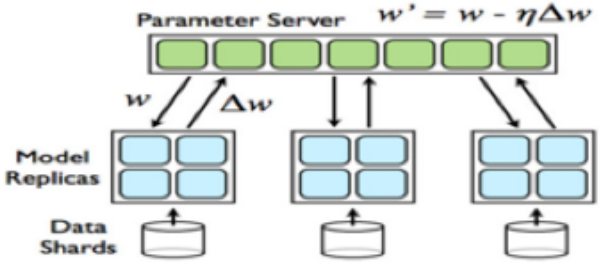
集成化统一开发环境的搭建

操作系统/驱动兼容安装配置

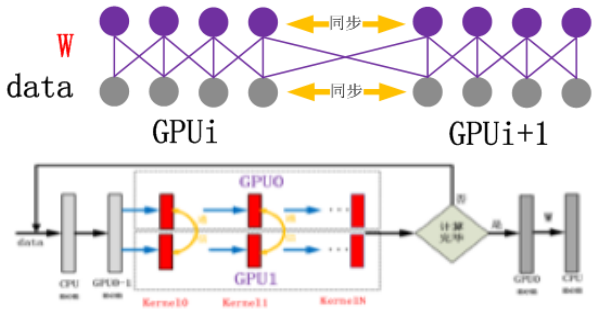
依赖计算资源:CUDA、GPU

模型训练加速

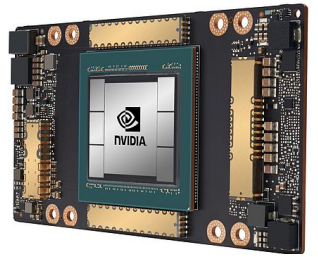
数据并行

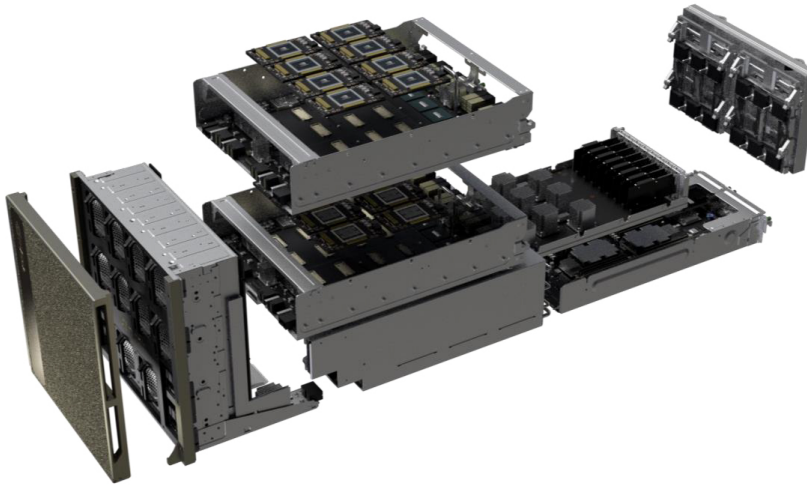


模型并行



GPU硬件平台

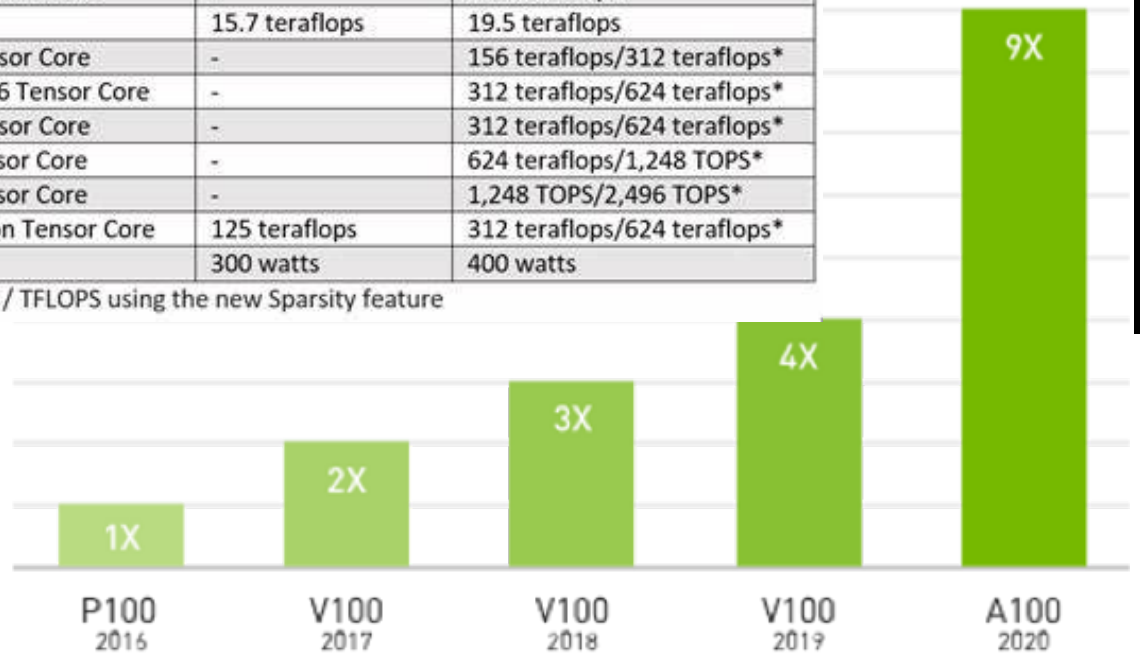


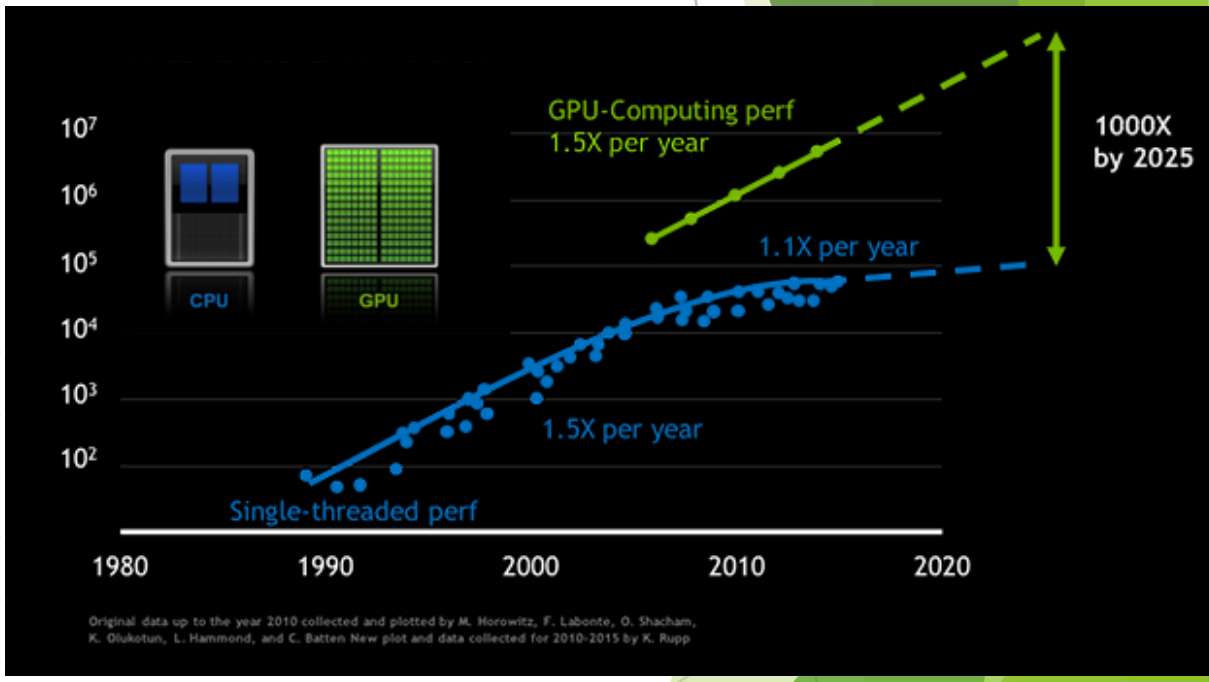


NVIDIA GPU Performance boost

Nvidia Datacenter GPU	Nvidia Tesla V100	Nvidia A100
GPU codename	GV100	GA100
GPU architecture	Volta	Ampere
Launch date	May 2017	May 2020
GPU process	TSMC 12nm	TSMC 7nm
Die size	815mm2	826mm2
Transistor Count	21.1 billion	54 billion
FP64 CUDA cores	2,560	3,456
FP32 CUDA cores	5,120	6,912
Tensor Cores	640	432
Streaming Multiprocessors	80	108
Peak FP64	7.8 teraflops	9.7 teraflops
Peak FP64 Tensor Core	-	19.5 teraflops
Peak FP32	15.7 teraflops	19.5 teraflops
Peak FP32 Tensor Core	-	156 teraflops/312 teraflops*
Peak BFLOAT16 Tensor Core	-	312 teraflops/624 teraflops*
Peak FP16 Tensor Core	-	312 teraflops/624 teraflops*
Peak INT8 Tensor Core	-	624 teraflops/1,248 TOPS*
Peak INT4 Tensor Core	-	1,248 TOPS/2,496 TOPS*
Mixed-precision Tensor Core	125 teraflops	312 teraflops/624 teraflops*
Max TDP	300 watts	400 watts

*Effective TOPS / TFLOPS using the new Sparsity feature





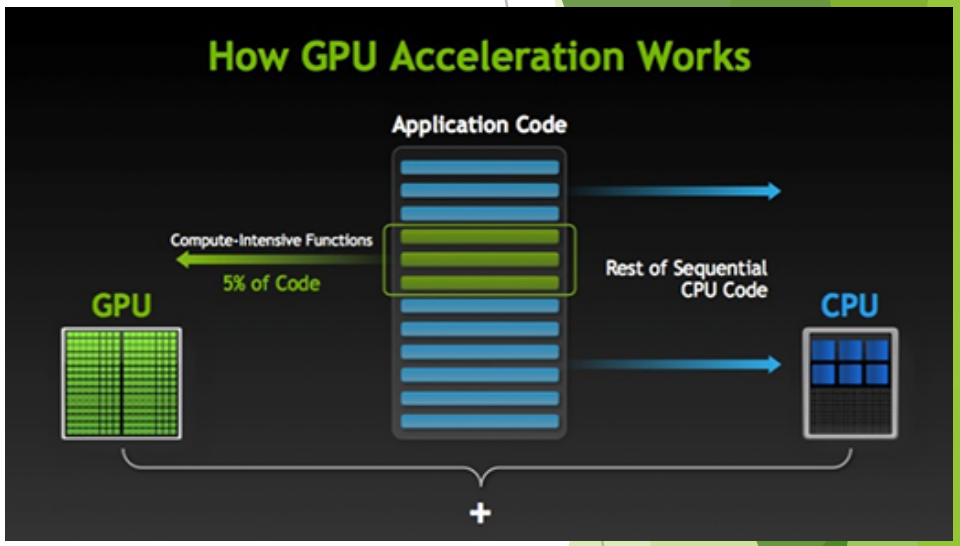
NVIDIA GPU

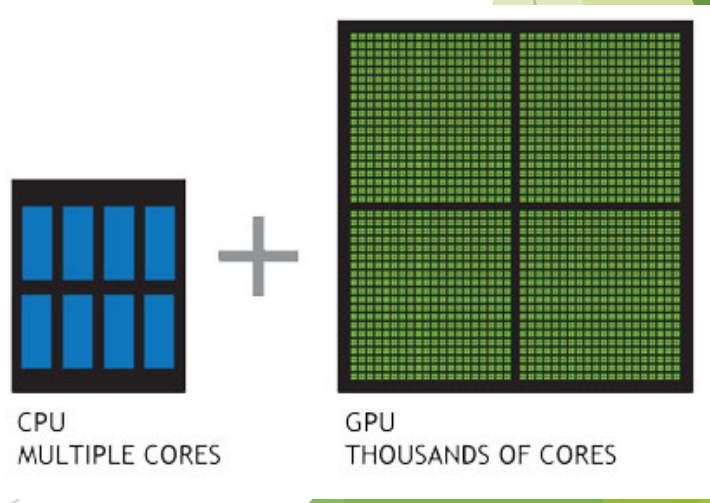
- GPU:Graphics Processing Unit,是利用一颗图形处理器(GPU)以及一颗 CPU 来加速科学、工程以及企业级应用程序。NVIDIA CUDA 并行计算平台可提供若干简单的 C 和 C++ 扩展程序，这些扩展程序能够表达细致与粗略的数据和任务并行机制。

- CPU 由专为顺序串行处理而优化的几个核心组成。另一方面，GPU 则由数以千计的更小、更高效的核心组成，这些核心专为同时处理多任务而设计。

- **GPU 拥有数以千计的核心，可高效地处理并行任务**

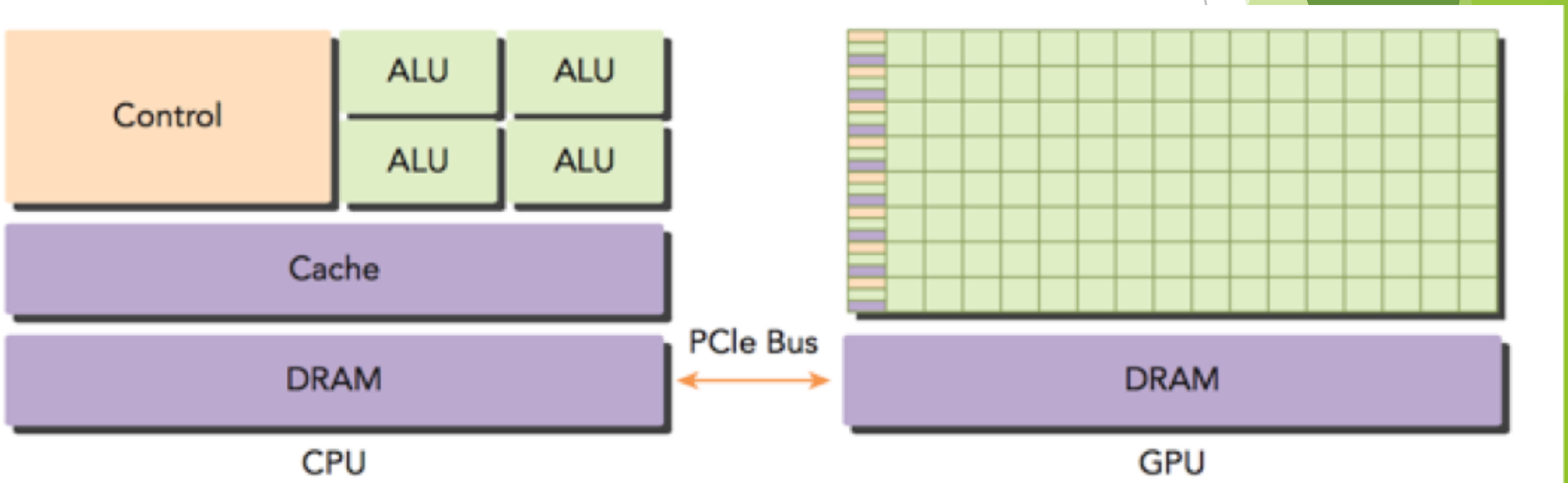
- NVIDIA Ampere 是全球快、效的高性能计算 GPU架构。新特性主要包括SMX，Dynamic Parallelism和Hyper-Q技术。





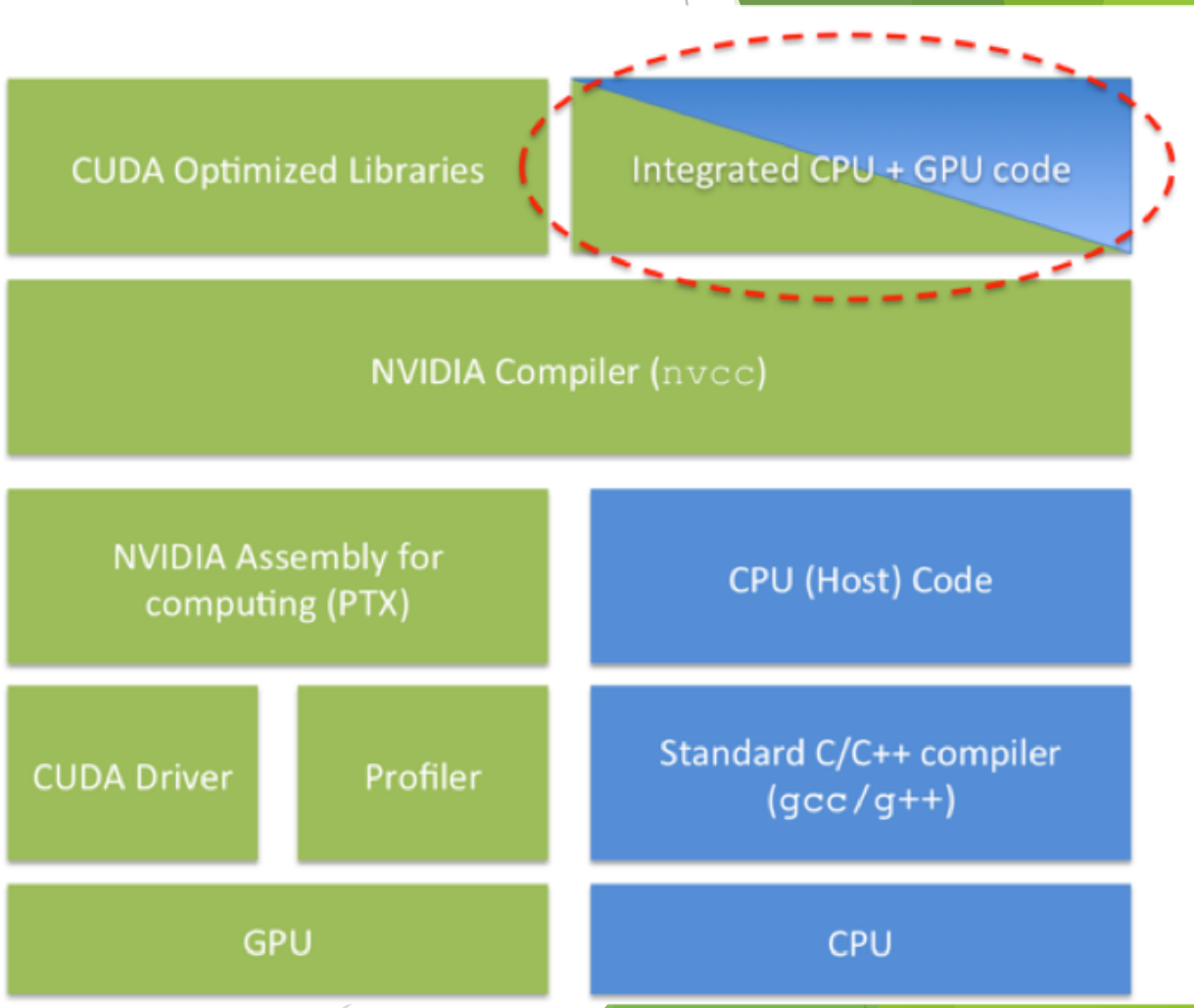
CUDA programming (Compute Unified Device Architecture)

- 在 CUDA 的架构下，一个程序分为两个部份：host 端和 device 端。Host 端是指在 CPU 上执行的部份，而 device 端则是在显示芯片上执行的部份。Device 端的程序又称为“kernel”。通常 host 端程序会将数据准备好后，复制到显卡的内存中，再由显示芯片执行 device 端程序，完成后再由 host 端程序将结果从显卡的内存中取回。



CUDA编程

- It is a model that combines both hardware and software.
- GPU has to be CUDA-enabled
- Through CUDA, you can program GPUs using **C, C++, Fortran, Java, Python,** and more
- **HOST(CPU)+Device(GPU)**



Single Float in Deep Learning

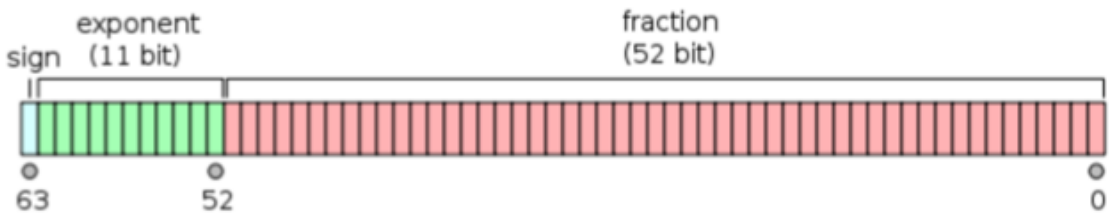
- 常用的浮点数有双精度和单精度。除此之外，随着深度学习的发展，出现了半精度的浮点数。

- 双精度64位，单精度32位，半精度是16位。
- 半精度是英伟达在2002年提出来的，双精度和单精度是为了计算，而半精度更多是为了降低数据传输和存储成本。

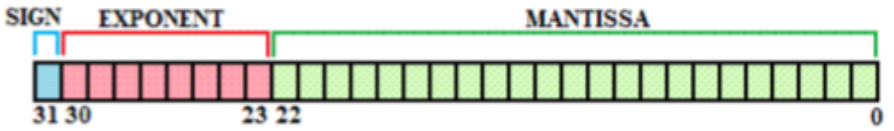
- 很多场景对于精度要求不高，例如分布式深度学习里面，如果用半精度的话，比起单精度来可以节省一半传输成本。考虑到深度学习的模型可能会有几亿个参数，使用半精度传输是非常有价值的。

类型	大小	备注
int8	1个字节	又名Byte
Int16	2个字节	又名Short
Int32	4个字节	又名Int
int64	8个字节	又名long long
Float32	4个字节	单精度浮点数，又名float
float16	2个字节	半精度浮点数

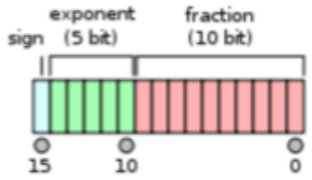
双精度浮点数



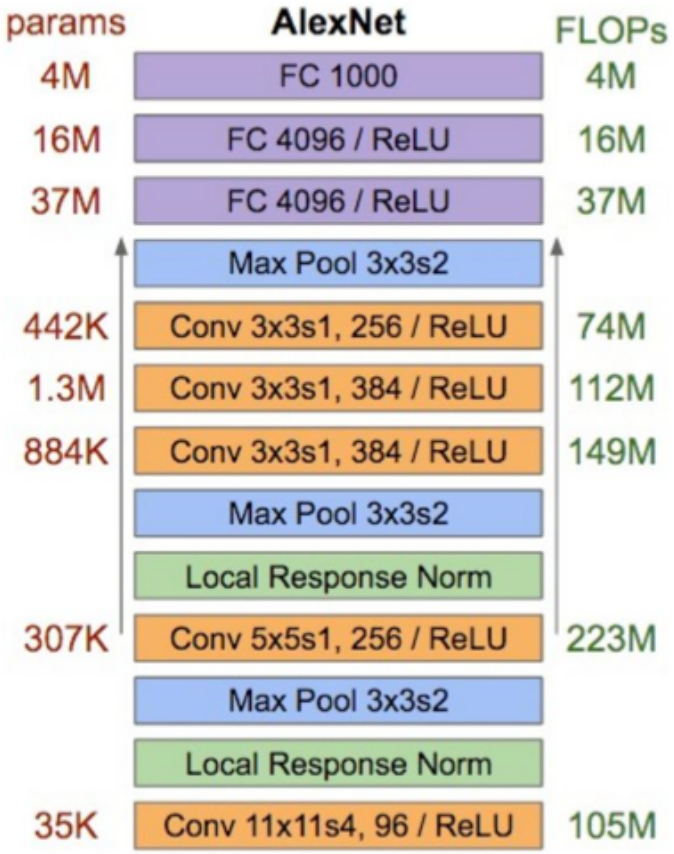
单精度浮点数

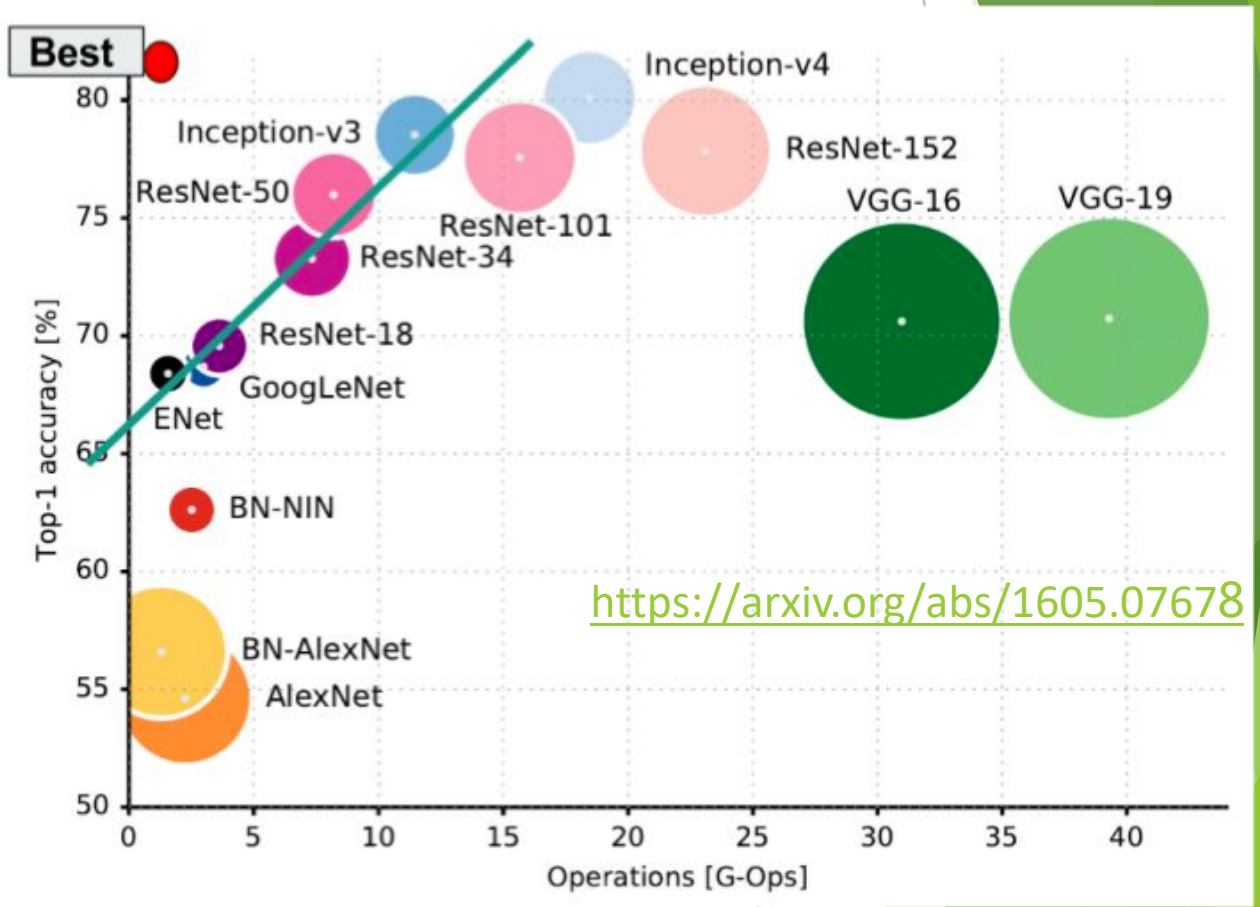


半精度浮点数



常用模型 显存/计算复杂度/准确率



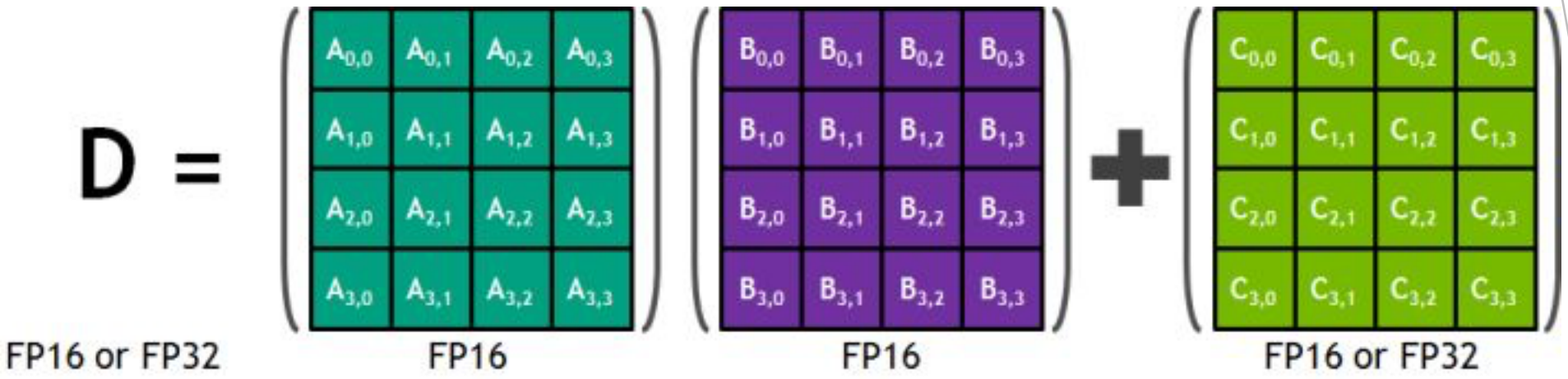


https://arxiv.org/abs/1605.07678

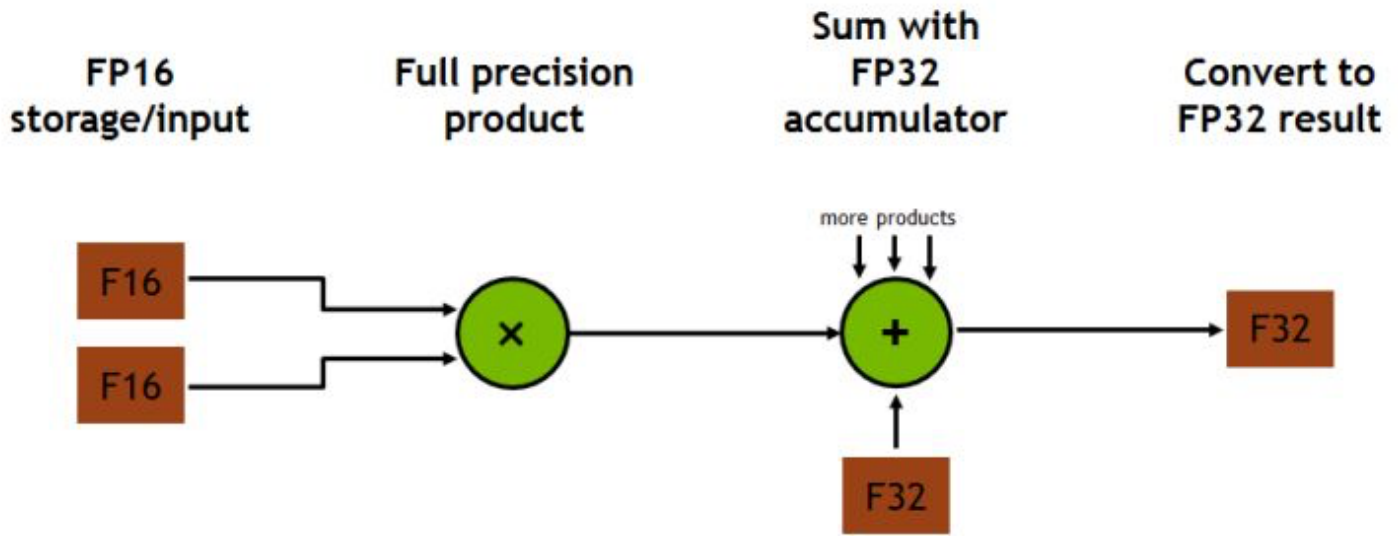
AlexNet的分析图，左边是每一层的参数数目（不是显存占用），右边是消耗的计算资源。

横座标是计算复杂度（越往右越慢，越耗时），纵座标是准确率（越高越好），圆的面积是参数数量（不是显存占用），参数量越多，保存的模型文件越大。

NVIDIA Tensor Core技术



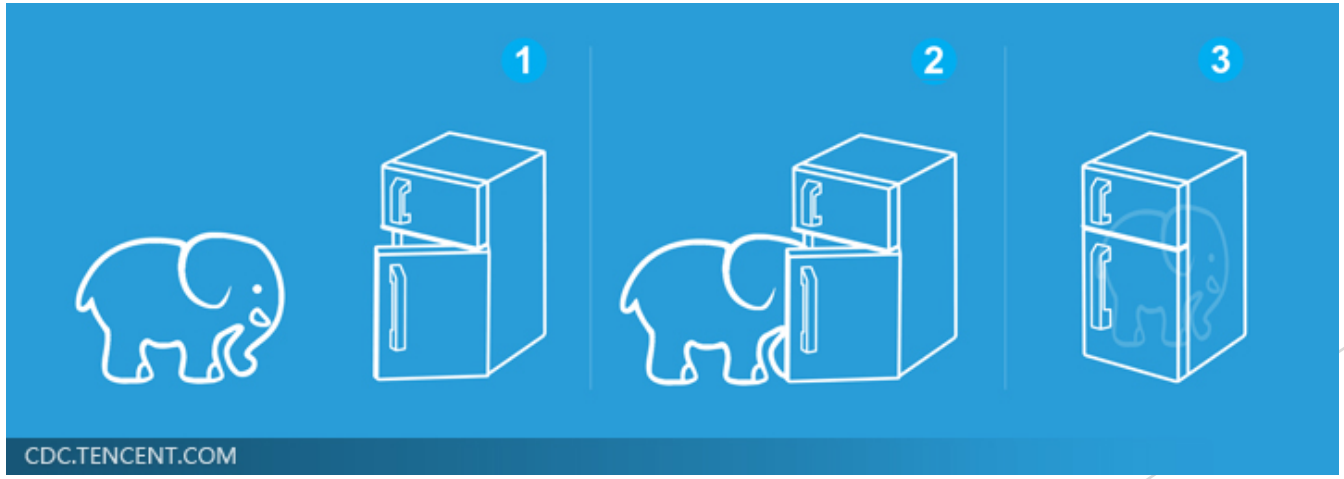
Tensor Core 的 4x4x4 矩阵乘法与累加。



深度学习学习三部曲

Step 1: define a set of function → Step 2: goodness of function → Step 3: pick the best function

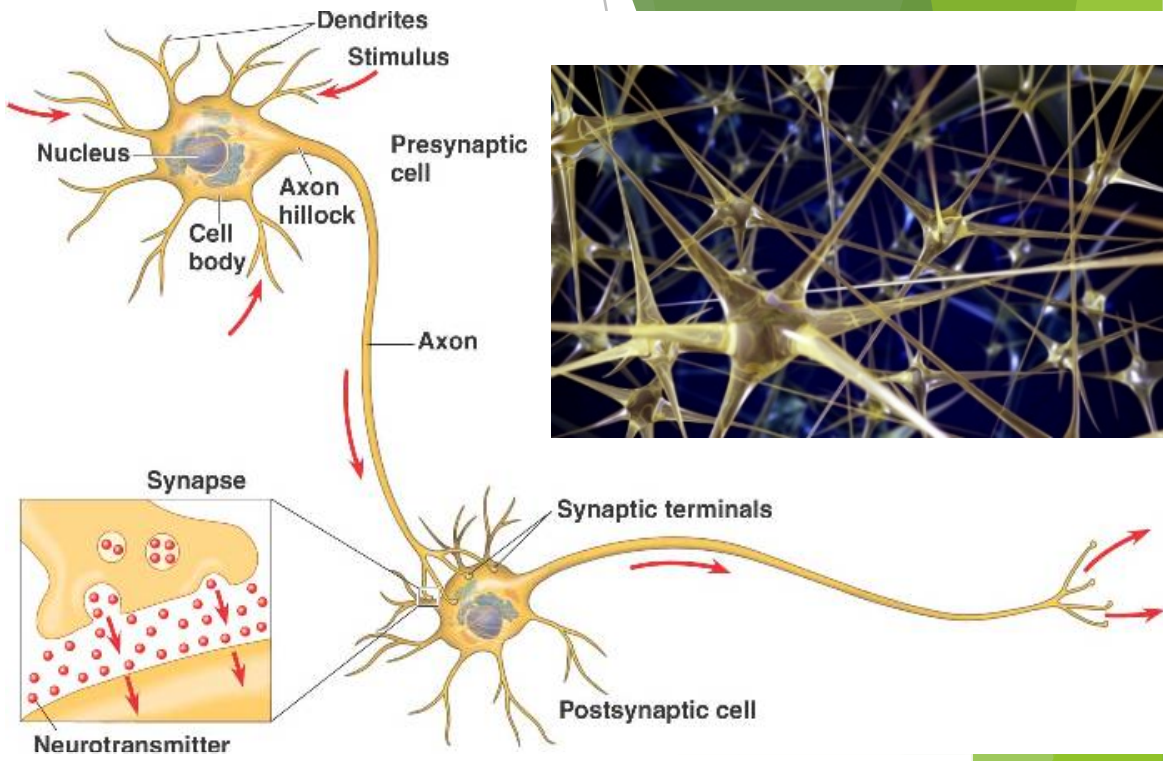
Deep Learning is so simple ……

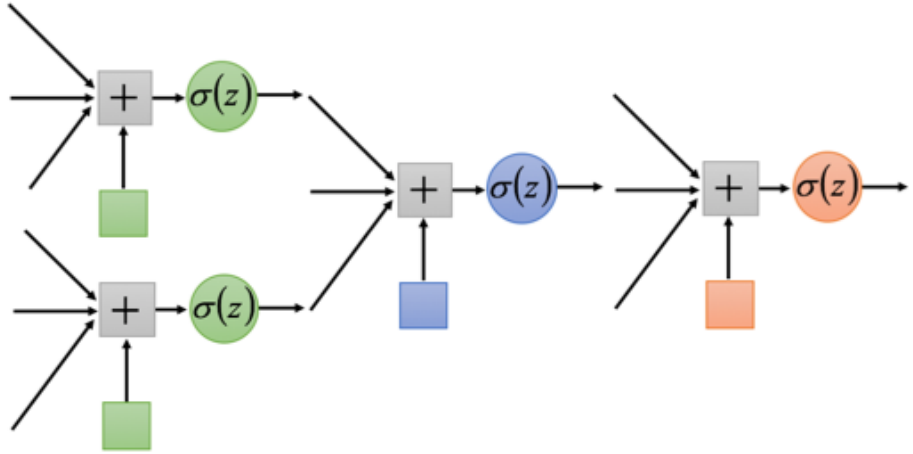


激活函数

- 在人工神经网络中，神经元节点的激活函数定义了对神经元输出的映射，简单来说，神经元的输出（例如，全连接网络中就是输入向量与权重向量的内积再加上偏置项）经过激活函数处理后再作为输出。

- 神经网络中激活函数的主要作用是提供网络的非线性建模能力，如不特别说明，激活函数一般而言是非线性函数。假设一个示例神经网络中仅包含线性卷积和全连接运算，那么该网络仅能够表达线性映射，即便增加网络的深度也依旧还是线性映射，难以有效建模实际环境中非线性分布的数据。加入（非线性）激活函数之后，深度神经网络才具备了分层的非线性映射学习能力。因此，激活函数是深度神经网络中不可或缺的部分。

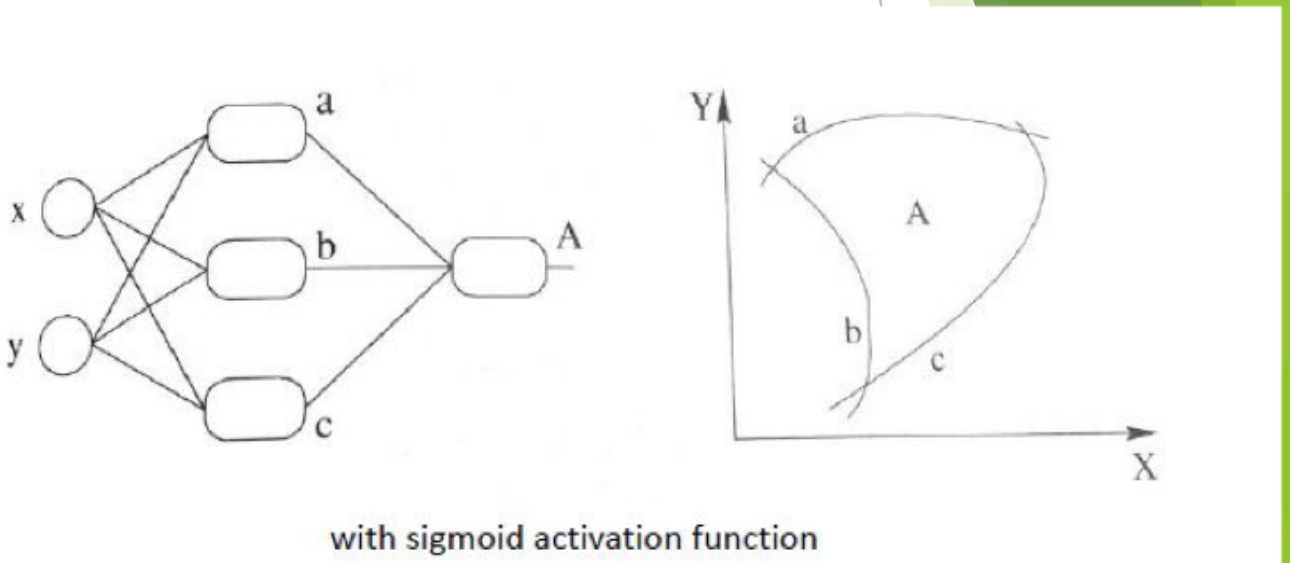




Universal approximation theorem

- 神经网络的万能近似定理:一个前馈神经网络如果具有线性层和至少一层具有"挤压"性质的激活函数（如**signmoid**等），给定网络足够数量的隐藏单元，它可以以任意精度来近似任何从一个有限维空间到另一个有限维空间的**borel**可测函数。

- 要相符上面的定理，也就是想拟合任意函数一个必须点是"要有带有"挤压"性质的激活函数"。这里的"挤压"性质是因为早期对神经网络的研究用的是**sigmoid**类函数，所以对其数学性质的研究也主要基于这一类性质：将输入数值范围挤压到一定的输出数值范围。（后来发现，其他性质的激活函数也可以使得网络具有普适近似器的性质，如**ReLU** 。



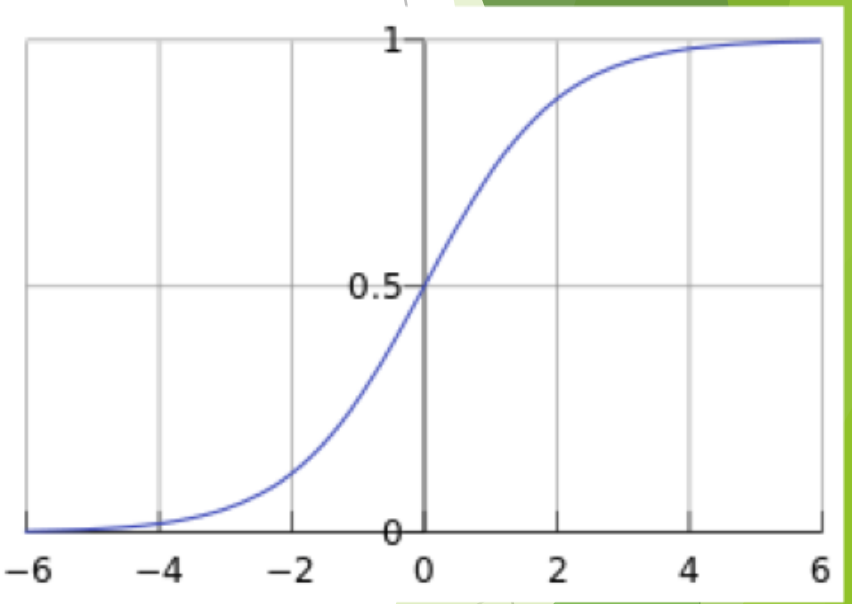
with sigmoid activation function

逻辑回归（Logistic Regression）-1

- 逻辑回归（Logistic Regression）是一种用于解决二分类（0 or 1）问题的机器学习方法，用于估计某种事物的可能性。
- 逻辑回归（Logistic Regression）与线性回归（Linear Regression）都是一种广义线性模型（generalized linear model）。逻辑回归假设因变量 y 服从伯努利分布，而线性回归假设因变量 y 服从高斯分布。

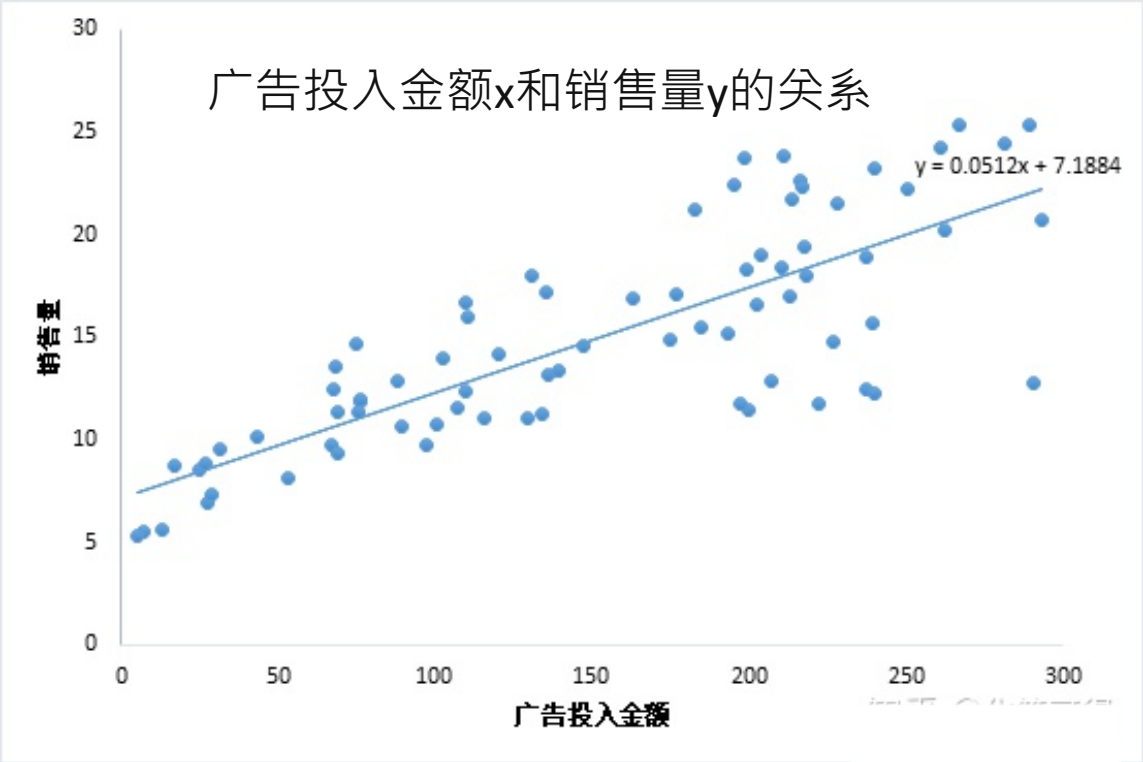
$$g(z) = \frac{1}{1 + e^{-z}}$$

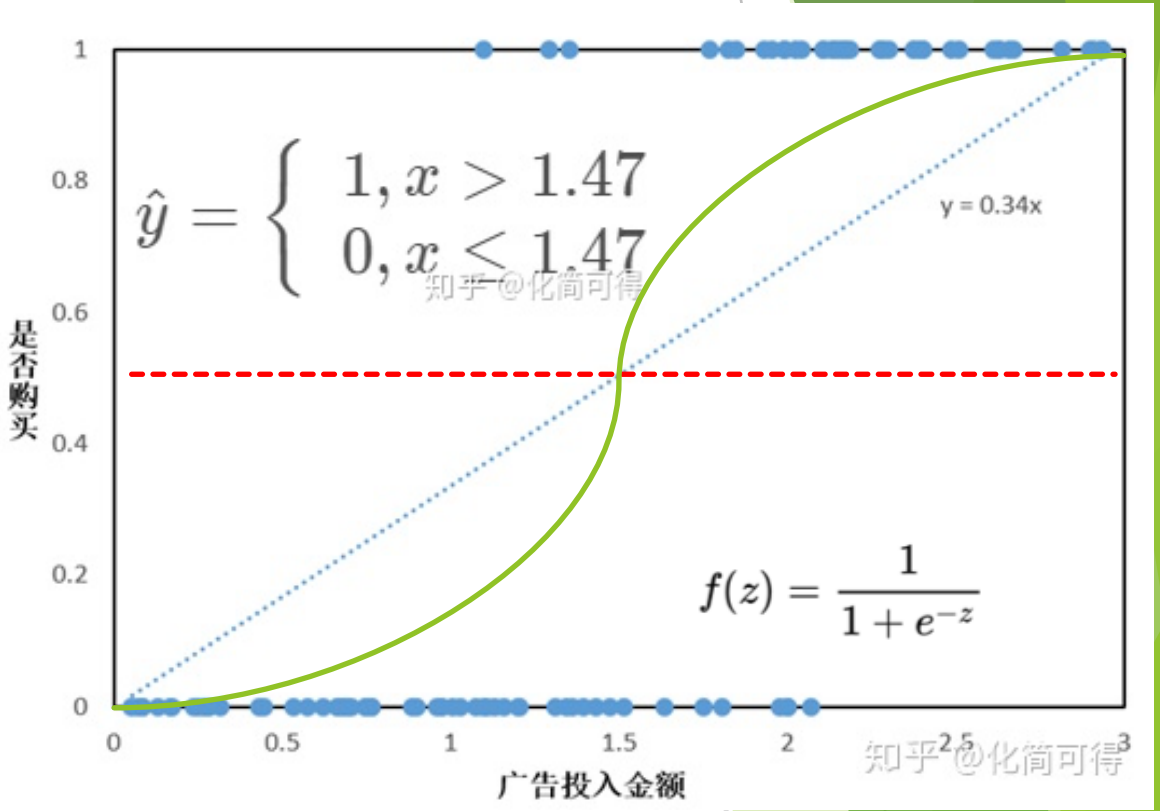
Sigmoid函数，也称为逻辑函数（Logistic function



从图可以看到sigmoid函数是一个s形的曲线

逻辑回归（Logistic Regression）-2





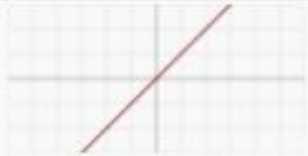
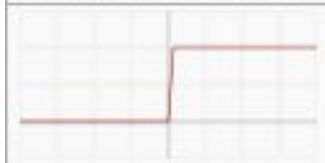
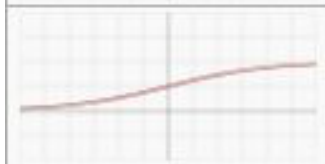
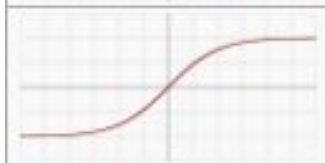
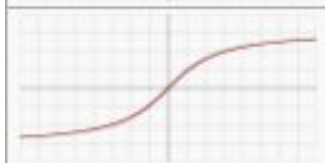
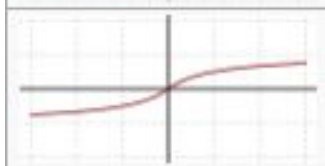
推广至多元场景

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p \qquad \boldsymbol{Y} = \boldsymbol{X\beta}$$

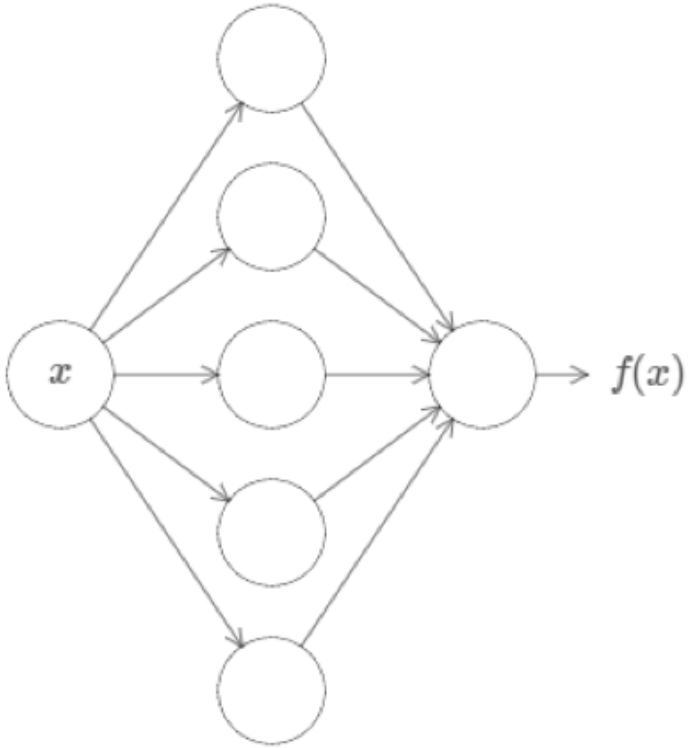
$$P(Y=1) = \frac{1}{1+e^{-X\beta}}$$

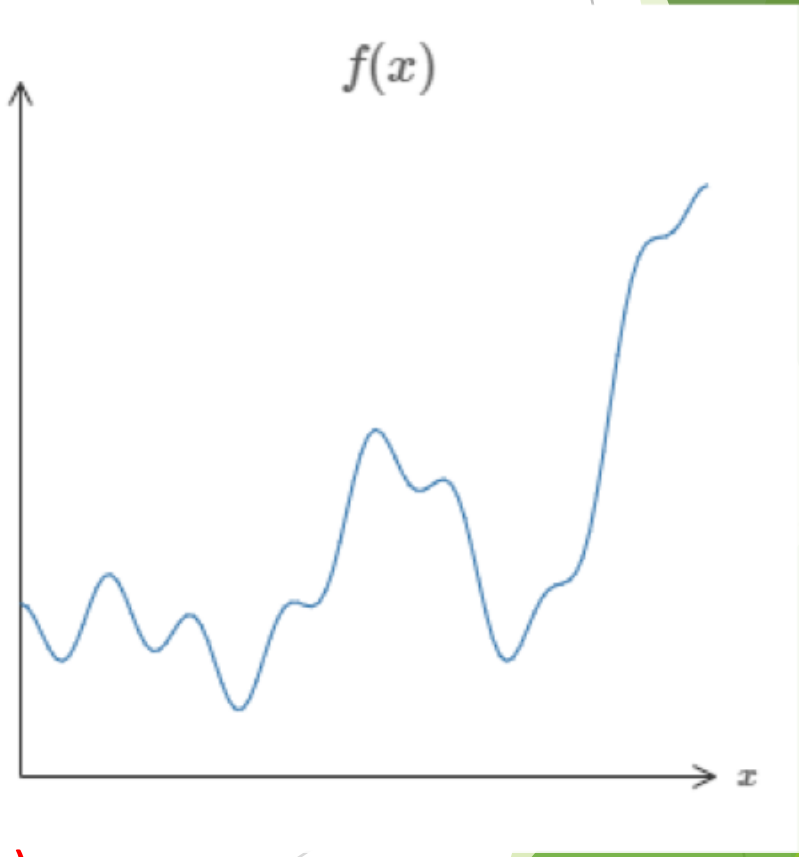
但在许多实际问题中，因变量y是分类型，只取0、1两个值，和x的关系不是上面那样。假设我们有这样一组数据：给不同的用户投放不同金额的广告，记录他们购买广告商品的行为，1代表购买，0代表未购买。

常见激活函数

	$f(x) = x$	$f'(x) = 1$
	$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
	$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
	$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
	$f(x) = \frac{x}{1+\lvert x\rvert}$	$f'(x) = \frac{1}{(1+\lvert x\rvert)^2}$
	$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$

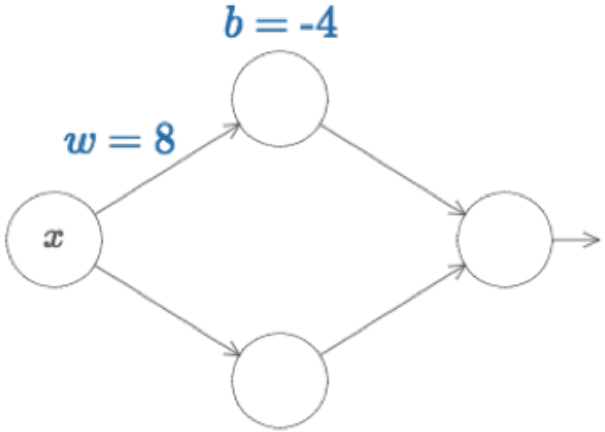
神经网络直观解释-1

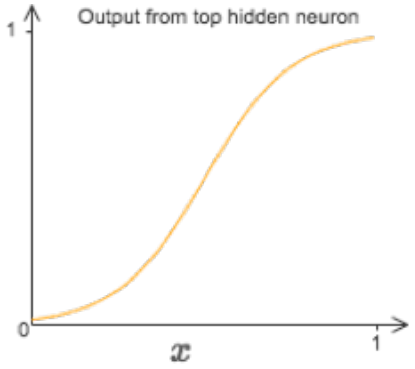


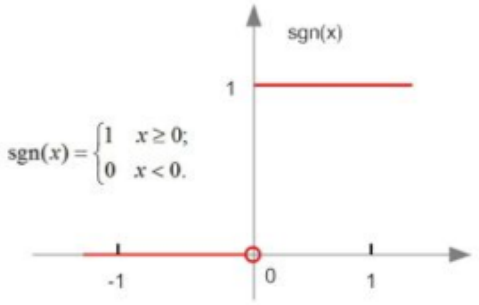


能够刻画出任何连续函数f(x)

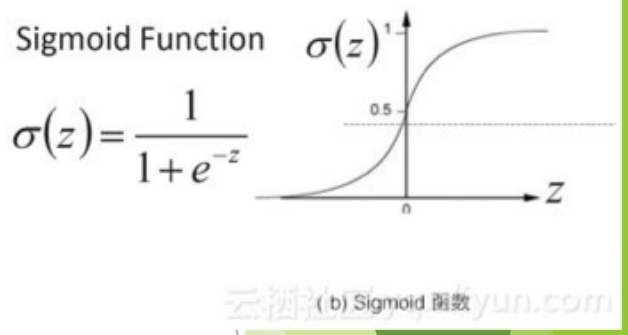
神经网络直观解释-2







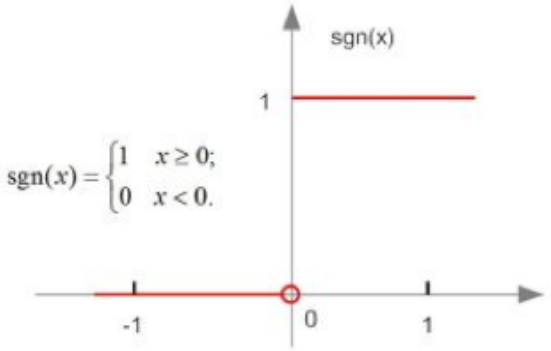
(a) 阶跃函数



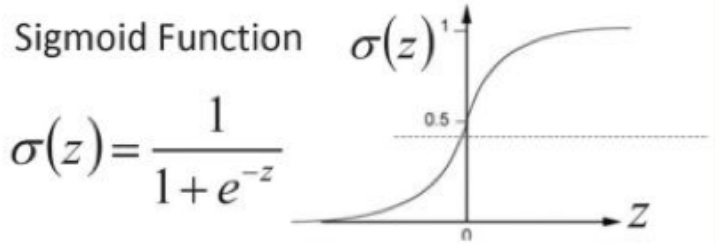
(b) Sigmoid 函数

激活函数 $\sigma(z) \equiv 1/(1+e^{-z})$

我们通常用Sigmoid函数来代替阶跃函数。
这个函数可以把较大变化范围内输入值（x）挤压输出在（0,1）范围之内，故此这个函数又称为“挤压函数（Squashing function）”。

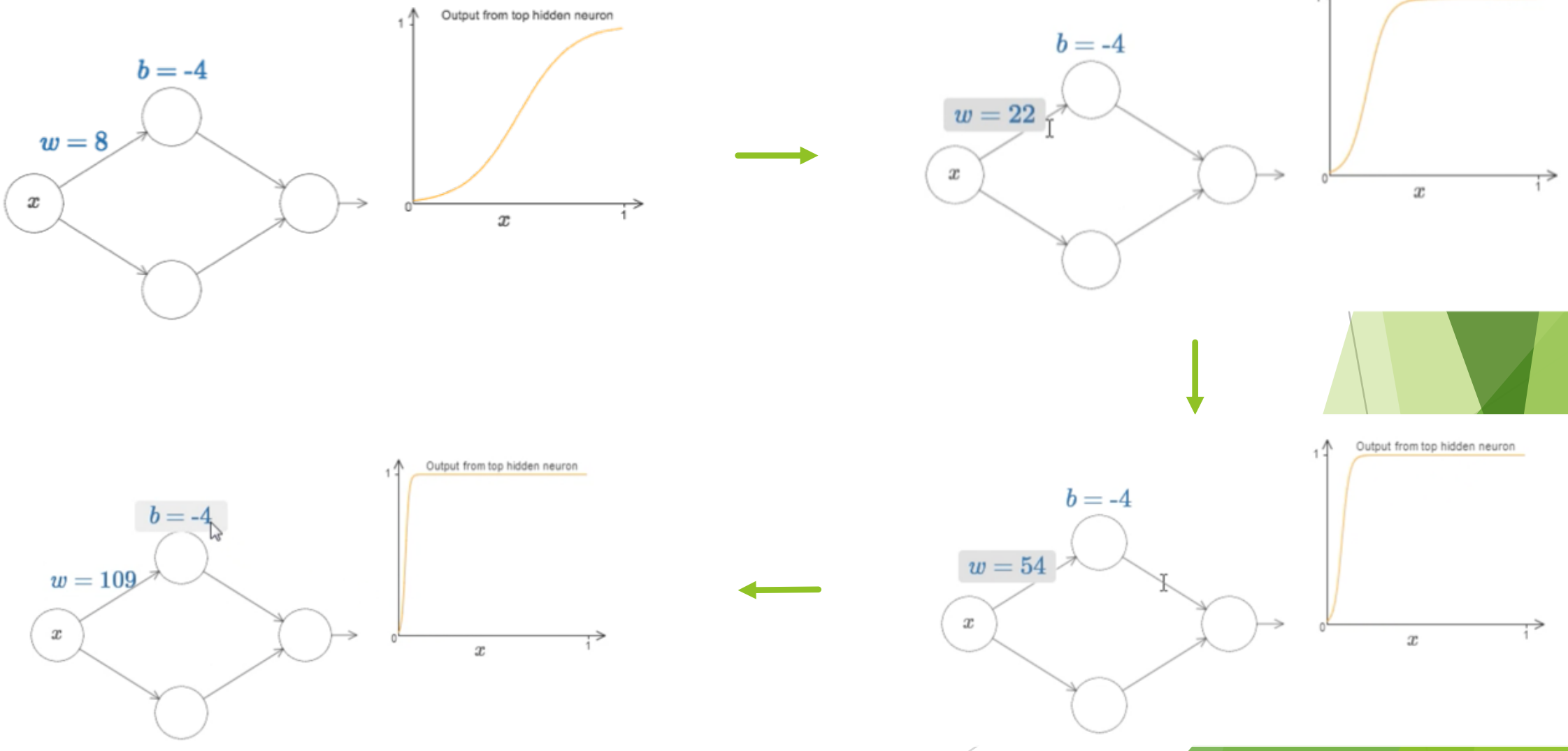


(a) 阶跃函数

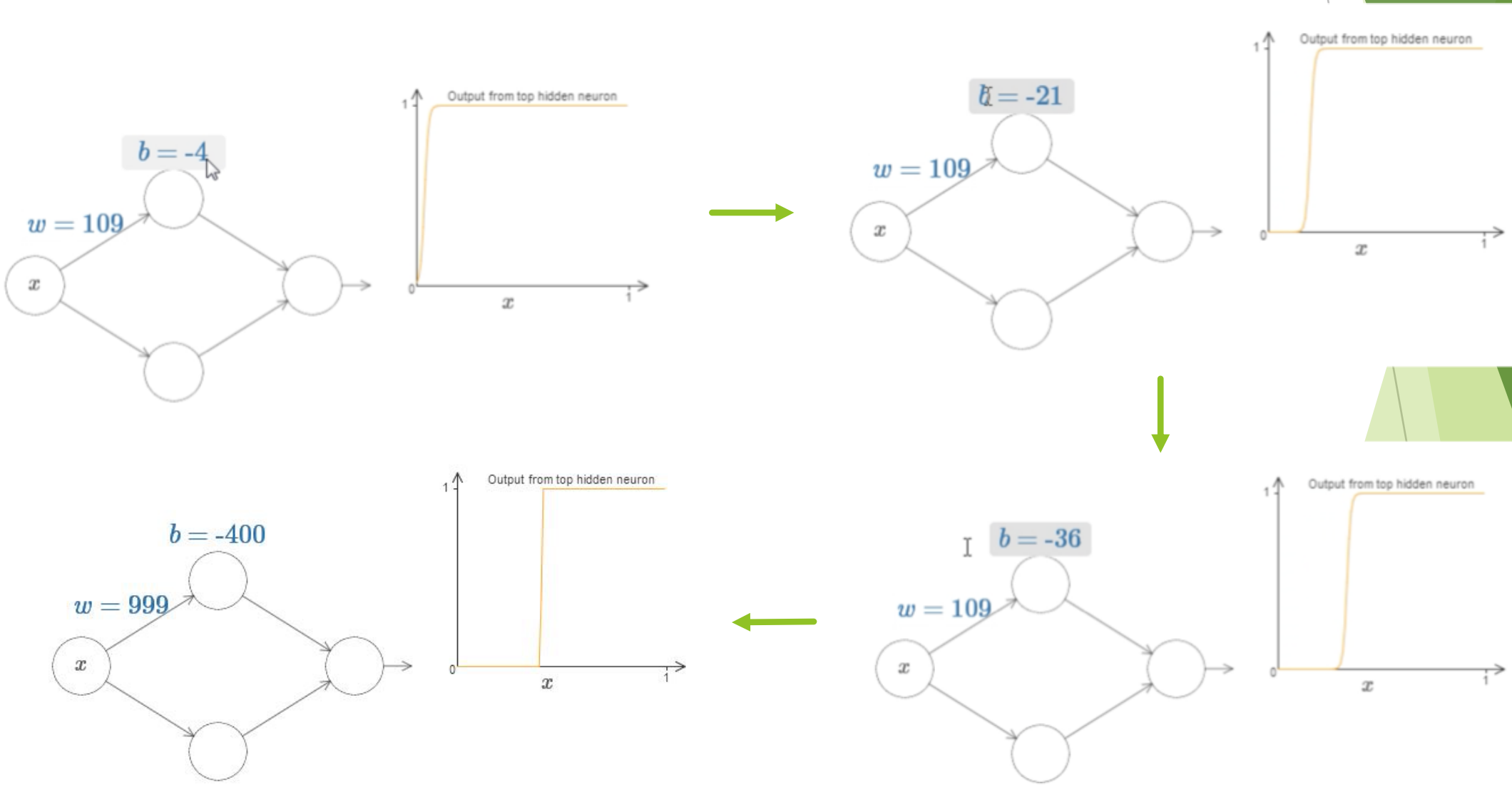


(b) Sigmoid 函数

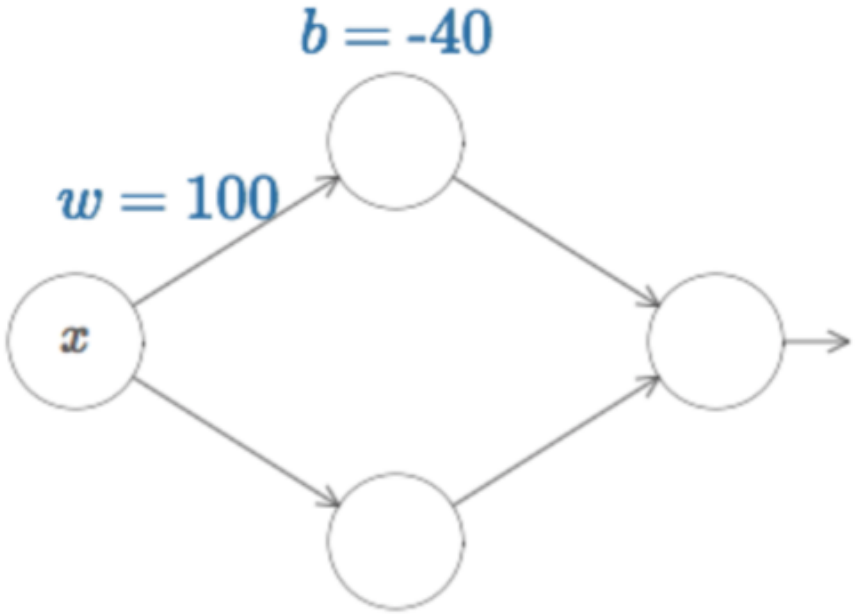
神经网络直观解释-3

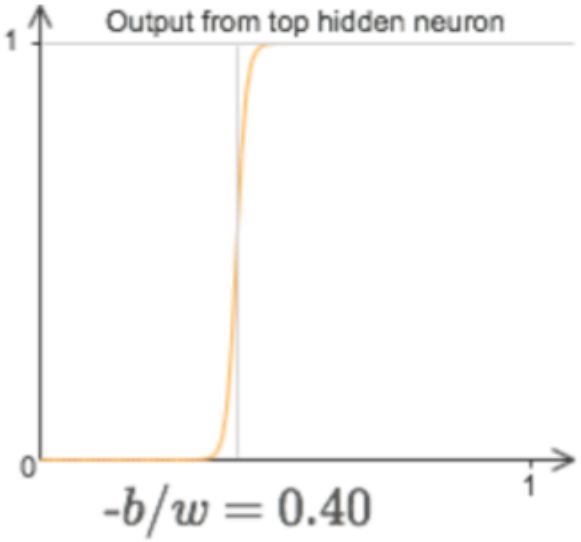

b = -4
w = 8
x
Output from top hidden neuron
1
0
x
1
b = -4
w = 22
x
Output from top hidden neuron
1
0
x
1
b = -4
w = 54
x
Output from top hidden neuron
1
0
x
1
b = -4
w = 109
x
Output from top hidden neuron
1
0
x
1

神经网络直观解释-4


b = -4
w = 109
x
Output from top hidden neuron
1
0
x
1
b = -21
w = 109
x
Output from top hidden neuron
1
0
x
1
b = -36
w = 109
x
Output from top hidden neuron
1
0
x
1
b = -400
w = 999
x
Output from top hidden neuron
1
0
x
1

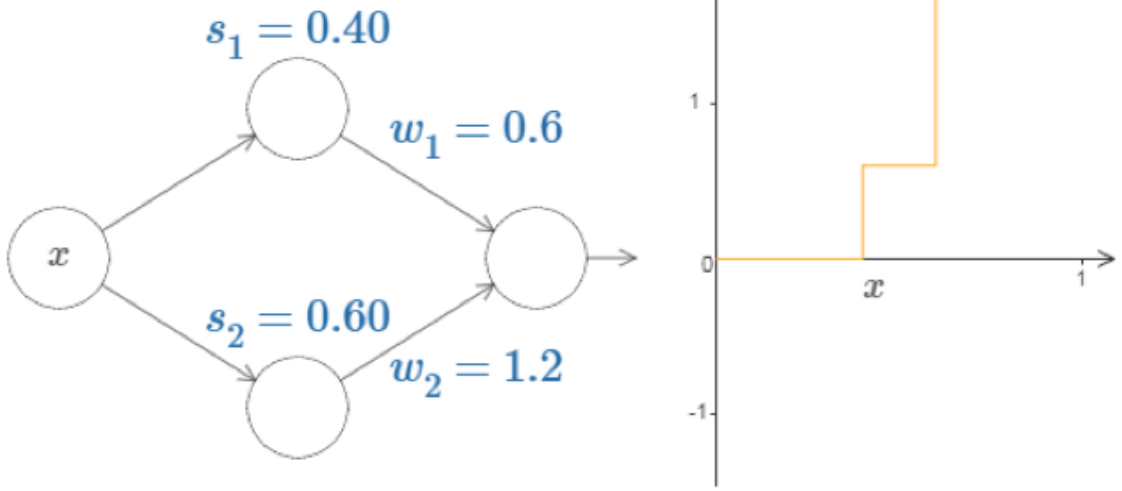
神经网络直观解释-5

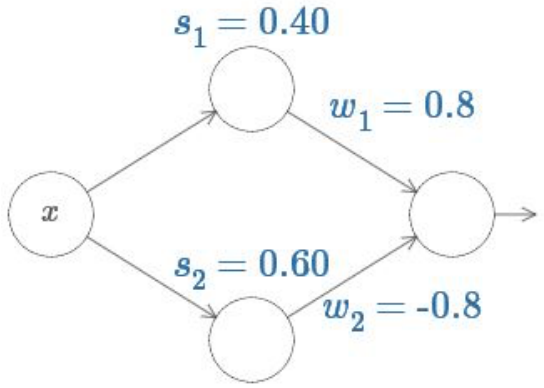


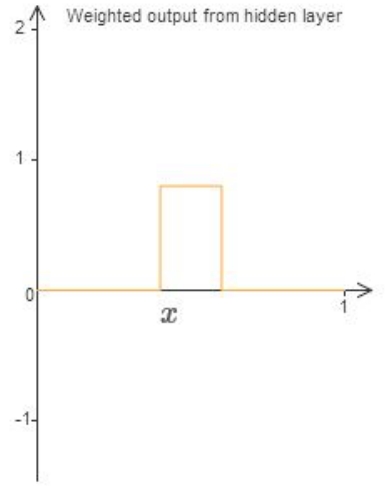


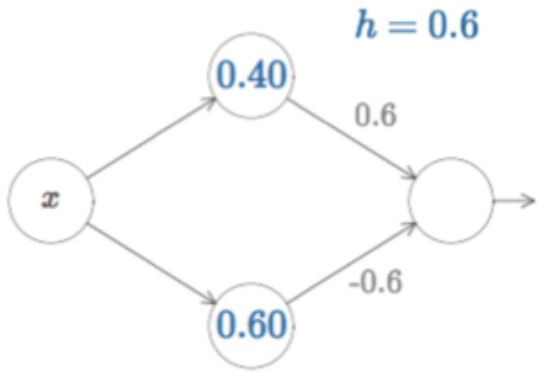
s=−b/w

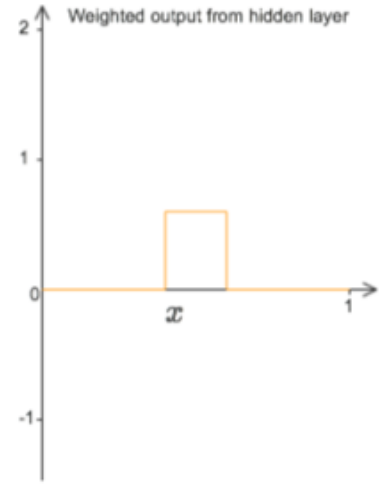
神经网络直观解释-6





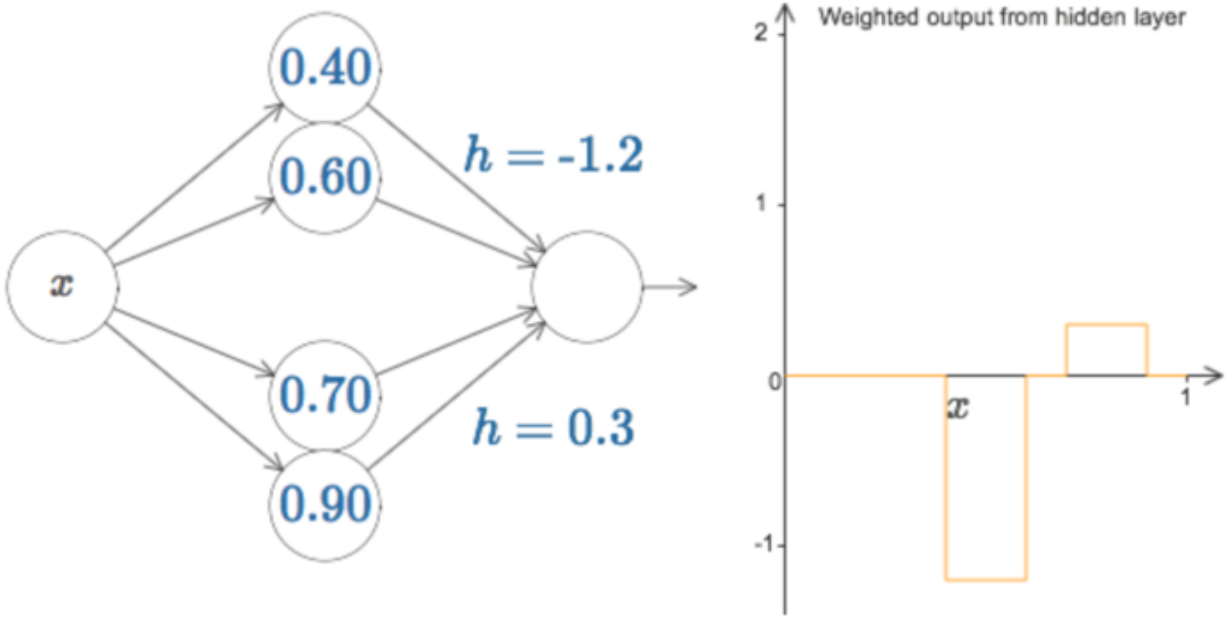


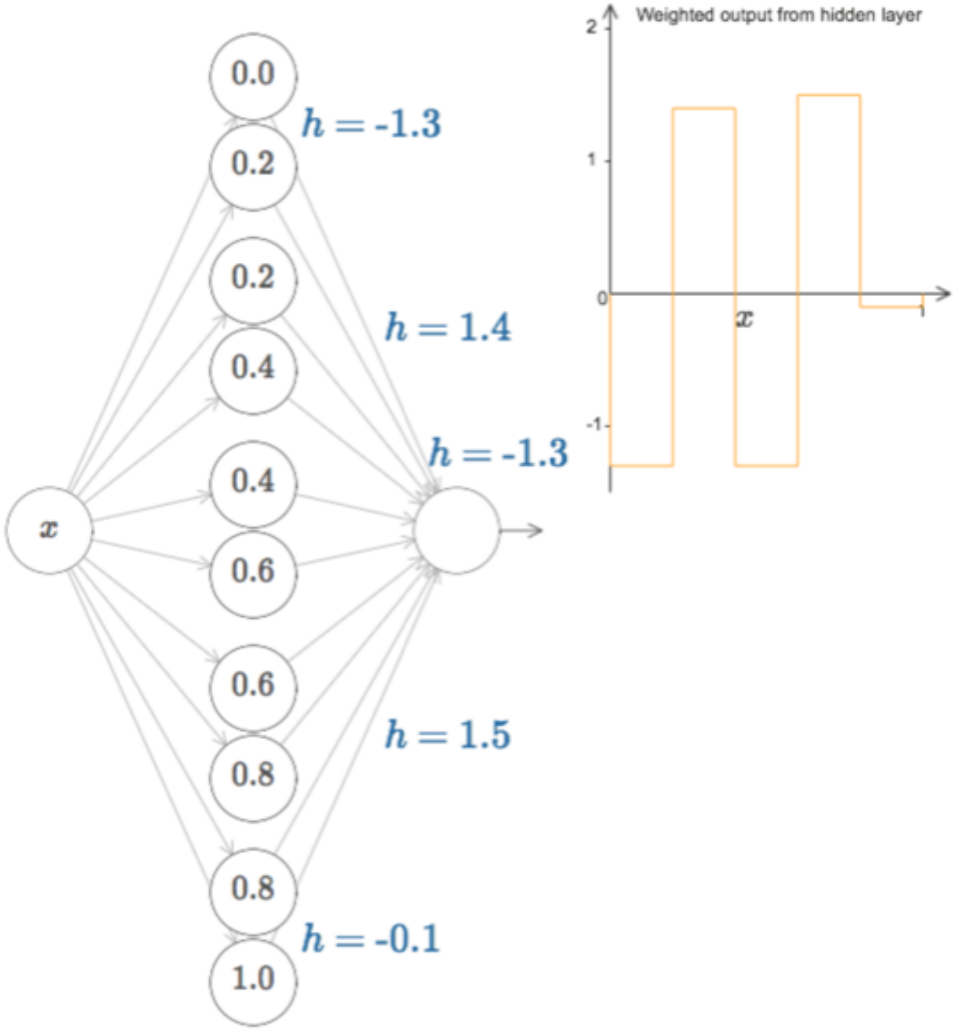




一个神经元可以拼凑出一个可控的阶梯函数，
两个神经元就可以叠加出这样的分段常数函数

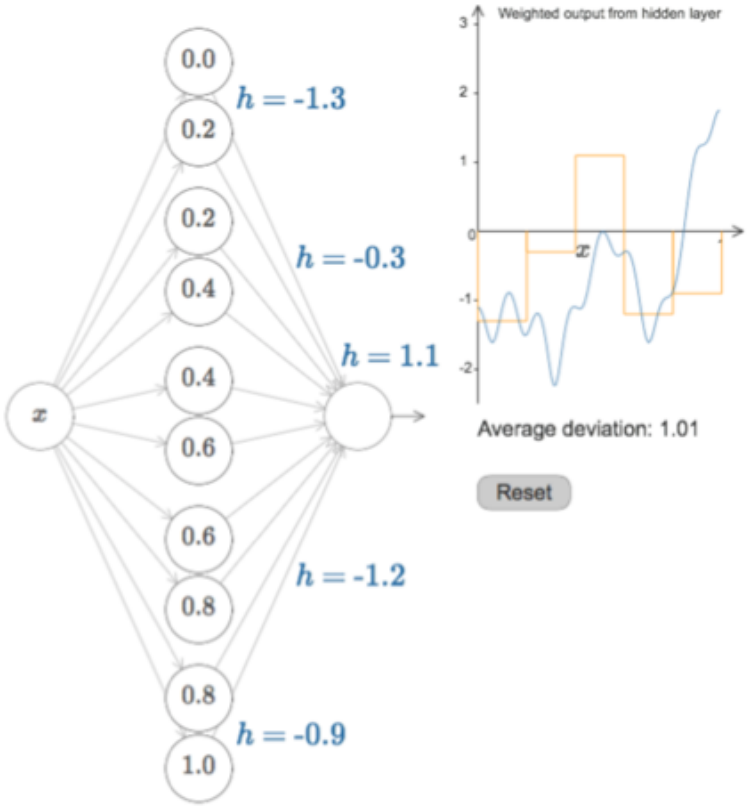
神经网络直观解释-7



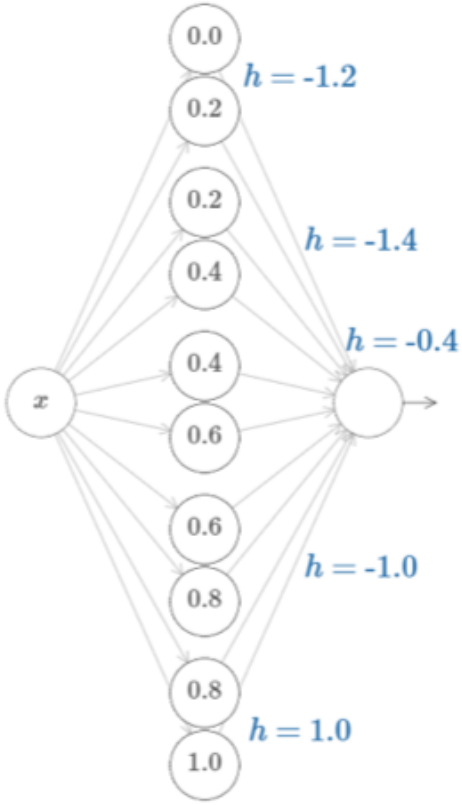


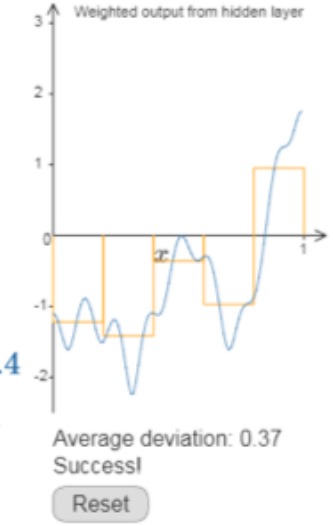
使用分段函数作为基本的单位
我们基本可以刻画任何一个连续函数

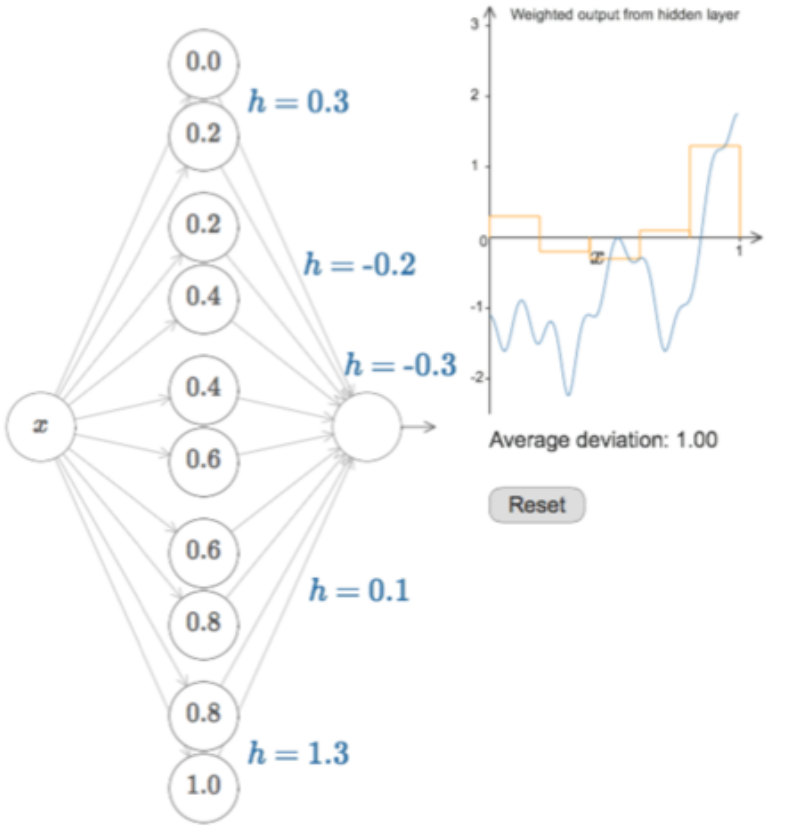
神经网络直观解释-8


0.0
0.2
0.2
0.4
0.4
0.6
0.6
0.8
0.8
1.0
$h = -1.3$
$h = -0.3$
$h = 1.1$
$h = -1.2$
$h = -0.9$
x

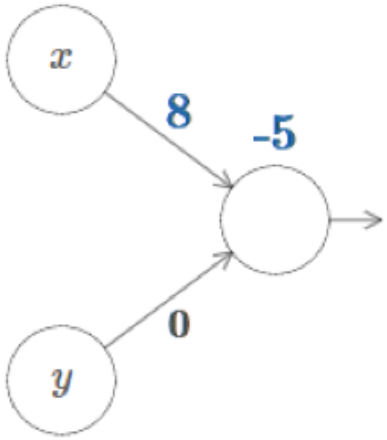
Weighted output from hidden layer
Average deviation: 1.01
Reset

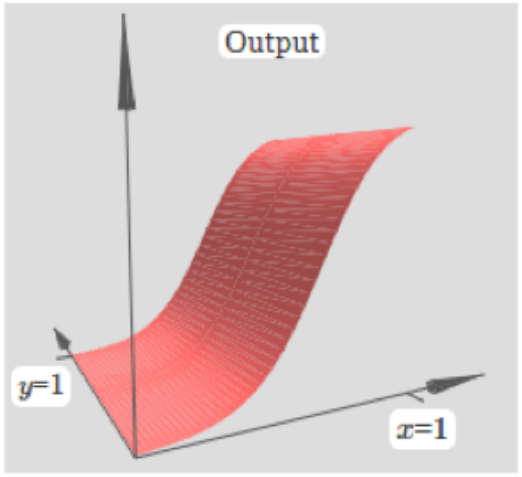

0.0
0.2
0.2
0.4
0.4
0.6
0.6
0.8
0.8
1.0
$h = -1.2$
$h = -1.4$
$h = -0.4$
$h = -1.0$
$h = 1.0$
x

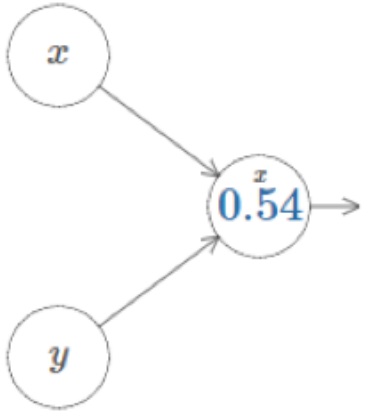

Weighted output from hidden layer
Average deviation: 0.37
Success!
Reset

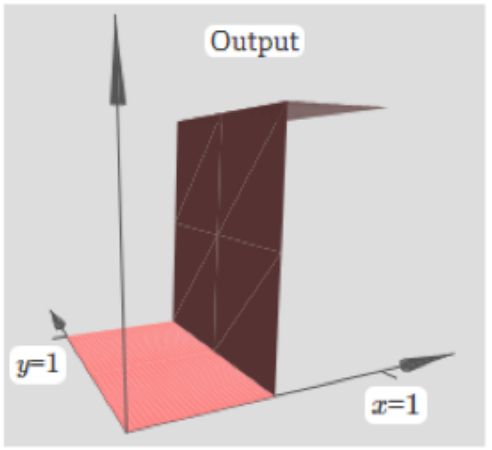

0.0
0.2
0.2
0.4
0.4
0.6
0.6
0.8
0.8
1.0
$h = 0.3$
$h = -0.2$
$h = -0.3$
$h = 0.1$
$h = 1.3$
x
Weighted output from hidden layer
Average deviation: 1.00
Reset

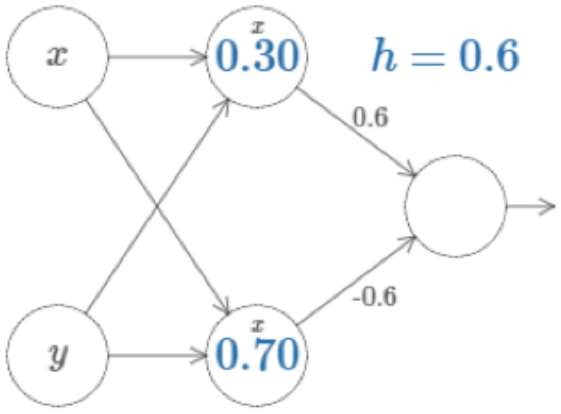
神经网络直观解释-9

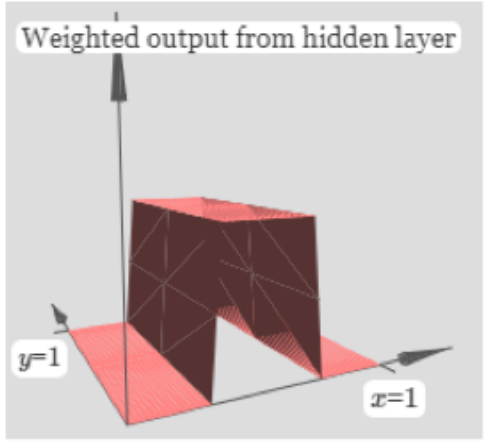

x
y
8
-5
0

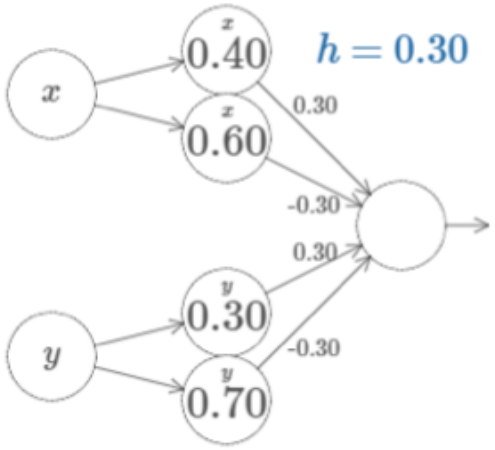

Output
y=1
x=1

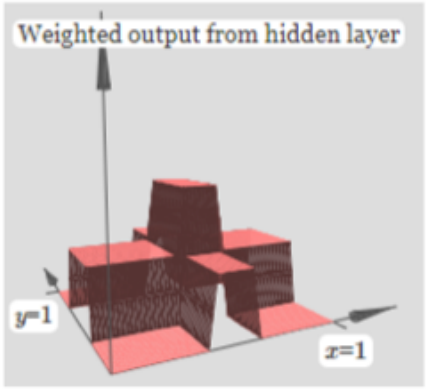

x
y
x
0.54


Output
y=1
x=1

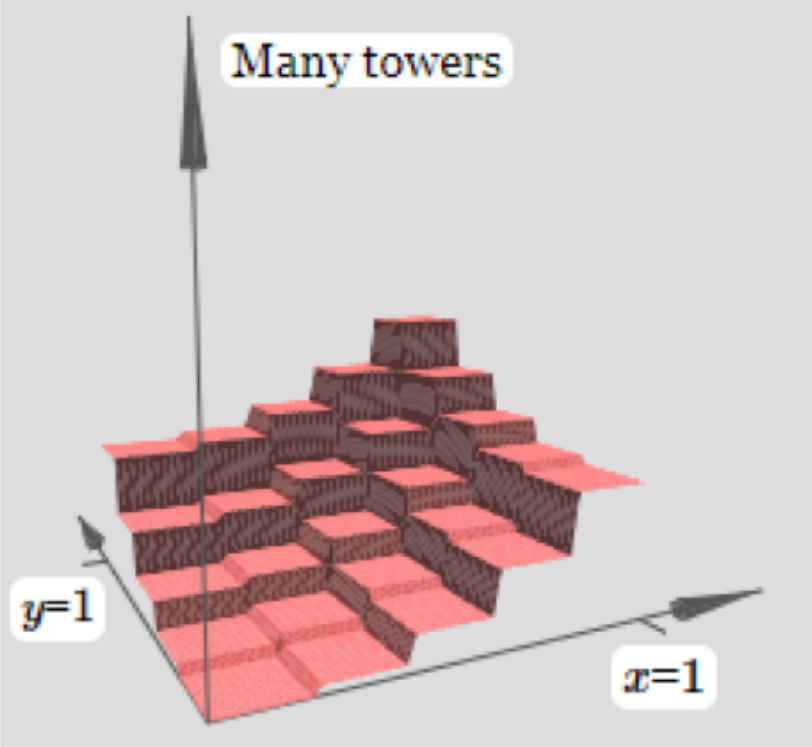

x
y
x
0.30
x
0.70
h = 0.6
0.6
-0.6


Weighted output from hidden layer
y=1
x=1


x
y
x
0.40
x
0.60
y
0.30
y
0.70
h = 0.30
0.30
-0.30
0.30
-0.30

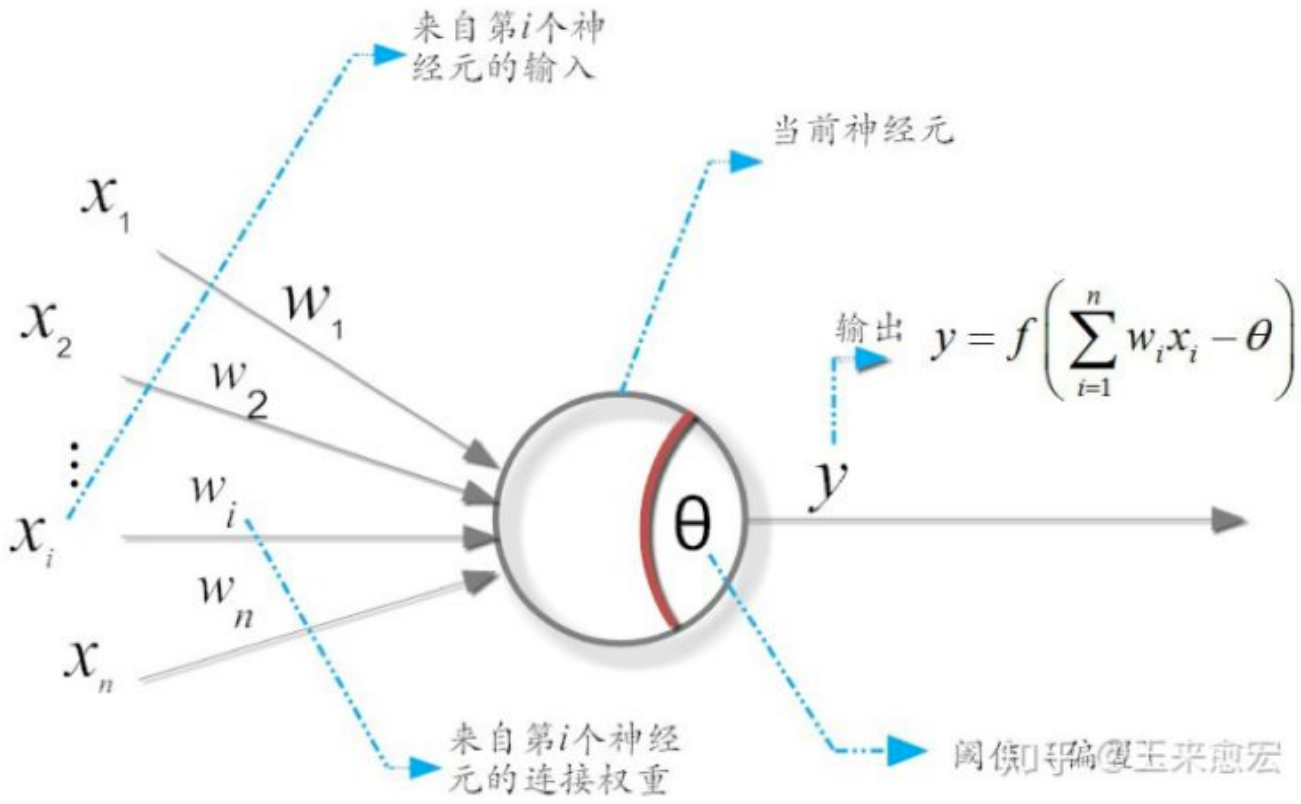

Weighted output from hidden layer
y=1
x=1

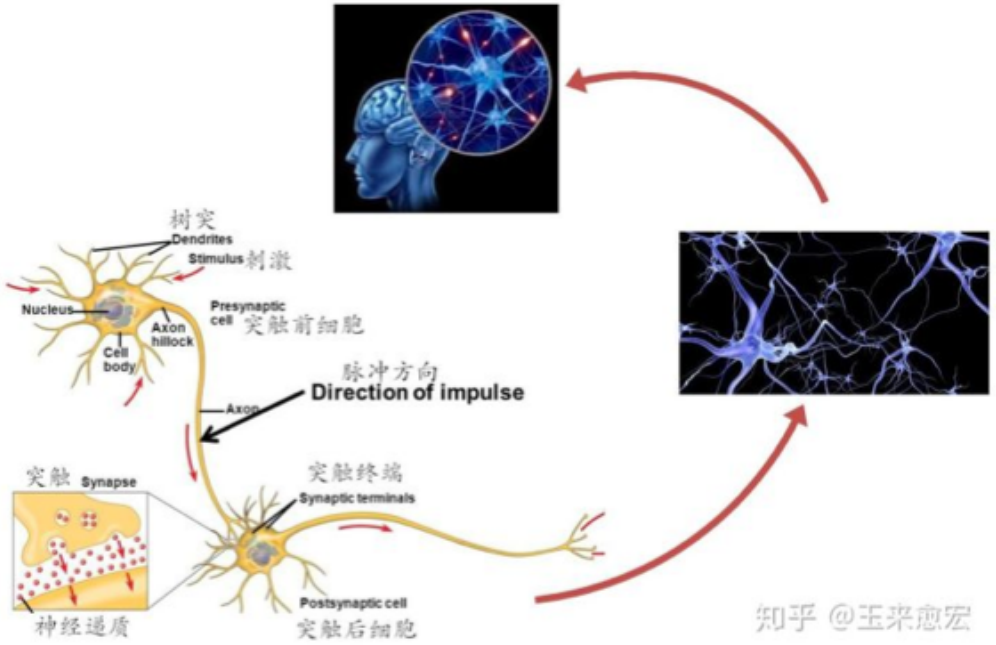
神经网络直观解释-10



结合激活函数
分段常数函数为“基”来“拟合”任何函数问题

M-P神经元模型





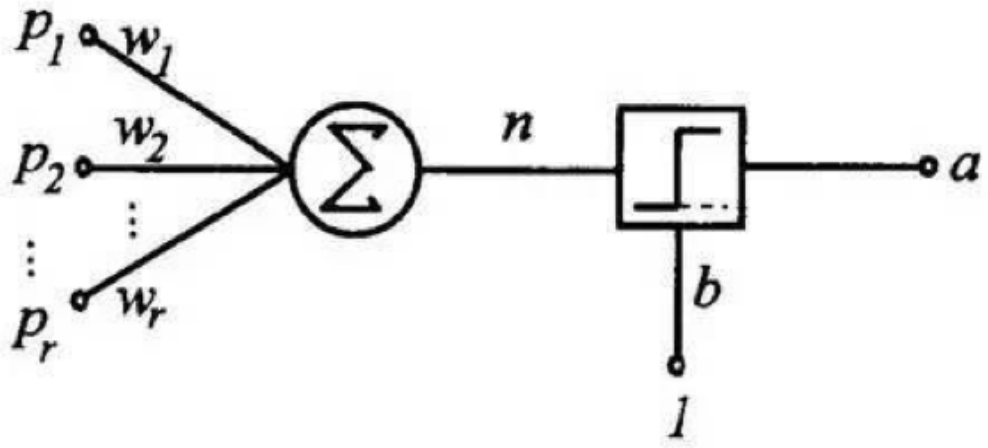
在这个模型中，神经元接收来自*n*个其它神经元传递过来的输入信号。这些信号的表达，通常通过神经元之间连接的权重（weight）大小来表示，神经元将接收到的输入值按照某种权重叠加起来。叠加起来的刺激强度*s*可用公式

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

$$S = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = \sum_{i=1}^{n} w_i x_i$$

感知机Perceptron

Sigmoid门



将当前神经元的阈值进行比较，然后通过“激活函数（activation function）”向外表达输出，这在概念上就叫感知机（perceptron）

用数学的语言来说，如果我们有m个样本，每个样本对应于n维特征和一个二元类别输出：

$$(x_1^{(0)}, x_2^{(0)}, \ldots x_n^{(0)}, y_0), (x_1^{(1)}, x_2^{(1)}, \ldots x_n^{(1)}, y_1), \ldots (x_1^{(m)}, x_2^{(m)}, \ldots x_n^{(m)}, y_m)$$

存在超平面：

$$\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n = 0 \qquad \sum_{i=0}^{n} \theta_i x_i = 0$$

使得：

- 一部分类别样本：

$$\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n > 0$$

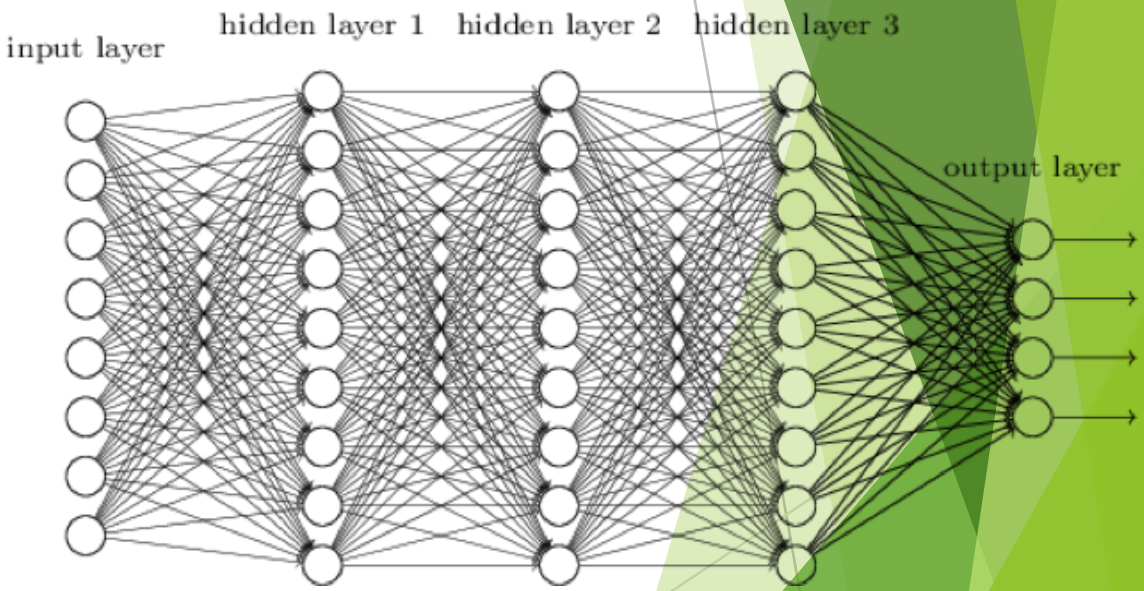
- 另一部分类别样本：

$$\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n < 0$$

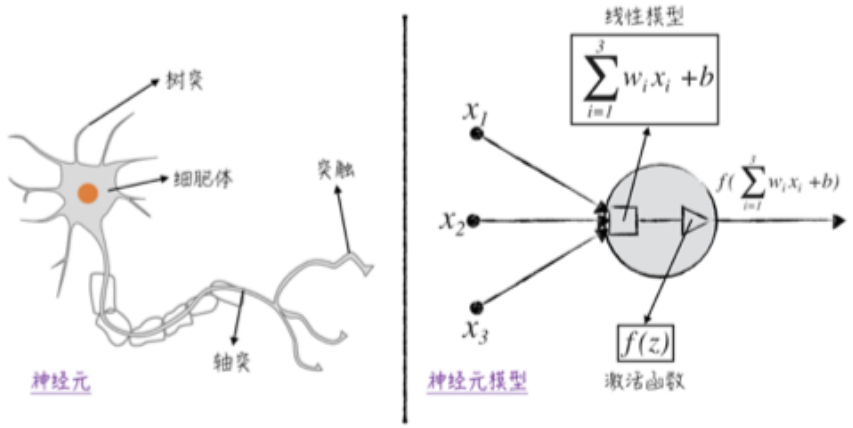
数据线性可分，这样的超平面一般都不是唯一的，也就是说感知机模型可以有多个解

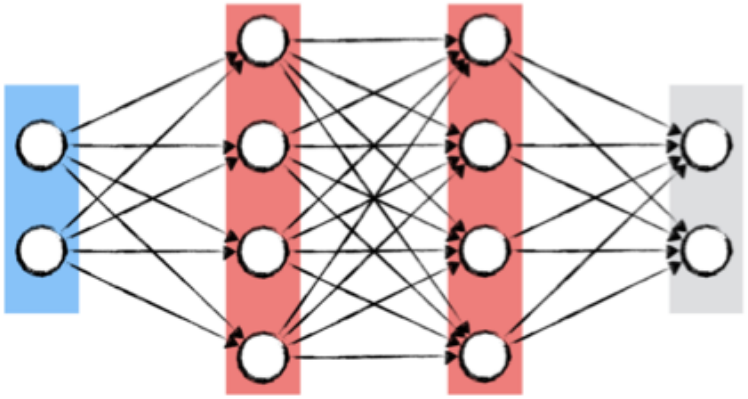
从感知机到神经网络

- 感知机模型模型只能用于二元分类，且无法学习比较复杂的非线性模型，因此在工业界无法使用。
- 神经网络则在感知机的模型上做了扩展，总结下主要有三点：
  - 加入了隐藏层，隐藏层可以有多层，增强模型的表达能力，增加了这么多隐藏层模型的复杂度也增加了很多。
  - 输出层的神经元也可以不止一个输出，可以有多个输出，这样模型可以灵活的应用于分类回归，以及其他的机器学习领域比如降维和聚类等。
  - 对激活函数做扩展，感知机的激活函数是sign(z)，虽然简单但是处理能力有限，因此神经网络中一般使用的其他的激活函数，比如逻辑回归里面使用的Sigmoid函数，还有后来出现的tanx, softmax,和ReLU等。通过使用不同的激活函数，神经网络的表达能力进一步增强。



逻辑回归和神经网络





3-层神经网络

全连接的网络

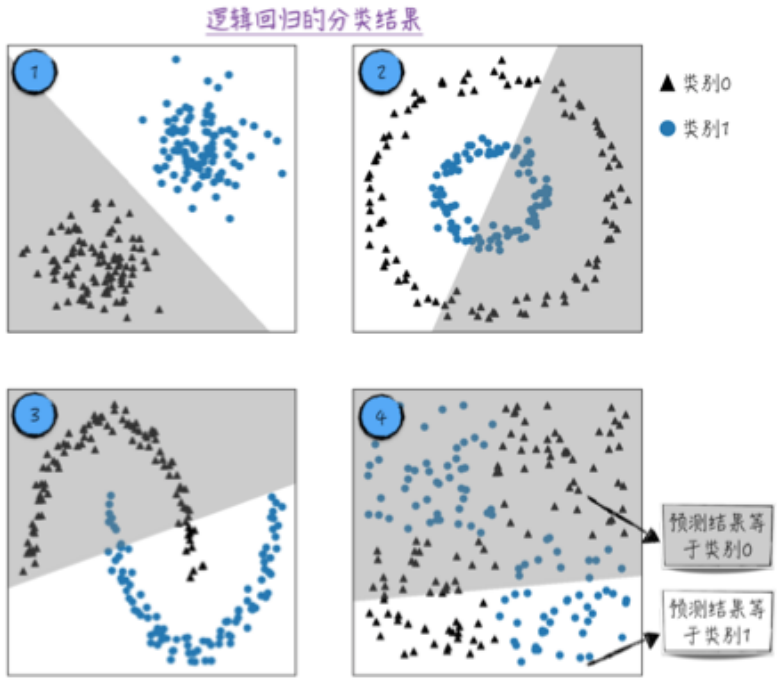
网络结构为[4, 4, 2]

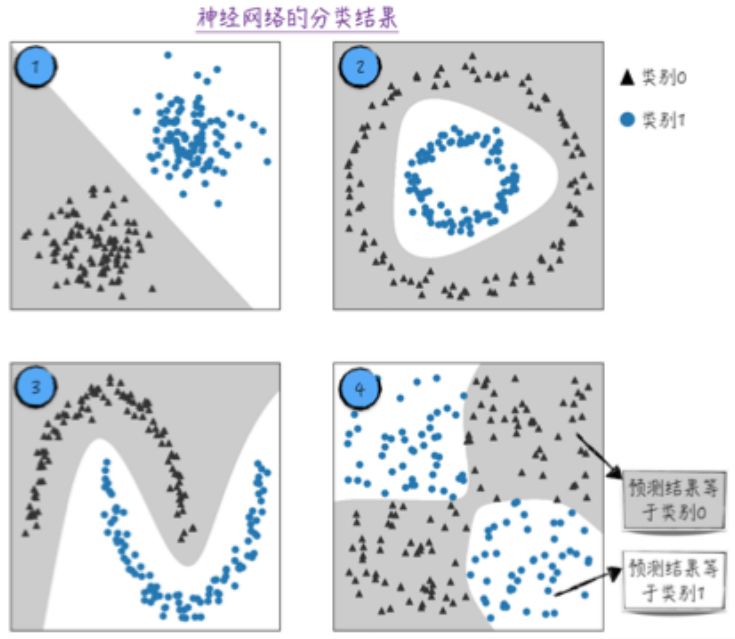
输入层 input layer

隐藏层1 hidden layer 1

隐藏层2 hidden layer 2

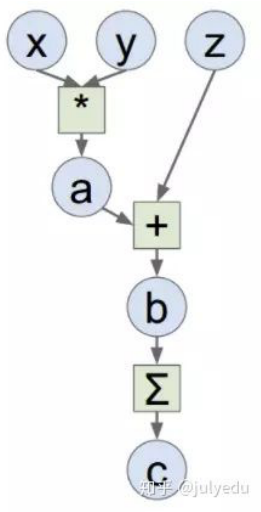
输出层 output layer

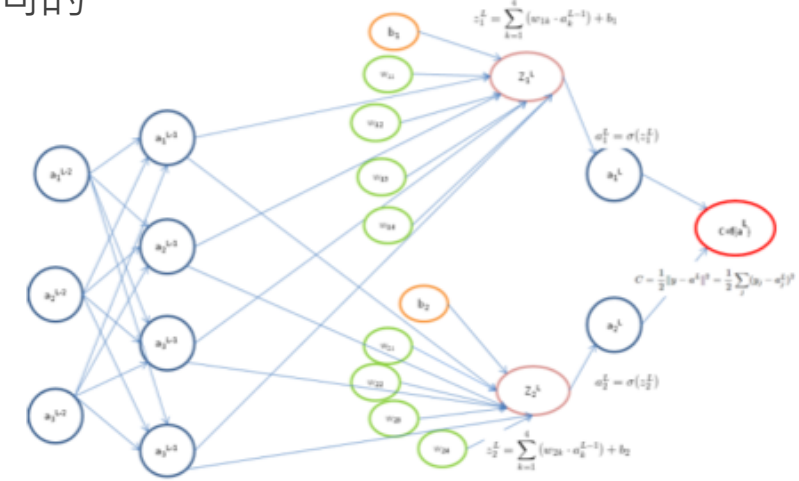


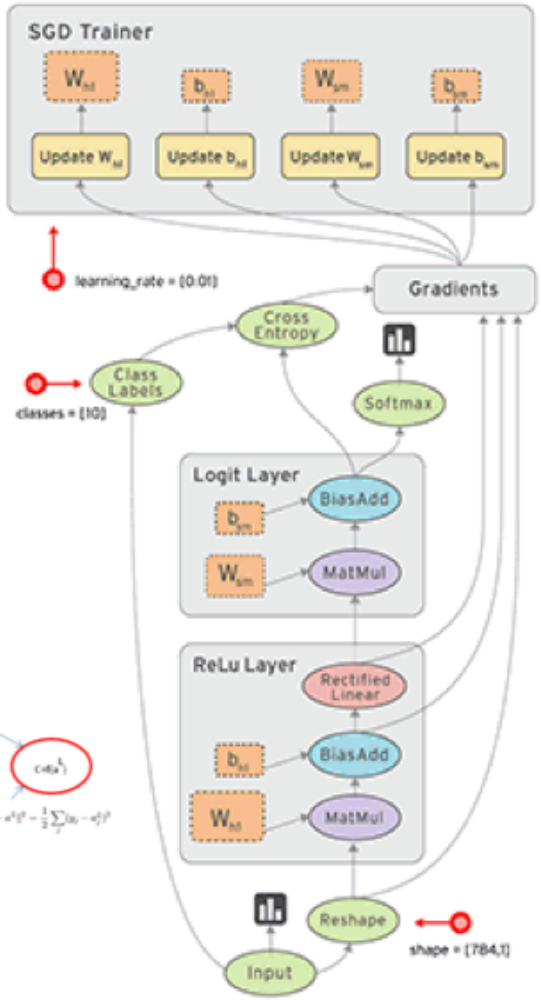


计算图Computational graphs

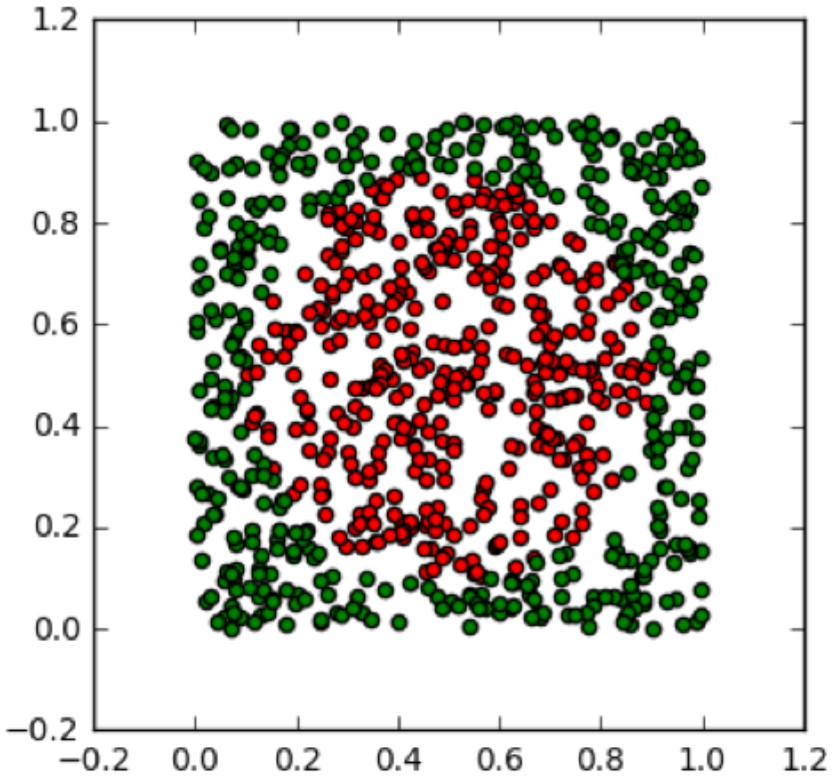
- 神经网络本身是计算图的一个特殊形式
- Computational graph 是有向图，其中的节点都对应着操作(Operation) 或者 变量(Variable)。
- Variable 可以把自己的值递送给 Operation，而 Operation 可以把自己的输出递送给其他的 Operation。
- Tensorflow是一个通过计算图的形式来表述计算的编程系统，计算图也叫数据流图，可以把计算图看做是一种有向图，Tensorflow中的每一个节点都是计算图上的一个Tensor, 也就是张量，而节点之间的边描述了计算之间的依赖关系(定义时)和数学操作(运算时)。

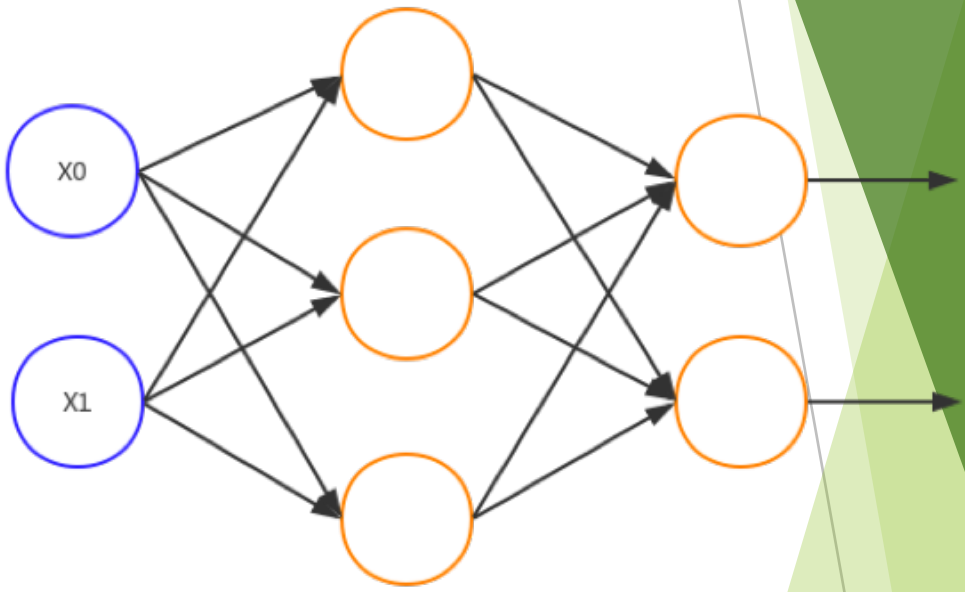






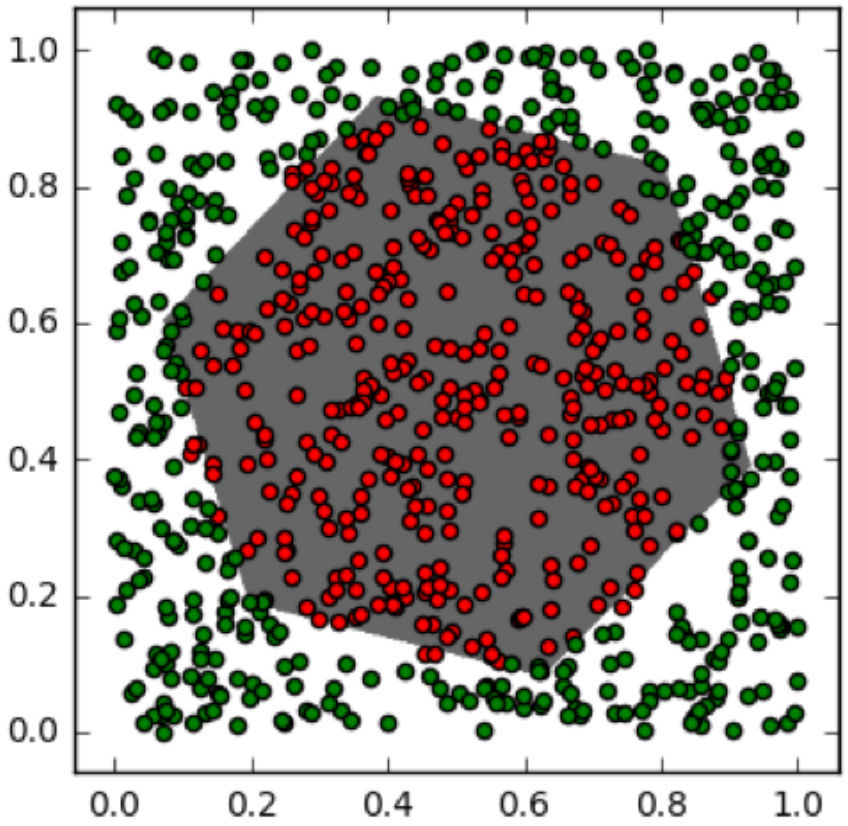
神经网络分类问题

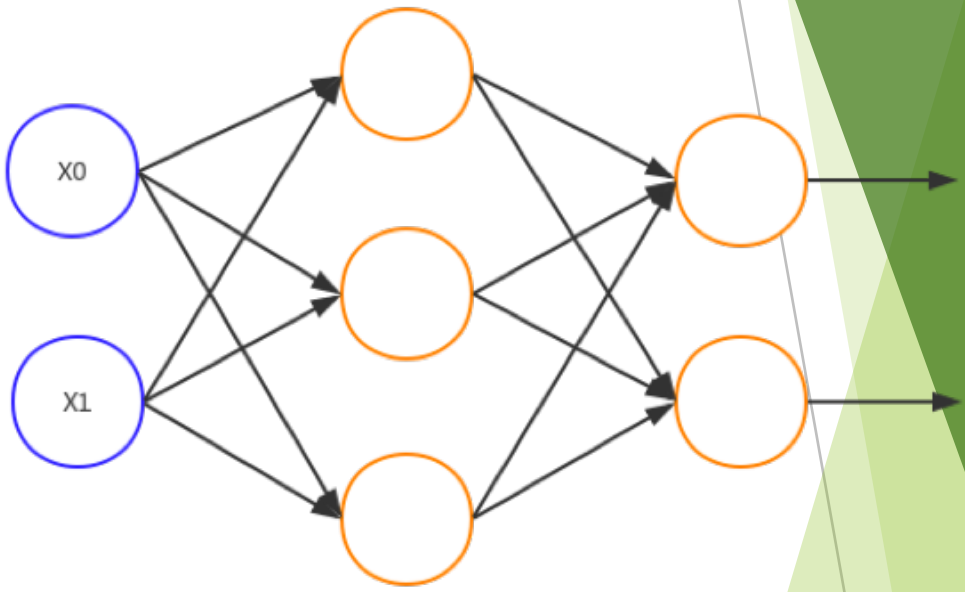




- 输入层 - 2维向量X
- 隐藏层(第一层) - ReLU层 (3个神经元)
- 输出层(第二层) - Softmax层 (2个神经元, 二元分类)

神经网络分类效果





- 输入层 - 2维向量X
- 隐藏层(第一层) - ReLU层 (3个神经元)
- 输出层(第二层) - Softmax层 (2个神经元, 二元分类)

ReLu神经元→超折面

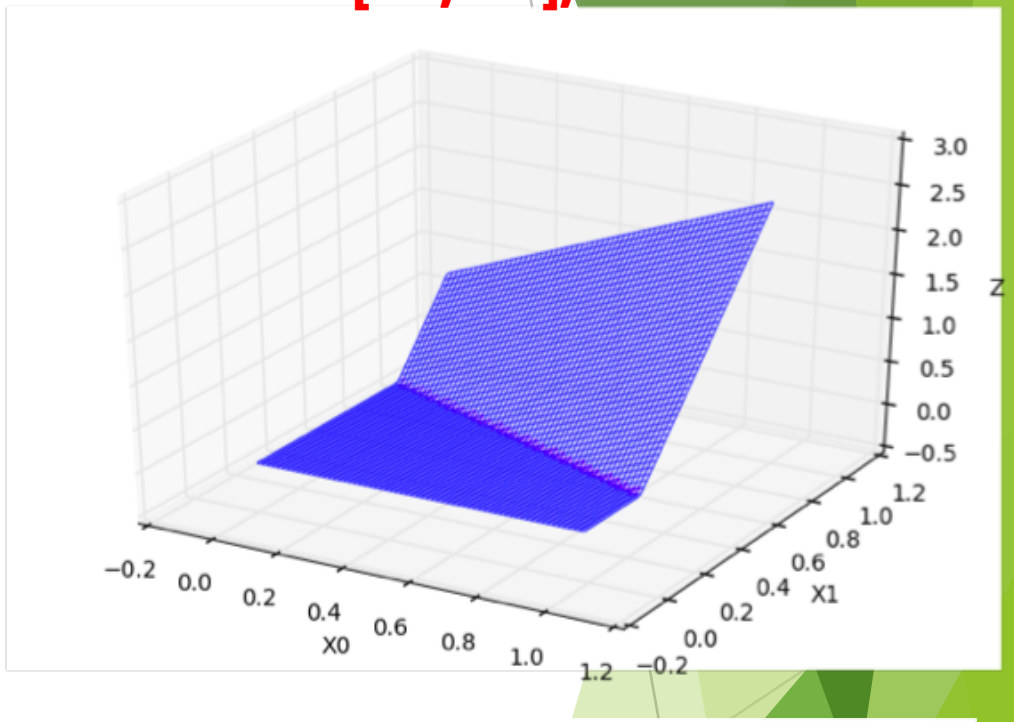
单个ReLU神经元

$$ReLU(x) = max(x, 0)$$

$$\begin{aligned} z &= ReLU(W * X + b) \\ &= ReLU(w_0 * x_0 + w_1 * x_1 + \cdots + w_n * x_n + b) \\ &= max(w_0 * x_0 + w_1 * x_1 + \cdots + w_n * x_n + b, 0) \end{aligned}$$

- W, X 均为向量
- X为该神经元的输入，W为该神经元的权重Weight参数，b为该神经元的偏置Bias参数

W = [1.5, 3.5], b = -2.5

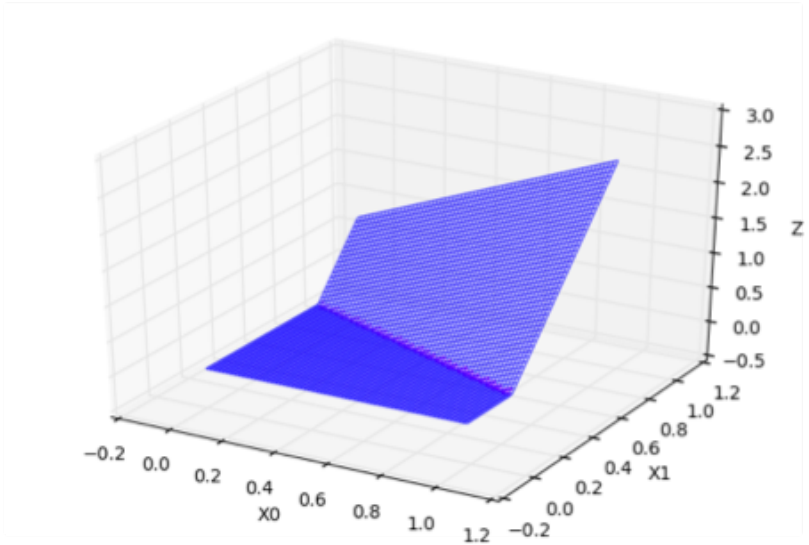


超折面角度

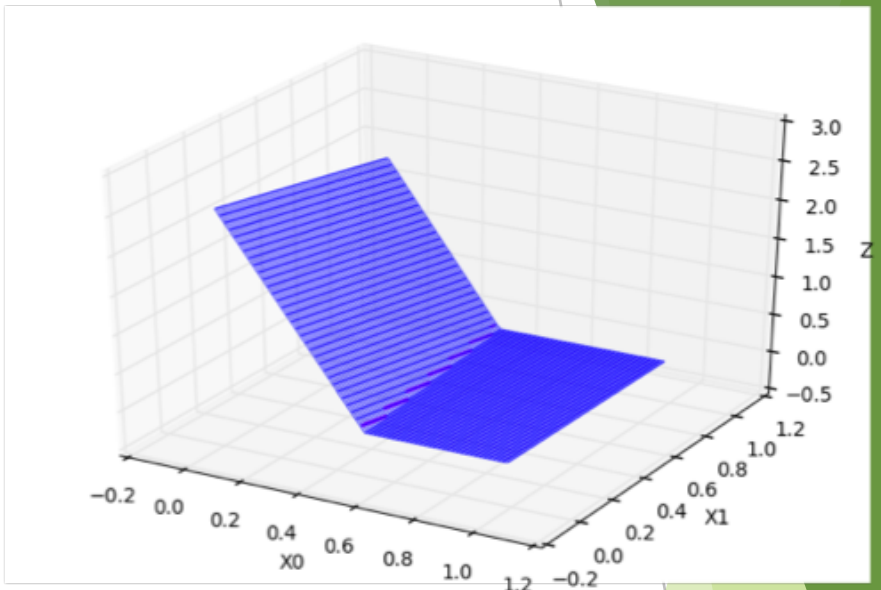
$$\begin{aligned} \theta &= \pi - \arccos\left(\frac{1}{\sqrt{w_0^2 + w_1^2 + 1}}\right) \\ &= \pi - \arccos\left(\frac{1}{\sqrt{1.5^2 + 3.5^2 + 1}}\right) \\ &= \pi - 1.3139824 \\ &= 104.714354° \end{aligned}$$

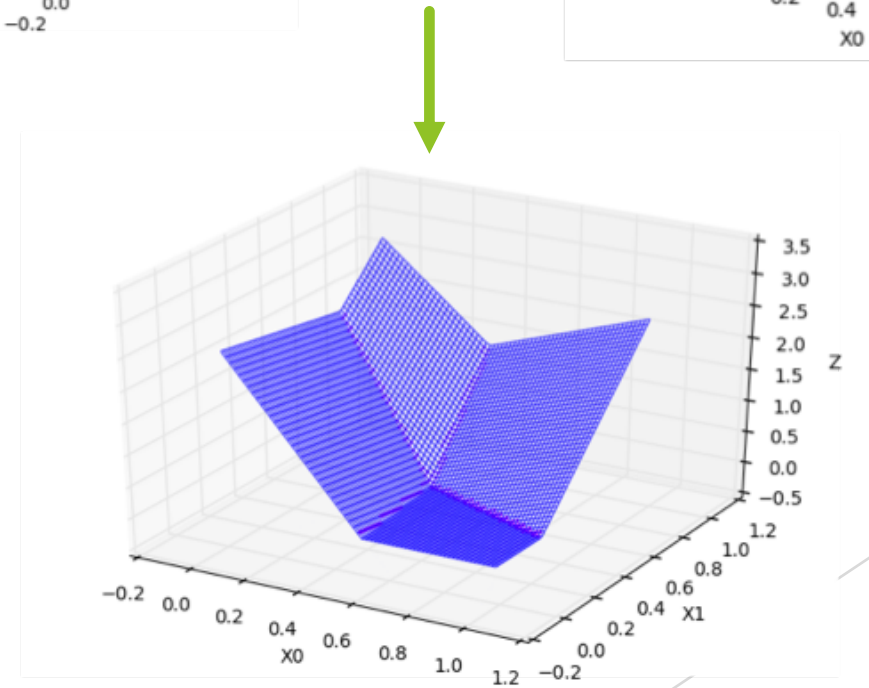
$$\theta \in (90°, 180°]$$

复合超折面


3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
Z
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
X0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
X1

+


3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
Z
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
X0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
X1


3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
Z
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
X0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2
X1

1 ReLU层 + 1 Softmax层 二元分类网络

- 输入为n维空间下的X
- 在n+1维的空间中的Z=0的超平面上生成m条超折线, 进行m次折叠 (m为第一层ReLU神经元的数目)
- 折叠后的图形进行线性相加组合(改变了各个折叠的角度和整体在Z轴上的位置)后和Z=0的超平面比较大小
- 在n+1维空间中, 其图形在Z=0的超平面的投影就是原始n维空间中的二元分类分界线

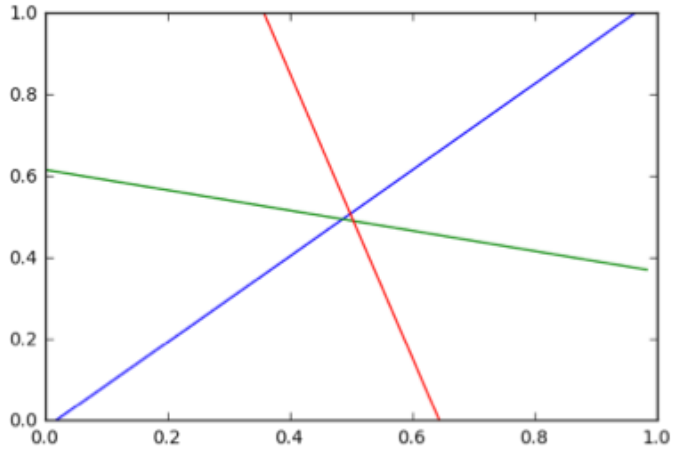
二元分类下的Softmax

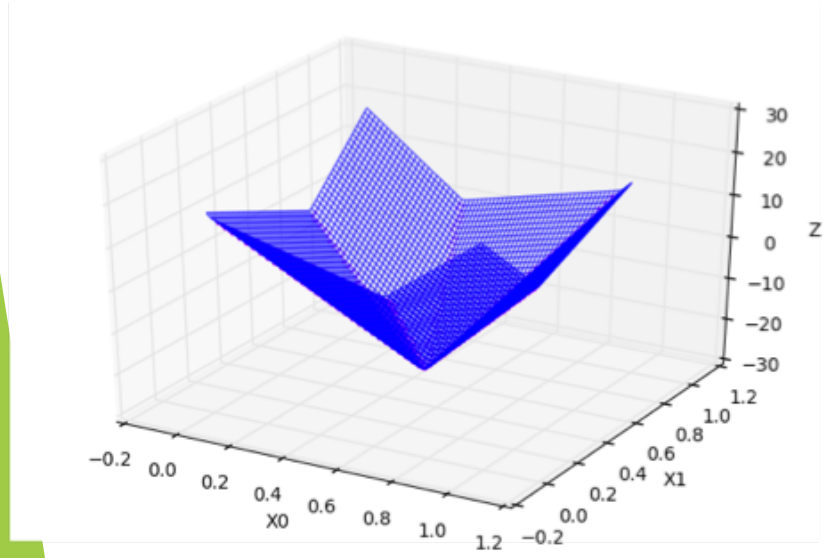
softmax(X) 将向量X的值转换为各个索引位置独立事件的概率(0~1)

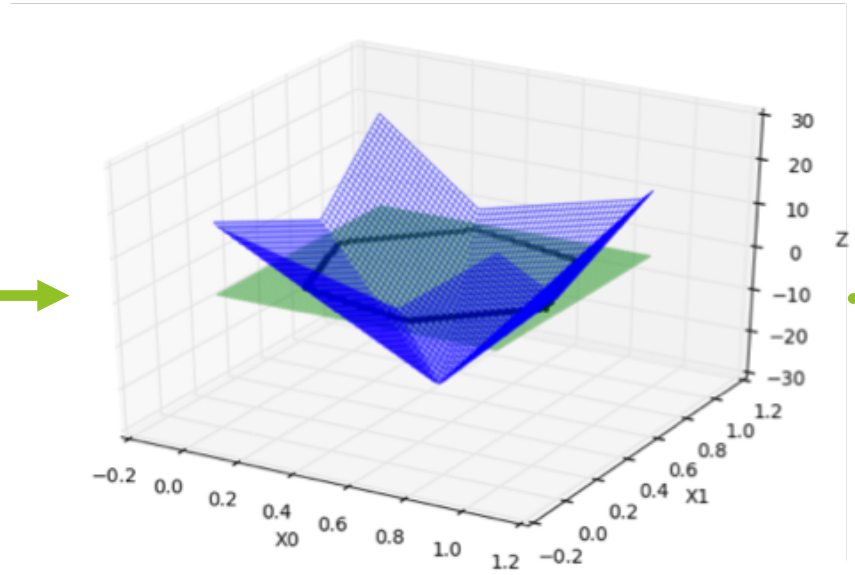
对于任意一个符合某种规律的有限分布, 只要提供足够多个数的ReLU神经元, 就能依靠高维空间的折叠, 而生成匹配这种分布的高维空间图形

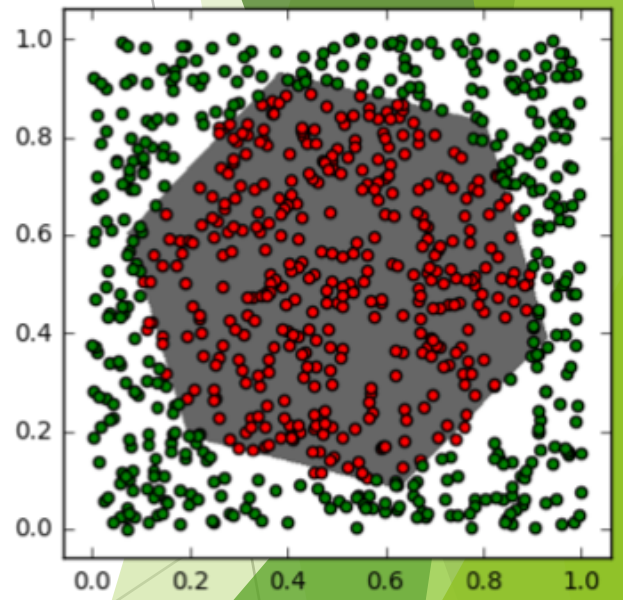
$$\text{softmax}(X) = \frac{1}{\sum_{i=0}^{n} e^{X_i}} \begin{bmatrix} e^{X_0} \\ e^{X_1} \\ \vdots \\ e^{X_n} \end{bmatrix}$$

神经网络ReLu分类示意图



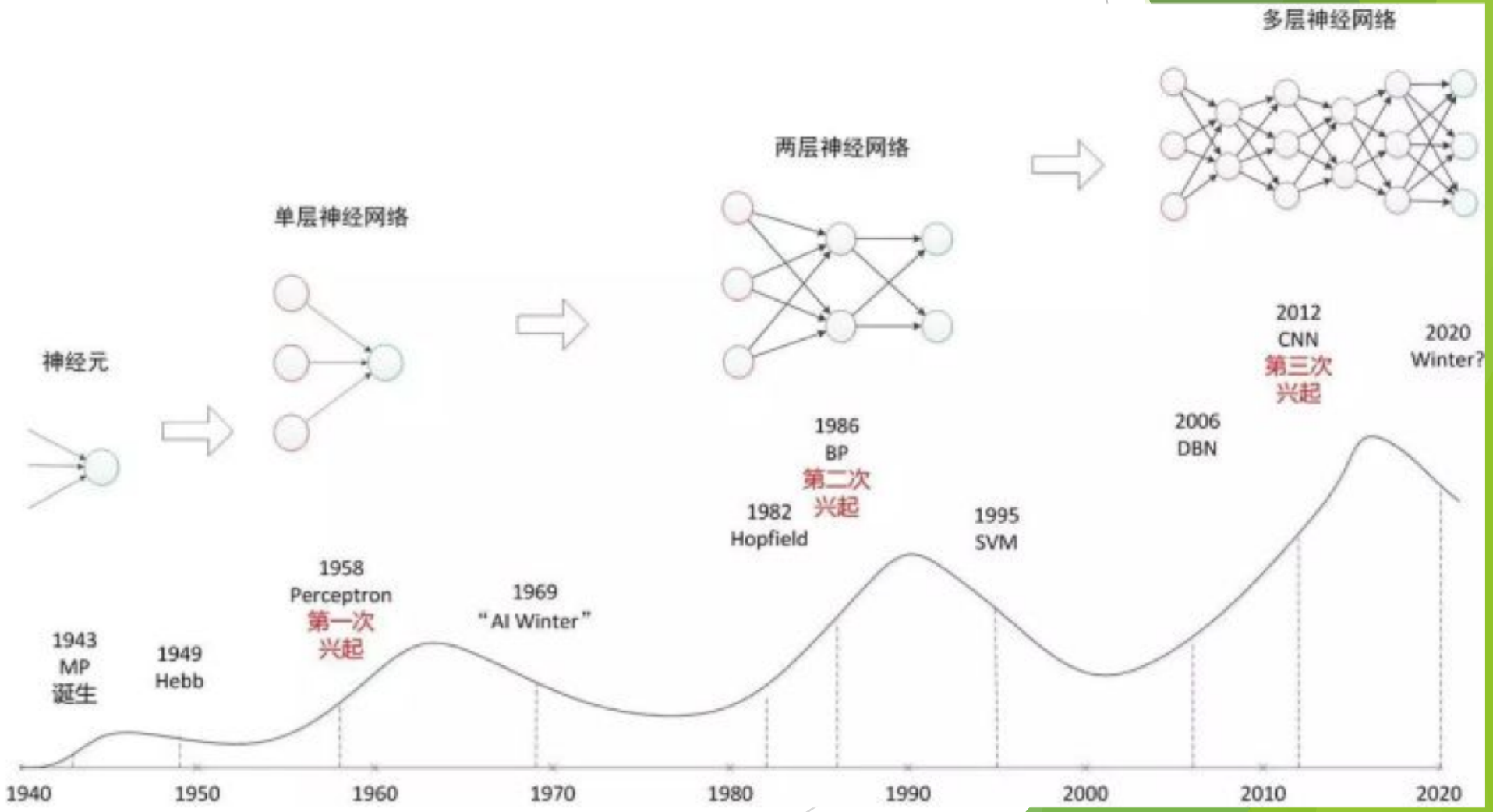






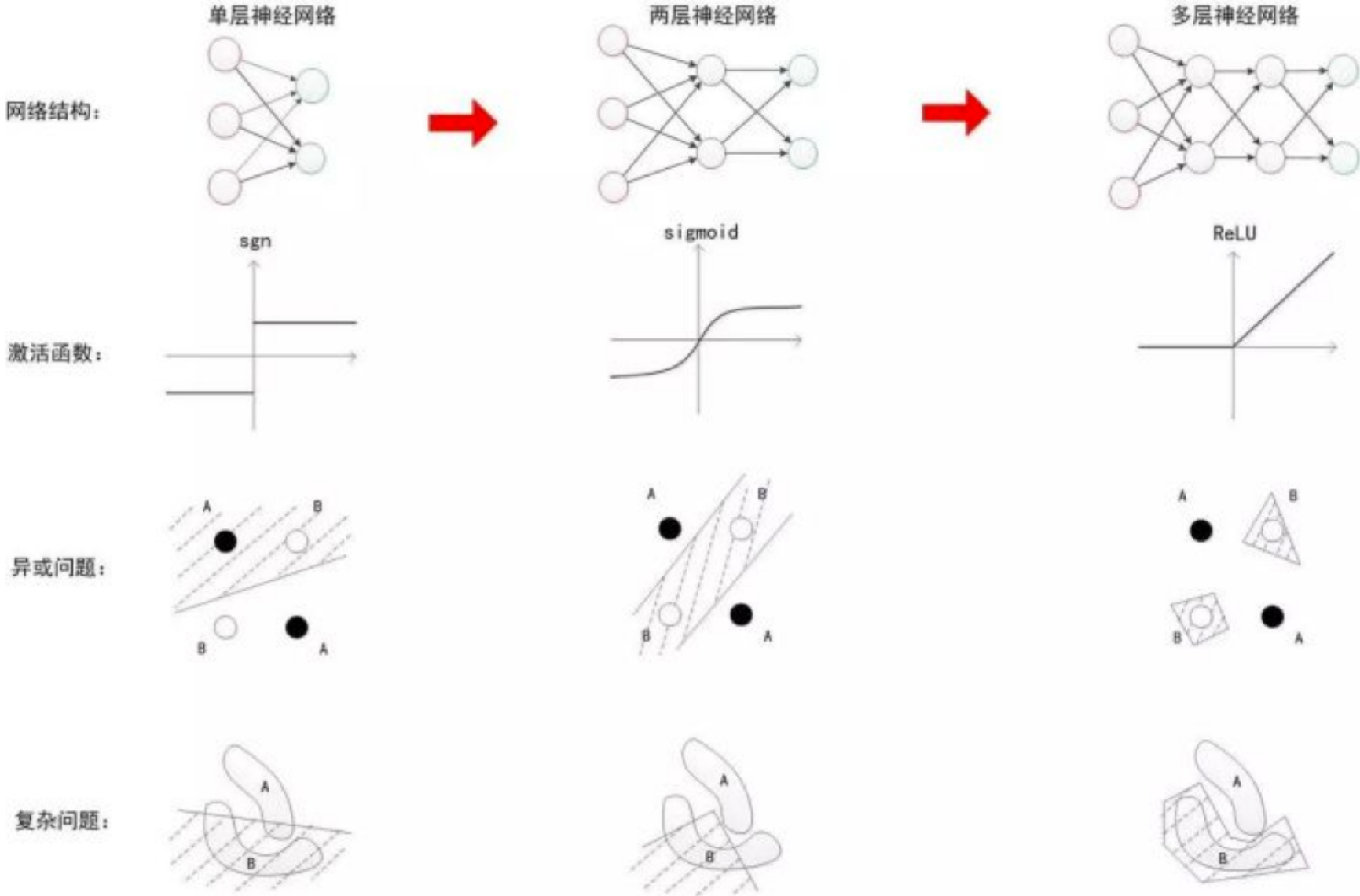
神经网络发展示意图

“三起三落”



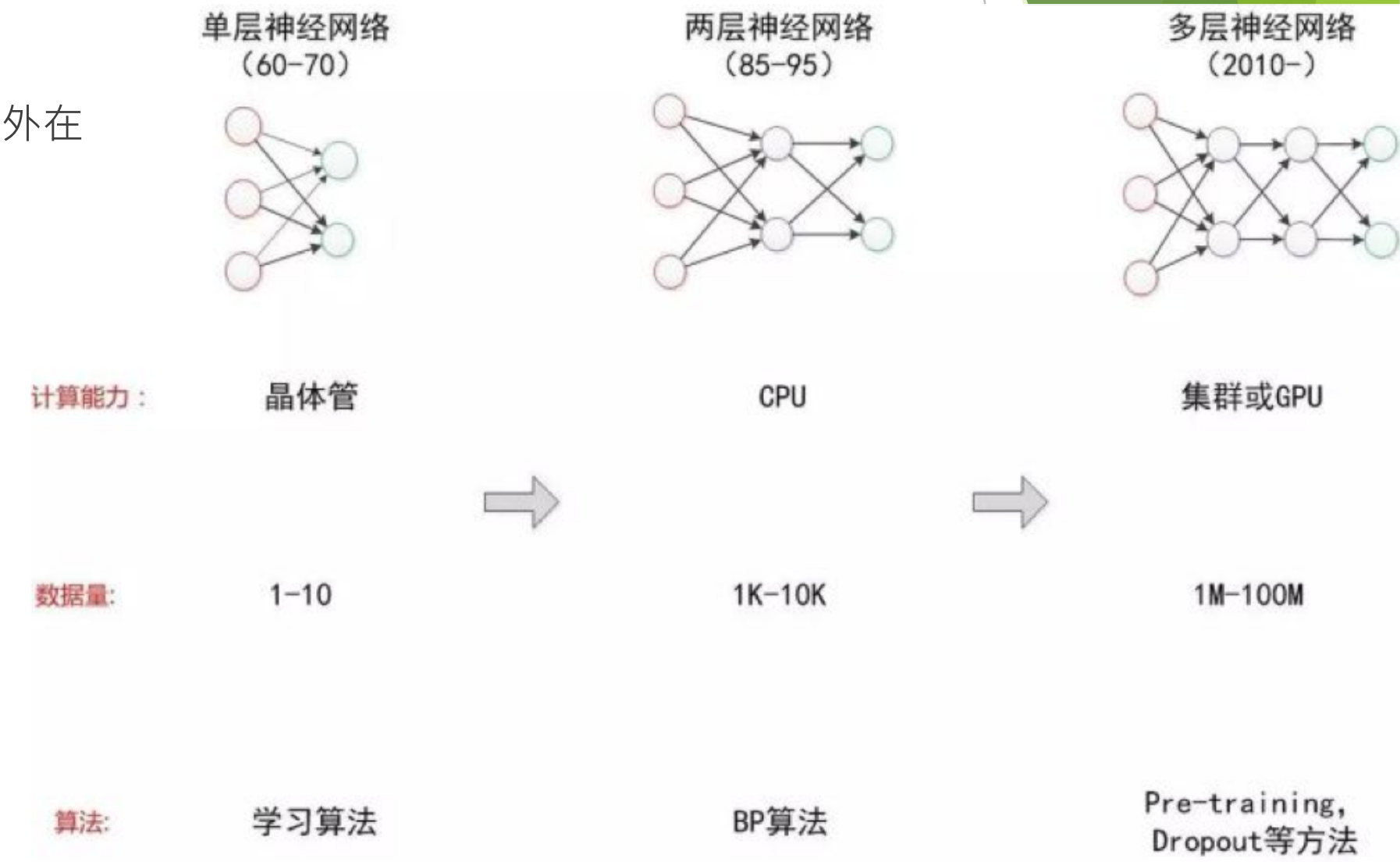
神经网络的效果

- 随着神经网络的发展，其表示性能越来越强。
- 从单层神经网络，到两层神经网络，再到多层神经网络，随着网络层数的增加，以及激活函数的调整，神经网络所能拟合的决策分界平面的能力。
- 随着层数增加，其非线性分界拟合能力不断增强。



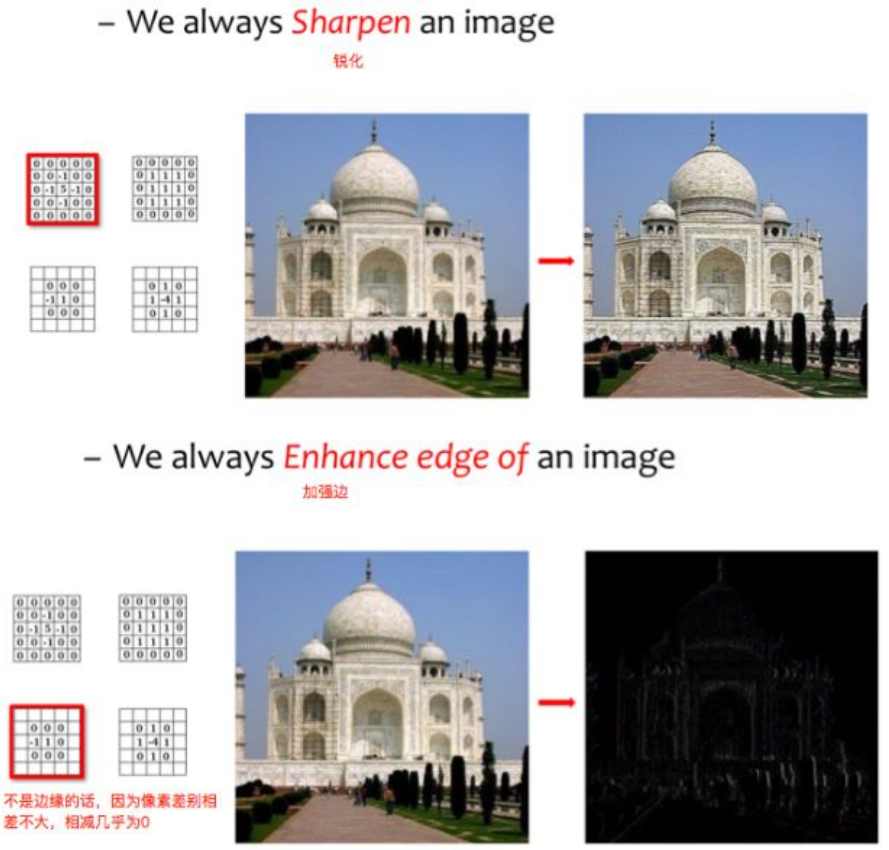
神经网络发展示意图

- 神经网络的发展背后的外在原因可以被总结为：
  - 更强的计算性能
  - 更多的数据
  - 更好的训练方法



神经网络适用场景

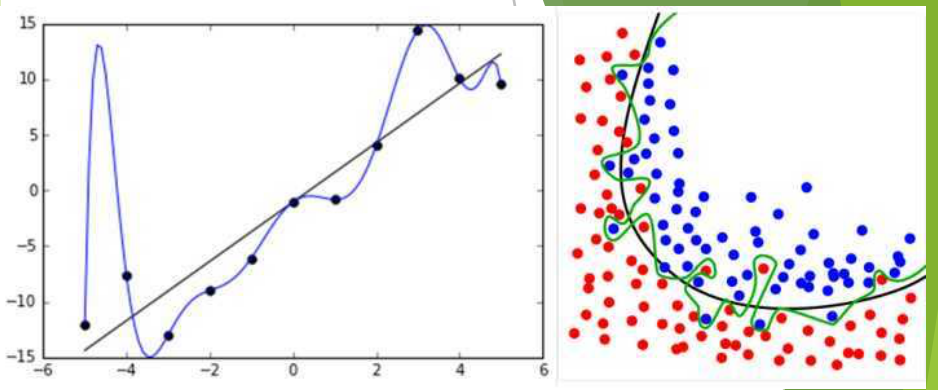
- 数据集太小，数据样本不足时，深度学习相对其它机器学习算法，没有明显优势。
- 数据集没有局部相关特性，目前深度学习表现比较好的领域主要是图像 / 语音 / 自然语言处理等领域，这些领域的一个共性是局部相关性。图像中像素组成物体，语音信号中音位组合成单词，文本数据中单词组合成句子，这些特征元素的组合一旦被打乱，表示的含义同时也被改变。对于没有这样的局部相关性的数据集，不适于使用深度学习算法进行处理。
- 举个例子：预测一个人的健康状况，相关的参数会有年龄、职业、收入、家庭状况等各种元素，将这些元素打乱，并不会影响相关的结果。

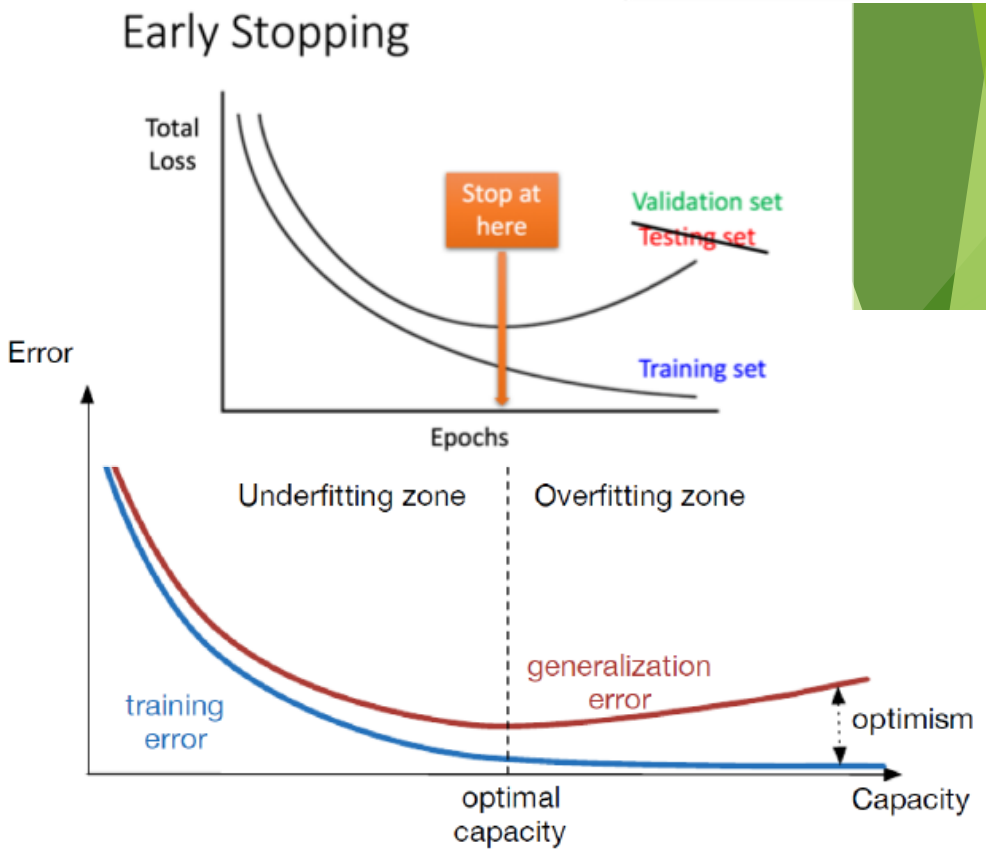


谢谢

神经网络训练学习中防止过拟合

- **Early Stop：**
  - 在模型训练过程中，在训练数据集上，代价函数会一直降低，但是训练出来的模型在测试集上的结果是先升高，在过了一定的训练轮数后，结果会在最高值附近波动甚至减低，这是因为模型学习到了训练数据集的一些独有特性，而这些特性是测试集不具有的，也就是说，这些特性不具有普遍性。
    一般来说，训练数据集划分为训练基、验证集、测试集，在训练过程中，每经过一个epoch，就在验证集上进行测试，当测试结果经过几轮不再提高时，就停止训练。
- **data expending（扩大训练数据）**
- **drop out(随机丢弃)**
- 在网络训练更新的过程中，我们随机抑制一些**隐层神经元**的表达（比如随机抑制50%的神经元），在每一个batch的训练过程中，先根据删去后的网络预测结果，然后用BP算法更新网络参数。训练很多epochs后，相当于得到了非常多的神经网络（可以说无穷多个），所以drop out会非常有效防止过拟合。
- **加入正则约束**





Dropout解决过拟合问题

- 首先随机（临时）删掉网络中一半的隐藏神经元，输入输出神经元保持不变（下图中虚线为部分临时被删除的神经元）
- 然后把输入x通过修改后的网络前向传播，然后把得到的损失结果通过修改的网络反向传播。一小批训练样本执行完这个过程后就按照随机梯度下降法更新（没有被删除的神经元）对应的参数（w，b）。
- 然后继续重复这一过程：
  - 恢复被删掉的神经元（此时 被删除的神经元 保持原样，而没有被删除的神经元已经有所更新）
  - 从隐藏神经元中随机选择一个一半大小的子集 临时删除掉（备份被删除神经元的参数）。
  - 对一小批训练样本，先前向传播然后反向传播损失并根据随机梯度下降法更新参数（w，b）（没有被删除的那一部分参数得到更新，删除的神经元参数保持被删除前的结果）
- 不断重复这一过程。