



Lecture 4

汇报人：扣扣





大纲

- 初识神经网络
 - CNN





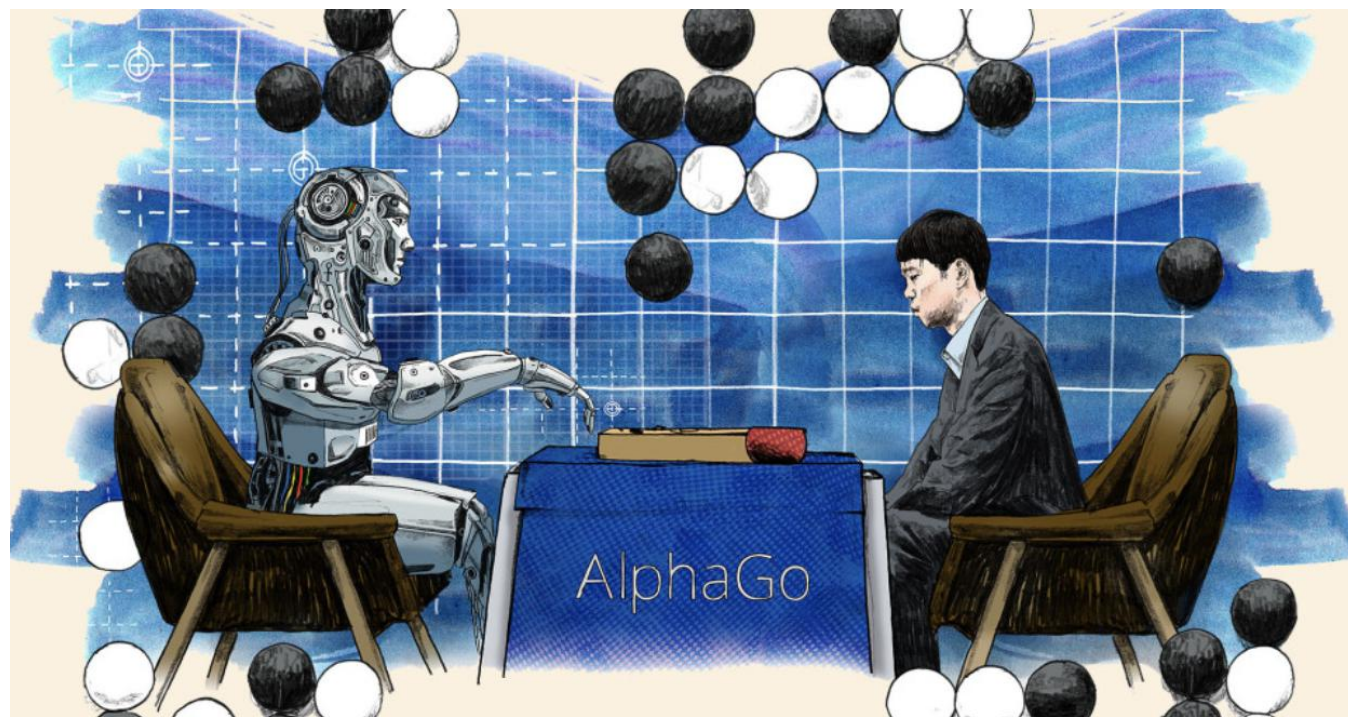
初识神经网络





大纲

- 引子：从猫的视神经实验说起
- 基础知识
 - 神经网络与大脑
 - 人类神经元
 - 人工神经元
 - 激活函数
- 感知机
- 多层感知机
- 误差反向传播算法

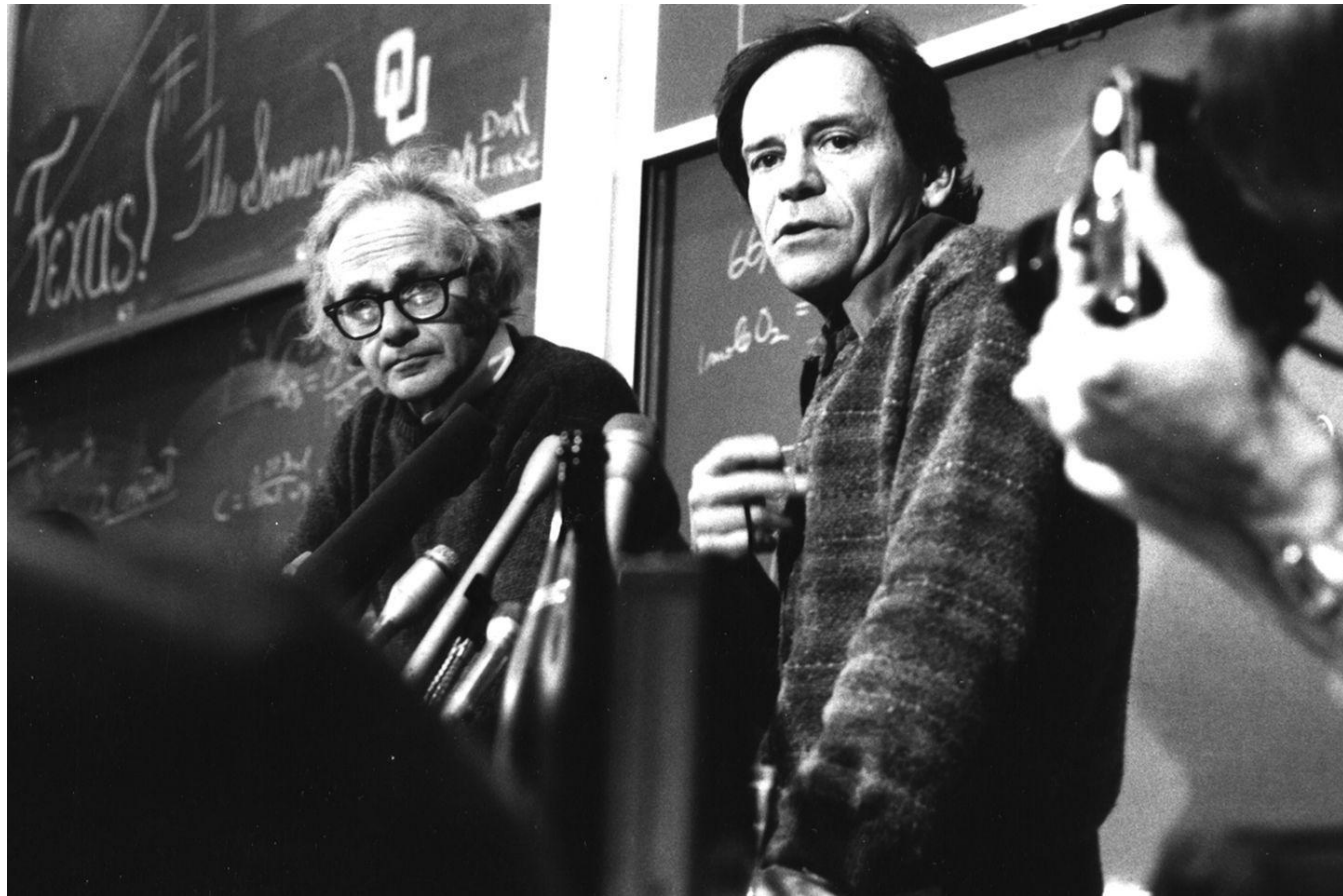


引子：从猫的视神经实验说起

- 为什么刚出生的婴儿没有远近的概念？
- 为什么在某种语言中如果描述某两种颜色的词汇若为同一词汇，以此语言为母语的人便无法分辨出这两种颜色？
- 为什么先天盲的人即使在后天完全复明后，还是存在诸多视觉认知障碍？
- 为什么有些人后脑勺受伤后会出现短暂性的失明？

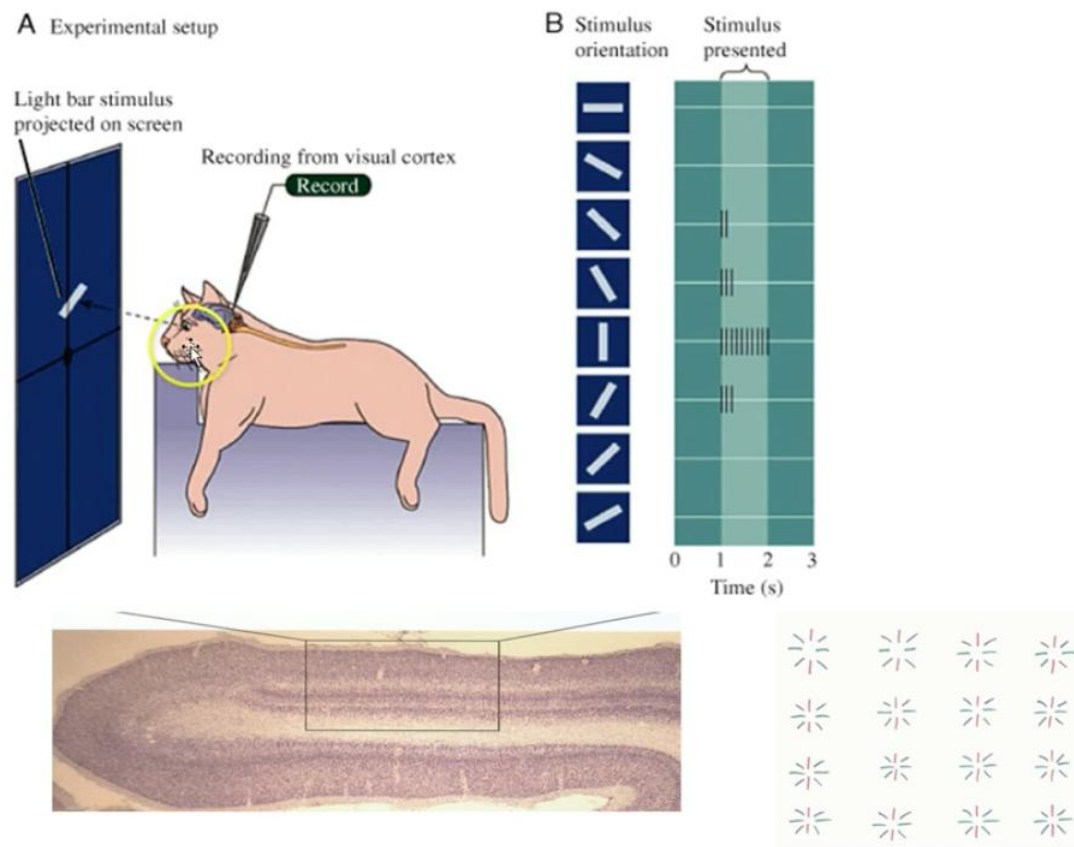
我们并不只是简简单单地用眼睛看见这个世界，眼睛只是接收图像信息的工具，由大脑深度加工信息以后，我们才能真正“看见”这个世界。

引子：从猫的视神经实验说起



1959年，加拿大-美籍神经科学家大卫·休伯尔（David Hunter Hubel），与合作者托斯坦·威泽尔（Torsten N. Wiesel）以猫为实验对象，通过对猫的初级视皮层的探究，首次发现了大脑的视觉加工机制，发表了论文《Receptive fields, binocular interaction and functional architecture in the cat's visual cortex》，并因此共同获得了1981年的诺贝尔生理学或医学奖。

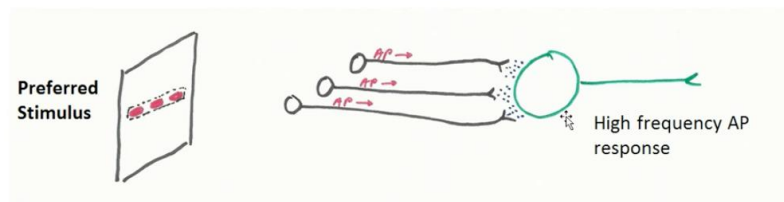
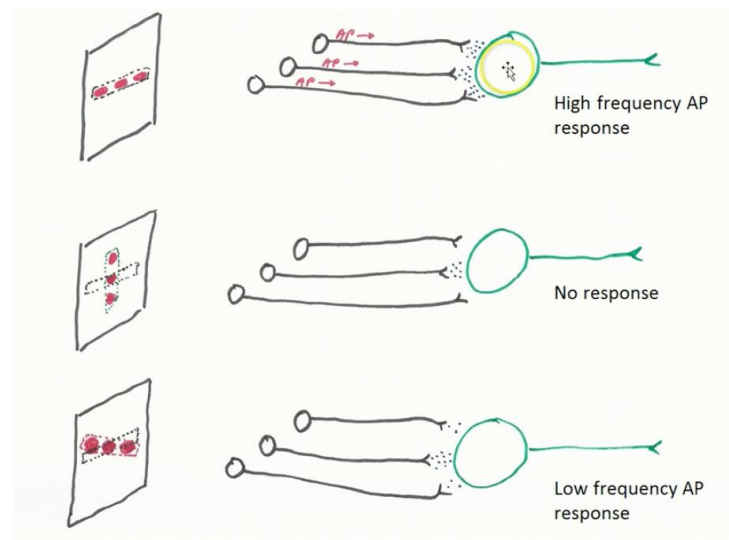
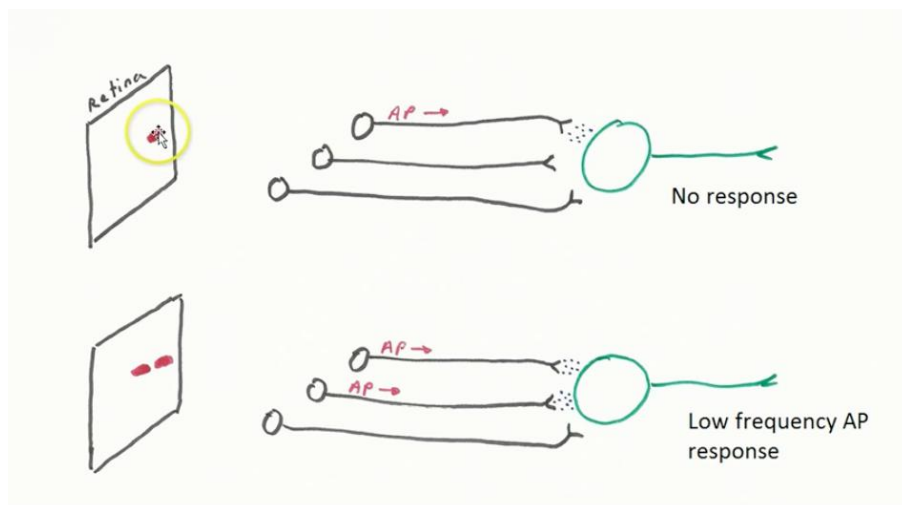
引子：从猫的视神经实验说起



在实验中，将玻璃包被的钨丝微电极插入麻醉猫的初级视皮层中的神经元，然后在置于猫前方的幕布上投射出一条光带，改变光带的空间方位角度，用微电极记录神经元的激活状态。

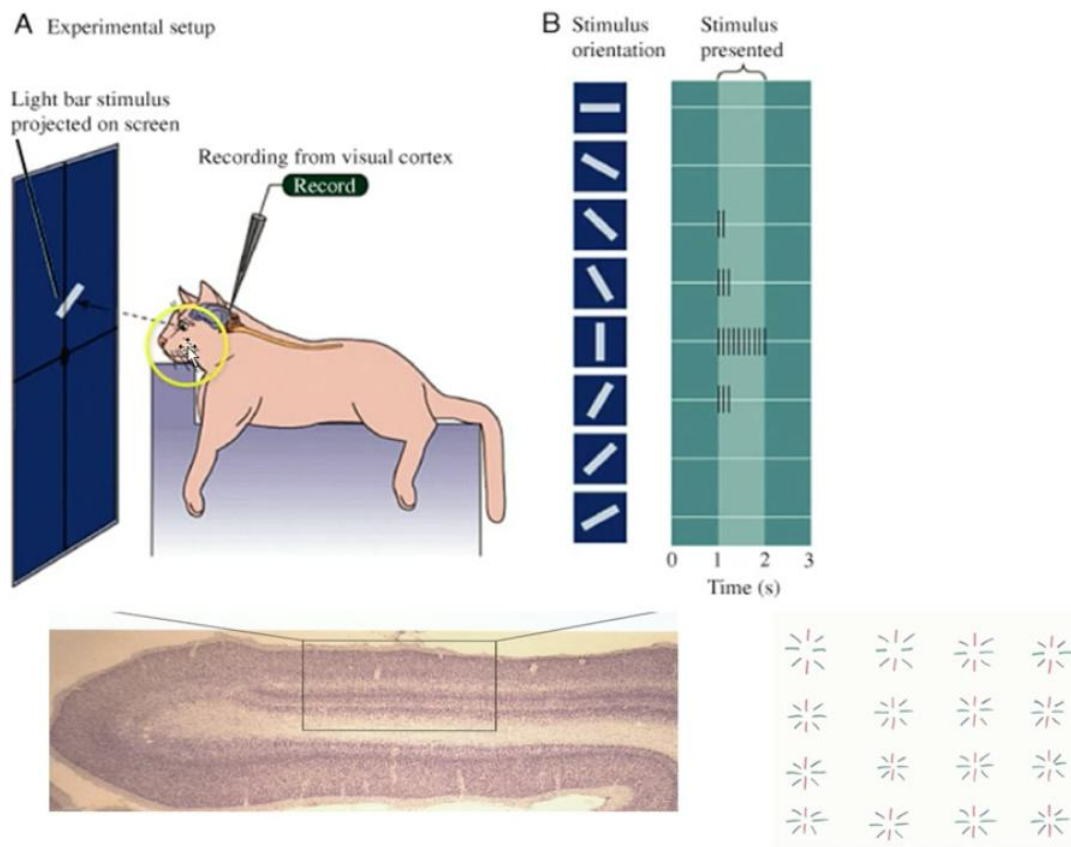
发现，当光带处于某个空间方位角度时，神经元激活状态最为强烈。而且，不同的神经元对不同空间方位的偏好不尽相同。另外，不同神经元对亮光带和暗光带的反应模式也不相同。

引子：从猫的视神经实验说起



The target cell's activity **REPRESENTS** the presence of the **preferred stimulus**... a 'horizontal edge' at that location on the retina.

引子：从猫的视神经实验说起

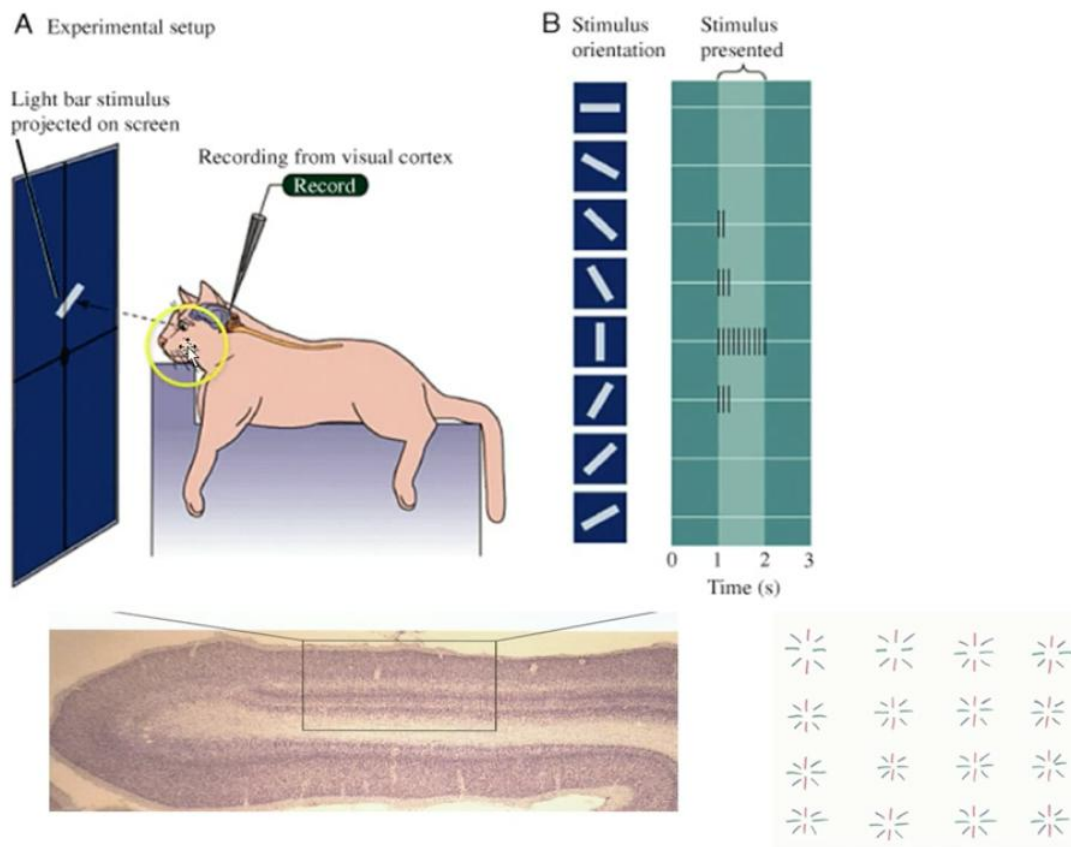


休伯尔和威泽尔将实验中的这种神经元称为**简单细胞**。而初级视皮层里的另外一些神经元，叫做**复杂细胞**，这些细胞对于其感受野中的边界信息比较敏感，还可以检测其感受野中复杂的运动信息。

基于此研究，后续科学家们又作了进一步的视觉实现，逐步发现了一些大脑视觉认知规律：

- 在视觉系统中，不同的细胞有不同的分工，如有些只能处理简单的线条信息，有些则能处理更复杂的形状；
- 在初级细胞加工完信息以后，会将信息传递至更高级的细胞。

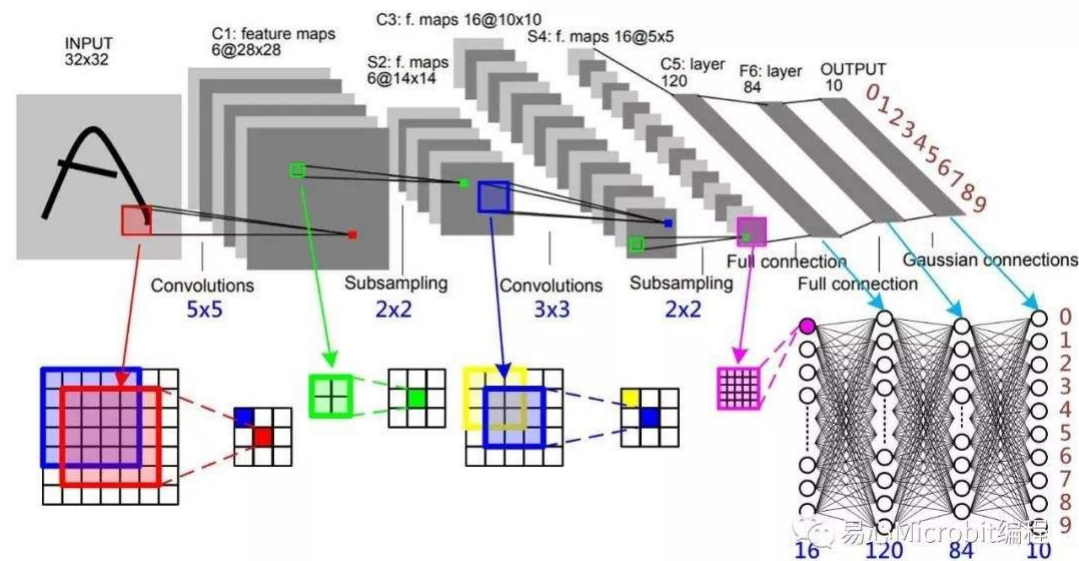
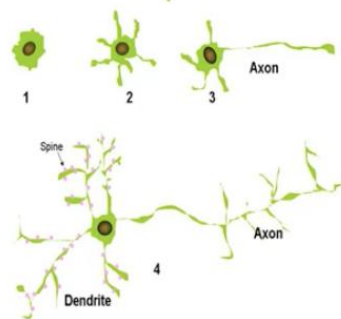
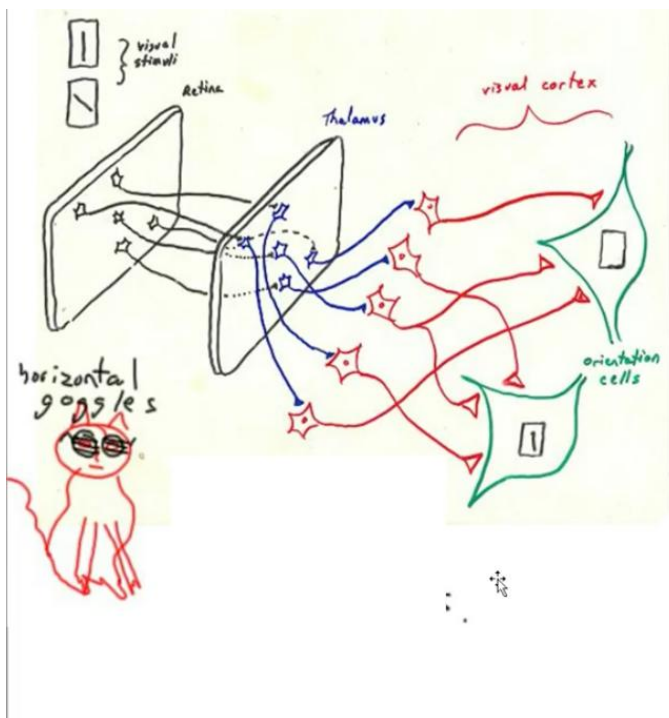
引子：从猫的视神经实验说起



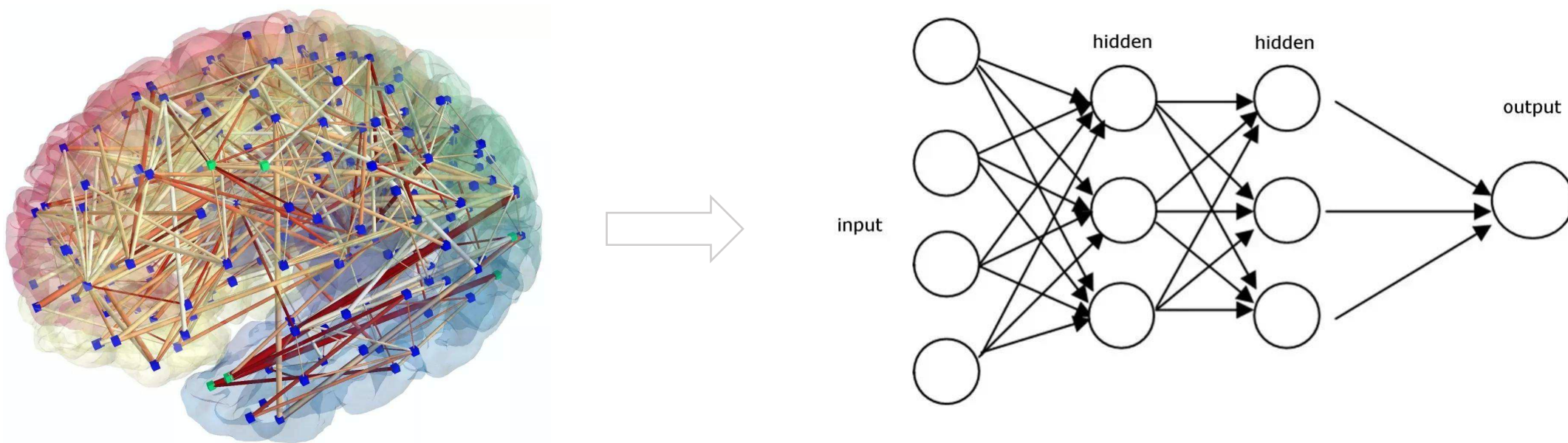
而在1980年，日本科学家福岛邦彦（Kunihiko Fukushima）从猫的视觉实验中得到启发，提出了Neocognitron（新认知机），在论文《Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position》中提出了一个包含卷积层、池化层的神经网络结构。

尔后的1998年，在Neocognitron的基础上，Yann Lecun在论文《Gradient-Based Learning Applied to Document Recognition》中提出了LeNet-5，将BP算法应用到这个神经网络结构的训练上，形成了当代卷积神经网络的雏形。

引子：从猫的视神经实验说起



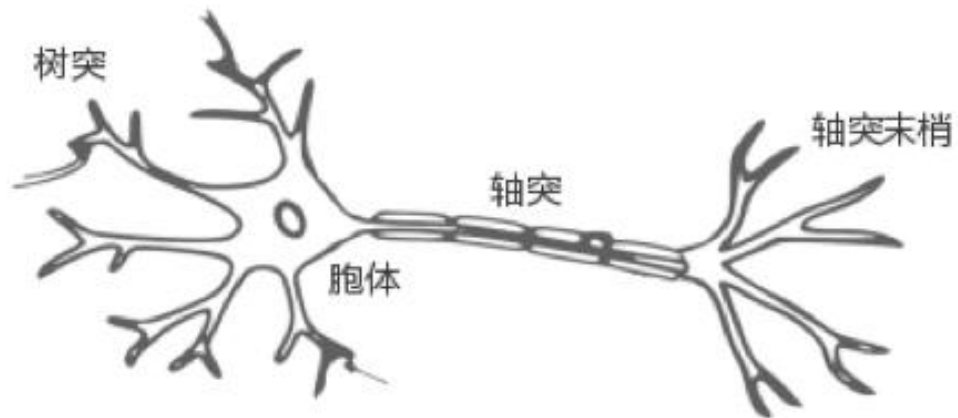
基础知识：神经网络与大脑



人类虽然拥有智慧，但对智慧是如何产生的却不得而知，对于大脑结构的模仿或许是一个探索的起点，神经网络的起点就在这里(认识智慧的产生机理，才能创造智慧)。

神经网络的核心：部分模拟一个人的大脑。

基础知识：人类神经元



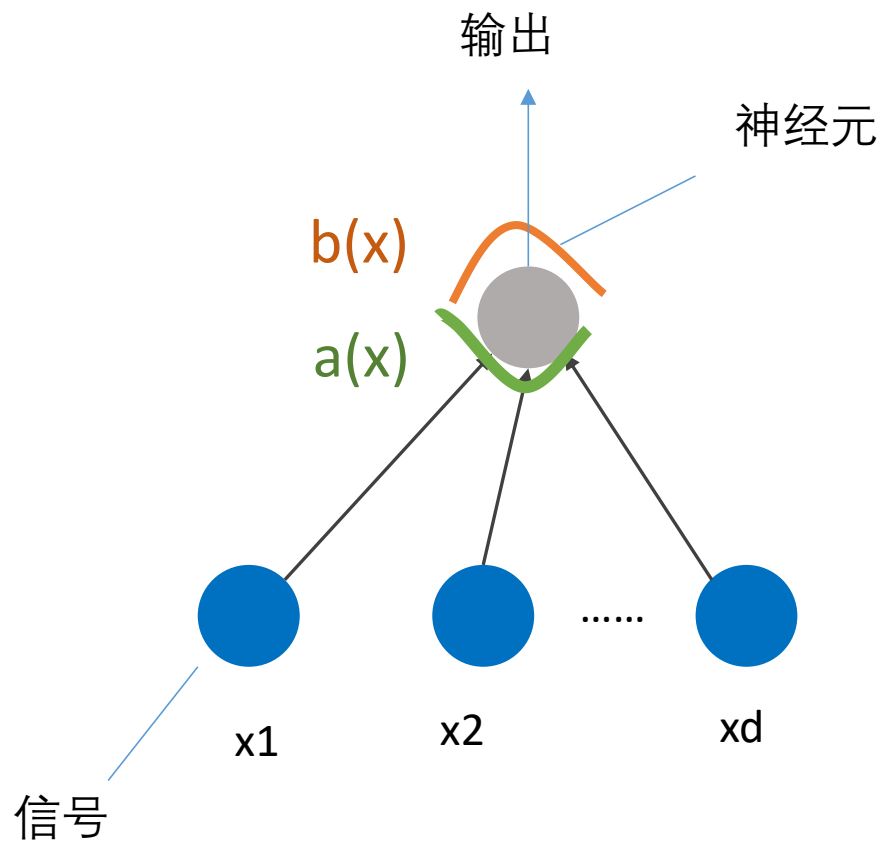
人类神经系统的基本单元是神经元，约有1000亿个，是一种高度分化的细胞。神经元能够接受、整合、传导和输出信息，实现信息的传递、重组以及交换，正是基于此，人类大脑能够作出极其复杂的决策。

胞体以及树突上的受体用于接收信息，当信息达到特定阈值后产生动作电位并且由轴突传递给另外的神经元。

由于有多个轴突末梢，因此信息可以传递给多个神经元，反之，同一个神经元也可以接收来自多个神经元的信息。

注意到这里有两个关键点：特定阈值；神经元之间多对多的关系。

基础知识：人工神经元



$$a(x) = b + \sum_i w_i x_i = b + w^T x$$

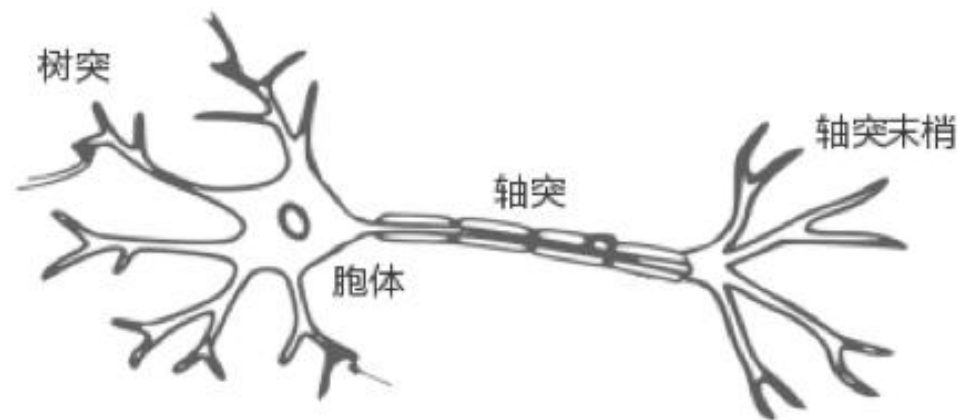
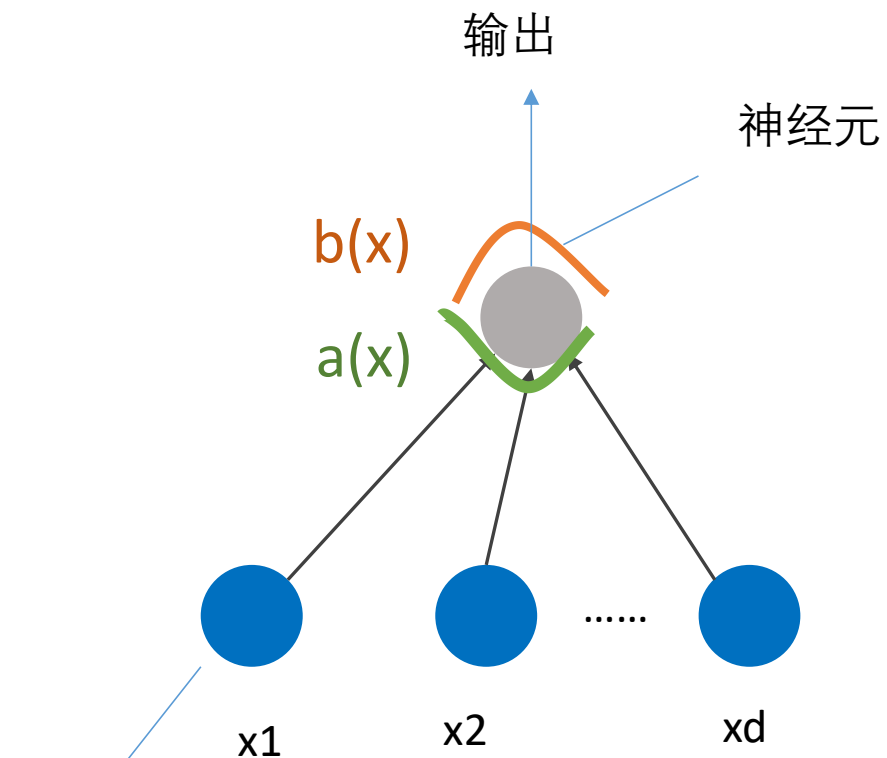
$$b(x) = g(a(x)) = g(b + w^T x)$$

↓
激活函数

神经元可以看作是一个处理信号的模块。

人们便是基于此设计了人工神经元的结构，其中， d 表示有 d 个 x 的输入， w 表示与输入进行线性运算的参数， b 为偏移量， f 表示非线性变化（也叫作激活函数）。

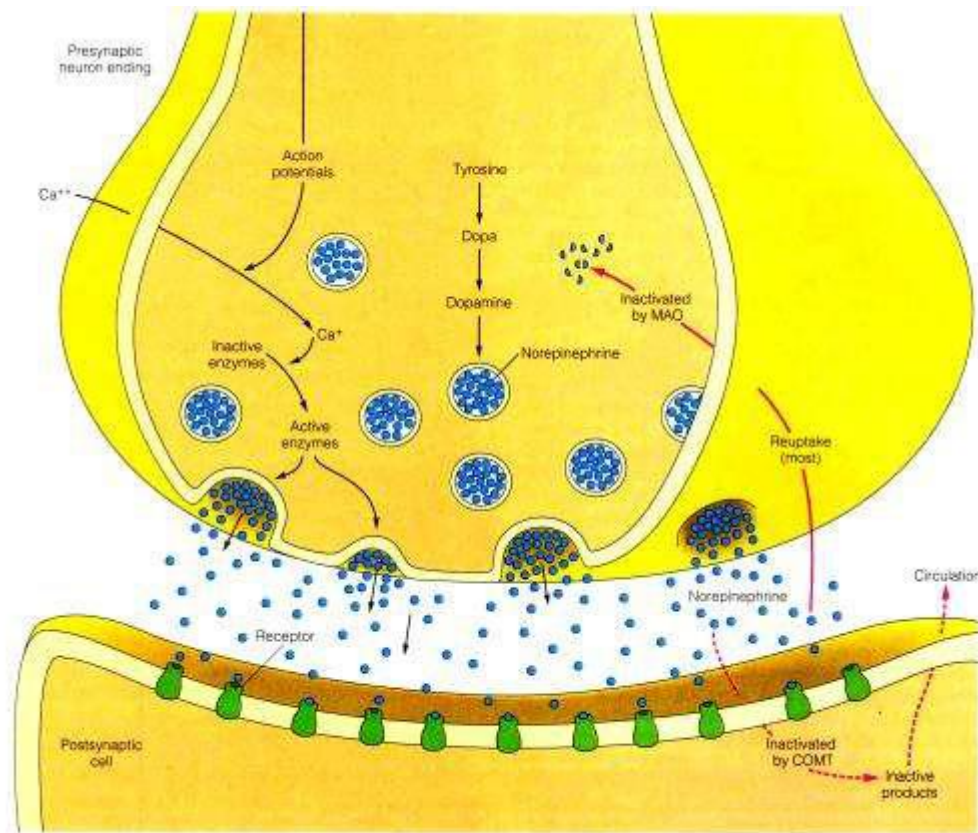
基础知识：人工神经元



由人工神经元的结构可知，其对信息的处理可以分为两部分：

- 线性运算
- 非线性运算（也称为激活函数运算）

基础知识： 激活函数

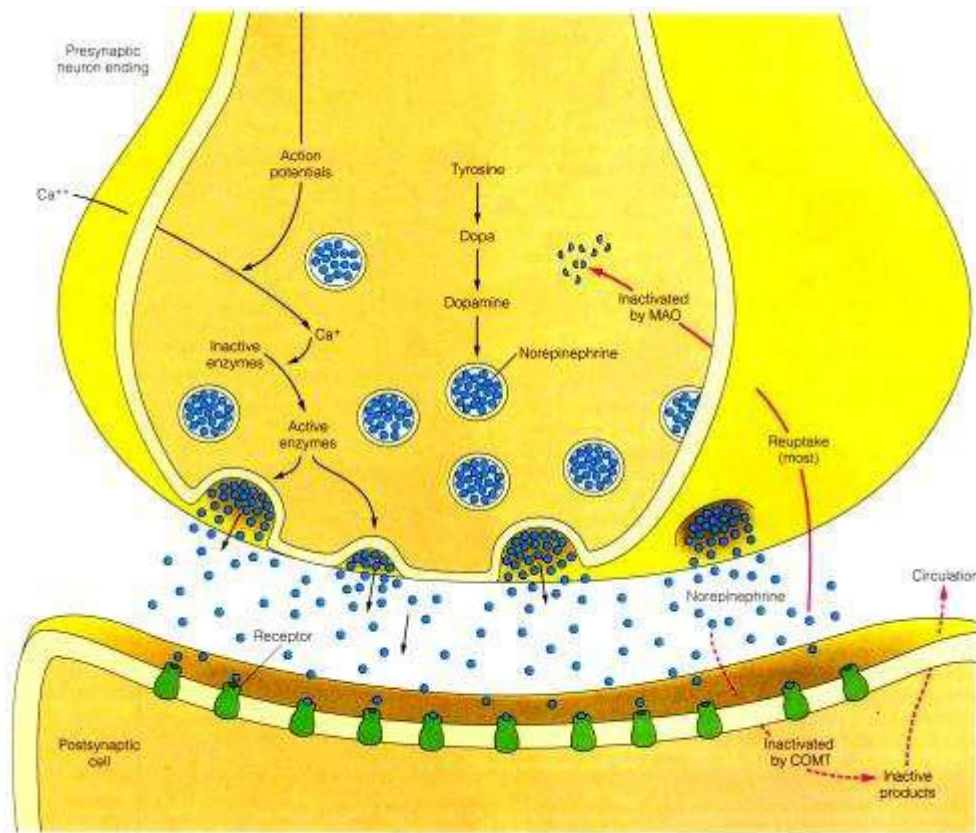


这激活函数又有何用呢？其实这也是对真实神经元的模拟。

在日常生活中，可以观察到，每个人对疼痛的敏感程度不一样，比如有的人稍微刺一下就很痛，有的人要用力刺一下才会痛。也就是说，对于疼痛的传递，需要超过一定的阈值才能传导到人类的感觉层面，并且由于每个人阈值的不同导致了对于同一刺激产生不同反应。

确实，从真实神经元的信息传导层面而言，当神经元之间的递质达到一定浓度时，兴奋才能进行传递。

基础知识： 激活函数

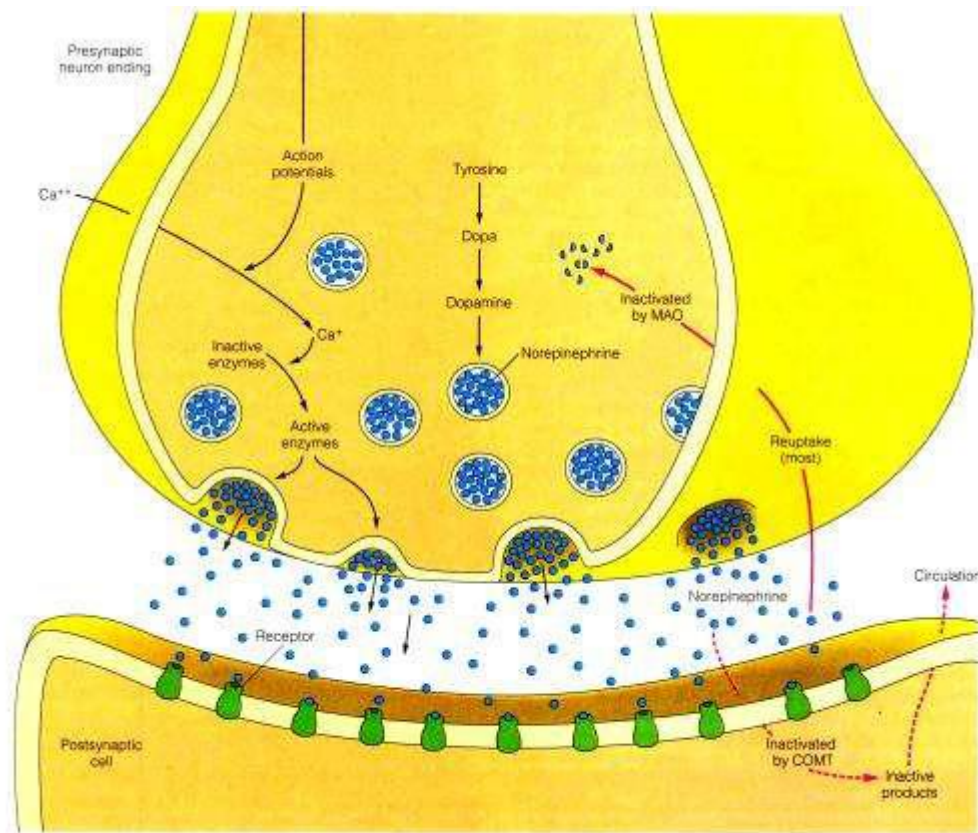


在真实世界中，输入（疼痛的刺激）与输出（疼痛的感觉）并不是呈简单的线性关系，往往用**非线性关系**来描述更确切。

如何使两者之间呈非线性关系？

非线性函数或者说激活函数：从“激活”本身的语义出发，激活函数可通俗地理解为，**定义在某些特定刺激条件下产生特定活动的函数**。

基础知识： 激活函数



数学角度：

如果神经元结构中只存在线性计算，即使有多个神经元组合计算，最终也属于线性计算，因此不能拟合出复杂的函数。

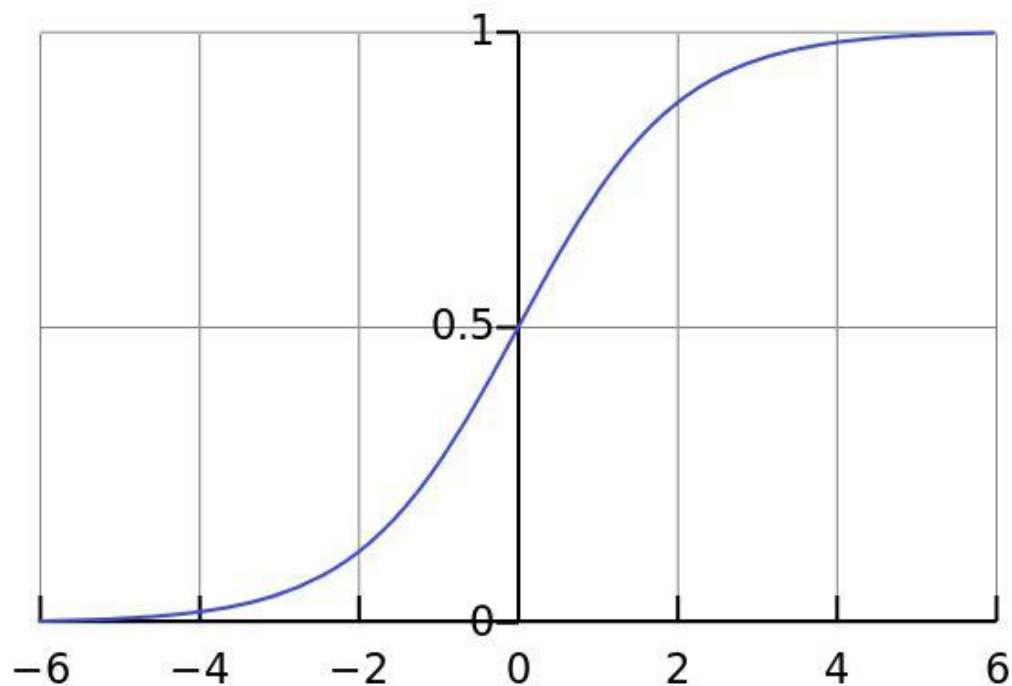
加入非线性计算之后，可以增强神经网络的表达能力，理论上能表示任何复杂的运算，Universal approximation theorem (Hornik et al., 1989; Cybenko, 1989, 中文称为万能近似定理)。因此激活函数是神经网络中的重要结构。

附：

在实践中UAT可能基于以下原因无法实现：

- 用于训练的优化算法可能找不到用于期望函数的参数值
- 训练算法可能由于过拟合而选择了错误的函数

基础知识： 激活函数

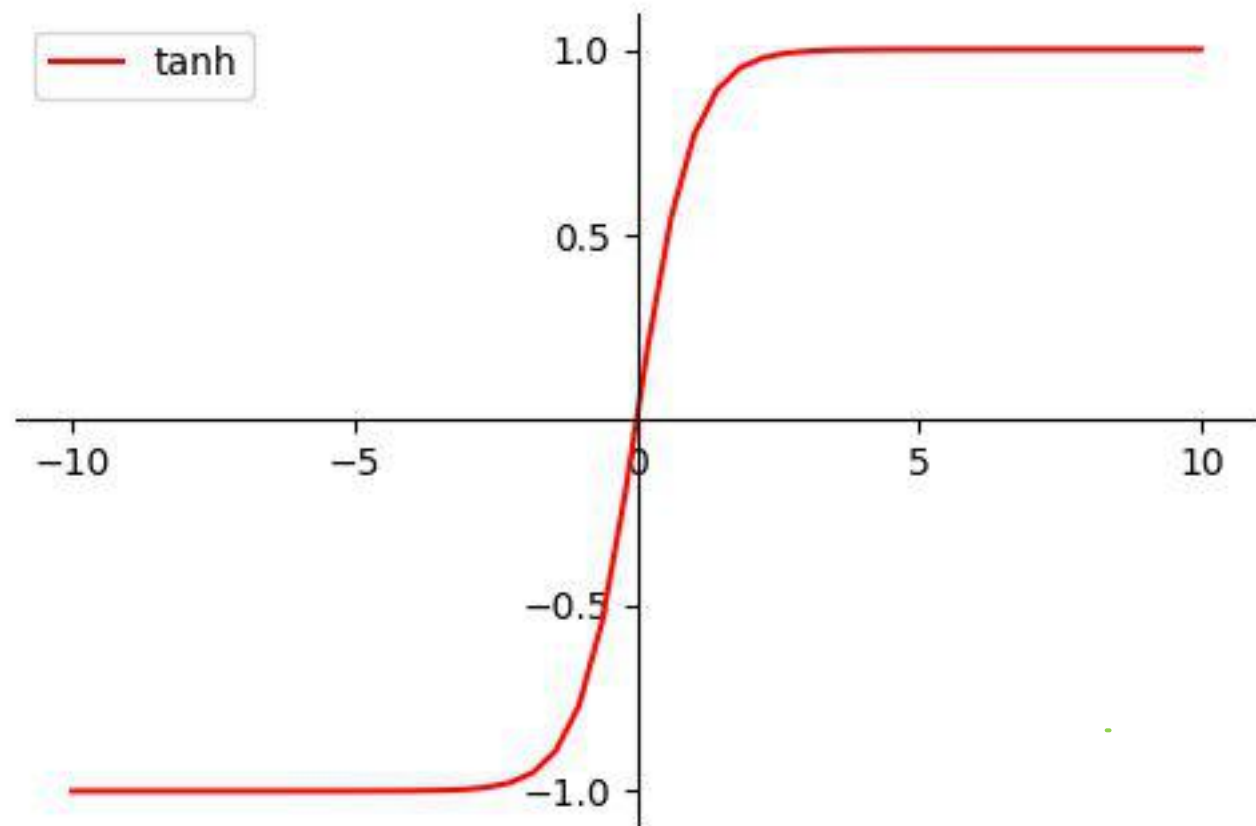


sigmoid能够把输入映射到0和1之间，在物理上最接近神经元，可以被表示成概率，或者用于数据的归一化。但是它有两个严重的缺陷：

其一，**梯度消失**——导数 $f'(x) = f(x)(1-f(x))$ ，当x趋于无穷时， $f(x)$ 的两侧导数逐渐趋于0。在后向传递时，sigmoid向下传递的梯度包含了一个 $f'(x)$ 因子，因此，一旦落入两端的平滑区， $f'(x)$ 就变得接近于0，导致了向后传递的梯度也非常小。此时，网络参数很难得到有效训练，这种现象被称为梯度消失，一般在5层以内就会产生梯度消失的现象。

其二，sigmoid函数的输出均大于0，使得输出不是0均值，称为**偏置现象**。

基础知识： 激活函数



tanh的特点有：

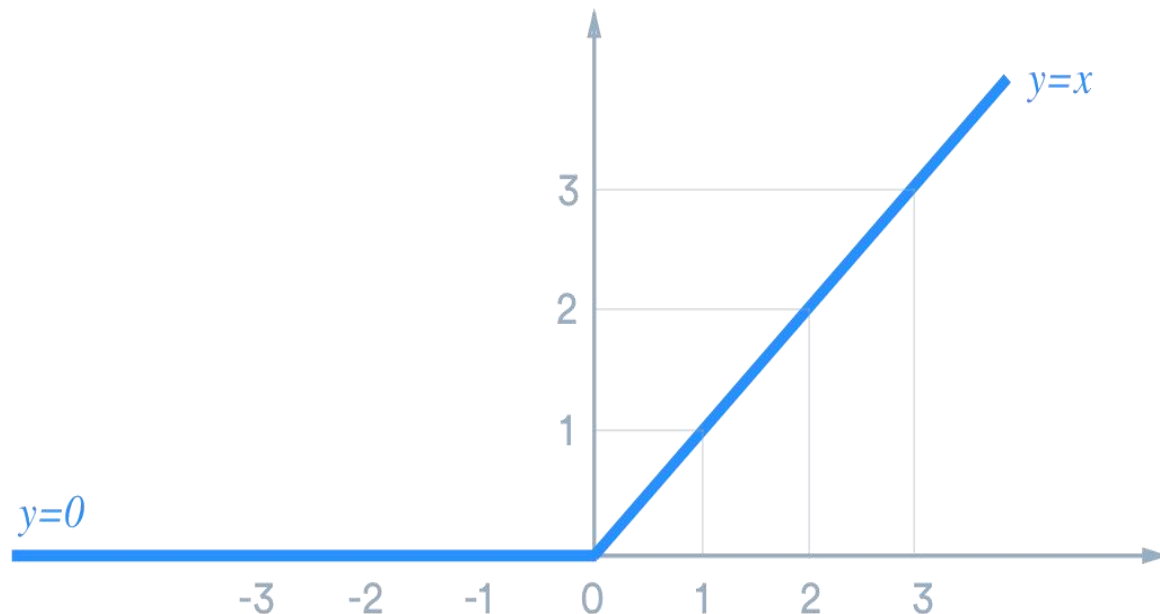
其一，能够把输入映射到-1和1之间，并且以0为中心对称。

其二，梯度比sigmoid形梯度强（导数更陡峭），因此收敛更快。

其三，同时也存在着与sigmoid函数类似的对两端值不敏感，梯度消失的问题。

基础知识： 激活函数

relu全称为Rectified Linear Units，可以翻译成线性整流单元或者修正线性单元。虽然其表达式非常简单，但却有其独特的优势：

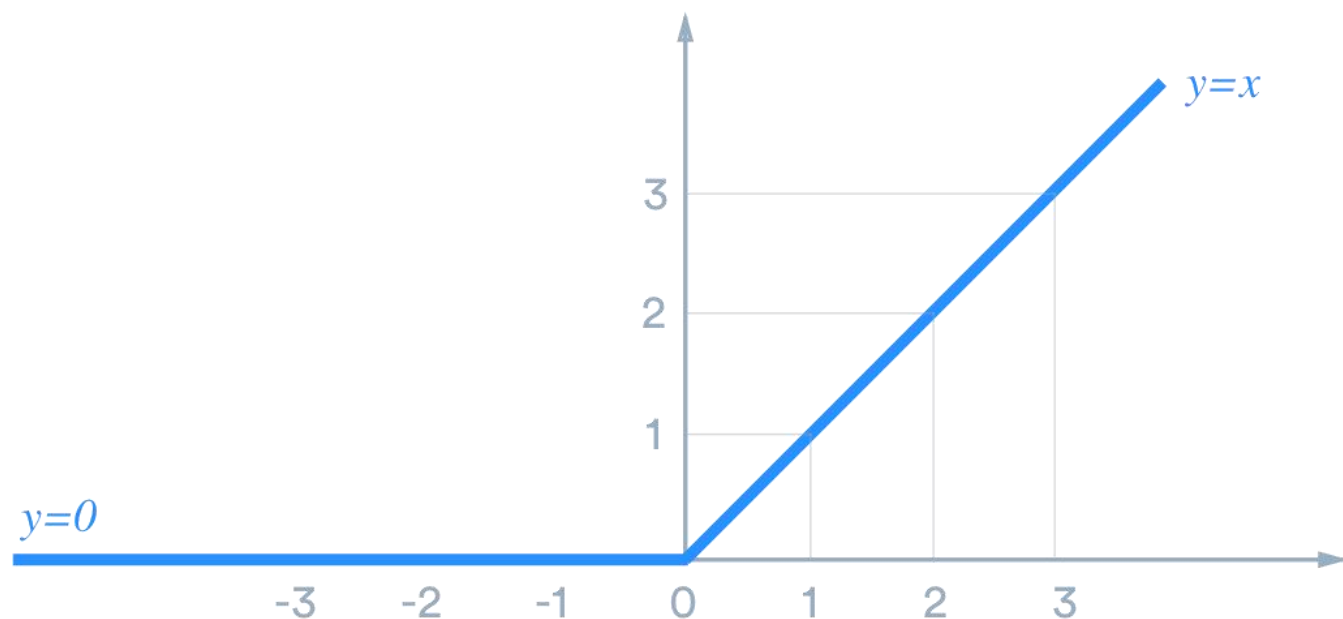


其一，计算非常简单，对于复杂的深度学习训练而言，能节省不少训练时间。

其二，这种单边的输出特性和生物学意义上的神经元阈值机制十分相像：当输入小于0时，输出为0，当输入大于等于0时，输出保持不变。也就是说，当输入小于0时，此神经元不产生任何作用，处于失活状态，这也和人脑的工作机制类似。在2003年，Lennie等神经科学家推测，在一般情况下，大脑同时被激活的神经元只有1~4%，也就是说，大脑中大部分神经元处于失活状态，只有少部分神经元处理特定任务，比如人在说话时只有一部分与语言相关的大脑区域处于激活状态。

其三，从数学特性而言，当 $x > 0$ 时，梯度不变，解决了sigmoid及tanh常见的梯度消失问题。

基础知识： 激活函数



在使用上，relu一般常用于多层感知机以及卷积神经网络，但是在循环神经网络中并不常见，这是因为在多次循环神经元的操作之后，可能具有非常大的输出，与有界值的情况相比，更容易发生输出值爆炸的情况。

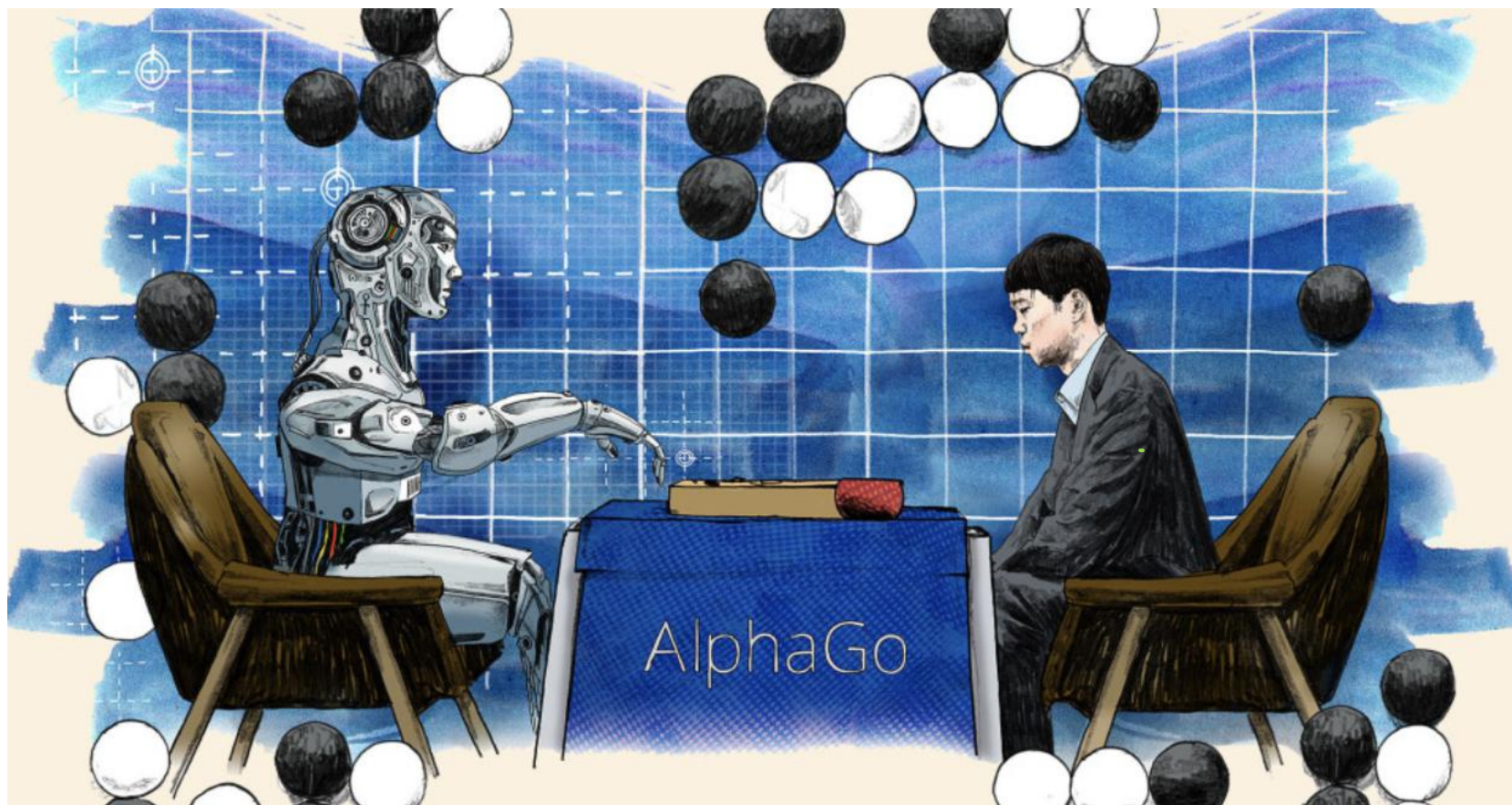
总结

- 神经网络的灵感一部分来源于人们对于**大脑本身的探索**，对于人脑的模拟可以说是研究人工智能的一大起点。
- 人工神经元包含了**线性与非线性运算**，按照**万能近似定理**，由多个及多层神经元组合的网络可以逼近任何复杂的运算，有潜力解决一些复杂的现实问题。
- 而神经网络中的非线性运算，即激活函数，目前人们构造了几十种函数，应用于各种不同的神经网络结构，深度学习工程师需要**熟悉常用的激活函数及其优缺点**，并在不同的场合下灵活运用。

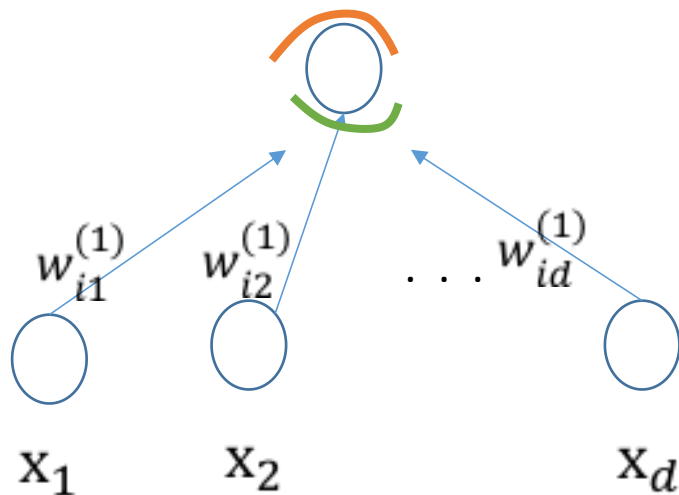
.



休息

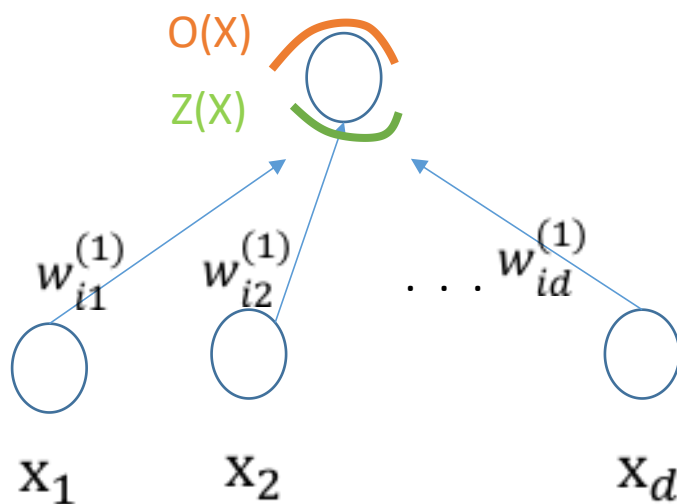


感知机



我们先简要地介绍深度学习的起源算法——感知机（Perceptron），感知机接受多个输入并且只有一个输出，模拟的便是人类神经元的基本结构，如图所示为感知机的基本结构。

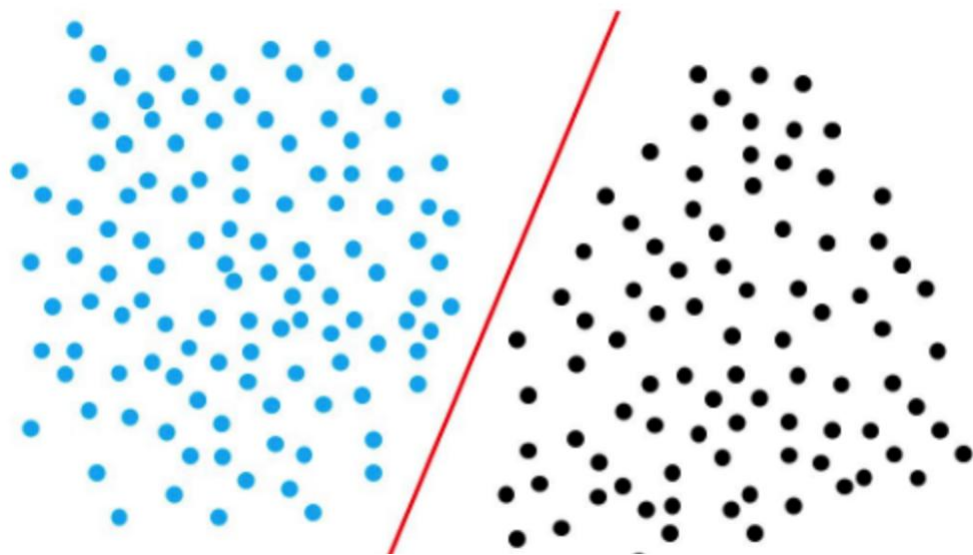
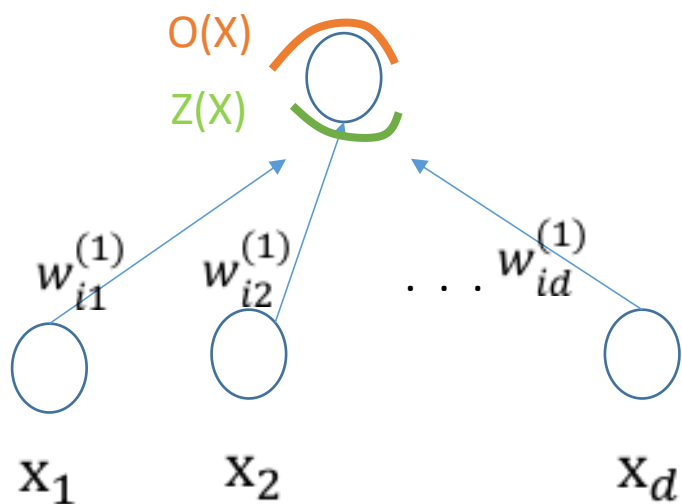
感知机



$$Z = \sum_{i=1}^d w_i x_i + b$$

$$O = \text{sign}(z) = \begin{cases} -1 & z \leq 0 \\ 1 & z > 0 \end{cases}$$

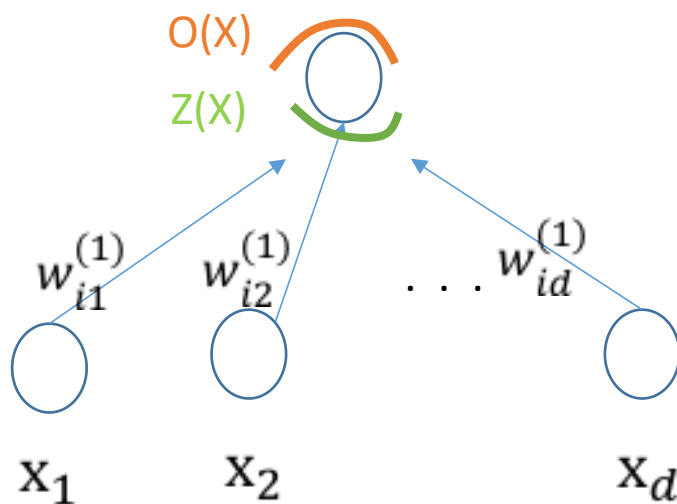
感知机



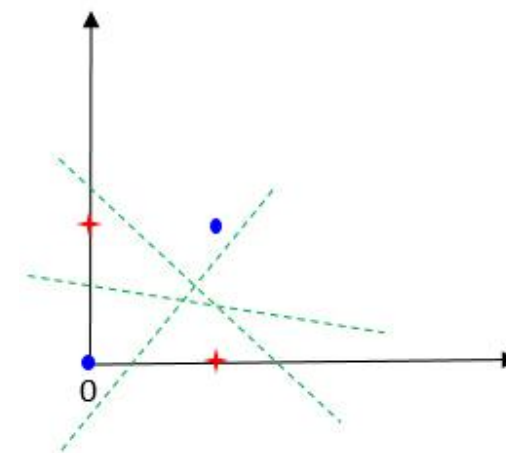
$$Z = \sum_{i=1}^d w_i x_i + b$$
$$O = \text{sign}(z) = \begin{cases} -1 & z < 0 \\ 1 & z > 0 \end{cases}$$

感知机是二分类的线性模型，其输入是实例的特征向量，输出的是事例的类别，分别是+1和-1，属于判别模型。

感知机



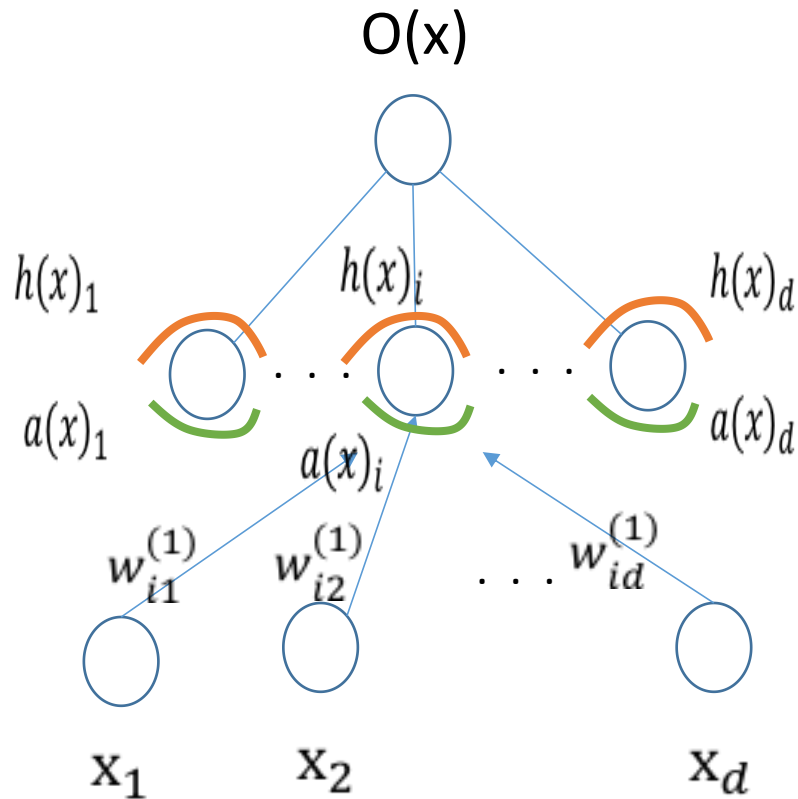
输入		输出
x_{i1}	x_{i2}	y_i
0	0	0
0	1	1
1	0	1
1	1	0



$$Z = \sum_{i=1}^d w_i x_i + b$$
$$O = \text{sign}(z) = \begin{cases} -1 & z < 0 \\ 1 & z > 0 \end{cases}$$

感知机不能解决简单的异或问题。

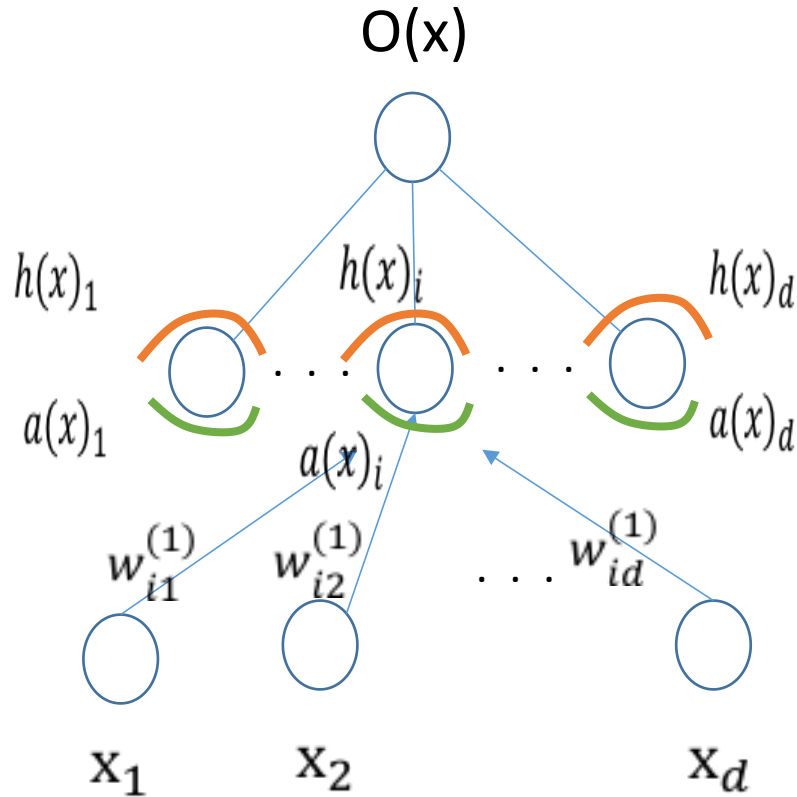
多层感知机



多层感知机 (MLP, Multilayer Perceptron)，又可称为深度神经网络 (Deep Neural Network, DNN)，是感知机的扩充结构，由输入层、隐含层（可以有多层）以及输出层构成，每一层之间都是全连接关系，如图所示。

- MLP在横向上进行了扩充，**具备多层**，因此能够进行更加复杂的运算，增强了模型的表达能力；
- 而且模型的**输出也能有多个**，可以针对多分类问题；
- 另外，感知机所应用的非线性函数（或者说激活函数）是简单的sign，处理过程过于粗暴，输出形式过于简单，而MLP结构可以应用**Sigmoid、Softmax、Tanh以及Relu**等，而且在不同地方分情况应用，拟合能力进一步增强。

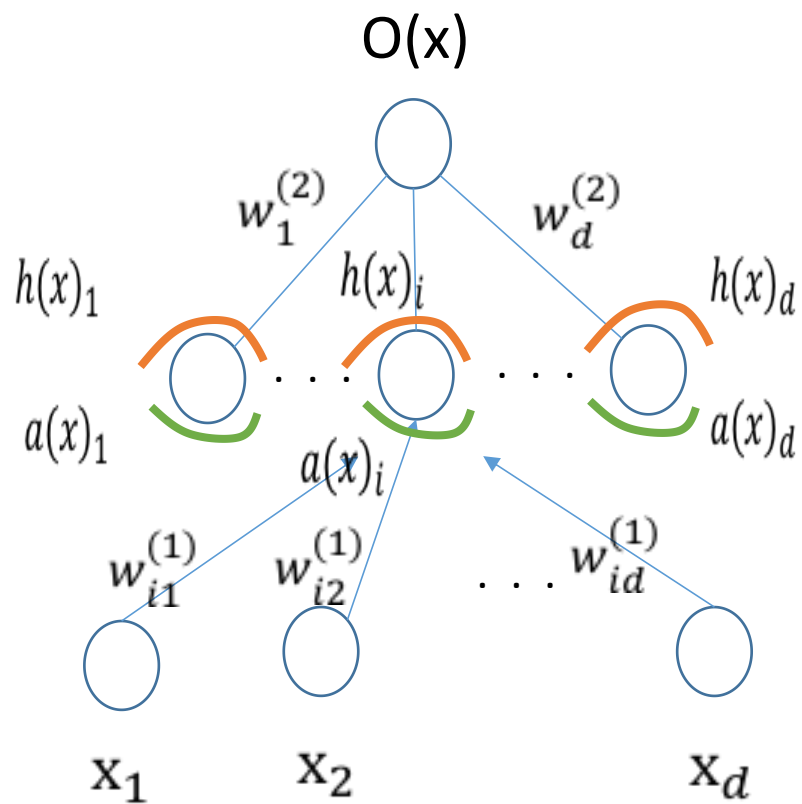
多层感知机



简单地理解为：感知机就像是人脑中单层神经元，信息处理方式简单，只能解决一些异常简单的问题，而MLP则是由众多的神经元构成的巨大网络，便能够协同运作，对信息进行复杂的加工以处理更有难度的问题。

MLP在上世纪80年代就相当流行，但由于环境限制效果不如支持向量机（SVM），如今随着数据量的增多、计算力的增强，MLP的结构能够设计地更加复杂，因此又在深度学习的浪潮下回归，一般在具体应用中与其它神经网络结构结合使用。

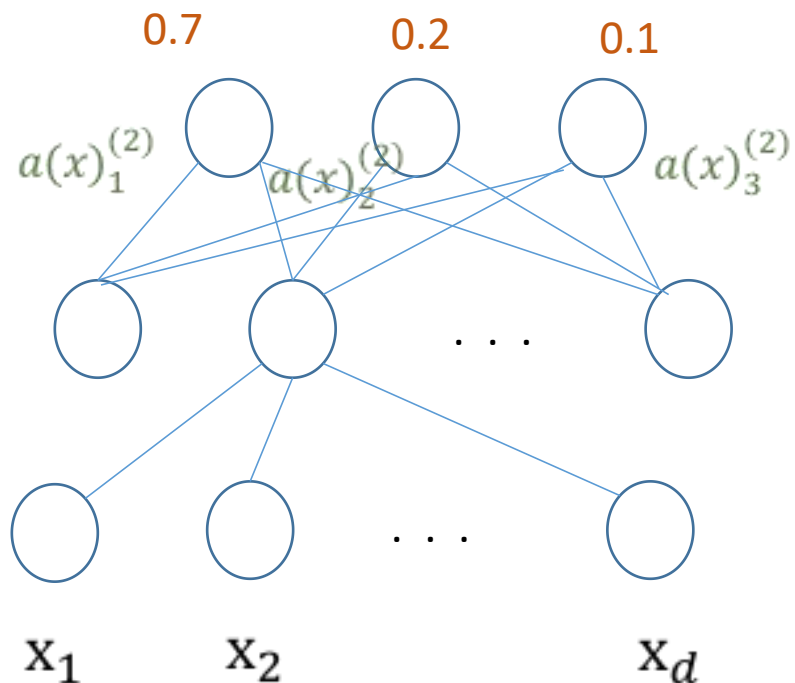
多层感知机：单输出



可用于回归问题。

多层感知机：多输出

$h(x)^{(2)}$ 分类问题



$$a(x)^{(2)} = (a(x)_1^{(2)}, a(x)_2^{(2)}, a(x)_3^{(2)})$$

$$h(x)^{(2)} = O(a(x)^{(2)})$$

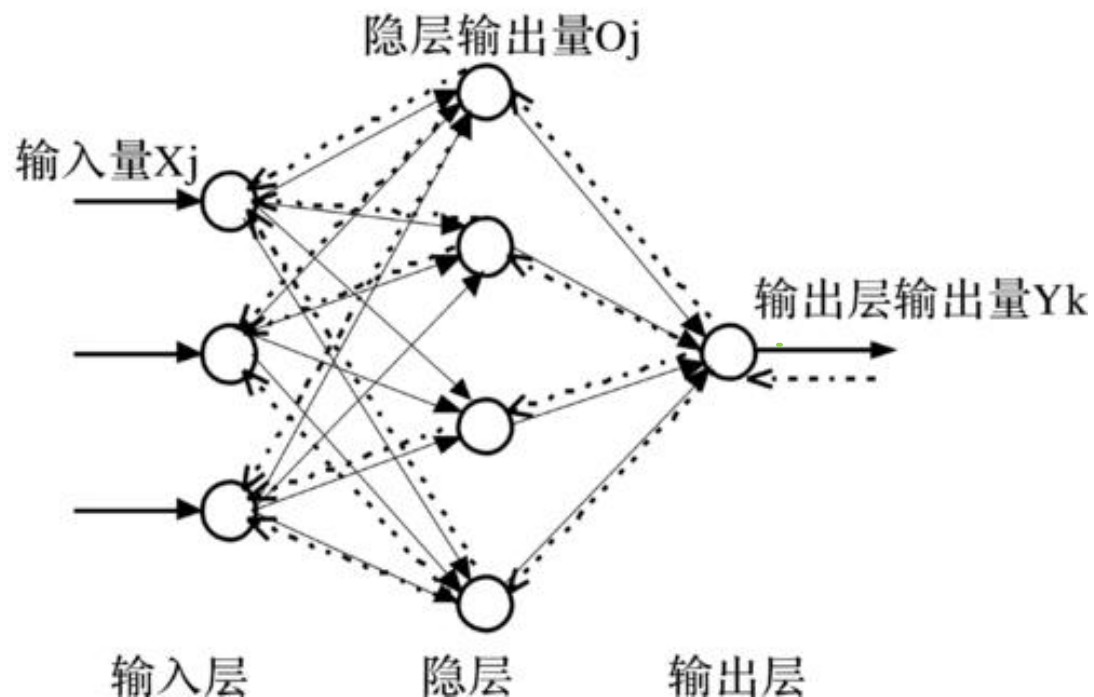
分类问题，此时的激活函数为Softmax函数

$$h(x)_1^{(2)} = \frac{\exp(a(x)_1^{(2)})}{\exp(a(x)_1^{(2)}) + \exp(a(x)_2^{(2)}) + \exp(a(x)_3^{(2)})}$$

$$h(x)_2^{(2)} = \frac{\exp(a(x)_2^{(2)})}{\exp(a(x)_1^{(2)}) + \exp(a(x)_2^{(2)}) + \exp(a(x)_3^{(2)})}$$

$$h(x)_3^{(2)} = \frac{\exp(a(x)_3^{(2)})}{\exp(a(x)_1^{(2)}) + \exp(a(x)_2^{(2)}) + \exp(a(x)_3^{(2)})}$$

误差反向传播算法



- 明确问题
 - 回归
 - 二分类
 - 多分类
- 设计网络结构→初始化参数
- 设计代价函数（预测值与准确值的距离）
- 准备大量数据
- 误差反向传播算法学习参数

回归：代价函数

$$MSE = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - f(x^{(i)}))^2$$

均方误差是指参数估计值与参数真值之差平方的期望值；MSE可以评价数据的变化程度，MSE的值越小，说明预测模型描述实验数据具有更好的精确度。

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (y^{(i)} - f(x^{(i)}))^2}$$

均方根误差是均方误差的算术平方根，能够直观观测预测值与实际值的离散程度。

$$MAE = \frac{1}{N} \sum_{i=1}^N |y^{(i)} - f(x^{(i)})|$$

平均绝对误差是绝对误差的平均值，平均绝对误差能更好地反映预测值误差的实际情况。

分类：代价函数

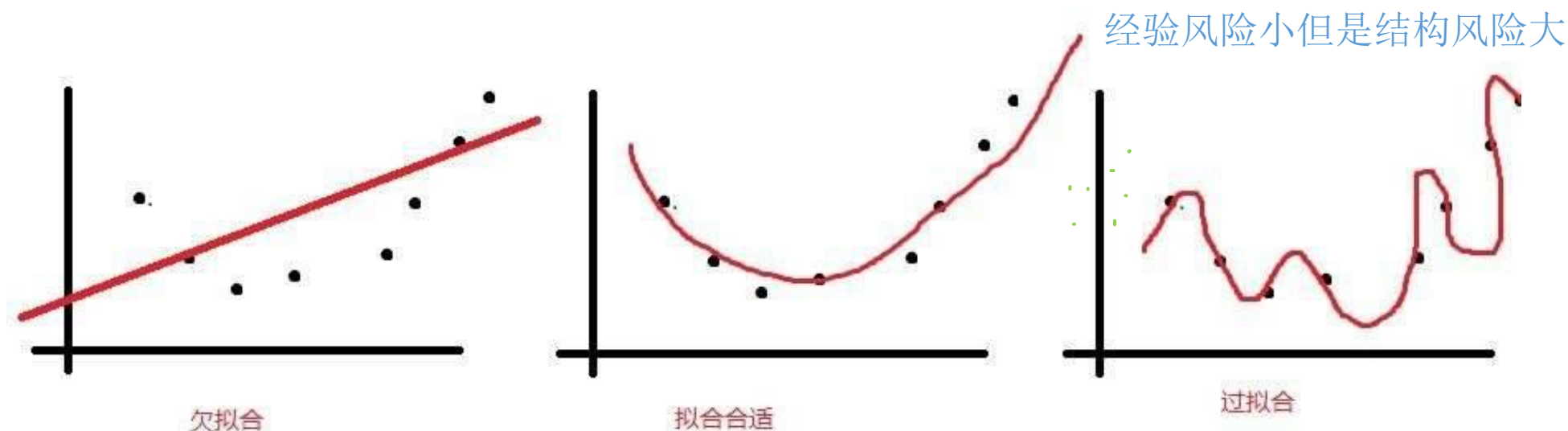
$$H(p, q) = - \sum_{i=1}^N p(x^{(i)}) \log q(x^{(i)})$$

交叉熵是用来评估当前训练得到的概率分布与真实分布的差异情况，减少交叉熵损失就是在提高模型的预测准确率。其中 $p(x)$ 是指真实分布的概率， $q(x)$ 是模型通过数据计算出来的概率估计。

$$L(w, b) = -\frac{1}{N} \sum_{i=1}^N (y^{(i)} \log f(x^{(i)}) + (1 - y^{(i)}) \log (1 - f(x^{(i)})))$$

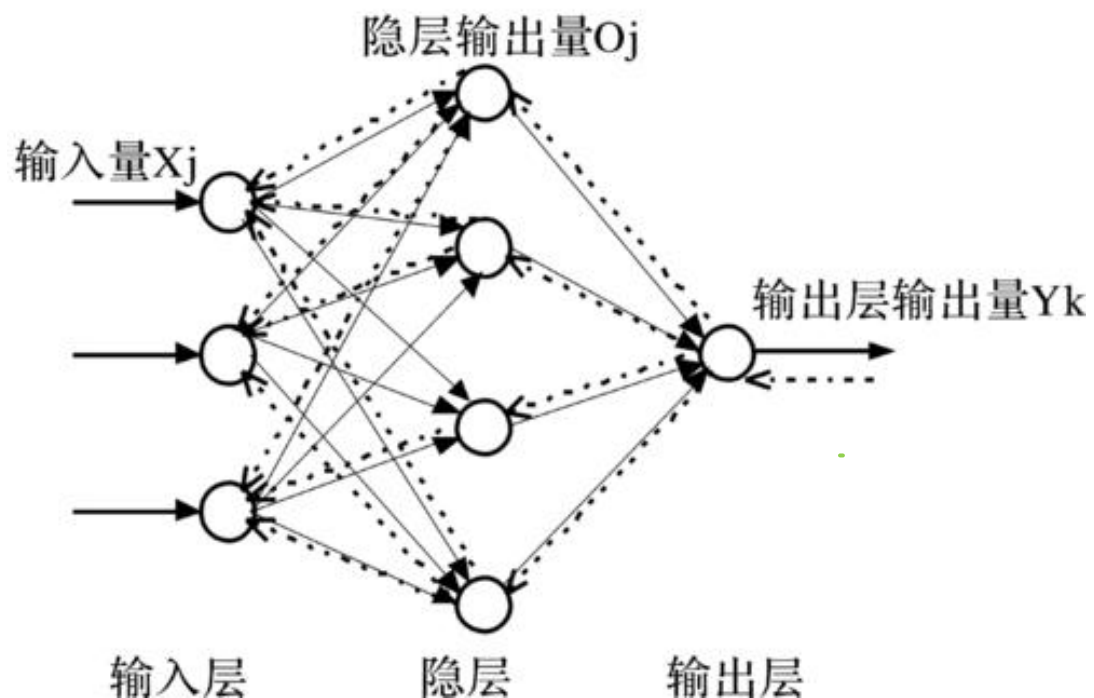
对于二分类模型的交叉熵代价函数。
其中 $f(x)$ 是sigmoid函数， $y(i) \in 0, 1$ 。

代价函数名词解析



- 损失函数 (Loss Function)：是定义在单个样本上的，是指一个样本的误差。
- 代价函数 (Cost Function)：是定义在整个训练集上的，是所有样本误差的平均，也就是所有损失函数值的平均。
- 一般情况下，不区分损失函数 (lost function) 和代价函数 (cost function)
- 损失函数 (lost function)，代价函数 (cost function) 针对的是经验风险最小化
- 目标函数是一个相关但更广的概念，针对的是经验风险以及结构风险最小化
- 可以简单地理解为：目标函数就是加了正则项的损失函数或者代价函数
- 一般情况下我们不会特意地去区分三者

误差反向传播算法（BP算法）



BP算法包含两个过程：

- 初始化参数后将训练数据代入，逐步计算每一层的结果（前向）；
- 以梯度下降法为基础，用反向传播的方式对参数进行更新。反复这两个过程直到模型的拟合能力达到一定程度或者达到最大迭代次数为止（后向）。

前向过程的表达式如下：

$$x_n \rightarrow network(\theta) \rightarrow y_n \overset{c_n}{\leftrightarrow} y'_n$$

误差反向传播算法

目标：
优化参数，最小化代价函数

为什么要BP

由于神经网络独特的结构，所以对它直接算梯度下降有一丢丢复杂，需要引入一个技术：反向传播算法（Backpropagation）

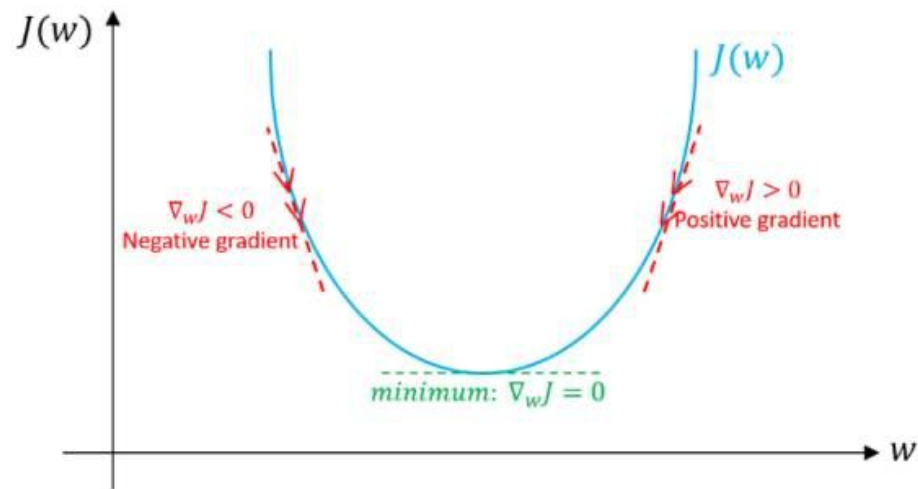
反向传播算法要点：

- 梯度下降法
- 链式求导法则
- 为什么反向

误差反向传播算法

梯度下降法

- 梯度到底是什么？
 - 梯度下降有什么用？
 - 如何下降？
- 其实“梯度”就是我们常说的“导数”，即代价函数的导数。
 - 梯度下降就是找让误差值（代价函数）最小的时候这个算法所对应的参数。
 - 按梯度的反方向变化参数。



$$W \leftarrow W - \alpha \nabla J$$

误差反向传播算法

Chain Rule

链式求导法则

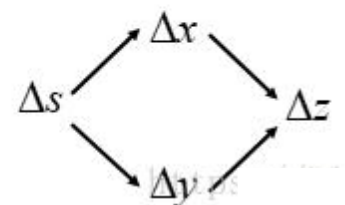
- 链式法则是微积分中的求导法则，用以求一个复合函数的导数。
- 所谓的复合函数，是指多个函数嵌套的情况。
- 神经网络可以看作是一个极其复杂的，包含很多参数的复合函数。

Case 1 $y = g(x) \quad z = h(y)$

$$\Delta x \rightarrow \Delta y \rightarrow \Delta z \quad \frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Case 2

$$x = g(s) \quad y = h(s) \quad z = k(x, y)$$


$$\frac{dz}{ds} = \frac{\partial z}{\partial x} \frac{dx}{ds} + \frac{\partial z}{\partial y} \frac{dy}{ds}$$

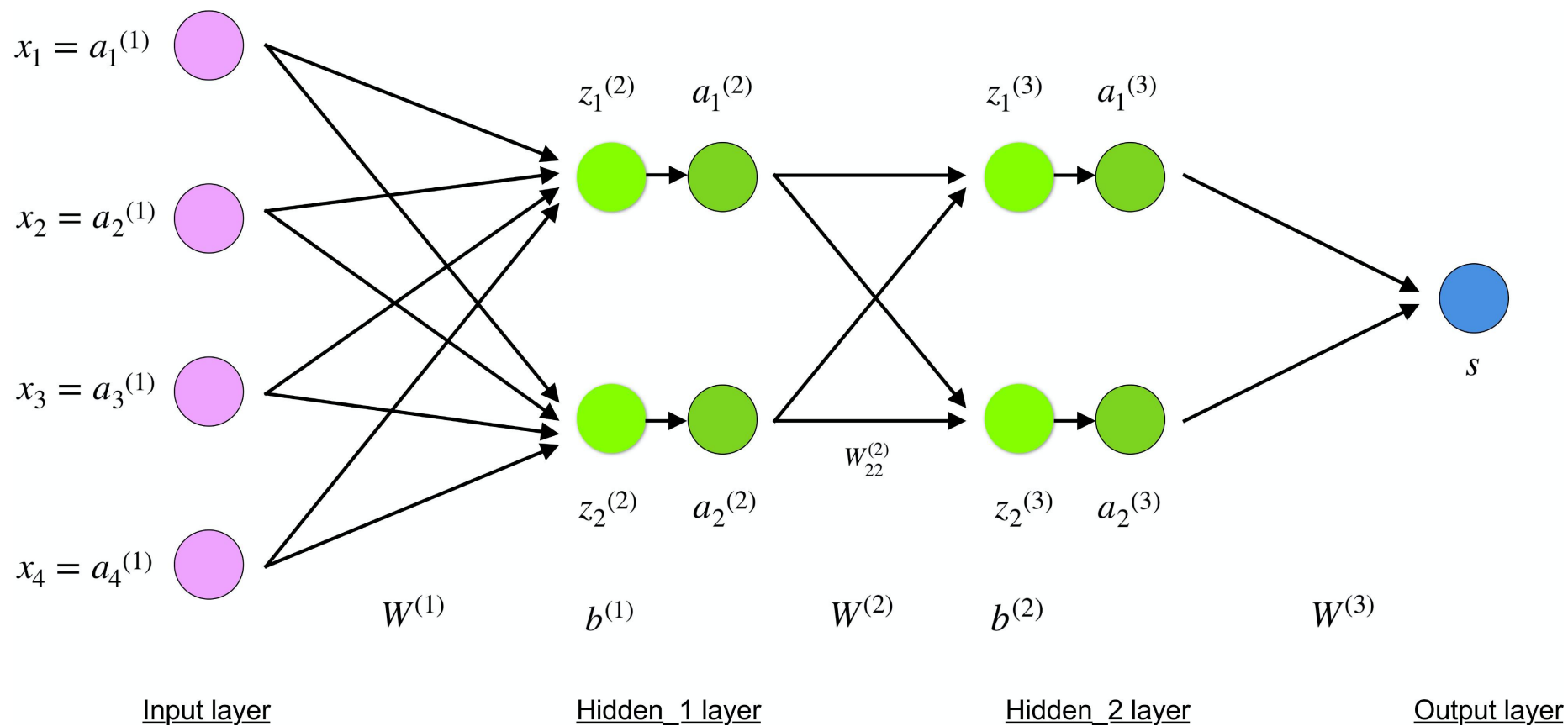
误差反向传播算法

为什么反向

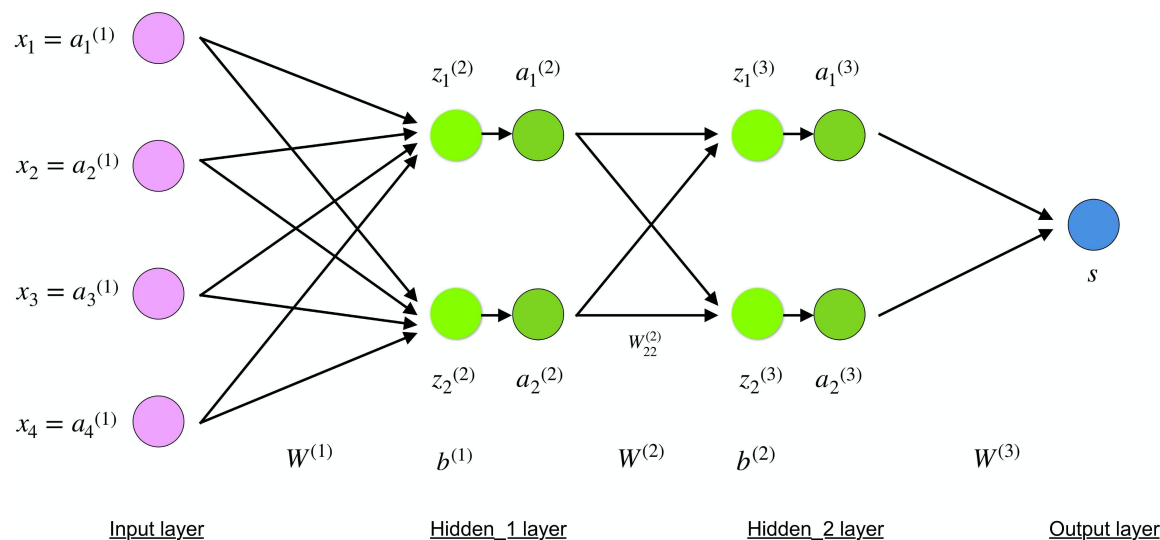
- 应用链式求导法则以及梯度下降可以更新参数了，那么为什么要说是反向呢？
- 更新顺序从后往前（why?）

误差反向传播算法：计算实例

定义一个4层神经网络，由输入层的4个神经元，隐藏层的4个神经元和输出层的1个神经元组成。



误差反向传播算法： 计算实例



使用激活函数 f 计算激活 a^2 和 a^3 。通常，此函数 f 是非线性的（例如S型，ReLU，tanh），并允许网络学习数据中的复杂模式。

注：表达式中所有 x ， z^2 ， a^2 ， z^3 ， a^3 ， W^1 ， W^2 ， b^1 和 b^2 都缺少上面4层网络图中显示的下标。原因是我们将所有参数值组合在参数矩阵中，并按图层分组。

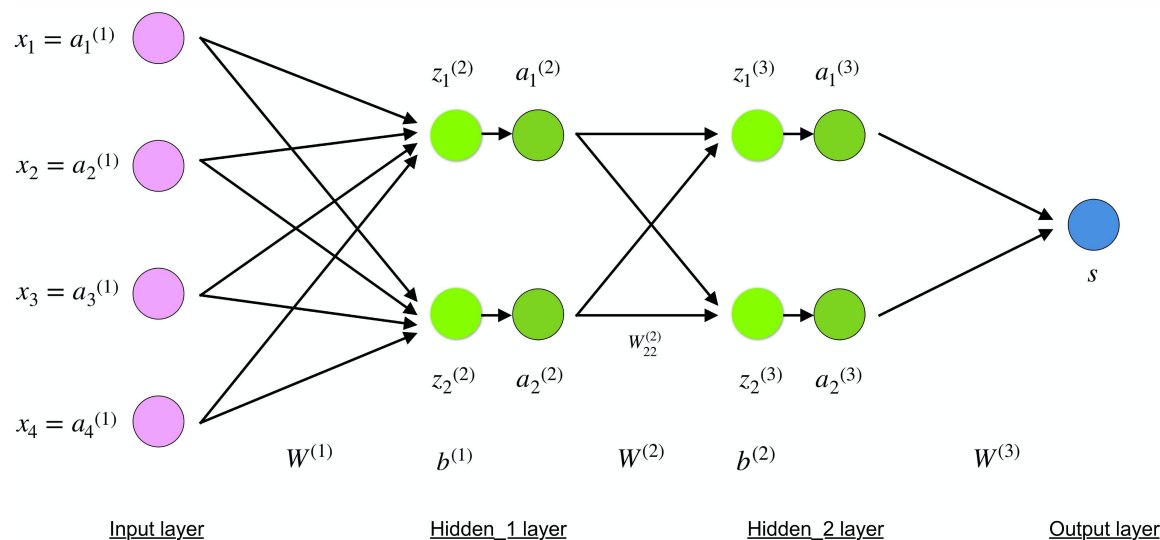
$$z^{(2)} = W^{(1)}x + b^{(1)}$$

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

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

$$a^{(3)} = f(z^{(3)})$$

误差反向传播算法： 计算实例



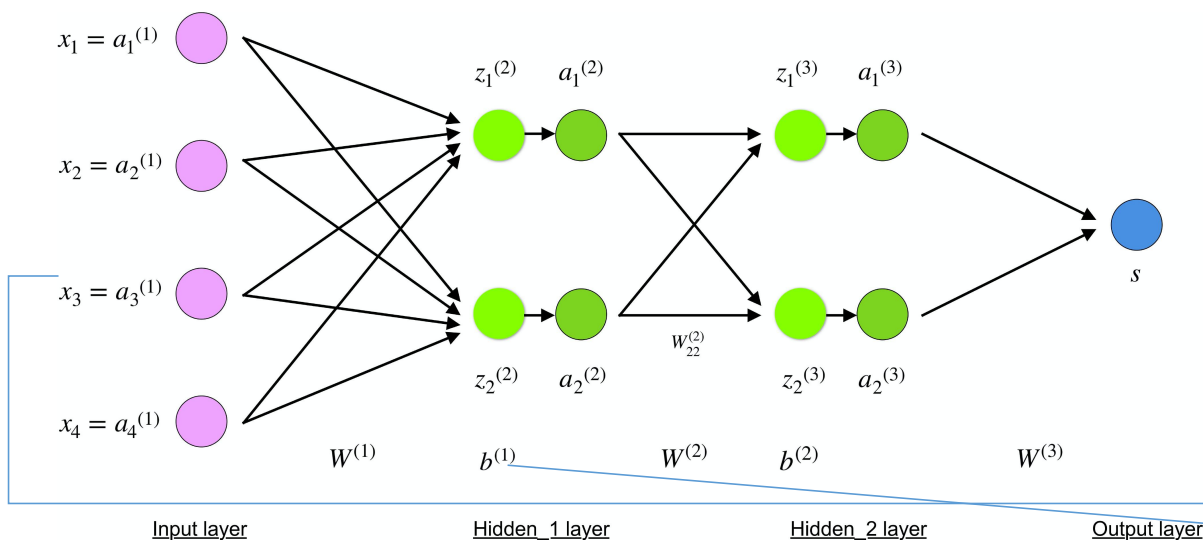
$W^{(1)}$ 是形状为 (n, m) 的权重矩阵，其中 n 是输出神经元的数量（在下一层中的神经元）， m 是输入神经元的数量（在上一层中的神经元）。此处， $n = 2$ ， $m = 4$ 。

注意：任何权重的下标中的第一个数字与下一层（在我们的情况下为Hidden_2层）的神经元的索引匹配，第二个数字与上一层的神经元的索引（在我们的情况下为Input的匹配）层）。

$$\begin{aligned} z^{(2)} &= W^{(1)}x + b^{(1)} \\ a^{(2)} &= f(z^{(2)}) \end{aligned} \quad \begin{aligned} z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\ a^{(3)} &= f(z^{(3)}) \end{aligned}$$

$$W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \end{bmatrix}$$

误差反向传播算法：计算实例



$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

x 是形状 $(m, 1)$ 的输入向量，其中 m 是输入神经元的数量。对我们来说， $m = 4$ 。

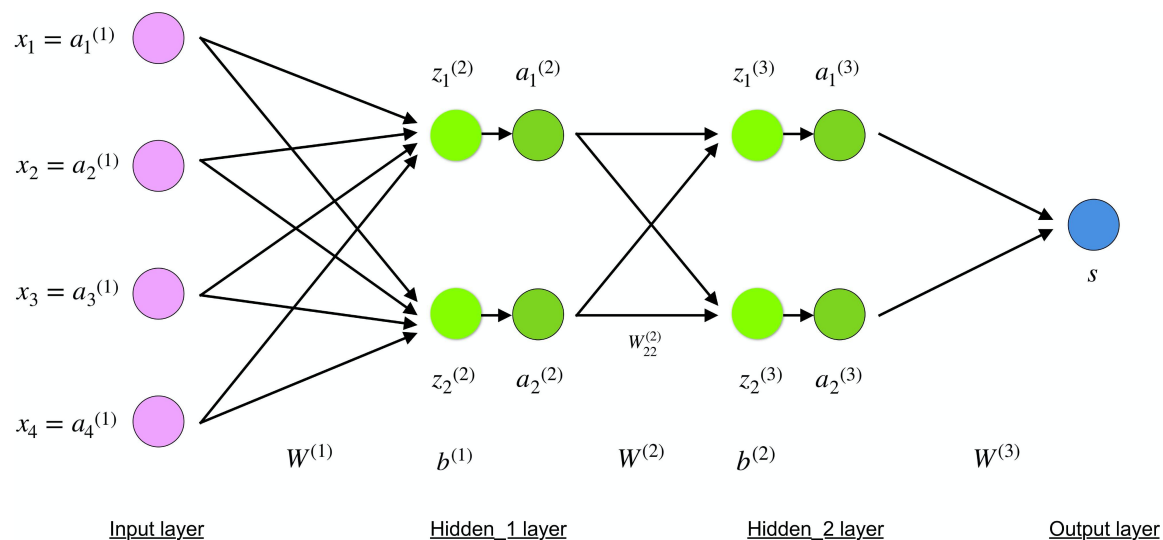
$$b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

$b^{(1)}$ 是形状 $(n, 1)$ 偏移向量，其中 n 是当前层中神经元的数量。这里， $n = 2$ 。

$$\begin{aligned} z^{(2)} &= W^{(1)}x + b^{(1)} & z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\ a^{(2)} &= f(z^{(2)}) & a^{(3)} &= f(z^{(3)}) \end{aligned}$$

$$W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \end{bmatrix}$$

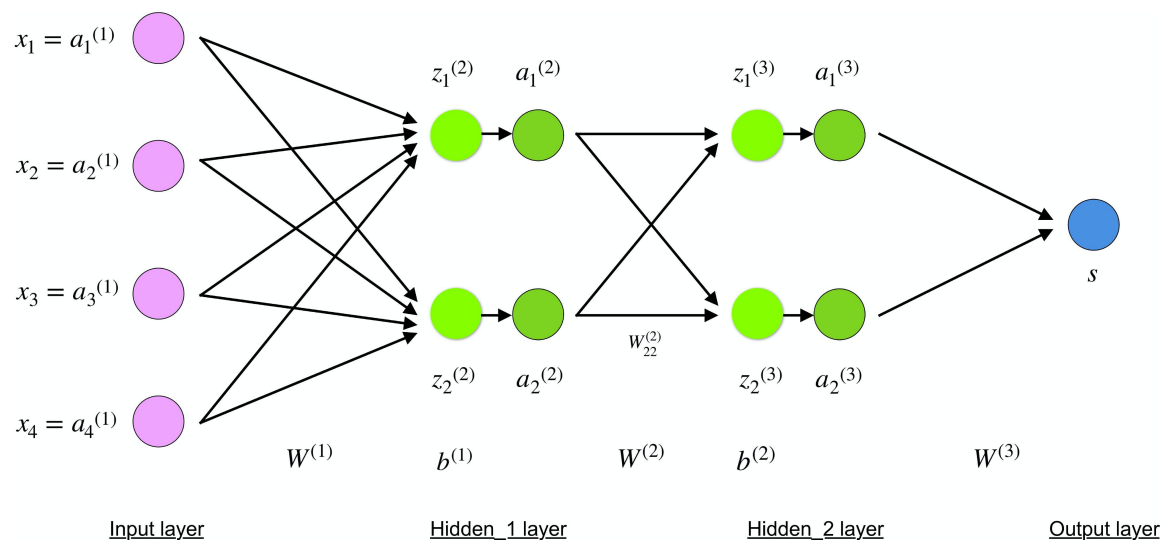
误差反向传播算法： 计算实例



$$z^{(2)} = \begin{bmatrix} W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + W_{14}^{(1)}x_4 \\ W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + W_{24}^{(1)}x_4 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

按照 z^2 的方程，我们可以使用 W^1 ， x 和 b^1 的上述定义来得出“ z^2 的方程”

误差反向传播算法： 计算实例



$x = a^{(1)}$ *Input layer*

$z^{(2)} = W^{(1)}x + b^{(1)}$ *neuron value at Hidden₁ layer*

$a^{(2)} = f(z^{(2)})$ *activation value at Hidden₁ layer*

$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$ *neuron value at Hidden₂ layer*

$a^{(3)} = f(z^{(3)})$ *activation value at Hidden₂ layer*

$s = W^{(3)}a^{(3)}$ *Output layer*

- 前向传递的最后一步是计算真实输出 y 与预测输出 s 的差别。
- 输出 y 是训练数据集 (x, y) 的一部分，其中 x 是输入。
- s 和 y 之间的评估是通过成本函数进行的，比如MSE（均方误差）。
- 我们将此成本函数命名为 C 并表示为： $C = \text{cost}(s, y)$
- 基于 C 的值，模型“知道”调整其参数的数量，以便更接近预期的输出 y 。

误差反向传播算法： 计算实例

对于某个 layer 中的单个权重 w ，梯度为：

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} \quad \text{chain rule}$$

$$z_j^l = \sum_{k=1}^m w_{jk}^l a_k^{l-1} + b_j^l \quad \text{by definition}$$

m - number of neurons in $l-1$ layer

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \quad \text{by differentiation (calculating derivative)}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} a_k^{l-1} \quad \text{final value}$$

难以计算

前向计算中已知

误差反向传播算法： 计算实例

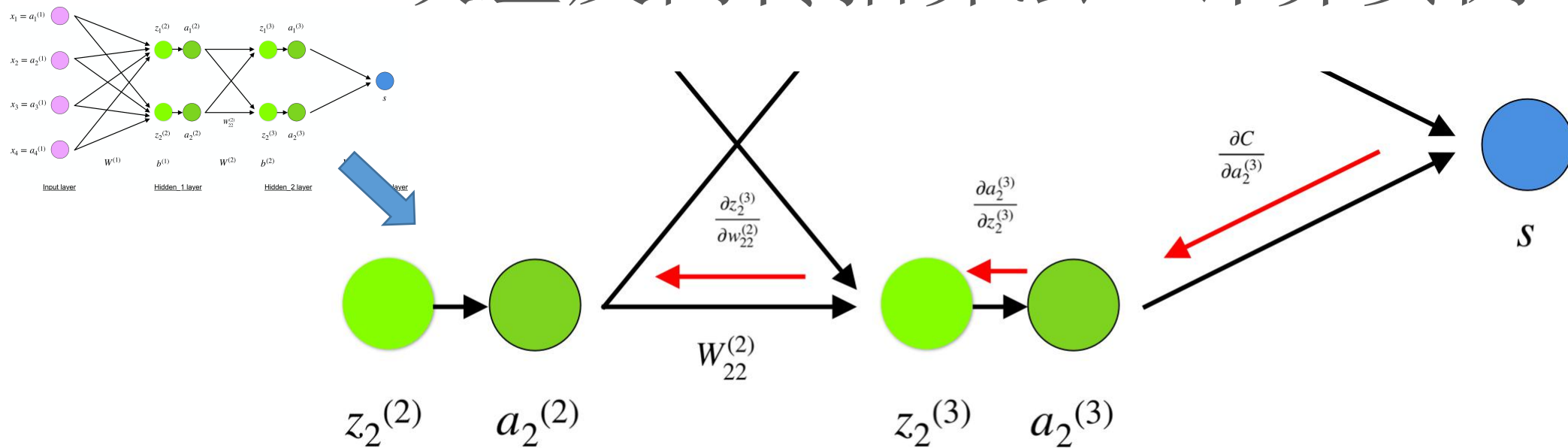
可以将类似的等式集应用于b:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} \quad \text{chain rule}$$

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \quad \text{by differentiation (calculating derivative)}$$

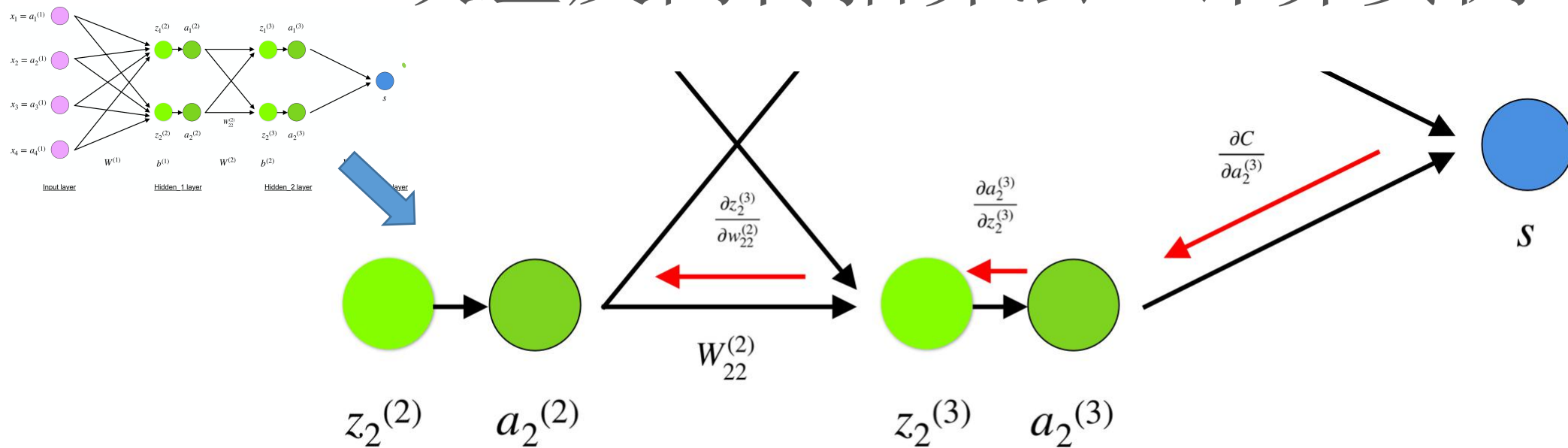
$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} 1 \quad \text{final value}$$

误差反向传播算法：计算实例



在该示例中，我们将计算C相对于单个权重 $(w_{22})^2$ 的梯度。

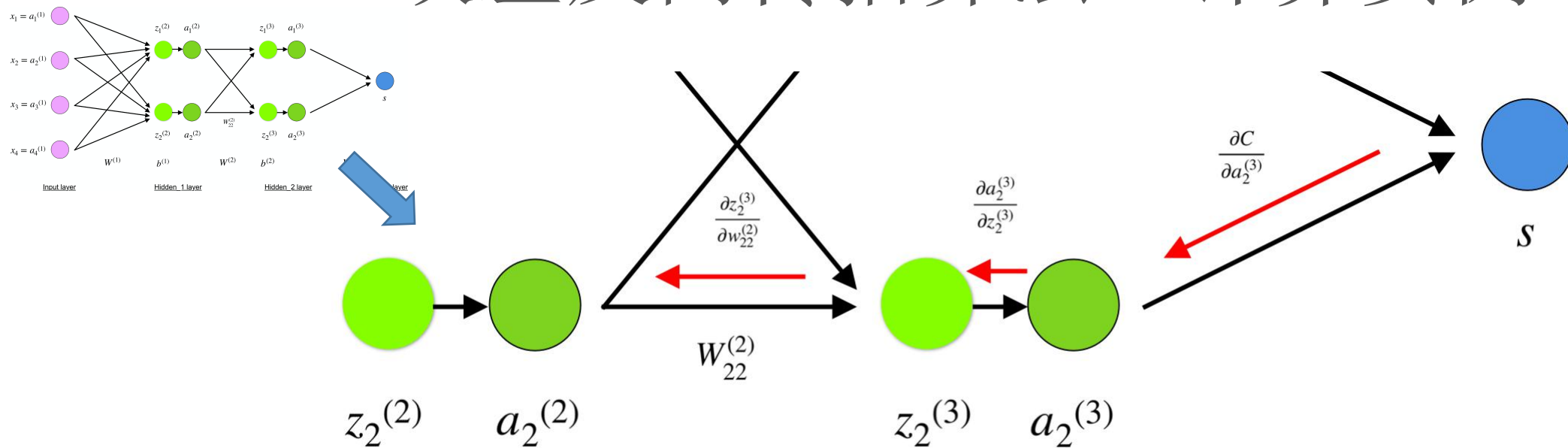
误差反向传播算法：计算实例



$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} \cdot f'(z_2^{(3)}) \cdot a_2^{(2)}$$

前向过程中已知

误差反向传播算法：计算实例



$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} \cdot f'(z_2^{(3)}) \cdot a_2^{(2)}$$

- 越是计算前面的参数梯度，越是复杂。
- 越是计算后面的参数梯度，越是简单。
- 前面的参数梯度的计算依赖于后面参数梯度的计算。
- 从后往前开始算梯度，并且保存计算结果（类似DP）。

误差反向传播算法： 计算实例

梯度下降优化模型参数：

while (termination condition not met)

$$w := w - \epsilon \frac{\partial C}{\partial w}$$

$$b := b - \epsilon \frac{\partial C}{\partial b}$$

end

w和b的初始值是随机选择的。

Epsilon (e) 是学习率。 它确定了梯度下降的速度。

w和b是权重和偏差的矩阵表示。 可以使用各个权重或偏差中C的偏导数来计算w或b中C的导数。

一旦代价函数最小化，则满足终止条件。



参考资料

Complete Guide of Activation Functions

<https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044>

为什么神经网络能以任意精度拟合任意复杂度的函数？

https://www.jianshu.com/p/9ed784e7557b?utm_campaign=maleskine&utm_content=note&utm_medium=reader_share&utm_source=weixin

Understanding-backpropagation-algorithm

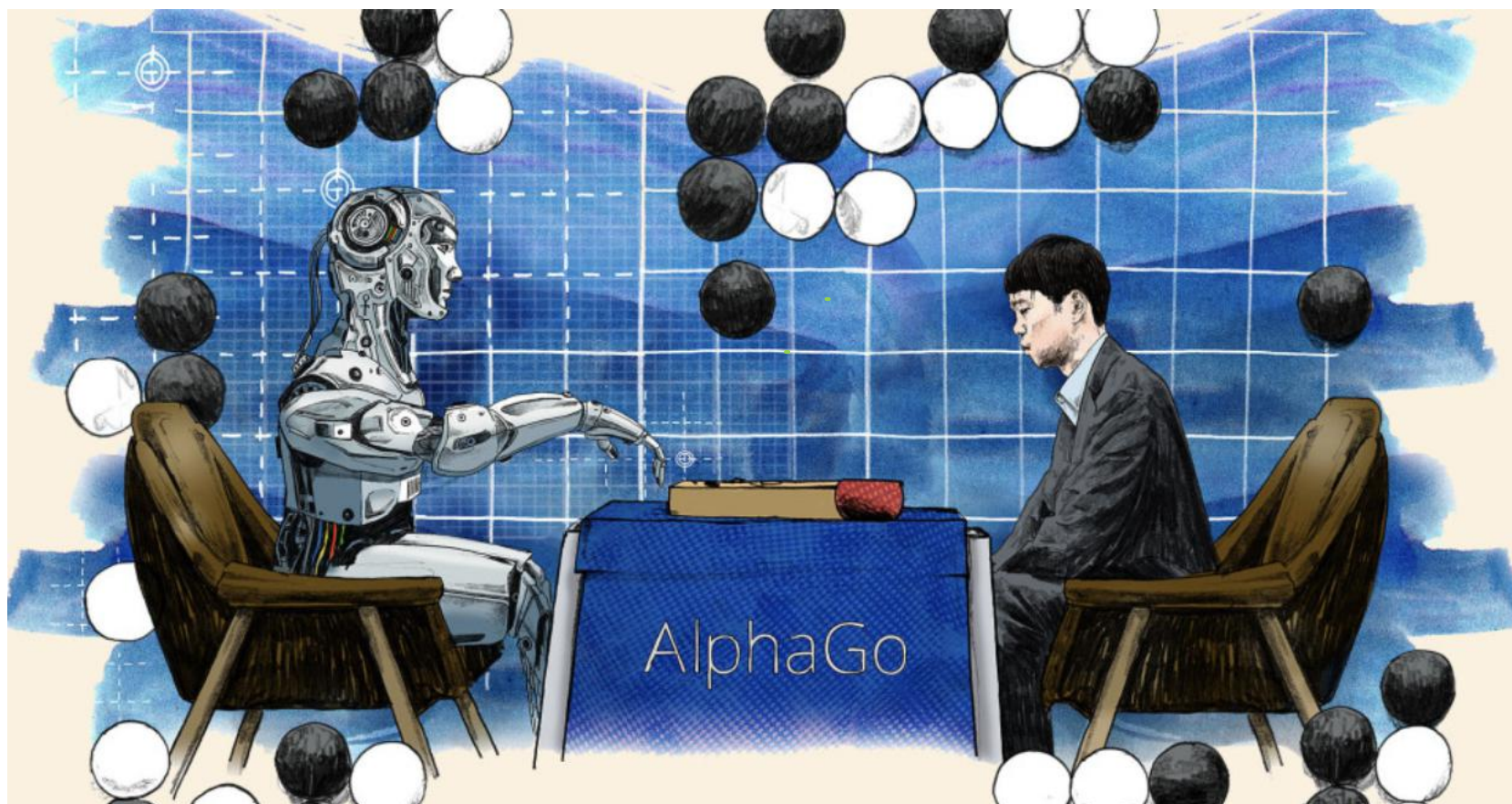
<https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd>

relu介绍

<https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/>



休息





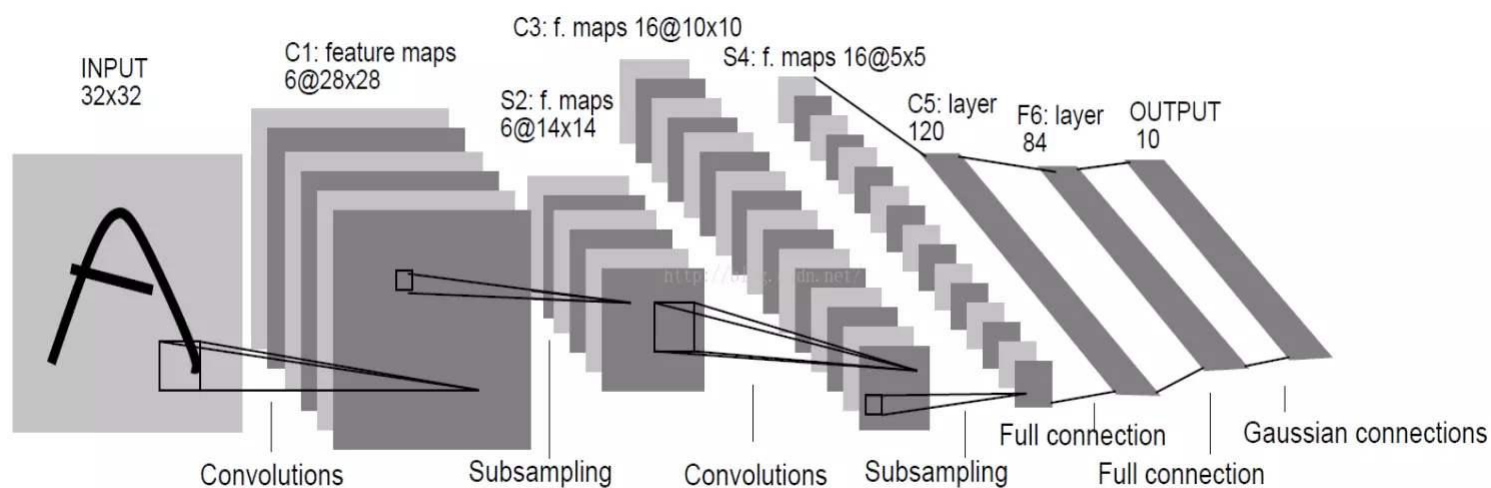
CNN





大纲

- 为什么CNN 常用于图像处理?
- CNN简意图
- CNN - 卷积层
- 卷积层与全连接层的比较
- CNN - 池化层
- 摊平 (Flatten)
- 开山之作: LeNet5
- 一战成名: AlexNet
- CNN进化史
- 应用

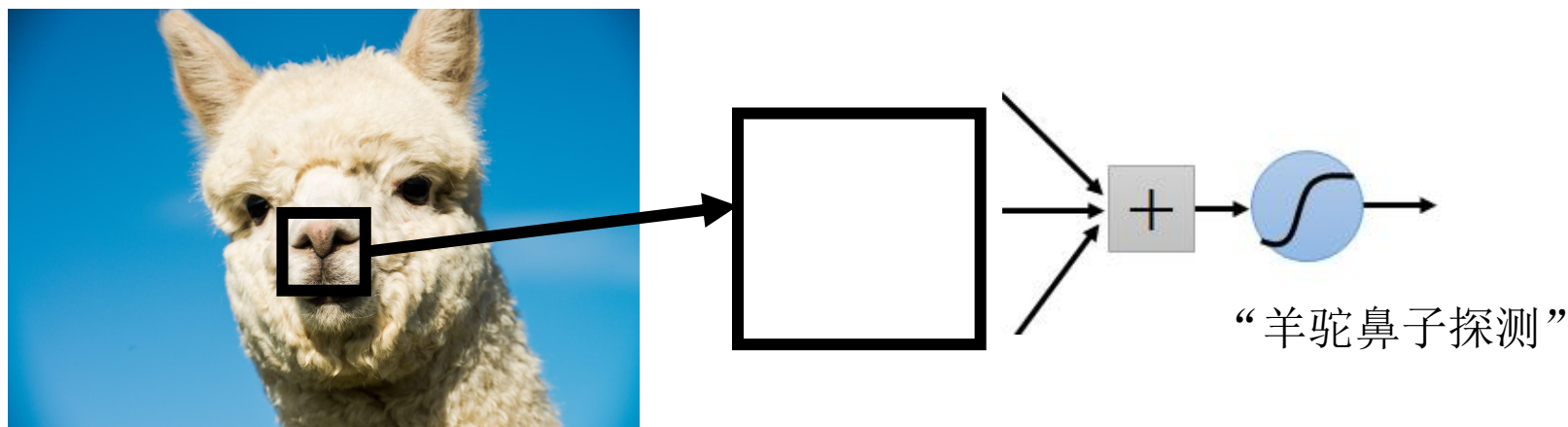




为什么CNN 常用于图像处理？

- 某些重要的局部特征远小于整张图片大小。
- 通过识别某重要的几个局部特征，便能有效地进行图片识别。
- 局部特征的解析只需要少量的参数。

*局部性

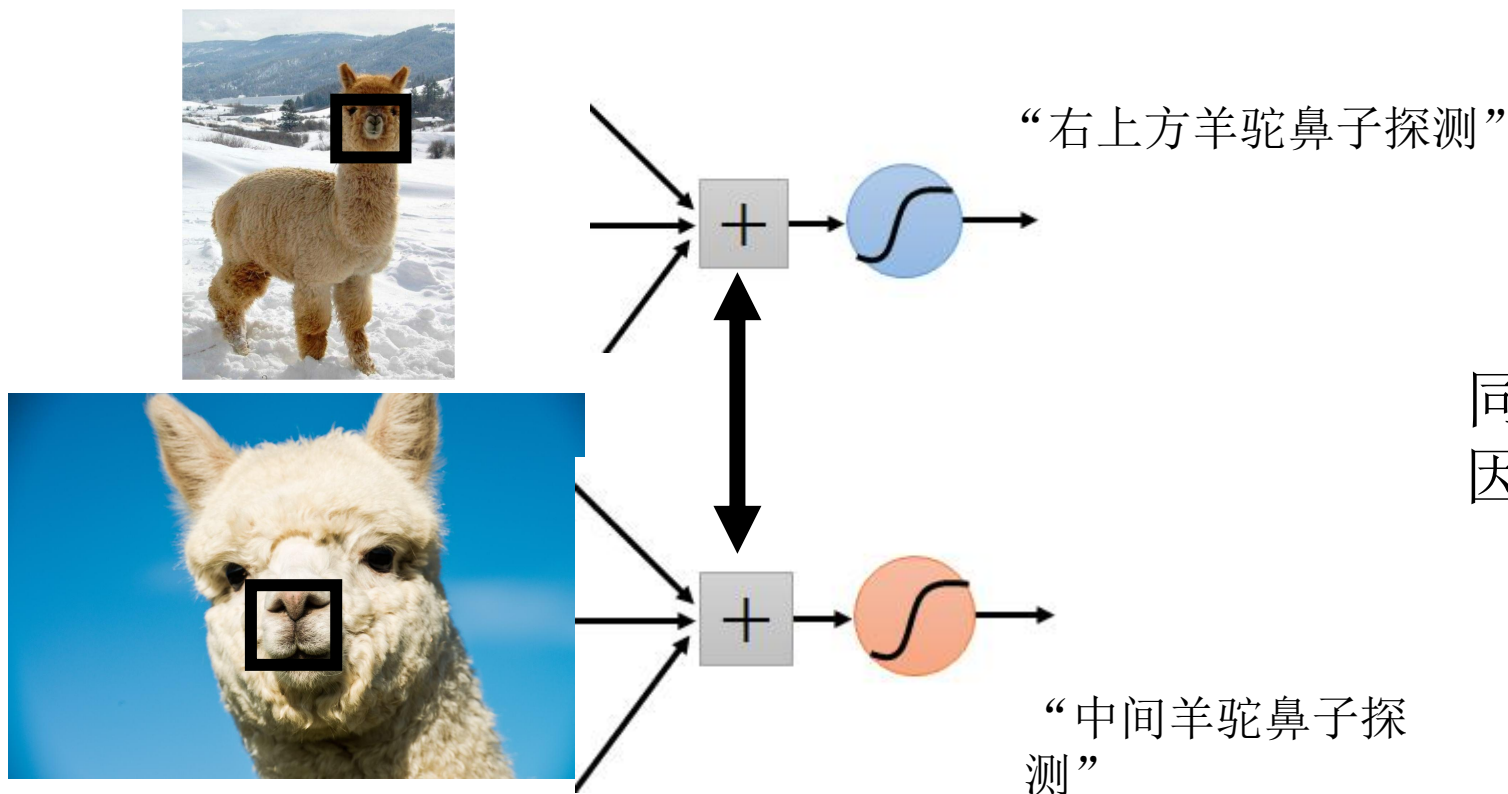




为什么CNN 常用于图像处理？

- 同样的特征可能出现在图片的不同位置。

*平移性



同为羊驼鼻子探测，
因此可以共用参数



为什么CNN 常用于图像处理？

- 适当降低分辨率不会影响识别效果。

*可缩性

羊驼



降低分辨率

羊驼



- 通过适当降低分辨率减小图片。
- 减小模型参数规模。



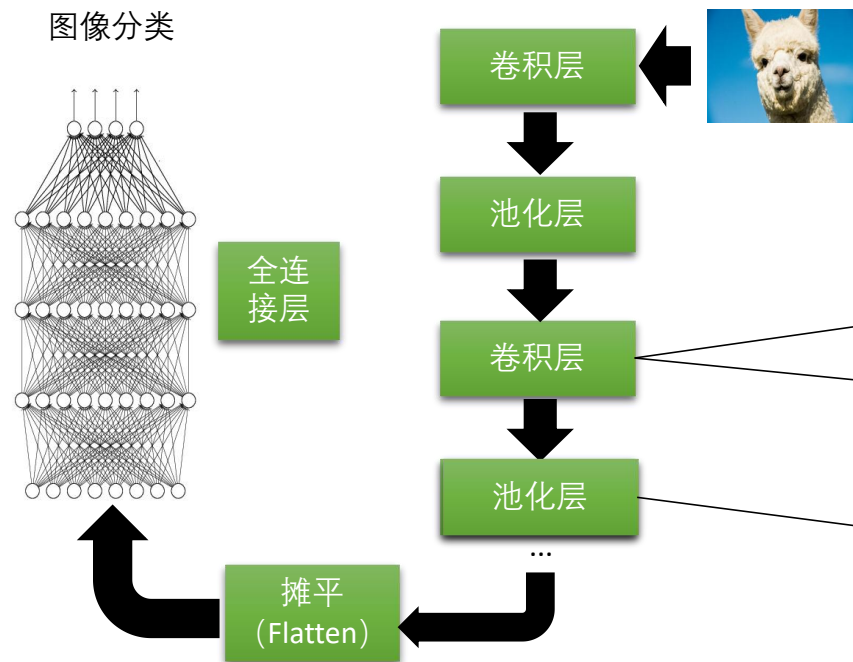
为什么CNN 常用于图像处理？

- *局部性
- *平移性
- *可缩性





CNN简意图



- 某些重要的局部特征远小于整张图片大小（**局部性**）。
- 同样的特征可能出现在图片的不同位置（**平移性**）。
- 适当降低分辨率不会影响识别效果（**可缩性**）。



CNN - 卷积层

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

输入图像

1	-1	-1
-1	1	-1
-1	-1	1

卷积核1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

卷积核2

Matrix

...

某些重要的局部特征远小于整张图片大小，每个卷积核包含的参数很少 (3×3)。



CNN - 卷积层

stride (步长) = 1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -1

卷积核 1

1	-1	-1
-1	1	-1
-1	-1	1



CNN - 卷积层

stride (步长) = 2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

卷积核 1

1	-1	-1
-1	1	-1
-1	-1	1



CNN - 卷积层

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

卷积核1

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

同样的特征可能出现在图片的不同位置。



CNN - 卷积层

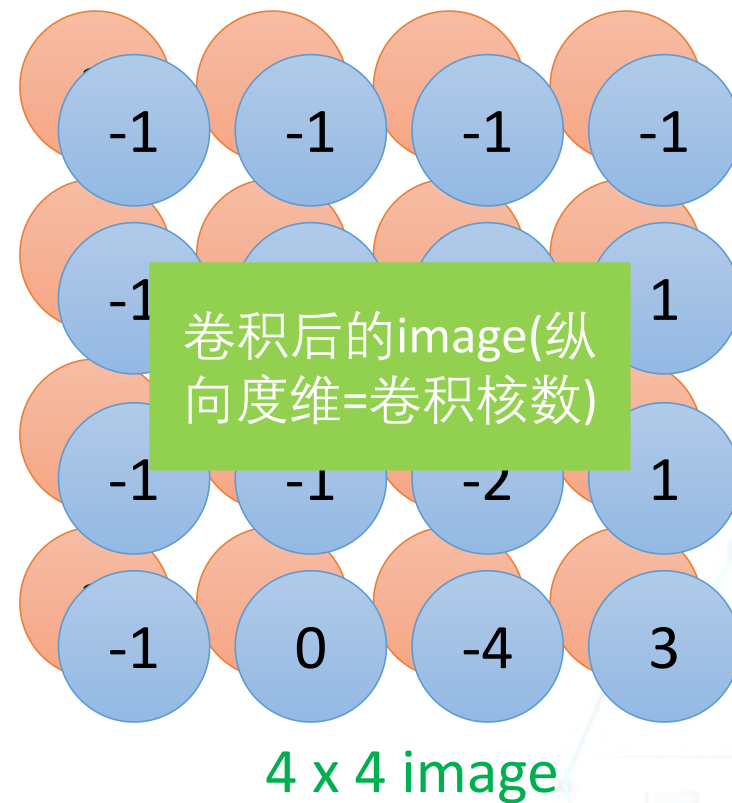
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

-1	1	-1
-1	1	-1
-1	1	-1

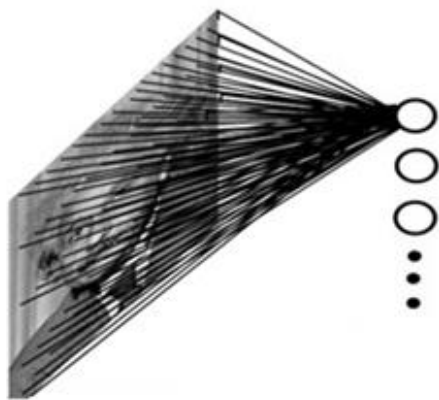
卷积核 2



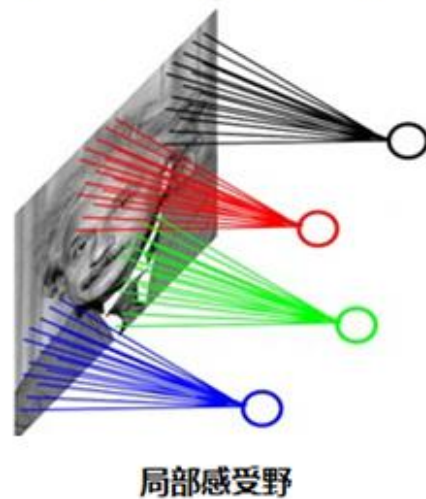


卷积层与全连接层的比较

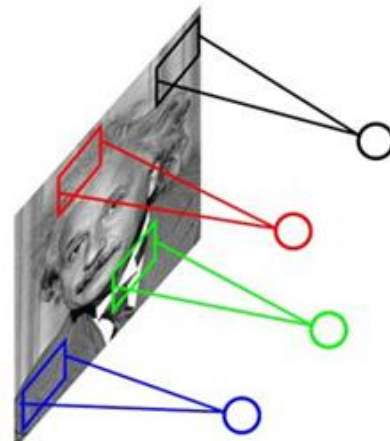
全连接模式（经典神经网络）



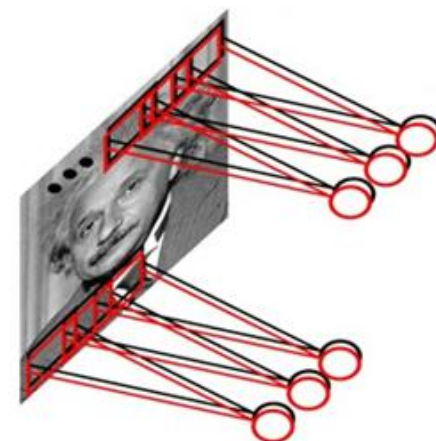
局部连接模式（卷积神经网络）



参数（权值）独立

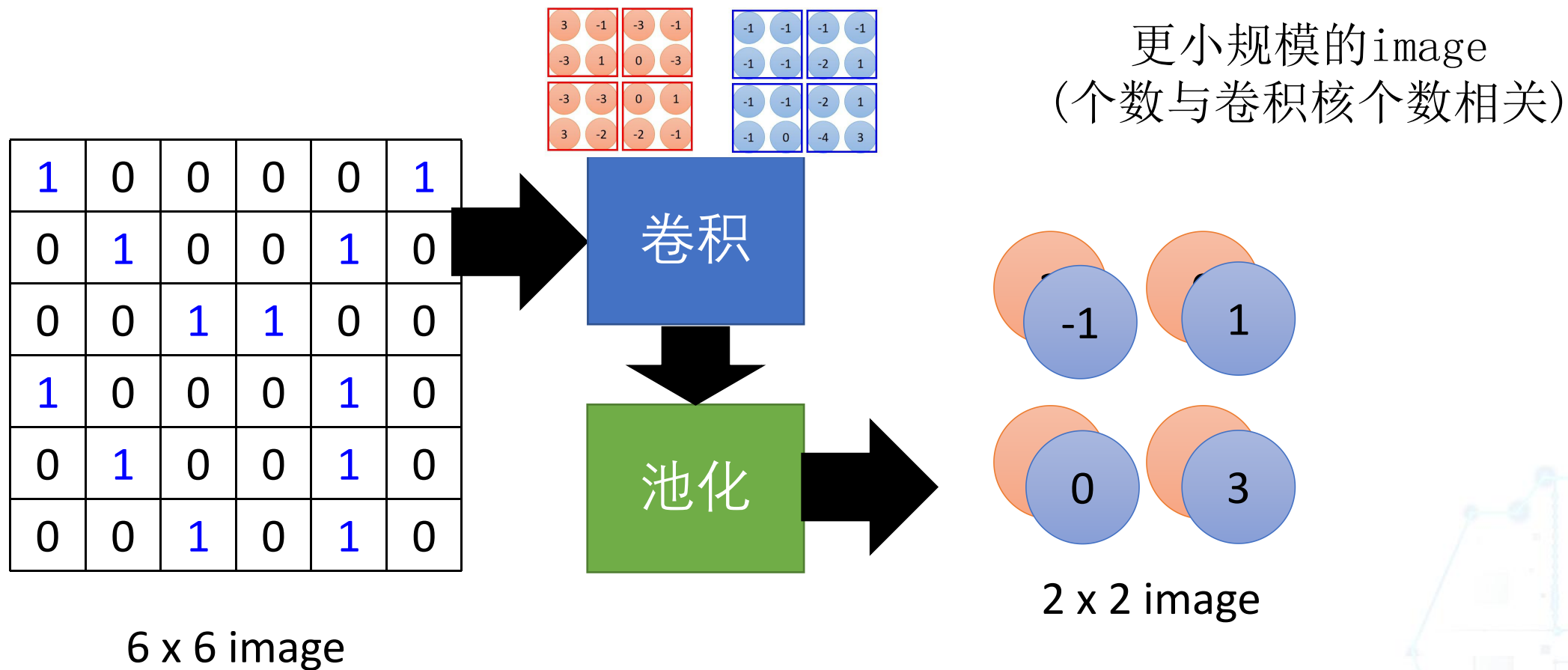


参数（权值）共享





CNN - 池化层





CNN - 池化层

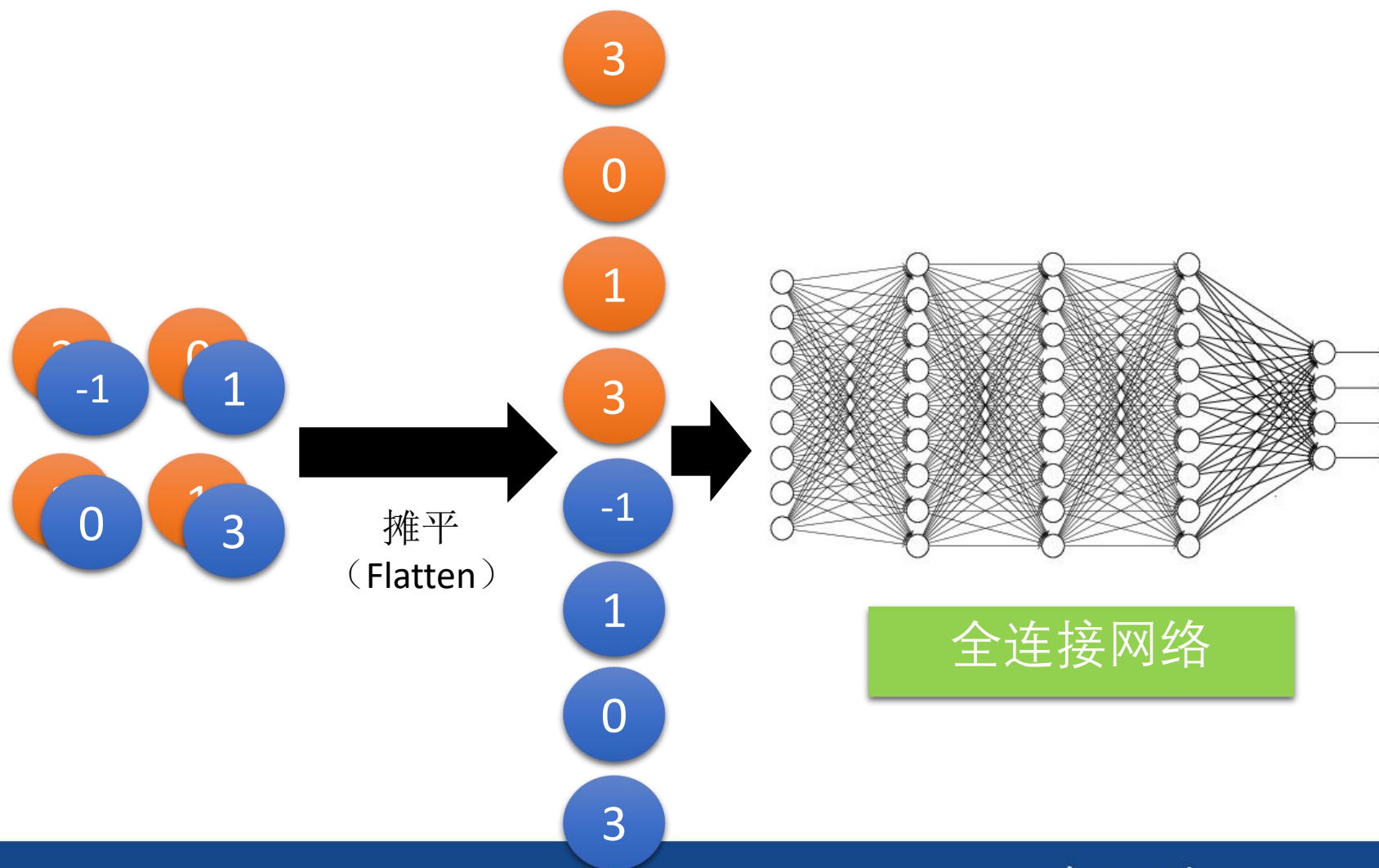
Max-pooling: 最为常见，最大池化是取整个区域的最大值作为特征，在自然语言处理中常用于分类问题，希望观察到的特征是强特征，以便可以区分出是哪一个类别。

Average-pooling: 通常是用于主题模型，常常是一个句子不止一个主题标签，如果是使用Max-pooling的话信息过少，所以使用Average的话可以广泛反映这个区域的特征。

K-max pooling: 选取一个区域的前k个大的特征。

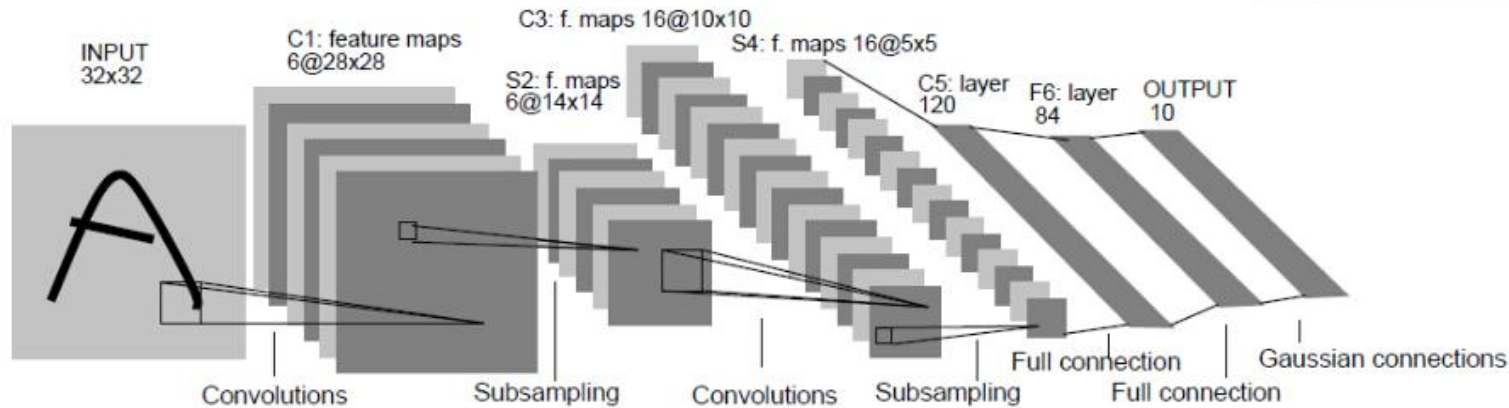


摊平 (Flatten)





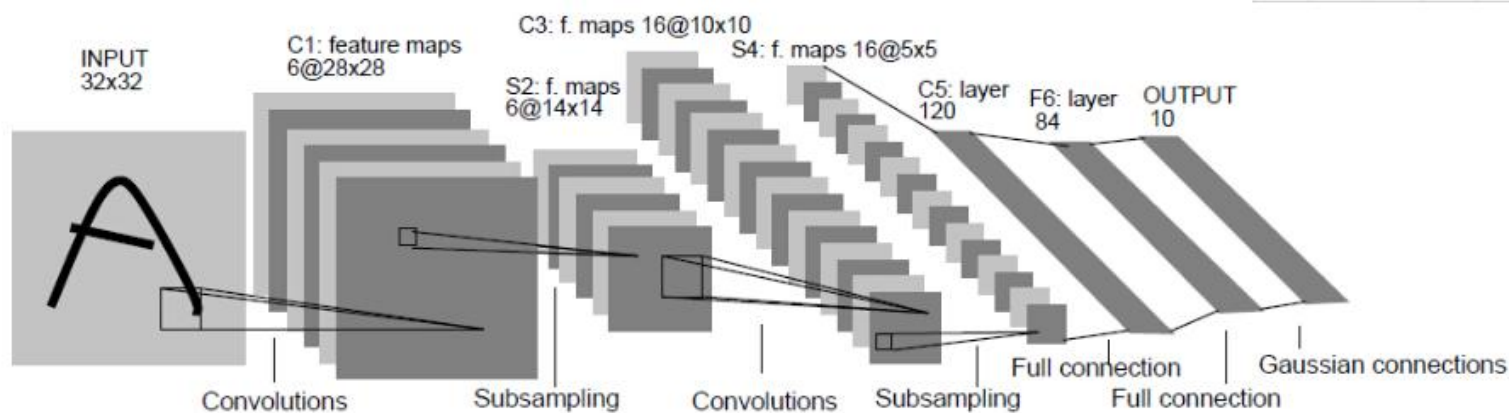
开山之作：LeNet5



- LeNet5诞生于1994年，是最早的卷积神经网络之一，由Yann LeCun完成，推动了深度学习领域的发展。
- 在那时候，没有GPU帮助训练模型，甚至CPU的速度也很慢，LeNet5通过巧妙的设计，利用卷积、参数共享、池化等操作提取特征，避免了大量的计算成本，最后再使用全连接神经网络进行分类识别。



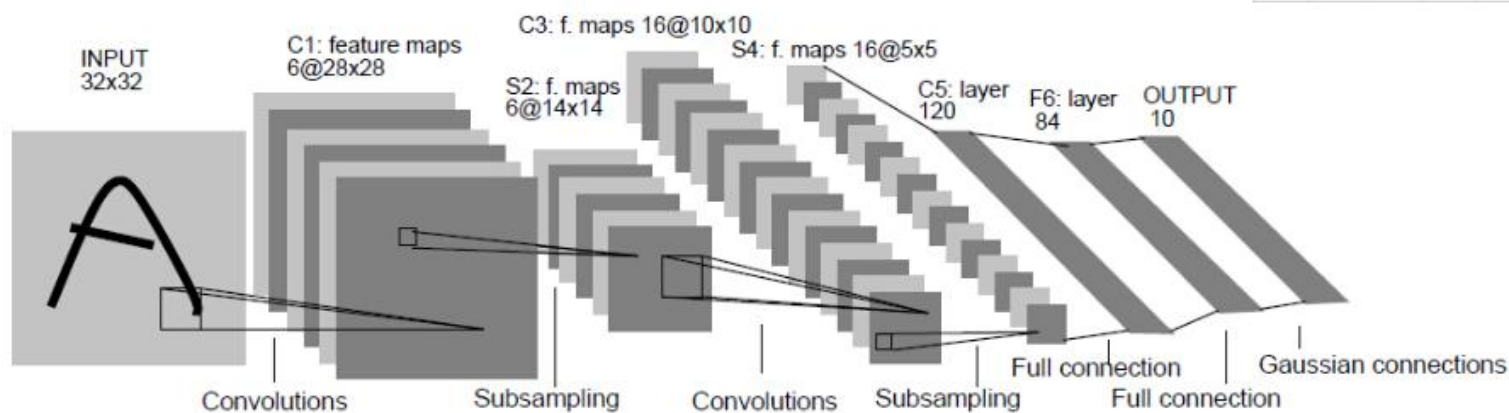
开山之作：LeNet5



- LeNet5由7层CNN（不包含输入层）组成，上图中输入的原始图像大小是 32×32 像素，卷积层用 C_i 表示，子采样层（pooling，池化）用 S_i 表示，全连接层用 F_i 表示。



开山之作：LeNet5



C1层（卷积层）：6@28×28

- 该层使用了6个卷积核，每个卷积核的大小为5×5，这样就得到了6个feature map（特征图）。
- feature map大小的计算？

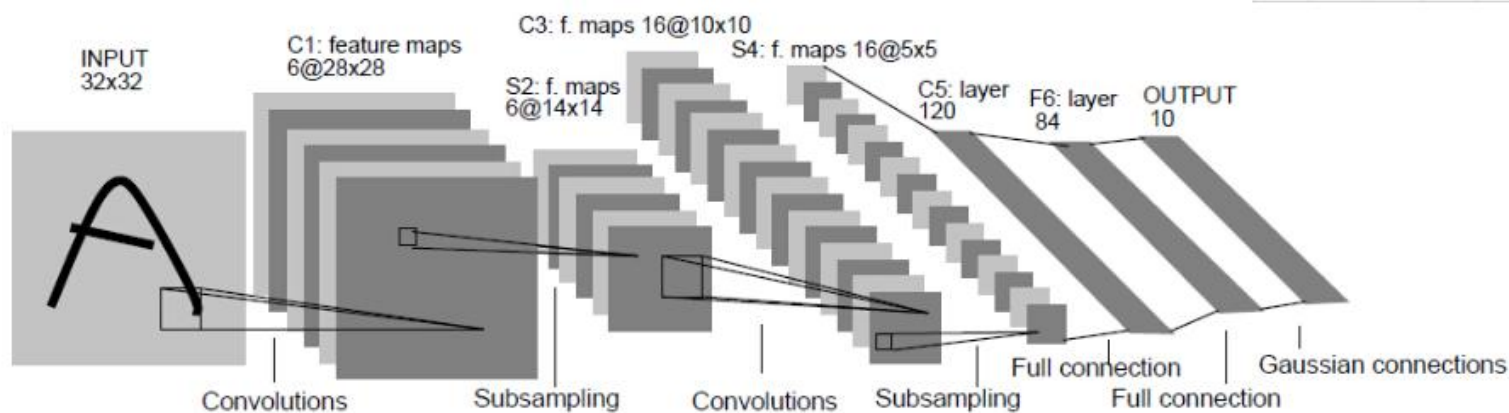
每个卷积核（5×5）与原始的输入图像（32×32）进行卷积，这样得到的feature map（特征图）大小为 $(32-5+1) \times (32-5+1) = 28 \times 28$

- 参数多少？

由于参数（权值）共享的原因，对于同个卷积核每个神经元均使用相同的参数，因此，参数个数为 $(5 \times 5 + 1) \times 6 = 156$ ，其中5×5为卷积核参数，1为偏置参数



开山之作：LeNet5



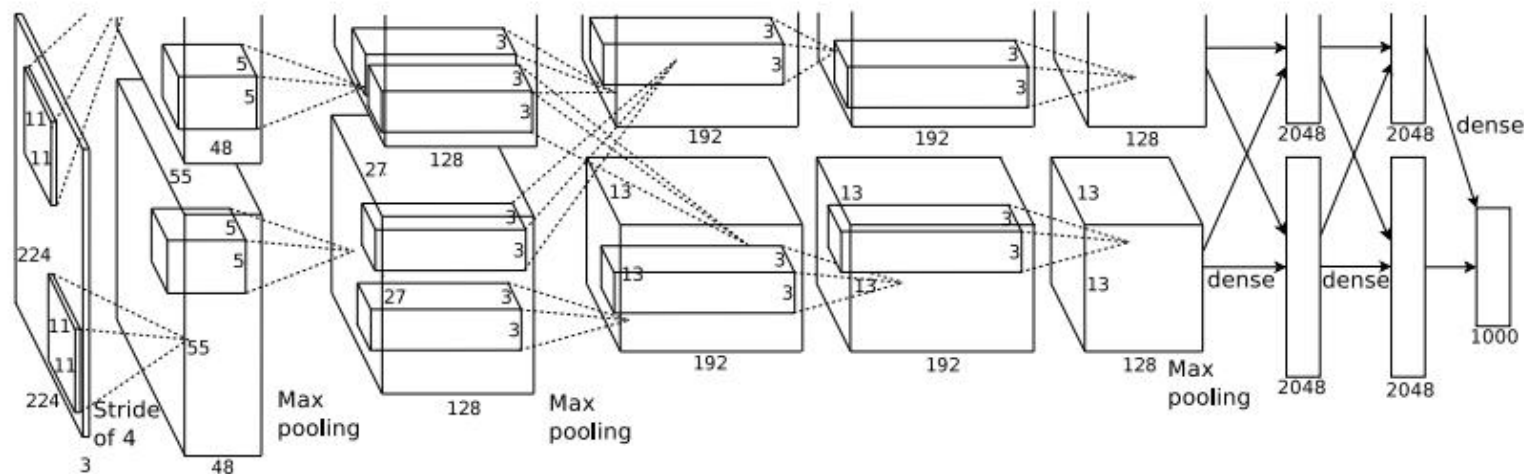
S2层（下采样层，也称池化层）：6@14×14

- 这一层主要是做池化或者特征映射（特征降维），池化单元为2×2
- 池化后的大小？

6个特征图的大小经池化后即变为14×14。经2×2池化后，每两行两列重新算出一个特征值出来，相当于图像大小减半，因此卷积后的28×28图像经2×2池化后就变为14×14。



一战成名：AlexNet



2012年，AlexNet在ImageNet竞赛中超过第二名10.9个百分点，成绩显赫。

- 更深的网络
- 数据增广：随机裁剪
- ReLU
- dropout
- 多GPU训练

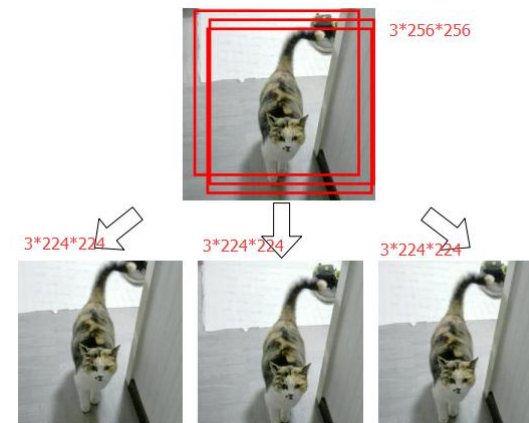
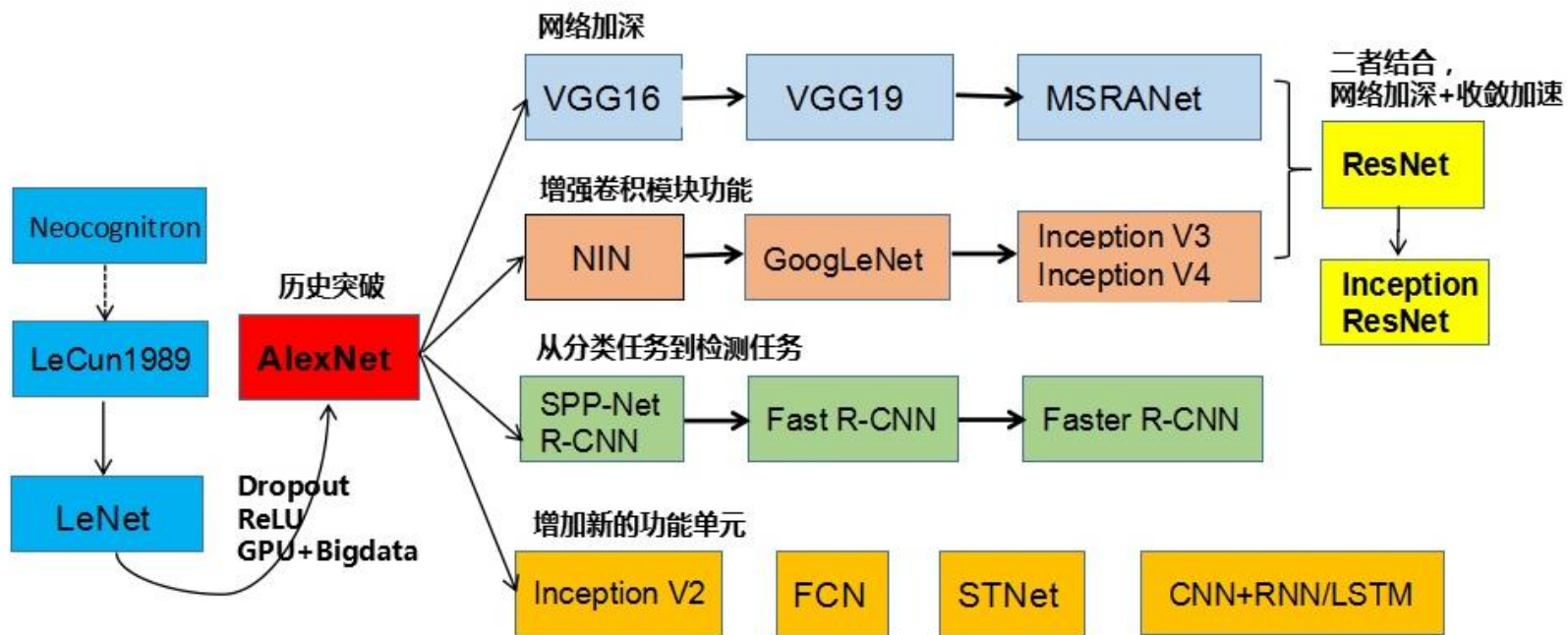


图 4-15 随机裁剪

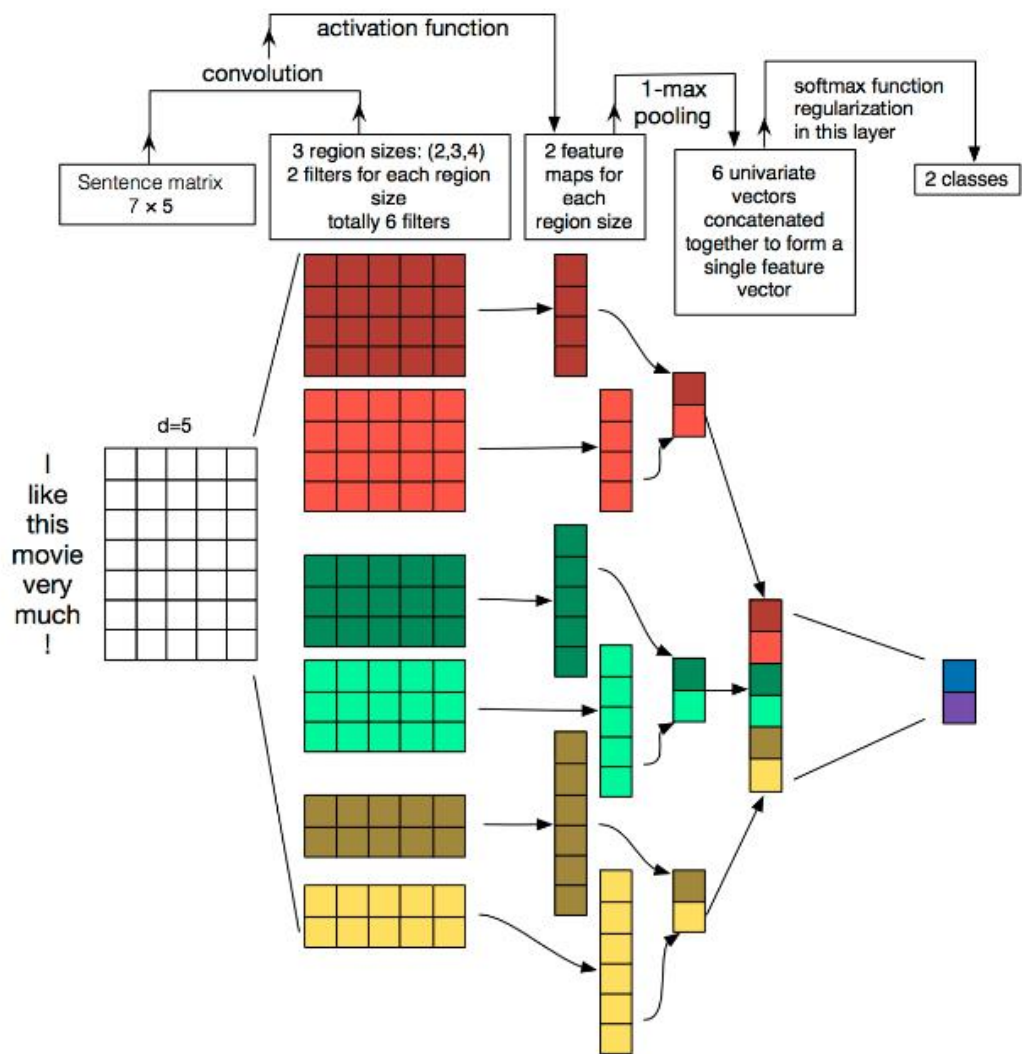


CNN进化史





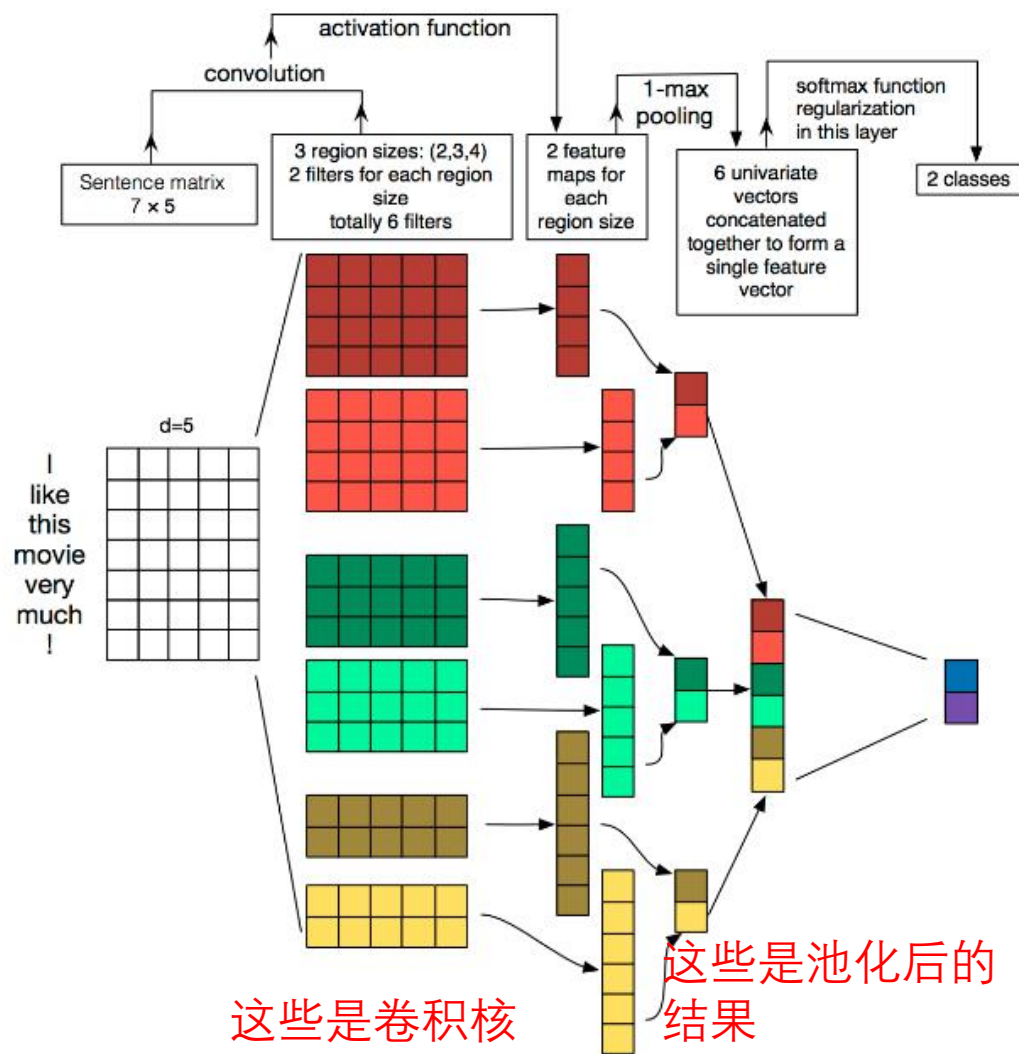
应用：文本



- 关键词，关键短语→局部性
- 关键信息可在不同位置出现→平移性
- 适当去除一些文字内容不影响语义理解→可缩性



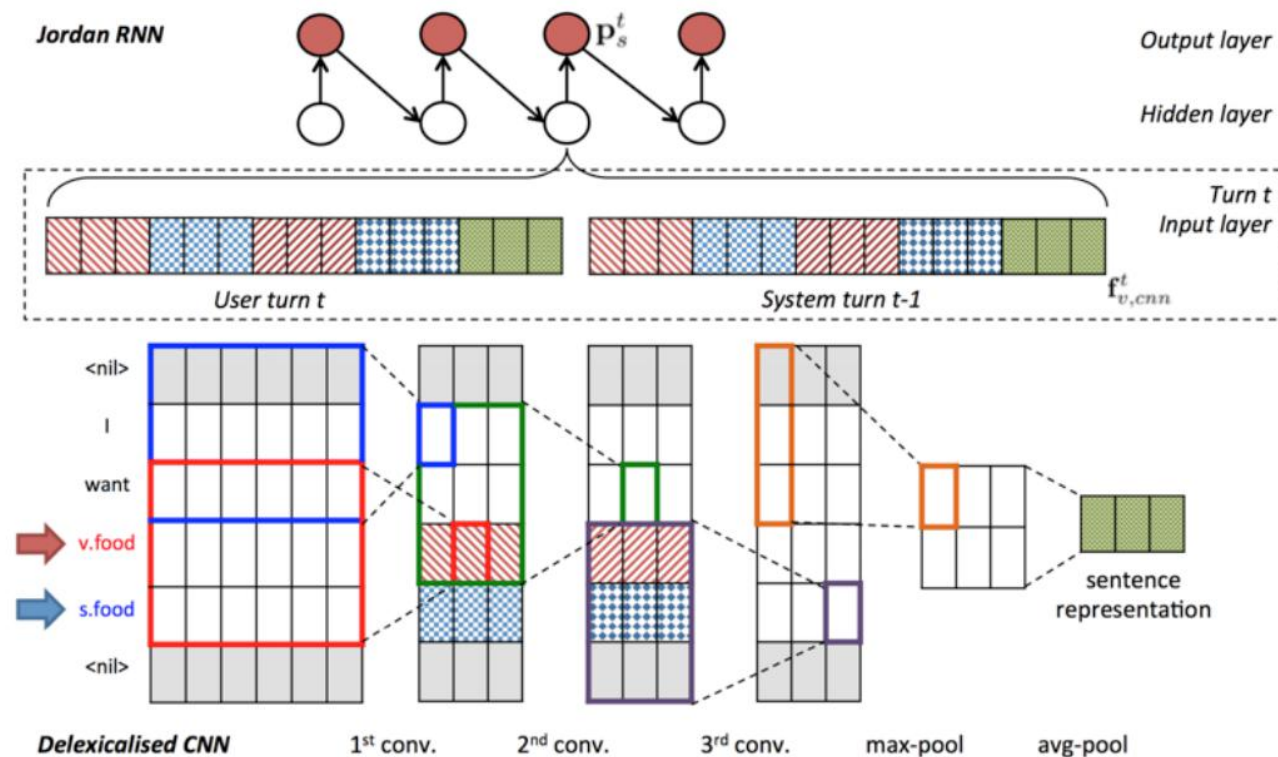
应用：文本



- 在text-CNN中，卷积核的宽度是与词向量的维度一致。
- 用卷积核进行卷积时，不仅考虑了词义而且考虑了词序及其上下文（类似于N-gram的思想）。
- A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification



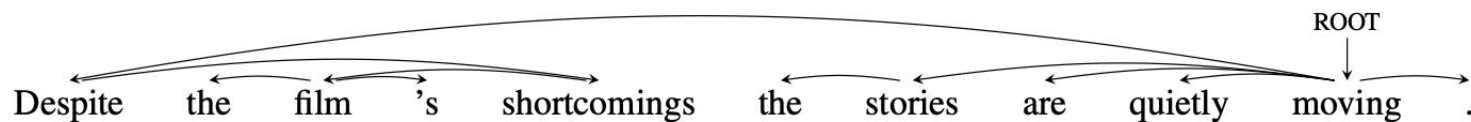
应用：文本



- 多轮对话：抓取更多的上下文信息。
- A Network-based End-to-End Trainable Task-oriented Dialogue System
- <https://github.com/shawnwun/NNDIAL>



应用：文本



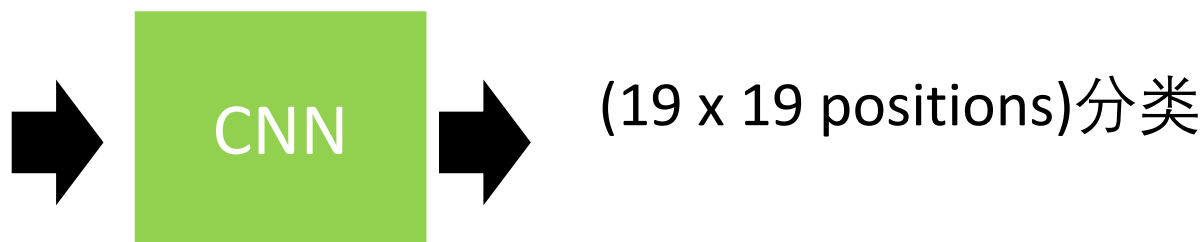
- 基于依存关系的卷积：不仅仅是利用句子中相邻的词信息作为特征信息，应用依存句法树的将句子的语义信息关系真正地提取出来。
- Dependency-based Convolutional Neural Networks for Sentence Embedding



应用：围棋



19 x 19 matrix
(image)



由于围棋的棋盘其实可以看成是一个19*19的图片，每个像素就是对应的位置的状态（是黑棋，还是白棋，还是没放棋）。因此我们可以把每个盘面表示成一个三通道的图片，对于位置 p ：

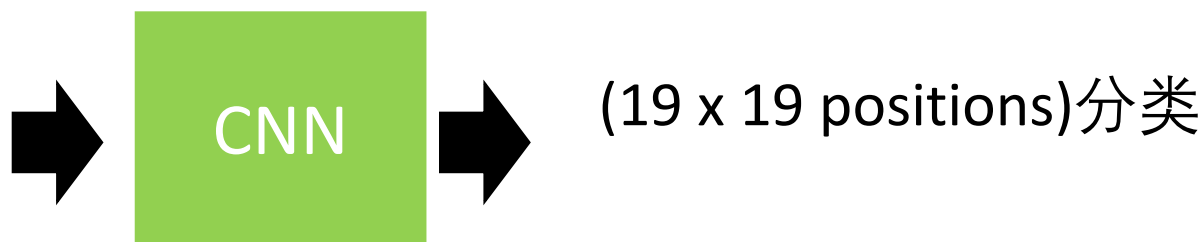
- 通道0 = 表示 p 放的我方的子
- 通道1 = 表示 p 放的对方的子
- 通道2 = 表示 p 没放子



应用：围棋



19 x 19 matrix
(image)



加围棋知识, 比如棋串, 气, 上一步的信息:

通道0 = 表示p放的是我方的棋子

通道1 = 表示p放的是对方的棋子

通道2 = 表示没有放子

通道3 = 表示p放的是我方的棋, 且于p相连的棋串只有1口气

通道4 = 表示p放的是我方的棋, 且于p相连的棋串有2口气

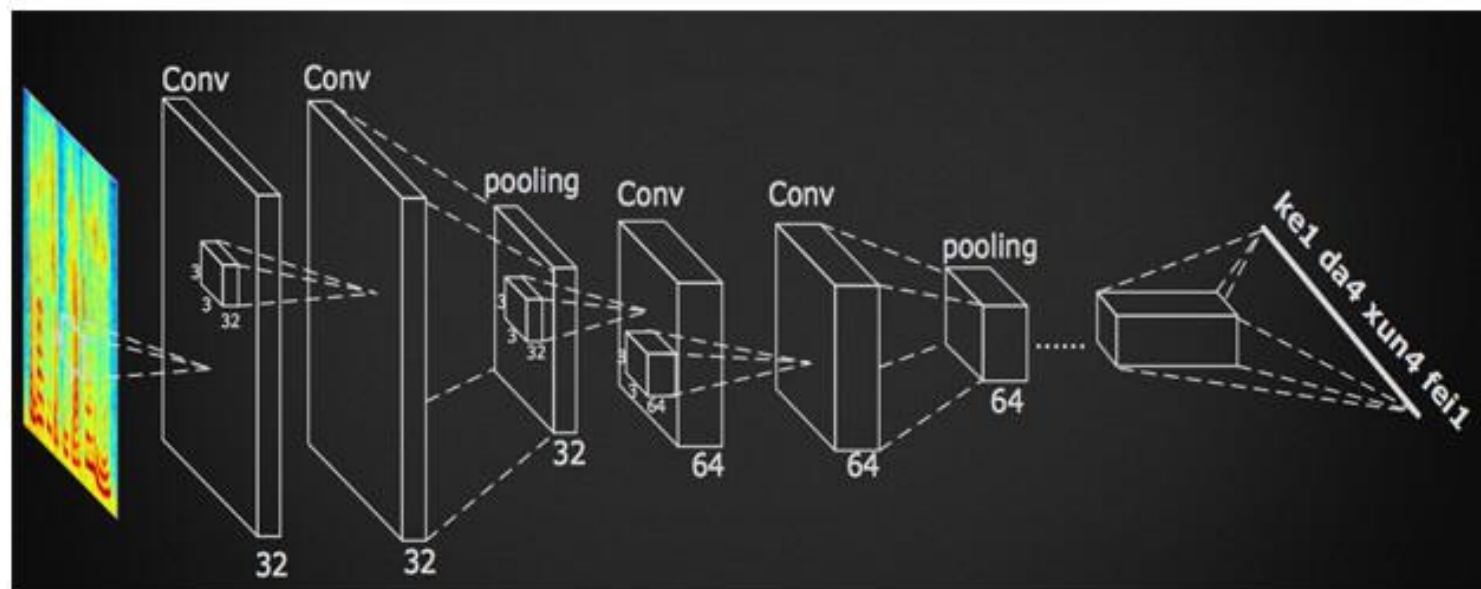
通道5 = 表示p放的是我方的棋, 且于p相连的棋串有3口气

...

通道11 = 表示p是上一步刚刚落的子



应用：语音识别



DFCNN使用大量的卷积层直接对整句语音信号进行建模。

- 在输入端DFCNN直接将语谱图作为输入，相比其他以传统语音特征作为输入的语音识别框架相比具有天然的优势。
- 在模型结构上，借鉴了图像识别的网络配置，每个卷积层使用小卷积核，并在多个卷积层之后再加上池化层，通过累积非常多的卷积池化层对，从而可以看到非常长的历史和未来信息。



参考资料

- 深度全序列卷积神经网络克服LSTM缺陷，成功用于语音转写
https://mp.weixin.qq.com/s?__biz=MzA4NjM4ODQzNQ==&mid=2651545214&idx=1&sn=a6b66b2b53040f6cc1f76c48c6f0bd80&scene=25#wechat_redirect
- CNN for visual recognition
<https://cs231n.github.io/understanding-cnn/>
- S. Zhang, C. Liu, H. Jiang, S. Wei, L. Dai, and Y. Hu, “Feedforward sequential memory networks: A new structure to learn long-term dependency,” arXiv preprint arXiv:1512.08301, 2015.
- 科大讯飞最新语音识别系统和框架深度剖析
<https://www.leiphone.com/news/201608/4HJoePG2oQfGpoj2.html>
- <https://my.oschina.net/u/876354/blog/1632862>
- <https://www.cnblogs.com/skyfsm/p/8451834.html>