西安交通大学 XI'AN JIAOTONG UNIVERSITY | IAIR Est. 1986 Institute of Artificial Intelligence and Robotics | 人工智能学院 College of Artificial Intelligence, XJTU

《高级机器学习》第七章

神经网络模型与深度学习
Neural Networks and Deep Learning

魏平
西安交通大学人工智能学院
人工智能与机器人研究所

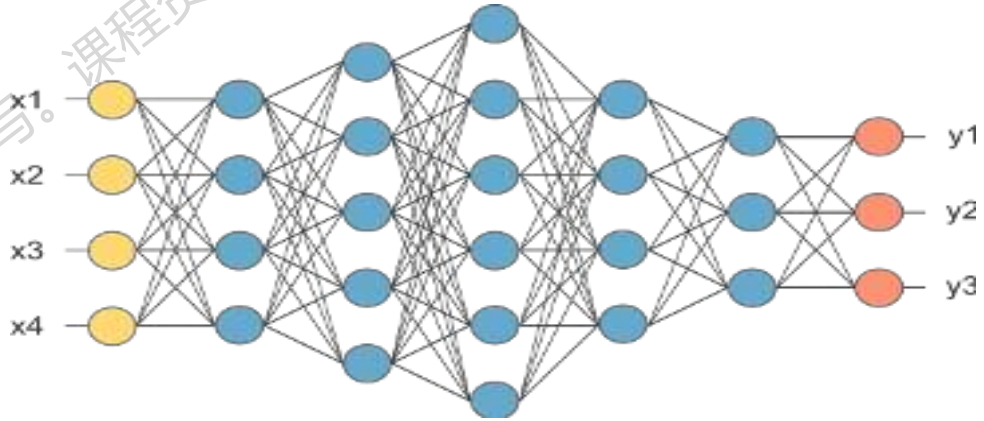
IAIR Est. 1986 Institute of Artificial Intelligence and Robotics

人工智能学院 College of Artificial Intelligence, XJTU

CONTENTS

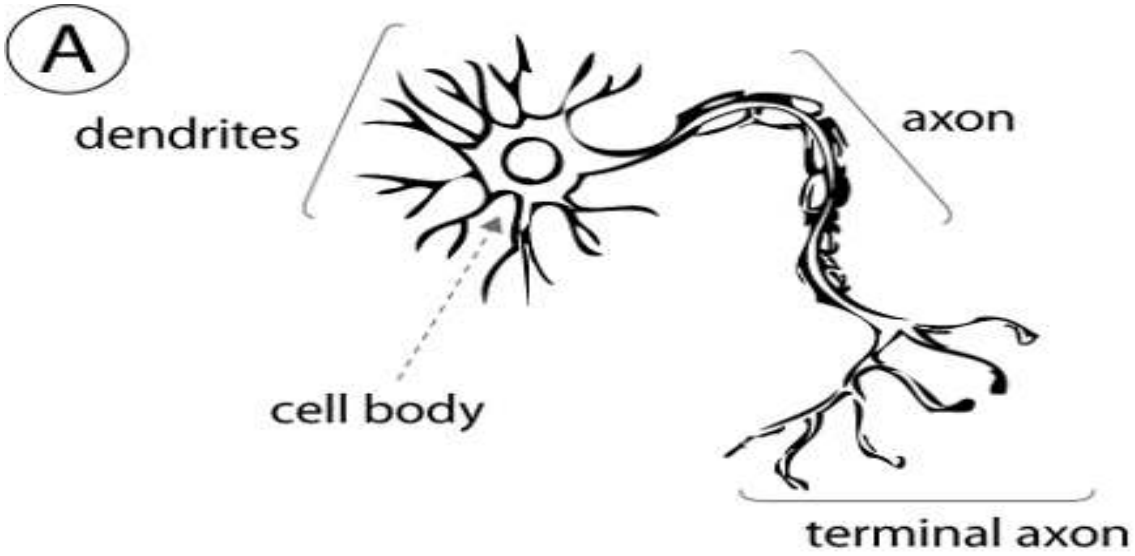
什么是人工神经网络?

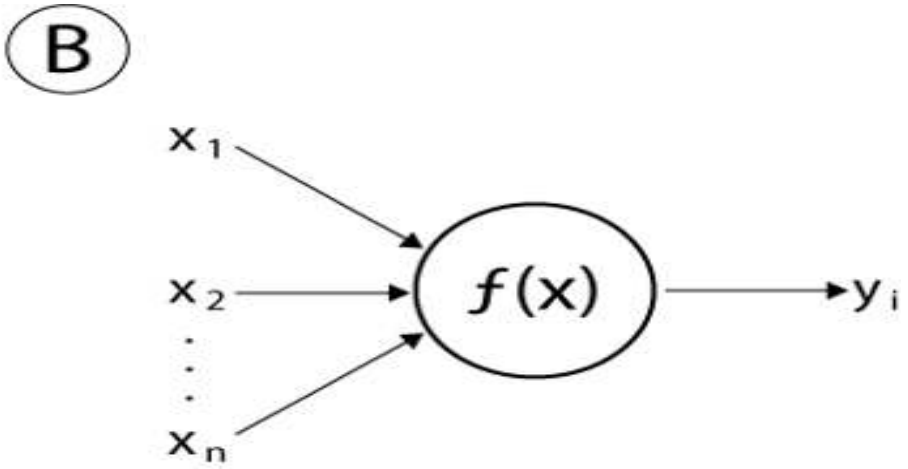
- 人工神经网络(Artificial Neural Network)简称神经网络，在机器学习和认知科学领域，是一种模仿生物神经网络的结构和功能的数学模型或计算模型，用于对函数进行估计或近似

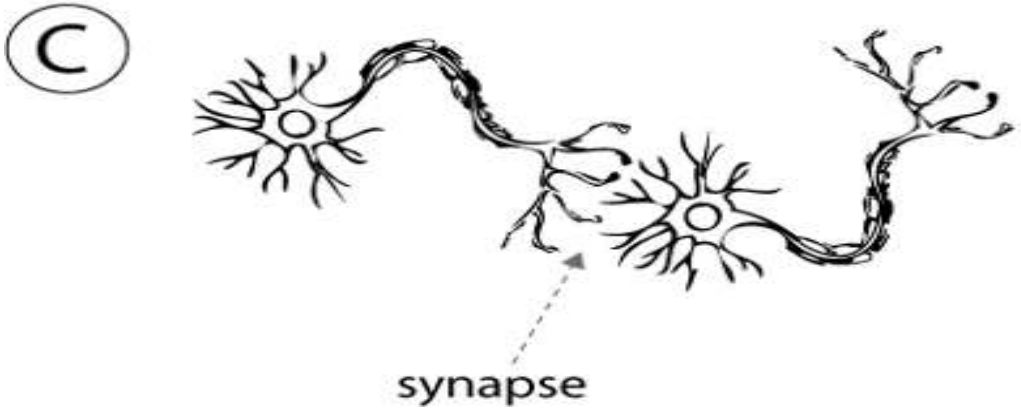


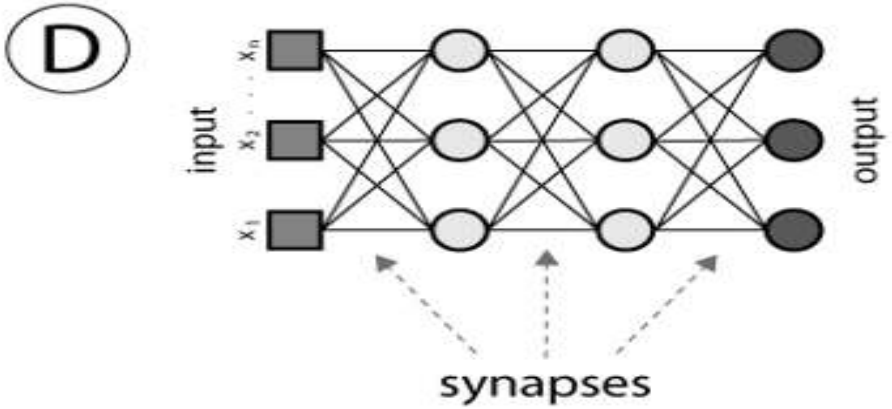
- **神经网络一般包含以下三个部分：**
  - **结构（Architecture）**
    网络中的变量和它们的拓扑关系
  - **激励函数（Activity Rule）**
    神经元根据其他神经元的活动来改变自己的激励值
  - **学习规则（Learning Rule）**
    网络中的权重如何学习和调整

人工神经网络的生物学启发









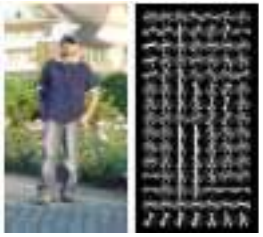
前馈神经网络基本概念

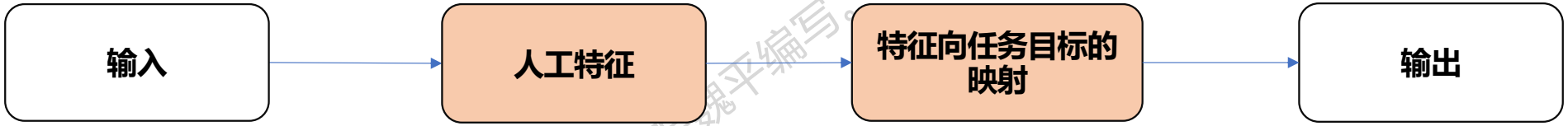
- 机器学习任务可以表达为一个广义映射$\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$；$f(\boldsymbol{x};\boldsymbol{\theta})$可以是线性模型$f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b}$，也可以是非线性模型
- ϕ是一个非线性函数，非线性模型可看做线性模型作用于输入$\boldsymbol{x}$的非线性变化$\phi(\boldsymbol{x})$，$f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\omega}^{\mathrm{T}}\phi(\boldsymbol{x})$，$\phi(\boldsymbol{x})$称为输入$\boldsymbol{x}$的特征
- **构建ϕ的三种方式：**
  - 通用映射ϕ，将$\boldsymbol{x}$变换至无限维空间$\phi(\boldsymbol{x})$
  - 人工设计ϕ，以人的经验选择特征，如边缘、HOG、SIFT
  - 自主学习ϕ，从数据中挖掘和学习实现某一任务目标的最佳$\phi(\boldsymbol{x})$。深度学习和神经网络即属于此类

前馈神经网络基本概念

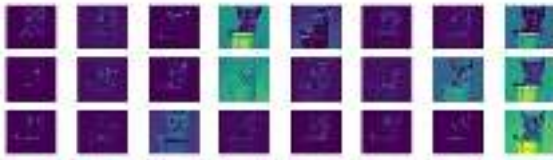
- 人工设计特征与自主学习特征

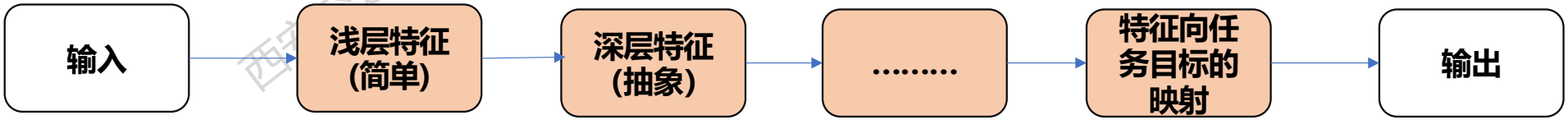
  - **传统机器学习算法**





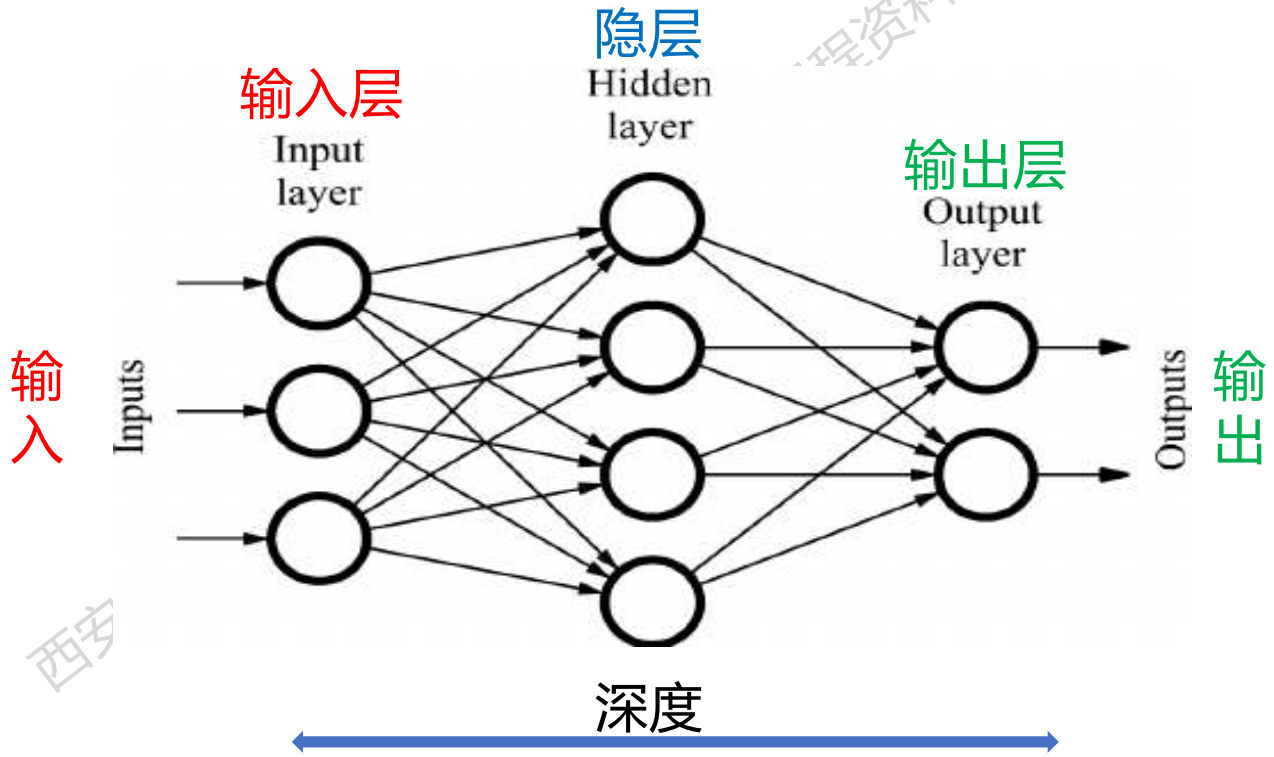
  - **神经网络方法**





前馈神经网络基本概念

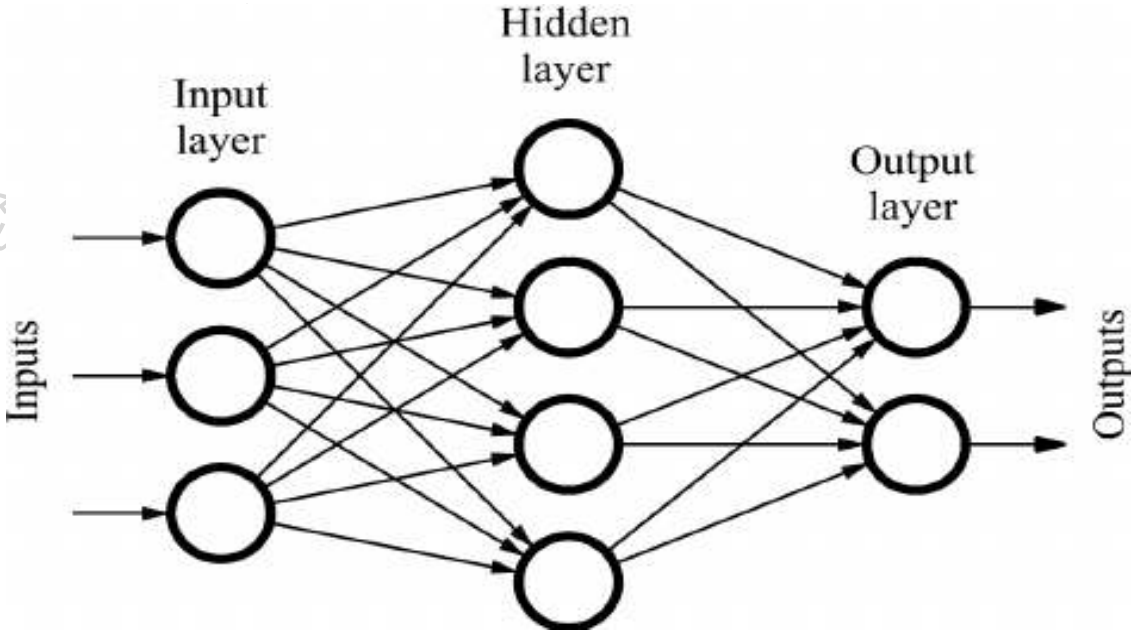
- 前馈神经网络 （feedforward neural network）也叫**多层感知机** （Multilayer perceptron），是一种在模型输出与模型本身之间没有反馈连接的神经网络



前馈神经网络基本概念

- 前馈网络定义了一个函数$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$来近似某个目标函数$f^*$；它学习参数$\boldsymbol{\theta}$的值，使其尽可能地近似$f^*$
- 前馈网络通常用许多不同函数复合在一起来表示；例如有三个函数$f^{(1)}$、$f^{(2)}$、$f^{(3)}$，在一个网络中可表示为：

$$f(\boldsymbol{x}) = f^{(3)}(f^{(2)}f^{(1)}(\boldsymbol{x}))$$

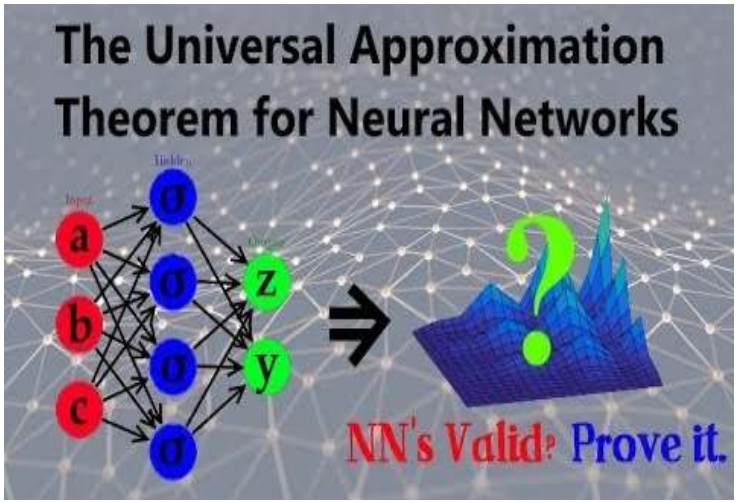


前馈神经网络基本概念

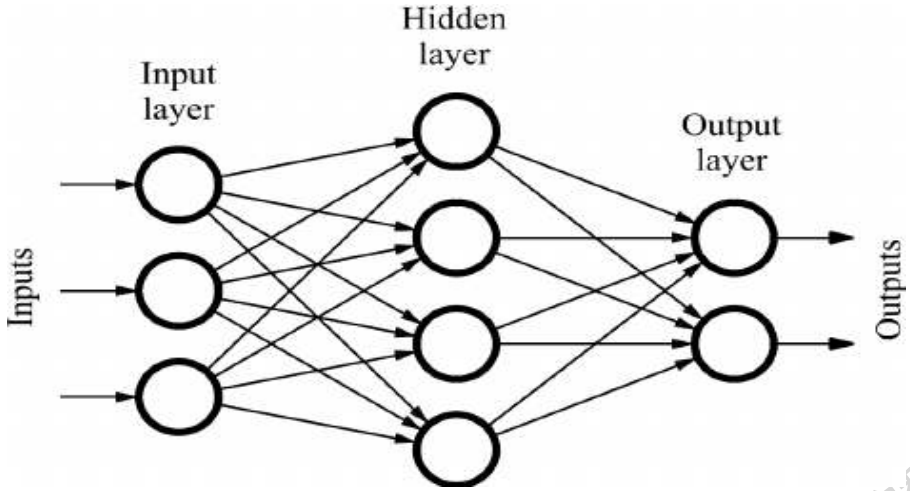
普遍近似原理 (Universal Approximation Theorem)

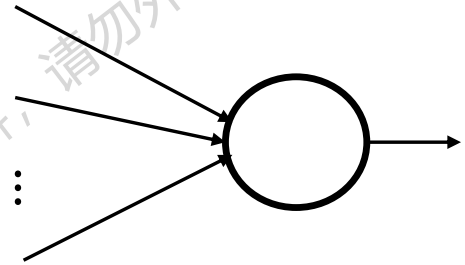
一个前馈神经网络如果具有线性输出层和至少一层具有任何一种“挤压”性质的激活函数的隐藏层，只要给予网络足够数量的隐藏单元，它可以以任意的精度来近似任何**从一个有限维空间到另一个有限维空间**的Borel可测函数

- Cybenko, G., Approximations by superpositions of sigmoidal functions, *Mathematics of Control, Signals, and Systems, 1989*
- Hornik, K., Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 1991

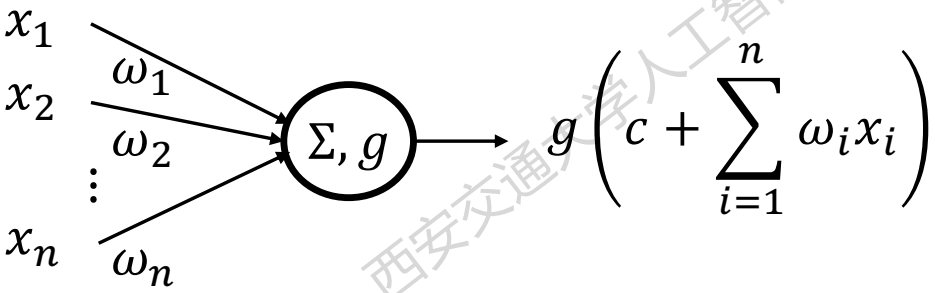


单个神经元 (Single Neuron)





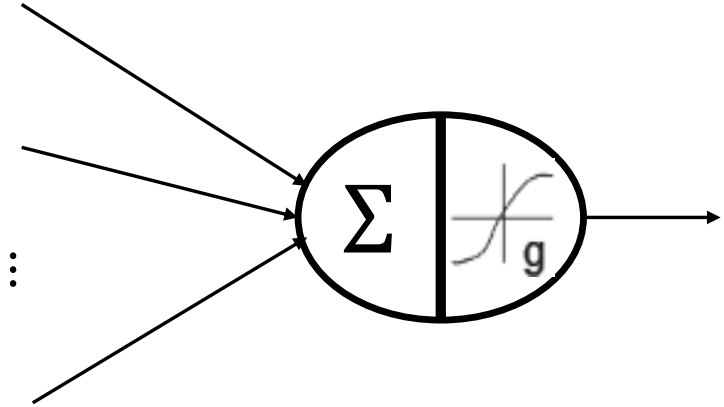
单个神经元

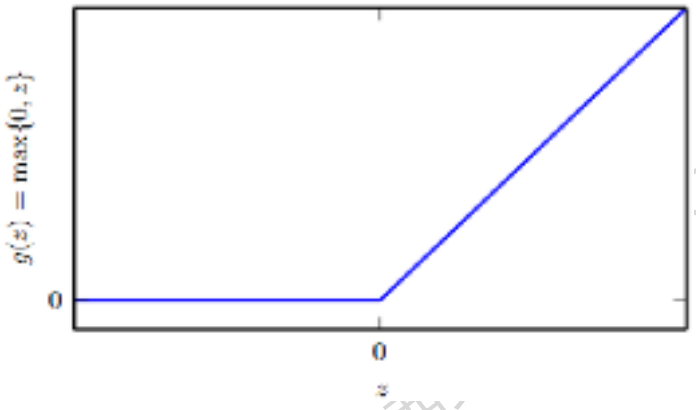


单个神经元计算功能：

1. 求和 Σ
2. 激励 g

常见的激活函数



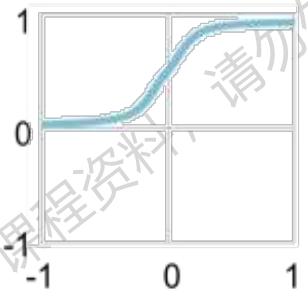


整流线性单元

Rectified Linear Unit (ReLU)

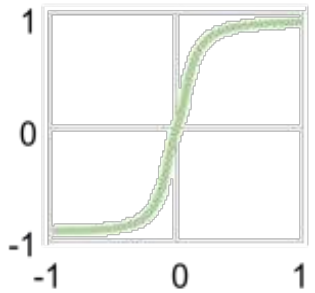
Traditional Non-Linear Activation Functions

Sigmoid



$y=1/(1+e^{-x})$

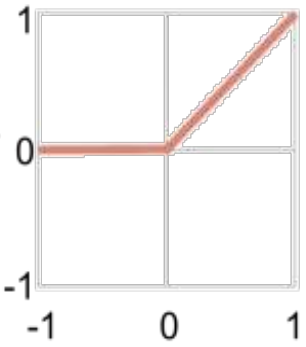
Hyperbolic Tangent



$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$

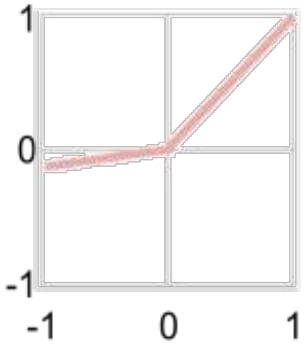
Modern Non-Linear Activation Functions

Rectified Linear Unit (ReLU)



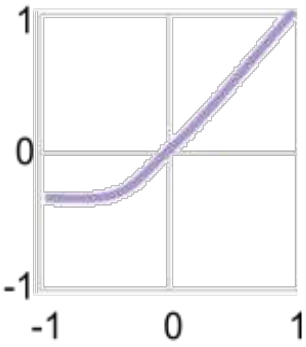
$y=\max(0,x)$

Leaky ReLU



$y=\max(\alpha x,x)$

Exponential LU

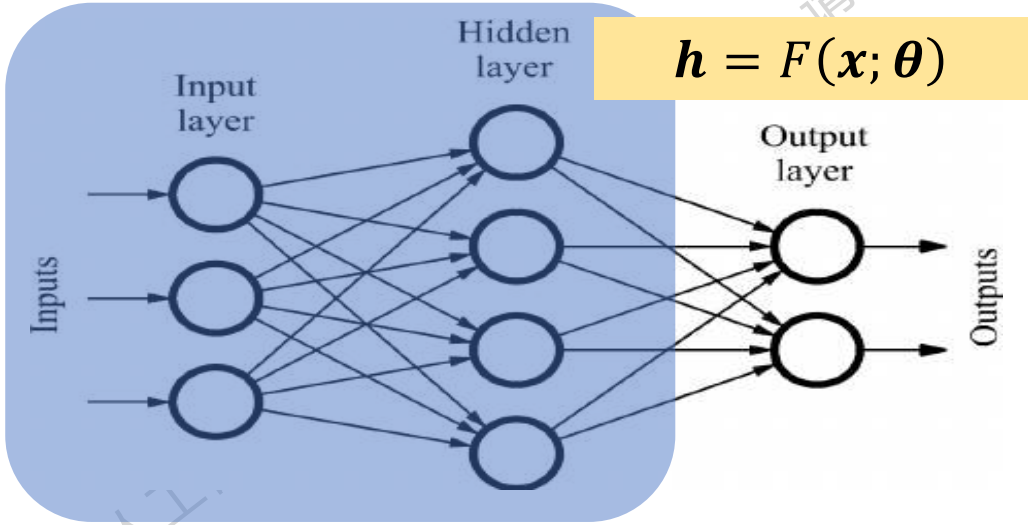


$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^{x}-1), & x<0 \end{cases}$

α = small const. (e.g. 0.1)

输出单元

- 除了输出层的神经网络是一个函数$\boldsymbol{h} = F(\boldsymbol{x}; \boldsymbol{\theta})$，$\boldsymbol{h}$称作特征，输出单元处理从特征到最终的结果输出



- 线性单元 (Linear Units) – 回归问题

$$\hat{\boldsymbol{y}} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{h} + \boldsymbol{b}$$

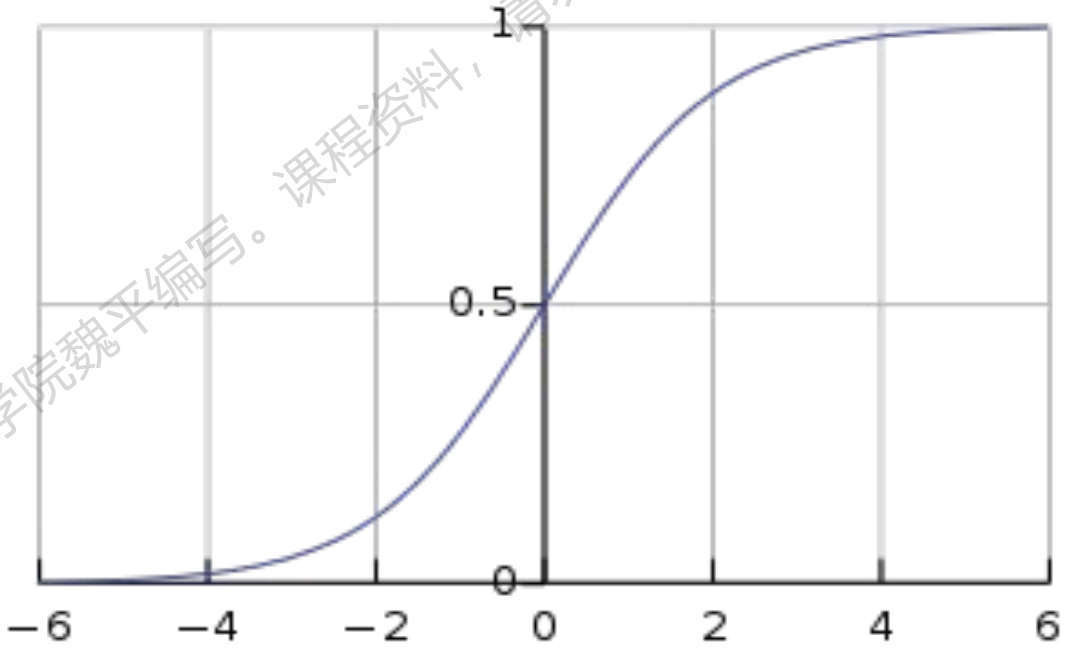
$$p(\boldsymbol{y}|\boldsymbol{x}) = N(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I})$$

输出单元

- Sigmoid单元 (Sigmoid Units) – 二分类问题

$$\hat{y}_1 = p(y = 1|\boldsymbol{x}) = \sigma\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{h} + \boldsymbol{b}\right)$$

$$\hat{y}_2 = p(y = -1|\boldsymbol{x}) = 1 - \sigma\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{h} + \boldsymbol{b}\right)$$



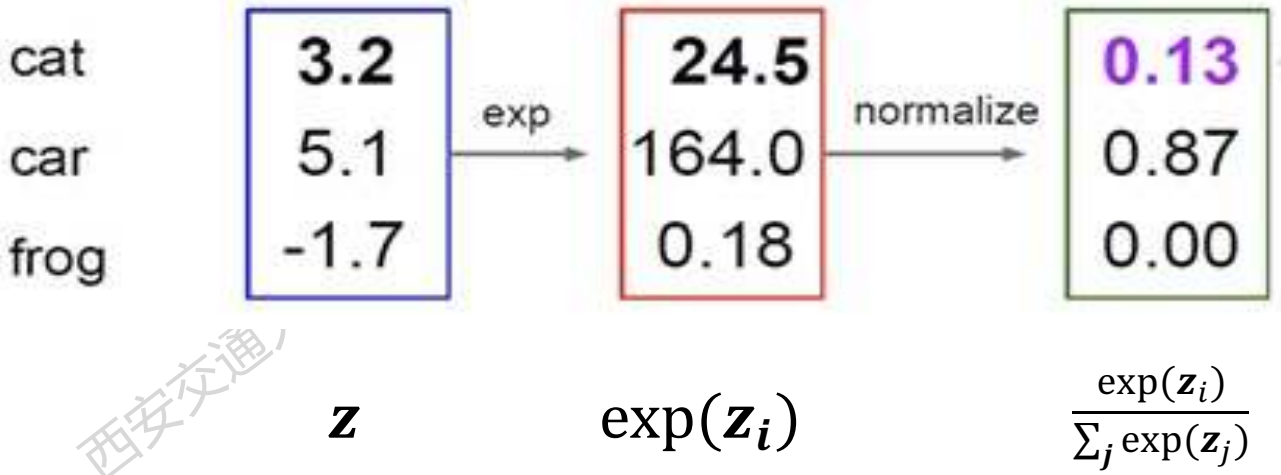
$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

输出单元

- Softmax单元 (Softmax Units) – 多分类问题

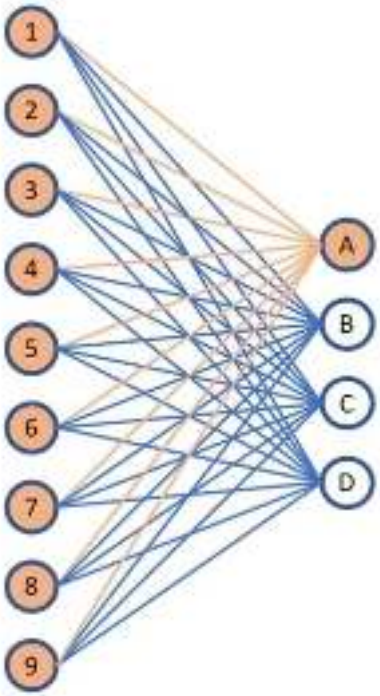
$$\boldsymbol{z} = \boldsymbol{W}^{\mathrm{T}}\boldsymbol{h} + \boldsymbol{b}$$

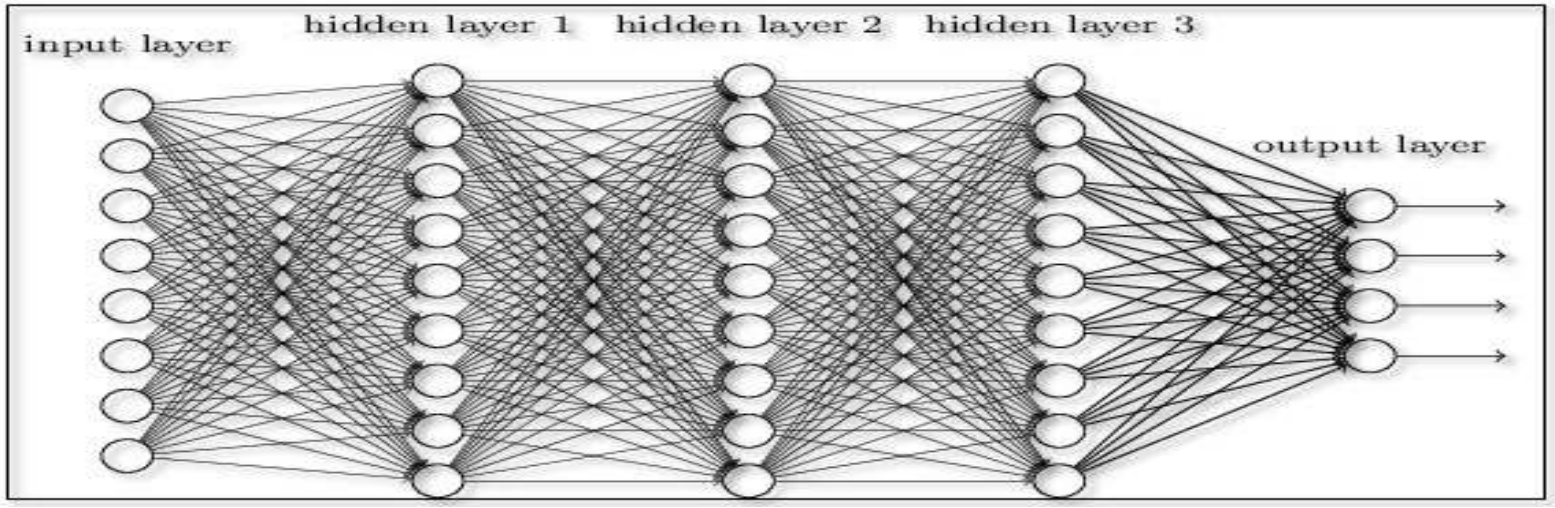
$$\mathrm{softmax}(\boldsymbol{z})_i = \frac{\exp(\boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{z}_j)}$$



全连接层与全连接网络

- 全连接层(fully connected layer, FC)指的是层中的每一个神经元都与前一层中的每个神经元相连接，即每个神经元都对前一层的输出向量做变换；全连接网络指的是网络的每一层都为全连接层

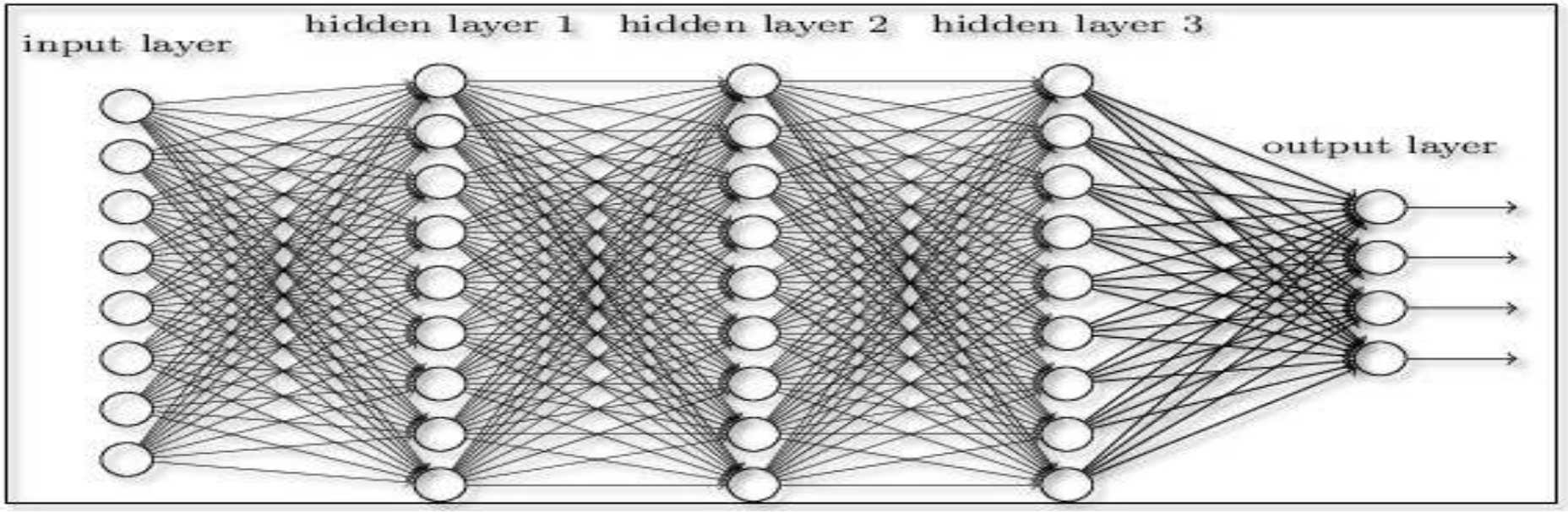




架构设计

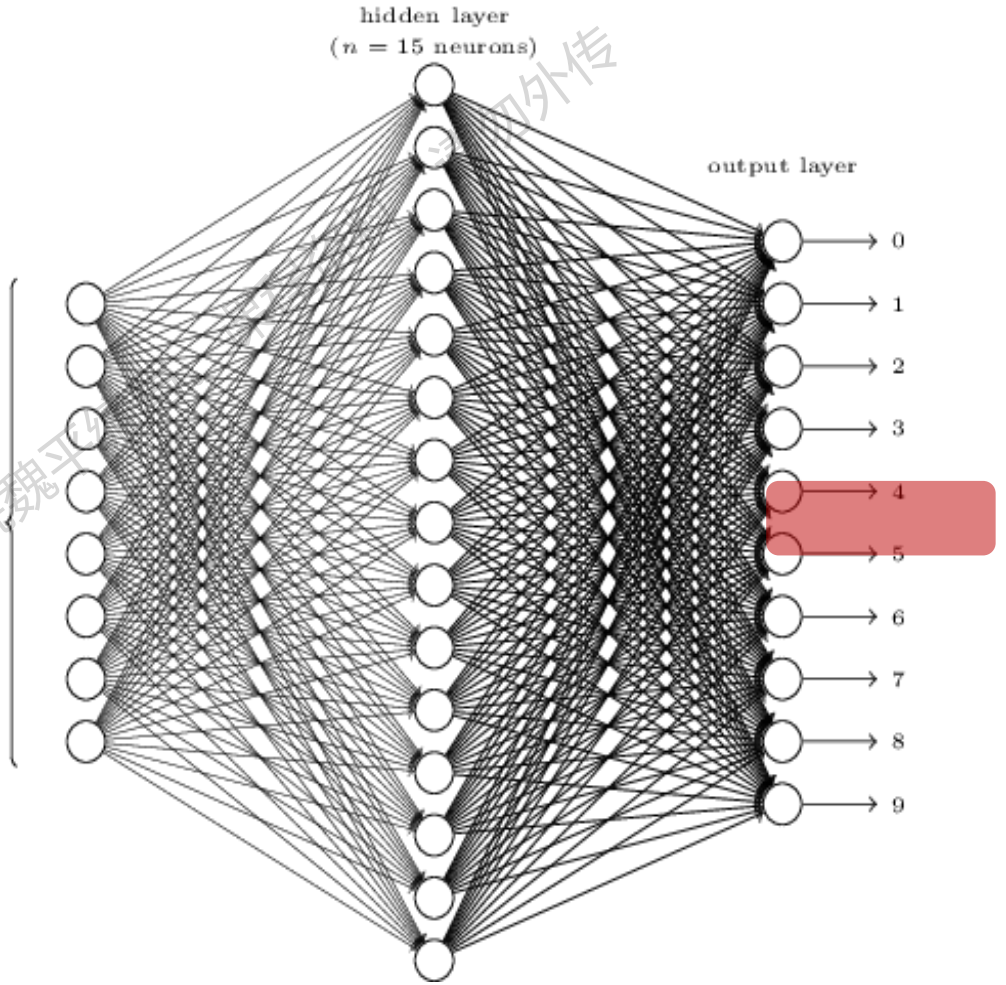
- 神经网络的架构(architecture)设计包括：

- 神经网络包含多少层
- 这些层之间如何连接
- 每层包含多少个单元



例子

- 三层神经网络
- 784维28*28图像输入
- 隐层15个神经元
- 10个输出单元
- 74行代码
- 96%正确率



5

IAIR Est. 1986 Institute of Artificial Intelligence and Robotics

人工智能学院 College of Artificial Intelligence, XJTU

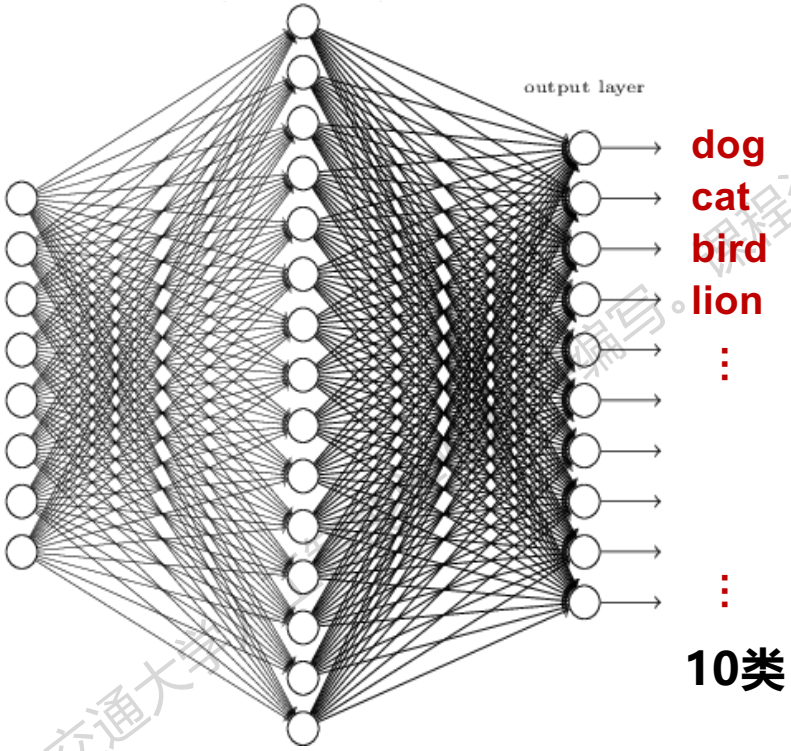
CONTENTS

- 神经网络基本概念
- 典型神经网络
- 深度学习与反向传播算法

卷积神经网络

全连接网络的缺陷1：参数量巨大

500 × 500 × 3



15个神经元

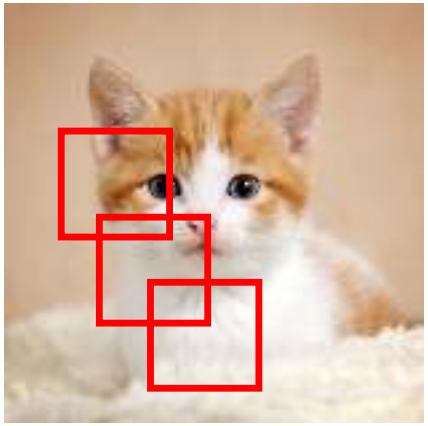
总参数量=

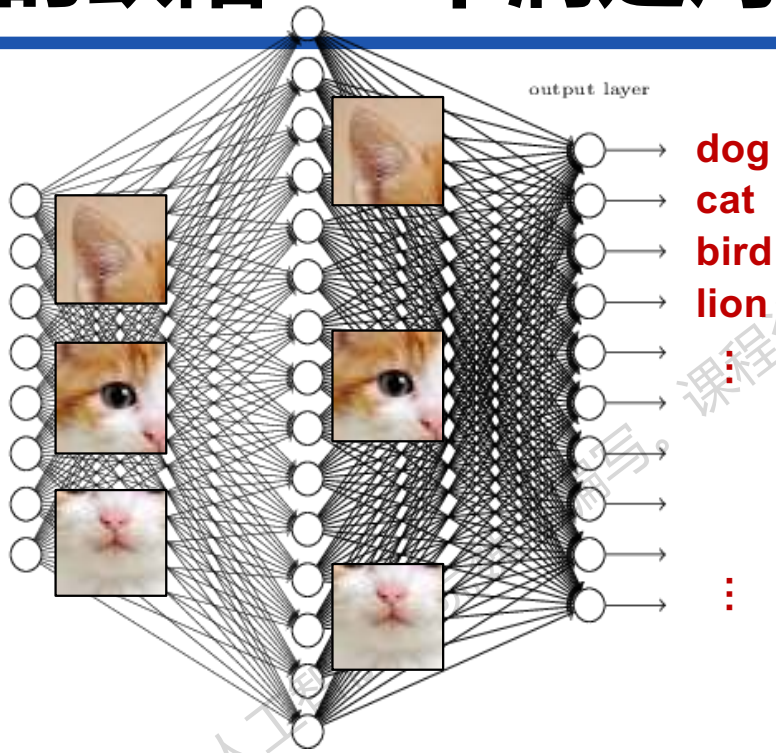
15 × (500 × 500 × 3+1)

+10 × (15+1)

=3 750 175

全连接网络的缺陷2：不满足局部不变性



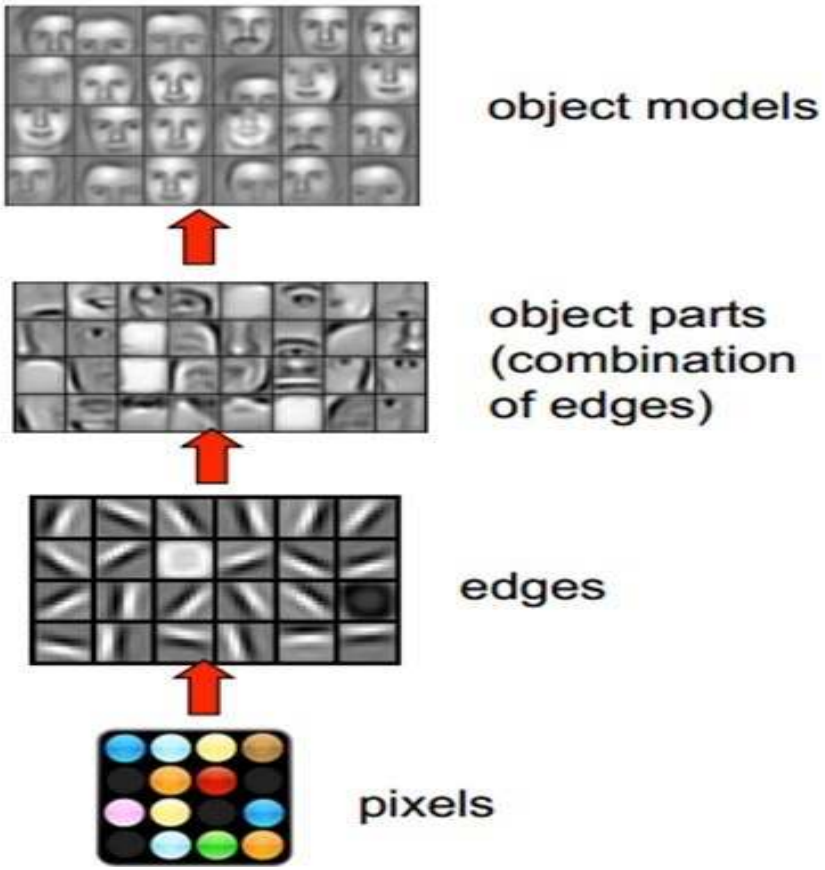


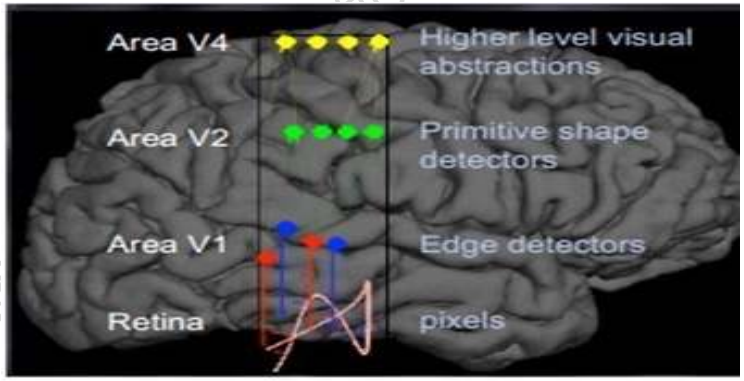
- **局部不变性特征**：特征对一定范围内的缩放、平移、旋转等不敏感
- 自然图像中的物体具有局部不变性特征，缩放、平移、旋转等操作不应影响目标语义信息

全连接前馈网络很难保持局部不变性

人类的视觉原理

- 视觉系统的分级信息处理 – 1981年诺贝尔生理及医学奖





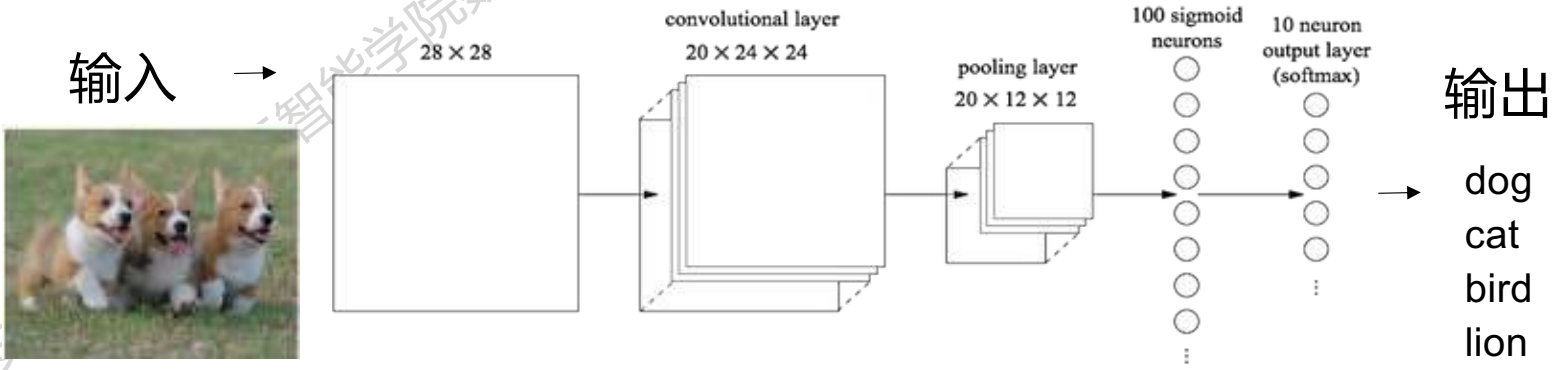
David Hunter Hubel

Torsten Nils Wiesel

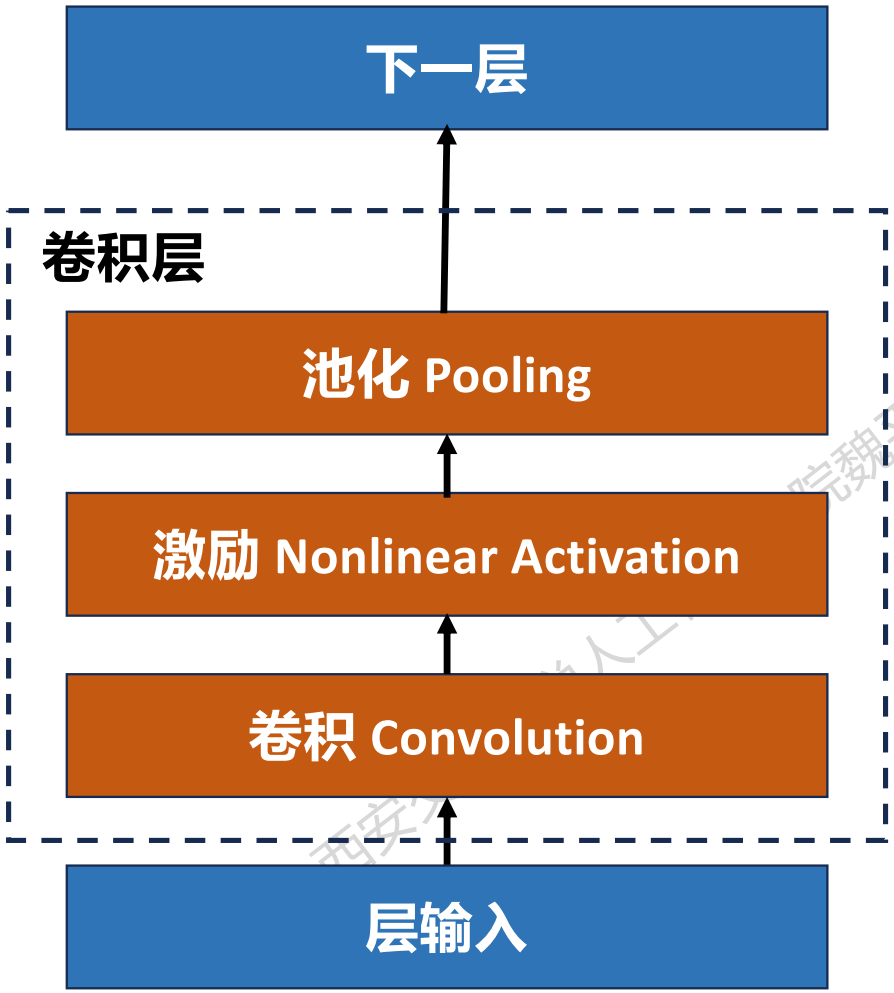
卷积神经网络(CNN)

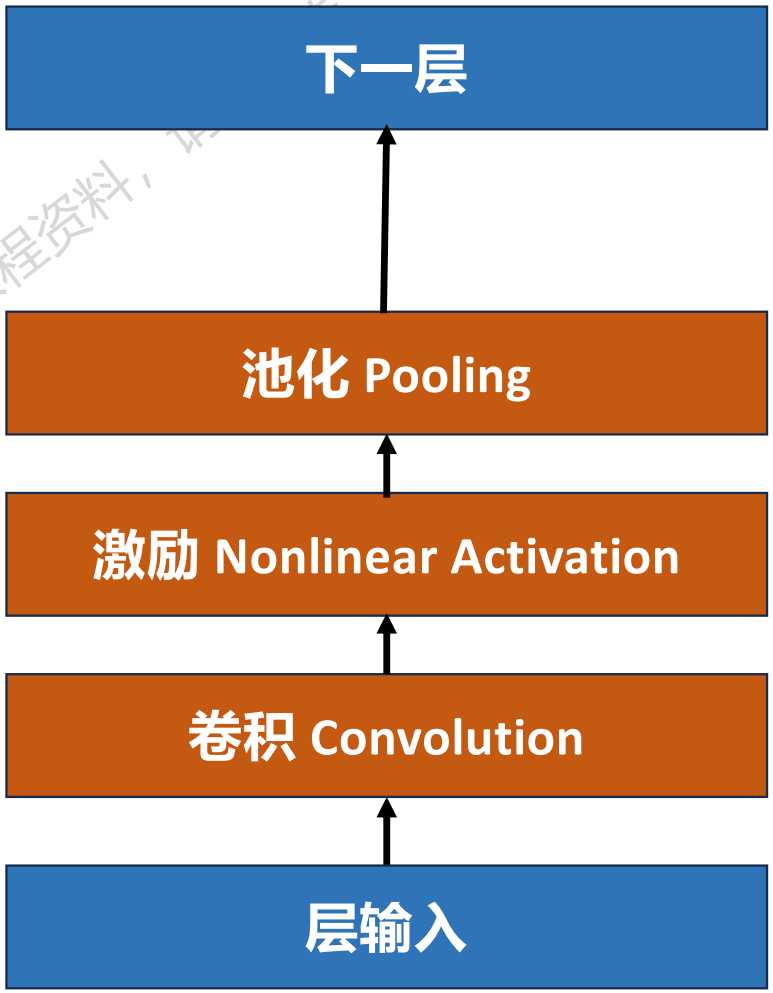
- 卷积神经网络(Convolutional Neural Networks, CNN)是一种前馈神经网络，由一个或多个卷积层和顶端的全连接层组成,通过卷积操作、池化操作和非线性激活函数映射等一系列操作的层层堆叠，将高层语义信息逐层由原始数据输入层中抽取出来，逐层抽象，直至完成目标任务,整个过程“端到端”

- **三个特点:**
  - 局部感知
  - 参数共享
  - 池化



CNN整体结构





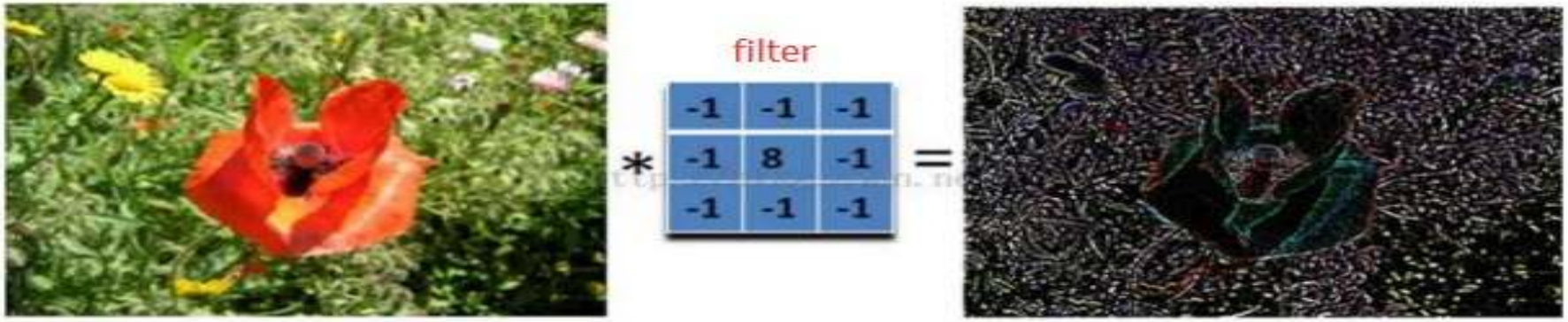
卷积单元 — 图像卷积

- **卷积(convolution):** 是分析数学中一种重要的运算。在信号处理或图像处理中，经常使用一维或二维卷积

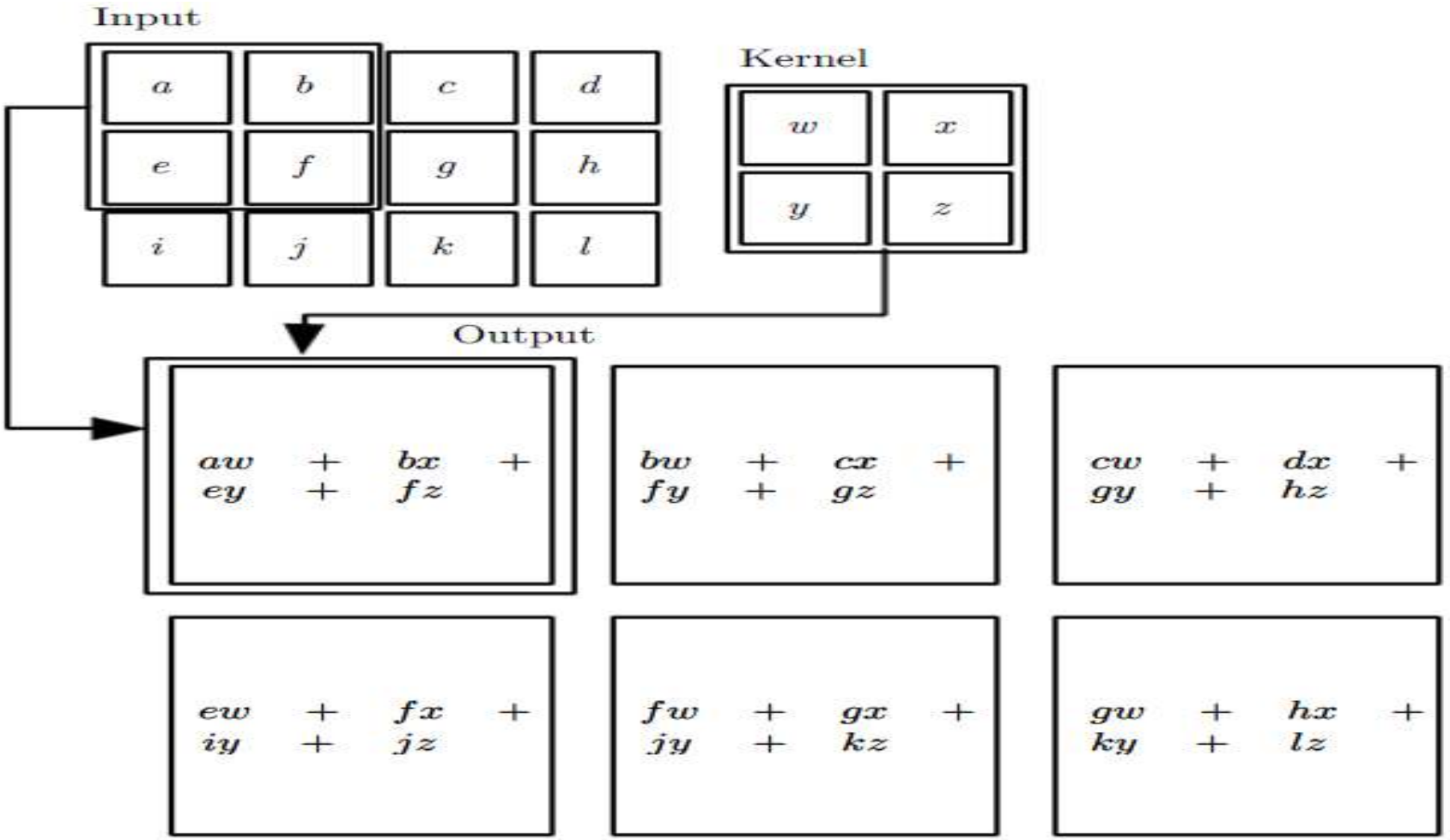
$$Y = W \otimes X$$

$$y_{ij} = \sum_{u=1}^{m}\sum_{v=1}^{n} w_{uv} \cdot x_{i-u+1,j-v+1}$$

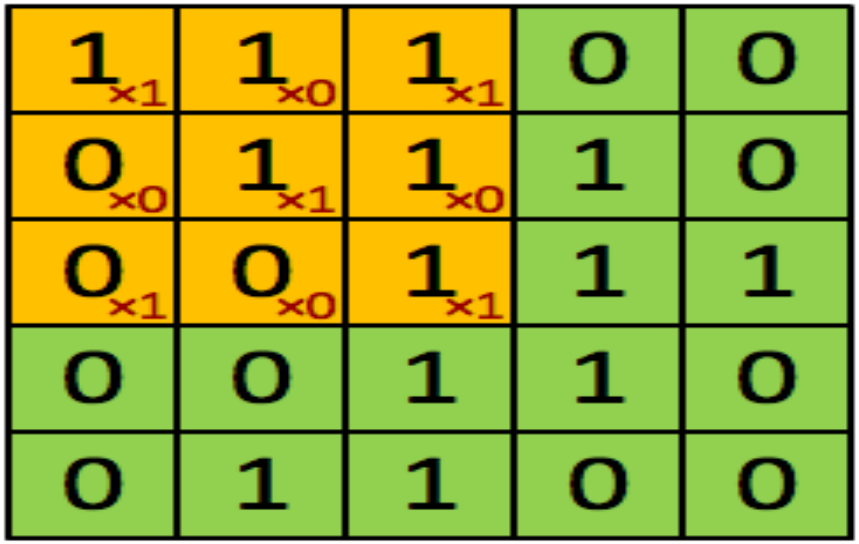
- 图像 $X \in R^{M\times N}$，滤波器 $W \in R^{m\times n}$，$m << M, n << N$



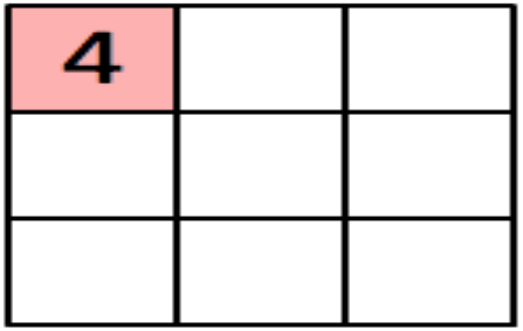
卷积单元— 图像卷积



卷积单元 — 图像卷积



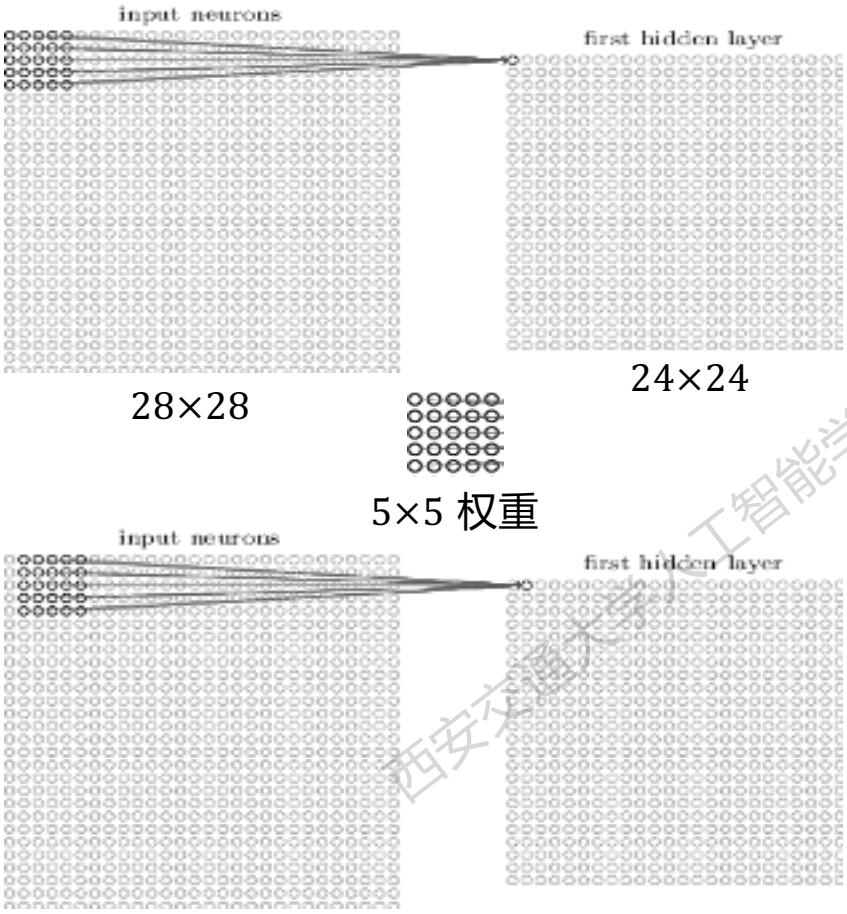
Image



Convolved Feature

卷积单元 — 局部感知域

- 局部感知域 (local receptive field)



stride 步长 = 2

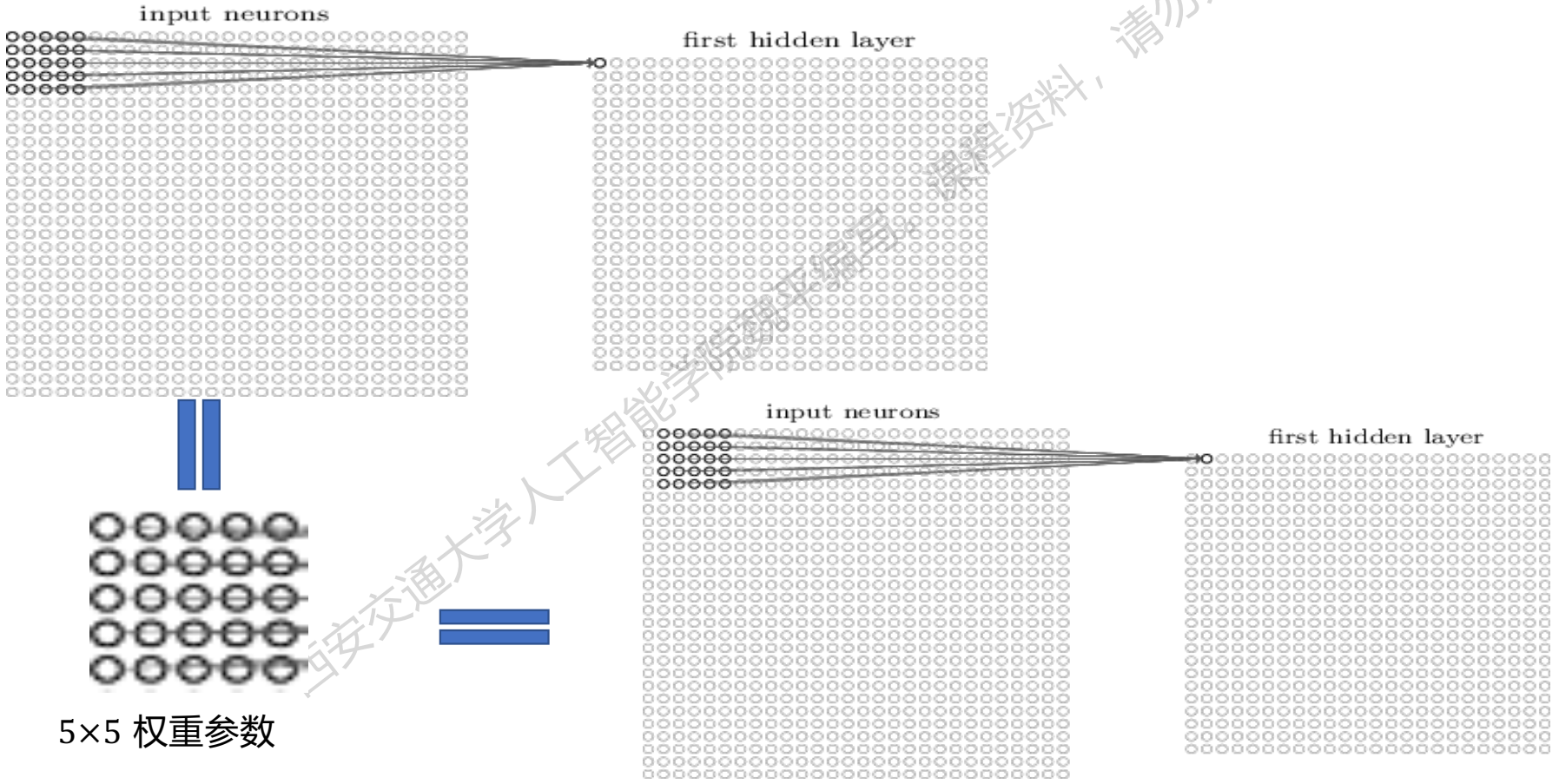
局部感知域

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

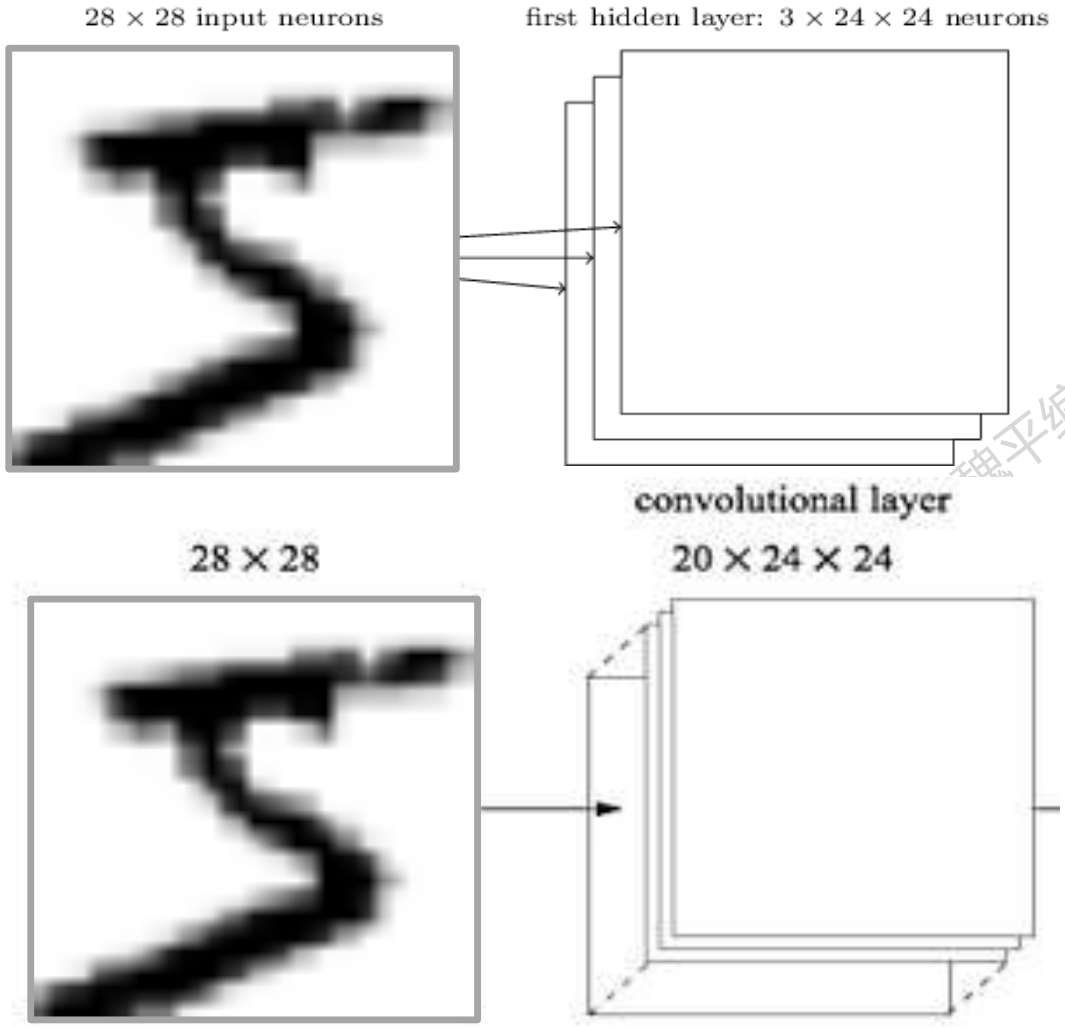
Padding 填补

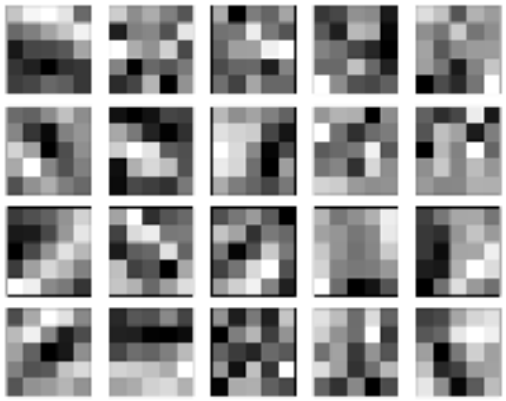
卷积单元 — 参数共享

- 参数共享 (Shared Weights)



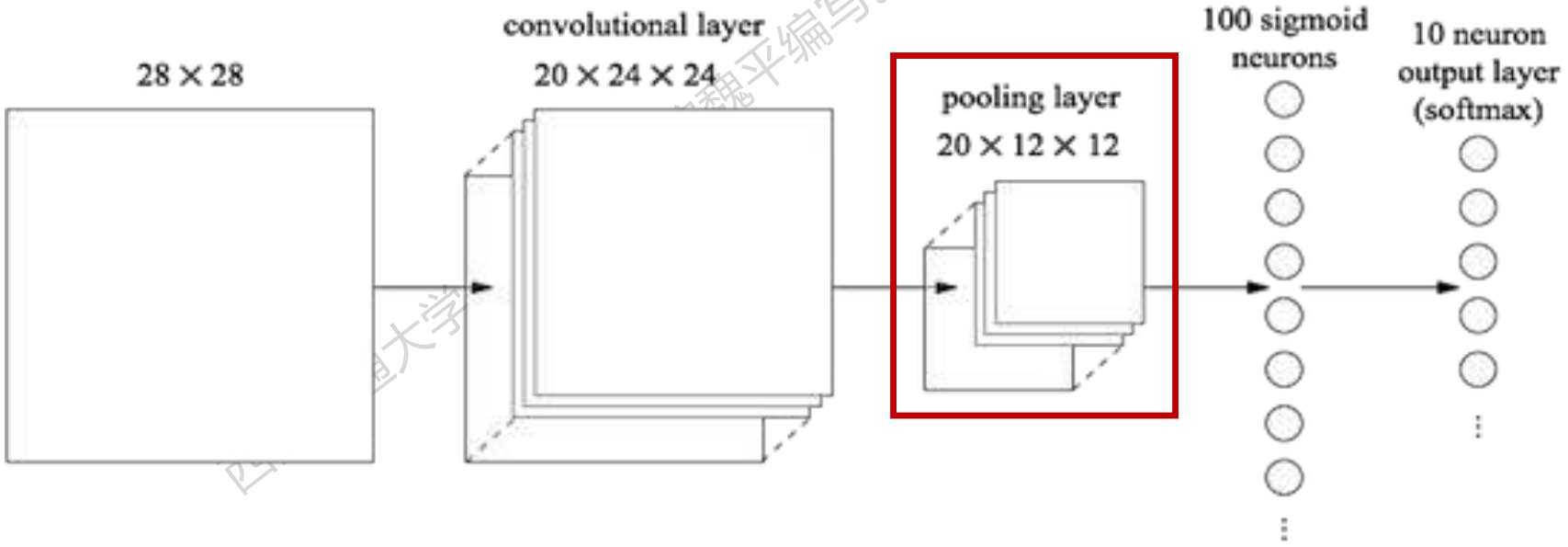
卷积单元 — 多通道卷积



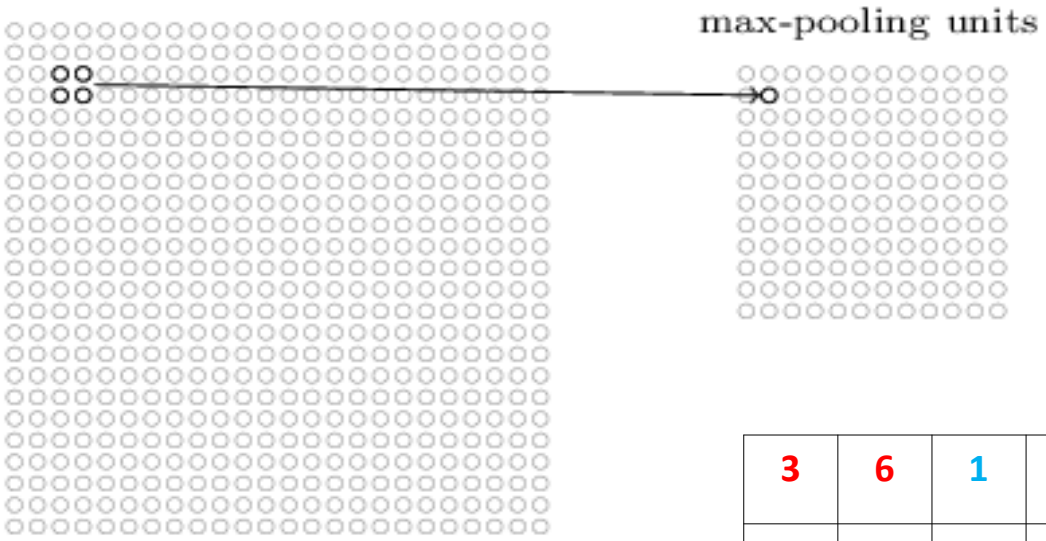


卷积单元 — 池化

- 池化(Pooling)层夹在连续的卷积层中间， 用于压缩数据和参数的量，减小过拟合，常用的池化函数主要为**最大值池化**（maximum pooling）和**平均池化**（mean pooling）



卷积单元 — 池化



3	6	1	4
4	7	7	8
2	2	1	2
2	4	3	4

最大值池化
Max pooling

7	8
4	4

Average pooling
平均值池化

5	5
2.5	2.5

卷积单元的作用

- **局部感知**

每个神经元不对全局图像进行感知，只对局部进行感知，然后在更高层将局部的信息综合起来得到了全局信息

- **权值共享**

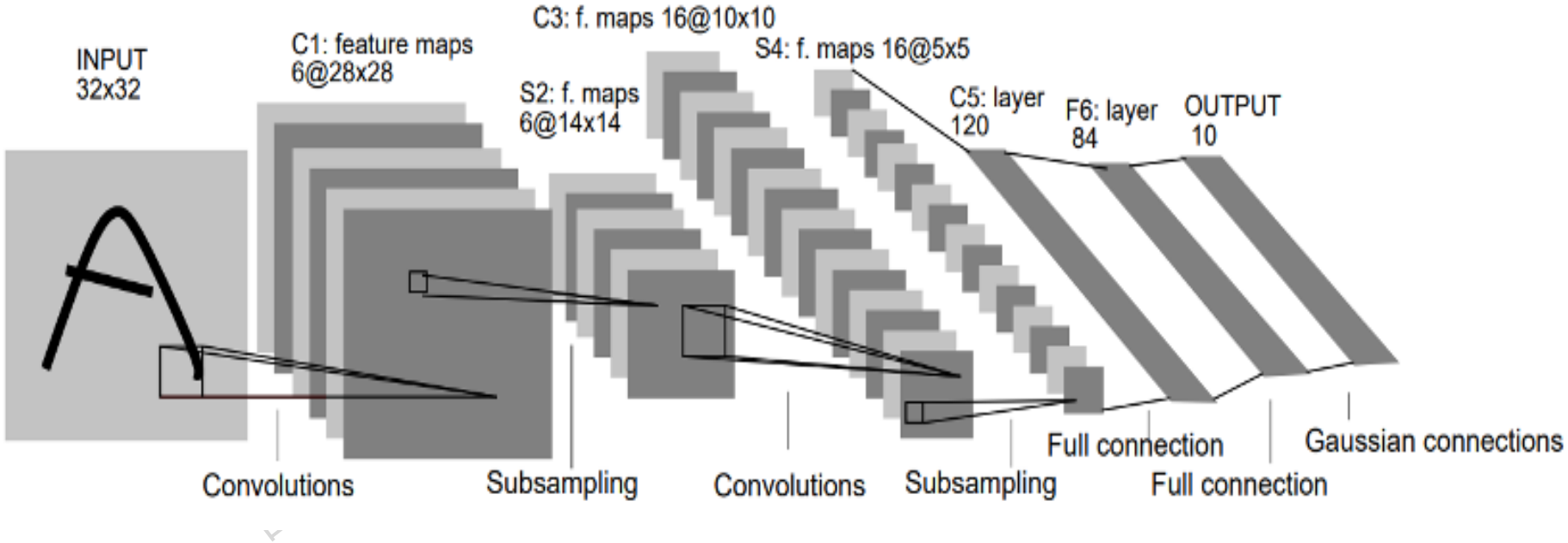
使用同一个滤波器进行卷积，不因位置不同而不同

- **池化降维**

通过池化降低了参数维度，保持了抽象信息，有利于减少过拟合

经典网络

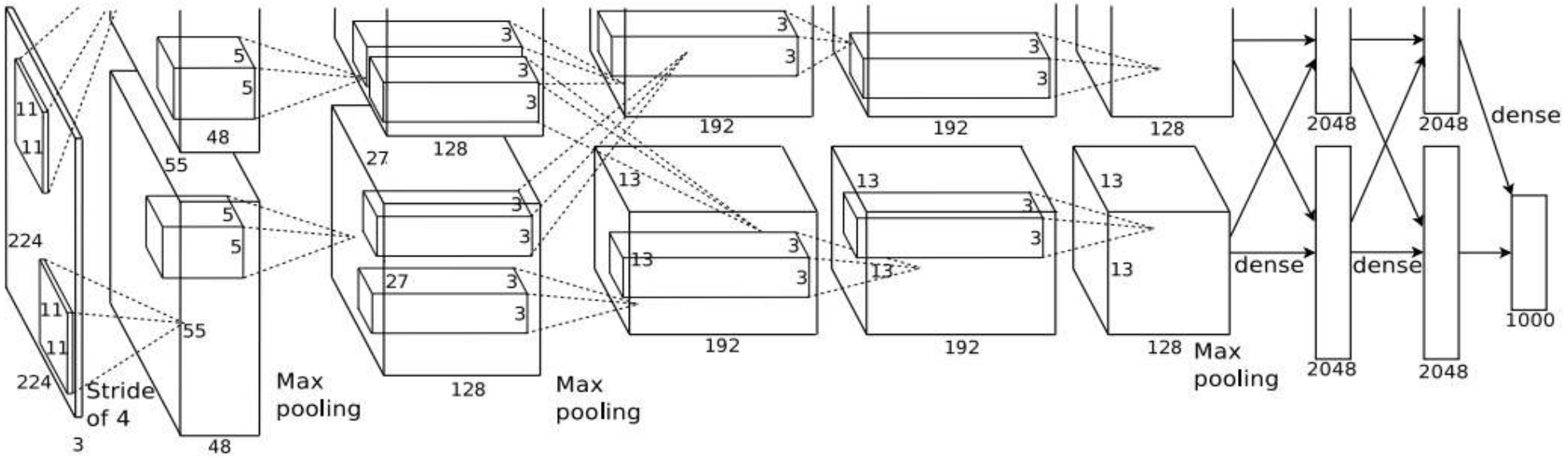
- LeNet-5（1998）
- LeNet-5 共有 7 层，用于手写数字识别，在 90 年代被美国很多银行使用，用来识别支票上面的手写数字



Yann Lecun et al, Gradient-Based Learning Applied to Document Recognition

经典网络

- Alexnet（2012）
- Alexnet是2012年ImageNet=大规模视觉识别挑战(ILSVRC)中的冠军，共8层。首次成功使用Relu激活函数，提出局部响应归一化（LRN）



Geoffrey E. Hinton et al, ImageNet Classification with Deep Convolutional Neural Networks

经典网络

- Alexnet（2012）
- Alexnet是2012年ImageNet=大规模视觉识别挑战(ILSVRC)中的冠军，共8层。首次成功使用Relu激活函数，提出局部响应归一化（LRN）

Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

大规模视觉识别挑战赛(ILSVRC)结果（错误率）比较

Geoffrey E. Hinton et al, ImageNet Classification with Deep Convolutional Neural Networks

经典网络

- VGG（2014）
- VGG在2014年在ILSVRC比赛上获得了分类项目的第二名和定位项目的第一名

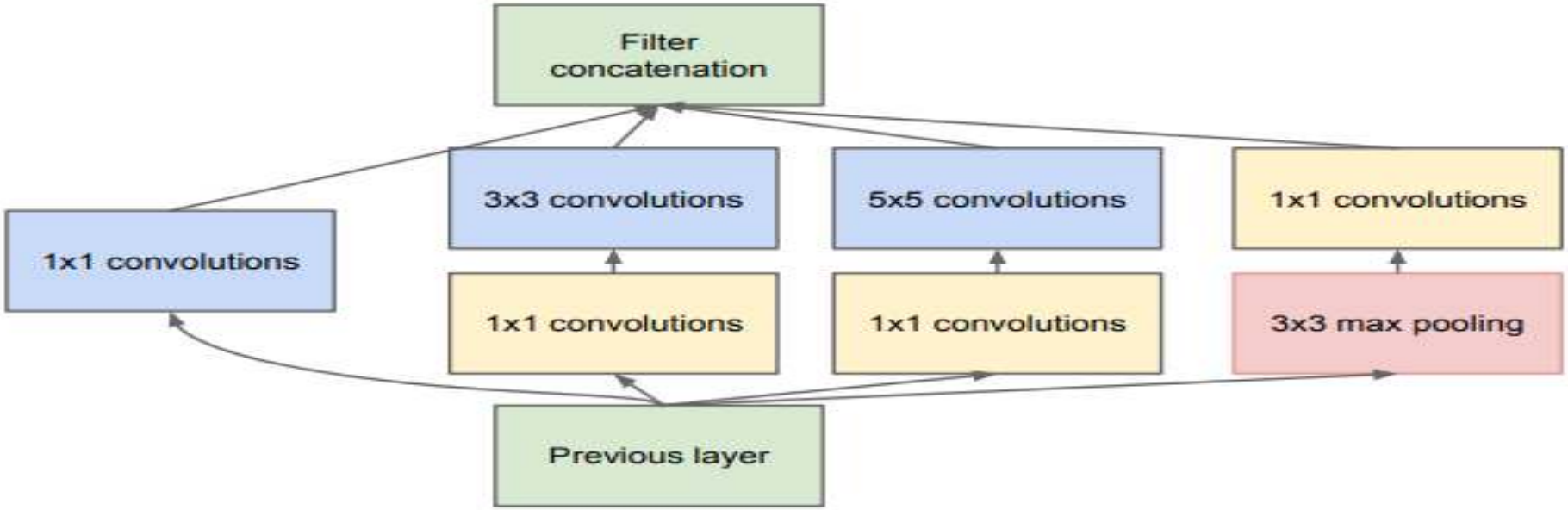
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Karen Simonyan, Andrew Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition

经典网络

GoogLeNet （2014）

Google在2014年提出，在ILSVRC比赛上获得冠军。采用Inception机制，即不同尺寸的卷积核的并行。构建密集的块结构来近似最优的稀疏结构，从而达到提高性能而又不大量增加计算量的目的

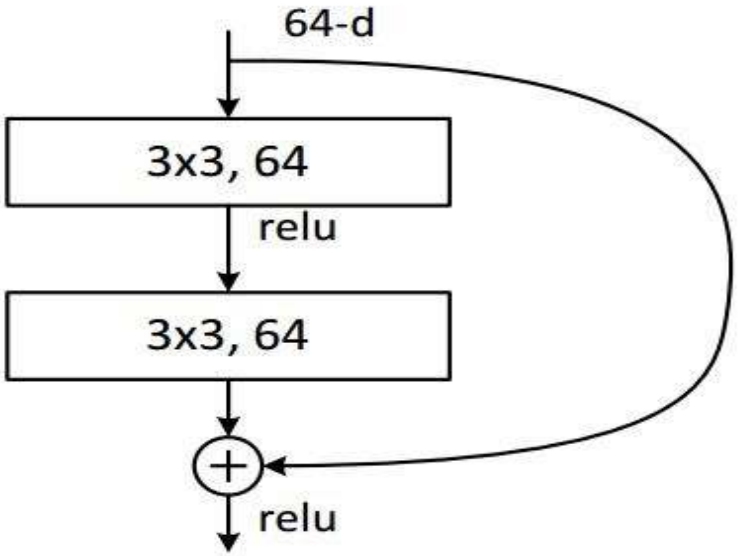


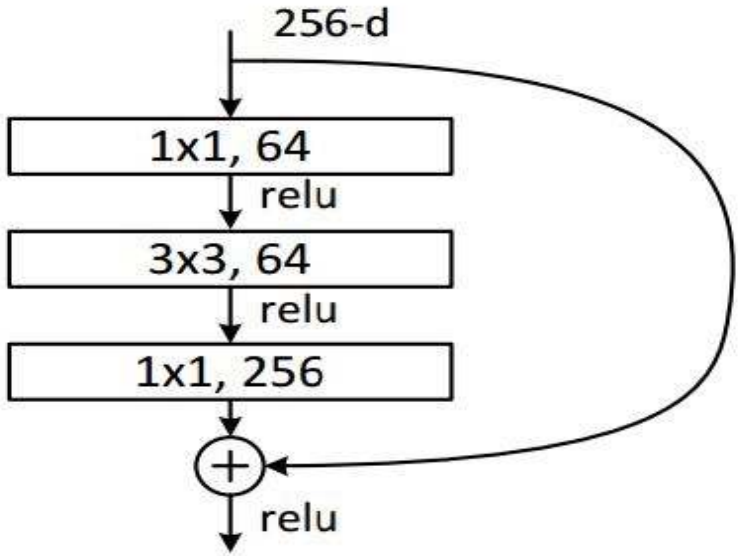
Christian Szegedy et al, Going Deeper with Convolutions, 2014

经典网络

- ResNet（2015）

ResNet 是2015 ILSVRC 的冠军网络，利用残差结构可以使网络层数更深

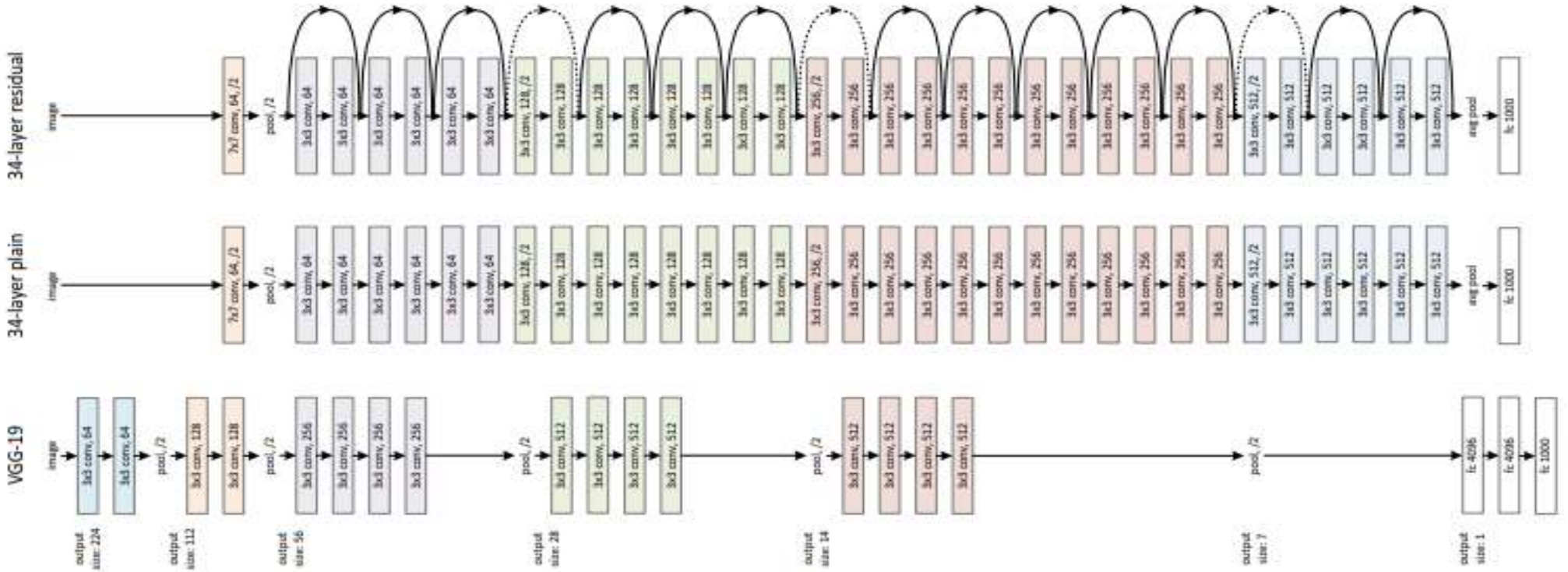




Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition

经典网络

- ResNet（2015）

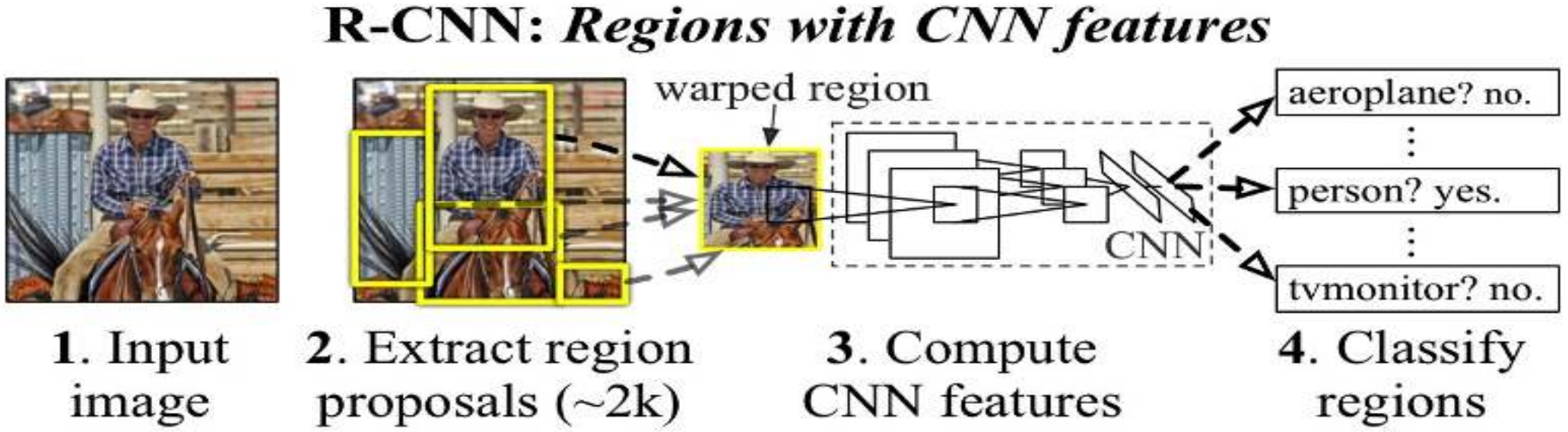


Paper : Deep Residual Learning for Image Recognition

经典网络

- R-CNN（目标检测）

R-CNN是将传统的CNN进行的改进，先提出检测的备选区域Region Proposal，然后，利用CNN去检测这些Region

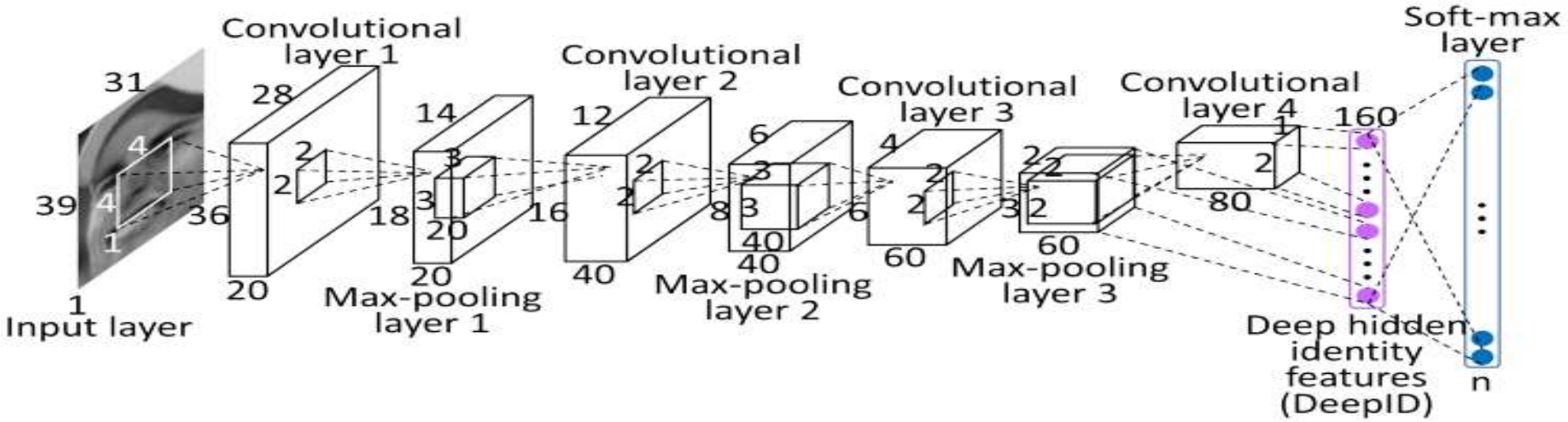


Jitendra Malik et al, Rich feature hierarchies for accurate object detection and semantic segmentation

经典网络

- DeepID（人脸验证）

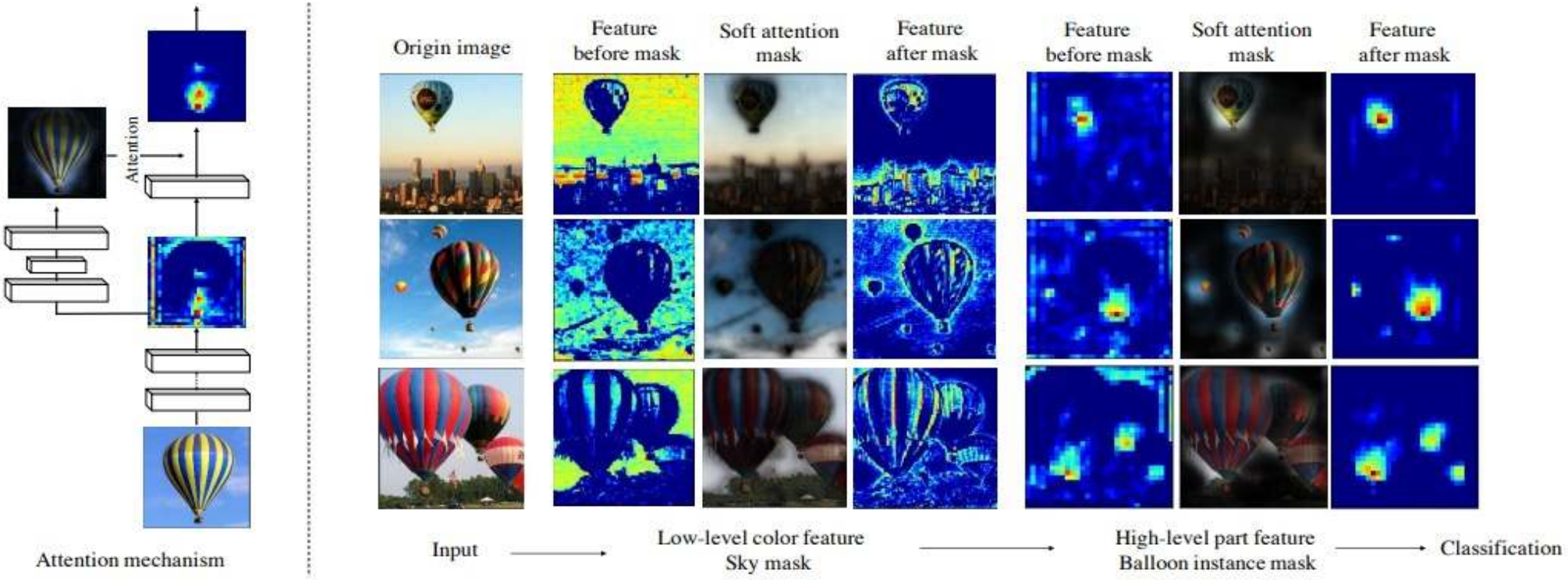
利用CNN提取特征（考虑局部的特征，又考虑全局的特征），推断两张图片是不是同一个人



Xiaoou Tang et al, Deep Learning Face Representation from Predicting 10,000 Classes

其它模型

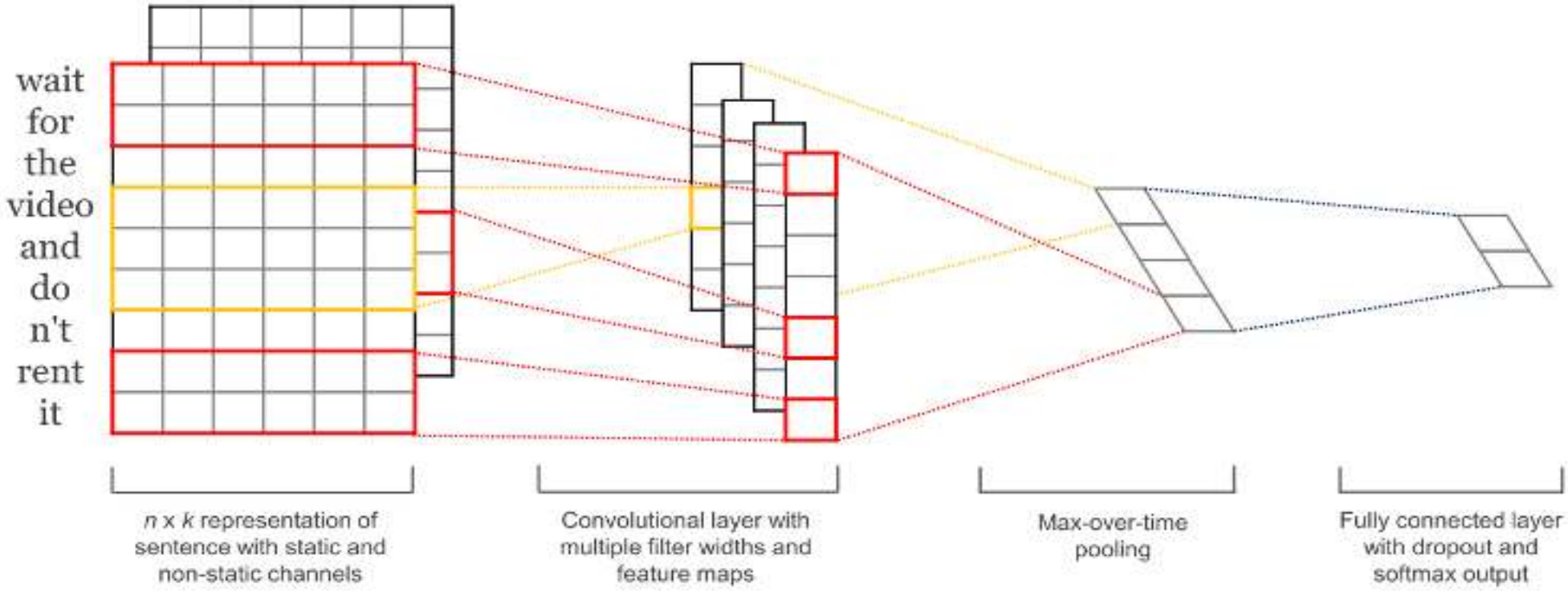
- Attention Model



Xiaoou Tang et al, Residual Attention Network for Image Classification

其它模型

- CNN+NLP



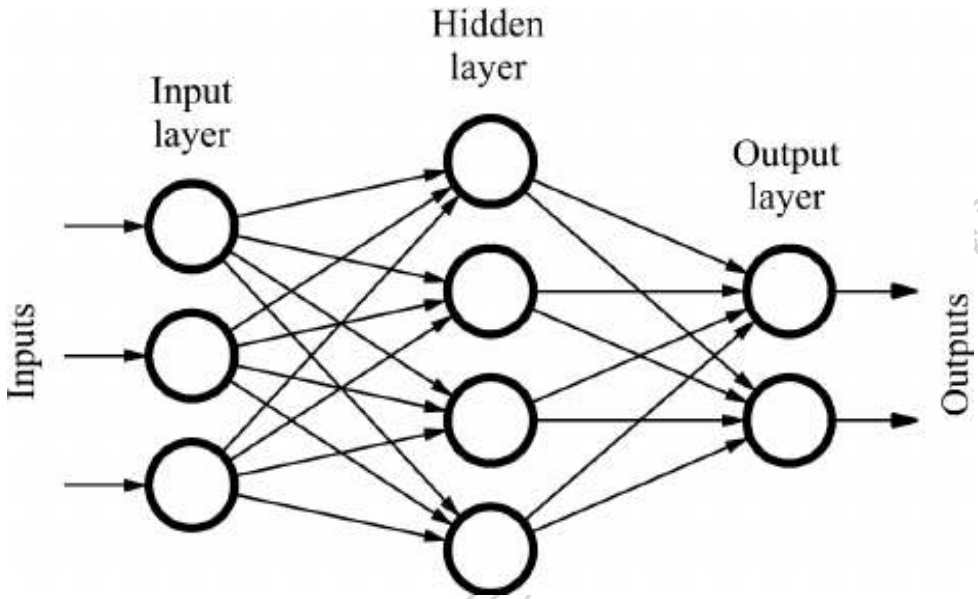
Paper : Convolutional neural networks for sentence classification

Demo

循环神经网络

前馈神经网络的不足

- 前馈神经网络，是一种在模型输出与模型本身之间没有反馈连接的神经网络



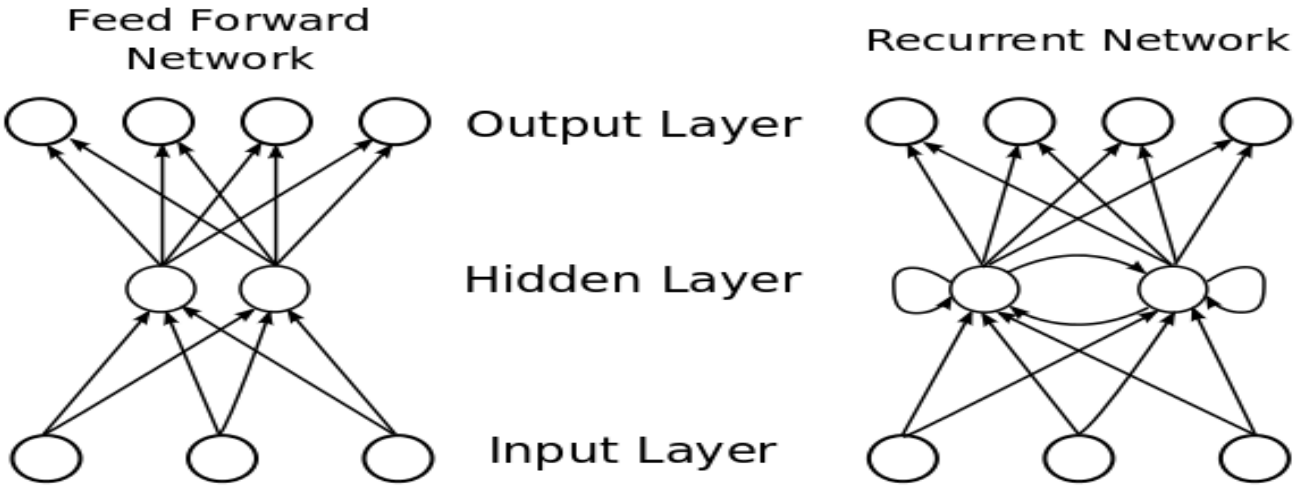
- **易处理网格数据，很难处理序列数据**
- **没有上下文关联能力和长期记忆能力**

1. 我喜欢吃苹果。
2. 苹果是一家高科技公司。

对“苹果”分类，要结合上下文背景关系

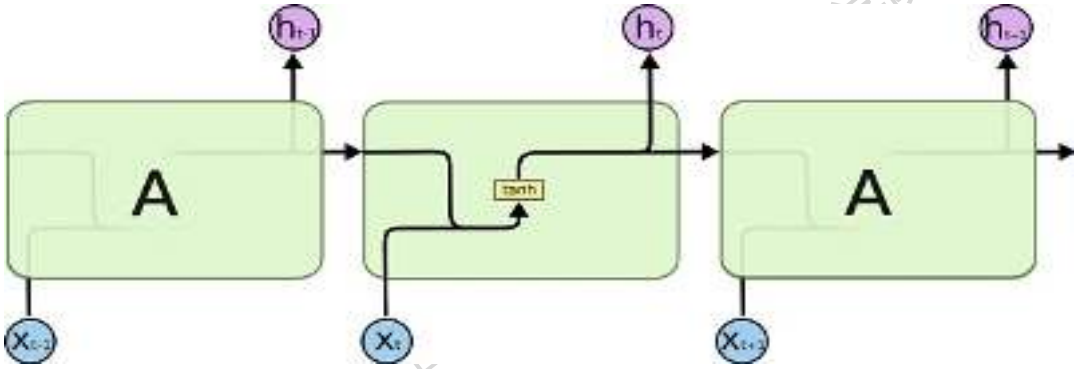
循环神经网络的基本概念

- 循环神经网络(Recurrent Neural Network)，是一种具有从后续层到前面层反馈连接或者同层之间神经元连接的神经网络，常用于处理顺序数据
- RNN 的核心思想在于引入循环连接，使得当前的计算不仅依赖于当前的输入，还依赖于前一时间步的内部状态。这种机制使模型具备“记忆” 历史信息的能力，并能够在理论上处理任意长度的输入序列
- 网络中具有环结构



简单循环神经网络模型—Elman 网络

- RNN每个时间点的网络拓扑结构相同，在任意t时间下，包含输入层、隐层、输出层。RNN的隐层的输出一分为二，一份传给输出层，一份与下一时刻外界输入一起作为隐层的输入。标准RNN的激活函数为一个tanh函数
- 隐层的输出不仅进入输出端，还进入了下一个时间步骤的隐层，所以它能够持续保留信息，能够根据之前状态推出后面的状态



隐藏状态：

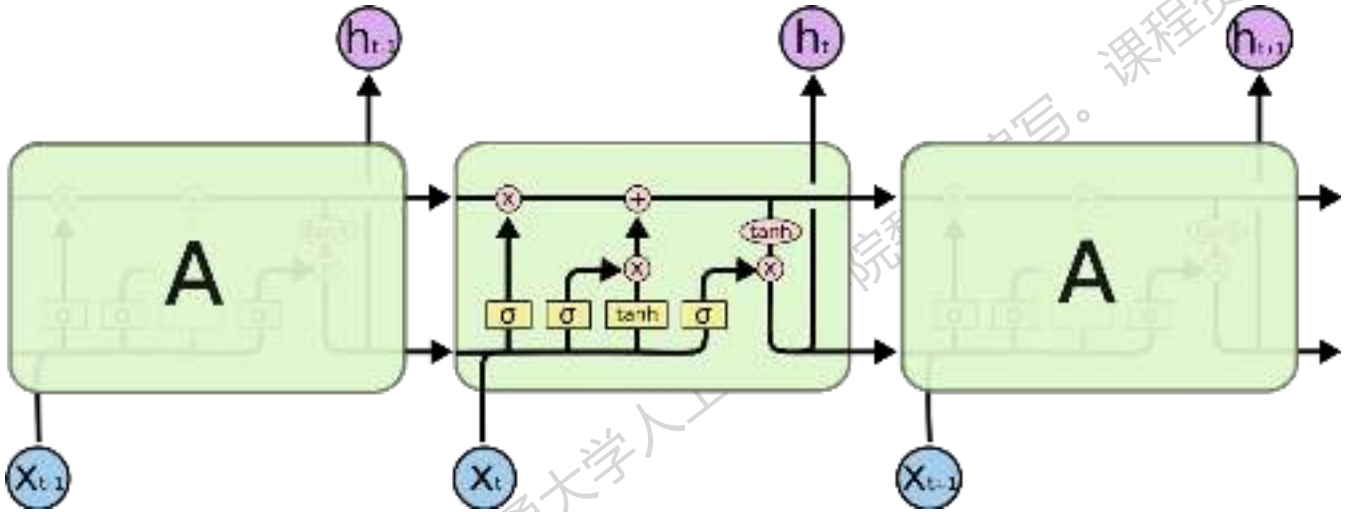
$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh}\boldsymbol{x}_t + \boldsymbol{b}_h)$$

输出：

$$\boldsymbol{y}_t = \boldsymbol{W}_{hy}\boldsymbol{h}_t + \boldsymbol{b}_y$$

长短记忆网络基本概念

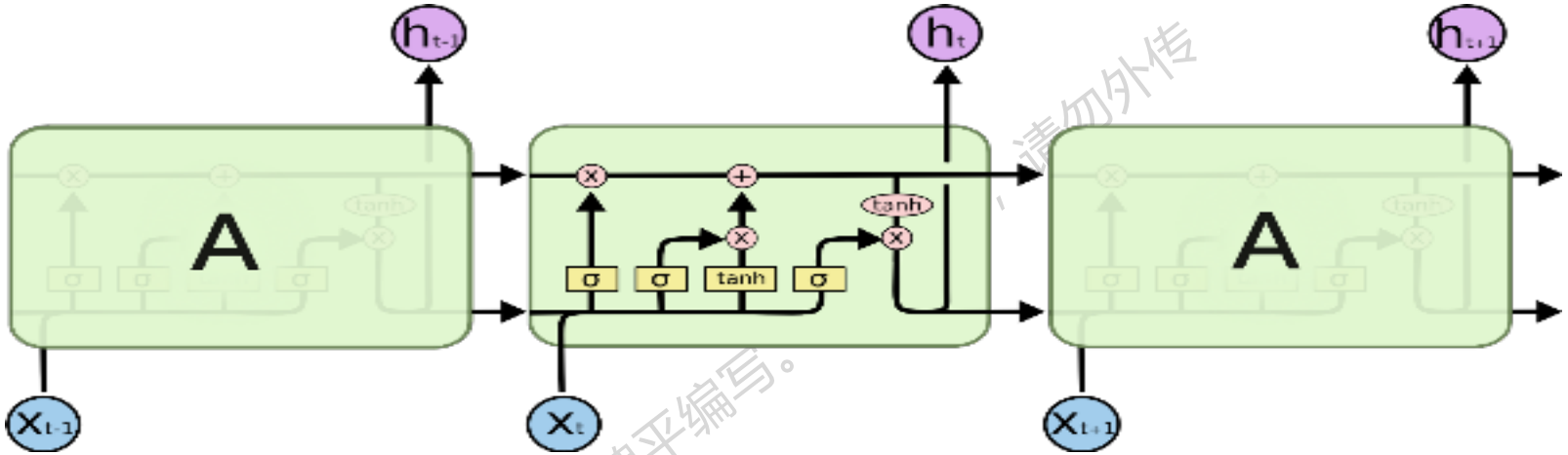
- LSTM（Long-Short Term Memory）是循环神经网络（RNN）的一种，适合处理和预测时间序列中间隔和延迟相对较长的重要事件



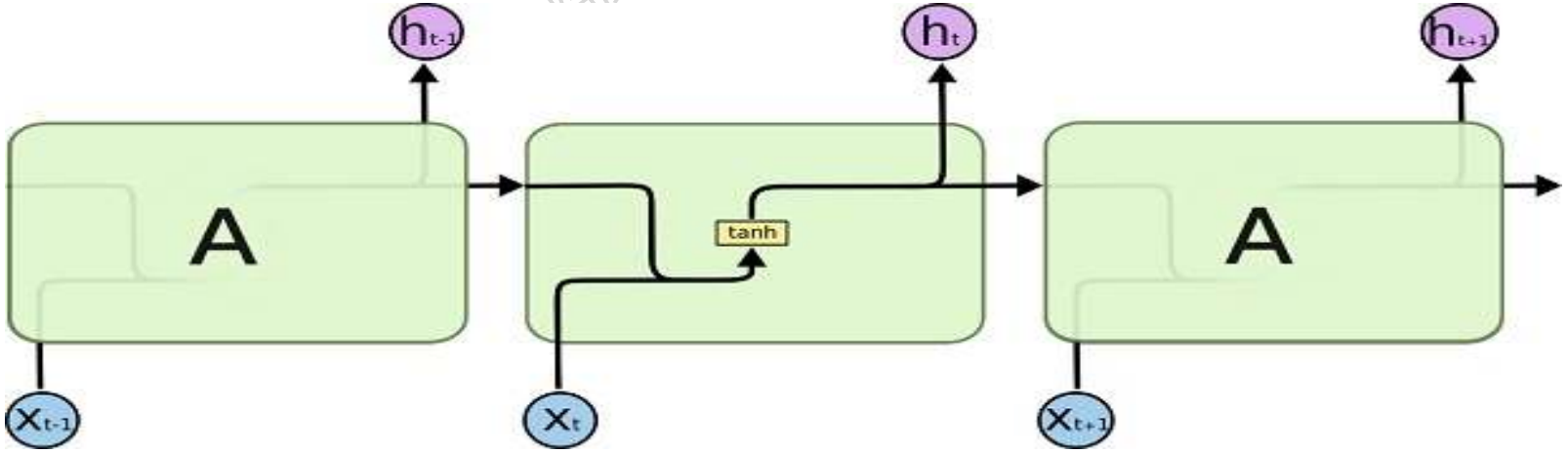
Jürgen Schmidhuber
瑞士达勒·莫尔人工智能研究所

长短记忆网络基本概念

LSTM



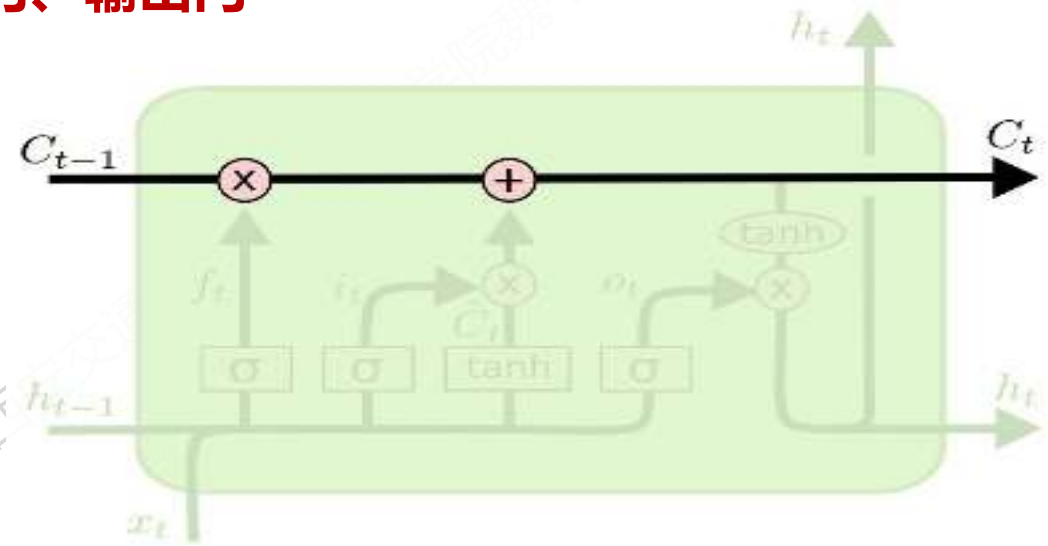
一般RNN



长短记忆网络单元结构

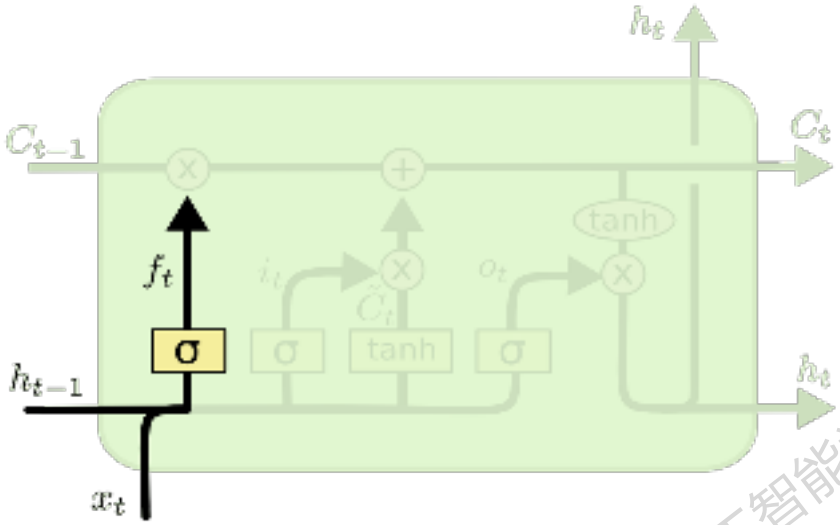
- LSTM 的核心是单元状态，即在图上方流动的水平线。单元状态类似于传送带，直接在整个链上运行，只有少量的线性交互，保证了信息在上面流动的稳定性

- LSTM 有通过精心设计的称作为“门”的结构来去除或者增加信息到单元状态的能力。门是一种让信息选择式通过的方法。LSTM 拥有三个门来保护和控制单元状态：**遗忘门、输入门、输出门**



长短记忆网络单元结构

遗忘门

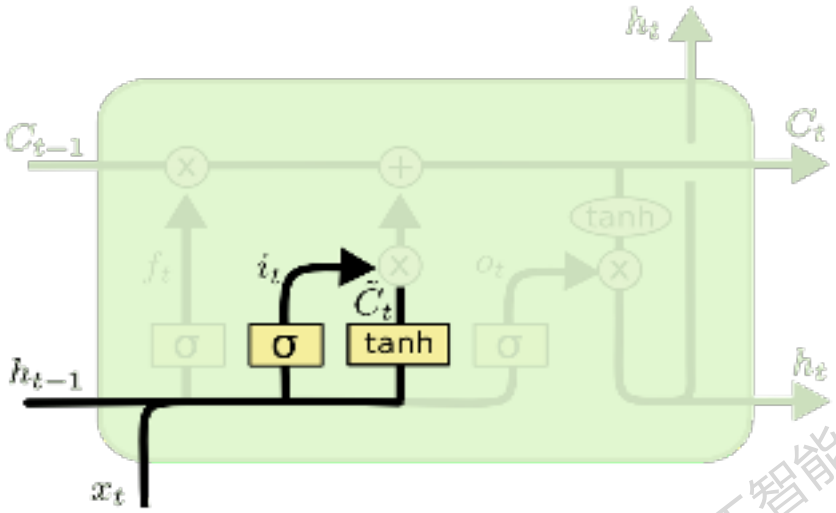


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

在LSTM 中的第一步是决定会从单元状态中丢弃什么信息，而这个决定通过一个称为遗忘门完成。该门会读取 h_{t-1} 和 x_t，经过σ函数处理，输出一个在 0 到 1 之间的数值f_t给每个在细胞状态 C_{t-1}中的数字。0 表示信息“完全舍弃”，而1 表示“完全保留”

长短记忆网络单元结构

输入门



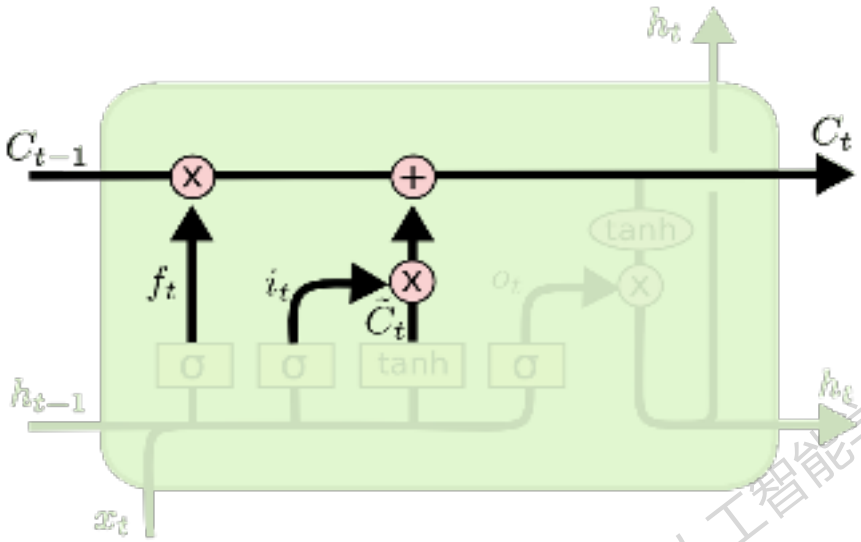
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

该步确定什么样的新信息被存放在单元状态中。这里包含两个部分，第一，σ层称 “输入门层”，决定什么值将要更新；同时，一个 tanh 层创建一个新的候选值向量$\tilde{C}_t$，被加入到状态中。

长短记忆网络单元结构

状态更新

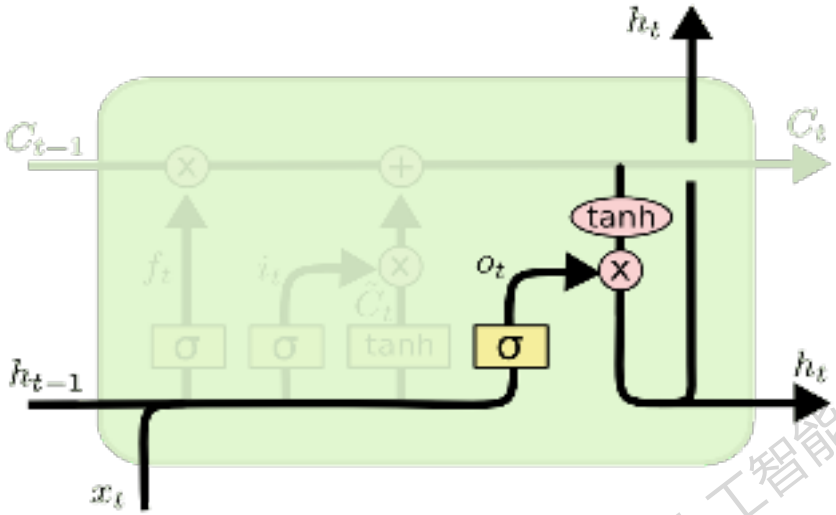


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

现在可以对旧单元状态进行更新了，即C_{t-1}更新为 C_t。将t-1时期的单元状态C_{t-1}与f_t相乘，丢弃掉确定需要丢弃的信息，然后加上 i_t与 $\tilde{C}_t$的乘积，就得到了t时期的单元状态C_t

长短记忆网络单元结构

信息输出

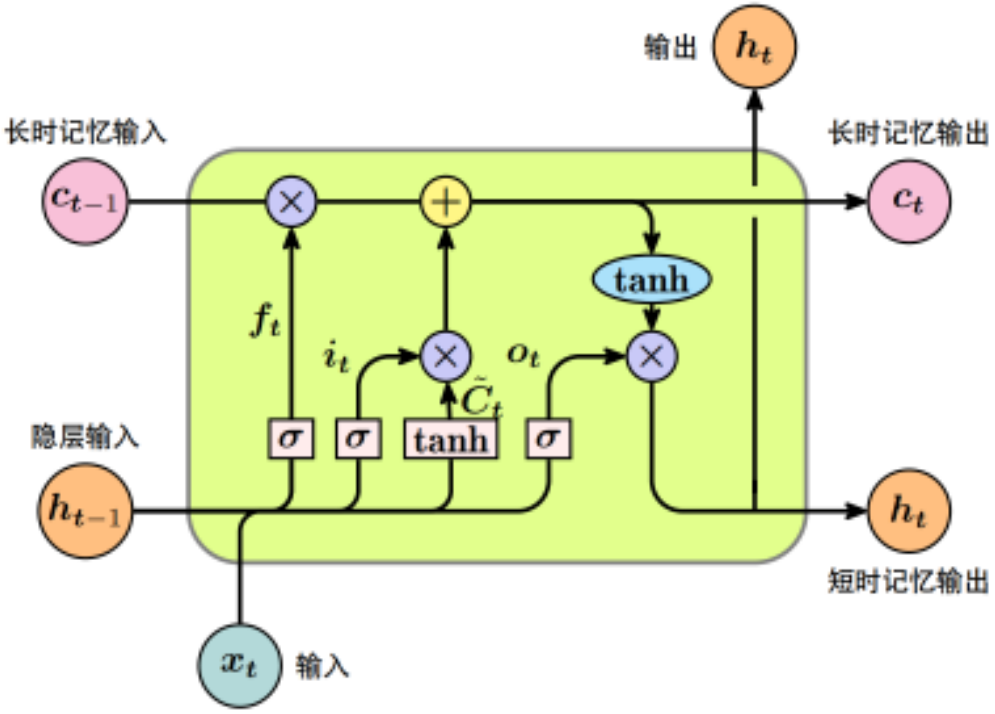


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

首先，读取 h_{t-1} 和 x_t，经过σ函数处理，输出一个在 0 到 1 之间的数值o_t给经过tanh函数的每个在当前单元状态 C_t中的数字，即将o_t与tanh(C_t)的乘积作为输出信息h_t

长短记忆网络整体结构



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$

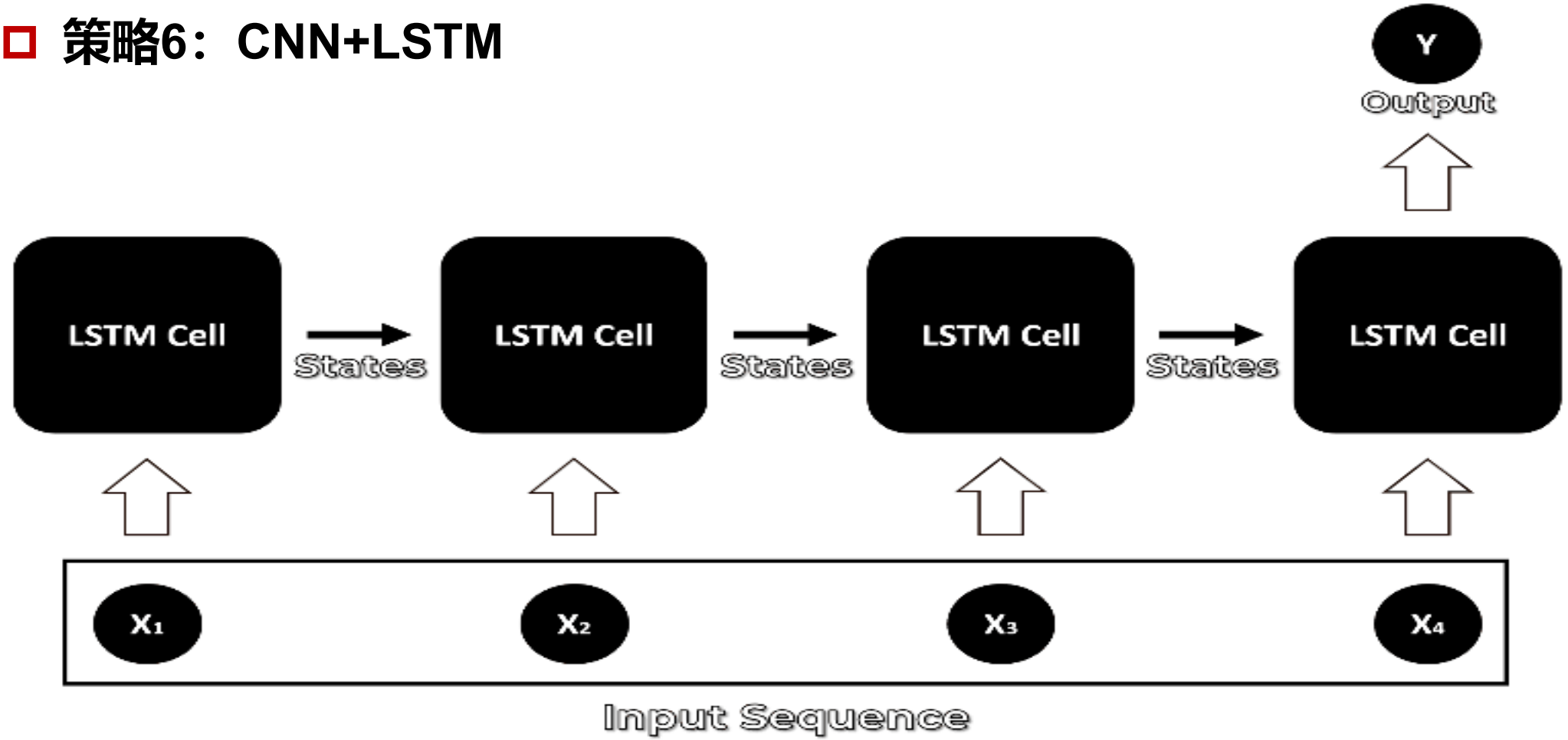
$$h_t = o_t * \tanh\left(C_t\right)$$

例: Action Recognition with LSTM

Backflipping

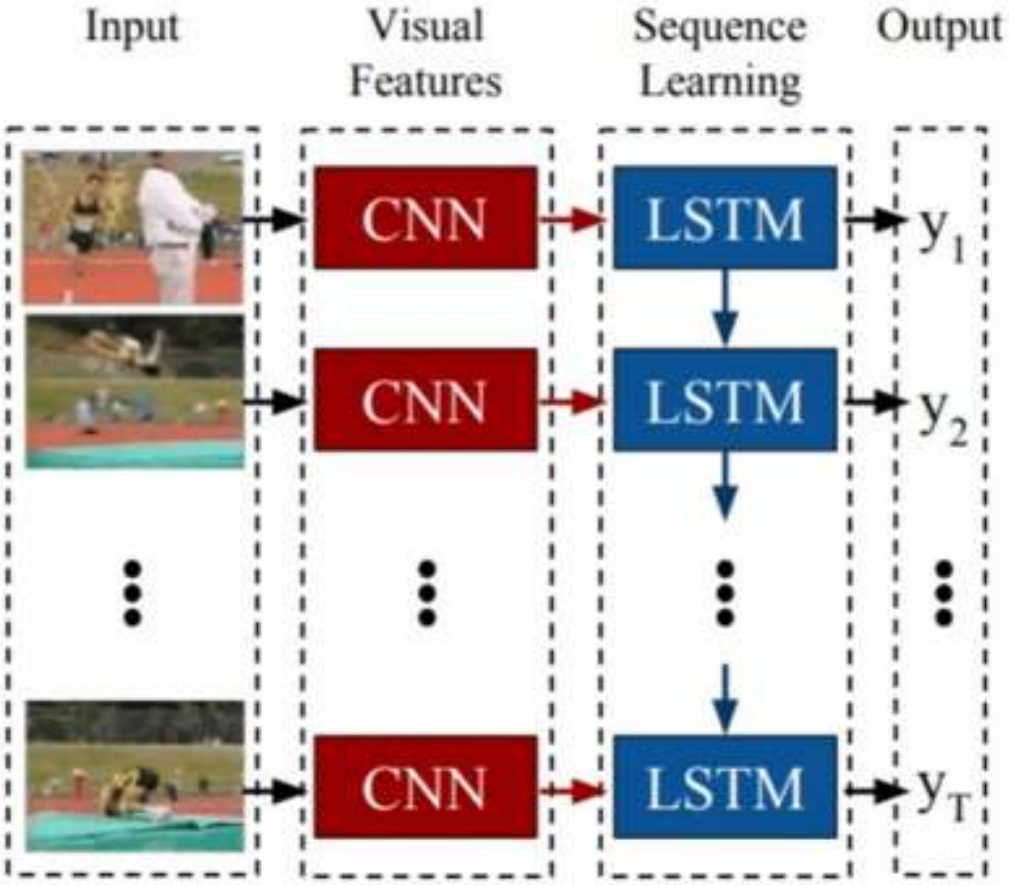
例: Action Recognition with LSTM

- 策略6：CNN+LSTM



例: Action Recognition with LSTM

策略6：CNN+LSTM



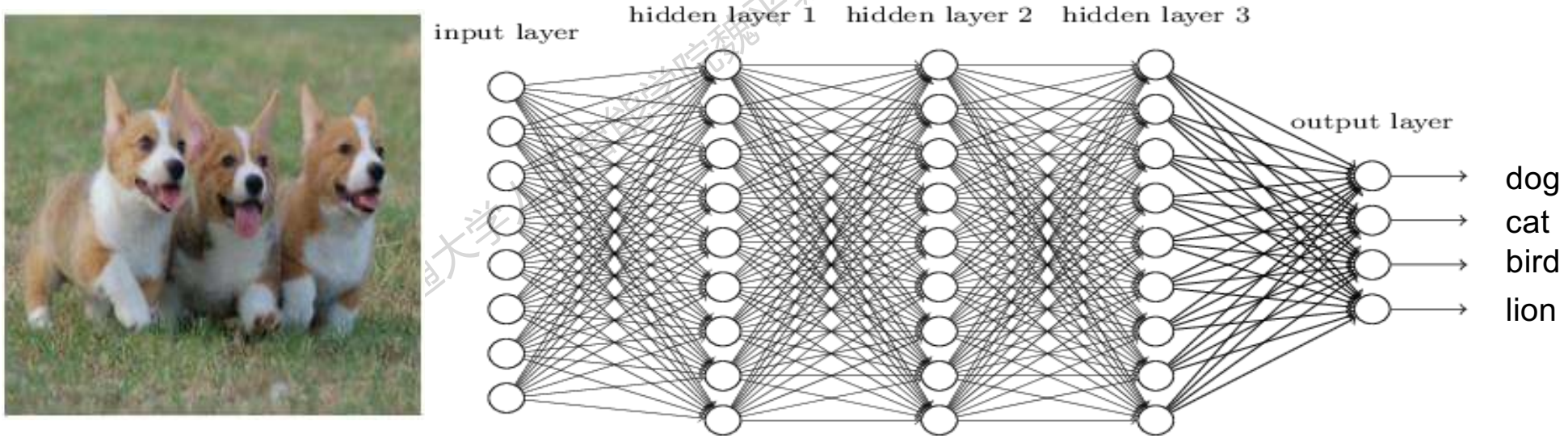
- 输入视频预处理为固定长度，如40帧
- 利用CNN提取每一帧特征，得到特征序列，40×25088
- 特征向量序列输入LSTM单元，输出分类结果

CONTENTS

- 神经网络基本概念
- 典型神经网络
- 深度学习与反向传播算法

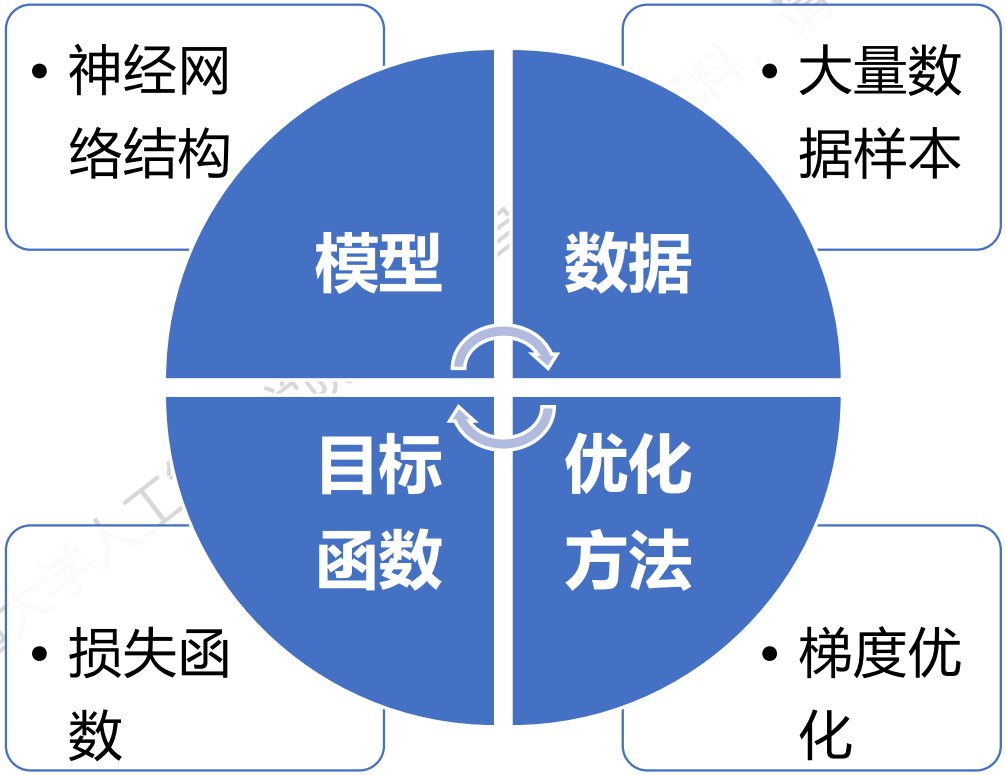
深度学习

- 神经网络由一定的结构定义了一个函数$y = f(\boldsymbol{x}; \theta)$，神经网络学习的目标是从数据中训练得到参数集合$\theta$，例如，前向神经网络神经元的基本操作为加权求和，因此学习的目标一般是从数据中训练得到网络权重集合$\boldsymbol{W}$
- 深度学习也即多层神经网络的学习，是机器学习的一个重要分支，其核心是通过多层神经网络模拟人类大脑的层级化信息处理方式，从数据中自动学习特征表示



深度学习

深度学习的要素



代价函数

- 代价函数（cost function）定义了神经网络所要达到的任务目标，反应人们使用神经网络的目的意图，也称作损失函数(loss function)

- 大多数应用模型可以抽象为一个条件分布$\mathrm{p}(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta})$，模型的代价函数由最大似然估计准则得到

- Minimizing the negative log-likelihood
- Minimizing the cross entropy

$$\boldsymbol{J}(\boldsymbol{\theta}) = -E_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{y}|\boldsymbol{x})$$

代价函数

- 利用函数$f(\boldsymbol{x};\boldsymbol{\theta})$从输入$\boldsymbol{x}$预测$\boldsymbol{y}$的模型，代价函数定义为：

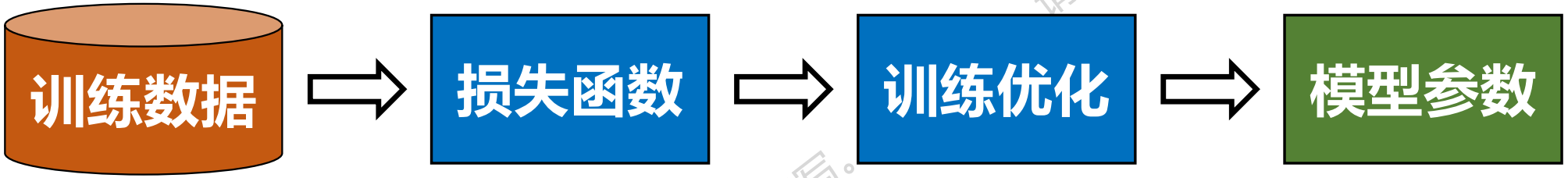
- 范数代价函数

$$f^* = \arg\min_f \; E_{\boldsymbol{x},\boldsymbol{y}\sim p_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x})\|^2$$

- 范数代价函数-平均绝对误差

$$f^* = \arg\min_f \; E_{\boldsymbol{x},\boldsymbol{y}\sim p_{\text{data}}} \|\boldsymbol{y} - f(\boldsymbol{x})\|$$

神经网络的训练过程



- 归一化
- 标准化
- 数据清洗

- 均方误差
- 交叉熵损失
- 约束损失

- 前向传播
- 反向传播
- 梯度下降

- 训练集学习
- 验证集调优
- 测试集测试

优化算法

- 优化算法的目标是通过迭代更新网络参数，最小化损失函数，从而提高模型对数据的拟合能力和泛化能力
- 神经网络定义了函数$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$，给定训练数据集$\boldsymbol{D} = \{(\boldsymbol{x}_i, \boldsymbol{t}_i) | i = 1, \dots, N\}$，学习的目标是从数据中训练得到参数集合$\boldsymbol{\theta}$
- 化算法的目标是改变$\boldsymbol{\theta}$以最小化损失函数$\mathcal{L}(\boldsymbol{t}_i, \boldsymbol{y}_i(\boldsymbol{\theta}))$，

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{t}_i, \boldsymbol{y}_i(\boldsymbol{\theta}))$$

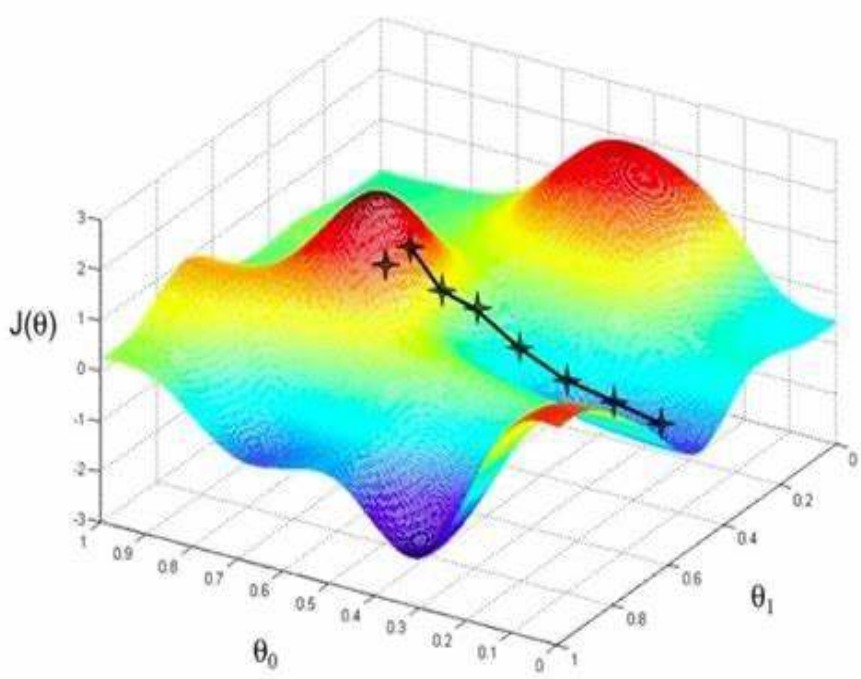
- 理想的优化算法应具备以下特点
  - **最小化训练损失**: 提高模型对训练数据的拟合能力
  - **收敛速度快**: 减少训练时间，提高效率
  - **训练稳定性强**: 避免梯度消失或梯度爆炸等问题导致的训练失败
  - **泛化性能好**: 确保模型在测试数据上的性能优异

梯度下降法

- 梯度（又叫梯度向量）的方向是函数值上升最快的方向
- 梯度下降法的思想是沿着损失函数梯度负方向更新模型参数，使损失函数值逐步减小

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$$

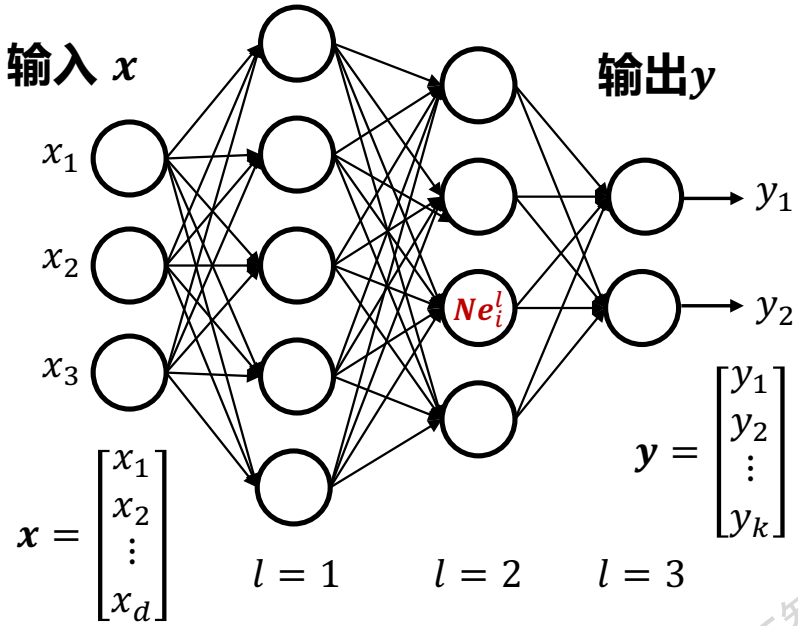
η：学习率 (learning rate)



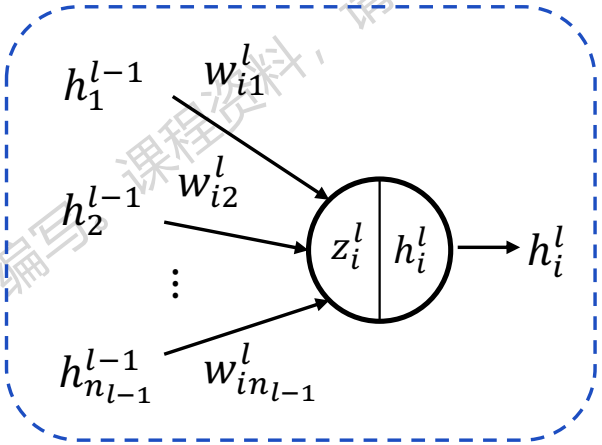
三种梯度下降法

算法	计算公式	更新样本量	特点
全批量梯度下降	$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{t}_i)$	全部样本	• 更新方向稳定 • 计算复杂度高
随机梯度下降	$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{t}_i)$	单个样本	• 计算复杂度低 • 梯度不稳定
小批量梯度下降	$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \boldsymbol{x}_i, \boldsymbol{t}_i)$	部分样本	• 计算更高效 • 收敛更稳定 • 样本更灵活

多层感知机前向传播



- 第l层第i个神经元 $\boldsymbol{Ne}_i^l$

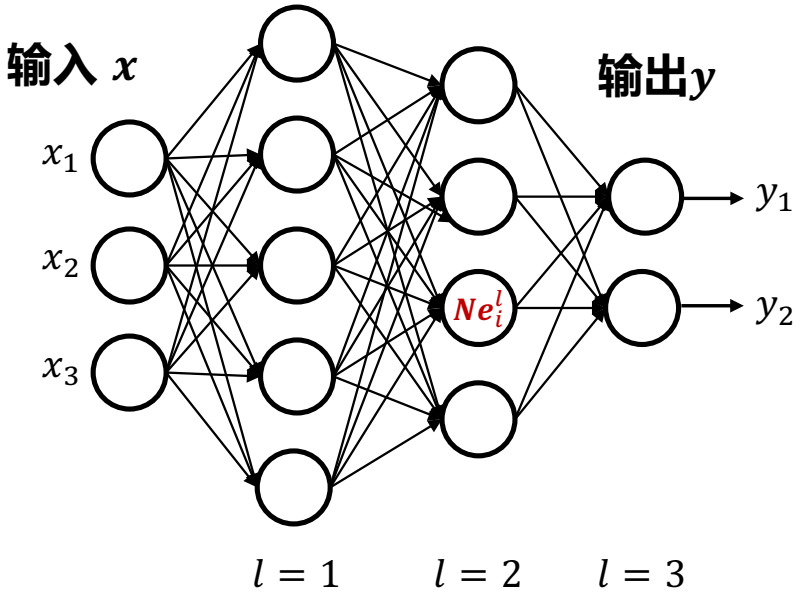


$$z_i^l = \sum_{j=1}^{n_{l-1}} w_{ij}^l h_j^{l-1} + b_i^l$$

$$h_i^l = g^l(z_i^l)$$

- **变量定义**：$l = 1, \dots, L$是层序号，$\boldsymbol{h}^l \in \mathbb{R}^{n_l \times 1}$是第$l$层输出，$\boldsymbol{h}^0 = \boldsymbol{x} \in \mathbb{R}^{d \times 1}$是输入向量，$\boldsymbol{y} \in \mathbb{R}^{k \times 1}$是输出向量，$\boldsymbol{W}^l \in \mathbb{R}^{n_l \times n_{l-1}}$是第$l$层的权重矩阵，$w_{ij}^l$是第$l$-1层的第$j$个神经元到到第$l$层的第$i$个神经元的权重，$\boldsymbol{b}^l \in \mathbb{R}^{n_l \times 1}$是第$l$层的偏置向量，$g^l(\cdot)$是第$l$层激活函数

多层感知机前向传播



$$\boldsymbol{h}^l = \begin{bmatrix} h_1^l \\ h_2^l \\ \vdots \\ h_{n_l}^l \end{bmatrix} \quad \boldsymbol{h}^{l-1} = \begin{bmatrix} h_1^{l-1} \\ h_2^{l-1} \\ \vdots \\ h_{n_{l-1}}^{l-1} \end{bmatrix} \quad \boldsymbol{b}^l = \begin{bmatrix} b_1^l \\ b_2^l \\ \vdots \\ b_{n_l}^l \end{bmatrix}$$

$$\boldsymbol{z}^l = \begin{bmatrix} z_1^l \\ z_2^l \\ \vdots \\ z_{n_l}^l \end{bmatrix} \quad \boldsymbol{W}^l = \begin{bmatrix} w_{11}^l & \cdots & w_{1n_{l-1}}^l \\ \vdots & \ddots & \vdots \\ w_{n_l1}^l & \cdots & w_{n_ln_{l-1}}^l \end{bmatrix}$$

前向传播计算

$$\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{h}^{l-1} + \boldsymbol{b}^l$$

$$\boldsymbol{h}^l = g^l(\boldsymbol{z}^l)$$

$$\boldsymbol{h}^l = g^l(\boldsymbol{W}^l \boldsymbol{h}^{l-1} + \boldsymbol{b}^l)$$

$$\left.\begin{aligned} \boldsymbol{h}^1 &= g^1(\boldsymbol{W}^1 \boldsymbol{x} + \boldsymbol{b}^1) \\ \boldsymbol{h}^2 &= g^2(\boldsymbol{W}^2 \boldsymbol{h}^1 + \boldsymbol{b}^2) \\ &\vdots \\ \boldsymbol{h}^L &= g^L(\boldsymbol{W}^L \boldsymbol{h}^{L-1} + \boldsymbol{b}^L) \end{aligned}\right\} \quad \boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$$

$$\boldsymbol{\theta} = \{\boldsymbol{W}^l, \boldsymbol{b}^l | l = 1, \ldots, L\}$$

反向传播算法 (Back Propagation)

- 反向传播(Backpropagation，BP) 算法是训练神经网络的核心算法，也是深度学习的基石。BP 算法的基本思想是: 据网络的预测误差, 从输出层开始, 逐层反向计算每个神经元对误差的贡献, 并据此调整网络的权重和偏置, 使得预测误差最小化

- 反向传播算法的形成是一系列科学家不断探索研究推动的结果

  - David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams . Learning representations by back-propagating errors. *Nature,* **323**, 533–536, 1986

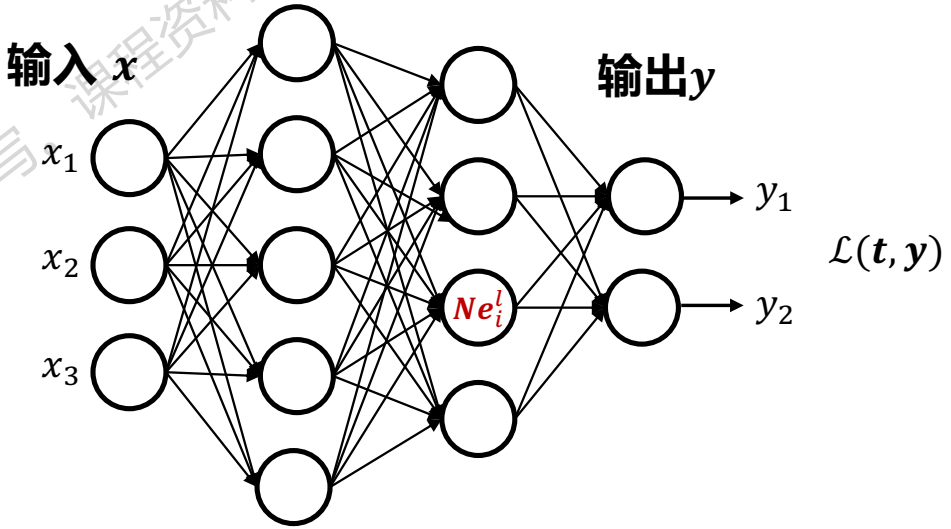
Geoffrey E. Hinton

反向传播算法基本原理

- L层的前馈神经网络，输入为 $\boldsymbol{x}$，网络预测输出为$\boldsymbol{y}$， $\boldsymbol{t}$为输出真值，$\mathcal{L}(\boldsymbol{t},\boldsymbol{y})$ 为损失函数。BP 算法的目标是计算损失函数关于每一层权重$\boldsymbol{W}^l$和偏置$\boldsymbol{b}^l$的梯度$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l}$和$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l}$

$$\boldsymbol{W}^l \leftarrow \boldsymbol{W}^l - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l}$$

$$\boldsymbol{b}^l \leftarrow \boldsymbol{b}^l - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l}$$



导数规则

求导类型

	标量 x，1×1	**向量** $\mathbf{x}$，$n\times1$	**矩阵** $\mathbf{X}$，$n\times k$
标量 y，1×1	$\frac{\partial y}{\partial x}, 1\times1$	$\frac{\partial y}{\partial \mathbf{x}}, 1\times n$	$\frac{\partial y}{\partial \mathbf{X}}, k\times n$
向量 $\mathbf{y}$，$m\times1$	$\frac{\partial \mathbf{y}}{\partial x}, m\times1$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}, m\times n$	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}, m\times k\times n$
矩阵 $\mathbf{Y}$，$m\times l$	$\frac{\partial \mathbf{Y}}{\partial x}, m\times l$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}, m\times l\times n$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}, m\times l\times k\times n$

链式法则

- $y=f(u),\ u=g(x) \Rightarrow \frac{dy}{dx}=\frac{dy}{du}\frac{du}{dx}$
- $y=f(u,v),\ u=g(x),\ v=h(x) \Rightarrow \frac{dy}{dx}=\frac{dy}{du}\frac{du}{dx}+\frac{dy}{dv}\frac{dv}{dx}$

反向传播算法基本原理

- BP 算法的核心是利用链式法则，递归地计算每一层的梯度

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l}\frac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{W}^l} = \delta^l \boldsymbol{h}^{l-1\mathrm{T}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l}\frac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{b}^l} = \delta^l$$

误差项： $\delta^l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l}$

$$\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{h}^{l-1} + \boldsymbol{b}^l \qquad \boldsymbol{h}^l = g^l(\boldsymbol{z}^l)$$

$$\boldsymbol{z}^l = \begin{bmatrix} z_1^l \\ z_2^l \\ \vdots \\ z_{n_l}^l \end{bmatrix} \quad \boldsymbol{W}^l = \begin{bmatrix} w_{11}^l & \cdots & w_{1n_{l-1}}^l \\ \vdots & \ddots & \vdots \\ w_{n_l 1}^l & \cdots & w_{n_l n_{l-1}}^l \end{bmatrix}$$

反向传播算法基本原理

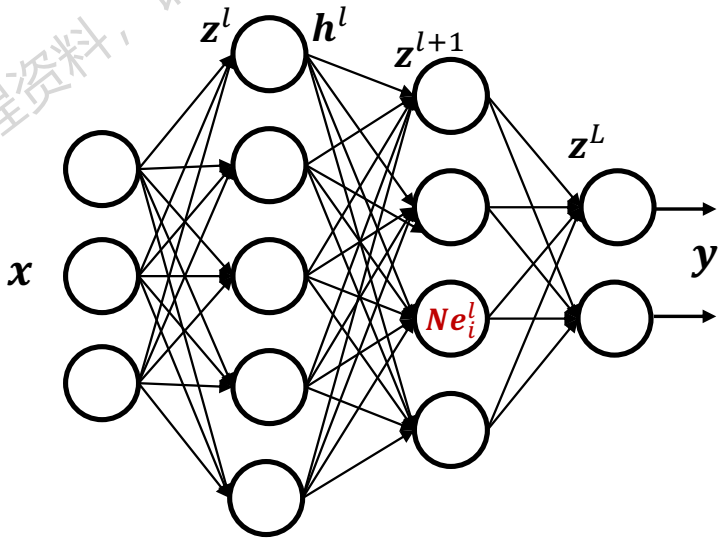
- BP 算法的核心是利用链式法则，递归地计算每一层的梯度

$$\delta^L = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^L} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}^L} = \nabla_{\boldsymbol{y}} \mathcal{L} \odot g^{L\prime}(\boldsymbol{z}^L)$$

$$\delta^l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{l+1}} \frac{\partial \boldsymbol{z}^{l+1}}{\partial \boldsymbol{h}^l} \frac{\partial \boldsymbol{h}^l}{\partial \boldsymbol{z}^l}$$

$$= (\boldsymbol{W}^{l+1\mathrm{T}} \delta^{l+1}) \odot g^{l\prime}(\boldsymbol{z}^l)$$

⊙：对应元素相乘



$$\boldsymbol{z}^{l+1} = \boldsymbol{W}^{l+1} \boldsymbol{h}^l + \boldsymbol{b}^{l+1}$$

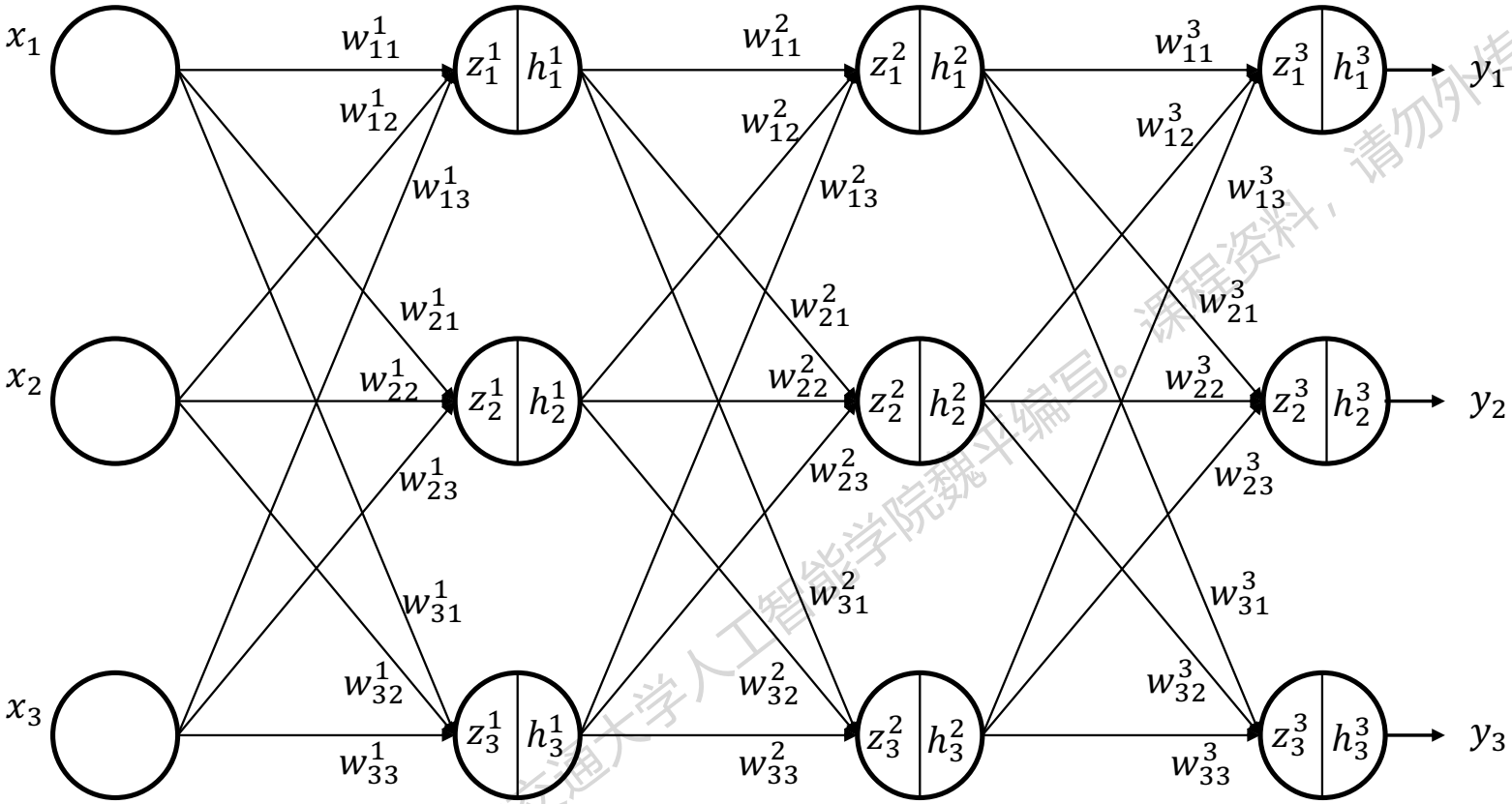
$$\boldsymbol{h}^l = g^l(\boldsymbol{z}^l) \qquad \boldsymbol{y} = g^L(\boldsymbol{z}^L)$$

利用反向传播算法进行参数学习流程

- **前向传播:** 根据输入计算每一层的激活值, 直到输出层
- **计算输出层误差项：** $\delta^L = \nabla_{\boldsymbol{y}}\mathcal{L} \odot g^{L\prime}(\boldsymbol{z}^L)$
- **反向传播误差项：** 从倒数第二层开始，递归计算每一层的误差项$\delta^l = (\boldsymbol{W}^{l+1^{\mathrm{T}}}\delta^{l+1}) \odot g^{l\prime}(\boldsymbol{z}^l)$，直到第一层
- **计算梯度:** 根据每一层的误差项和激活值, 计算损失函数关于权重和偏置的梯度$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l} = \delta^l \boldsymbol{h}^{l-1^{\mathrm{T}}}$，$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l} = \delta^l$
- $\boldsymbol{W}^l \leftarrow \boldsymbol{W}^l - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l}$，$\boldsymbol{b}^l \leftarrow \boldsymbol{b}^l - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^l}$

- 上述推导是基于单个样本的，在实际训练中，通常使用一个小批量(mini-batch) 的样本来计算平均梯度, 这样可以利用矩阵运算加速计算, 也有助于梯度估计的稳定性
- 上述推导$\mathcal{L}$是标量，是向量时原理一致，形式不同

例子



$$\boldsymbol{W}^1 = \begin{bmatrix} 0.1 & 0.3 & 0.4 \\ 0.2 & 0.2 & 0.3 \\ 0.3 & 0.7 & 0.9 \end{bmatrix}$$

$$\boldsymbol{W}^2 = \begin{bmatrix} 0.2 & 0.3 & 0.6 \\ 0.3 & 0.5 & 0.4 \\ 0.5 & 0.7 & 0.8 \end{bmatrix}$$

$$\boldsymbol{W}^3 = \begin{bmatrix} 0.1 & 0.3 & 0.5 \\ 0.4 & 0.7 & 0.2 \\ 0.8 & 0.2 & 0.9 \end{bmatrix}$$

$\boldsymbol{x} = [0.1 \quad 0.2 \quad 0.7]^{\mathrm{T}}$

$\boldsymbol{t} = [1.0 \quad 0.0 \quad 0.0]^{\mathrm{T}}$

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

$b_i^1 = 1$

$\sigma(x) = \dfrac{1}{1 + e^{-x}}$

$b_i^2 = 1$

$\sigma(x) = \dfrac{1}{1 + e^{-x}}$

$b_i^3 = 1$

例子 — 前向传播

$$\boldsymbol{z}^1 = \boldsymbol{W}^1\boldsymbol{x} + \boldsymbol{b}^1$$

$$= \begin{bmatrix} 0.1 & 0.3 & 0.4 \\ 0.2 & 0.2 & 0.3 \\ 0.3 & 0.7 & 0.9 \end{bmatrix} \begin{bmatrix} 0.1 \\ 0.2 \\ 0.7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= [1.35 \;\; 1.27 \;\; 1.80]^{\mathrm{T}}$$

$$\boldsymbol{h}^1 = g^1(\boldsymbol{z}^1) = [1.35 \;\; 1.27 \;\; 1.80]^{\mathrm{T}}$$

$$g^1(\boldsymbol{z}^1) = ReLU(\boldsymbol{z}^1) = \max(0, \boldsymbol{z}^1)$$

$$\boldsymbol{z}^2 = \boldsymbol{W}^2\boldsymbol{h}^1 + \boldsymbol{b}^2$$

$$= \begin{bmatrix} 0.2 & 0.3 & 0.6 \\ 0.3 & 0.5 & 0.4 \\ 0.5 & 0.7 & 0.8 \end{bmatrix} \begin{bmatrix} 1.35 \\ 1.27 \\ 1.80 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= [2.731 \;\; 2.76 \;\; 4.004]^{\mathrm{T}}$$

$$\boldsymbol{h}^2 = g^2(\boldsymbol{z}^2) = [0.939 \;\; 0.94 \;\; 0.982]^{\mathrm{T}}$$

$$g^2(\boldsymbol{z}^2) = \sigma(\boldsymbol{z}^2) = \frac{1}{1 + e^{-\boldsymbol{z}^2}}$$

$$\boldsymbol{z}^3 = \boldsymbol{W}^3\boldsymbol{h}^2 + \boldsymbol{b}^3$$

$$= \begin{bmatrix} 0.1 & 0.3 & 0.5 \\ 0.4 & 0.7 & 0.2 \\ 0.8 & 0.2 & 0.9 \end{bmatrix} \begin{bmatrix} 0.939 \\ 0.94 \\ 0.982 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= [1.867 \;\; 2.23 \;\; 2.823]^{\mathrm{T}}$$

$$\boldsymbol{h}^3 = g^3(\boldsymbol{z}^3) = [0.866 \;\; 0.903 \;\; 0.944]^{\mathrm{T}}$$

$$\boldsymbol{y} = \boldsymbol{h}^3 \; [0.866 \;\; 0.903 \;\; 0.944]^{\mathrm{T}}$$

$$\boldsymbol{t} = [1.0 \;\; 0.0 \;\; 0.0]^{\mathrm{T}}$$

$$g^3(\boldsymbol{z}^3) = \sigma(\boldsymbol{z}^3) = \frac{1}{1 + e^{-\boldsymbol{z}^3}}$$

例子 — 反向传播

- 损失函数 $\mathcal{L}(\boldsymbol{t},\boldsymbol{y})=\frac{1}{2}\|\boldsymbol{y}-\boldsymbol{t}\|^2$

$$\nabla_{\boldsymbol{y}}\mathcal{L}=\begin{bmatrix}y_1-t_1\\y_2-t_2\\y_3-t_3\end{bmatrix}=\begin{bmatrix}-0.134\\0.903\\0.944\end{bmatrix}$$

$$g^{3'}(\boldsymbol{z}^3)=\begin{bmatrix}\sigma(z_1^3)(1-\sigma(z_1^3))\\\sigma(z_2^3)(1-\sigma(z_2^3))\\\sigma(z_3^3)(1-\sigma(z_3^3))\end{bmatrix}=\begin{bmatrix}0.116\\0.088\\0.053\end{bmatrix}$$

$$g^{2'}(\boldsymbol{z}^2)=\begin{bmatrix}\sigma(z_1^2)(1-\sigma(z_1^2))\\\sigma(z_2^2)(1-\sigma(z_2^2))\\\sigma(z_3^2)(1-\sigma(z_3^2))\end{bmatrix}=\begin{bmatrix}0.057\\0.056\\0.018\end{bmatrix}$$

$$g^{1'}(\boldsymbol{z}^1)=\begin{bmatrix}1\\1\\1\end{bmatrix}$$

$$\delta^3=\nabla_{\boldsymbol{y}}\mathcal{L}\odot g^{3'}(\boldsymbol{z}^3)=\begin{bmatrix}-0.016\\0.079\\0.05\end{bmatrix}$$

$$\delta^2=(\boldsymbol{W}^{3\mathrm{T}}\delta^3)\odot g^{2'}(\boldsymbol{z}^2)=\begin{bmatrix}0.004\\0.003\\0.001\end{bmatrix}$$

$$\delta^1=(\boldsymbol{W}^{2\mathrm{T}}\delta^2)\odot g^{1'}(\boldsymbol{z}^1)=\begin{bmatrix}0.002\\0.003\\0.004\end{bmatrix}$$

例子 — 反向传播

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^l} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^l}\frac{\partial \boldsymbol{z}^l}{\partial \boldsymbol{W}^l} = \delta^l \boldsymbol{h}^{l-1\mathrm{T}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^3} = \delta^3 \boldsymbol{h}^{2\mathrm{T}} = \begin{bmatrix} -0.015 & -0.015 & -0.0157 \\ 0.0742 & 0.0743 & 0.0776 \\ 0.0469 & 0.0470 & 0.0491 \end{bmatrix}$$

$$\boldsymbol{W}^3 = \boldsymbol{W}^3 - 0.5\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^3} = \begin{bmatrix} 0.1075 & 0.3075 & 0.5079 \\ 0.3629 & 0.6628 & 0.1612 \\ 0.7766 & 0.1765 & 0.8755 \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^2} = \delta^2 \boldsymbol{h}^{1\mathrm{T}} = \begin{bmatrix} 0.0054 & 0.0051 & 0.0072 \\ 0.0041 & 0.0038 & 0.0054 \\ 0.0014 & 0.0013 & 0.0018 \end{bmatrix}$$

$$\boldsymbol{W}^2 = \boldsymbol{W}^2 - 0.5\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^2} = \begin{bmatrix} 0.1973 & 0.2975 & 0.5964 \\ 0.2980 & 0.4981 & 0.3973 \\ 0.4993 & 0.6994 & 0.7991 \end{bmatrix}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^1} = \delta^1 \boldsymbol{h}^{0\mathrm{T}} = \begin{bmatrix} 0.0002 & 0.0004 & 0.0014 \\ 0.0003 & 0.0006 & 0.0021 \\ 0.0004 & 0.0008 & 0.0028 \end{bmatrix}$$

$$\boldsymbol{W}^1 = \boldsymbol{W}^1 - 0.5\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^1} = \begin{bmatrix} 0.0999 & 0.2998 & 0.3993 \\ 0.1998 & 0.1997 & 0.2989 \\ 0.2998 & 0.6996 & 0.8986 \end{bmatrix}$$

西安交通大學
XI'AN JIAOTONG UNIVERSITY

IAIR Est. 1986
Institute of Artificial Intelligence and Robotics

人工智能学院
College of Artificial Intelligence, XJTU

The End