—武大本科生课程

第15讲　深度学习

(Lecture 15 Deep Learning)

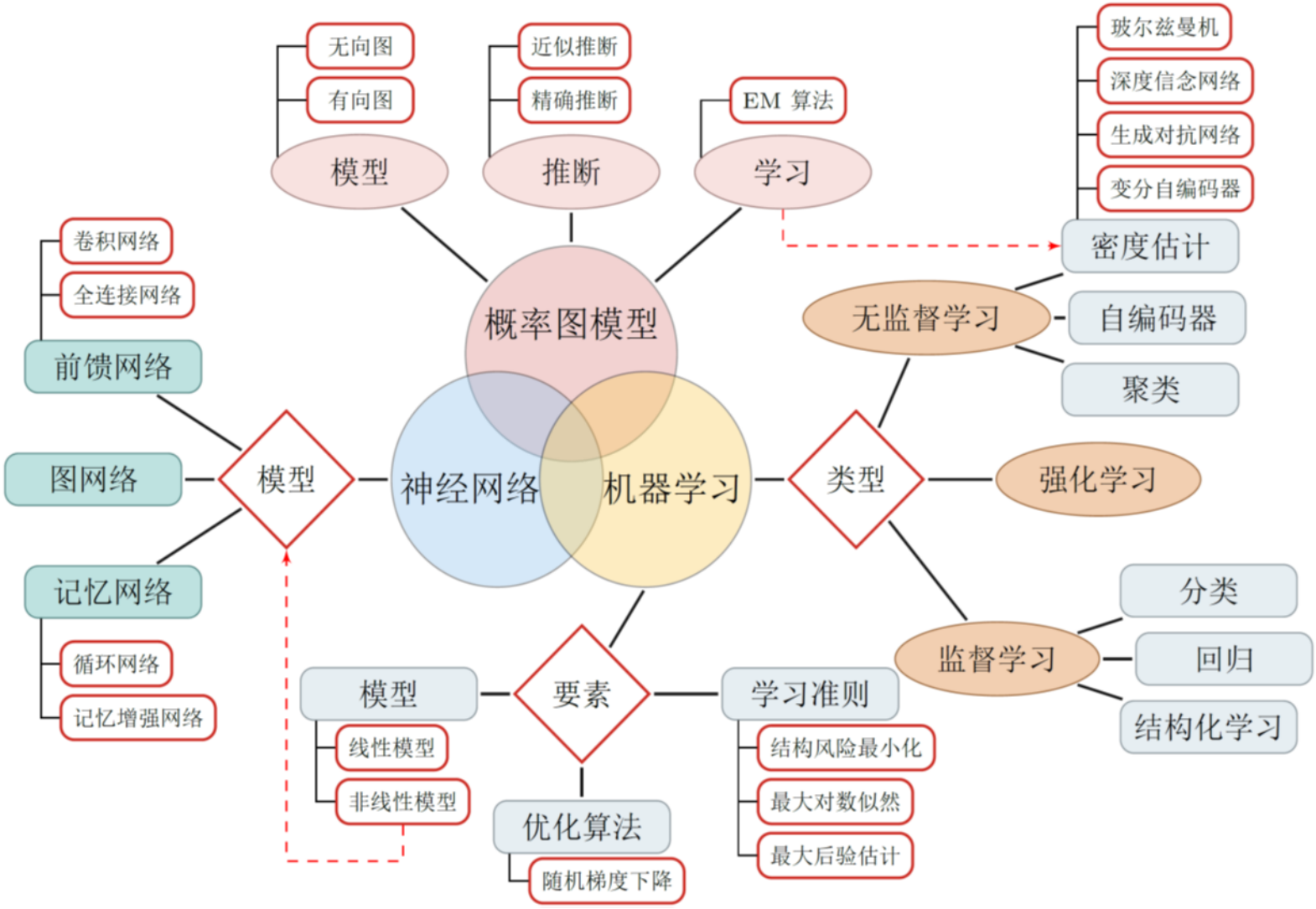
武汉大学计算机学院机器学习课程组

2021.06

Ch13　深度学习

内容目录

深度学习


机器学习
概率图模型
神经网络
模型
无向图
有向图
推断
近似推断
精确推断
学习
EM 算法
前馈网络
卷积网络
全连接网络
图网络
记忆网络
循环网络
记忆增强网络
类型
无监督学习
密度估计
玻尔兹曼机
深度信念网络
生成对抗网络
变分自编码器
自编码器
聚类
强化学习
监督学习
分类
回归
结构化学习
要素
模型
线性模型
非线性模型
学习准则
结构风险最小化
最大对数似然
最大后验估计
优化算法
随机梯度下降

13.1 深度学习简介

深度学习(Deep Learning)是一种基于无监督特征学习和特征层次结构的学习方法。可能的名称还有：特征学习或无监督特征学习。

------学习深度学习要具备一定的神经网络知识：一般需先学习掌握传统的人工神经网络(主要有感知器、BP神经网络等)的基础知识，再学习研究“深度学习”相关部分。

图　深度学习、机器学习、人工智能三者关系

在机器学习中，获得好的特征是识别成功的关键。

1. 为什么要自动学习特征

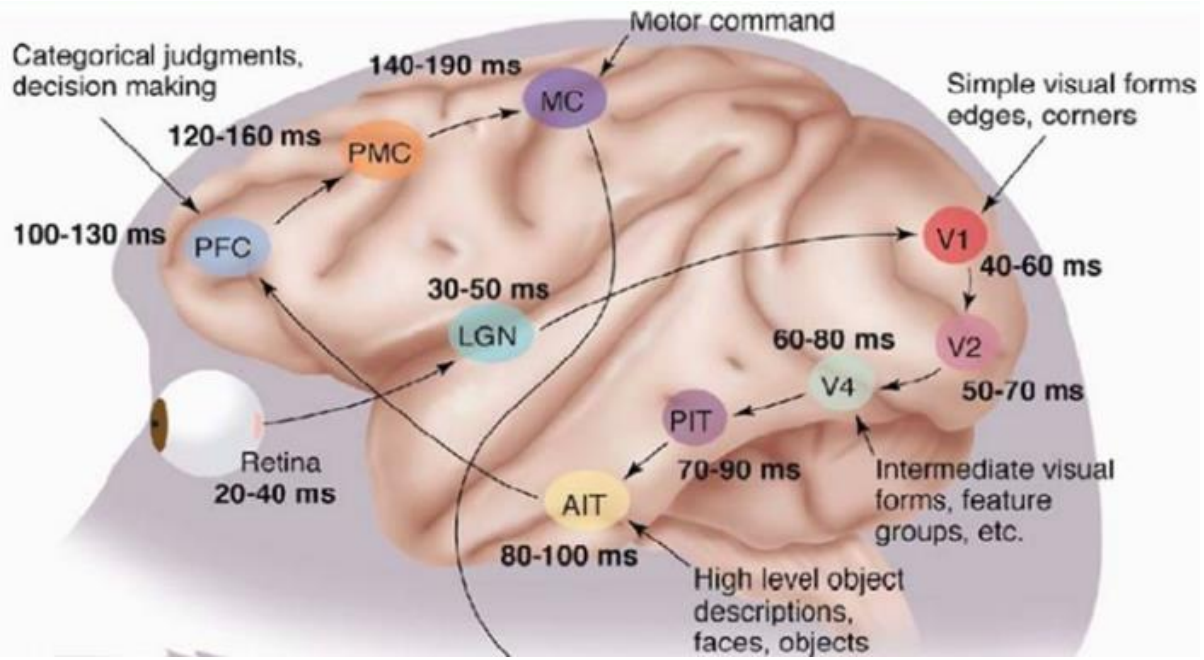
· 一般而言，特征越多，给出信息就越多，识别准确性会得到提升；
· 但特征多，计算复杂度增加，探索的空间大，可以用来训练的数据在每个特征上就会变得稀疏。
· 结论：不一定特征越多越好！有多少个特征，需要学习确定。

2. 为什么采用层次网络结构

■人脑视觉机理

√1981年的诺贝尔医学奖获得者 David Hubel和Torsten Wiesel发现了视觉系统的信息处理机制；

√发现了一种被称为“方向选择性细胞”的神经元细胞，当瞳孔发现了眼前物体的边缘，而且这个边缘指向某个方向时，这种神经元细胞就会活跃。



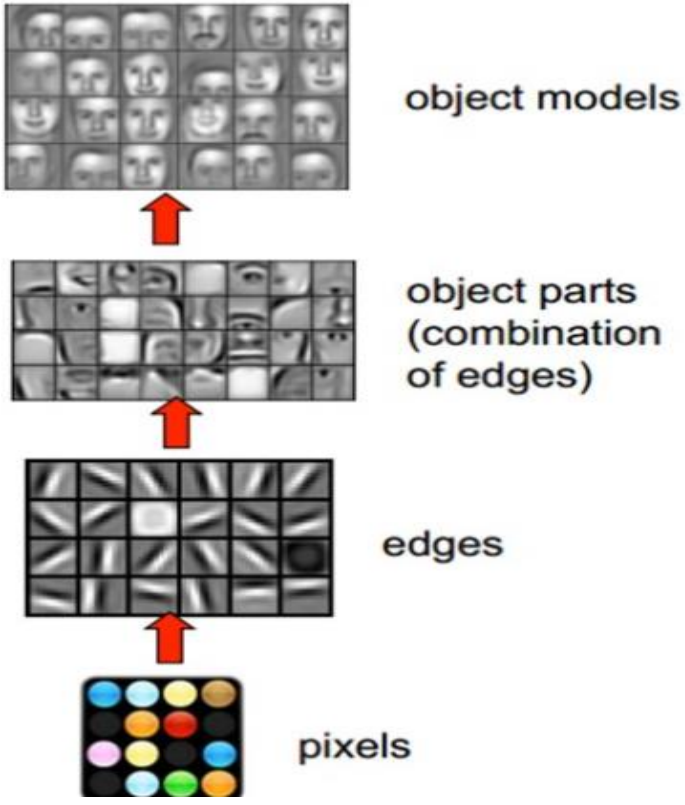
2. 为什么采用层次网络结构

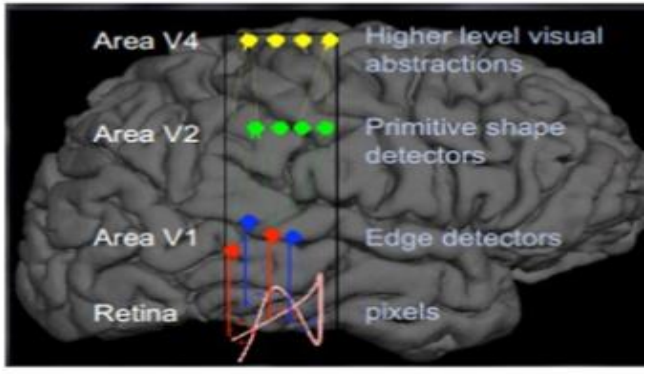
■人脑视觉机理

√人的视觉系统信息处理是分级的；

√高层的特征是低层特征的组合，从低层到高层的特征表示越来越抽象，越来越能表现语义或者意图；

√抽象层面越高，存在的可能猜测就越少，就越有利于分类。





2. 为什么采用层次网络结构

■浅层学习的局限

√人工神经网络（BP算法）

—虽被称作多层感知器，但实际应用中基本上是只含有一层隐层节点的浅层模型

√SVM、Boosting、最大熵方法（如LR：Logistic Regression）

—带有一层隐层节点（如SVM、Boosting），或没有隐层节点（如LR）的浅层模型

局限性：有限样本和计算单元情况下对复杂函数的表示能力有限，针对复杂分类问题其泛化能力受限。

深度学习

- 2006年，加拿大多伦多大学教授、机器学习领域的泰斗、深度学习之父Geoffrey Hinton在《Science》上发表论文提出深度学习主要观点[1]：

1) 多隐层的人工神经网络具有优异的特征学习能力，学习得到的特征对数据有更本质的刻画，从而有利于可视化或分类；

2) 深度神经网络在训练上的难度，可以通过“逐层初始化”（layer-wise pre-training）来有效克服，逐层初始化可通过无监督学习实现。

Reference

[1] G. E. Hinton, et al. Reducing the Dimensionality of Data with Neural Networks. Science 28 July, Pages: 504-507, 2006.

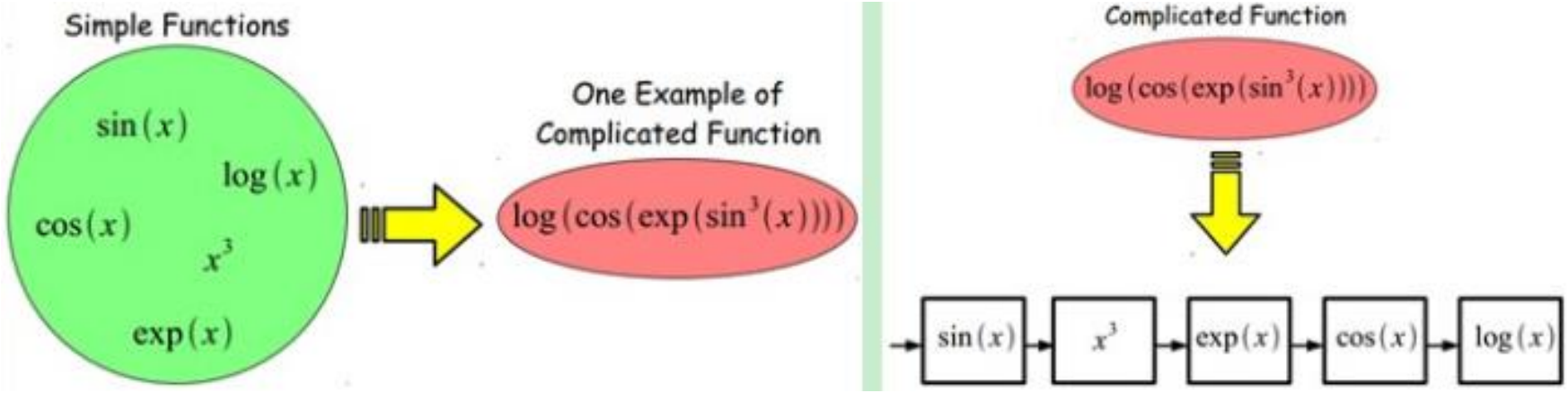
深度学习

- **本质：深度学习的实质**，是通过构建具有**很多隐层的机器学习模型**和**海量的训练数据**，来学习更有用的特征，从而最终提升分类或预测的准确性。**“深度模型”是手段，“特征学习”是目的。**
- **与浅层学习(shallow learning)区别：**

1) 强调了模型结构的深度，通常有5~1000层的隐层节点；
2) 明确突出了特征学习的重要性，通过逐层特征变换，将样本在原空间的特征表示变换到一个新特征空间，从而使分类或预测更加容易。**与人工规则构造特征的方法相比，利用大数据来学习特征，更能够刻画数据的丰富内在信息。**

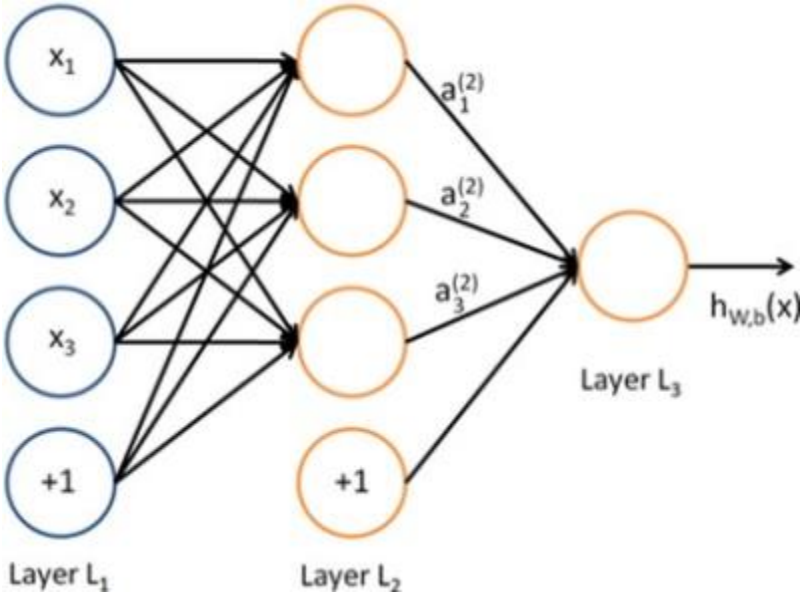
深度学习

- 优点：可通过学习一种深层非线性网络结构，实现复杂函数逼近，表征输入数据分布式表示。

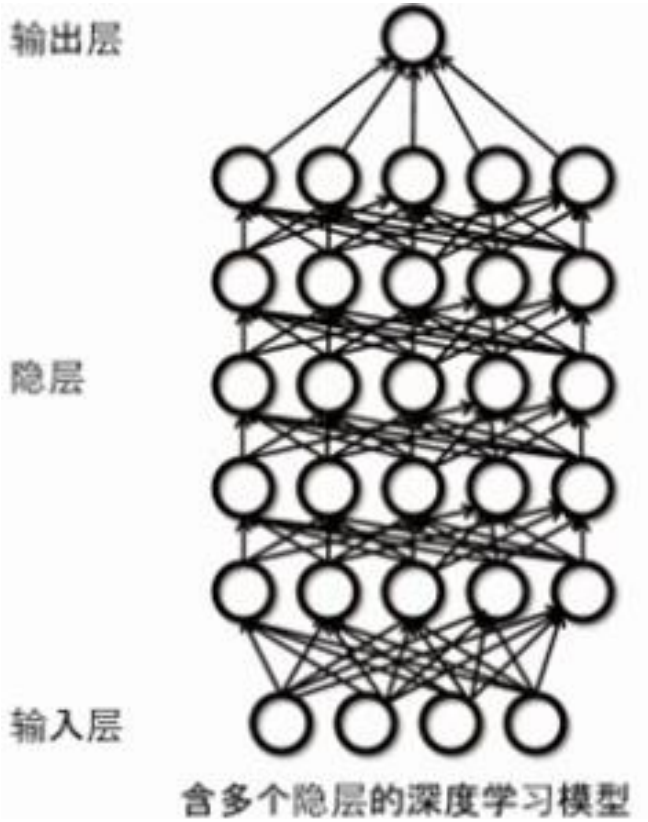


深度学习 vs. 神经网络

神经网络：



深度学习：



深度学习 vs. 神经网络

相同点：二者均采用分层结构，系统包括输入层、隐层（多层）、输出层组成的多层网络，只有相邻层节点之间有连接，同一层以及跨层节点之间相互无连接，每一层可以看作是一个**Logistic回归模型**。

不同点：

神经网络：采用BP算法调整参数，即采用迭代算法来训练整个网络。随机设定初值，计算当前网络的输出，然后根据当前输出和样本真实标签之间的差去改变前面各层的参数，直到收敛；

深度学习：采用逐层训练机制。采用该机制的原因在于如果采用BP机制，对于一个deep network（5层以上），残差传播到最前面的层将变得很小，出现所谓的gradient diffusion (**梯度弥散/梯度扩散**)。

深度学习 vs. 神经网络

- 神经网络的局限性：

1) 较容易过拟合，参数较难调整，而且需要不少技巧；
2) 训练速度较慢，在层次比较少(小于等于3)的情况下效果并不比其它方法更优；

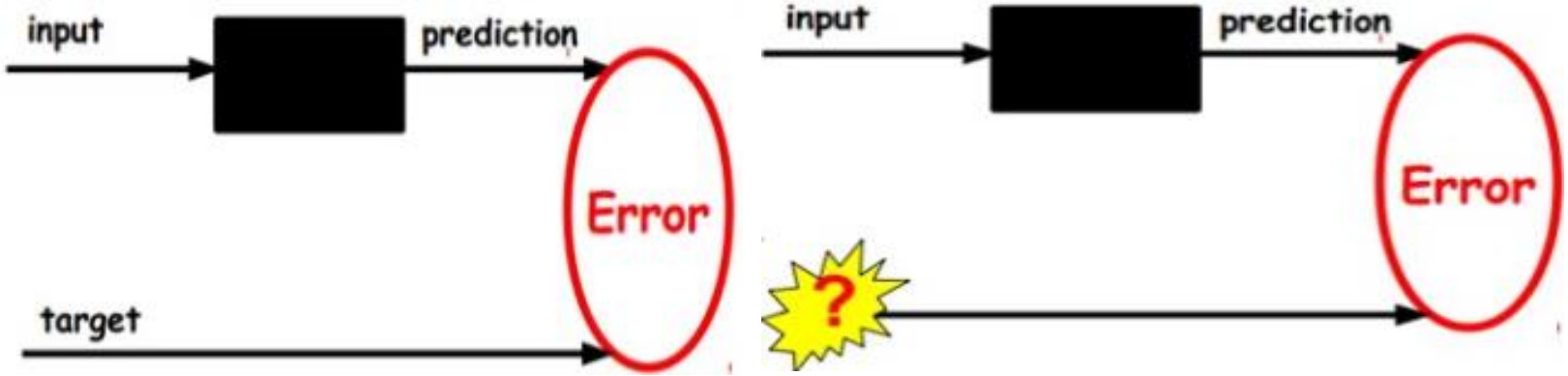
13.3 深度学习训练过程

- 不采用BP算法的原因

（1）反馈调整时，梯度越来越稀疏，从顶层越往下，误差校正信号越来越小；

（2）收敛易陷入局部极小，由于是采用随机值初始化，当初值是远离最优区域时易导致这一情况；

（3）BP算法需要有标签数据来训练，但大部分数据是无标签的；



13.2 深度学习训练过程

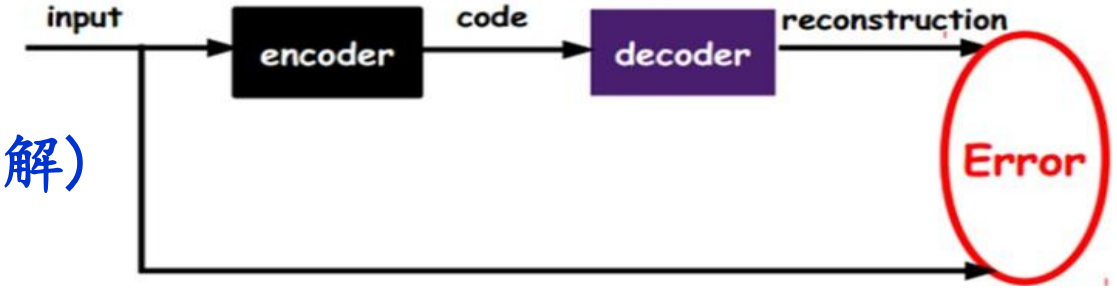
- 第一步：采用自下而上的无监督学习

1）逐层构建单层神经元。

2）每层采用wake-sleep算法进行调优。每次仅调整一层，逐层调整。

该过程可以看作是一个feature learning的过程，是和传统神经网络区别最大的部分。

13.2 深度学习训练过程



- wake-sleep算法(*：了解)

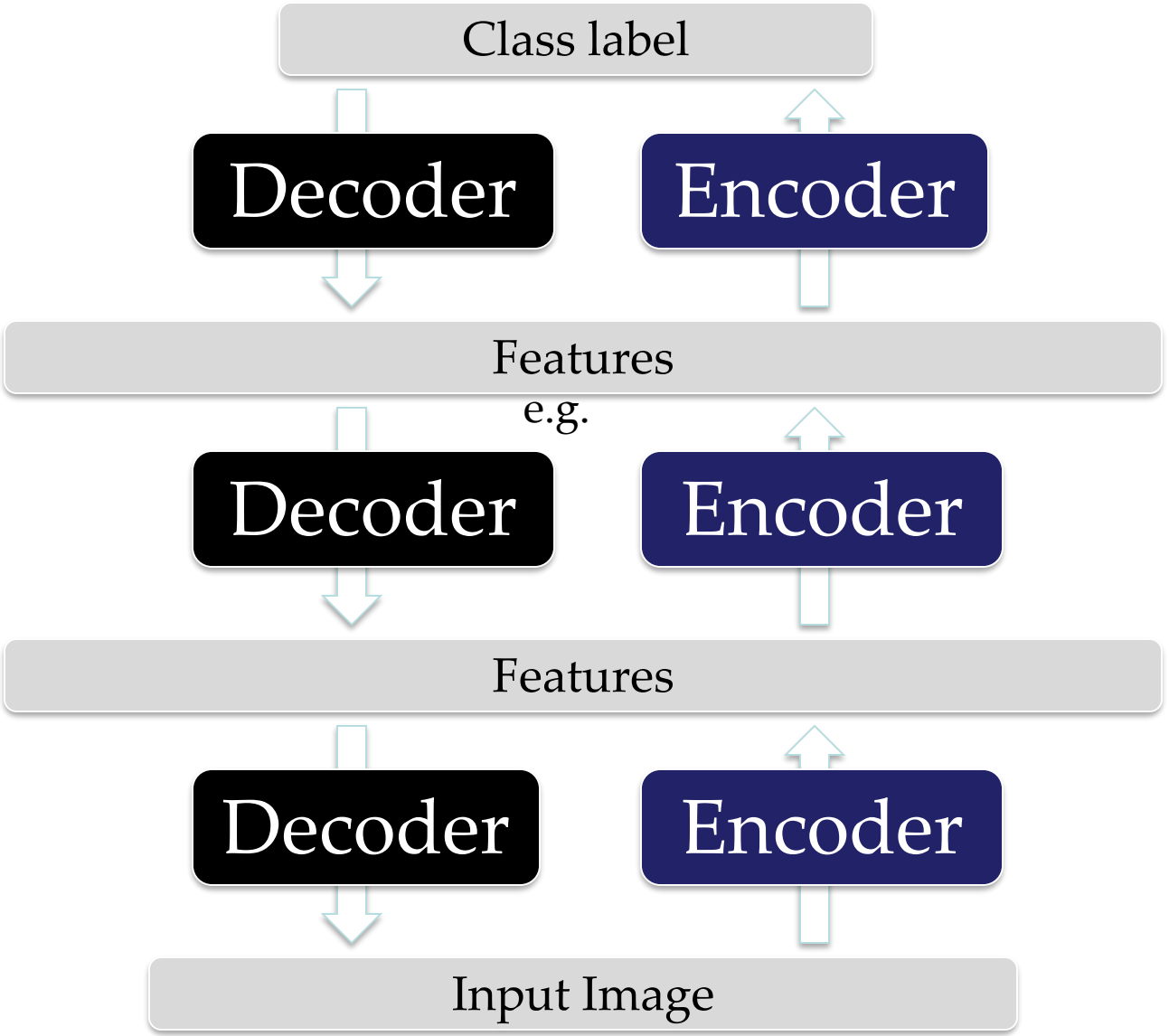
1）wake阶段

认知过程(bottom-up，从下到上)，通过下层的输入特征(Input)和向上的认知(Encoder)权重产生每一层的抽象表示/概念(Code)，再通过当前的生成(Decoder)权重产生一个重建信息(Reconstruction)，计算输入特征和重建信息残差，使用梯度下降修改层间的下行生成(Decoder)权重。也就是“如果现实跟我想象的不一样，改变我的生成权重使得我想象的东西变得与现实一样”。

2）sleep阶段

生成过程(top-down，从上到下)，通过上层概念(Code)和向下的生成(Decoder)权重，生成下层的状态，再利用认知(Encoder)权重产生一个抽象景象。利用初始上层概念和新建抽象景象的残差，利用梯度下降修改层间向上的认知(Encoder)权重。也就是“如果梦中的景象不是我脑中的相应概念，改变我的认知权重使得这种景象在我看来就是这个概念”。

13.2 深度学习训练过程

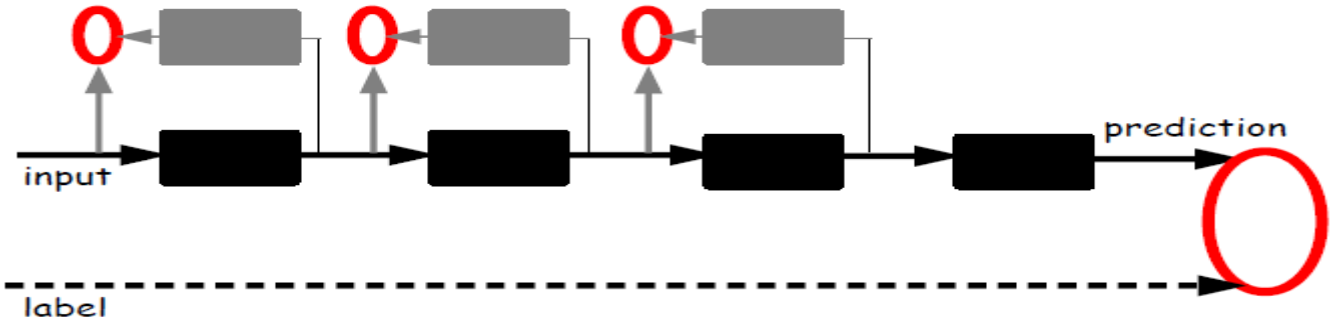


13.2 深度学习训练过程

- 第二步：采用自顶向下的监督学习

这一步是在第一步学习获得各层参数的基础上，在最顶的编码层添加一个分类器（例如Logistic回归、ANN、SVM等），然后通过带标签数据的有监督学习，利用梯度下降法去微调整个网络参数。

深度学习的第一步实质上是一个网络参数初始化过程。区别于传统神经网络初值随机初始化，深度学习模型是通过无监督学习输入数据的结构得到的，因而这个初值更接近全局最优，从而能够取得更好的效果。



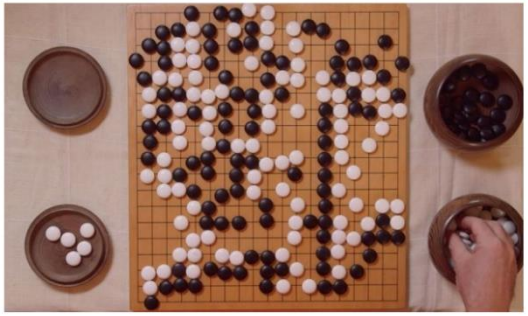
Review:

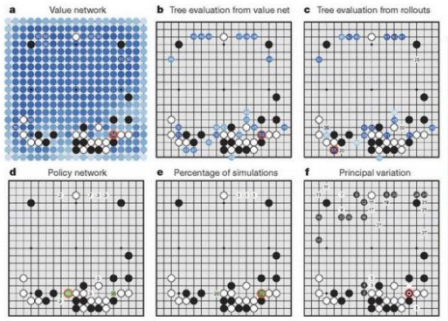
- **Deep Learning:** a class of machine learning techniques, where many layers of information processing stages in hierarchical architectures are exploited for unsupervised feature learning and for pattern analysis/classification. The essence of deep learning is to compute hierarchical features or representations of the observational data, where the higher-level features or factors are defined from lower-level ones. (机器学习的一类技术，它通过分层结构的分阶段信息处理来探索无监督的特征学习和模式分析/分类。深度学习的本质是计算观察数据的分层特征或表示，其中高层特征或因子由低层特征得到。)

13.2 深度学习的常见模型及方法

- 卷积神经网络CNN
- 循环神经网络RNN (Recurrent Neural Networks): LSTM (Long Short-Term Memory networks), GRU (Gated Recurrent Unit)
- 图神经网络GNN (Graph Neural Networks)：Transformer
- 生成对抗网络GAN (Generative Adversarial Networks)
- 自动编码器AutoEncoder
- 稀疏自动编码器Sparse AutoEncode r
- 限制波尔兹曼机 Restricted Boltzmann Machine (RBM)
- 深度置信网络 Deep Belief Networks

AlphaGo





The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

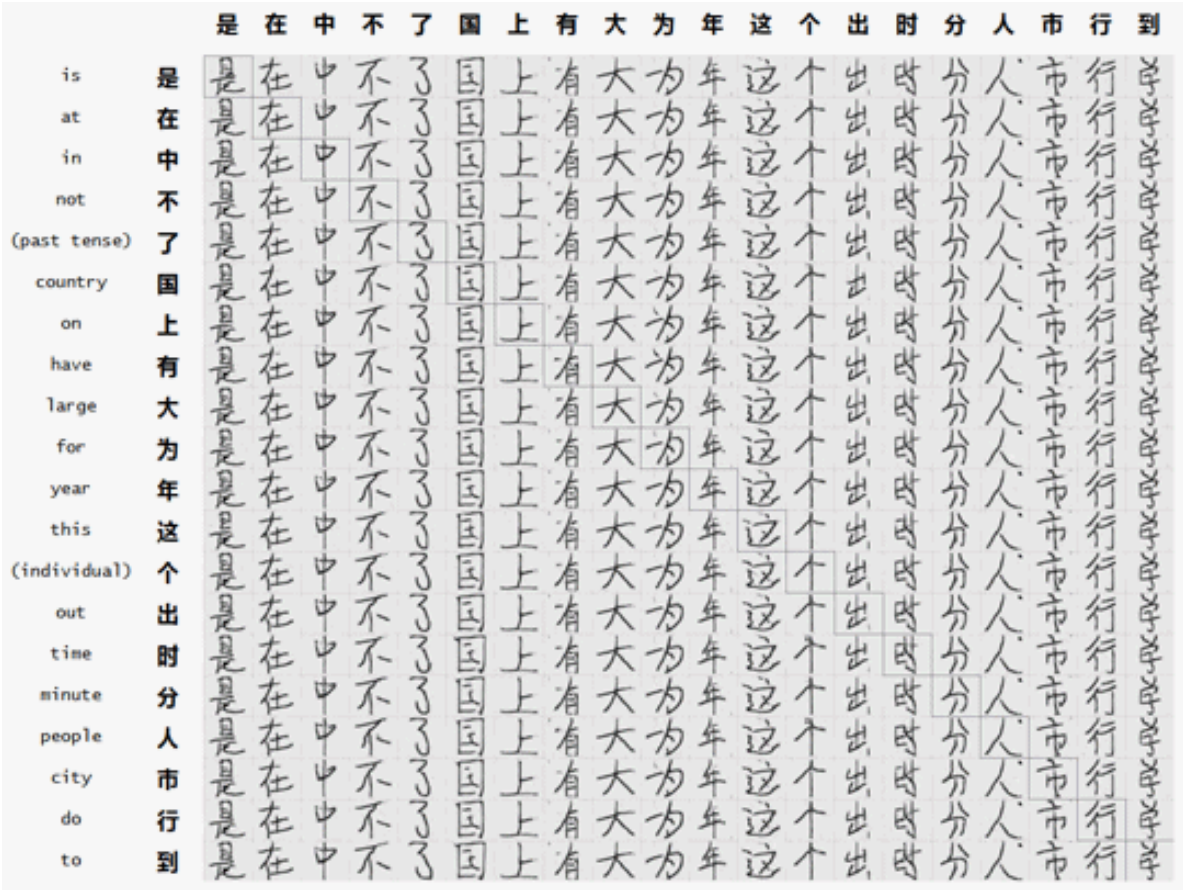
policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

- 分布式系统：1202 个CPU 和176 块GPU
- 单机版：48 个CPU 和8 块GPU
- 走子速度：3 毫秒-2 微秒

Mask RCNN

Figure 4. More results of **Mask R-CNN** on COCO test images, using ResNet-101-FPN and running at 5 fps, with 35.7 mask AP (Table 1).

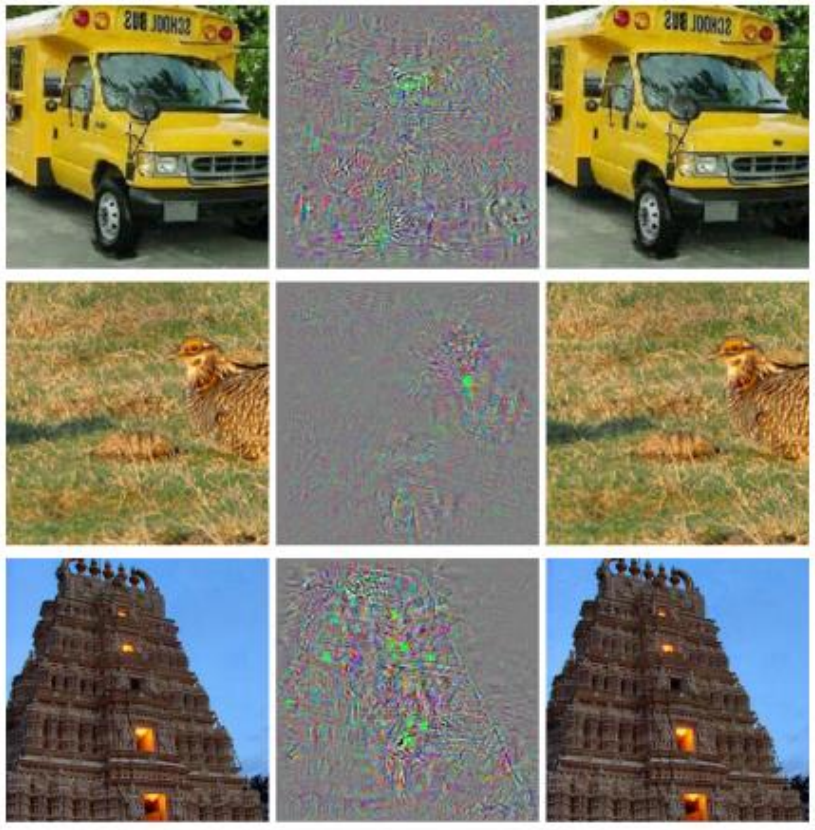
图像生成


是 在 中 不 了 国 上 有 大 为 年 这 个 出 时 分 人 市 行 到
is 是
at 在
in 中
not 不
(past tense) 了
country 国
on 上
have 有
large 大
for 为
year 年
this 这
(individual) 个
out 出
time 时
minute 分
people 人
city 市
do 行
to 到

Deep Dream

画风迁移

对抗样本



(a)

(b)

RNN—前馈网络的一些不足

- 连接存在层与层之间，每层的节点之间是无连接的（无循环）
- 输入和输出的维数都是固定的，不能任意改变。无法处理变长的序列数据
- 假设每次输入都是独立的，也就是说每次网络的输出只依赖于当前的输入

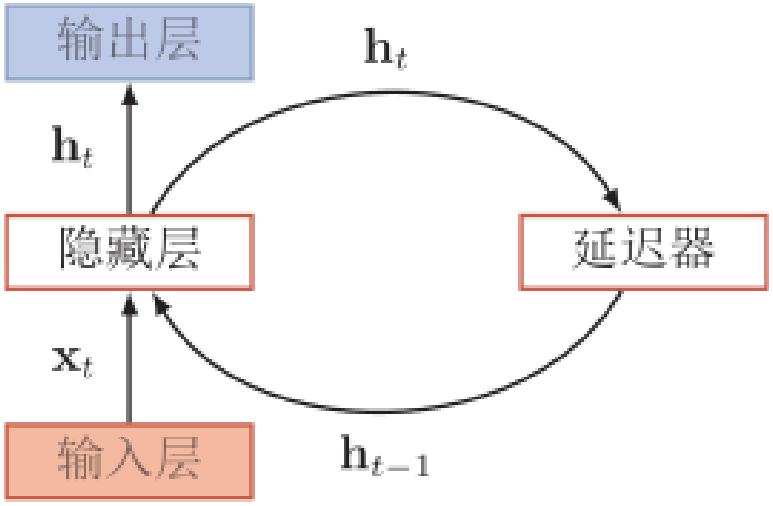
循环神经网络

- 循环神经网络
  - 循环神经网络通过使用带自反馈的神经元，能够处理任意长度的序列。
  - 循环神经网络比前馈神经网络更加符合生物神经网络的结构。
  - 循环神经网络已经被广泛应用在语音识别、语言模型以及自然语言生成等任务上。

循环神经网络

给定一个输入序列$\mathbf{x}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$，循环神经网络通过下面公式更新带反馈边的隐藏层的**活性值**$\mathbf{h}_t$：

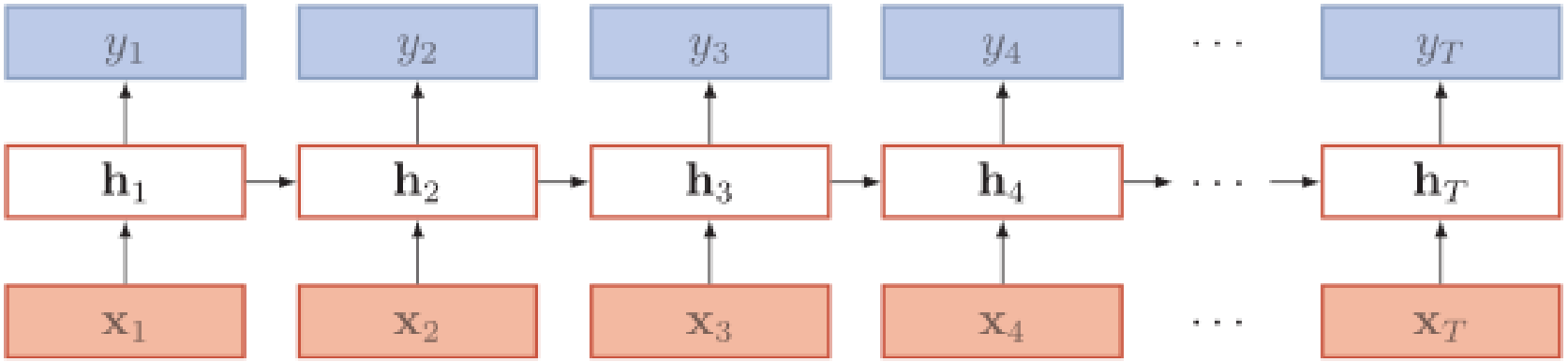
$$\mathbf{h}_t = \begin{cases} 0 & t = 0 \\ f(\mathbf{h}_{t-1}, \mathbf{x}_t) & \text{otherwise} \end{cases}$$



简单循环网络

- 状态更新：

$$\mathbf{h}_t = f(U\mathbf{h}_{t-1} + W\mathbf{x}_t + \mathbf{b}),$$



RNN是图灵完全等价的 (Siegelmann and Sontag, 1995)

FNN：模拟任何函数
RNN：模拟任何程序（计算过程）。

长期依赖问题

- 循环神经网络在时间维度上非常深！
  - 梯度消失或梯度爆炸

- 如何改进？
  - 梯度爆炸问题
    - 权重衰减
    - 梯度截断
  - 梯度消失问题
    - 改进模型

长期依赖问题

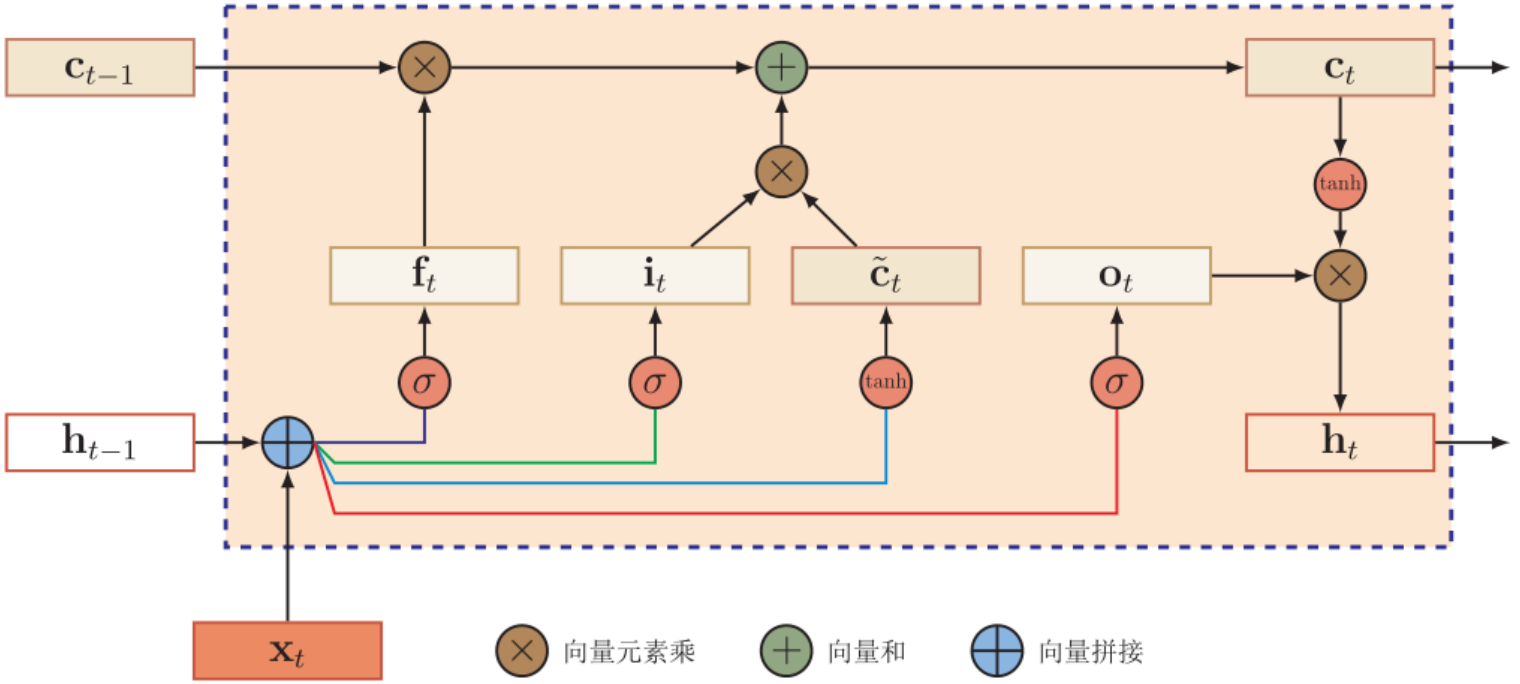
- 改进方法
  - 循环边改为线性依赖关系

$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t; \theta),$$

  - 增加非线性

$$\mathbf{h}_t = \mathbf{h}_{t-1} + g(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta),$$

长短时记忆神经网络：LSTM



$$\mathbf{i}_t = \sigma(W_i\mathbf{x}_t + U_i\mathbf{h}_{t-1} + \mathbf{b}_i),$$

$$\mathbf{f}_t = \sigma(W_f\mathbf{x}_t + U_f\mathbf{h}_{t-1} + \mathbf{b}_f),$$

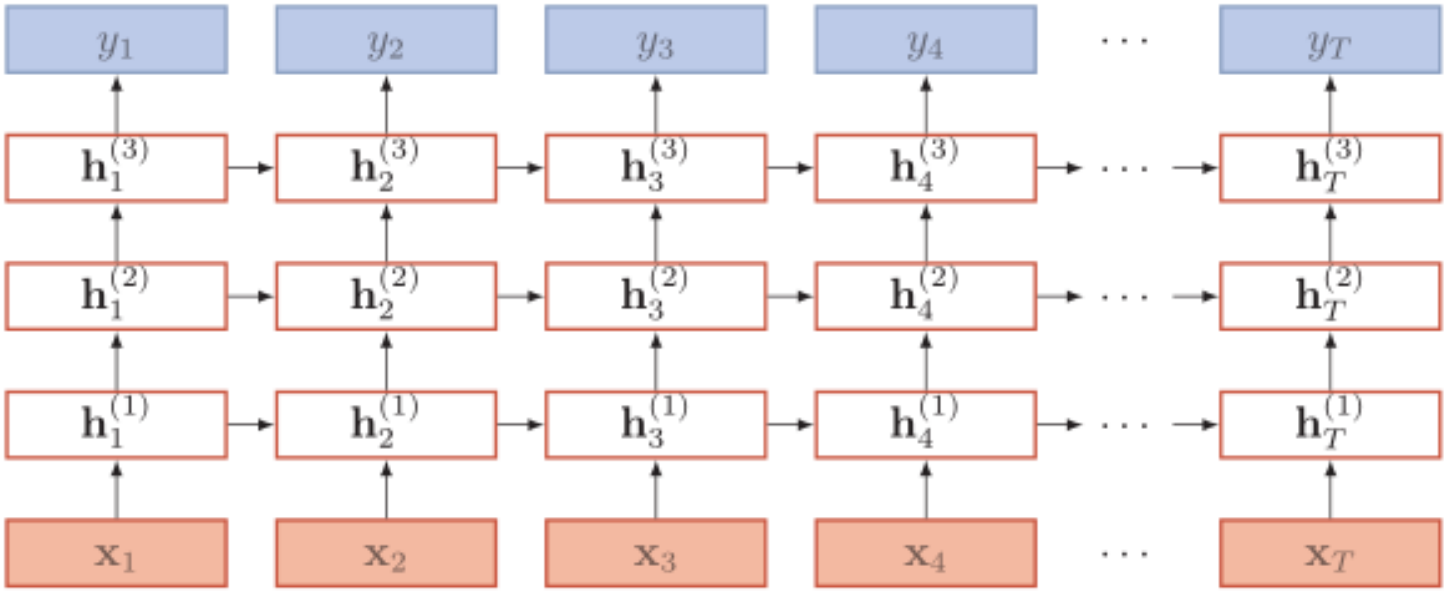
$$\mathbf{o}_t = \sigma(W_o\mathbf{x}_t + U_o\mathbf{h}_{t-1} + \mathbf{b}_o),$$

$$\tilde{\mathbf{c}}_t = \tanh(W_c\mathbf{x}_t + U_c\mathbf{h}_{t-1} + \mathbf{b}_c)$$

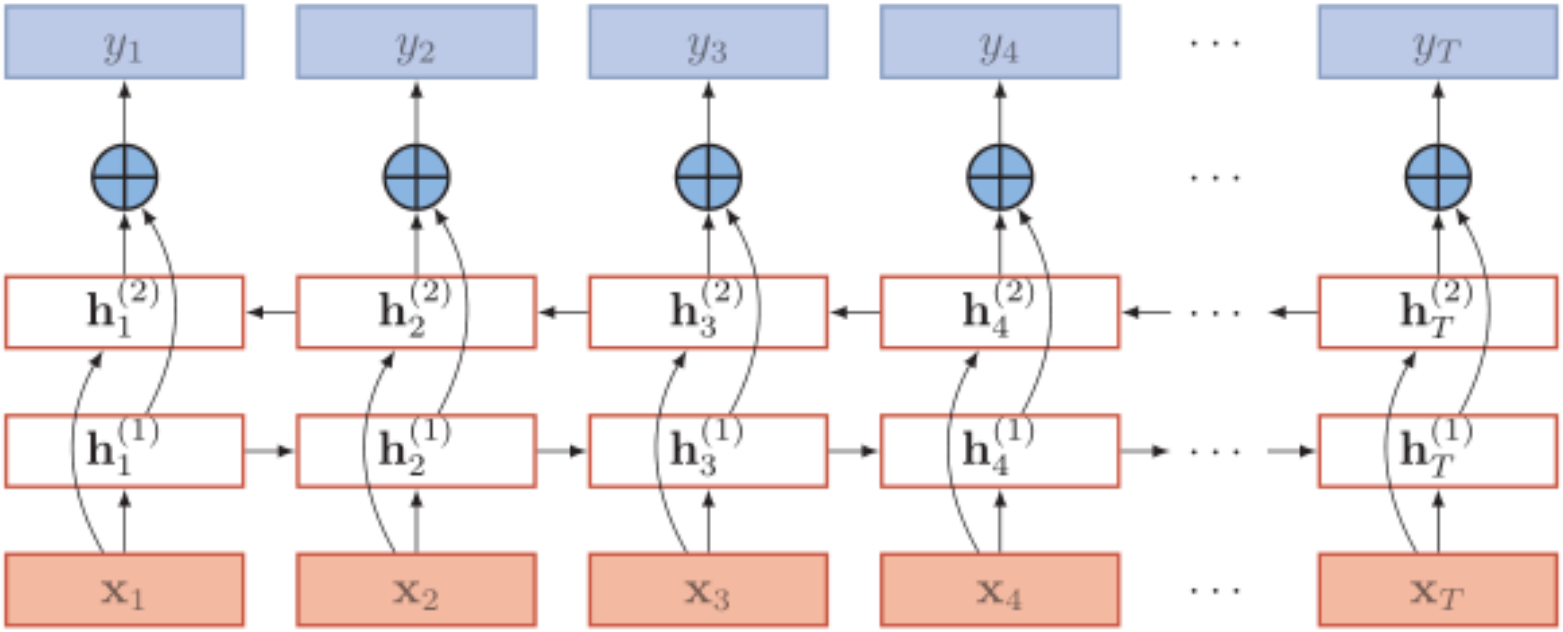
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t,$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),$$

堆叠循环神经网络


y_1 y_2 y_3 y_4 $\cdots$ y_T
$\mathbf{h}_1^{(3)}$ $\mathbf{h}_2^{(3)}$ $\mathbf{h}_3^{(3)}$ $\mathbf{h}_4^{(3)}$ $\cdots$ $\mathbf{h}_T^{(3)}$
$\mathbf{h}_1^{(2)}$ $\mathbf{h}_2^{(2)}$ $\mathbf{h}_3^{(2)}$ $\mathbf{h}_4^{(2)}$ $\cdots$ $\mathbf{h}_T^{(2)}$
$\mathbf{h}_1^{(1)}$ $\mathbf{h}_2^{(1)}$ $\mathbf{h}_3^{(1)}$ $\mathbf{h}_4^{(1)}$ $\cdots$ $\mathbf{h}_T^{(1)}$
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$ $\mathbf{x}_4$ $\cdots$ $\mathbf{x}_T$

双向循环神经网络


y_1
y_2
y_3
y_4
$\cdots$
y_T
$\mathbf{h}_1^{(2)}$
$\mathbf{h}_2^{(2)}$
$\mathbf{h}_3^{(2)}$
$\mathbf{h}_4^{(2)}$
$\mathbf{h}_T^{(2)}$
$\mathbf{h}_1^{(1)}$
$\mathbf{h}_2^{(1)}$
$\mathbf{h}_3^{(1)}$
$\mathbf{h}_4^{(1)}$
$\mathbf{h}_T^{(1)}$
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_4$
$\mathbf{x}_T$

循环神经网络的扩展

- 递归神经网络

- 图网络

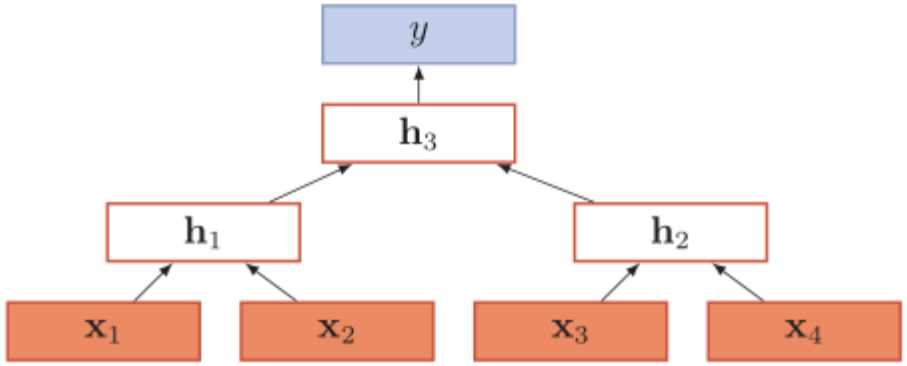
递归神经网络
Recursive Neural Network

- 递归神经网络实在一个有向图无循环图上共享一个组合函数

$$\mathbf{h}_1 = f(W \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} + \mathbf{b}),$$

$$\mathbf{h}_2 = f(W \begin{bmatrix} \mathbf{x}_3 \\ \mathbf{x}_4 \end{bmatrix} + \mathbf{b}),$$

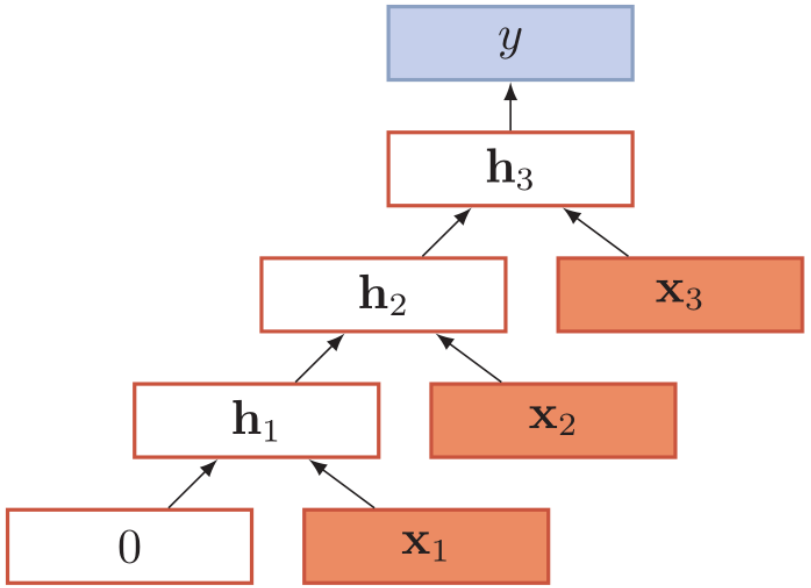
$$\mathbf{h}_3 = f(W \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{bmatrix} + \mathbf{b}),$$



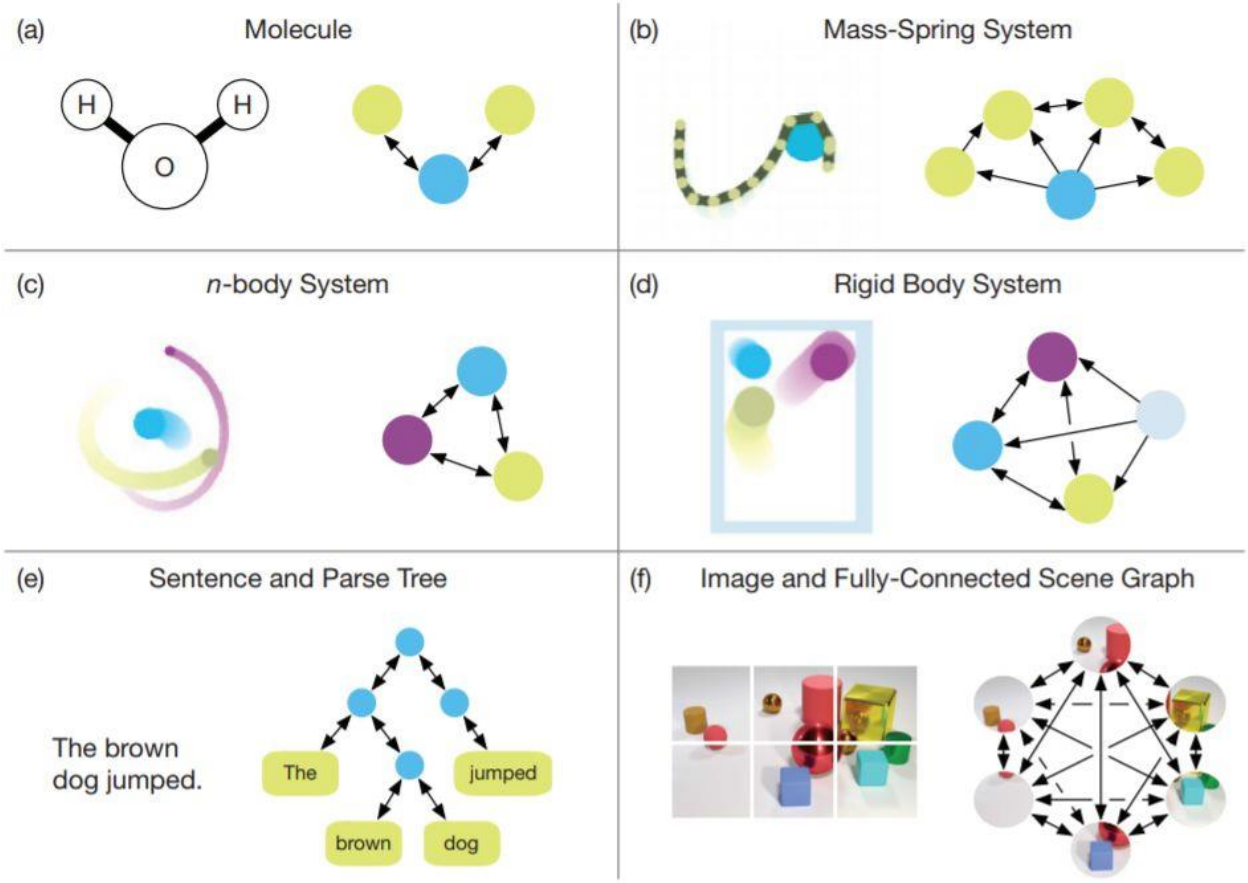
递归神经网络

- 退化为循环神经网络

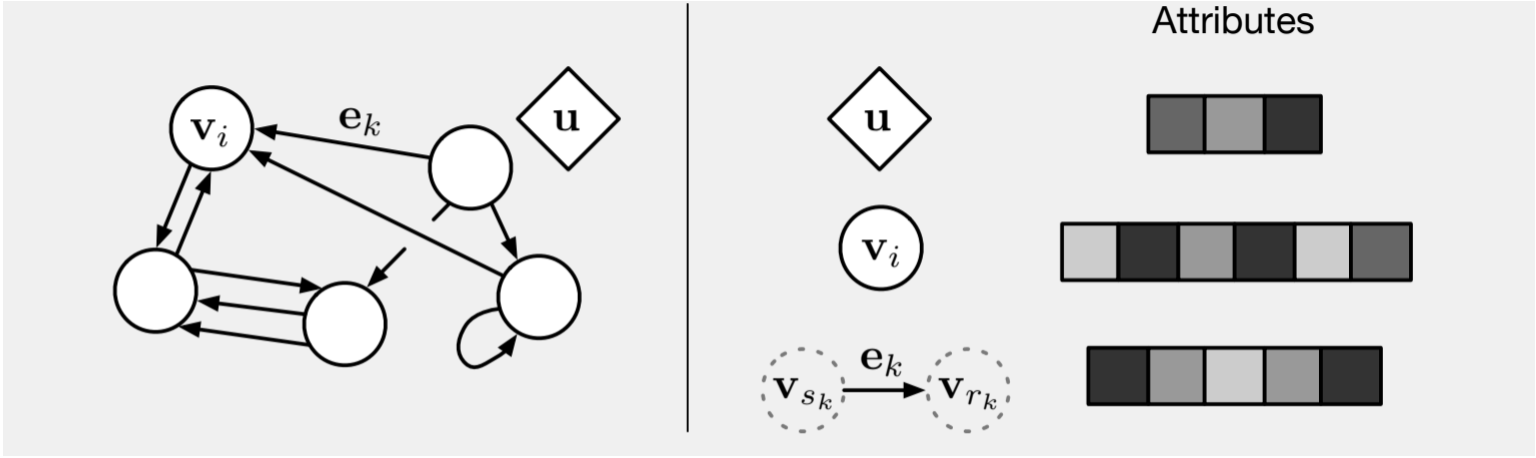
$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t),$$

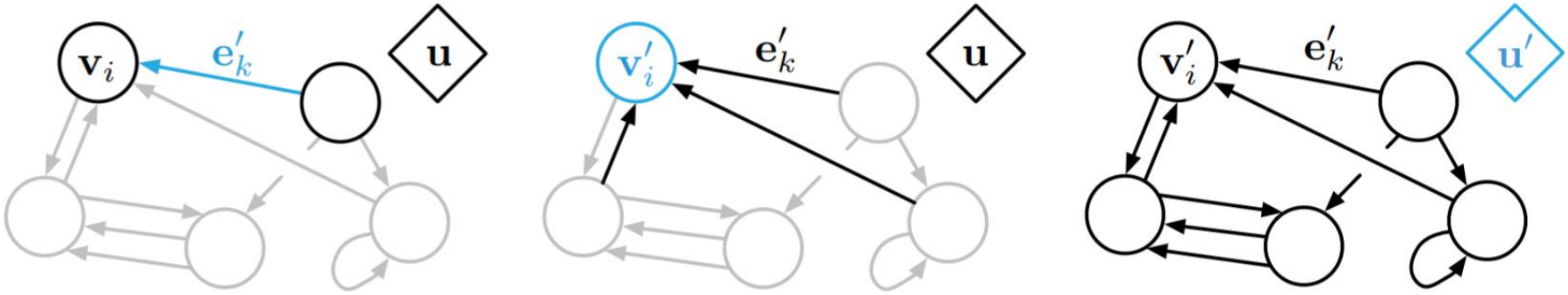


图网络


(a)
Molecule
H
H
O
(b)
Mass-Spring System
(c)
n-body System
(d)
Rigid Body System
(e)
Sentence and Parse Tree
The brown dog jumped.
The
jumped
brown
dog
(f)
Image and Fully-Connected Scene Graph

图网络





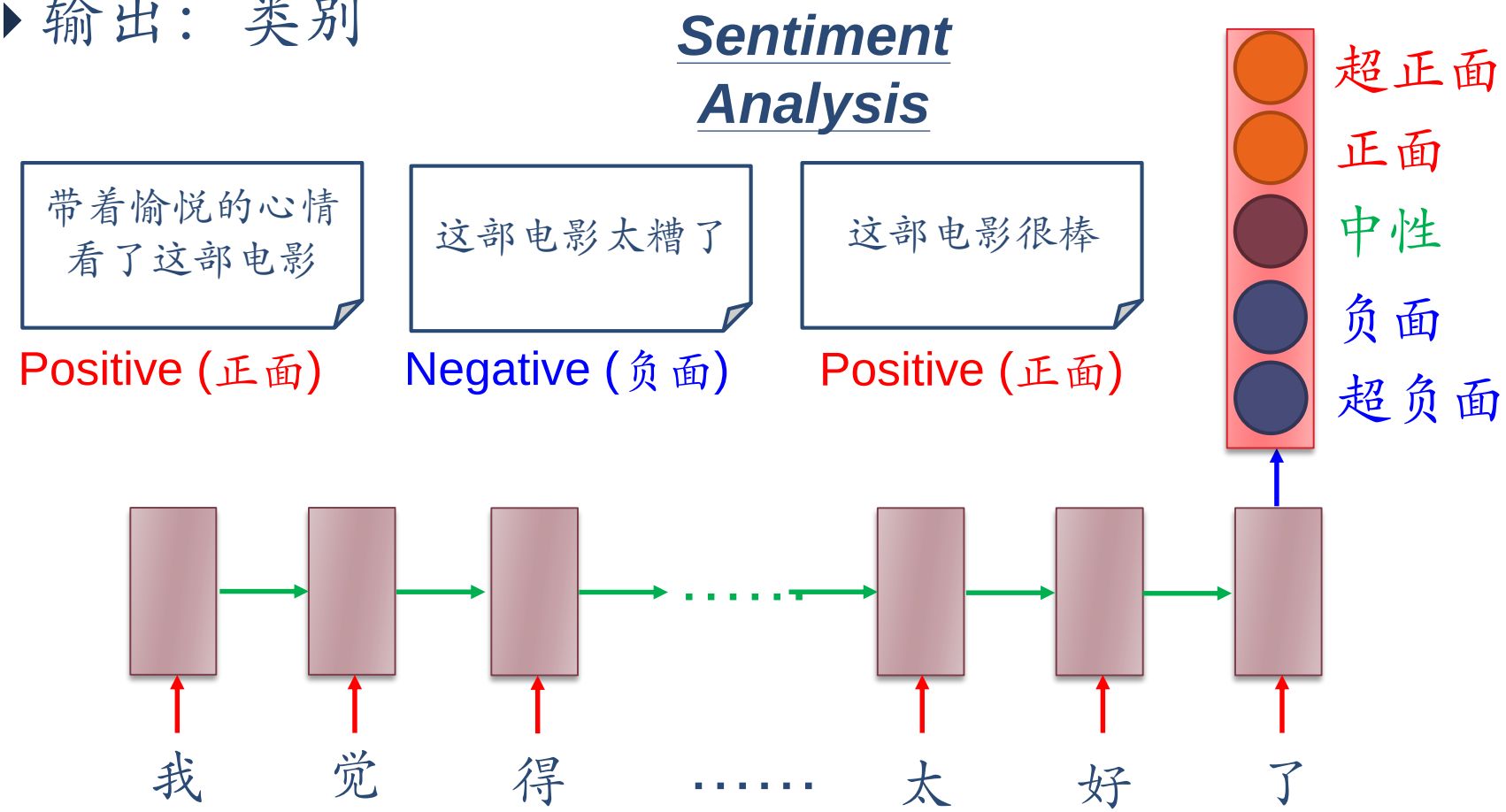
(a) Edge update (b) Node update (c) Global update

Relational inductive biases, deep learning, and graph networks

序列到类别

来源：李宏毅《1天搞懂深度学习》

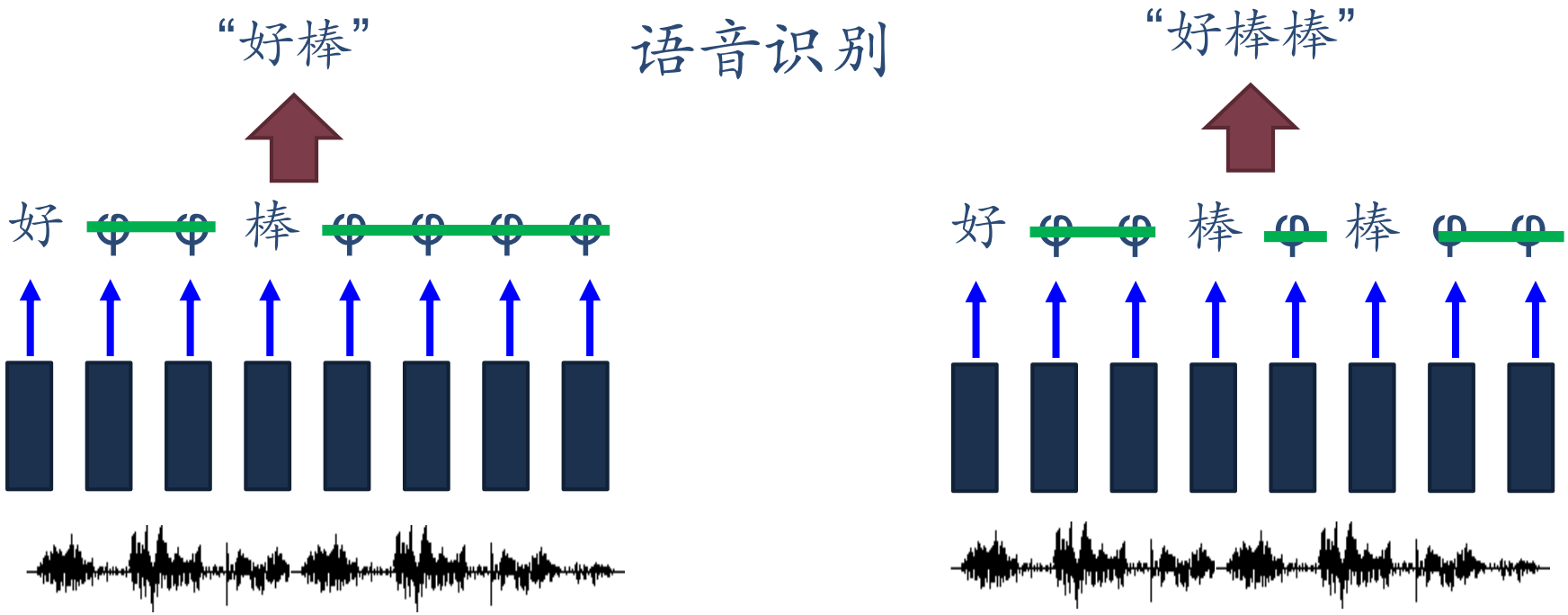
- 输入：序列
- 输出：类别



同步的序列到序列模式

来源：李宏毅《1天搞懂深度学习》

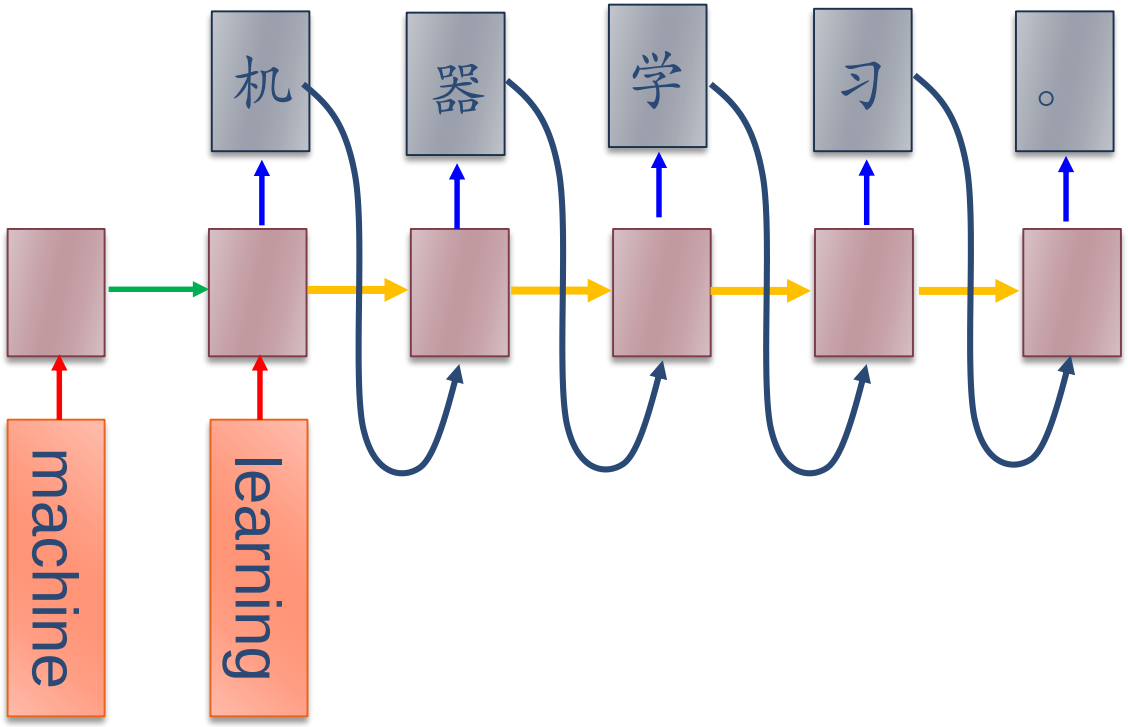
- Connectionist Temporal Classification (CTC) [Alex Graves, ICML’06][Alex Graves, ICML’14][Haşim Sak, Interspeech’15][Jie Li, Interspeech’15][Andrew Senior, ASRU’15]



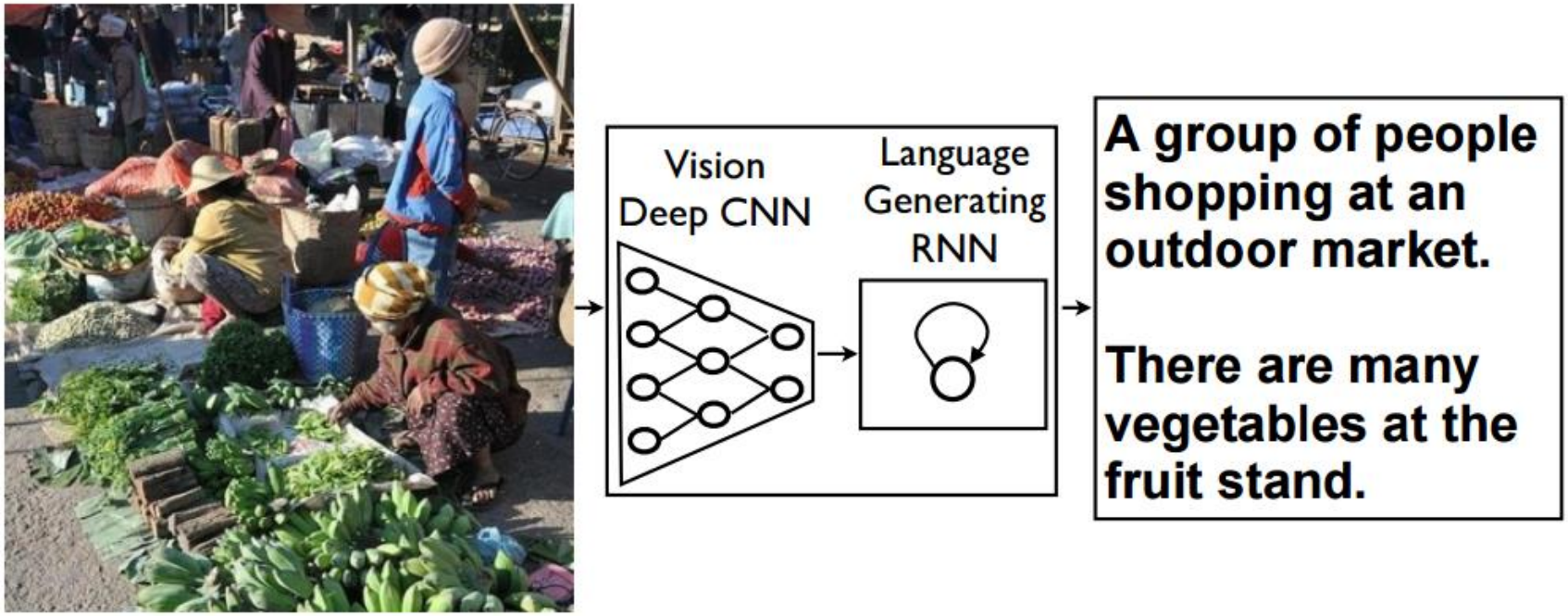
异步的序列到序列模式

来源：李宏毅《1天搞懂深度学习》

- 机器翻译



看图说话


Vision
Deep CNN
Language
Generating
RNN
A group of people shopping at an outdoor market.
There are many vegetables at the fruit stand.

看图说话

Figure 5. A selection of evaluation results, grouped by human rating.

生成LINUX内核代码

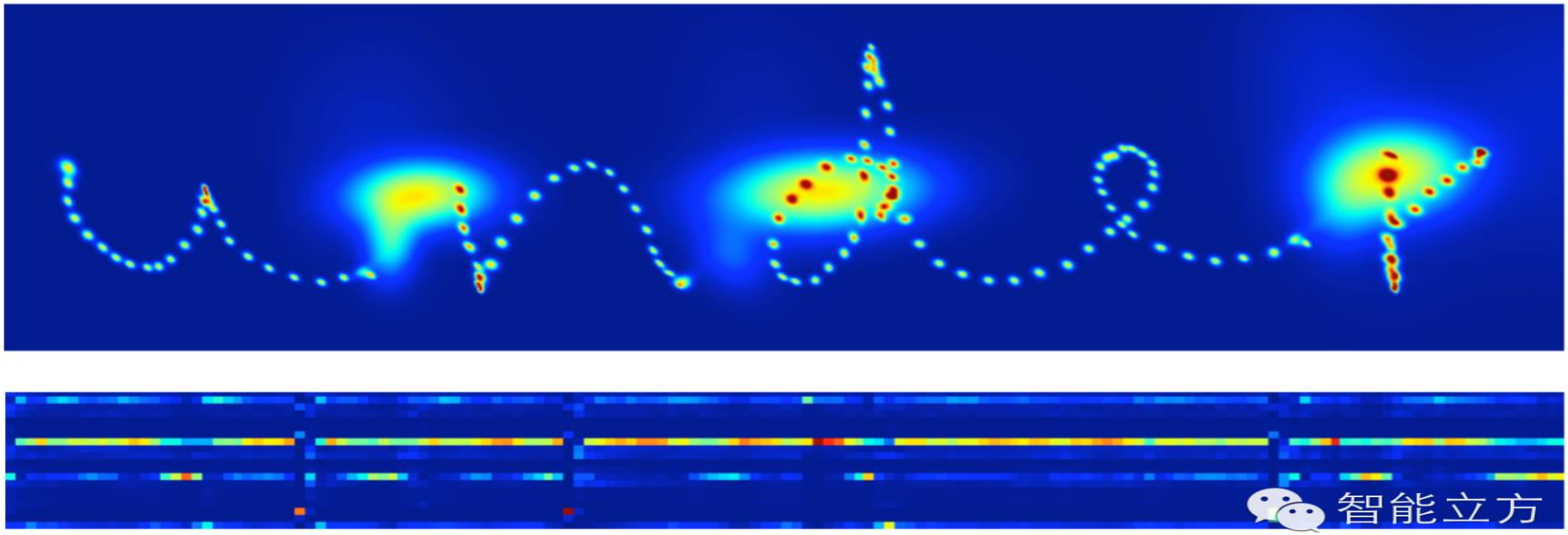
```
/*
 * If this error is set, we will need anything right after that BSD.
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long) *FIRST_COMPAT);
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  div_u64_w(val, inb_p);
  spin_unlock(&disk->queue_lock);
  mutex_unlock(&s->sock->mutex);
  mutex_unlock(&func->mutex);
  return disassemble(info->pending_bh);
}

static void num_serial_settings(struct tty_struct *tty)
{
  if (tty == tty)
    disable_single_st_p(dev);
  pci_disable_spool(port);
```

作诗

白鹭窥鱼立， Egrets stood, peeping fishes. 青山照水开. Water was still, reflecting mountains. 夜来风不动， The wind went down by nightfall, 明月见楼台. as the moon came up by the tower.	满怀风月一枝春， Budding branches are full of romance. 未见梅花亦可人. Plum blossoms are invisible but adorable. 不为东风无此客， With the east wind comes Spring. 世间何处是前身. Where on earth do I come from?

写字

- 把一个字母的书写轨迹看作是一连串的点。一个字母的“写法”其实是每一个点相对于前一个点的偏移量，记为(offset x, offset y)。再增加一维取值为0或1来记录是否应该“提笔”。



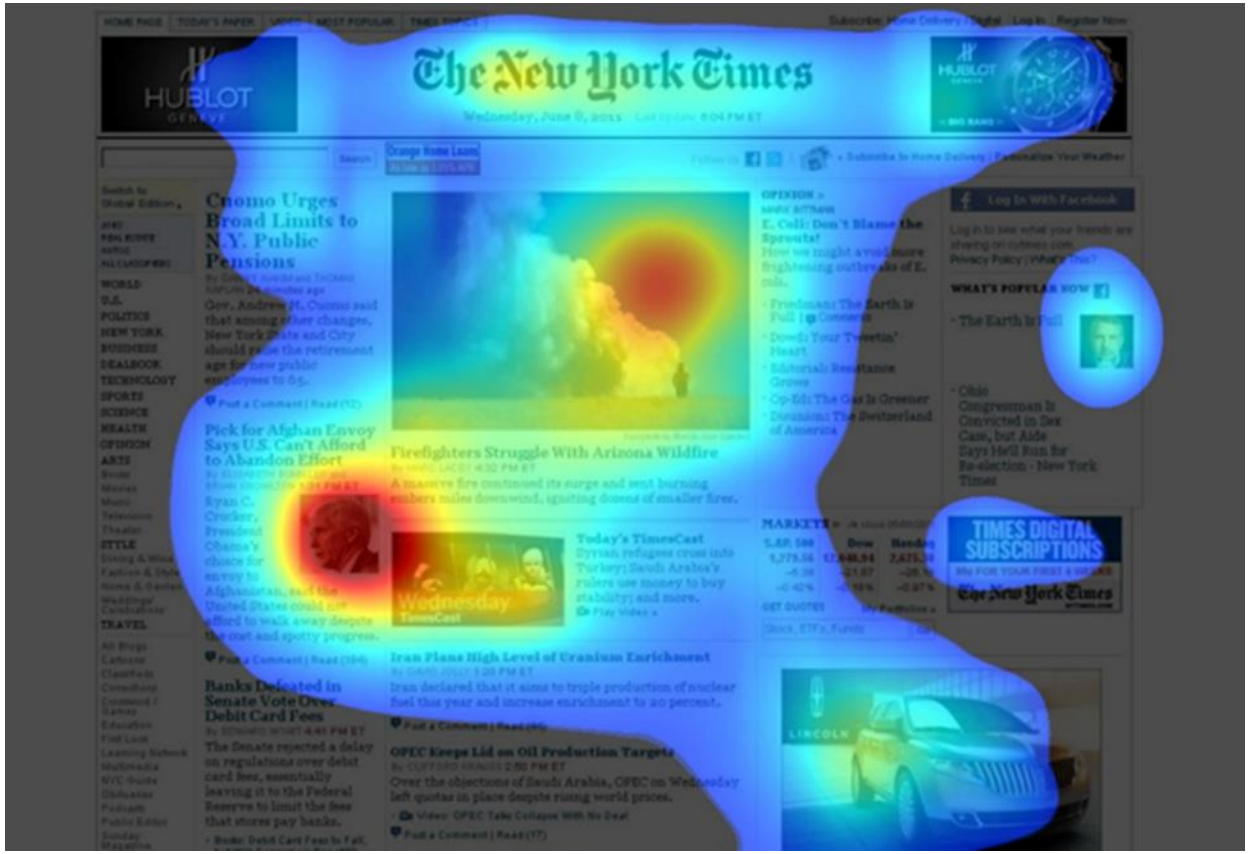
注意力机制——通用近似定理

- 由于优化算法和计算能力的限制，神经网络在实践中很难达到通用近似的能力。
  - 网络不能太复杂（参数太多）
- 如何提高网络能力
  - 局部连接
  - 权重共享
  - 汇聚操作
  - ？

大脑中的注意力

- 人脑每个时刻接收的外界输入信息非常多，包括来源于视觉、听觉、触觉的各种各样的信息。
- 但就视觉来说，眼睛每秒钟都会发送千万比特的信息给视觉神经系统。
- 人脑通过注意力来解决信息超载问题。

注意力示例


The New York Times
HUBLOT
Cuomo Urges Broad Limits to N.Y. Public Pensions
Pick for Afghan Envoy Says U.S. Can't Afford to Abandon Effort
Banks Defeated in Senate Vote Over Debit Card Fees
Firefighters Struggle With Arizona Wildfire
Today's TimesCast
Iran Plans High Level of Uranium Enrichment
OPEC Keeps Lid on Oil Production Targets
TIMES DIGITAL SUBSCRIPTIONS
LINCOLN

如何实现?

- 自下而上

汇聚（pooling）

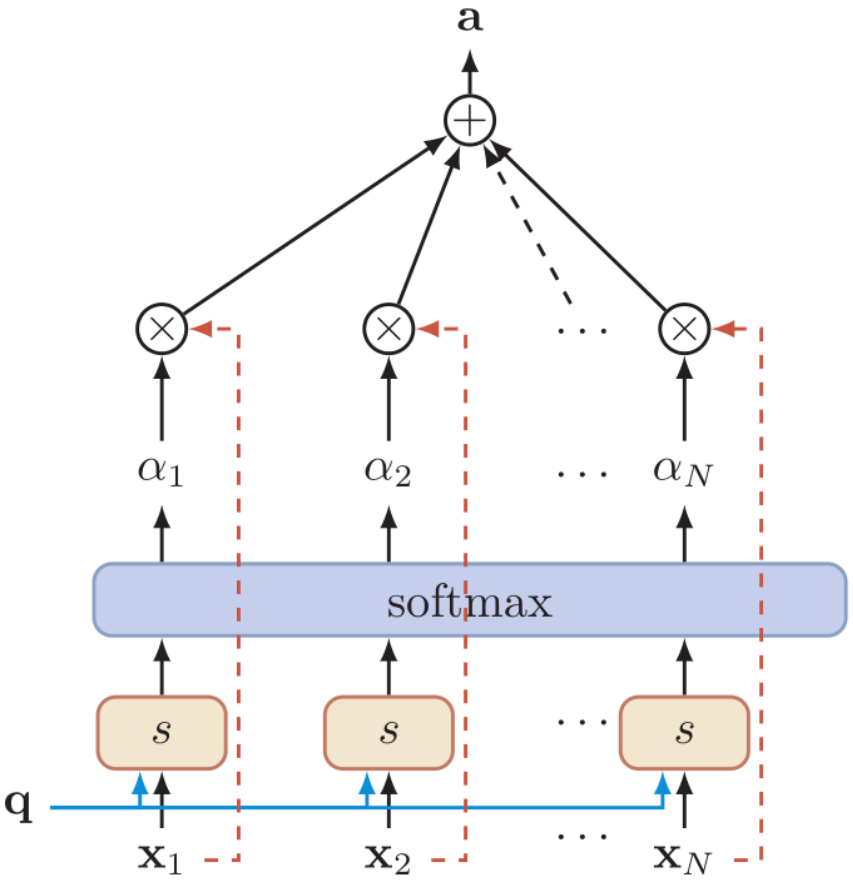
- 自上而下

会聚（focus）

注意力分布

- 给定查询q和输入信息$\mathbf{x}_{1:N}$

$$\begin{aligned}\alpha_i &= p(z=i|\mathbf{x}_{1:N},\mathbf{q}) \\ &= \text{softmax}\Big(s(\mathbf{x}_i,\mathbf{q})\Big) \\ &= \frac{\exp\Big(s(\mathbf{x}_i,\mathbf{q})\Big)}{\sum_{j=1}^{N}\exp\Big(s(\mathbf{x}_j,\mathbf{q})\Big)},\end{aligned}$$



- s($\mathbf{x}_i$,q)为注意力打分函数
  - 加性模型 $s(\mathbf{x}_i,\mathbf{q}) = \mathbf{v}^{\mathrm{T}}\tanh(W\mathbf{x}_i + U\mathbf{q}),$
  - 乘法模型 $s(\mathbf{x}_i,\mathbf{q}) = \mathbf{x}_i^{\mathrm{T}}\mathbf{q},$

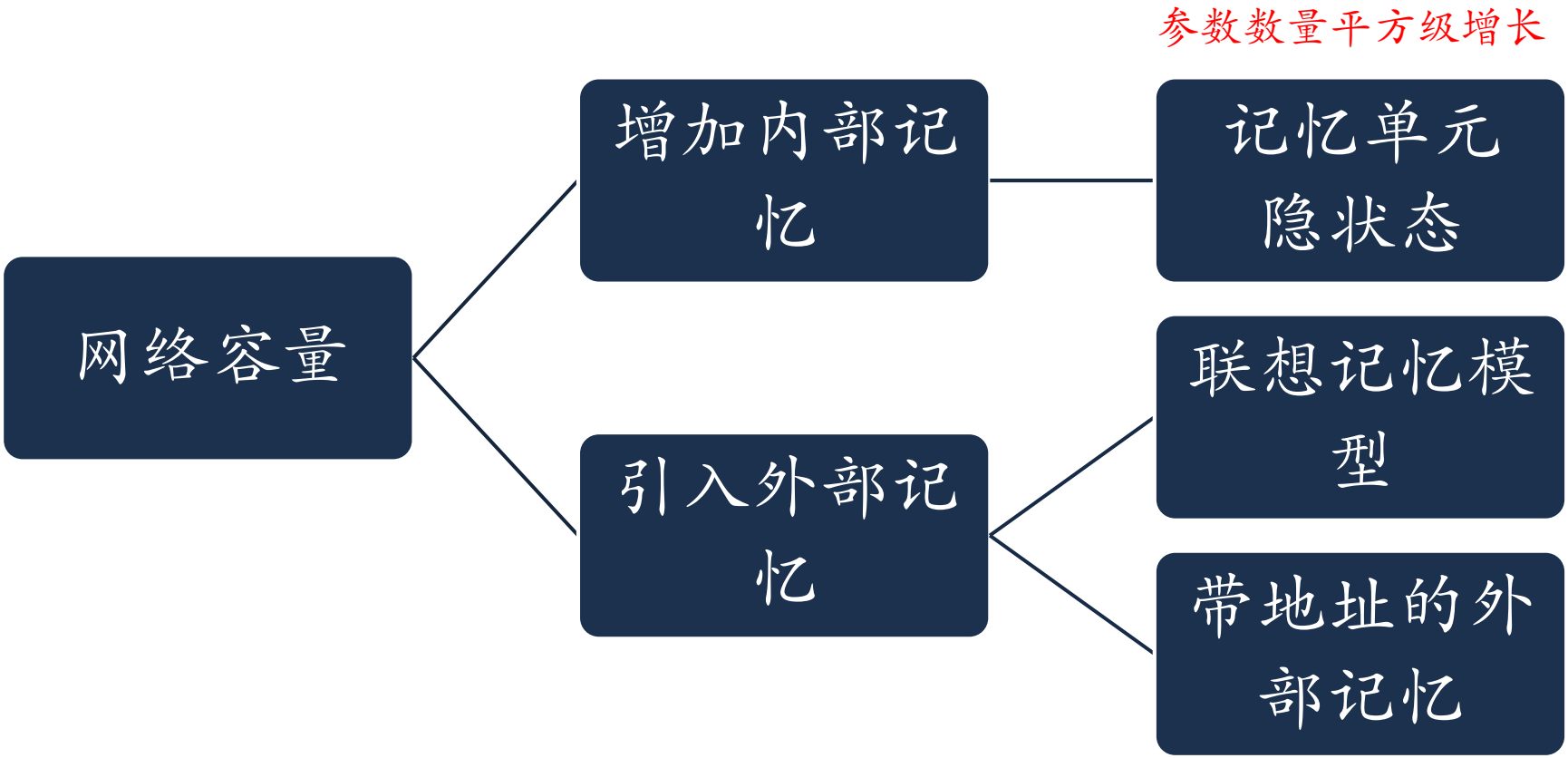
$$s(\mathbf{x}_i,\mathbf{q}) = \mathbf{x}_i^{\mathrm{T}}W\mathbf{q},$$

注意力的变种

- 多头注意力Multi-Head Attention
- 硬注意力 Hard Attention
- 结构化注意力 Structure Attention
- 指针网络 Pointer Network
- 双向注意力Bi-Directional Attention
- 键值对注意力 Key-Value Attention
- 自注意力 Self/Intra Attention
- …

如何增加网络容量?

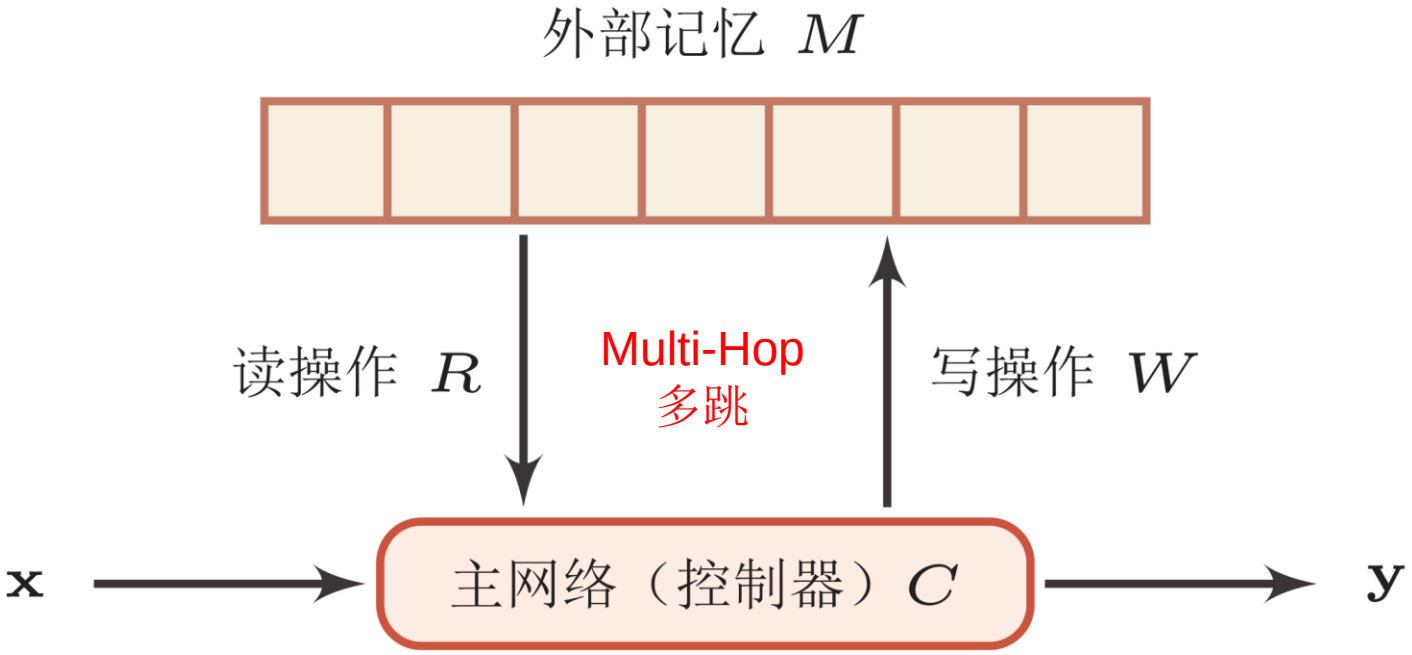
- 以LSTM为例



外部记忆

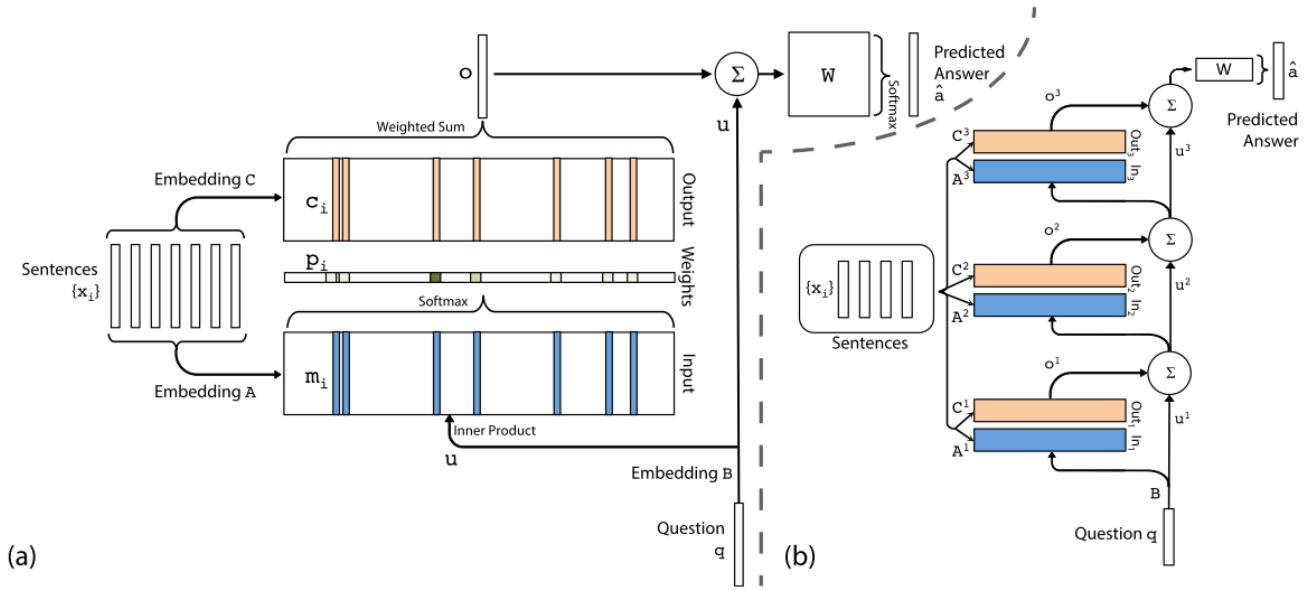
- 外部记忆定义为矩阵$\mathrm{M} \in \mathrm{R}^{\mathrm{d} \times k}$
  - k 是记忆片段的数量，d 是每个记忆片段的大小
- 外部记忆类型
  - 只读
    - Memory Network
    - RNN中的h_t
  - 可读写
    - NTM

记忆网络的结构



按内容寻址：通常利用注意力机制来完成。

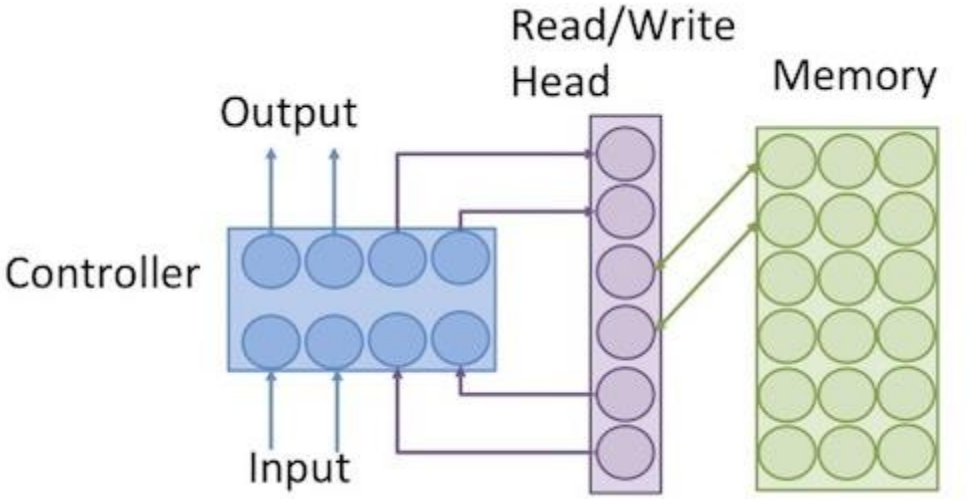
端到端记忆网络



Sukhbaatar, S., Szlam, A., Weston, J., & Fergus, R. (2015). End-To-End Memory Networks, 1–11. http://arxiv.org/abs/1503.08895

神经图灵机

- 组件
  - 控制器
  - 外部记忆
  - 读写操作
- 整个架构可微分



图片来源：
http://cpmarkchang.logdown.com/posts/279710-neural-network-neural-turing-machine

Graves, A., Wayne, G., & Danihelka, I. (2014). Neural Turing Machines. Arxiv, 1–26. http://arxiv.org/abs/1410.5401

不严格的类比

记忆周期	计算机	人脑	神经网络
短期	寄存器	工作记忆	隐状态
中期	内存	情景记忆	外部记忆
长期	外存	结构记忆	可学习参数

13.2 深度学习的常见模型及方法

- 卷积神经网络CNN(Convolutional Neural Networks)

CNN小结

卷积神经网络主要组成及关键方法：

1. 卷积层(Convolutional layer)：卷积运算的目的是提取输入的不同特征，第一层卷积层可能只能提取一些低级的特征如边缘、线条和角等层级，更多层的网络能从低级特征中迭代提取更复杂的特征。

2. 池化层(Pooling)：其实质是一种形式的子采样。有多种不同形式的非线性池化函数，最大池化 (Max pooling)、平均池化 (Average pooling)子采样最为常见。

3. 全连接层(Full connection)： 与传统的神经网络连接方式类似，一般都在最后几层。

Pooling层的作用： Pooling层相当于把一张分辨率较高的图片转化为分辨率较低的图片； Pooling层可进一步缩小最后全连接层中结点的个数，从而达到减少整个神经网络中参数的目的。

与传统的神经网络在图像分析方面比较，卷积神经网络主要优点如下：1. 输入图像和网络的拓扑结构能很好的吻合；2. 特征提取和模式分类同时进行，并同时在训练中产生；3. 权值共享能减少网络的训练参数，使神经网络结构变得更为简单，适应性更强。

还有，CNN结构只是一种参考，实际使用中，每层特征图数目、卷积核大小、池化采样率的多少等都是可变的，这就是所谓的CNN调参，很多时候我们需要灵活多变。

13.3 LeNet数字识别编程举例

卷积神经网络 (CNN)是分析图像等多维信号的一种最优技术。目前已有许多深度学习框架可以实现深度学习，包括实现CNN。目前流行的深度学习框架有TensorFlow/Keras、PyTorch(Caffee)、MXNet、MindSpore、PaddlePaddle等。这些深度学习框架的库提供了抽象的 API，因此能大大降低开发难度，并避免实现的复杂度。

LeNet数字识别编程举例： 一种用于数字识别的LeNet网络模型见图所示。基于Anaconda3 +Keras2.3.1+TensorFlow2.0 Backend编程环境， 现采用图中的LeNet网络模型编制MNIST数字识别程序。

LeNet Network Topology

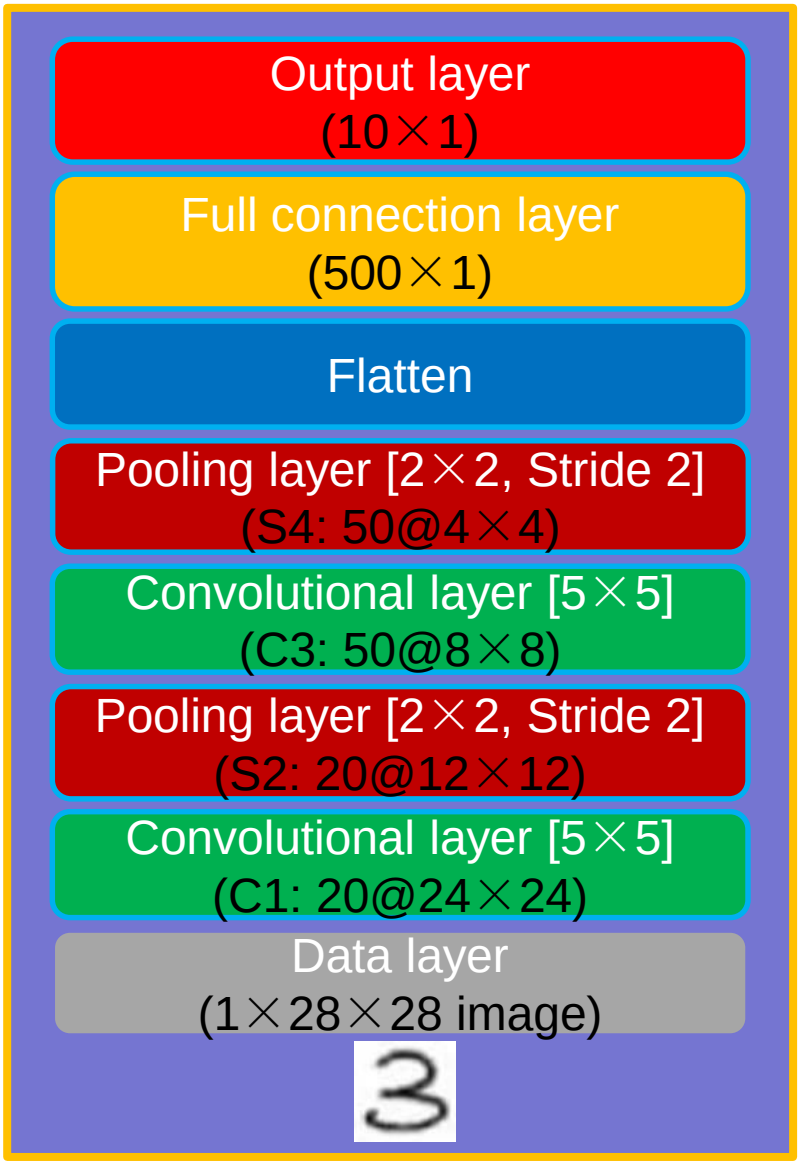


图　用于MNIST数字识别的一种LeNet网络拓扑结构

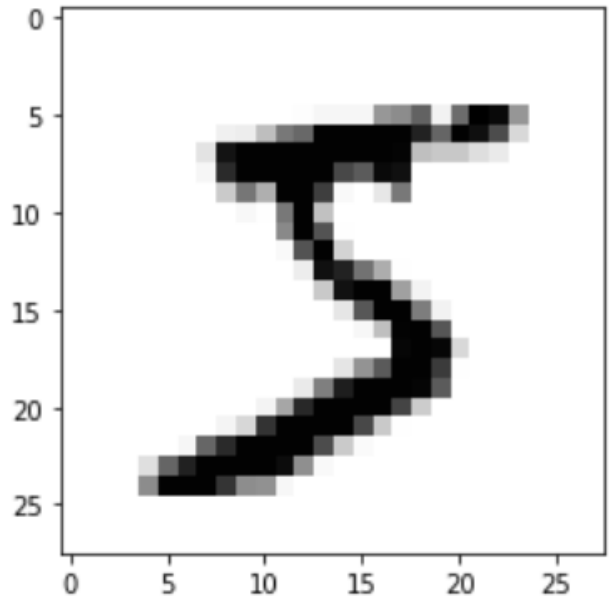
Keras+TensorFlow Backend程序清单：

In [15]:
```
import numpy as np # 导入NumPy数学工具箱
import matplotlib.pyplot as plt # 导入绘图工具包
import tensorflow as tf
from tensorflow.keras.datasets import mnist #从Keras中导入mnist数据集
from tensorflow.keras.utils import to_categorical # 导入keras.utils工具箱的类别转换工具
#读入训练集和测试集
path = "d:\exp_keras_dataset/mnist.npz" # 将mnist数据集文件下载到本地,path为mnist数据集存储的实际路径
(X_train_image, y_train_lable), (X_test_image, y_test_lable)=tf.keras.datasets.mnist.load_data(path) #法3：使用keras的load_data加载
X_train = X_train_image.reshape(-1, 28, 28, 1) #-1表示自动推导
X_test = X_test_image.reshape(-1, 28, 28, 1)
X_train = X_train/255    # 训练集X_train归一化(normalization)
X_test = X_test/255      # 测试集X_test归一化(normalization)
y_train = to_categorical(y_train_lable, 10) # 训练集y_train_lable类标签转换为独热编码
y_test = to_categorical(y_test_lable, 10) # 测试集y_test_lable类标签转换为独热编码
```

In [16]:
```
print ("第一个数据样本的标签：", y_train_lable[0])
print ("数据集张量形状：", X_train.shape) # 数据集张量的形状
print ("第一个数据标签：",y_train[0]) # 输出训练标签集中第一个数据标签的独热编码(One-Hot Encoding)
plt.imshow(X_train_image[0].reshape(28, 28),cmap='Greys') # 输出该图像
```

```
第一个数据样本的标签： 5
数据集张量形状： (60000, 28, 28, 1)
第一个数据标签： [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

Out[16]:
```
<matplotlib.image.AxesImage at 0x23bf10d0d68>
```



In [17]:
```
from tensorflow.keras import models # 导入Keras模型
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten # 导入LeNet网络中要使用的层
model = models.Sequential() # 用序贯方式建立模型
model.add(Conv2D(input_shape=(28, 28, 1), kernel_size=(5, 5), filters=20, activation='relu')) #C1: Convolutional Layer
model.add(MaxPooling2D(pool_size=(2,2), strides=2, padding='same'))  #S2: Subsampling/Pooling Layer
model.add(Conv2D(kernel_size=(5, 5), filters=50,  activation='relu', padding='same')) #C3: Convolutional Layer
model.add(MaxPooling2D(pool_size=(2,2), strides=2, padding='same'))  #S4: Subsampling/Pooling Layer
model.add(Flatten())       #Flatten(平整化)
model.add(Dense(500, activation='relu'))   #Full connection layer/Dense layer(致密层): Feedforward Neural Network
model.add(Dense(10, activation='softmax')) #Output layer: Softmax
```

In [18]:
```
# 编译模型
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])#指定优化器、损失函数及评估指标
print('训练')
model.fit(X_train, y_train, epochs=2, batch_size=32)
print('\n测试')
loss, accuracy = model.evaluate(X_test, y_test)
print('\n测试损失率(Loss): ', loss)
print('\n测试准确率(Accuracy): ', accuracy)
```

```
训练
Train on 60000 samples
Epoch 1/2
60000/60000 [==============================] - 62s 1ms/sample - loss: 0.0984 - accuracy: 0.9691
测试损失率(Loss):  0.03456440292959139
                                    =======] - 63s 1ms/sample - loss: 0.0378 - accuracy: 0.9890s - loss: 0.0380 - accu
测试准确率(Accuracy):  0.9912
```

下面对这里代码中相关知识做进一步的说明。

损失函数说明：

损失函数(Loss function)又称成本函数、代价函数、目标函数或优化函数；损失函数不仅以衡量模型预测值与真实值偏离的程度，而且通过求解损失函数的最小值可实现求解模型参数、优化模型参数和评价模型参数学习效果的目的。

常用的两种损失函数：

1.均方误差损失函数：某样本x_i预测值y_i与其真实值d_i差值的平方定义为单样本损失$L(y_i,d_i)=(y_i-d_i)^2$，评估m个样本的损失用全部损失的均方误差损失表示为$J(w,b) = 1/m*Sum[L(y_i,d_i)]$

2.交叉熵损失函数：

以逻辑回归为例，单样本交叉熵损失函数：

$$L(y_i,d_i) = -[d_i\log(y_i)+(1-d_i)\log(1-y_i)]$$

M个样本的交叉熵损失函数：

$$J(w,b) = -1/m*Sum[d_i\log(y_i)+(1-d_i)\log(1-y_i)]$$

在以上两种损失函数中，均方误差损失函数常用于回归问题，交叉熵损失函数一般用于分类问题。各种流行的机器学习框架预定义了多种损失函数，实践中可灵活选择，也可根据问题需要自定义损失函数。

梯度下降法是贯穿神经网络的算法灵魂，根据样本数据参加训练的方式，可分为3种：

1.(批量)梯度下降[(B)GD, (Batch) Gradient Descent]：全部样本一起完成一次正向与反向传播，即一次梯度下降；该方法的优点是所有样本共同决定梯度的方向，可以用最少的迭代步数逼近最优值，缺点是一次性装入过多的样本，对内存的需求很大，对算力要求很高。

2.随机梯度下降(SGD, Stochastic Gradient Descent)：一次用一个样本完成一次正向与反向传播；该方法的优点是单个样本决定梯度的方向，适合在线学习，计算速度快，内存需求小，缺点是单个样本决定梯度的方向需过度依赖样本的质量，容易导致下降的方向飘忽不定，需要更多的迭代，而且难于逼近最优值。

3.小批量梯度下降(MBGD，Mini-Batch Gradient Descent)：是前两种梯度下降法的改进，将整个训练集随机划分为若干不同的组，每次输入一组数据，一次用一组数据完成一次正向与反向传播，既可确保梯度下降的方向，又可降低对内存与算力过高的要求

梯度下降法的几种优化算法说明：

1.Momentum梯度下降法：是对Mini-Batch梯度下降法的一种改进，在Mini-Batch完成单步梯度计算后不立即更新参数，而是用移动加权平均思想计算当前梯度的移动平均值v_{dw}和v_{db}去更新w和b。

2.RMSprop梯度下降法：亦采用移动加权平均思想，平滑梯度纵向的波动，加快横向的收敛速度；与Momentum梯度下降法不同的是，其移动平均的计算采用dw^2和db^2，参数更新的策略也有变化，分别用dw和db除以各自的移动均方根，为了避免分母为0，分母增加了一个ε调节项；

3.Adam梯度下降法：是对Momentum和RMSprop两种方法的综合改进，兼顾了二者的优点，该方法同时用Momentum和RMSprop方法的移动平均平滑各个方向的梯度，并进行移动平均修正，在参数更新阶段，分子用Momentum的移动平均值，分母用RMSprop均方根，实践中Adam往往优于前两种方法，但Adam算法计算量要大一些。

In [19]:
```
print("预测概率：",model.predict(X_test))
print("预测答案：",model.predict_classes(X_test))
```

```
预测概率： [[2.91118892e-15 6.38629124e-13 1.07206063e-10 ... 1.00000000e+00
  1.45517533e-14 7.61763708e-11]
 [7.16359125e-11 1.60977121e-10 1.00000000e+00 ... 1.65387312e-16
  1.01864003e-12 3.30024233e-16]
 [5.05917519e-13 1.00000000e+00 1.20307861e-10 ... 3.20198618e-10
  1.71808114e-08 7.20054658e-12]
 ...
 [8.17591548e-22 9.54555770e-13 2.87375945e-17 ... 1.22167926e-14
  5.52432960e-11 1.20560246e-11]
 [3.75782737e-15 2.50745655e-16 9.20359536e-20 ... 3.75495382e-19
  7.62491581e-08 4.00250681e-15]
 [3.42298412e-11 3.00493103e-14 3.36671283e-13 ... 1.56065758e-21
  1.65191021e-11 2.12200266e-17]]
预测答案： [7 2 1 ... 4 5 6]
```

In [20]:
```
from sklearn.metrics import confusion_matrix
pre=model.predict_classes(X_test)
confusion_matrix(y_test_lable,pre)  #使用sklearn输出混淆矩阵
```

Out[20]:
```
array([[ 975,    0,    0,    0,    0,    0,    3,    1,    1,    0],
       [   0, 1131,    1,    1,    0,    2,    0,    0,    0,    0],
       [   2,    0, 1029,    0,    0,    0,    0,    1,    0,    0],
       [   1,    1,    1,  995,    0,   10,    0,    1,    1,    0],
       [   0,    0,    1,    0,  975,    0,    1,    0,    2,    3],
       [   1,    0,    0,    1,    0,  888,    1,    1,    0,    0],
       [   0,    2,    0,    0,    1,    2,  953,    0,    0,    0],
       [   0,    5,    7,    1,    0,    1,    0, 1010,    1,    3],
       [   3,    0,    1,    0,    0,    2,    0,    0,  967,    1],
       [   2,    1,    0,    1,    4,    9,    1,    1,    1,  989]],
      dtype=int64)
```

In [21]:
```
import pandas as pd # 导入Pandas数据处理工具箱
pre=model.predict_classes(X_test)
pd.DataFrame(confusion_matrix(y_test_lable,pre))    #使用Pandas输出混淆矩阵
```

Out[21]:

	0	1	2	3	4	5	6	7	8	9
0	975	0	0	0	0	0	3	1	1	0
1	0	1131	1	1	0	2	0	0	0	0
2	2	0	1029	0	0	0	0	1	0	0
3	1	1	1	995	0	10	0	1	1	0
4	0	0	1	0	975	0	1	0	2	3
5	1	0	0	1	0	888	1	1	0	0
6	0	2	0	0	1	2	953	0	0	0
7	0	5	7	1	0	1	0	1010	1	3
8	3	0	1	0	0	2	0	0	967	1
9	2	1	0	1	4	9	1	1	1	989

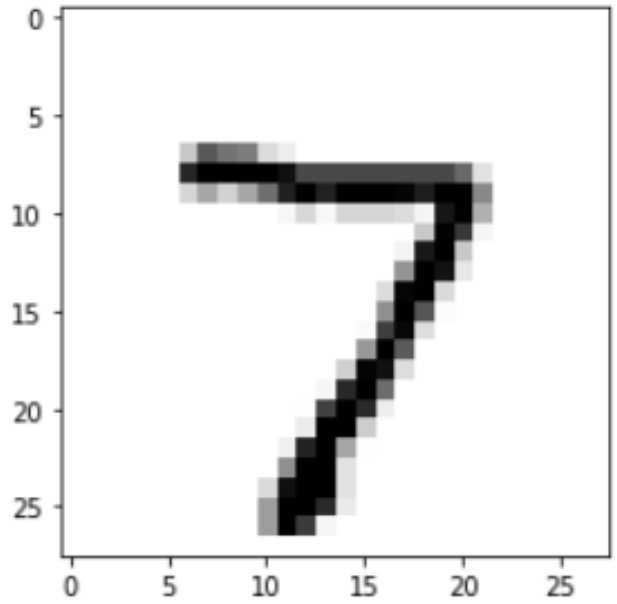
In [22]:
```
false_Index=np.nonzero(pre!=y_test_lable)[0]
print("预测错误的数字个数 =",len(false_Index))
```

```
预测错误的数字个数 = 88
```

In [23]:
```
pred = model.predict((X_test[0]).reshape(1, 28, 28, 1))
print(pred[0],"转换一下格式得到：",pred.argmax()) # 把one-hot编码转换为数字
import matplotlib.pyplot as plt # 导入绘图工具包
plt.imshow(X_test[0].reshape(28, 28),cmap='Greys') # 输出这幅图像
```

```
[2.91121116e-15 6.38627877e-13 1.07206063e-10 1.31693658e-12
 7.70555534e-17 5.49057768e-15 2.06553989e-22 1.00000000e+00
 1.45517533e-14 7.61766622e-11] 转换一下格式得到： 7
```

Out[23]:
```
<matplotlib.image.AxesImage at 0x23bcc08d4a8>
```



13.5 深度学习展望

未来需解决的问题：

- 对于一个特定的框架，多少维的输入它可以表现得较优？
- 对捕捉短时或者长时间的时间依赖，哪种架构才是有效的？
- 如何对于一个给定的深度学习架构，融合多种感知的信息？
- 如何分辨和利用学习获得的中、高层特征语义知识？
- 有什么正确的机理可以去增强一个给定的深度学习架构，以改进其鲁棒性和对变形及数据丢失的不变性？
- 模型方面是否有其他更为有效且有理论依据的深度模型学习算法？
- 是否存在更有效的可并行训练算法？

End of This Leture!