前沿技术讲习班
Advanced Technology Tutorial

自然语言处理
基于预训练模型的方法（上）

车万翔

社会计算与信息检索研究中心
哈尔滨工业大学

2021年7月22日

目 录

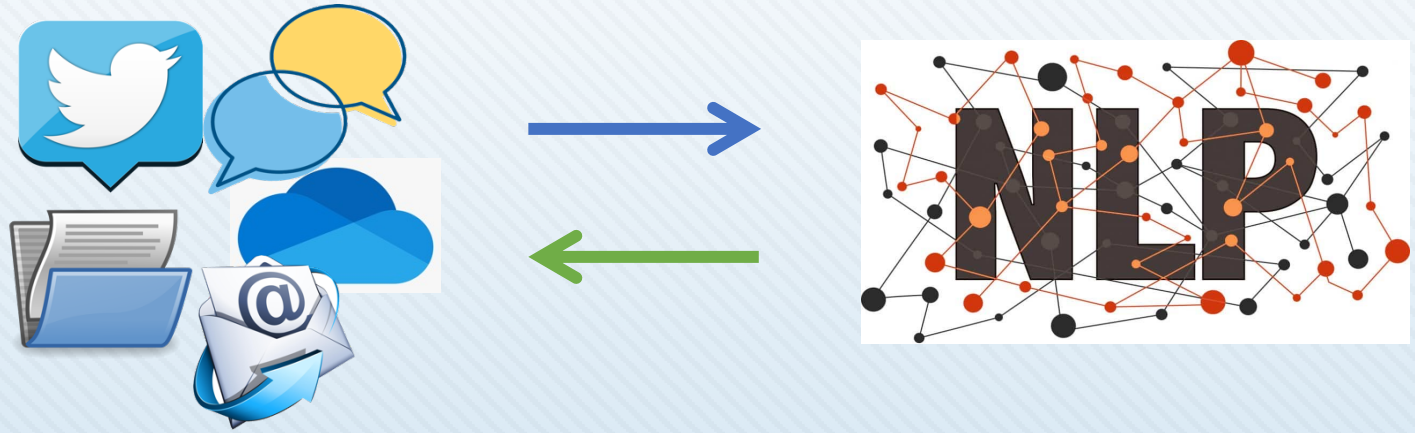
CONTENTS

目 录

CONTENTS

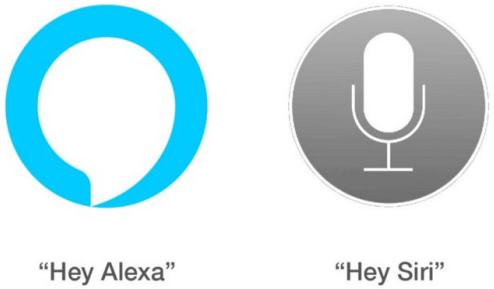
什么是自然语言处理？

- 语言是**思维的载体**，是人类交流思想、表达情感最自然、最方便的工具
  - 人类历史上大部分知识是以语言文字形式记载和流传的
- 自然语言处理（Natural Language Processing，NLP）
  - 用计算机来**理解**和**生成**自然语言的各种理论和方法
  - 自然语言指的是人类语言，特指**文本符号**，而非语音信号



自然语言处理的代表性应用

机器翻译

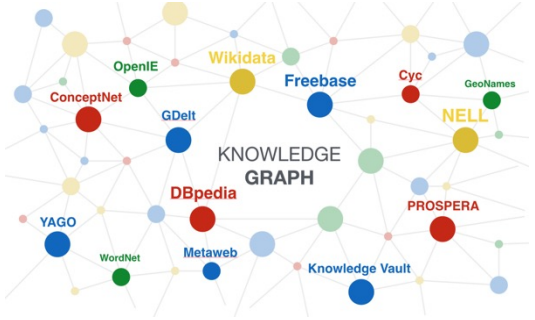


智能助手

文本校对

舆情分析

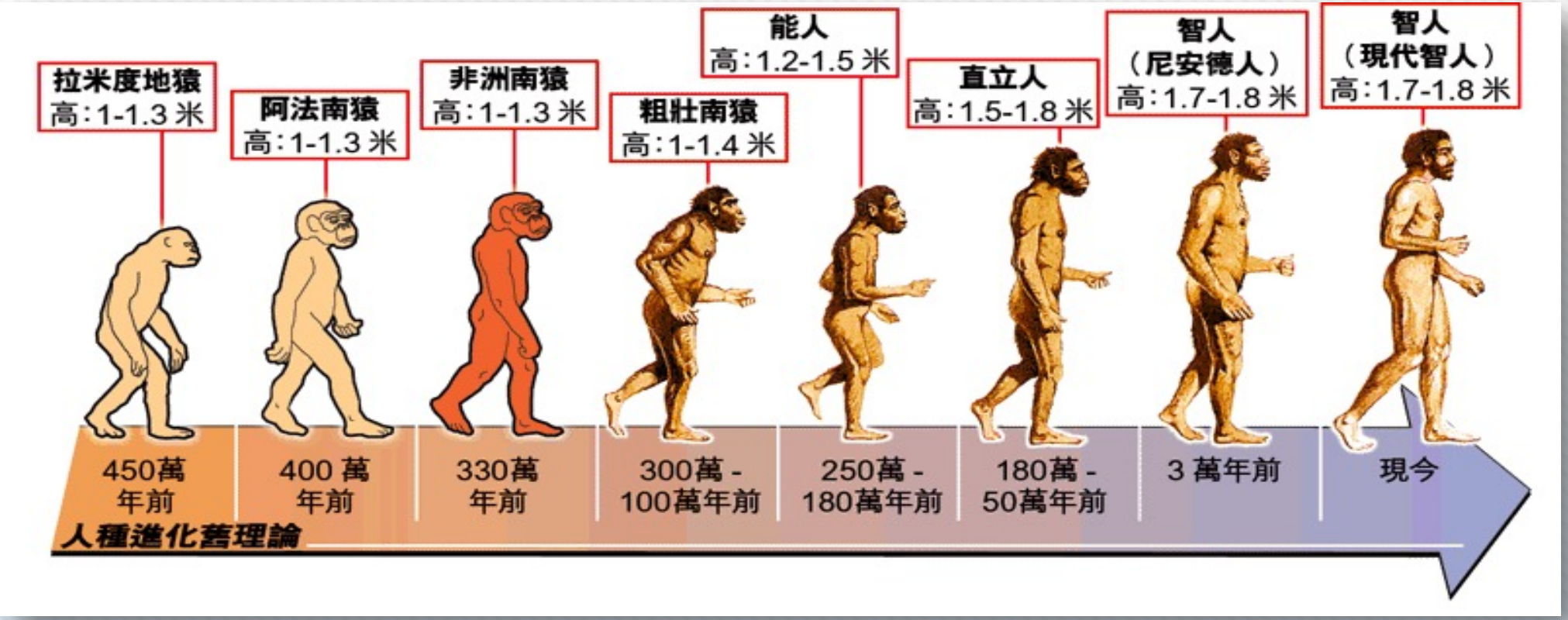
智能教育



知识图谱

自然语言处理属于认知智能任务

- **认知智能**是人类和动物的主要区别之一
  - 需要更强的抽象和推理能力



自然语言处理的难点与特点

领导：“*你这是什么意思？*”

阿呆：“*没什么意思，意思意思。*”

领导：“*你这就不够意思了。*”

阿呆：“*小意思，小意思。*”

领导：“*你这人真有意思。*”

阿呆：“*其实也没有别的意思。*”

领导：“*那我就不好意思了。*”

阿呆：“*是我不好意思。*”

“人工智能皇冠上的明珠”

- 自然语言处理成为**制约人工智能取得更大突破和更广泛应用的瓶颈**

“深度学习的下一个前沿课题是自然语言理解”

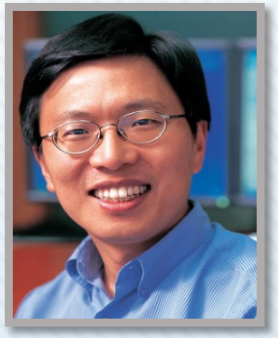
Yann LeCun
图灵奖得主、Facebook AI 负责人

“深度学习的下一个大的进展应该是让神经网络真正理解文档的内容”

Geoffrey Hinton
图灵奖得主、深度学习之父

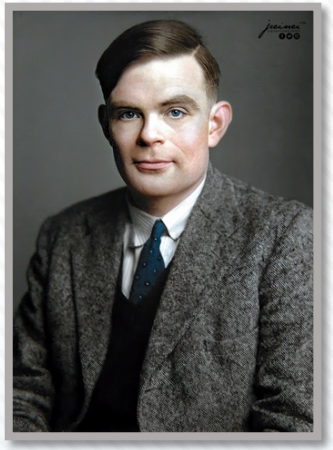
“如果给我10亿美金，我会用这10亿美金建造一个NASA级别的自然语言处理研究项目”

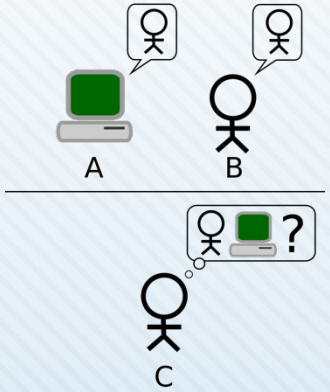
Michael I. Jordan
美国双院院士、世界知名机器学习专家



“下一个十年，懂语言者得天下”

沈向洋
美国工程院院士、微软前全球执行副总裁



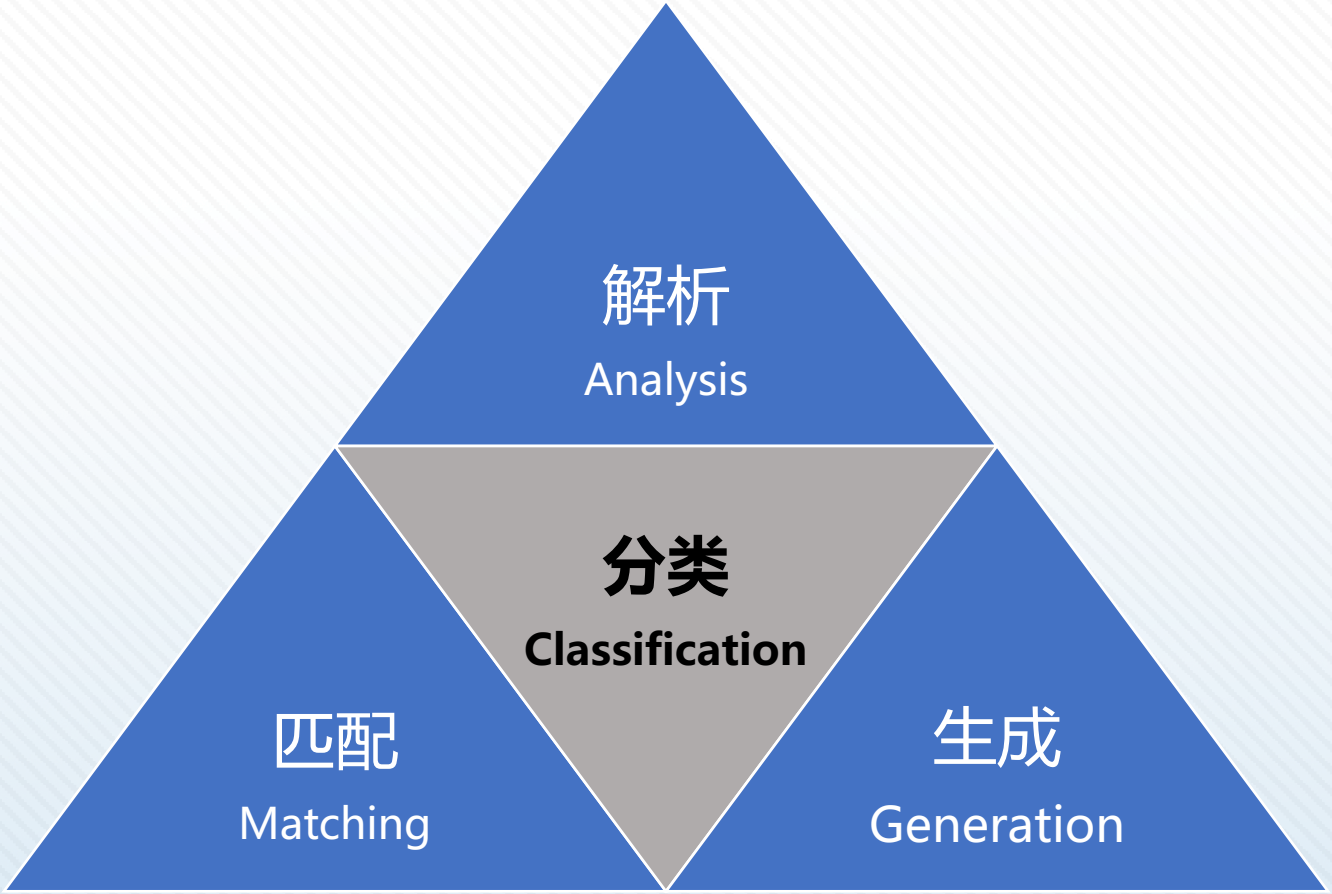


图灵测试

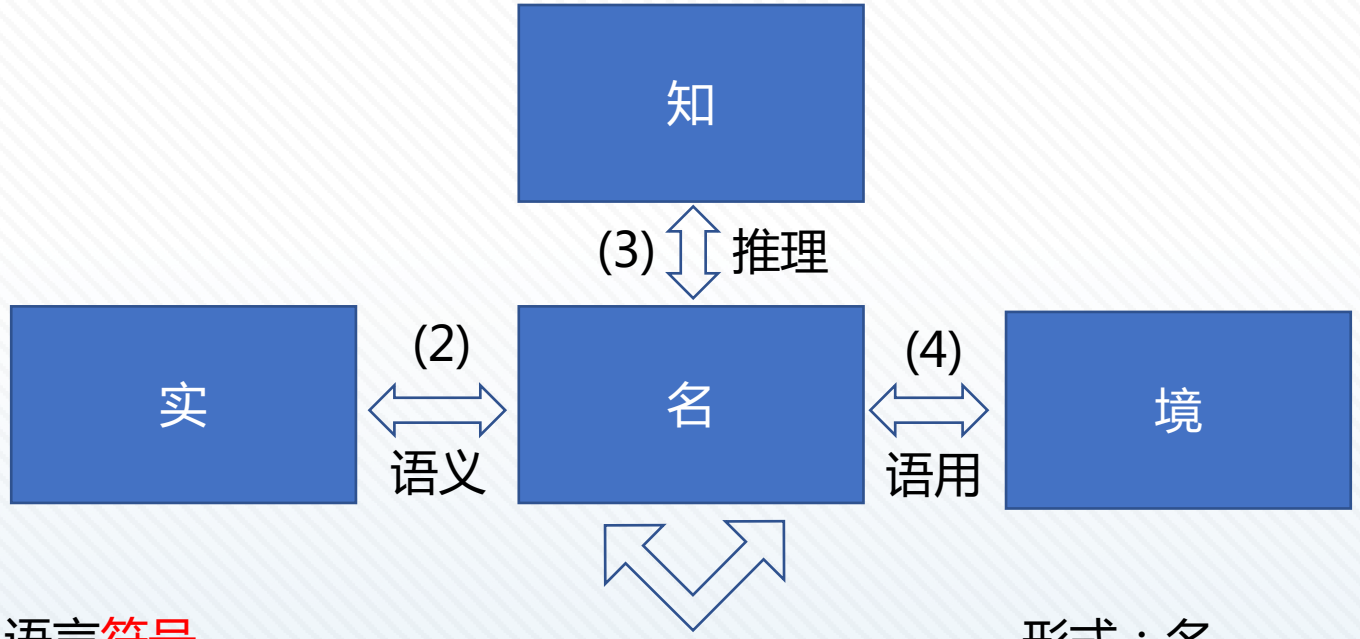
自然语言处理任务层级


应用系统 (NLP+)
• 教育，医疗，司法，金融，机器人等
应用任务
• 信息抽取，情感分析，机器翻译，对话系统等
基础任务
• 分词，词性标注，句法分析，语义分析等
资源建设
• 语言学知识库建设，语料库资源建设等

自然语言处理任务类别



自然语言处理研究对象与层次



名：语言符号
实：客观事实、主观意见
知：知识
境：语言所处环境

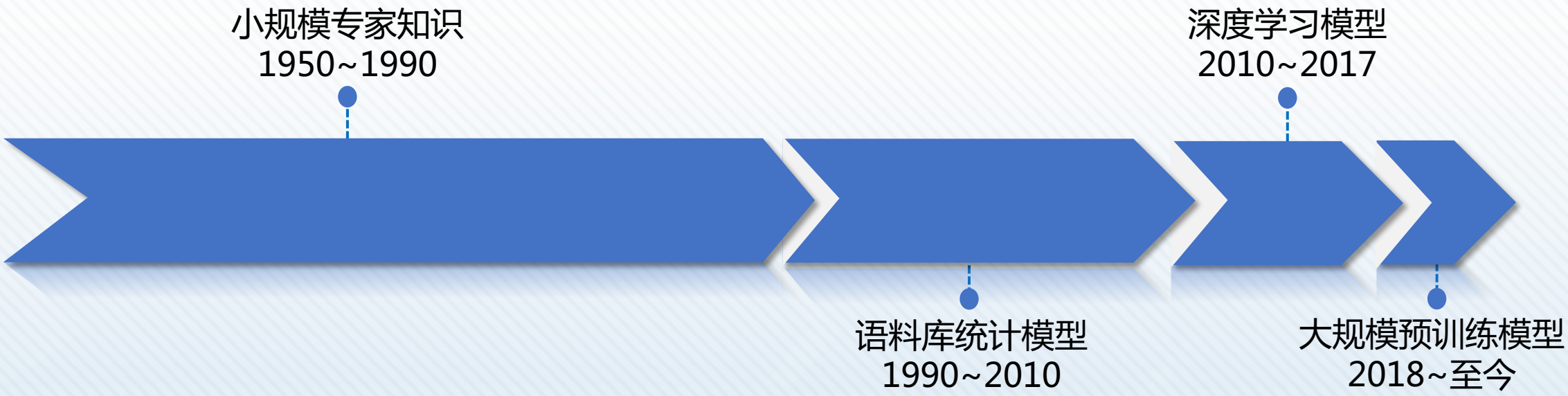
形式：名
语义：名+实
推理：名+实+知
语用：名+实+知+境

“层次 x 任务”二维表

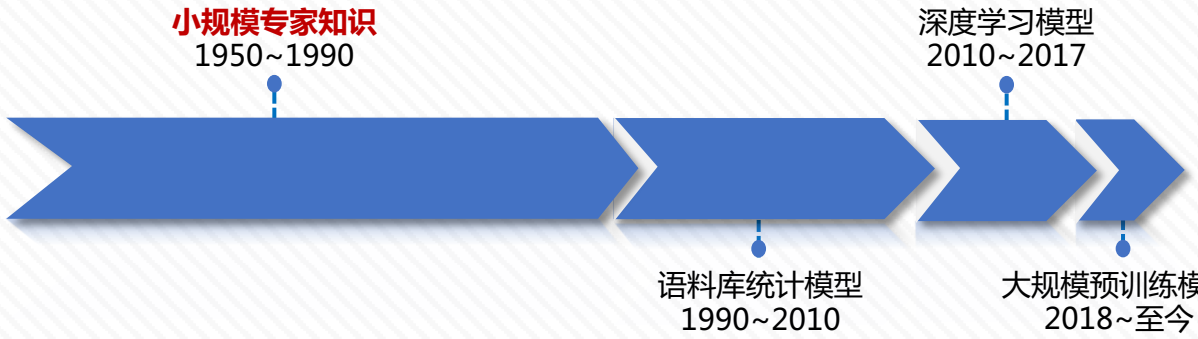
	分类	解析	匹配	生成
形式	文本分类	词性标注 句法分析	搜索	机械式文摘
语义	情感分析	命名实体识别 语义角色标注	问答	机器翻译
推理	隐式情感分析		文本蕴含	写故事结尾
语用	反语			聊天

自然语言处理技术发展历史

- 语义在计算机内部是如何**表示**的？
- 根据表示方法的不同，自然语言处理技术经历了**四次范式变迁**



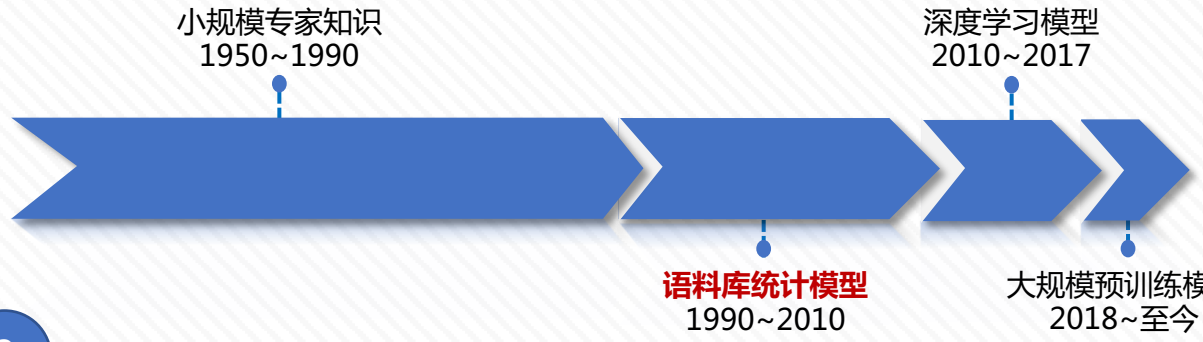
基于符号（字符串）表示的专家知识

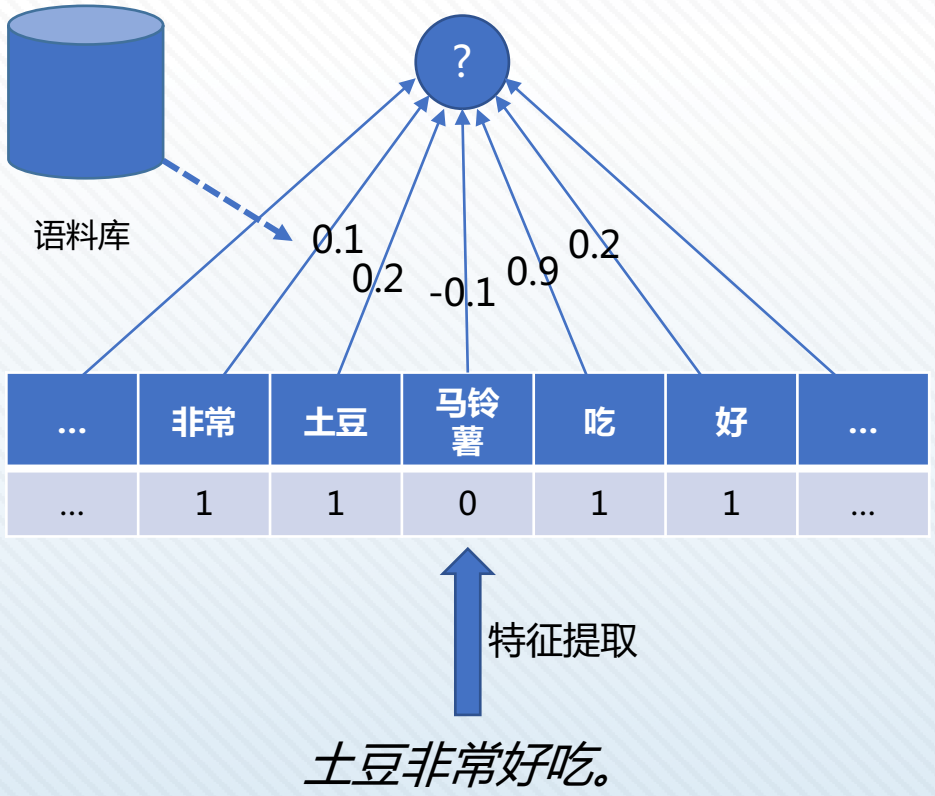


- “*土豆非常好吃。*”的情感倾向性？
  - 如果：出现褒义词“好”“喜欢”等
  - 那么：结果为褒义
  - 如果：出现“不”
  - 那么：结果倾向性取反

- 优点
  - 符合人类的直觉
  - 可解释、可干预性好
- 缺点
  - 知识完备性不足
  - 需要专家构建和维护
  - 不便于计算

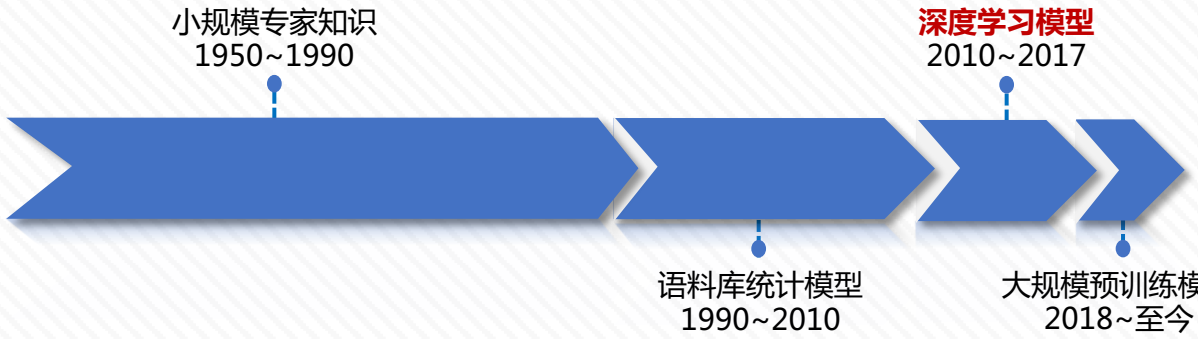
基于向量表示的统计模型





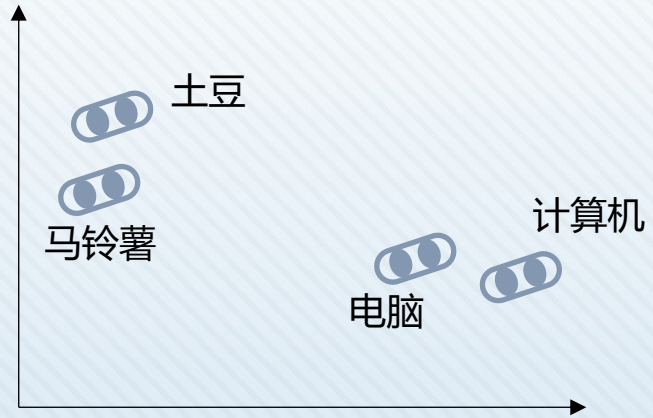
- 使用**高维、离散、稀疏**的向量表示词
  - 维度为词表大小，其中只有一位为1，其余为0
    - **土豆**：[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...]
    - **马铃薯**：[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...]
- 缺点
  - 无法处理“**多词一义**”的现象

基于嵌入表示的深度学习模型



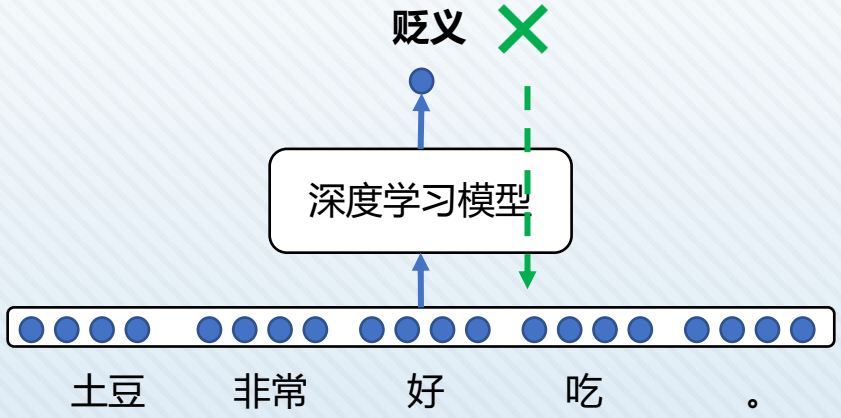
词嵌入（Word Embedding）

- 直接使用一个**低维、连续、稠密**的向量表示词（Bengio等2003）

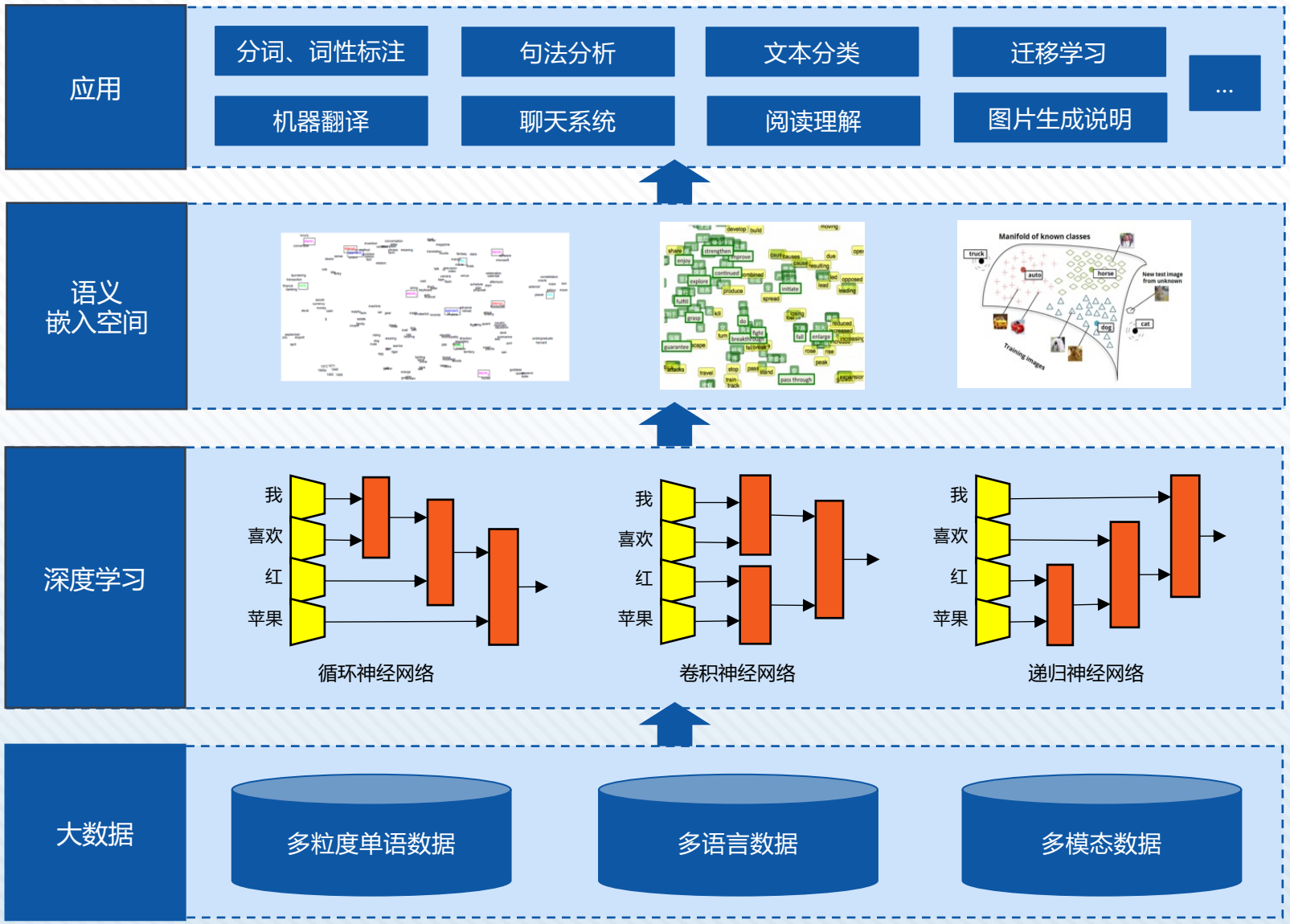


词嵌入表示的赋值方法

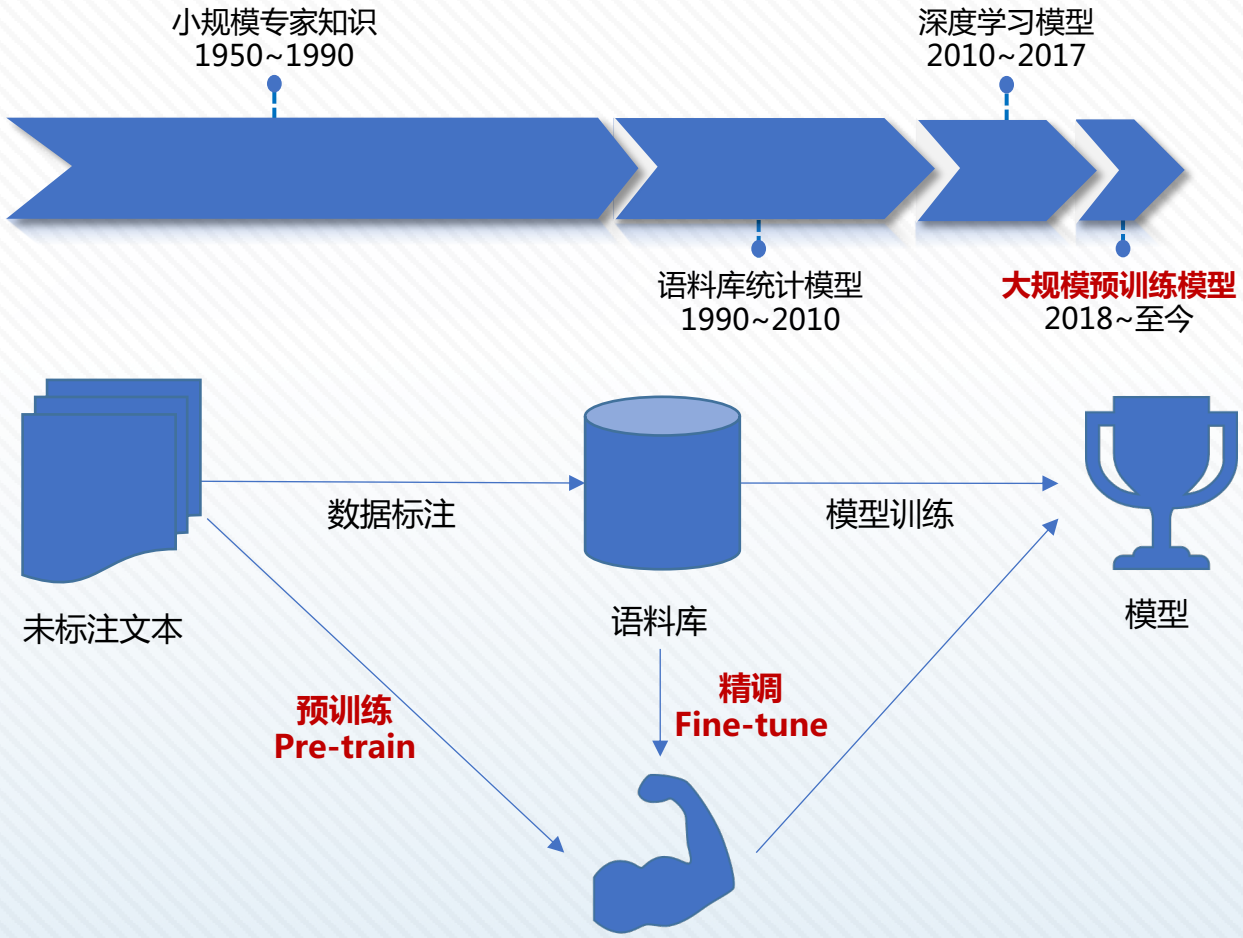
- 通过优化在**下游任务**上的表现自动学习



各种语言单元的统一嵌入表示



预训练模型获得更好的表示



预训练 + 精调 = 自然语言处理新范式

目 录

CONTENTS

语言模型

- 语言模型（Language Model，LM）
  - 描述一段自然语言的概率或给定上文时下一个词出现的概率
    - $P(w_1, \cdots w_l)$，$P(w_{l+1}|w_1, \cdots w_l)$
  - 以上两种定义等价（链式法则）

$$P(w_1 w_2 \cdots w_l) = P(w_1)P(w_2|w_1)P(w_3|w_1 w_2)\cdots P(w_l|w_1 w_2 \cdots w_{l-1})$$

$$= \prod_{i=1}^{l} P(w_i|w_{1:i-1})$$

- 广泛应用于多种自然语言处理任务
  - 机器翻译（词排序）
    - P(the cat is small) > P(small the is cat)
  - 语音识别（词选择）
    - P(there are **four** cats) > P(there are **for** cats)

N元语言模型

- 使用最大似然估计概率

$$P(w_i|w_{1:i-1}) = \frac{C(w_{1:i})}{C(w_{1:i-1})}$$

- 随着i的增加，$C(w_{1:i})$变小甚至为0
- 马尔科夫假设（Markov Assumption）
  - 下一个词出现的概率只依赖于它前面n-1个词

$$P(w_i|w_{1:i-1}) \approx P(w_i|w_{i-(n-1):i-1})$$

  - 又被称为*N*元语言模型
- 语言模型的评价指标：困惑度（Perplexity，PPL）
  - 模型分配给测试集中每一个词的概率的几何平均值的倒数，越小越好

中文分词

- 词（Word）
  - 最小的能独立使用的音义结合体
  - 以汉语为代表的汉藏语系，以阿拉伯语为代表的闪-含语系中不包含明显的词之间的分隔符
- 中文分词是将中文字序列切分成一个个单独的词
- 分词的歧义
  - 如：严守一把手机关了
    - 严守一/ 把/ 手机/ 关/ 了
    - 严守/ 一把手/ 机关/ 了
    - 严守/ 一把/ 手机/ 关/ 了
    - 严守一/ 把手/ 机关/ 了
    - ……

子词切分（Subword）

- 以英语为代表的印欧语系语言，是否需要进行分词？
- 这些语言词形变化复杂
  - 如：*computer*、*computers*、*computing*等
- 仅用空格切分的问题
  - 数据稀疏
  - 词表过大，降低处理速度
- 子词切分
  - 将一个单词切分为若干连续的片段（子词）
  - 方法众多，基本原理相似
    - 使用尽量长且频次高的子词对单词进行切分

字节对编码（Byte Pair Encoding，BPE）

Algorithm 1: BPE 中子词词表构造算法

Input: 大规模生文本语料库；期望的子词词表大小 L

Output: 子词词表

将语料库中每个单词切分成字符作为子词;

用切分的子词构成初始子词词表;

while 子词词表小于等于 L **do**

  在语料库中统计单词内相邻子词对的频次;

  选取频次最高的子词对，合并成新的子词;

  将新的子词加入子词词表;

  将语料库中不再存在的子词从子词词表中删除;

end

```
语料库: {'l o w e r </w>': 2, 'n e w e s t </w>': 6, 'w i d e s t </w>': 3}

初始子词词表: {'l', 'o', 'w', 'e', 'r', '</w>', 'n', 's', 't', 'i', 'd'}

合并子词词表: {'l', 'o', 'w', 'e', 'r', '</w>', 'n', 't', 'i', 'd', 'es'}

语料库: {'l o w e r </w>': 2, 'n e w es t </w>': 6, 'w i d es t </w>': 3}

合并子词词表: {'l', 'o', 'w', 'e', 'r', '</w>', 'n', 'i', 'd', 'est'}

语料库: {'l o w e r </w>': 2, 'n e w est </w>': 6, 'w i d est </w>': 3}
```

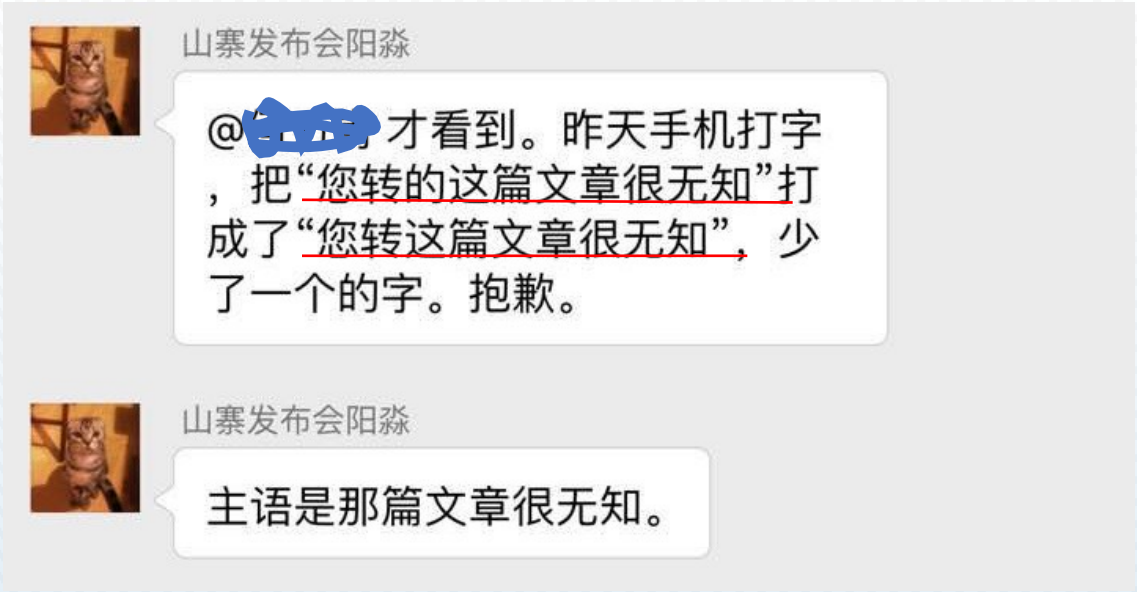
子词词表构造示例

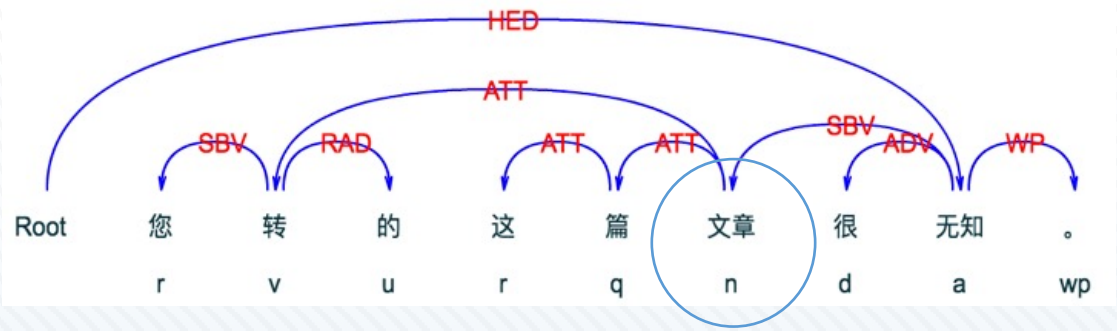
字节对编码

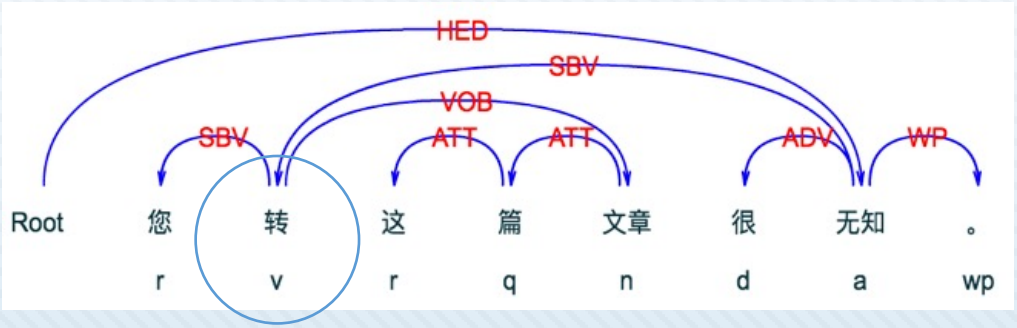
- BPE子词切分算法
  1. 将子词词表按照子词的长度由大到小进行排序
  2. 从前向后遍历子词词表，依次判断一个子词是否为单词的子串
  3. 如果是则将该单词进行切分，然后继续向后遍历子词词表
  4. 如果子词词表全部遍历结束，单词中仍然有子串没有被切分，那么这些子串一定为低频串，则使用统一的标记，如'<UNK>'进行替换
- 更多子词切分算法
  - WordPiece
  - Unigram Language Model（ULM）
  - SentencePiece
    - https://github.com/google/sentencepiece

句法分析

- 分析句子的句法成分，如主谓宾定状补等
- 将词序列表示的句子转换成树状结构

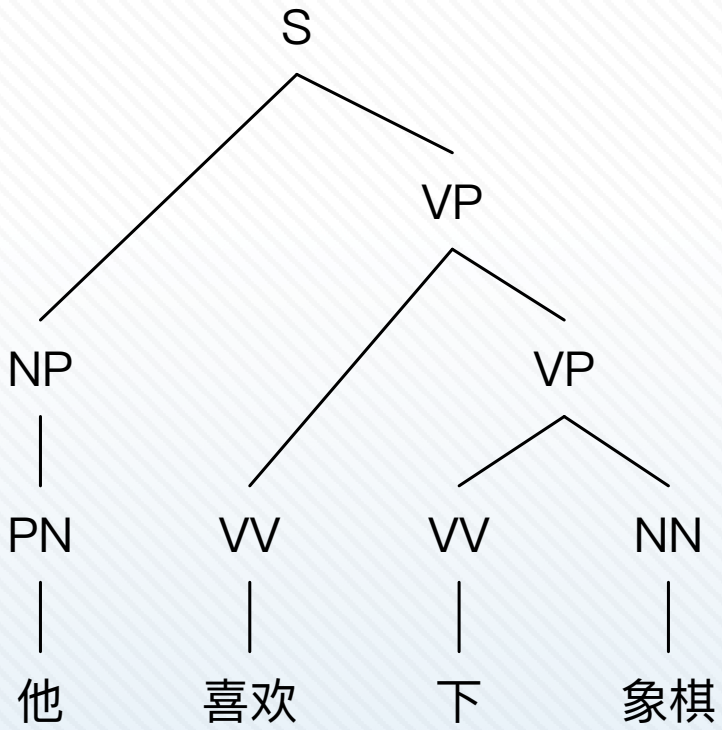




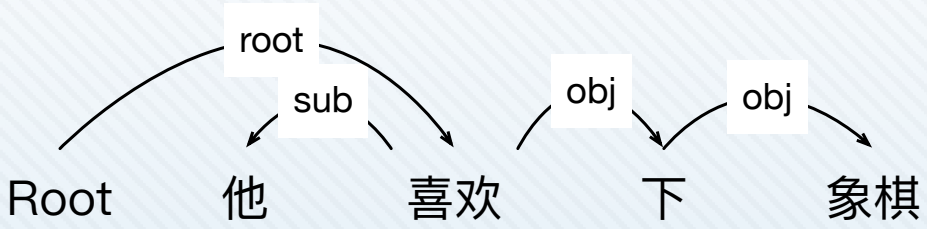


http://ltp.ai/demo.html

常用句法分析结构



(a) 短语结构句法树



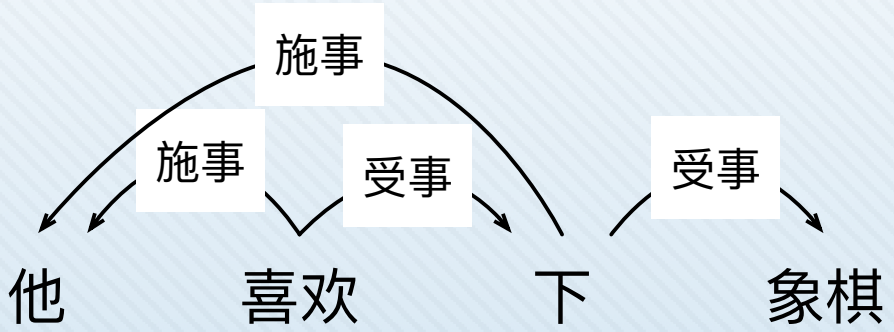
(b) 依存结构句法树

语义分析

- 词义消歧（Word Sense Disambiguation，WSD）
- 语义角色标注（Semantic Role Labeling，SRL）
  - 也称谓词论元结构（Predicate-Argument Structure）

输入	他	喜欢	下	象棋	。
输出 1	施事	谓词	受事		
输出 2	施事		谓词	受事	

- 语义依存图（Semantic Dependency Graph）



信息抽取

- 信息抽取（Information Extraction，IE）
  - 从非结构化的文本中自动提取结构化信息
- 输入
  - *10月28日，AMD宣布斥资350亿美元收购FPGA芯片巨头赛灵思。这两家传了多年绯闻的芯片公司终于走到了一起。*

信息抽取子任务	抽取结果
命名实体识别	公司名：AMD 公司名：赛灵思
关系抽取	赛灵思 $\xrightarrow{\text{从属}}$ AMD
时间表达式抽取	*10 月 28 日*
时间表达式归一化	*10 月 28 日* → *2020 年 10 月 28 日*
事件抽取	事件：收购 时间：*2020 年 10 月 28 日* 收购者：AMD 被收购者：赛灵思 收购金额：*350 亿美元*

情感分析

- 情感分析（Sentiment Analysis）
  - 个体对外界事物的态度、观点或倾向性，如正面、负面等
  - 人自身的情绪（Emotion），如喜怒哀惧等
- 输入
  - *这款手机的屏幕很不错，性能也还可以。*

情感分析子任务	分析结果
情感分类	褒义
情感信息抽取	评价词：不错；可以 评价对象：屏幕；性能 评价搭配：屏幕 ⇔ 不错；性能 ⇔ 可以

问答系统

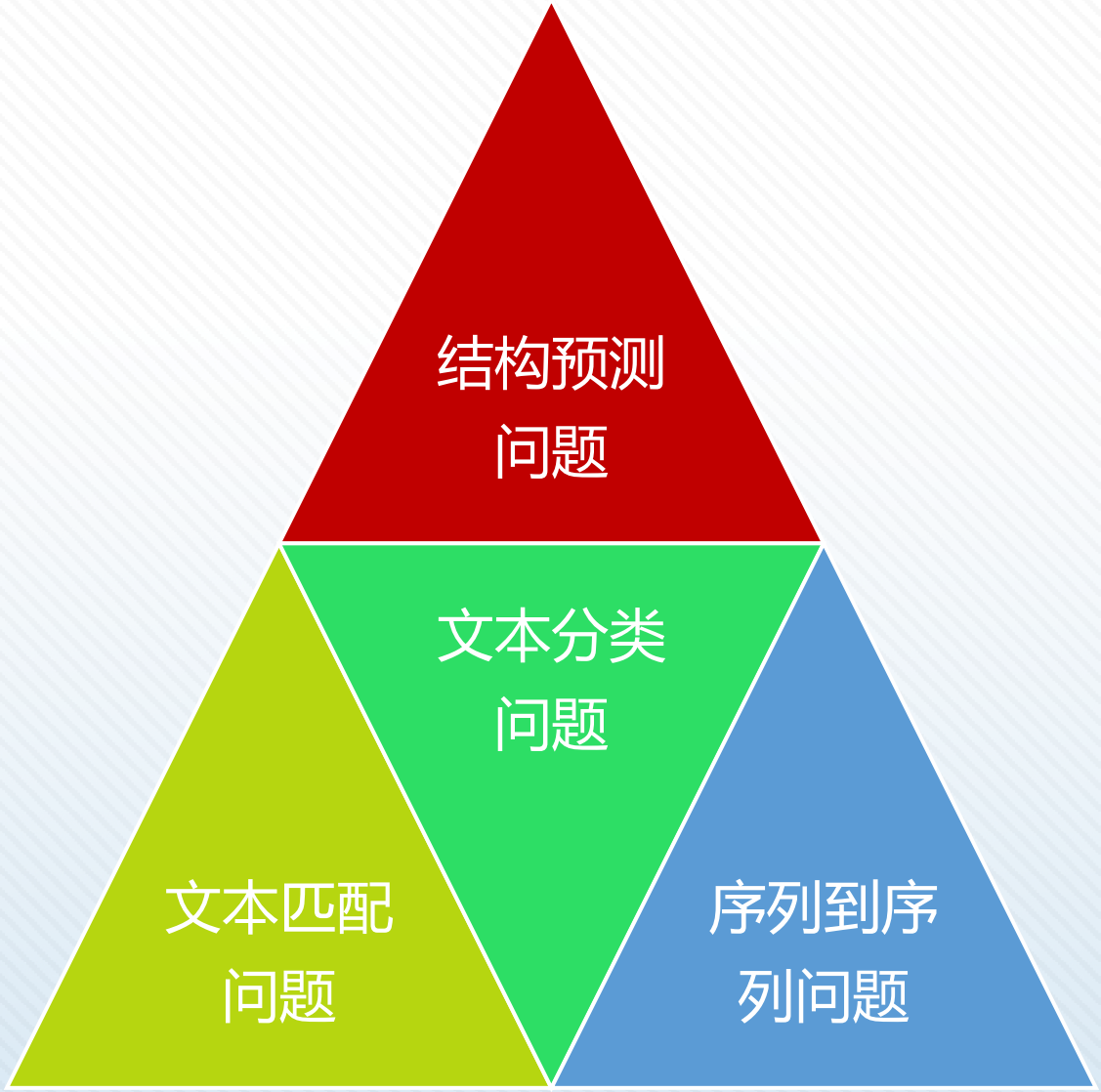
- 问答系统（Question Answering，QA）
  - 用户以自然语言形式描述问题，从异构数据中获得答案
- 根据数据来源的不同，问答系统可以分为4种主要的类型
  - 检索式问答系统
    - 答案来源于固定的文本语料库或互联网，系统通过查找相关文档并抽取答案完成问答
  - 知识库问答系统
    - 回答问题所需的知识以数据库等结构化形式存储，问答系统首先将问题解析为结构化的查询语句，通过查询相关知识点，并结合知识推理获取答案
  - 常问问题集问答系统
    - 通过对历史积累的常问问题集合进行检索，回答用户提出的类似问题
  - 阅读理解式问答系统
    - 通过抽取给定文档中的文本片段或生成一段答案来回答用户提出的问题

更多任务

- 机器翻译（Machine Translation，MT）
- 对话系统（Dialogue System）

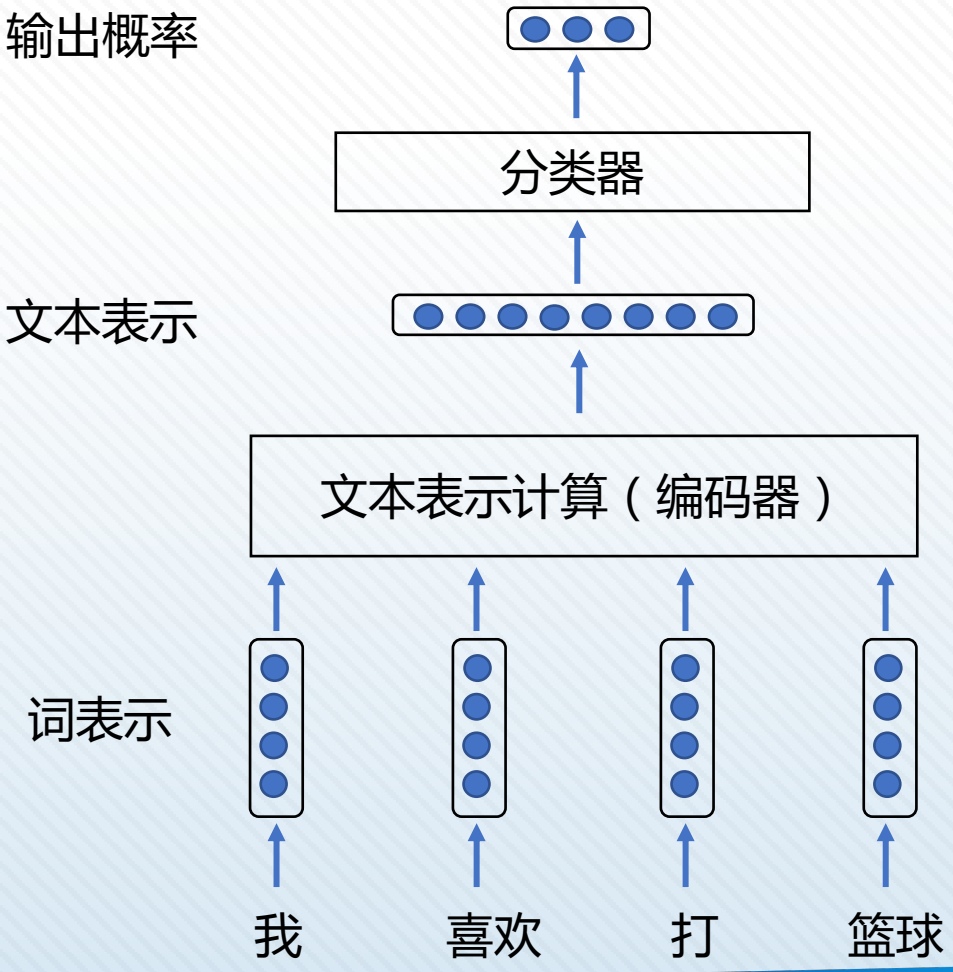
	任务型 Task	聊天 Chat	知识问答 Knowledge	推荐 Recommen-dation
目的	完成任务或动作	闲聊	知识获取	信息推荐
领域	特定域（垂类）	开放域	开放域	特定域
以话轮数评价	越少越好	话轮越多越好	越少越好	越少越好
应用	虚拟个人助理	娱乐、情感陪护	客服、教育	个性化推荐
典型系统	Siri、Cortana、Google Assistant、度秘	小冰、笨笨	Watson、Wolfram Alpha	阿里小蜜

归结为四个基本问题



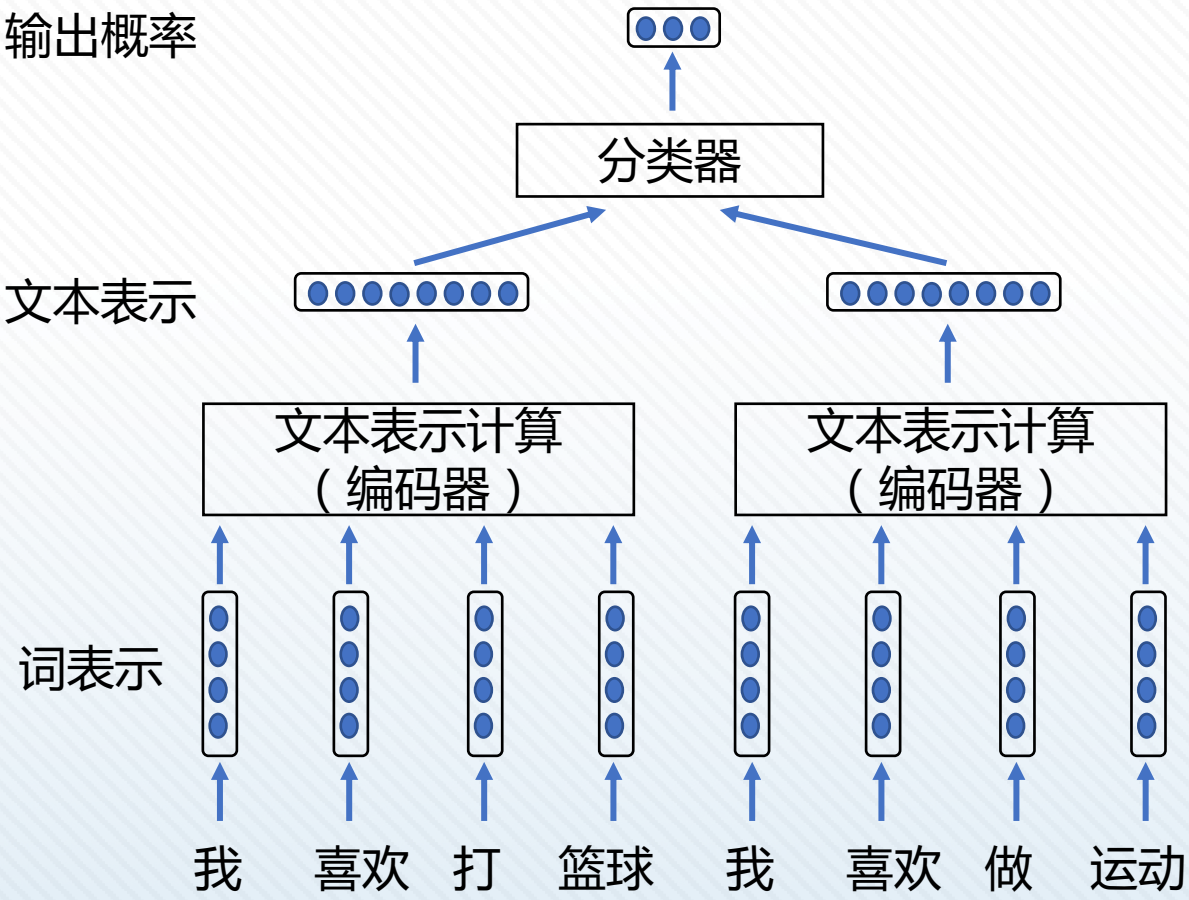
文本分类问题

- 将输入文本映射为所属类别（预定义的封闭集合）
- 最简单最基本的自然语言处理问题
- 应用场景
  - 垃圾邮件过滤、情感分类等
- 很多问题可以转化为文本分类问题



文本匹配问题

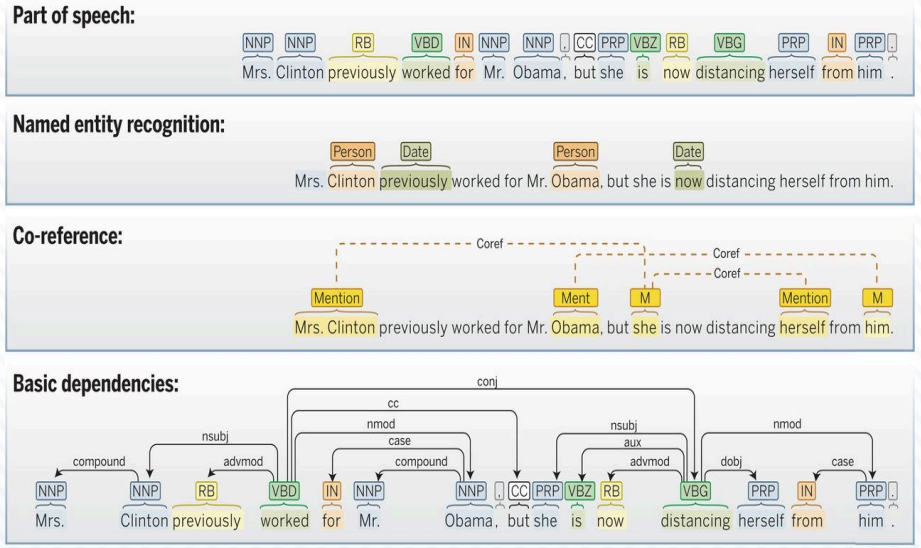
- 判断两段文本之间的匹配关系
  - 如复述关系、蕴含关系等
- 解决方案
  - 双塔结构
    - 两段文本分别通过两个模型映射为向量，然后判断两个向量之间的匹配关系
  - 单塔结构
    - 将两段文本直接拼接，然后进行匹配关系分类



双塔结构示意图

结构预测问题

- 输出类别之间具有较强的相互关联性
- 自然语言处理的本质问题
- 三种典型的结构预测问题
  - 序列标注
    - 为输入文本序列中的每个词标注相应的标签
    - 如词性标注：他/PN 喜欢/VV 下/VV 象棋/NN
  - 序列分割
    - 在文本序列中切分出子序列
    - 可以转化为序列标注问题
  - 图结构生成
    - 输入自然语言，输出以图表示的结构



Julia Hirschberg and Christopher D. Manning. Advances in Natural Language Processing. **Science** 2015

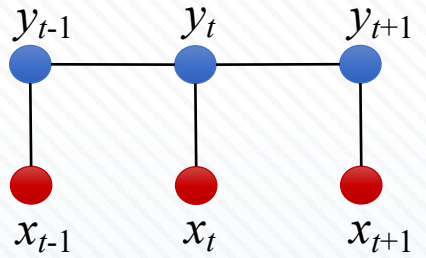
Science AAAS

输入	我	爱	北	京	天	安	门	。
分词输出	B	B	B	I	B	I	I	B
命名实体输出	O	O	B-LOC	I-LOC	I-LOC	I-LOC	I-LOC	O

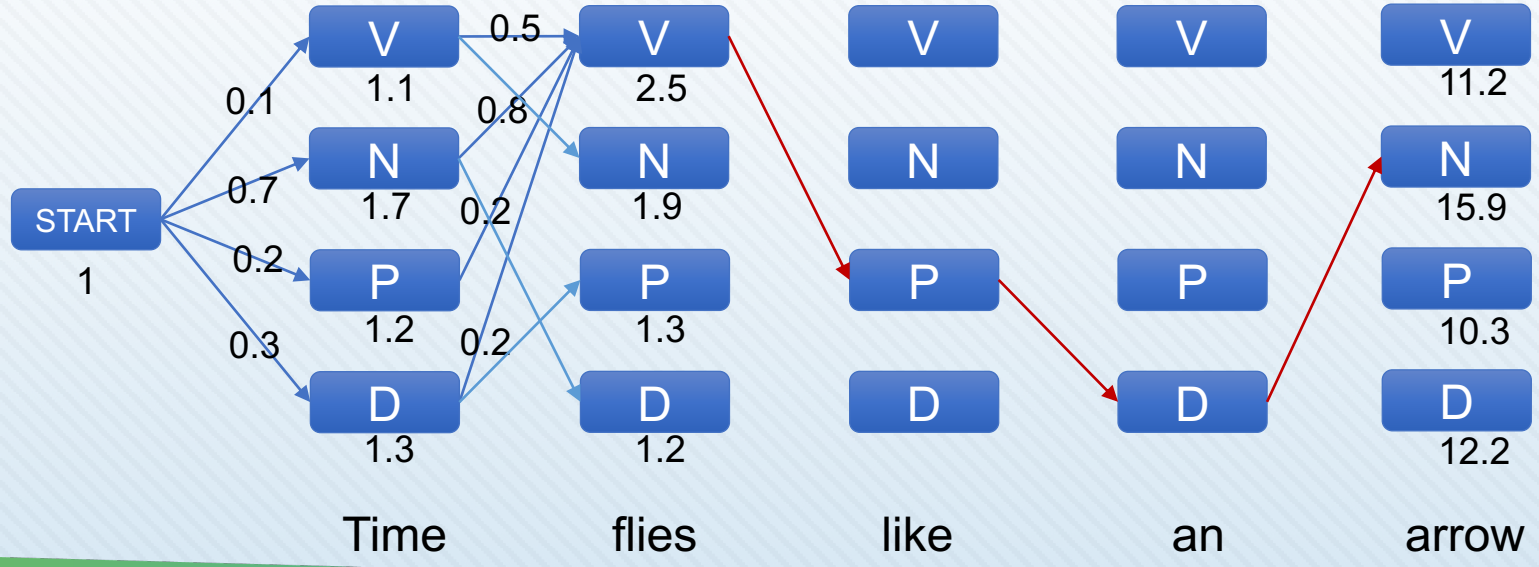
结构预测问题：序列标注

- 经典的序列标注模型
  - 条件随机场（Conditional Random Field，CRF）

CRF $$P(y_{[1:n]}|x_{[1:n]}) \propto \frac{1}{Z_{y_{[1:n]}}} \prod_{t=1}^{n} \exp\left(\begin{matrix} \sum_j \lambda_j f_j(y_t, y_{t-1}) \\ + \sum_k \mu_k g_k(y_t, x_t) \end{matrix} \right)$$

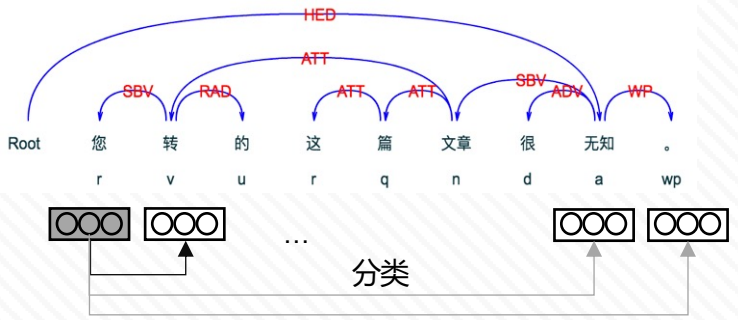


- 维特比（Viterbi）解码算法



结构预测问题：图结构生成

- 基于图的算法
  - 计算图中任意两个节点之间连边的分数（类别）
  - 根据解码算法生成图（树）结构



- 基于转移的算法
  - 将图结构的构建过程转化为状态转移序列
    - 如标准弧转移算法
    - 转移状态由一个栈和一个队列构成
    - 三种转移动作
      - 移进（SH）、左弧（RL）和右弧（RR）
  - 对旧状态执行一个移动作转换为新状态
  - 转移动作的选择本质上也是分类问题

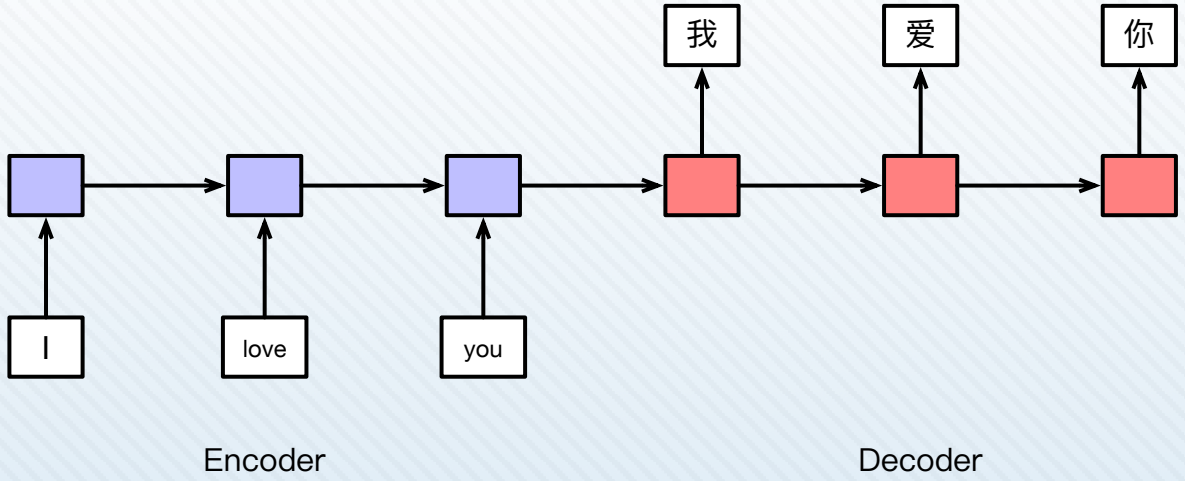
步骤	栈	队列	下一步动作
0		他 喜欢 下 象棋	SH
1	他	喜欢 下 象棋	SH
2	他 喜欢	下 象棋	RL
3	喜欢	下 象棋	SH
4	喜欢 下	象棋	SH
5	喜欢 下 象棋		RR
6	喜欢 下		RR
7	喜欢		FIN

面向依存句法分析的标准弧转移算法示例

序列到序列问题

- 将输入序列转换为输出序列
  - 输入和输出的序列不要求等长，也不要求词表一致
  - 泛化为“编码器—解码器”模型（Encoder-Decoder）
    - 本质上也是分类问题
- 典型任务

任务	输入	输出
机器翻译	源语言	目标语言
文本摘要	原文	摘要
回复生成	用户语句	机器回复
图片描述生成	图片	文本描述
语音识别	语音	转写文本



“编码器—解码器”模型示例

评价指标

- 准确率（Accuracy）
  - $\text{Acc} = \dfrac{\text{正确分类的样本数}}{\text{测试样本总数}}$
  - 不适用于针对某一类别的评价

输入	张	三	是	哈	尔	滨	人	。
正确标注序列	B-PER	I-PER	O	B-LOC	I-LOC	I-LOC	O	O
预测标注序列	B-PER	O	O	B-LOC	I-LOC	I-LOC	O	O

- F值（针对某一类别）
  - 精确率（Precision）和召回率（Recall）加权调和平均
  - $\text{F}_1 = \dfrac{2PR}{P+R}$，$P = \dfrac{\text{正确识别的某类样本数目}}{\text{识别出的该类样本总数}}$，$R = \dfrac{\text{正确识别的某类样本数目}}{\text{测试数据中该类样本的总数}}$

其它评价指标

- 依存分析
  - 分析结果是树结构
    - 以词为单位，父节点识别的准确率
    - UAS（Unlabeled Attachment Score）词的父节点被正确识别的准确率
    - LAS（Labeled Attachment Score）同时考虑词与父节点的关系是否正确
  - 分析结果是图结构
    - F值评价
- 指代消解等聚类任务
  - 多种复杂的评价指标
- 针对机器翻译等文本生成任务
  - BLEU
- 对话系统的评价仍然是一个开放问题

目 录

CONTENTS

基础工具集与常用数据

基础工具集

- NLTK
- LTP
- PyTorch

常用数据集

- Wikipedia
- Common Crawl

NLTK

- Natural Language Toolkit
  - https://www.nltk.org/
- 多种语料库和词典资源
  - 生文本、PennTreebank样例
  - WordNet
- 基本的自然语言处理工具集
  - 分句
  - 标记解析
  - 词性标注
  - 句法分析
- 更多英文自然语言处理工具集
  - CoreNLP、spaCy等

https://data-flair.training/blogs/nltk-python-tutorial/

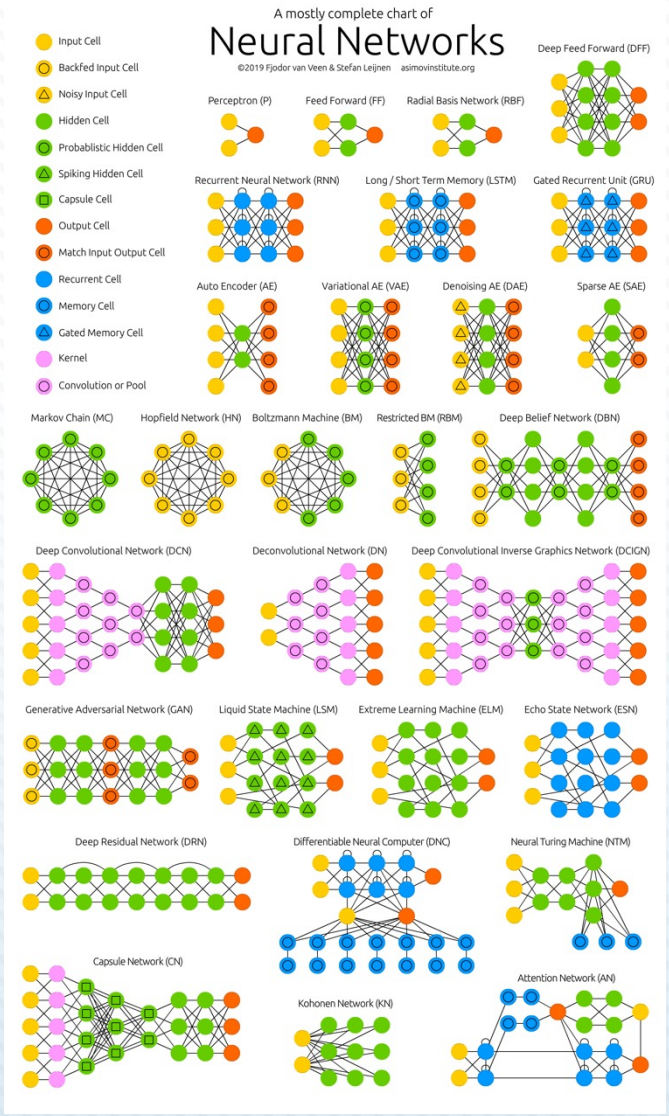
- 语言技术平台（Language Technology Platform，LTP）
  - http://ltp.ai
  - 高效、高精度中文自然语言处理基础平台
  - 中文词法、句法、语义分析等6项自然语言处理核心技术
- 2020年7月发布4.0版本
  - 基于预训练模型
  - 多任务学习机制
- 安装
  - $ pip install ltp

LTP调用示例

```
>>> sentences = ltp.sent_split(["南京市长江大桥。", "汤姆生病了。他去了医院。"
    ]) # 分句
>>> print(sentences)
['南京市长江大桥。', '汤姆生病了。', '他去了医院。']
>>> segment, hidden = ltp.seg(sentences)
>>> print(segment)
[['南京市', '长江', '大桥', '。'], ['汤姆', '生病', '了', '。'], ['他', '去', '
    了', '医院', '。']]
>>> pos_tags = ltp.pos(hidden) # 词性标注
>>> print(pos_tags) # 词性标注的结果为每个词所对应的词性，LTP使用的词性标记集与
    # NLTK不尽相同，但基本大同小异
[['ns', 'ns', 'n', 'wp'], ['nh', 'v', 'u', 'wp'], ['r', 'v', 'u', 'n', 'wp']]
```

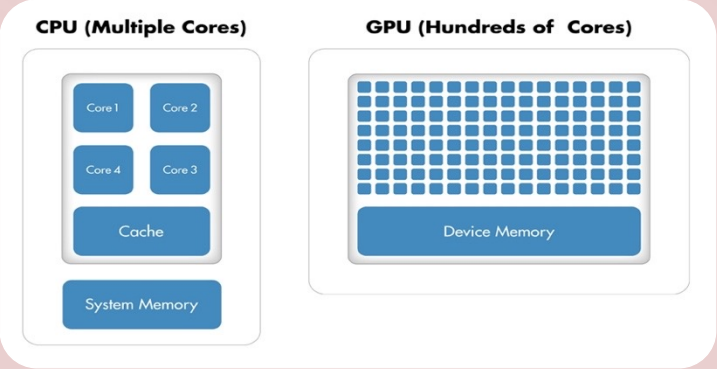
PyTorch

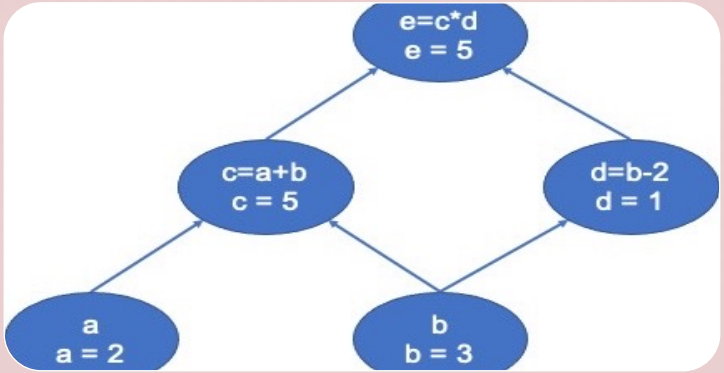
- Facebook开发的开源深度学习库
  - https://pytorch.org/
- 基于张量（Tensor）的数学运算工具包
  - GPU加速的张量计算
  - 自动进行微分计算
- PyTorch的优势
  - 框架简洁
  - 入门简单，容易上手
  - 支持动态神经网络构建
  - 与Python语言无缝结合
  - Debug方便

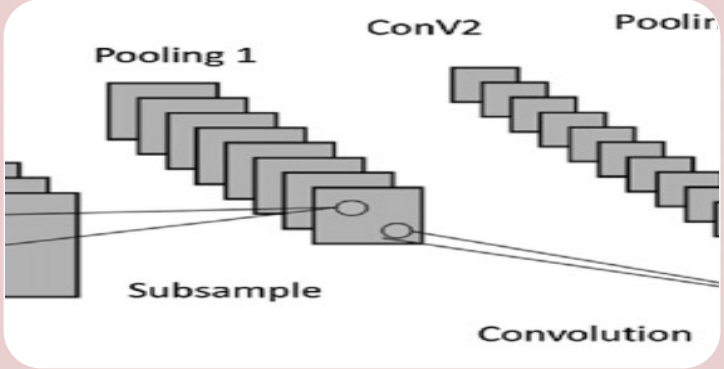


https://www.asimovinstitute.org/neural-network-zoo/

PyTorch的三要素







张量（Tensor）

- 数据存储
- 支持GPU

表达式（Expression）

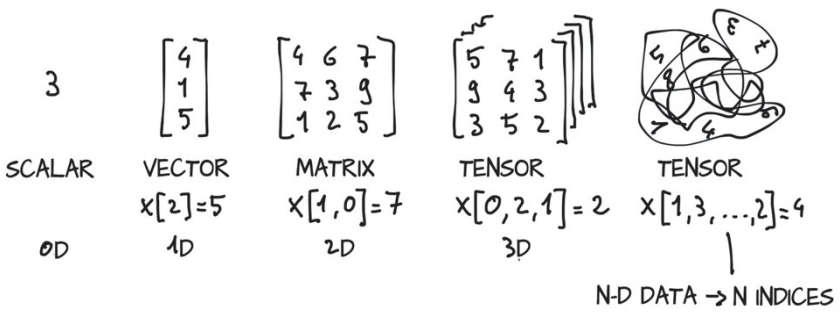
- 数据运算
- 自动微分计算

模块（Module）

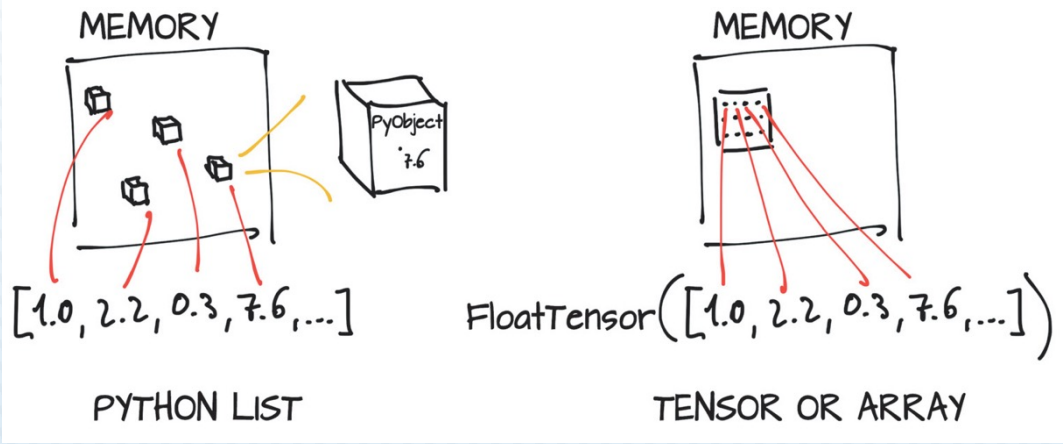
- 神经网络层
- 支持自定义

张量（Tensor）

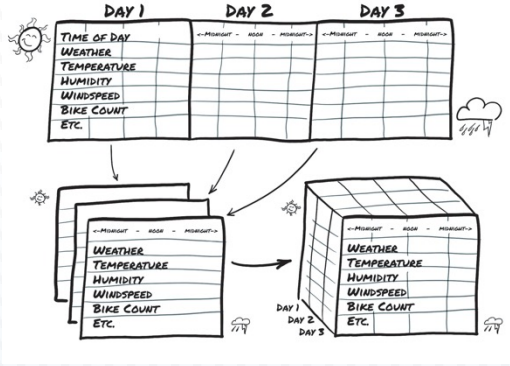
- 又称为多维数组



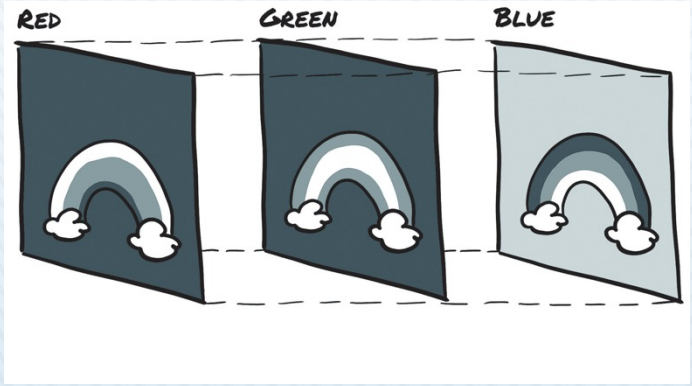
- 与Python列表的区别



- 可表示现实世界中的各种数据
  - 表格、时间序列



  - （彩色）图像



batch = torch.zeros(100, 3, 256, 256, dtype=torch.uint8)

PyTorch作为张量库

- 支持各种张量操作（类似NumPy）
  - 创建张量
  - 索引、切片
- 支持GPU
  - 快！

$M * M * M$ $M \in \mathbb{R}^{1000\times1000}$

Numpy

```
In [2]: M = numpy.random.randn(1000,1000)

In [3]: timeit -n 500 M.dot(M).dot(M)
500 loops, best of 3: 30.7 ms per loop
```

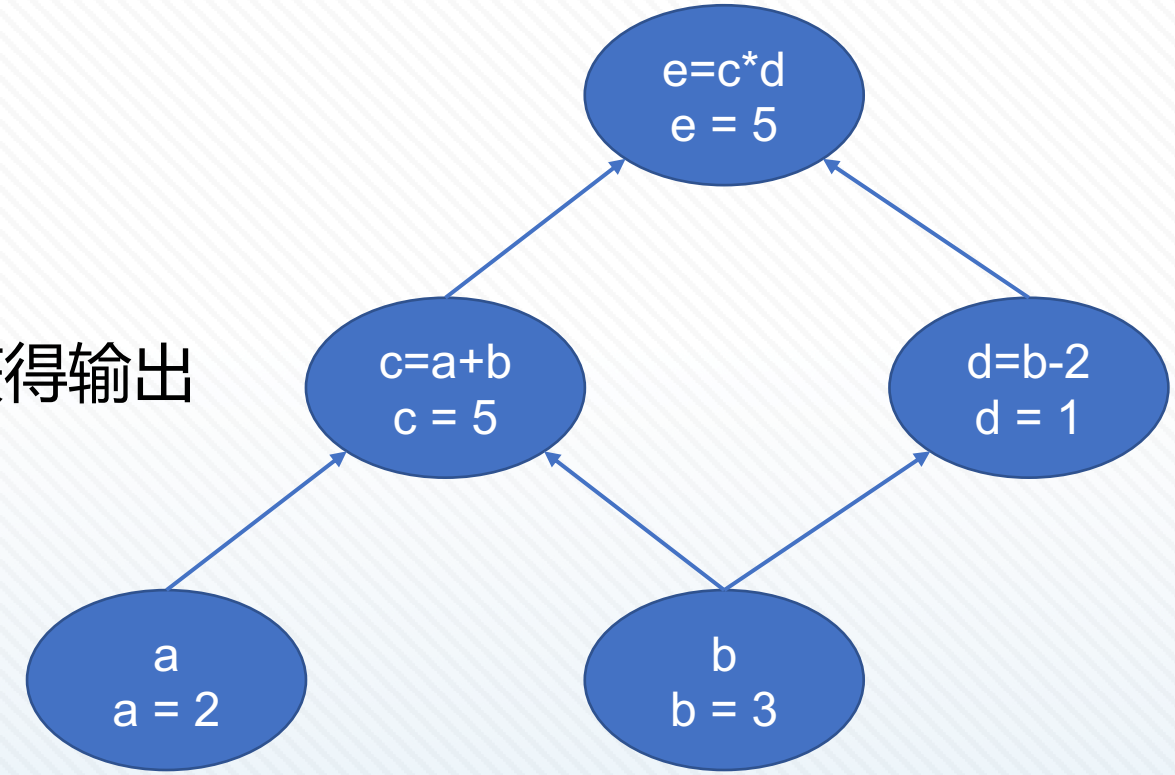
PyTorch

```
In [4]: N = torch.randn(1000,1000).cuda()

In [5]: timeit -n 500 N.mm(N).mm(N)
500 loops, best of 3: 474 µs per loop
```

表达式（Expession）

- 通过计算图描述表达式
  - 如：e = (a + b) * (b - 2)
- 计算图中每个节点执行一个运算
  - 前向运算（Forward）：根据输入获得输出

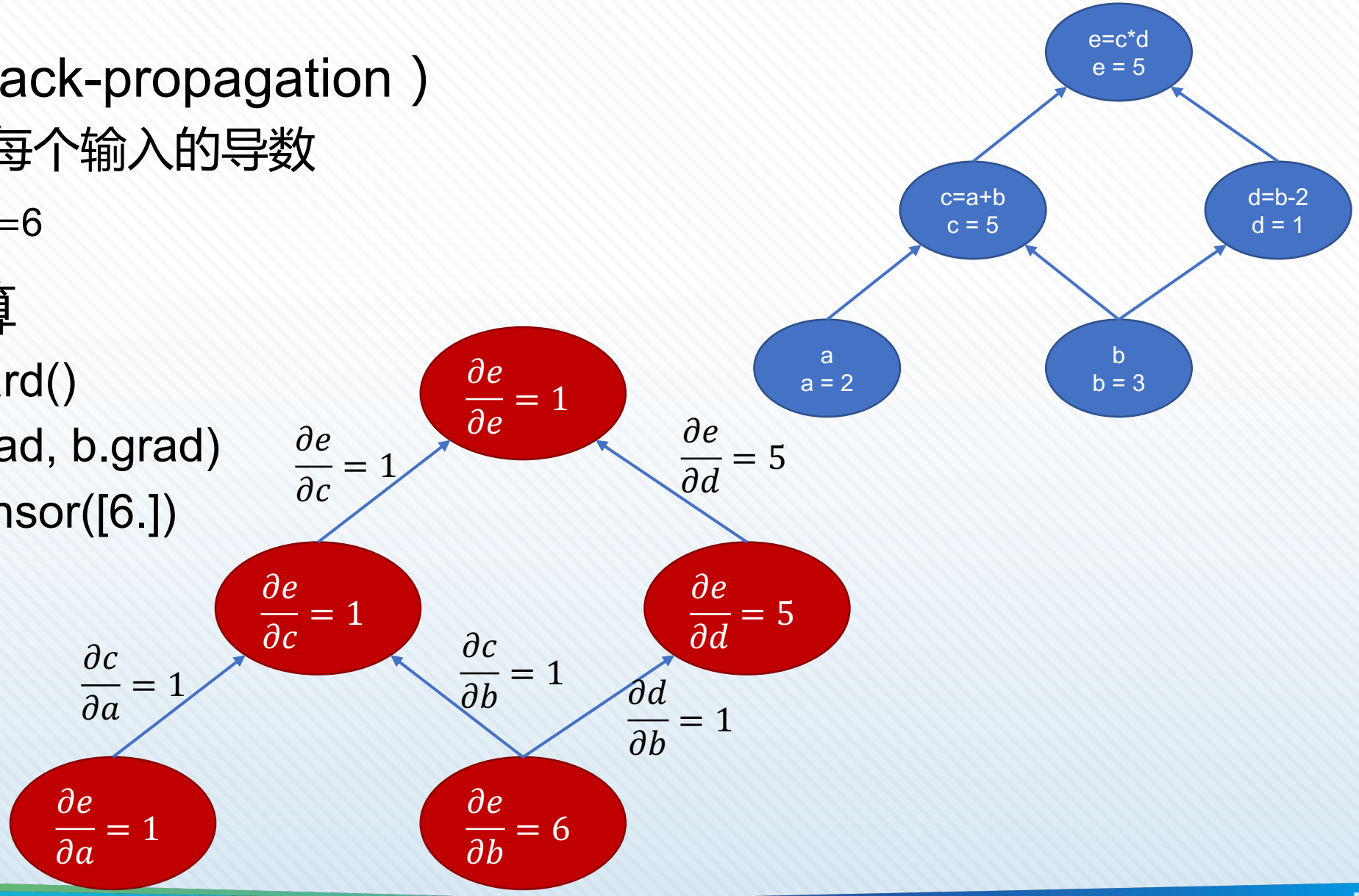
```
>>> a = torch.tensor([2.])
>>> b = torch.tensor([3.])
>>> c = a + b
>>> d = b - 2
>>> e = c * d
>>> print(c, d, e)
tensor([5.], grad_fn=<AddBackward0>)
tensor([1.], grad_fn=<SubBackward0>)
tensor([5.], grad_fn=<MulBackward0>)
```



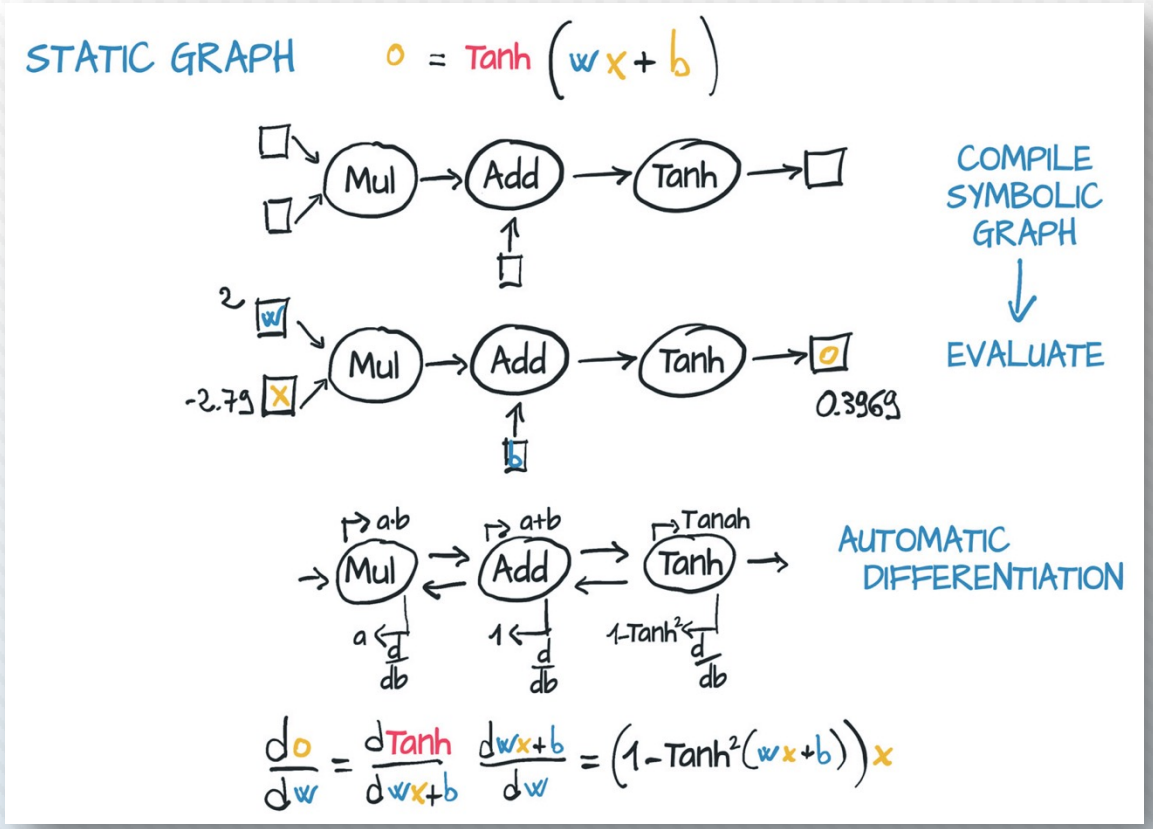
表达式（Expession）

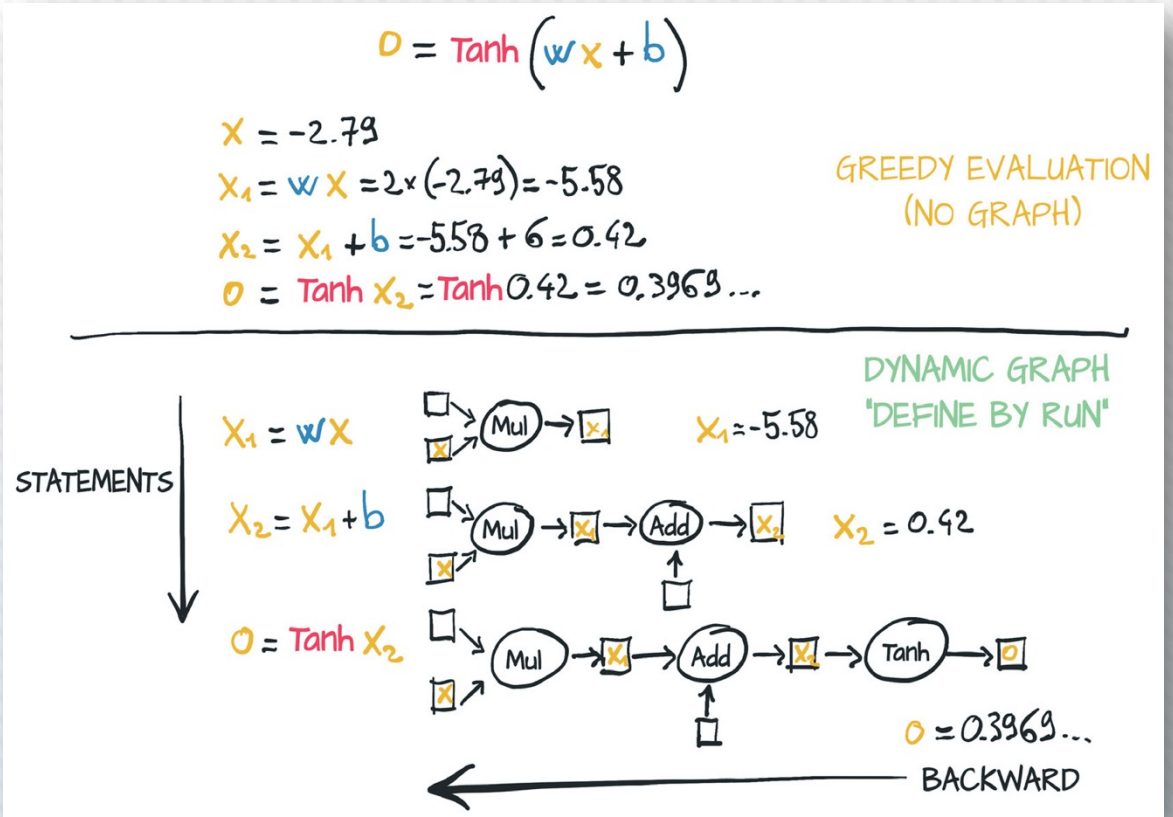
- 反向传播（Back-propagation）
  - 计算输出对每个输入的导数
    - $\frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial b} = 6$
- 自动微分计算

```
>>> e.backward()
>>> print(a.grad, b.grad)
tensor([1.]) tensor([6.])
```



静态图与动态图


STATIC GRAPH
o = Tanh (wx + b)
Mul
Add
Tanh
2
w
-2.79
x
b
0.3969
COMPILE SYMBOLIC GRAPH
EVALUATE
a·b
a+b
Tanah
1-Tanh²
AUTOMATIC DIFFERENTIATION
do/dw = dTanh/dwx+b · dwx+b/dw = (1 - Tanh²(wx+b)) x


o = Tanh (wx + b)
x = -2.79
x₁ = w x = 2 x (-2.79) = -5.58
x₂ = x₁ + b = -5.58 + 6 = 0.42
o = Tanh x₂ = Tanh 0.42 = 0.3969...
GREEDY EVALUATION (NO GRAPH)
DYNAMIC GRAPH "DEFINE BY RUN"
STATEMENTS
x₁ = wx
x₂ = x₁ + b
o = Tanh x₂
x₁ = -5.58
x₂ = 0.42
o = 0.3969...
BACKWARD

模块（Module）

包名	功能	描述
torch	Tesnor / Expression	类似NumPy的张量库，支持GPU以及自动微分
torch.nn	Module	灵活的神经网络库，提供多种神经网络层
torch.optim	Module	多种优化算法
torch.utils	Module	数据集、数据加载等辅助工具

大规模预训练数据集

- 维基百科（Wikipedia）
  - 快照（2020年10月23日）https://dumps.wikimedia.org/

文件名	内容	大小/MB
zhwiki-latest-abstract.xml.gz	所有词条摘要	≈ 147
zhwiki-latest-all-titles.gz	所有词条标题	≈ 33
zhwiki-latest-page.sql.gz	所有词条标题及摘要	≈ 204
zhwiki-latest-pagelinks.sql.gz	所有词条外链	≈ 890
zhwiki-latest-pages-articles.xml.bz2	所有词条正文	≈ 1,952

  - 处理流程
    - 纯文本抽取 → 中文繁简转换 → 数据清洗
- Common Crawl数据（https://commoncrawl.org/）
  - 大规模网络爬虫数据集
  - 使用Facebook的CC-Net工具下载和处理
    - https://github.com/facebookresearch/cc_net

目 录

CONTENTS

感知器模型

- 感知器（Perceptron）
  - 最简单也是最早出现的机器学习模型
  - 灵感直接来源于生产生活的实践
- 面试评分
  - $s = x_1 + x_2 + x_3 + x_4$
    - 如果$s \geq t$则录用，否则不录用
  - 考虑面试官的经验（权重）
    - $s = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$
- 感知器模型

$$y = \begin{cases} 1 \text{ if } \sum_i w_i x_i + b \geq 0 \\ 0 \text{ else} \end{cases}$$，其中 $b = -t$

Logistic回归模型

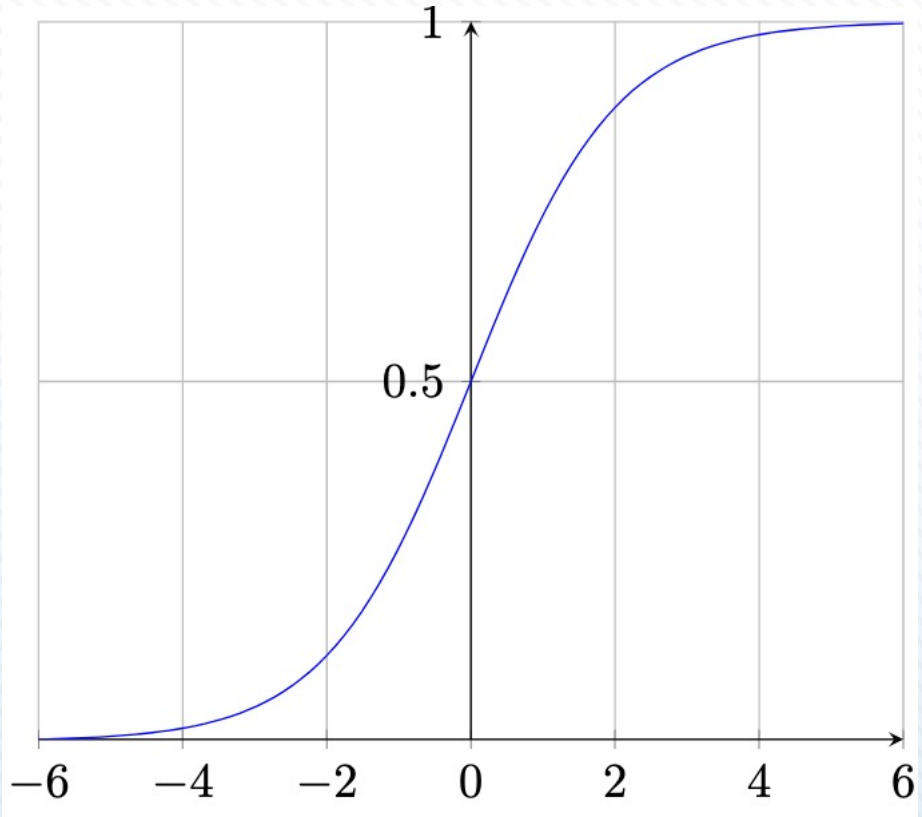
- 线性回归（Linear Regression）
  - $y = \sum_i w_i x_i + b$
- Logistic回归（Logistic Regression）
  - Logistic函数

$$y = \frac{L}{1 + \mathrm{e}^{-k(z - z_0)}}$$

  - 设$z = \sum_i w_i x_i + b$
  - Sigmoid函数（Logistic函数的特例）

$$y = \frac{1}{1 + \mathrm{e}^{-z}}$$

  - 处理二元分类问题，y为输出的概率
    - 垃圾邮件过滤、褒贬识别



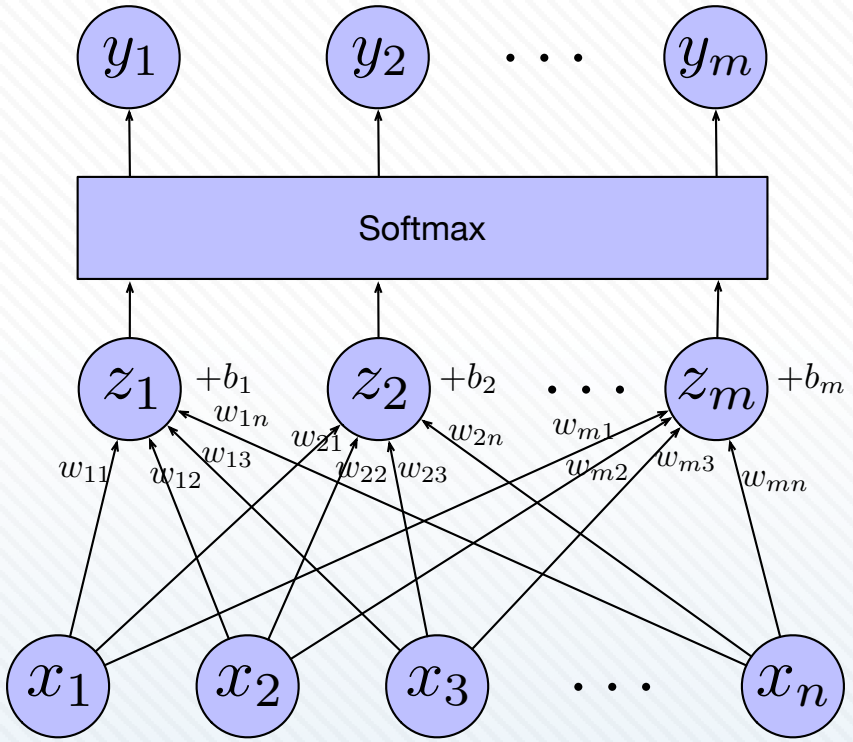
Sigmoid函数示意图

Softmax回归

- 使用Softmax回归处理多元分类
- Softmax函数

$$y_i = \text{Softmax}(\boldsymbol{z})_i = \frac{\mathrm{e}^{z_i}}{\mathrm{e}^{z_1} + \mathrm{e}^{z_2} + \cdots + \mathrm{e}^{z_m}}$$

  - 其中y_i为第i个类别的概率，$z_i = \sum_j w_{ij} x_k + b$，$w_{ij}$为第$i$个类别所对应的第$j$个输入的权重

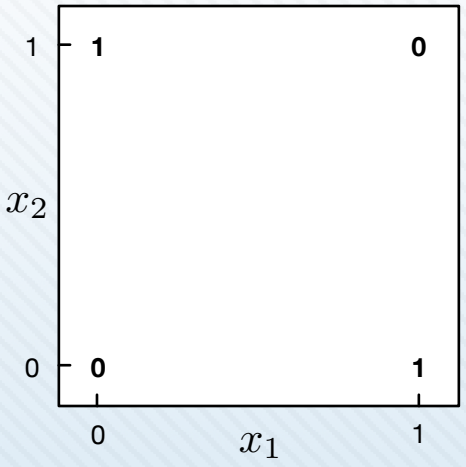


$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \text{Softmax} \begin{pmatrix} w_{11}x_1 + w_{12}x_2 + \cdots + w_{1n}x_n + b_1 \\ w_{21}x_1 + w_{22}x_2 + \cdots + w_{2n}x_n + b_2 \\ \vdots \\ w_{m1}x_1 + w_{m2}x_2 + \cdots + w_{mn}x_n + b_m \end{pmatrix}$$

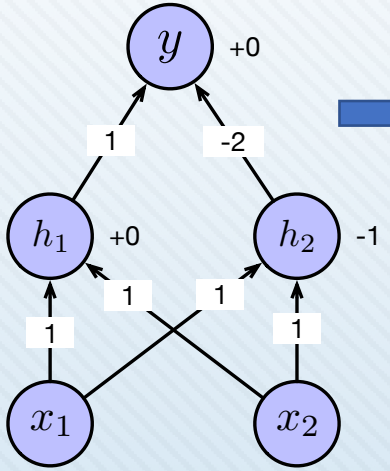
$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \text{Softmax} \left(\begin{bmatrix} w_{11}, w_{12}, \cdots, w_{1n} \\ w_{21}, w_{22}, \cdots, w_{2n} \\ \vdots \\ w_{m1}, w_{m2}, \cdots, w_{mn} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \right)$$

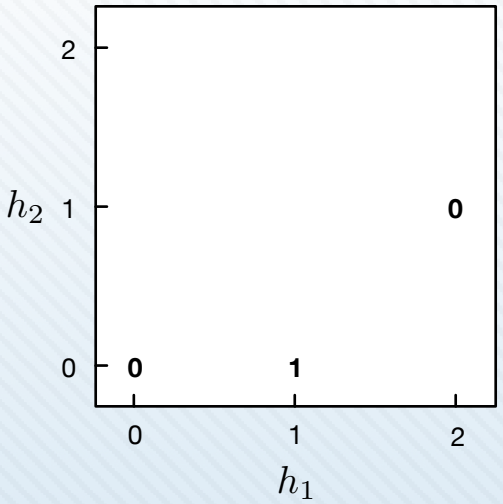
$$\boldsymbol{y} = \text{Softmax}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

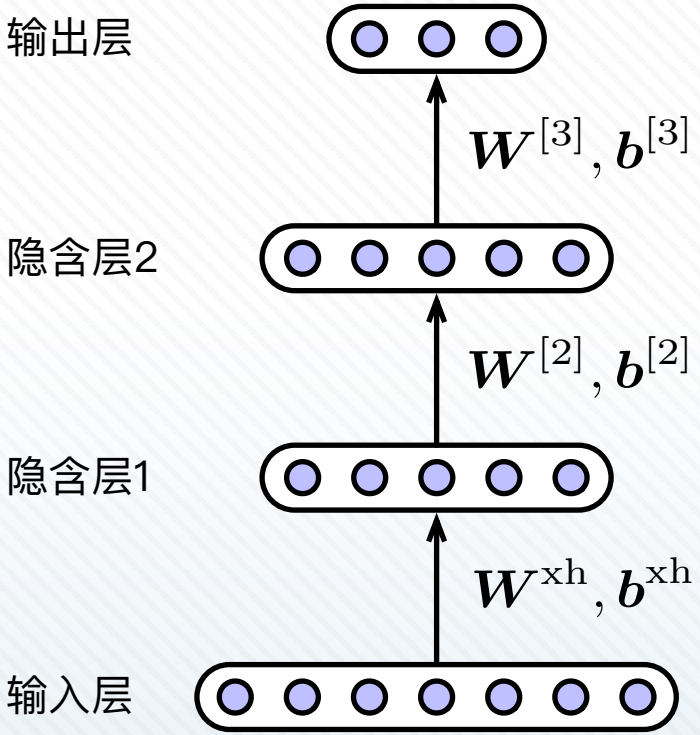
- 多层感知器（Multi-layer Perceptron，MLP）
  - 堆叠多层感知器（线性回归+非线性激活函数）
  - 可解决线性不可分问题
- 使用MLP解决异或（XOR）问题



$$z = W^{[1]}x + b^{[1]}$$
$$h = \mathrm{ReLU}(z)$$
$$y = W^{[2]}h + b^{[2]}$$







多层感知器PyTorch实现

引入PyTorch模块

- >>> from torch import nn

线性层

```
>>> linear = nn.Linear(32, 2) # 输入32维，输出2维
>>> inputs = torch.rand(3, 32) # 创建一个形状为(3, 32)的随机张量，3为批次大小
>>> outputs = linear(inputs) # 输出张量形状为(3, 2)
>>> print(outputs)
tensor([[ 0.5387, -0.4537],
        [ 0.2181, -0.3745],
        [ 0.3704, -0.8121]], grad_fn=<AddmmBackward>)
```

激活函数

- >>> from torch.nn import functional as F

```
>>> activation = F.sigmoid(outputs)
>>> print(activation)
tensor([[0.6315, 0.3885],
        [0.5543, 0.4075],
        [0.5916, 0.3074]], grad_fn=<SigmoidBackward>)
>>> activation = F.softmax(outputs, dim=1)
    # 沿着第2维进行Softmax运算，即对每批次中的各样例分别进行Softmax运算
>>> print(activation)
tensor([[0.7296, 0.2704],
        [0.6440, 0.3560],
        [0.7654, 0.2346]], grad_fn=<SoftmaxBackward>)
>>> activation = F.relu(outputs)
>>> print(activation)
tensor([[0.5387, 0.0000],
        [0.2181, 0.0000],
        [0.3704, 0.0000]], grad_fn=<ReluBackward0>)
```

多层感知器PyTorch实现

自定义神经网络实现

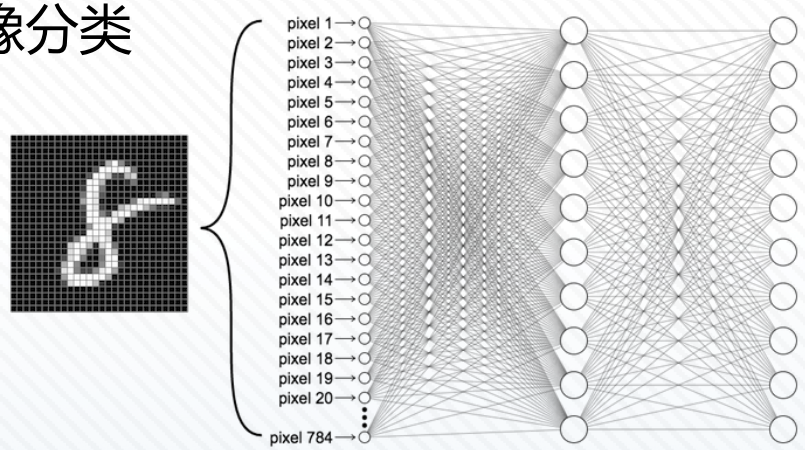
```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, num_class):
        super(MLP, self).__init__()
        # 线性变换：输入层->隐含层
        self.linear1 = nn.Linear(input_dim, hidden_dim)
        # 使用ReLU激活函数
        self.activate = F.relu
        # 线性变换：隐含层->输出层
        self.linear2 = nn.Linear(hidden_dim, num_class)

    def forward(self, inputs):
        hidden = self.linear1(inputs)
        activation = self.activate(hidden)
        outputs = self.linear2(activation)
        probs = F.softmax(outputs, dim=1) # 获得每个输入属于某一类别的概率
        return probs
```

```
mlp = MLP(input_dim=4, hidden_dim=5, num_class=2)
inputs = torch.rand(3, 4) # 输入形状为(3, 4)的张量，其中3表示有3个输入，4表示每个
    输入的维度
probs = mlp(inputs) # 自动调用forward函数
print(probs) # 输出3个输入对应输出的概率
```

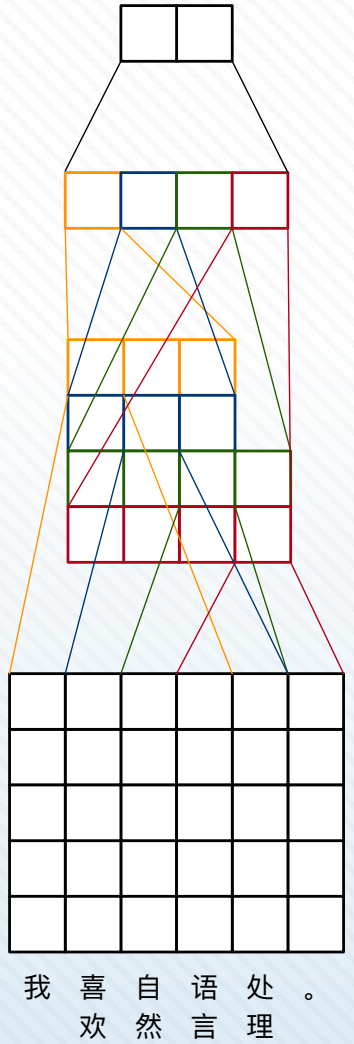
https://colab.research.google.com/drive/1EMJJffsmlSlqO3KwuG5WlP-lxyNVXTrK?usp=sharing

卷积神经网络

- 全连接的多层感知器无法处理输入的偏移情况
  - 图像分类



  - 情感分类
    - *我喜欢自然语言处理。* vs. *我非常喜欢自然语言处理。*
- 解决方案
  - 使用小的全连接层抽取局部特征（又称卷积核或滤波器）
    - 如遍历文本中的N-gram等
  - 使用多个卷积核提取不同种类的特征
  - 使用池化层将特征进行聚合（最大、平均、求和等）

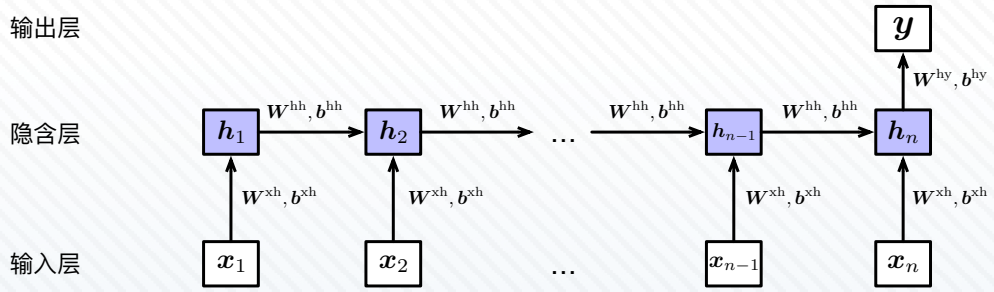


colab

https://colab.research.google.com/drive/1HaPOfEj8wrLkqeUDuirtS-cNKGxeRdY5?usp=sharing

循环神经网络

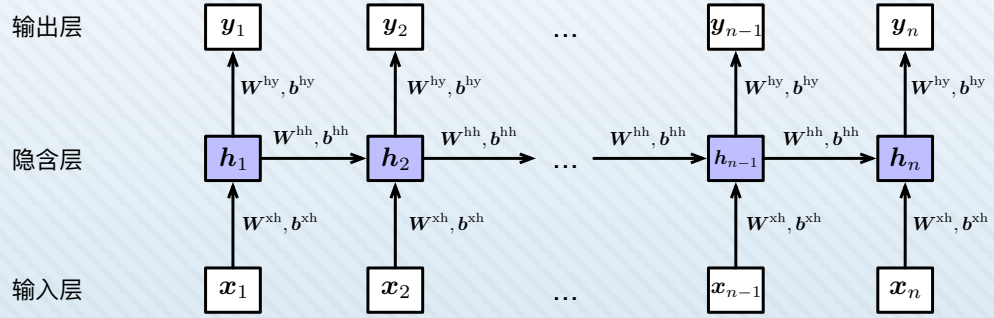
- 卷积神经网络无法处理任意长度的依赖
- 循环神经网络的解决方案
  - 每个时刻的隐状态依赖于当前时刻的输入以及上一时刻的隐状态

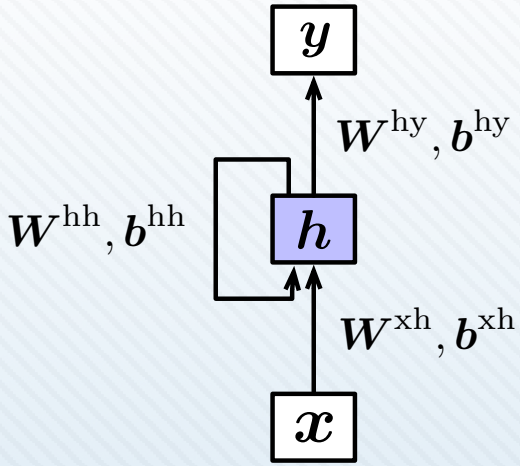


$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}^{\mathrm{xh}}\boldsymbol{x}_t + \boldsymbol{b}^{\mathrm{xh}} + \boldsymbol{W}^{\mathrm{hh}}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{\mathrm{hh}})$$

$$\boldsymbol{y} = \mathrm{Softmax}(\boldsymbol{W}^{\mathrm{hy}}\boldsymbol{h}_n + \boldsymbol{b}^{\mathrm{hy}})$$

  - 每个时刻的参数共享（“循环”的由来）
  - 每个时刻也可以有相应的输出（序列标注）





长短时记忆网络（LSTM）

- 序列过长时，原始的循环神经网络容易导致信息损失
  - 梯度爆炸或梯度消散
- 新的加性隐状态更新方式

$$\boldsymbol{u}_t = \tanh(\boldsymbol{W}^{\mathrm{xh}}\boldsymbol{x}_t + \boldsymbol{b}^{\mathrm{xh}} + \boldsymbol{W}^{\mathrm{hh}}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{\mathrm{hh}})$$

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \boldsymbol{u}_t$$

long term　　short term

  - 相当于直接将 $\boldsymbol{h}_k$ 与 $\boldsymbol{h}_t$（$k < t$）进行了跨层连接
    - $\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \boldsymbol{u}_t = \boldsymbol{h}_{t-2} + \boldsymbol{u}_{t-1} + \boldsymbol{u}_t = \boldsymbol{h}_k + \boldsymbol{u}_{k+1} + \boldsymbol{u}_{k+2} + \cdots + \boldsymbol{u}_t$
- 进一步改进
  - 遗忘门：考虑旧状态 $\boldsymbol{h}_{t-1}$ 和新状态 $\boldsymbol{u}_t$ 的贡献

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

$$\boldsymbol{h}_t = \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{f}_t) \odot \boldsymbol{u}_t$$

  - 输入门：独立控制 $\boldsymbol{h}_{t-1}$ 和 $\boldsymbol{u}_t$ 的贡献

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

$$\boldsymbol{h}_t = \boldsymbol{f}_t \odot \boldsymbol{h}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{u}_t$$

  - 输出门：对输出进行控制

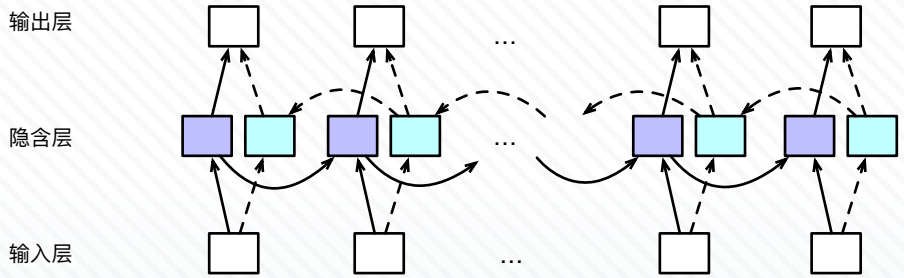
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}^{\mathrm{o,xh}}\boldsymbol{x}_t + \boldsymbol{b}^{\mathrm{o,xh}} + \boldsymbol{W}^{\mathrm{o,hh}}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{\mathrm{o,hh}})$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{u}_t$$

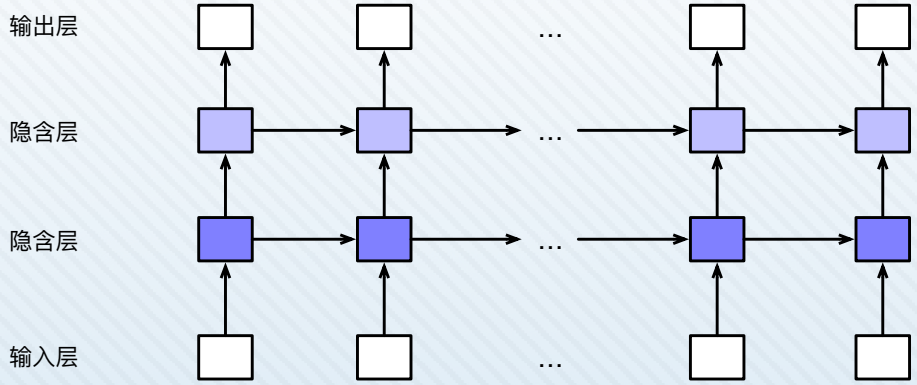
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)$$

循环神经网络的应用

双向循环神经网络

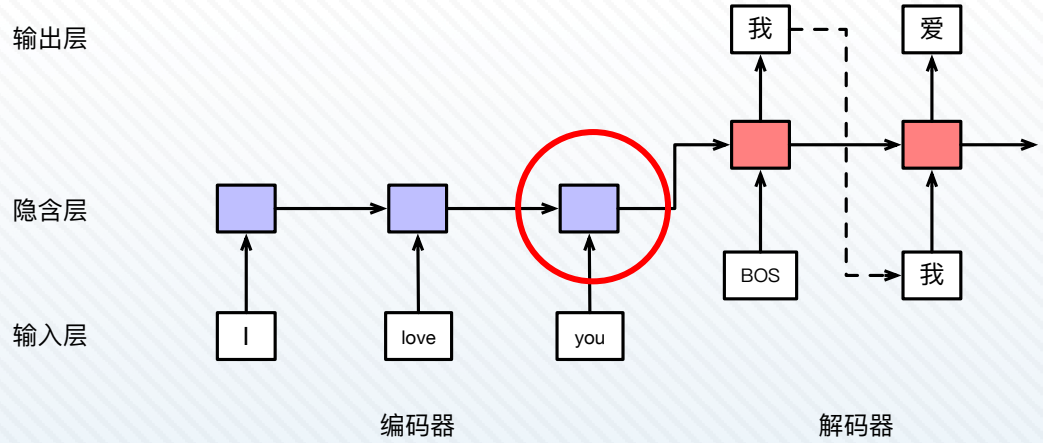


堆叠循环神经网络



序列到序列模型

- 也称“编码器—解码器”模型
- 机器翻译等多种应用

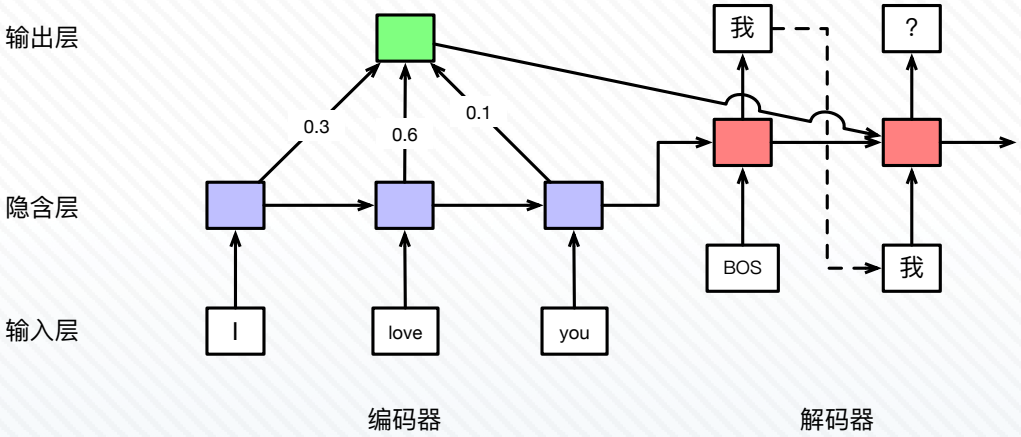


You can’t cram the meaning of a whole %&!$ing sentence into a single $&!*ing vector!

Ray Mooney

注意力机制（Attention Mechanism）

- 当前状态除和前一个状态及输入相关外，还应关注原序列的状态

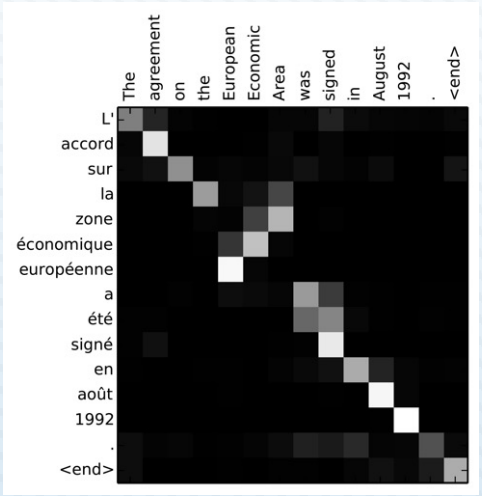


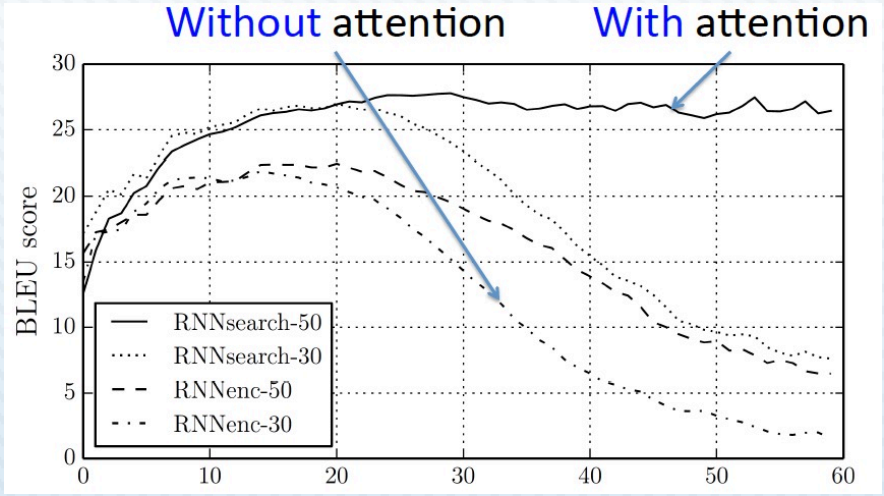
- 打分公式

$$\hat{\alpha}_s = \text{attn}(\boldsymbol{h}_s, \boldsymbol{h}_{t-1})$$

$$\alpha_s = \text{Softmax}(\hat{\boldsymbol{\alpha}})_s$$

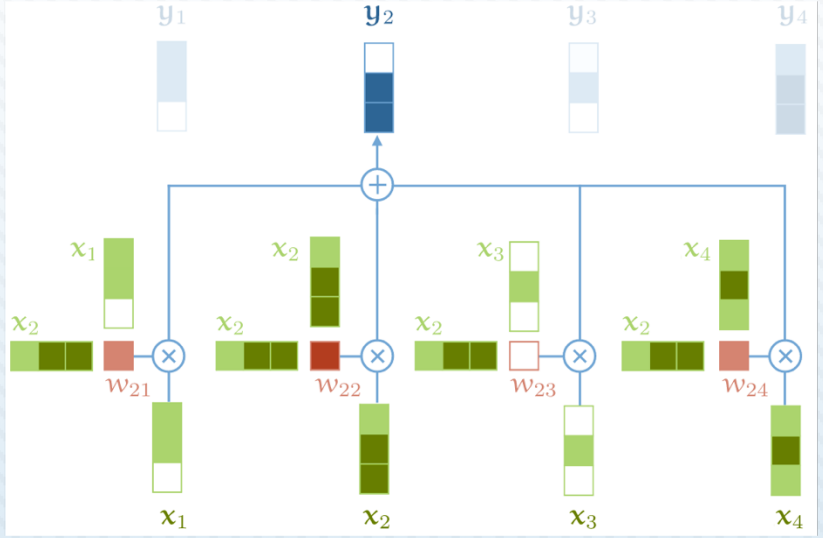
$$\text{attn}(\boldsymbol{q}, \boldsymbol{k}) = \begin{cases} \boldsymbol{w}^\top \tanh(\boldsymbol{W}[\boldsymbol{q}; \boldsymbol{k}]) \\ \boldsymbol{q}^\top \boldsymbol{W} \boldsymbol{k} \\ \boldsymbol{q}^\top \boldsymbol{k} \\ \frac{\boldsymbol{q}^\top \boldsymbol{k}}{\sqrt{d}} \end{cases}$$





自注意力模型

- 通过某一输入与周围输入的相关性（注意力）来更新该输入
  - “观其伴、知其义”
- 输入：n个向量构成的序列$\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots \boldsymbol{x}_n$
- 输出：每个向量对应的新向量$\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots \boldsymbol{y}_n$
- $\boldsymbol{y}_i = \sum_{j=1}^{n} \alpha_{ij} \boldsymbol{x}_j$ ， α_{ij}为 $\boldsymbol{x}_i$ 与 $\boldsymbol{x}_j$之间的注意力值




```python
import torch
import torch.nn.functional as F

x = torch.rand(3, 4, 5)

# torch.bmm is a batched matrix multiplication. It
# applies matrix multiplication over batches of
# matrices.

raw_weights = torch.bmm(x, x.transpose(1, 2))

weights = F.softmax(raw_weights, dim=2)

y = torch.bmm(weights, x)

print(y)
```

Transformer

- Vaswani et al., **Attention** Is All You Need, NIPS 2017
- Transformer的翻译
  - 变压器？
  - 变形金刚？
- 自注意力模型还需要解决的几个问题
  - 没有考虑输入的位置信息
  - 输入向量 x_i 同时承担三种角色，不易学习
    - 计算注意力权重时的两个向量以及被加权的向量
  - 只考虑了两个输入向量之间的关系，无法建模多个向量之间的更复杂关系
  - 自注意力计算结果互斥，无法同时关注多个输入

融入位置信息

- 位置嵌入（Position Embeddings）
  - 类似词嵌入，每个绝对位置赋予一个连续、低维、稠密的向量表示
  - 向量参数参与模型学习
- 位置编码（Position Encodings）
  - 直接将一个整数（位置索引值）映射为一个向量

$$\mathrm{PosEnc}(p,i) = \begin{cases} \sin\left(\frac{p}{10000^{\frac{i}{d}}}\right) & \text{if } i \text{ is even} \\ \cos\left(\frac{p}{10000^{\frac{i-1}{d}}}\right) & \text{else} \end{cases}$$

- 输入向量 = 词向量 + 位置嵌入/编码

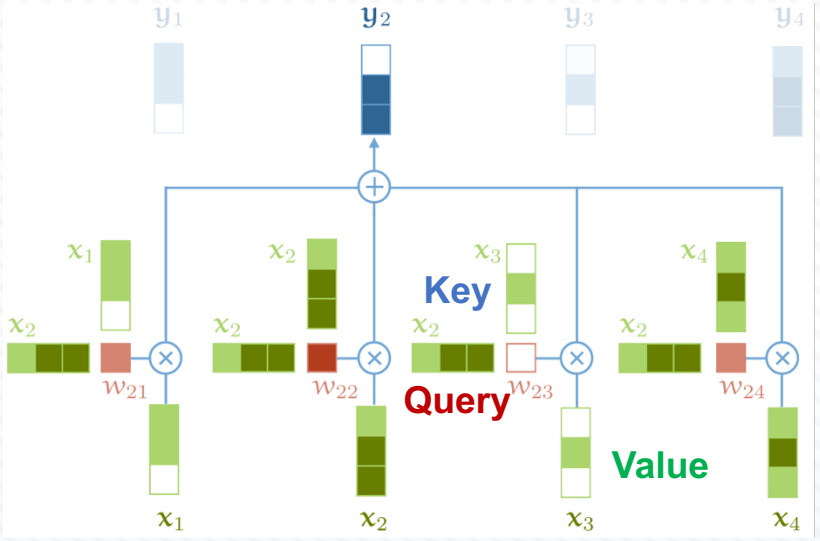
赋予输入向量角色信息

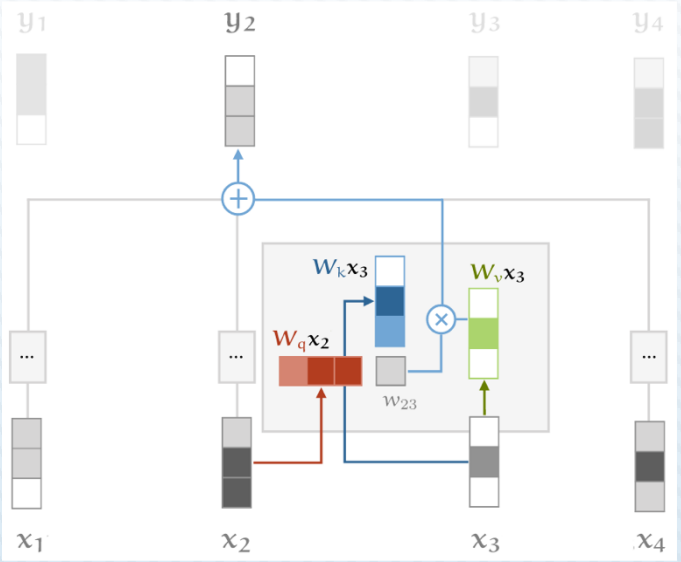
- 输入向量的三种角色
  - 查询（Query）
  - 键（Key）
  - 值（Value）
- 分别对输入向量进行线性映射
  - $\boldsymbol{q}_i = \boldsymbol{W}^{\mathrm{q}}\boldsymbol{x}_i$
  - $\boldsymbol{k}_i = \boldsymbol{W}^{\mathrm{k}}\boldsymbol{x}_i$
  - $\boldsymbol{v}_i = \boldsymbol{W}^{\mathrm{v}}\boldsymbol{x}_i$
- 新的自注意力模型

$$\boldsymbol{y}_i = \sum_{j=1}^{n} \alpha_{ij}\boldsymbol{v}_j$$

$$\alpha_{ij} = \mathrm{Softmax}(\hat{\boldsymbol{\alpha}}_i)_j$$

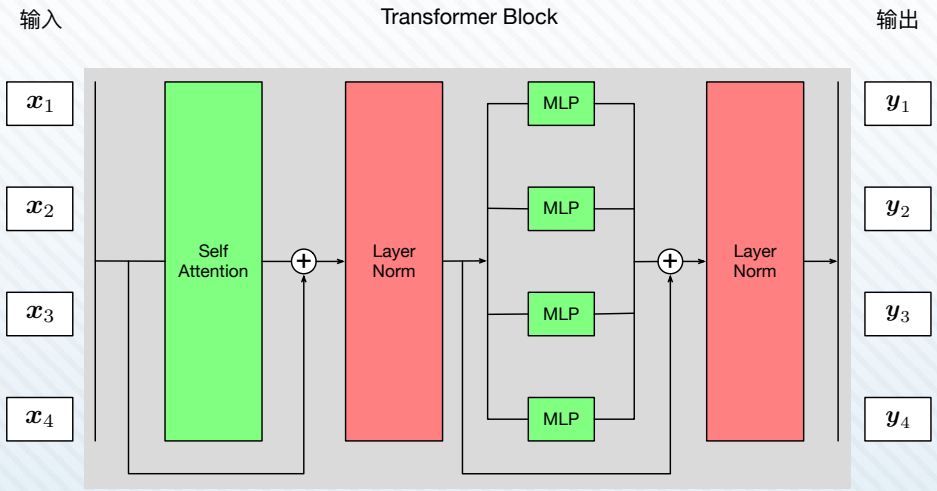
$$\hat{\alpha}_{ij} = \mathrm{attn}(\boldsymbol{q}_i, \boldsymbol{k}_j)$$





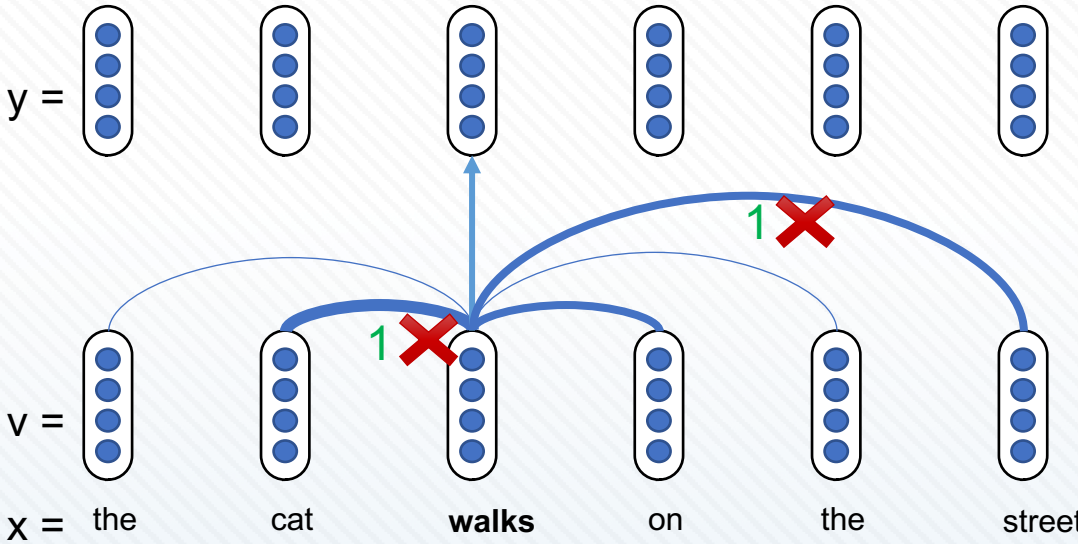
多层自注意力

- 原始自注意力模型仅考虑了任意两个向量之间的关系
- 如何建模高阶关系？
  - 直接建模高阶关系导致模型复杂度过高
  - 堆叠多层自注意力模型（消息传播机制）
- 增强模型的表示能力——增加非线性
  - 增加非线性的多层感知器模型（MLP）
- 使模型更容易学习
  - 层归一化（Layer Normalization）
  - 残差连接（Residual Connections）

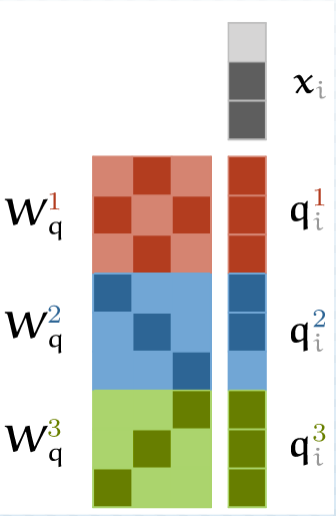


多头自注意力

- 由于Softmax函数的性质，无法使得多个自注意力分数趋近于1



- 使用多组自注意力模型产生多组不同的注意力结果
  - 设置多组输入映射矩阵
  - 类似使用多个卷积核提取不同的特征



Transformer的优缺点

优点

- 直接建模更长距离的依赖关系
- 更快的训练速度（与RNN相比）

缺点

- 参数量过大导致模型不容易训练
- 需要基于大规模数据预训练

神经网络模型的训练

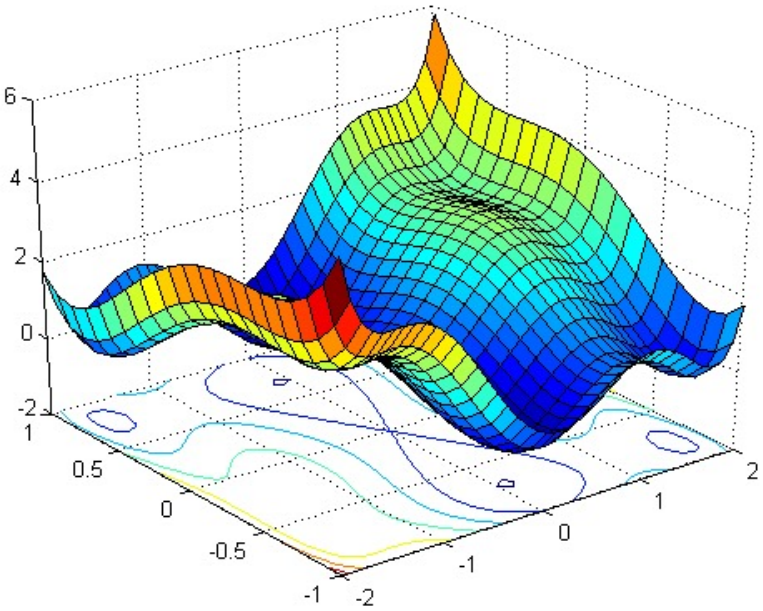
- 寻找一组优化的模型参数
  - 又叫做模型训练或学习
- 损失函数（Loss Function）
  - 评估参数好坏的准则
  - 为什么不直接使用准确率等指标进行评估？
- 两种常用的损失函数
  - 均方误差（Mean Squared Error，MSE）

$$\mathrm{MSE} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

  - 交叉熵（Cross-Entropy，CE）

$$\mathrm{CE} = -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{c} y_j^{(i)} \log \hat{y}_j^{(i)} \quad \Rightarrow \quad \mathrm{CE} = -\frac{1}{m}\sum_{i=1}^{m} \log \hat{y}_t^{(i)}$$

负对数似然损失
Negative Log Likelihood，NLL



梯度下降

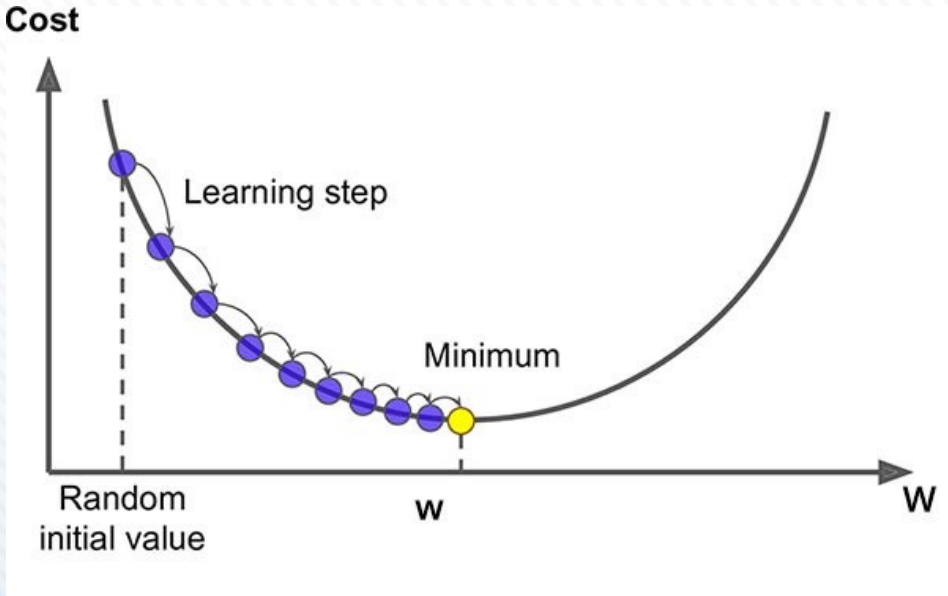
- 梯度（Gradient）
  - 以向量的形式写出的对多元函数各个参数求得的偏导数
  - 是函数值增加最快的方向
  - 沿着梯度相反的方向，更加容易找到函数的极小值
- 梯度下降算法（Gradient Descent，GD）

算法 4.1 梯度下降算法

Input: 学习率 α；含有 m 个样本的训练数据

Output: 优化参数 $\boldsymbol{\theta}$

1. 设置损失函数为 $L(f(\boldsymbol{x};\boldsymbol{\theta}),y)$;
2. 初始化参数 $\boldsymbol{\theta}$。
3. **while** 未达到终止条件 **do**
4. 　计算梯度 $\boldsymbol{g}=\frac{1}{m}\nabla_{\boldsymbol{\theta}}\sum_i^m L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),y^{(i)})$;
5. 　$\boldsymbol{\theta}=\boldsymbol{\theta}-\alpha\boldsymbol{g}$。
6. **end**



- 小批次梯度下降法（Mini-batch Gradient Descent）
  - 每次随机采样小规模的训练数据来估计梯度
  - 提高算法的运行速度

PyTorch代码示例

```
# 创建多层感知器模型，输入层大小为2，隐含层大小为5，输出层大小为2（即有两个类别）
model = MLP(input_dim=2, hidden_dim=5, num_class=2)

criterion = nn.NLLLoss() # 当使用log_softmax输出时，需要调用负对数似然损失（Negative Log Likelihood，NLL）
optimizer = optim.SGD(model.parameters(), lr=0.05) # 使用梯度下降参数优化方法，学习率设置为0.05

for epoch in range(500):
    y_pred = model(x_train) # 调用模型，预测输出结果
    loss = criterion(y_pred, y_train) # 通过对比预测结果与正确的结果，计算损失
    optimizer.zero_grad() # 在调用反向传播算法之前，将优化器的梯度值置为零，否则每次循环的梯度将进行累加
    loss.backward() # 通过反向传播计算参数的梯度
    optimizer.step() # 在优化器中更新参数，不同优化器更新的方法不同，但是调用方式相同
```

colab https://colab.research.google.com/drive/1v-4G7WgiEhV_wOuOC_2IoSIVNzRHQlnP?usp=sharing

目 录

CONTENTS

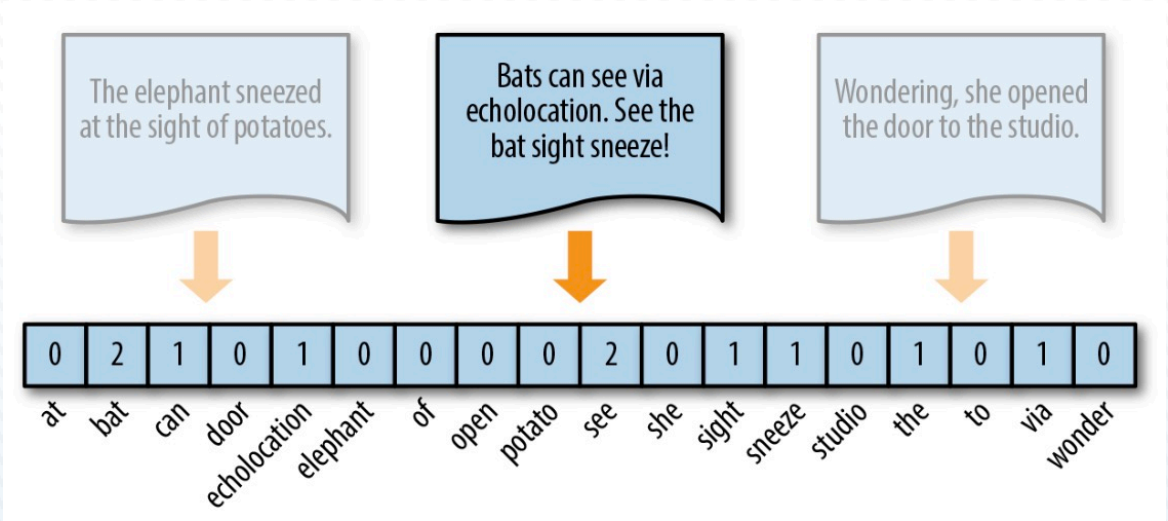
什么是词向量？

- 词的一种机内表示形式，便于计算
- 传统使用one-hot词向量表示词
  - 高维、稀疏、离散
  - 导致严重的数据稀疏问题
  - 所有向量都是正交的

土豆 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...]

马铃薯 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...]

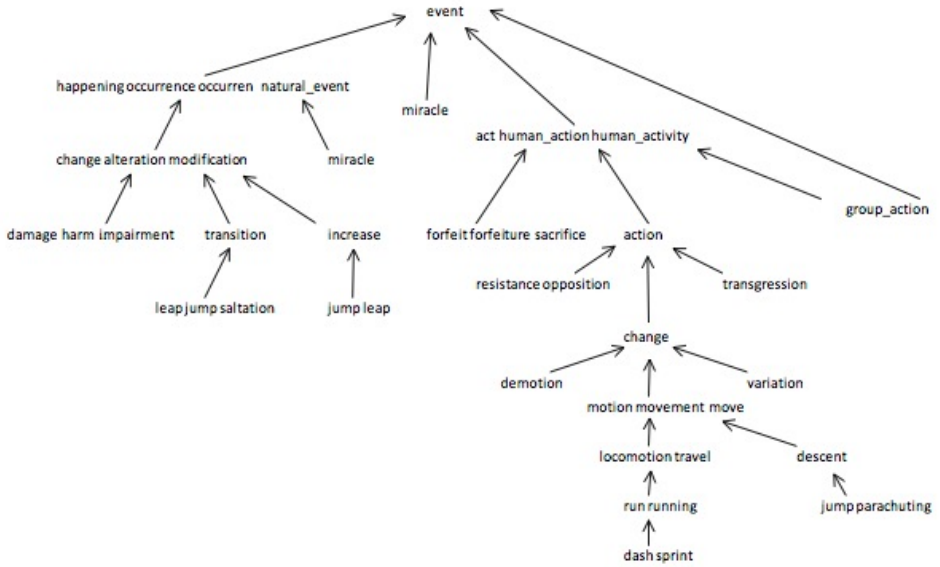
sim(土豆, 马铃薯) = 0

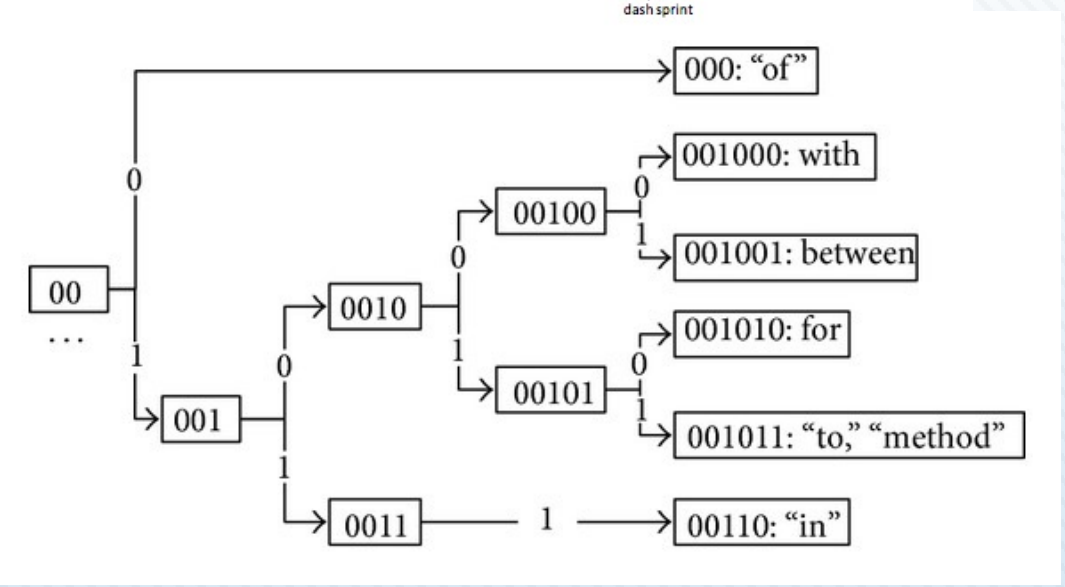


词袋模型 (Bag of Words Model)

传统解决方案

- 增加额外的特征
  - 词性特征：名词、动词、形容词
  - 前后缀特征：re-、-tion、-er
- 语义词典
  - WordNet、HowNet等
  - 如词的上位信息表示语义类别
  - 需要解决一词多义问题
  - 收录的词不全且更新慢
- 词聚类特征
  - 如Brown Clustering (Brown et al., CL 1992)





词的分布语义假设

- 分布语义假设（Distributional semantic hypothesis）
  - 词的含义可由其上下文词的分布进行表示
    - *You shall know a word by the company it keeps* -- Firth J.R. 1957

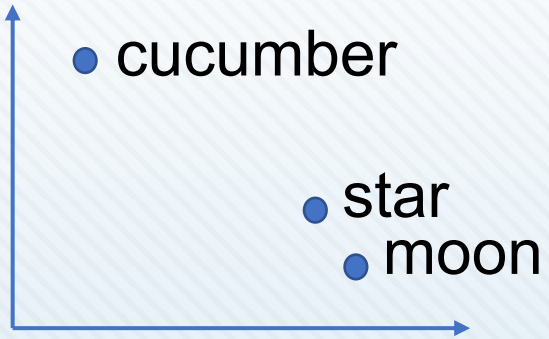
he curtains open and the moon shining in on the barely
ars and the cold , close moon " . And neither of the w
rough the night with the moon shining so brightly , it
made in the light of the moon . It all boils down , wr
surely under a crescent moon , thrilled by ice-white
sun , the seasons of the moon ? Home , alone , Jay pla
m is dazzling snow , the moon has risen full and cold
un and the temple of the moon , driving out of the hug
in the dark and now the moon rises , full and amber a
bird on the shape of the moon over the trees in front
But I could n't see the moon or the stars , only the
rning , with a sliver of moon hanging among the stars
they love the sun , the moon and the stars . None of
the light of an enormous moon . The plash of flowing w
man 's first step on the moon ; various exhibits , aer
the inevitable piece of moon rock . Housing The Airsh
oud obscured part of the moon . The Allied guns behind

词的分布（Distributional）表示

- 分布词向量

	shinning	bright	trees	dark	look
moon	38	45	2	27	12

- 语义相似度通过计算向量相似度获得



- 仍然存在高维、稀疏、离散的问题

分布表示的加权

降低高频词的权重

点互信息（Pointwise Mutual Information，PMI）

$$\mathrm{PMI}(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$$

我 喜欢 自然 语言 处理 。
我 爱 深度 学习 。
我 喜欢 机器 学习 。

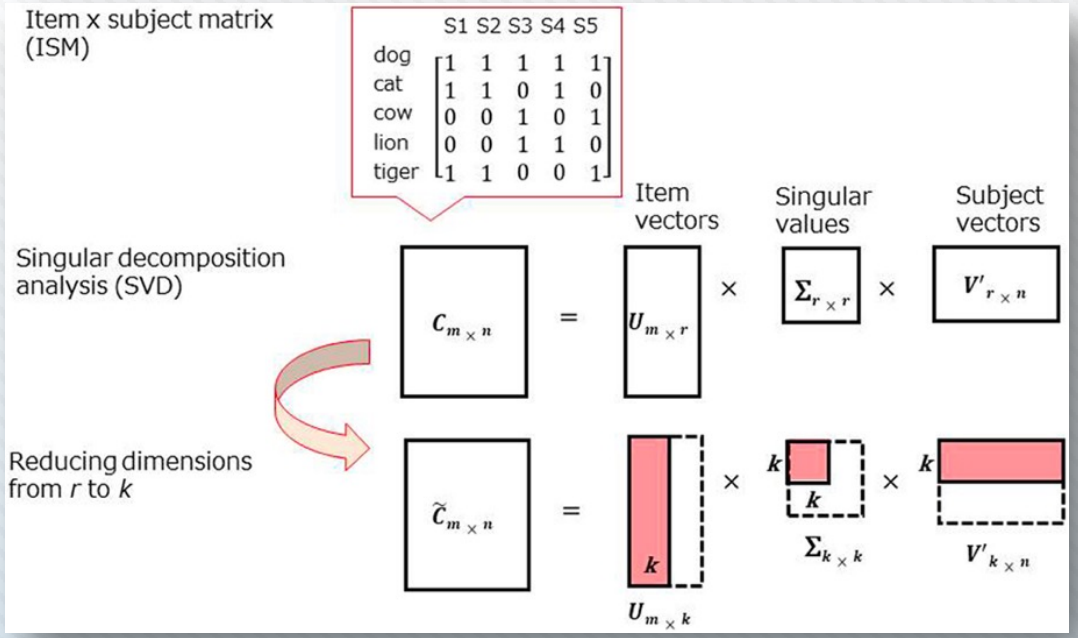
	我	喜欢	自然	语言	处理	爱	深度	学习	机器	。
我	0	2	1	1	1	1	1	2	1	3
喜欢	2	0	1	1	1	0	0	1	1	2
自然	1	1	0	1	1	0	0	0	0	1
语言	1	1	1	0	1	0	0	0	0	1
处理	1	1	1	1	0	0	0	0	0	1
爱	1	0	0	0	0	0	1	1	0	1
深度	1	0	0	0	0	1	0	1	0	1
学习	2	1	0	0	0	1	1	0	1	1
机器	1	1	0	0	0	0	0	1	0	1
。	3	2	1	1	1	1	1	2	1	0

```
def pmi(M, positive=True):
    col_totals = M.sum(axis=0) # 按列求和
    row_totals = M.sum(axis=1) # 按行求和
    total = col_totals.sum() # 总频次
    expected = np.outer(row_totals, col_totals) / total # 获得每个元素的分子
    M = M / expected
    with np.errstate(divide='ignore'): # 不显示log(0)的警告:
        M = np.log(M)
    M[np.isinf(M)] = 0.0  # 将log(0)置为0
    if positive:
        M[M < 0] = 0.0
    return M
```

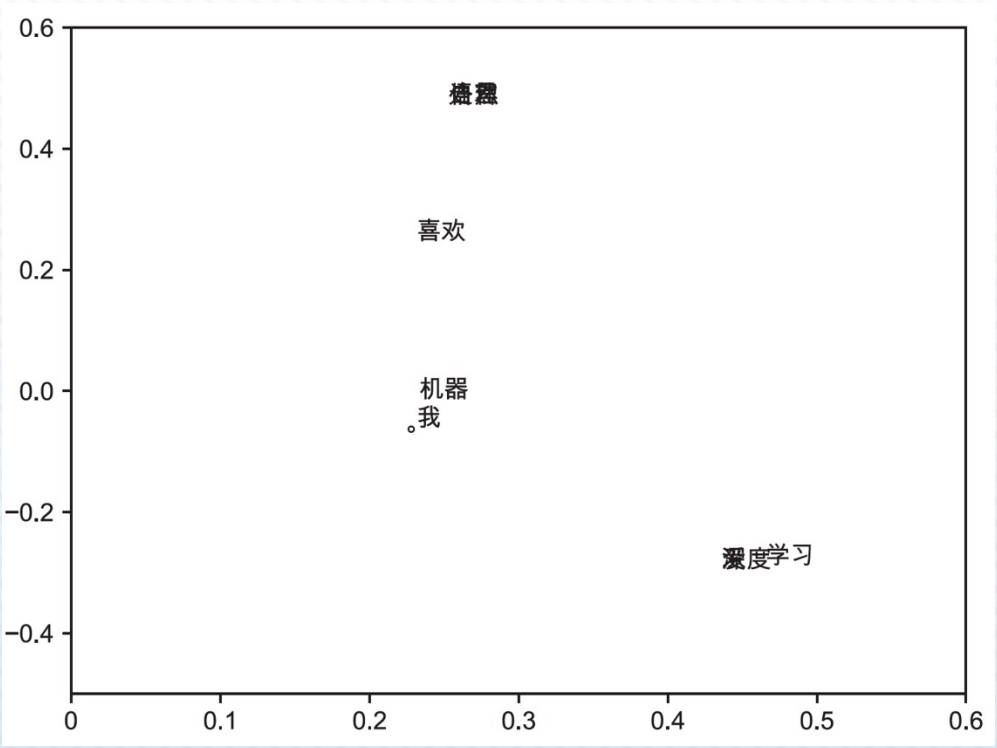
```
[[0.   0.17 0.06 0.06 0.06 0.28 0.28 0.28 0.28 0.28]
 [0.17 0.   0.43 0.43 0.43 0.   0.   0.   0.65 0.25]
 [0.06 0.43 0.   1.02 1.02 0.   0.   0.   0.   0.14]
 [0.06 0.43 1.02 0.   1.02 0.   0.   0.   0.   0.14]
 [0.06 0.43 1.02 1.02 0.   0.   0.   0.   0.   0.14]
 [0.28 0.   0.   0.   0.   0.   1.46 0.77 0.   0.36]
 [0.28 0.   0.   0.   0.   1.46 0.   0.77 0.   0.36]
 [0.42 0.09 0.   0.   0.   0.9  0.9  0.   0.9  0.  ]
 [0.28 0.65 0.   0.   0.   0.   0.   0.77 0.   0.36]
 [0.2  0.17 0.06 0.06 0.06 0.28 0.28 0.28 0.28 0.  ]]
```

分布表示的降维

- 避免稀疏性，反映高阶共现关系
  - 奇异值分解（Singular Value Decomposition，SVD）

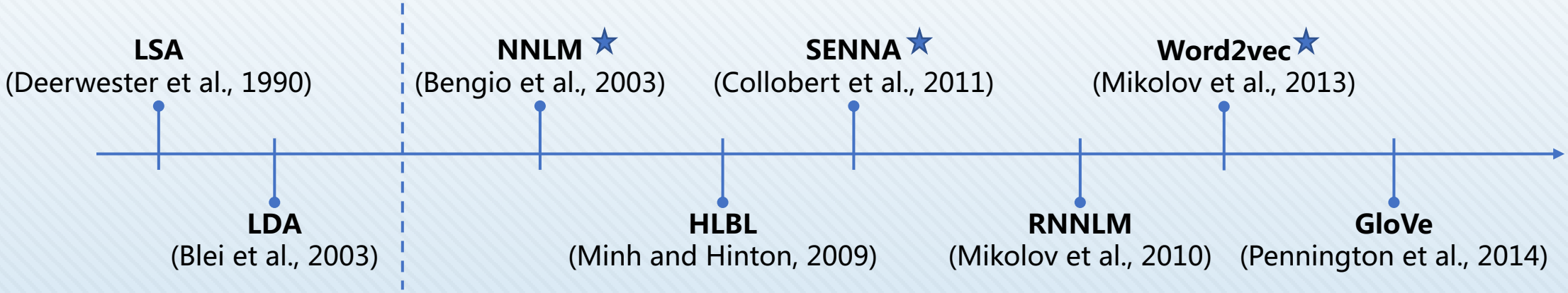



```
U, s, Vh = np.linalg.svd(M_pmi)
```



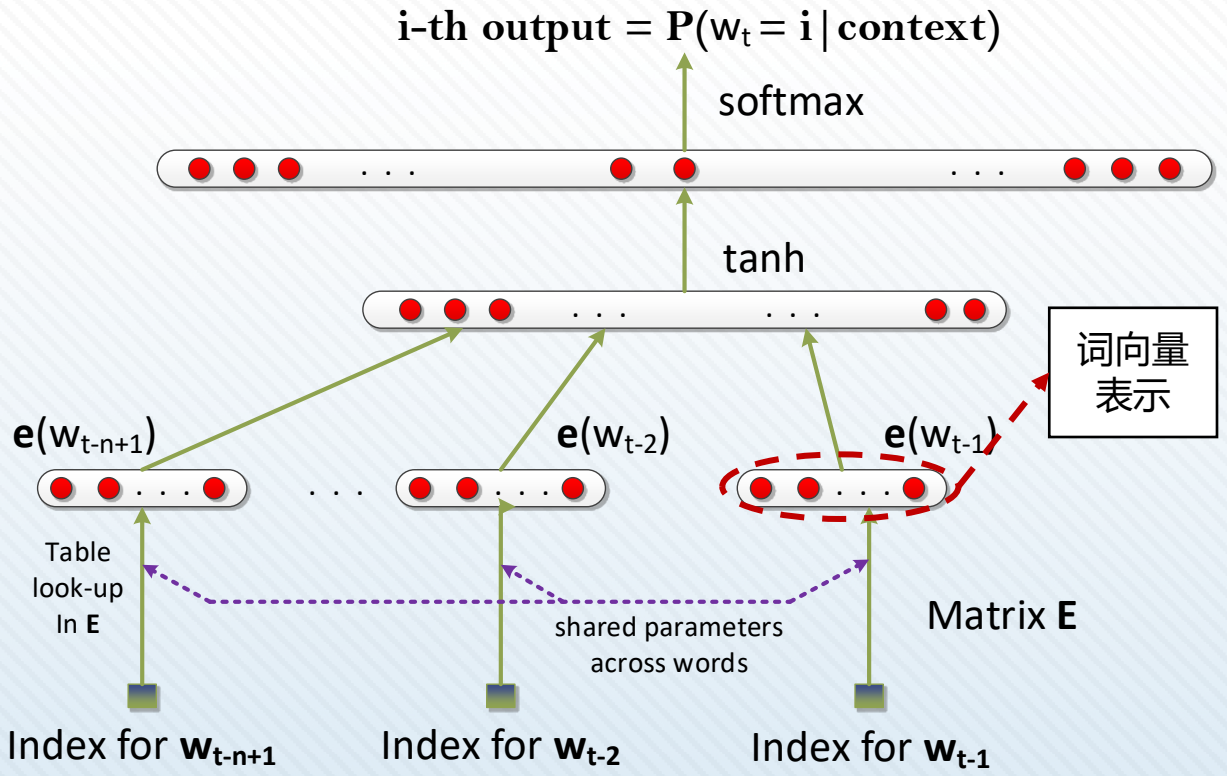
分布式（Distributed）词表示

- 分布表示的缺点
  - 训练速度慢，增加新语料库困难
  - 不易扩展到短语、句子表示
- 分布式表示直接使用低维、稠密、连续的向量表示词
  - 通过“自监督”的方法直接学习词向量
  - 也称词嵌入（Word Embedding）
- 发展历程



神经网络语言模型（NNLM）

- Neural Network Language Models (Bengio et al., JMLR 2003)
  - 根据前*n*-1个词预测第*n*个词（语言模型）
  - 模型结构为前向神经网络
  - 通过查表，获得词的向量表示
    - Word Embeddings/Vectors
    - 支撑图灵奖的重要工作
  - 通过梯度下降优化词向量表示



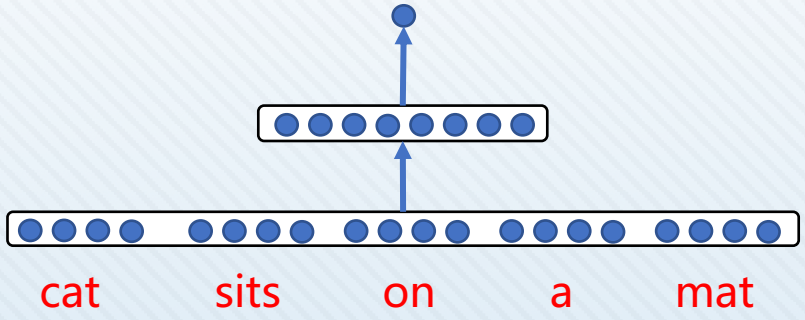
SENNA

- Semantic/syntactic Extraction using a Neural Network Architecture
  - Natural Language Processing (Almost) from Scratch (Collobert et al., JMLR 2011)
- “换词”的思想
  - 一个词和它的上下文构成正例
  - 随机替换掉该词构成负例

✚ cat sits on a mat

▬ cat sits Harbin a mat

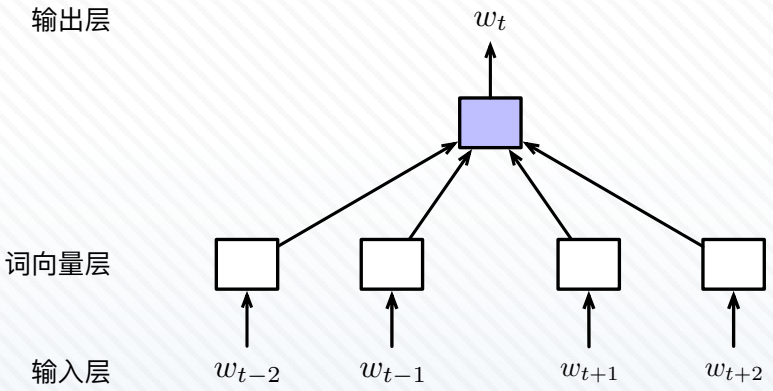
- 优化目标
  - score(cat sits on a mat) > score(cat sits Harbin a mat)
  - score的计算方式



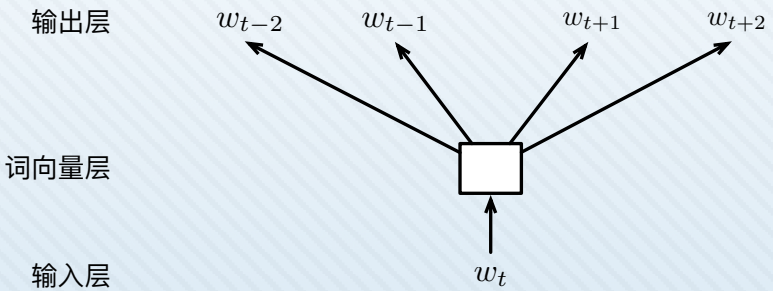
- 训练速度慢，在当年的硬件条件下需要训练**1个月**

Word2vec

- https://code.google.com/archive/p/word2vec/
  - Mikolov et al., ICLR 2013
- CBOW (Continuous Bag-of-Word)
  - 周围词向量加和预测中间的词
- Skip-Gram
  - 中间词预测周围词
- 训练速度快
  - 可利用大规模数据
  - 弥补了模型能力的不足

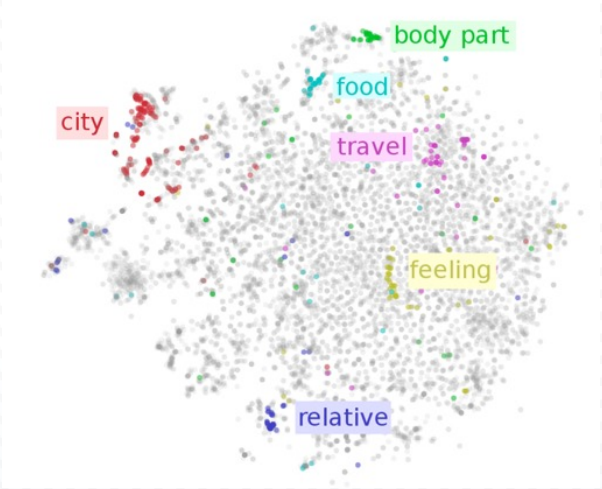


CBOW模型示意图

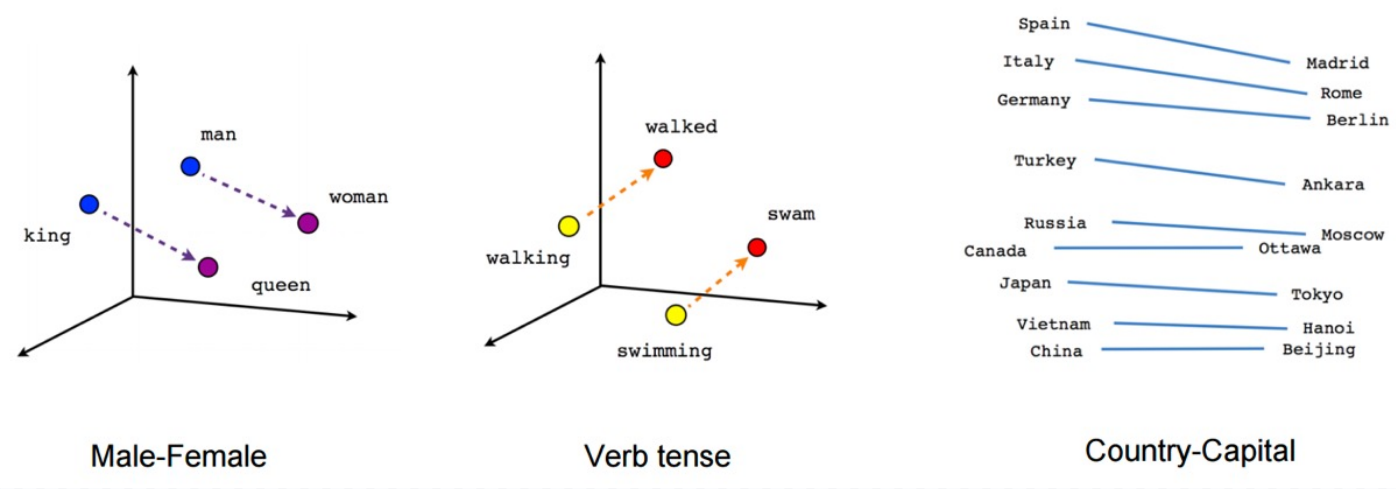


Skip-gram模型示意图

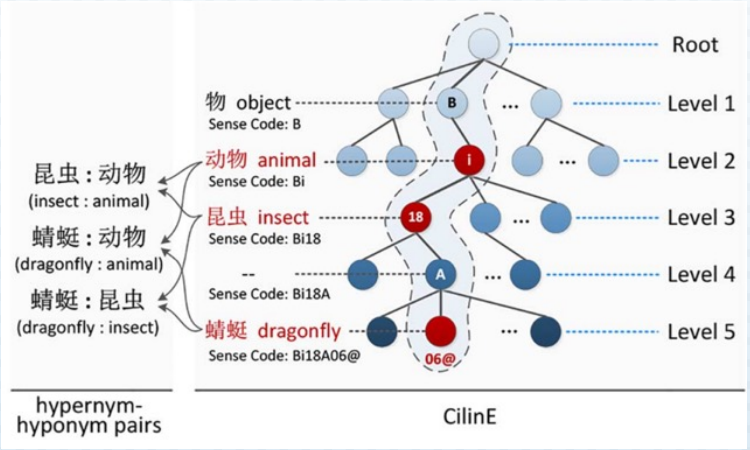
词向量的应用



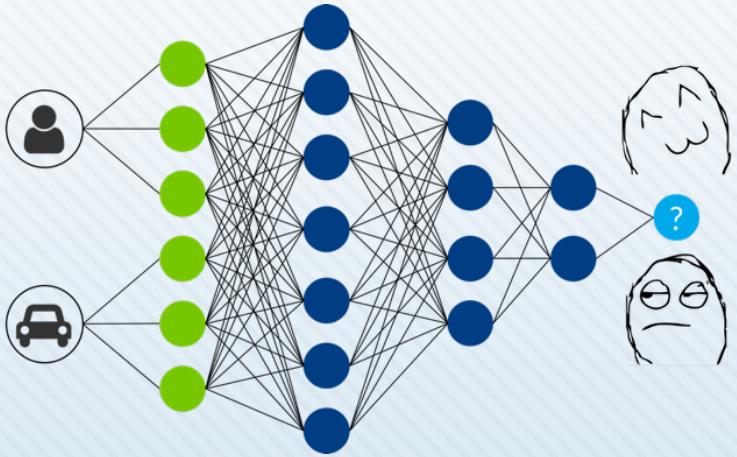
词义相似度计算



词类比关系计算



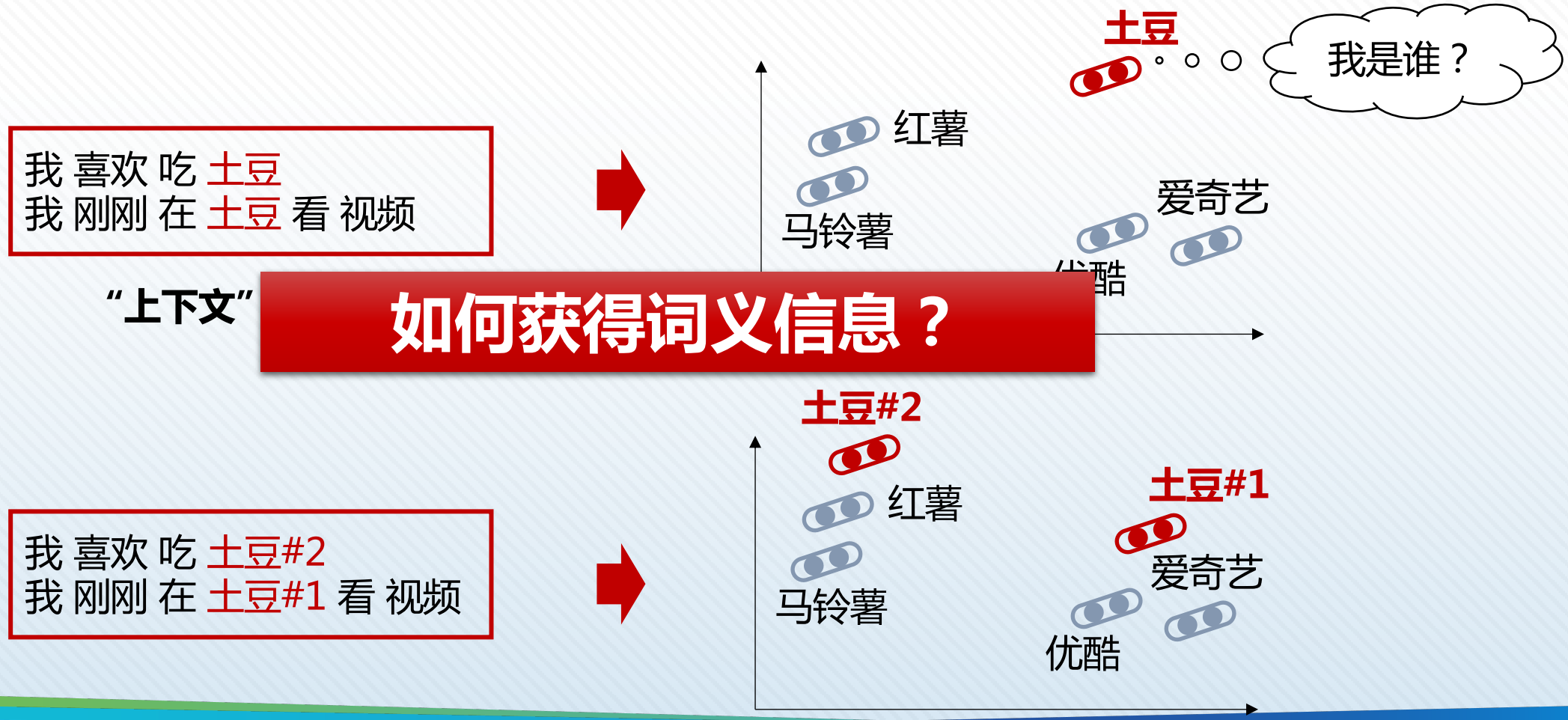
知识图谱补全



推荐系统

一词多义现象

- 静态词向量假设一个词由**唯一**的词向量表示
  - 无法处理一词多义现象



谢 谢！

理解语言，认知社会

以中文技术，助民族复兴

长按二维码，关注哈工大SCIR

微信号：HIT_SCIR

参考书

- 《自然语言处理：基于预训练模型的方法》
- 出版社：电子工业出版社
- 作者：车万翔，郭江，崔一鸣 著；刘挺主审
- 书号：ISBN 978-7-121-41512-8
- 出版时间：2021.7