

# AI大模型应用：NLP与大模型

2025年

*内部资料，请勿外传*

# 大模型开发生态

大模型开发	RAG智能问答：定制化知识库管理、检索和问答，代替人工客服； NLP信息抽取：文本分类、实体识别、关系抽取等，代替传统BERT模型； 决策与自动规划：生成决策步骤，并通过code interpreter多次验证；
大模型编排	开源工具：Dify、FastGPT、langflow、MaxKB、RagFlow 商业工具：Coze、千帆AppBuilder
大模型微调与部署	大模型微调：PEFT、LLaMA-Factory、DeepSpeed、unsloth 大模型部署：ollama、vllm、sglang、llama.cpp、TensorRT-LLM
大模型中间件	大模型应用API：langchain、llama_index、haystack 向量数据库：faiss、Milvus 文档存储与检索：ElasticSearch 缓存数据库：redis
底层大模型	开源大模型：Qwen、DeepSeek、GLM、LLaMA、Mistral 闭源（商业）大模型：OpenAI

# 检索模型

## 1.布尔模型 (Boolean Model):

1. 基本思想：使用布尔逻辑运算符（AND、OR、NOT）来连接检索查询中的关键词，从而筛选出匹配的文档。
2. 特点：检索结果是二元的，即文档要么匹配查询，要么不匹配。没有考虑文档和查询之间的相关性，只关注关键词的存在与否。

## 2.向量空间模型 (Vector Space Model):

1. 基本思想：将文档和查询表示为向量，其中向量的维度是词汇表中的词汇，每个维度上的值表示对应词汇在文档或查询中的权重。
2. 特点：考虑了文档和查询之间的相关性，通过计算向量之间的相似度来排序检索结果。常用的相似度计算方法包括余弦相似度。

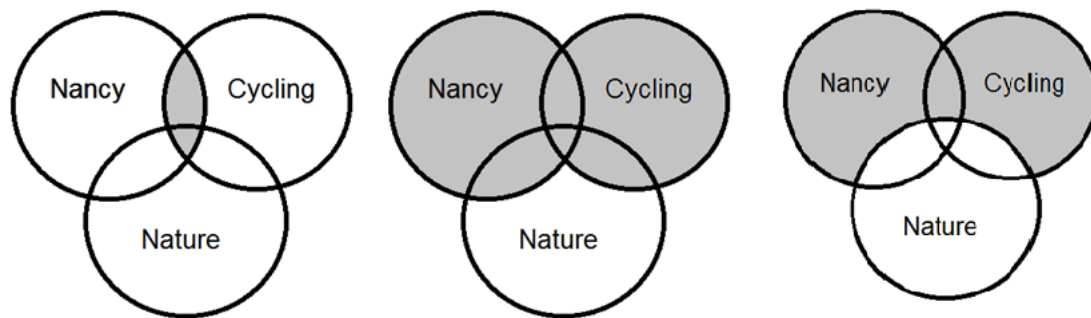
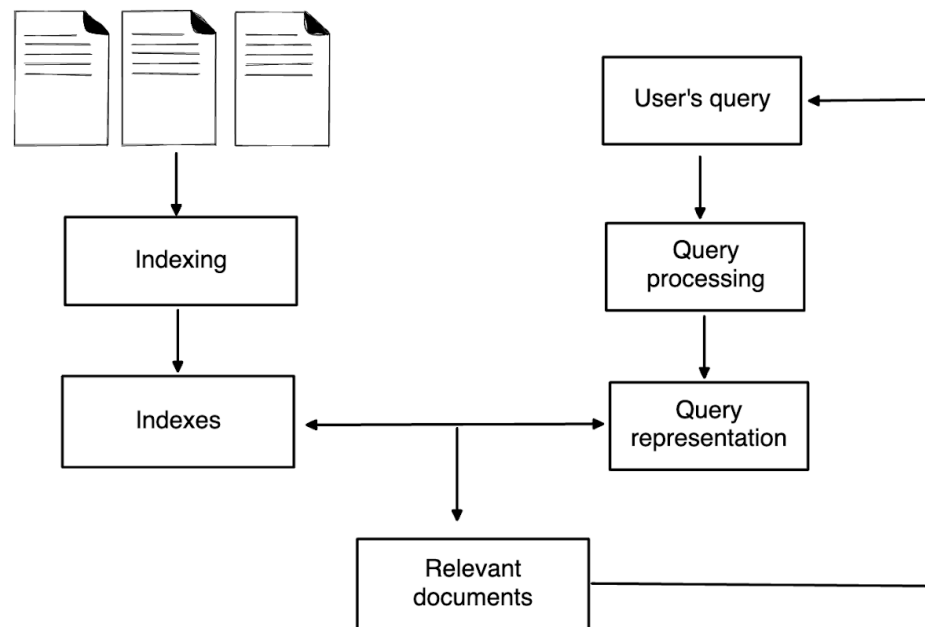
## 3.概率检索模型 (Probabilistic Model):

1. 基本思想：假设检索过程涉及到概率模型，其中文档和查询之间存在概率分布。
2. 特点：考虑了文档和查询之间的概率关系，能够更灵活地处理相关性。典型的概率检索模型包括Okapi BM25。

# 全文检索与布尔检索

**全文索引** (Full-Text Indexing) 是一种数据结构，它记录了文档集中所有词项的位置信息，目的是为了支持高效的 **全文检索**。可以把它想象成一本书的索引，但这个索引不是按章节或主题分类，而是包含了书中出现的每一个有意义的词，并指明了这些词在哪些页码（文档）的什么位置出现。

**布尔检索** 是一种 **信息检索模型**，它基于布尔逻辑 (AND, OR, NOT) 来处理用户查询并返回结果。纯粹的布尔检索模型不会对返回的文档进行相关性排序。所有匹配查询的文档都被视为同样相关。



# 全文检索与反向索引

**反向索引**（也称为**倒排索引**）是**全文检索最核心、最基础的数据结构**。它的名称“反向”或“倒排”源于它与传统数据库“正向索引”的对比。

- ✓ **正向索引**：传统数据库通常维护一个从“文档ID（或记录ID）”到“文档内容”的映射。当你查找特定内容时，需要遍历文档，检查其内容是否匹配。
- ✓ **反向索引**：恰好相反，它维护一个从“词项”到“包含该词项的文档列表”的映射。当你查找某个词时，可以直接从这个列表中找到所有相关的文档。

Doc 1				Doc 2			
I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.				So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:			
term	docID	term	docID	term	doc. freq.	→	postings lists
I	1	ambitious	2	ambitious	1	→	2
did	1	be	2	be	1	→	2
enact	1	brutus	1	brutus	2	→	1 → 2
julius	1	brutus	2	capitol	1	→	1
caesar	1	capitol	1	caesar	2	→	1 → 2
I	1	caesar	1	did	1	→	1
was	1	caesar	2	enact	1	→	1
killed	1	caesar	2	hath	1	→	2
i'	1	did	1	I	1	→	1
the	1	enact	1	i'	1	→	1
capitol	1	hath	1	it	1	→	2
brutus	1	I	1	julius	1	→	1
killed	1	I	1	killed	1	→	1
me	1	i'	1	let	1	→	2
so	2	it	2	me	1	→	1
let	2	julius	1	noble	1	→	2
it	2	killed	1	so	1	→	2
be	2	killed	1	the	2	→	1 → 2
with	2	let	2	told	1	→	2
caesar	2	me	1	you	1	→	2
the	2	noble	2	was	2	→	1 → 2
noble	2	so	2	with	1	→	2
brutus	2	the	1				
hath	2	the	2				
told	2	told	2				
you	2	you	2				
caesar	2	was	1				
was	2	was	2				
ambitious	2	with	2				

# 全文检索与反向索引

**词项词典 (Term Dictionary / Vocabulary) :** 存储了文档集中所有出现过的**唯一词项**。这些词项通常是经过分词、标准化、词干提取等预处理后的结果。

**倒排列表 (Postings List / Postings File) :** 词典中的每个词项都对应一个倒排列表。这个列表记录了所有包含该词项的**文档ID**。除了文档ID，倒排列表还可以包含更丰富的信息，以支持更高级的检索和排序功能：

- 词频 (Term Frequency, TF) :** 词项在特定文档中出现的次数。这对于计算文档与查询的相关性 (如TF-IDF) 至关重要。
- 位置信息 (Positions) :** 词项在文档中出现的具体位置。这对于支持短语查询 (如“quick brown fox”必须是连续出现的) 和临近查询非常有用。
- 偏移量 (Offsets) :** 词项在文档中的起始和结束字符位置，用于高亮显示搜索结果中的关键词。
- 字段信息 (Field Information) :** 如果文档有结构 (如标题、正文、作者)，可以记录词项出现在哪个字段中。

**Doc 1**  
I did enact Julius Caesar: I was killed  
i' the Capitol; Brutus killed me.

**Doc 2**  
So let it be with Caesar. The noble Brutus  
hath told you Caesar was ambitious:

term	docID	term	docID	term	doc. freq.	→	postings lists
I	1	ambitious	2	ambitious	1	→	2
did	1	be	2	be	1	→	2
enact	1	brutus	1	brutus	2	→	1 → 2
julius	1	brutus	2	capitol	1	→	1
caesar	1	capitol	1	caesar	2	→	1 → 2
I	1	caesar	1	did	1	→	1
was	1	caesar	2	enact	1	→	1
killed	1	caesar	2	hath	1	→	2
i'	1	did	1	I	1	→	1
the	1	enact	1	i'	1	→	1
capitol	1	hath	1	it	1	→	2
brutus	1	I	1	julius	1	→	1
killed	1	I	1	killed	1	→	1
me	1	i'	1	let	1	→	2
so	2	it	2	me	1	→	1
let	2	julius	1	noble	1	→	2
it	2	killed	1	so	1	→	2
be	2	killed	1	the	2	→	1 → 2
with	2	let	2	told	1	→	2
caesar	2	me	1	you	1	→	2
the	2	noble	2	was	2	→	1 → 2
noble	2	so	2	with	1	→	2
brutus	2	the	1				
hath	2	the	2				
told	2	told	2				
you	2	you	2				
caesar	2	was	1				
was	2	was	2				
ambitious	2	with	2				

# 倒排索引

倒排索引用来记录有哪些文档包含了某个单词。

- ✓ 以词(Term)为核心对文档进行索引;
- ✓ 记录包含某个词的文档编号、该词在每个文档中出现的次数(TF)及出现位置等信息;

关键点:

- ✓ 是不是所有的单词都被索引?
- ✓ 是不是所有的单词重要性都一样?

Doc 1

... news about

Doc 2

... news about  
organic food  
campaign ...

Doc 3

... news of presidential campaign ...  
... presidential candidate ...

Dictionary (or lexicon)

Term	# docs	Total freq
news	3	3
campaign	2	2
presidential	1	2
food	1	1
...	...	...

Postings

Doc ID	Freq	Position
1	1	p1
2	1	p2
3	1	p3
2	1	p4
3	1	p5
3	2	p6, p7
2	1	p8
...	...	
...	...	

# TFIDF与BM25

TF-IDF (Term Frequency-Inverse Document Frequency) 和BM25 (Best Matching 25) 都是基于词频 (TF) 的方法。

## TF-IDF (Term Frequency-Inverse Document Frequency) :

### 1. Term Frequency (TF):

- 表示一个词在文档中出现的频率。
- 计算公式:  $TF(t, d) = \frac{\text{词}t\text{在文档}d\text{中的出现次数}}{\text{文档}d\text{中所有词的总数}}$

### 2. Inverse Document Frequency (IDF):

- 表示一个词对整个文集的重要性。
- 计算公式:  $IDF(t, D) = \log \left( \frac{\text{文集}D\text{中文档的总数}}{\text{包含词}t\text{的文档数}+1} \right)$
- 分母加一是为了避免分母为零的情况。

### 3. TF-IDF Weight:

- 将TF和IDF相乘, 得到一个词在文档中的权重。
- 计算公式:  $TF-IDF(t, d, D) = TF(t, d) \times IDF(t, D)$

## BM25 (Best Matching 25) :

### 1. Term Frequency (TF):

- 同样表示一个词在文档中出现的频率。

### 2. Inverse Document Frequency (IDF):

- 在BM25中, IDF的计算稍有不同, 使用了饱和函数, 避免在极端情况下IDF过于极端。
- 计算公式:  $IDF(t, D) = \log \left( \frac{\text{文集}D\text{中文档的总数}-\text{包含词}t\text{的文档数}+0.5}{\text{包含词}t\text{的文档数}+0.5} \right)$

### 3. BM25 Weight:

- 计算公式:  $BM25(t, d, D) = IDF(t, D) \times \frac{TF(t, d) \times (k_1 + 1)}{TF(t, d) + k_1 \times \left( 1 - b + b \times \frac{\text{文档}d\text{的长度}}{\text{平均文档长度}} \right)}$
- $k_1$  和  $b$  是调节参数。

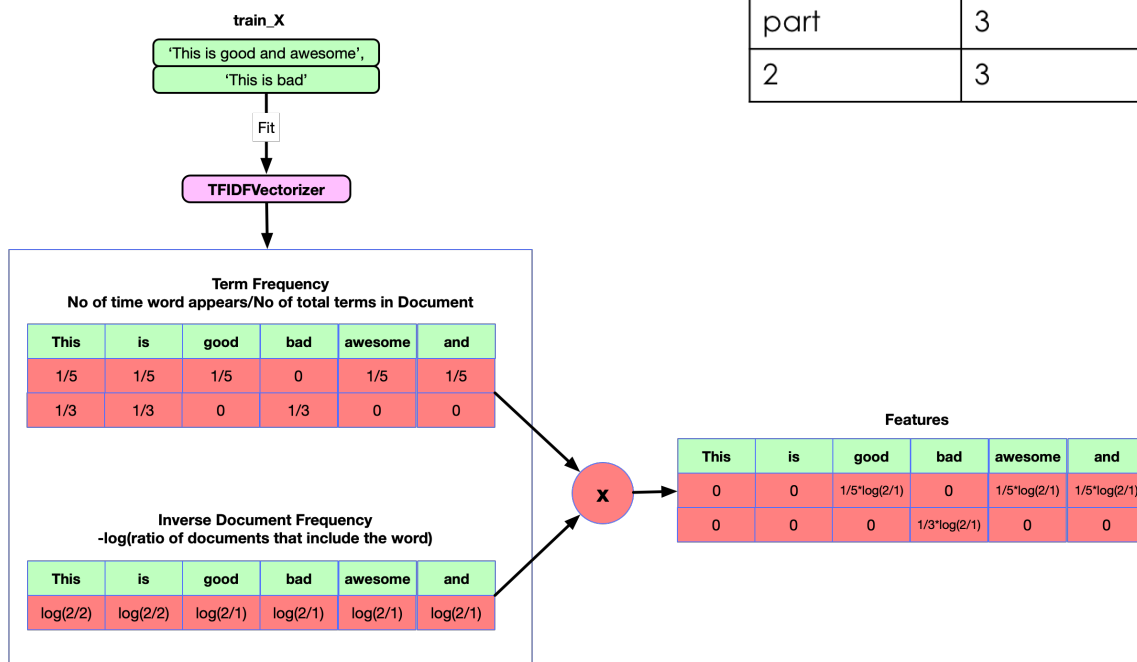
# 自然语言处理：倒排、TFIDF和BM25

Corpus

Document ID	Content
1	The Shawshank Redemption
2	The Godfather
3	The Godfather: Part 2

Inverted Index

Key	Value
the	1, 2, 3
shawshank	1
redemption	1
godfather	2, 3
part	3
2	3



$$\text{score}(D, Q) = \sum_{t \in Q} \frac{f_{t,D} \cdot (k_1 + 1)}{f_{t,D} + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgl}})} \cdot \log \frac{N - n_t + 0.5}{n_t + 0.5}$$

sum the scores for each query term

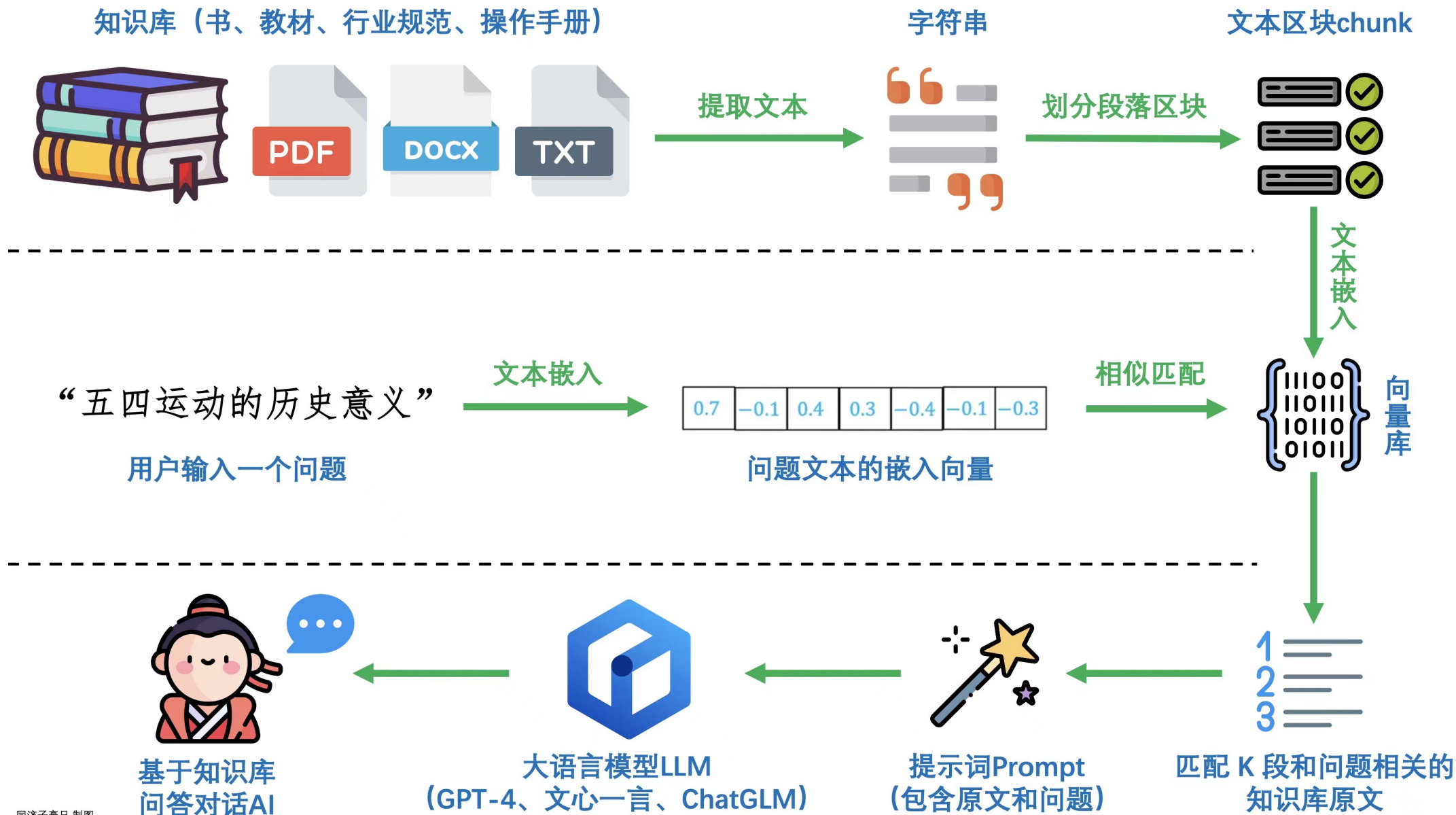
forget about this: it doesn't affect score relationships so Lucene took it out

probabilistic flavor of IDF: Lucene adds a 1 inside the log, making it basically the same as traditional IDF

term frequency saturation trick

adjust saturation curve based on document length

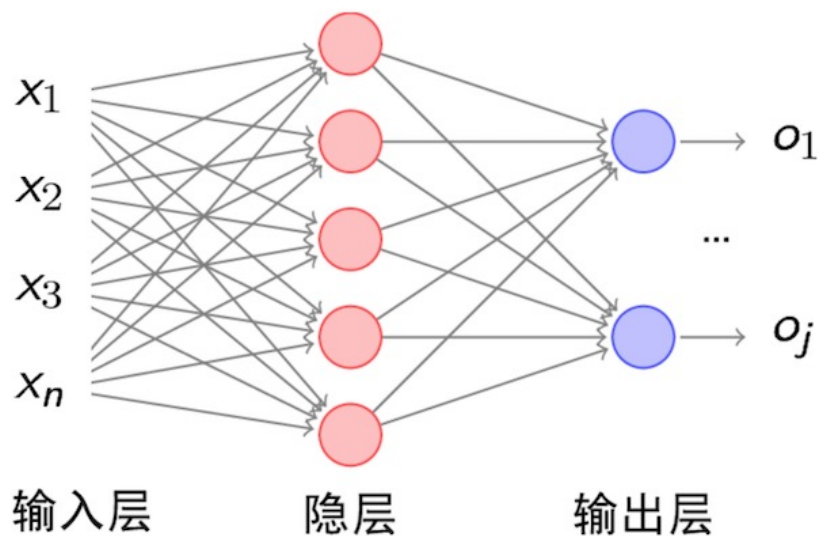
# 大模型应用：RAG



# 深度学习基础

深度学习（Deep Learning）是包含多层神经元的结构，与人脑神经元的结构类似。

- ✓ 深度学习是端到端的计算，从输入到输出；
- ✓ 深度学习是一个有向计算图，图的节点表示计算过程；

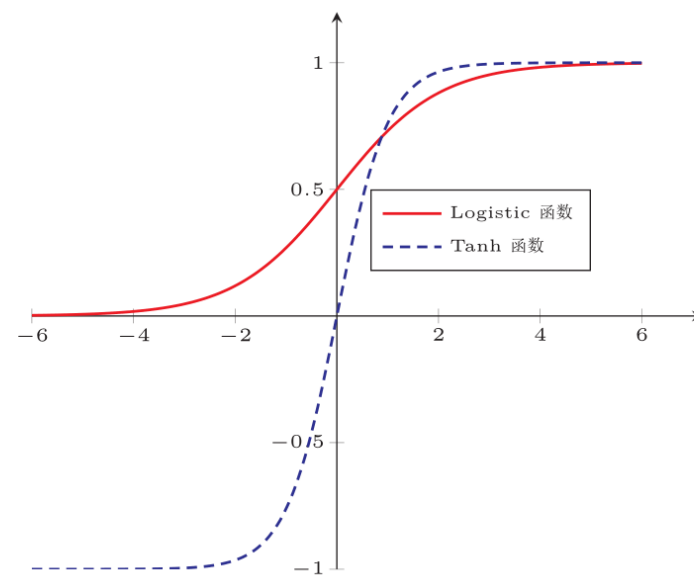
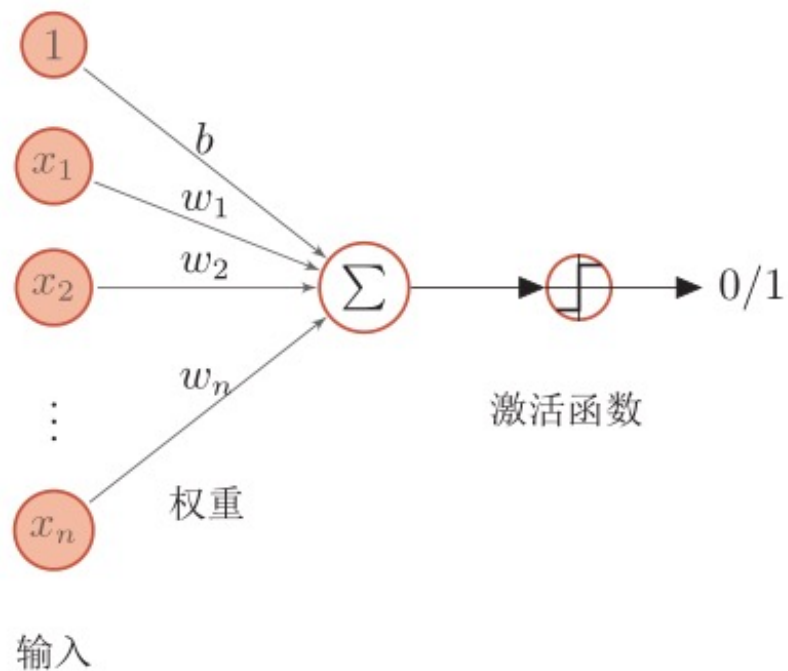
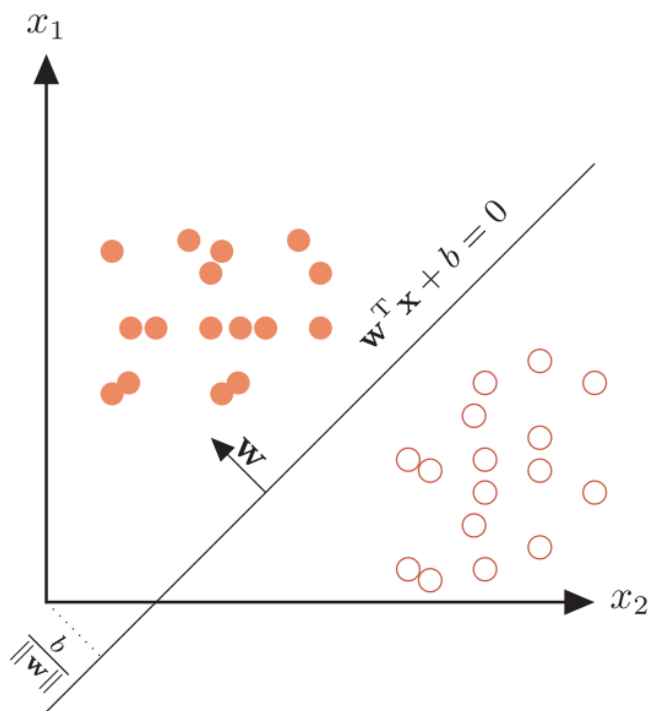


输入层：数据输入，假设N维数据；

隐含层：中间计算节点；

输出层：最终结果输出；

# 深度学习基础：神经元



# 深度学习基础：神经元

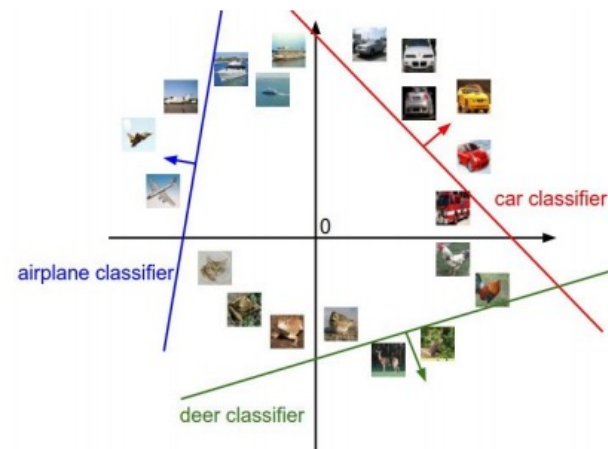
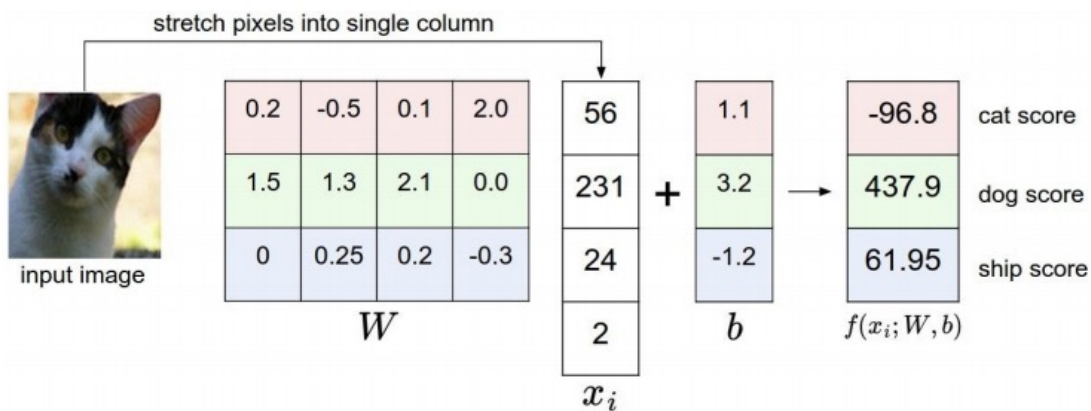


[32x32x3]

array of numbers 0...1  
(3072 numbers total)

image parameters  
 $f(\mathbf{x}, \mathbf{W})$

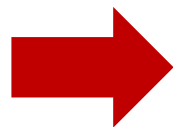
10 numbers, indicating  
class scores



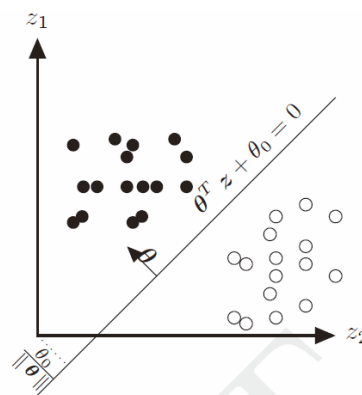
# 深度学习基础：神经元

$D_1$ : “我喜欢读书”

$D_2$ : “我讨厌读书”



	我	喜欢	讨厌	读书
$D_1$	1	1	0	1
$D_2$	1	0	1	1



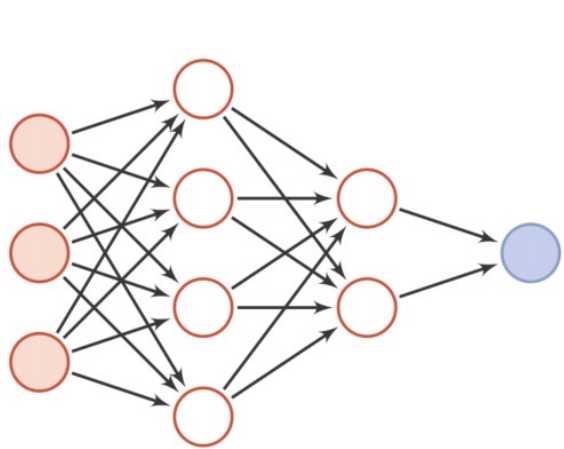
+

-

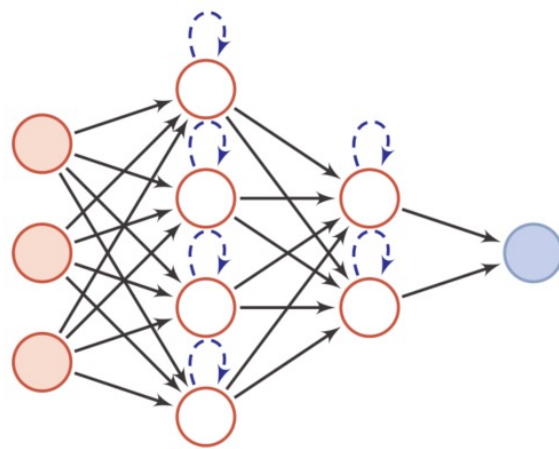
# 深度学习网络结构

人工神经网络主要由大量的神经元以及它们之间的有向连接构成。因此考虑三方面：

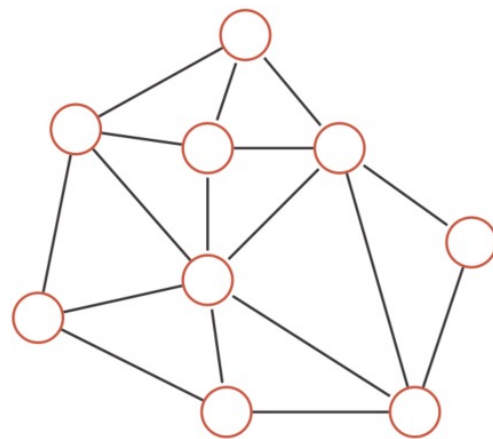
- ✓ 神经元的激活规则：主要是指神经元输入到输出之间的映射关系，一般为非线性函数。
- ✓ 网络的拓扑结构：不同神经元之间的连接关系。
- ✓ 学习算法：通过训练数据来学习神经网络的参数。



(a) 前馈网络



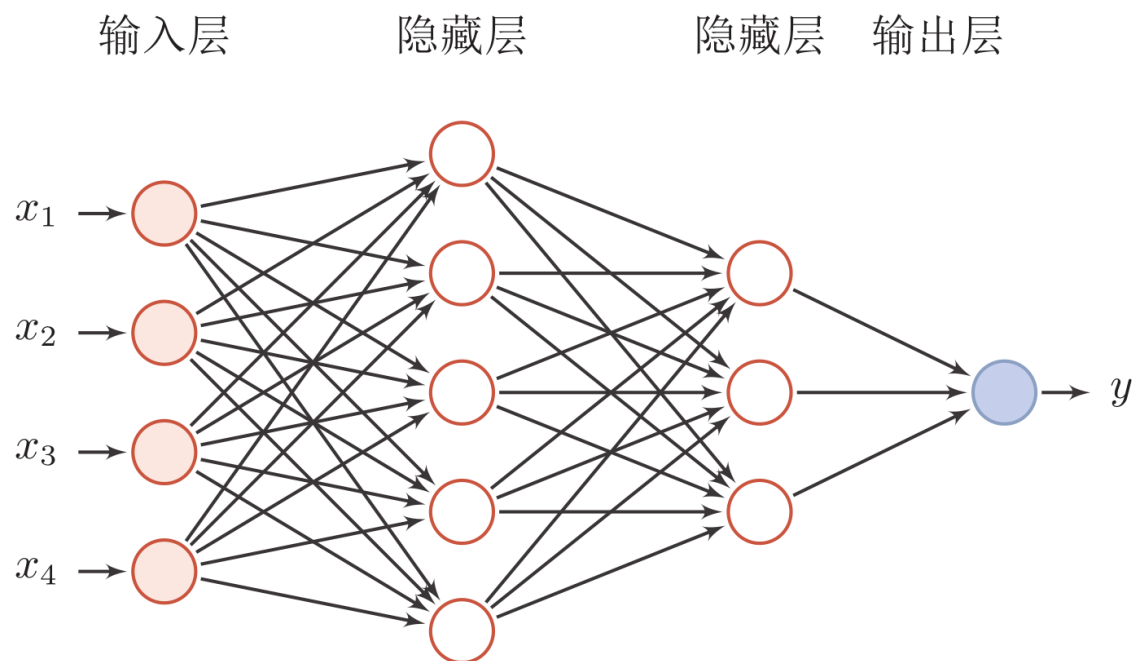
(b) 反馈网络



(c) 图网络

# 深度学习网络结构

在前馈神经网络中，各神经元分别属于不同的层。整个网络中无反馈，信号从输入层向输出层单向传播，可用一个有向无环图表示。



## ► 模型

$$y = f^5(f^4(f^3(f^2(f^1(x)))))$$

## ► 学习准则

$$L(y, y^*)$$

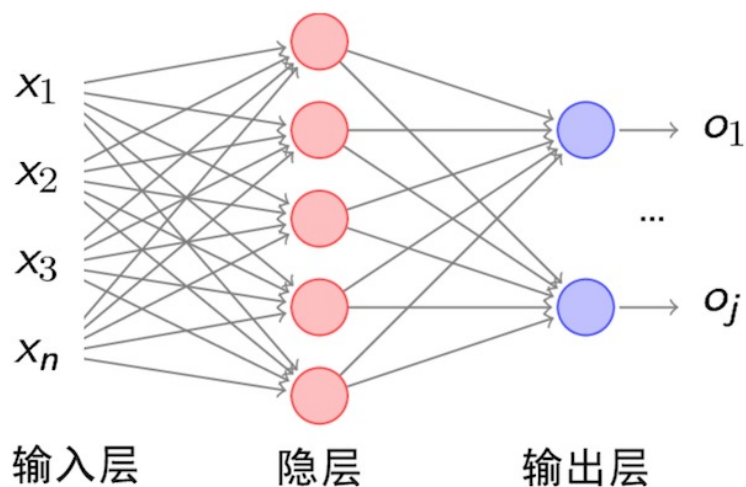
## ► 优化

### ► 梯度下降

$$\frac{\partial L(y, y^*)}{\partial f^1} = \frac{\partial f^2}{\partial f^1} \times \frac{\partial f^3}{\partial f^2} \times \frac{\partial f^4}{\partial f^3} \times \frac{\partial f^5}{\partial f^4} \times \frac{\partial L(y, y^*)}{\partial f^5}$$

链式法则，可以自动计算！

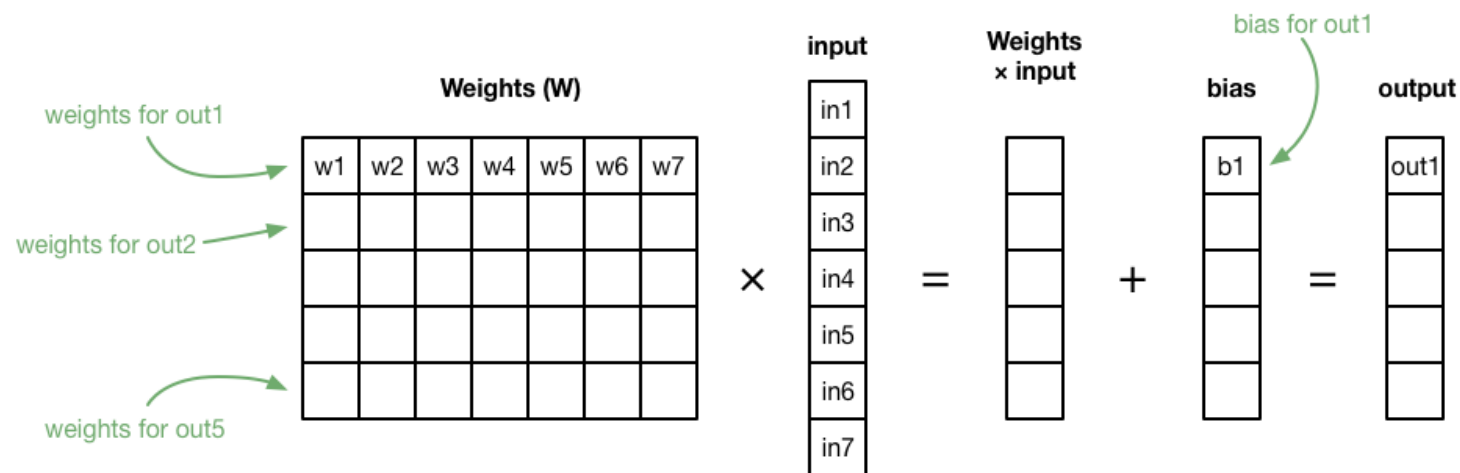
# 深度学习网络结构



输入层：数据输入，假设N维数据；

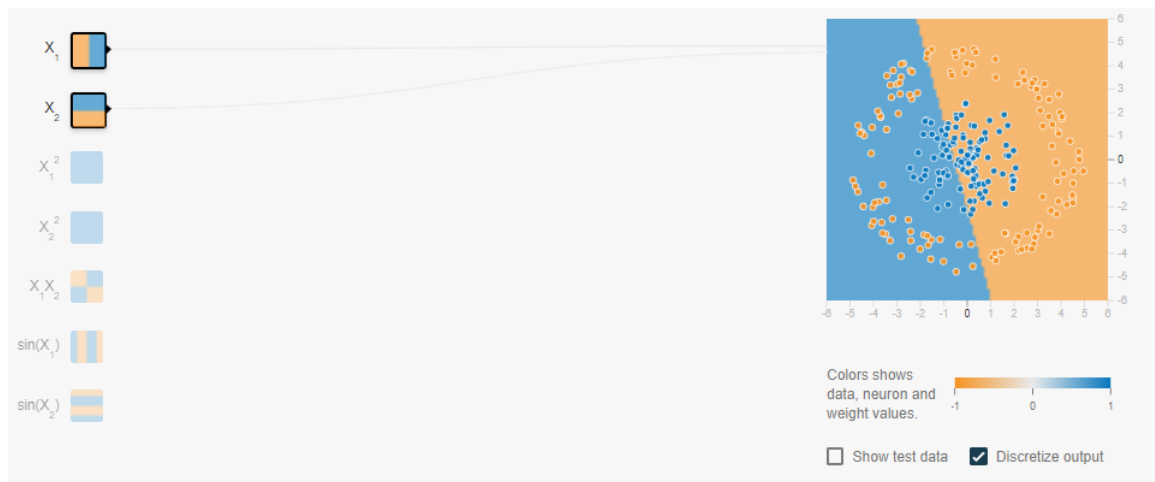
隐含层：中间计算节点；

输出层：最终结果输出；

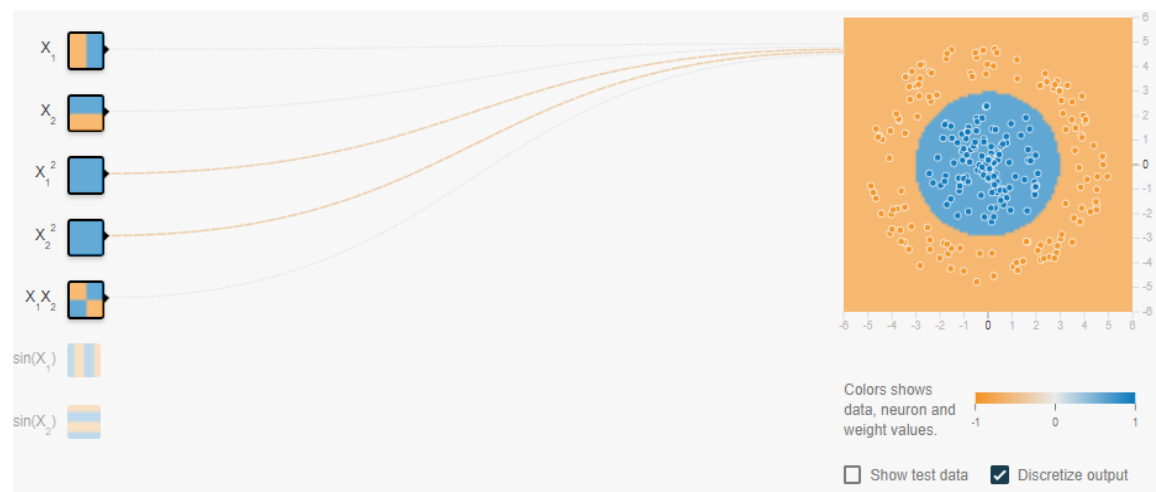


$$\text{output} = f(\text{Weights} \times \text{input} + \text{bias})$$

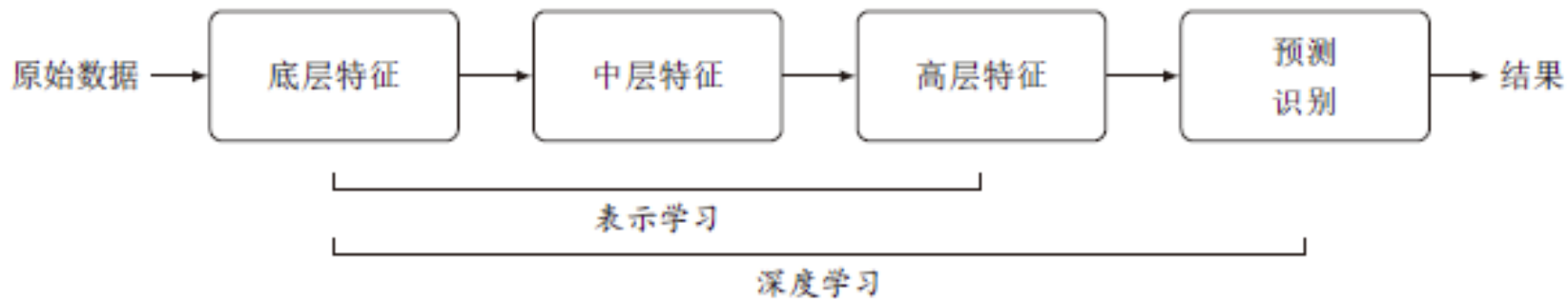
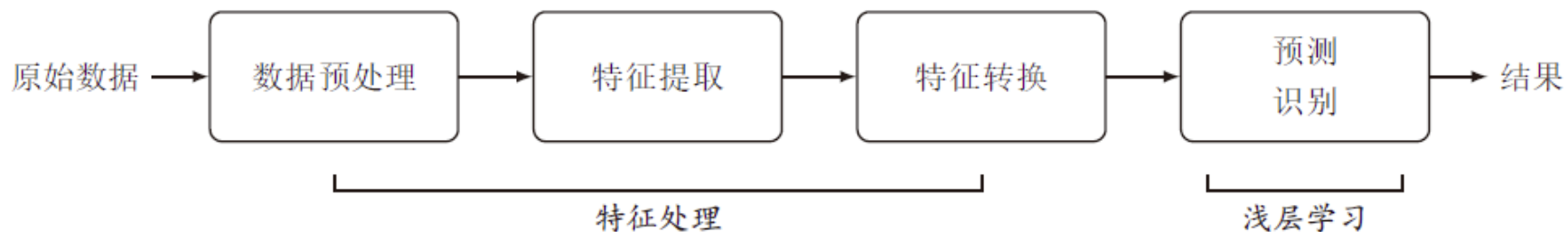
# 深度学习基础：非线性问题



<http://playground.tensorflow.org>

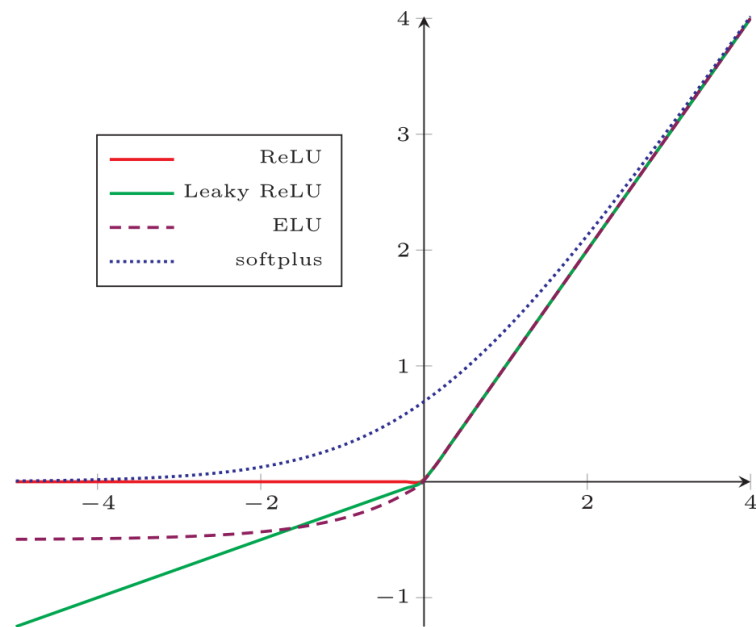


# 深度学习基础：非线性问题

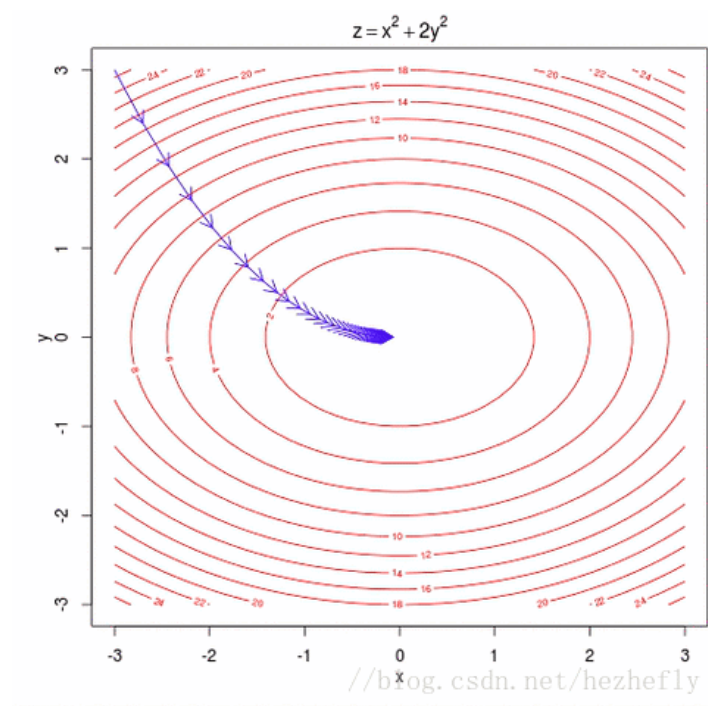
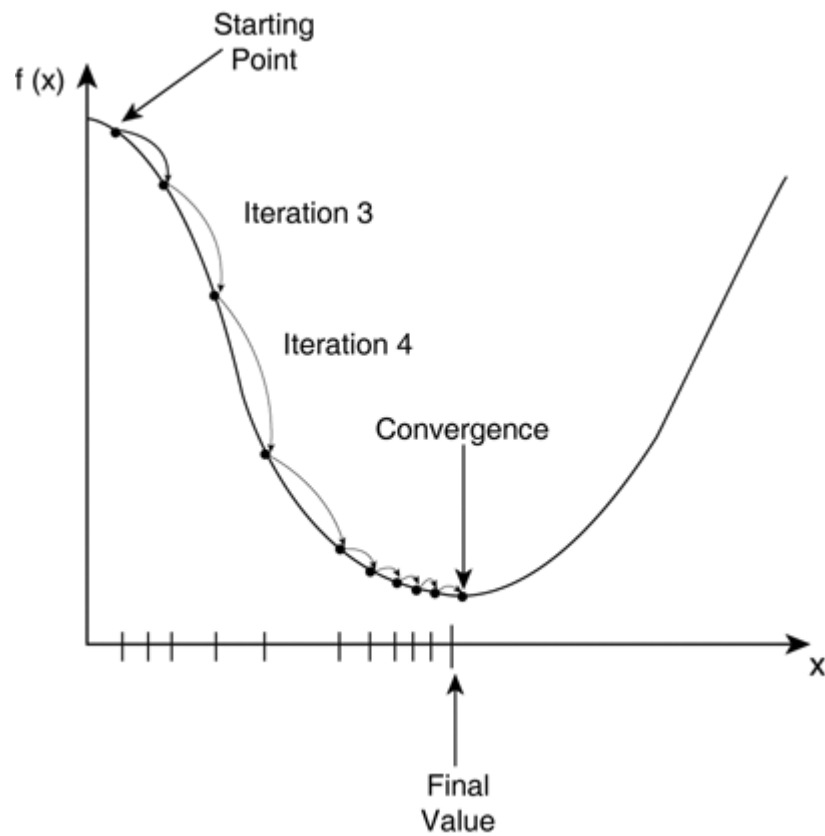


# 深度学习基础：非线性问题

激活函数	函数	导数
Logistic 函数	$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh 函数	$f(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ReLU	$f(x) = \max(0, x)$	$f'(x) = I(x > 0)$
ELU	$f(x) = \max(0, x) + \min(0, \gamma(\exp(x) - 1))$	$f'(x) = I(x > 0) + I(x \leq 0) \cdot \gamma \exp(x)$
SoftPlus 函数	$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$



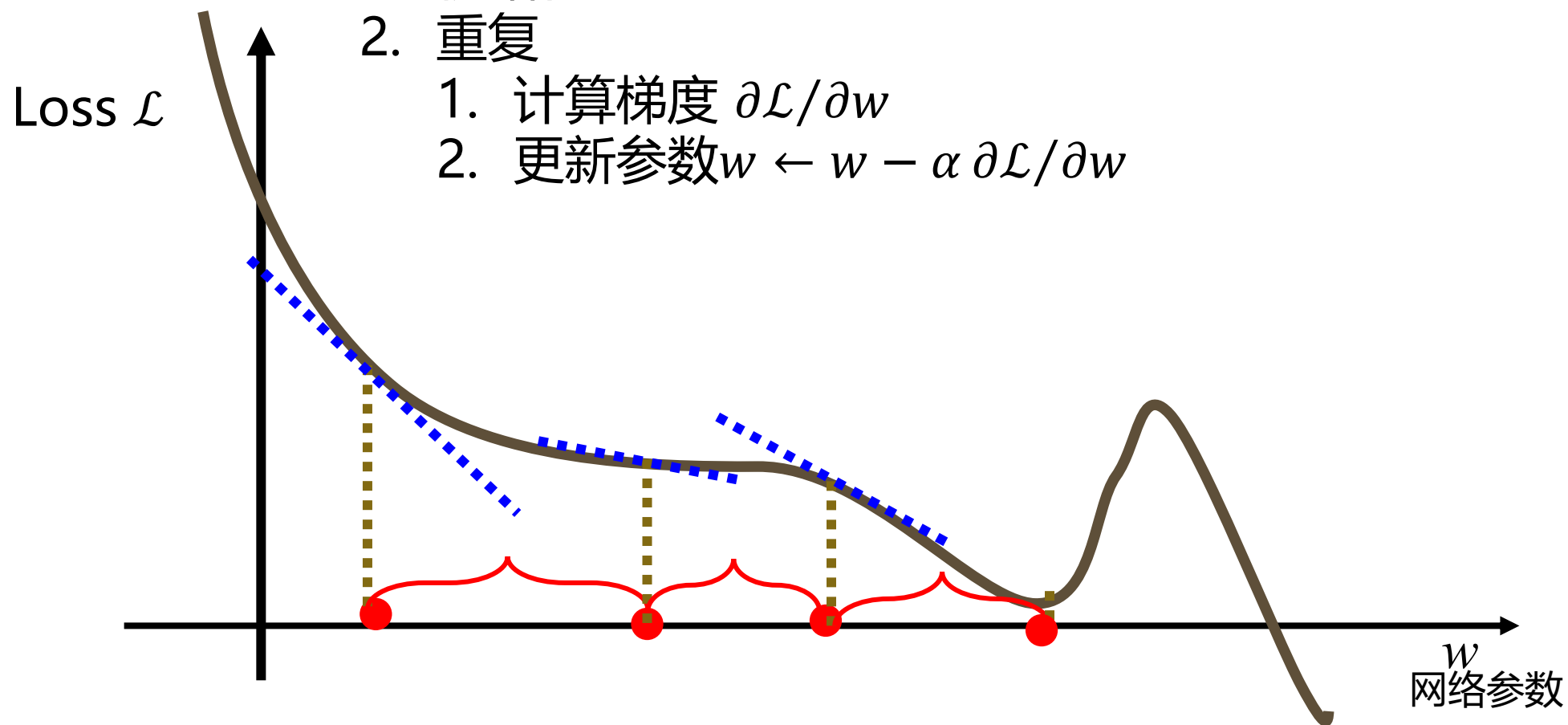
# 深度学习优化



# 深度学习优化

$$\text{梯度: } \frac{\partial f(w)}{\partial w} = \lim_{\Delta w \rightarrow 0} \frac{f(w + \Delta w) - f(w)}{\Delta w}$$

1. 初始化 $w$
2. 重复
  1. 计算梯度  $\partial \mathcal{L} / \partial w$
  2. 更新参数  $w \leftarrow w - \alpha \partial \mathcal{L} / \partial w$

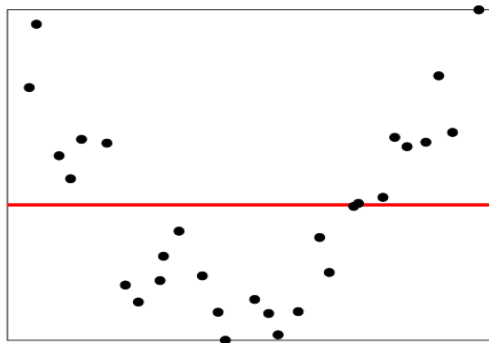


# 深度学习优化

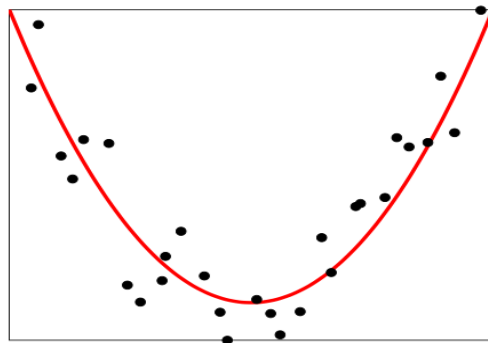
输入: 训练集  $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}, n = 1, \dots, N$ , 验证集  $\mathcal{V}$ , 学习率  $\alpha$

```
1 随机初始化  $\theta$ ;  
2 repeat  
3   对训练集  $\mathcal{D}$  中的样本随机重排序;  
4   for  $n = 1 \dots N$  do  
5     从训练集  $\mathcal{D}$  中选取样本  $(\mathbf{x}^{(n)}, y^{(n)})$ ;  
6     // 更新参数  
7      $\theta \leftarrow \theta - \alpha \frac{\partial \mathcal{L}(\theta; x^{(n)}, y^{(n)})}{\partial \theta}$ ;  
8   end  
9 until 模型  $f(\mathbf{x}, \theta)$  在验证集  $\mathcal{V}$  上的错误率不再下降;  
输出:  $\theta$ 
```

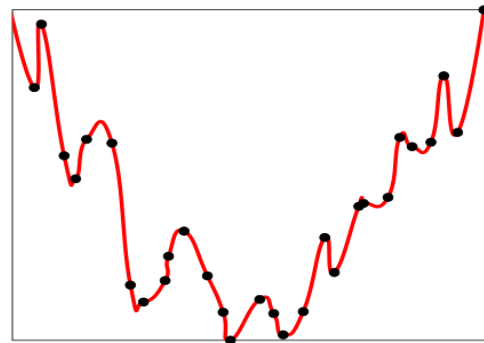
欠拟合



正常



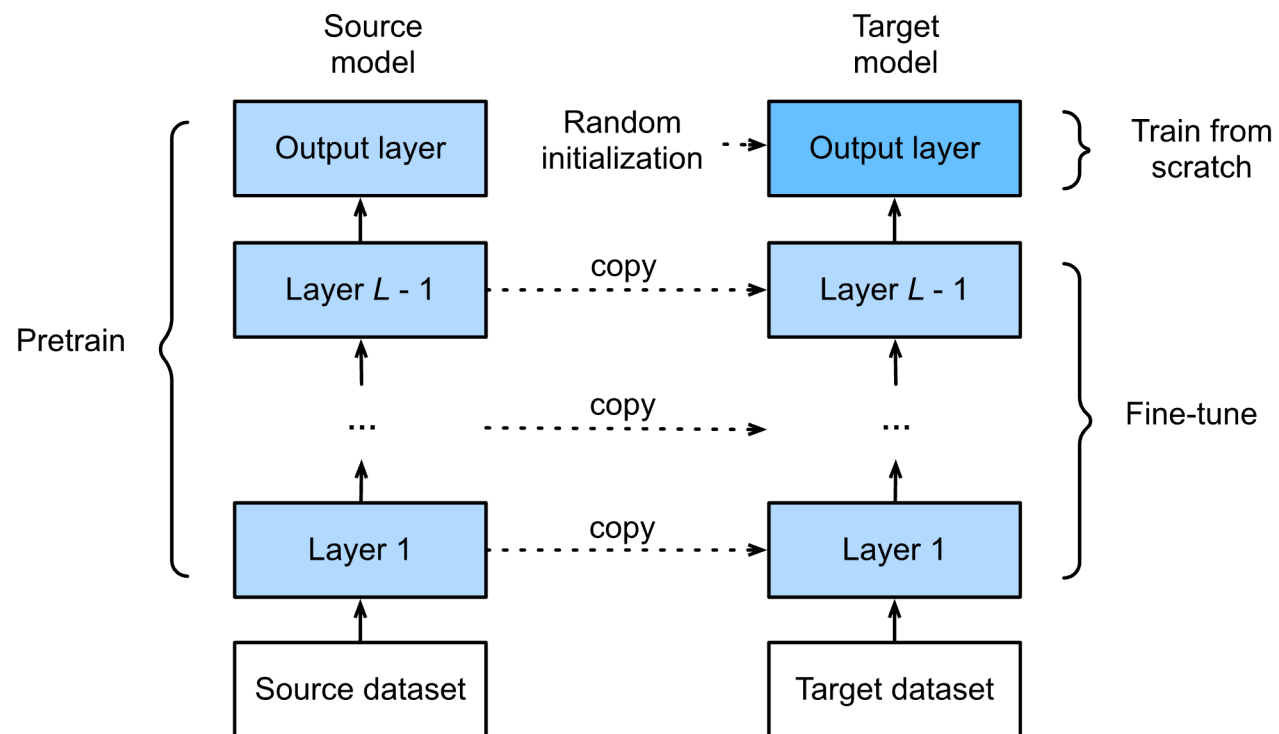
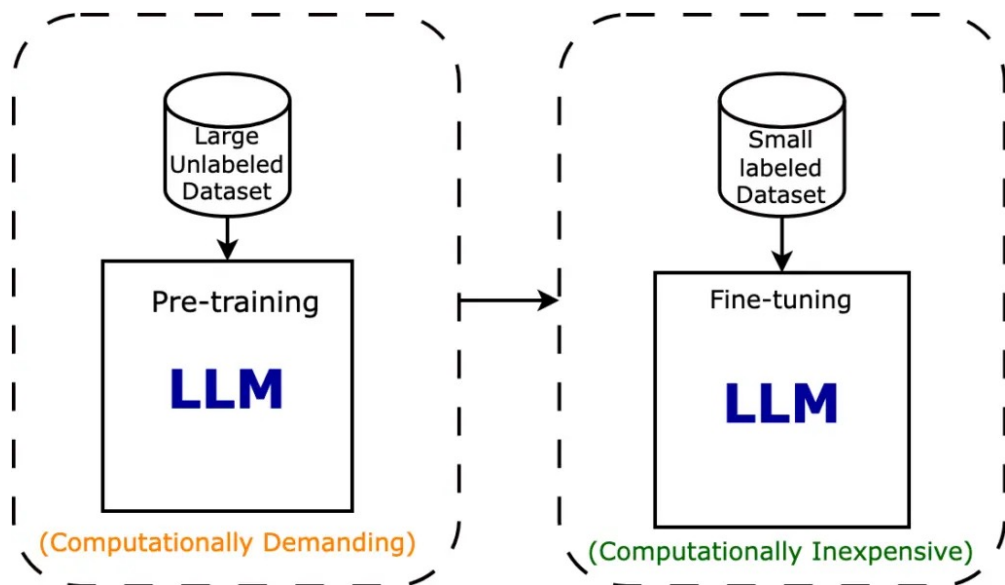
过拟合



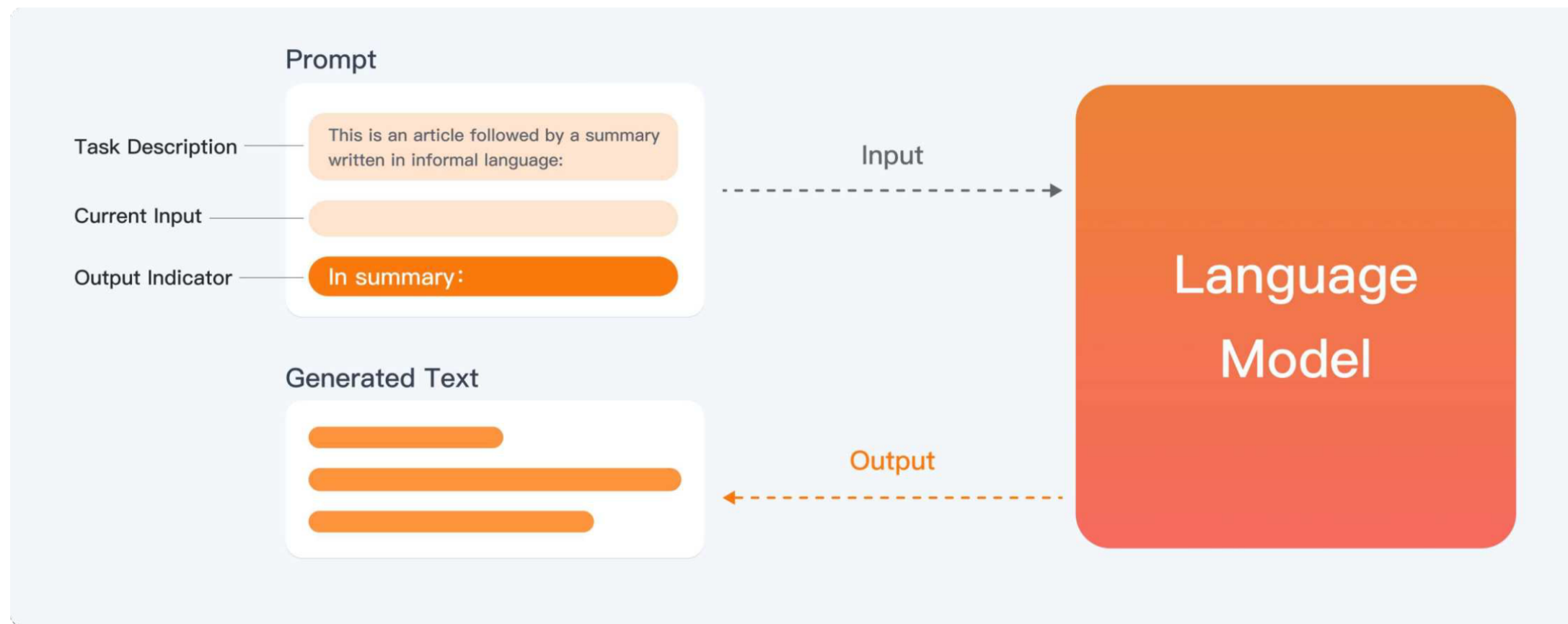
# 迁移学习

- ✓ **预训练 (Pre-training)**: 这是训练的一种特殊形式，通常用于处理**没有大量标签数据**的问题。研究人员会用一个**大型的通用数据集**（比如整个互联网的文本或图片）来训练一个庞大的模型。这个模型在预训练阶段学习到了通用的语言理解或视觉特征提取能力。
- ✓ **微调 (Fine-tuning)**: 在模型完成预训练后，如果我们想让它完成某个**特定的、更具体的任务**（比如文本情感分析或医学图像识别），就需要进行微调。这个过程使用少量**针对特定任务的标签数据**，在预训练模型的基础上，继续对模型的参数进行调整。微调的好处是，模型已经有了强大的通用能力，只需要少量数据就能快速适应新任务，而不需要从头开始训练。
- ✓ **迁移学习 (Transfer Learning)**: 这是一个更广泛的概念，它描述了**将一个模型从一个任务中学到的知识应用到另一个不同任务**的过程。预训练和微调就是迁移学习最典型的应用。它的核心思想是，**在相关任务中，一些学到的特征和模式是可以共享的**。

# 迁移学习



# 大模型应用：提示词工程

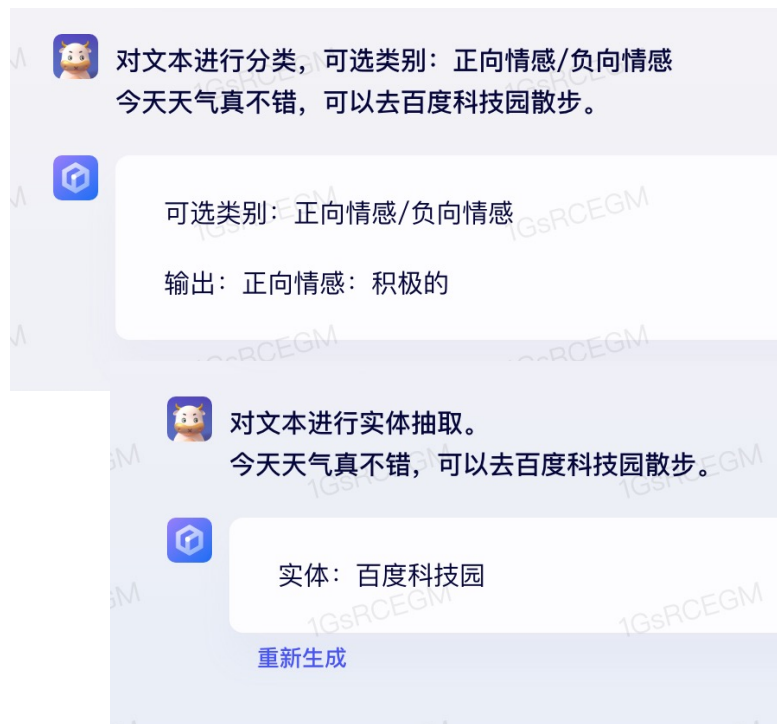


在任务描述中添加额外的背景信息非常有用，背景信息一般放在我们尝试处理的输入文本之后。

# 大模型应用：提示词工程

Prompt（提示）包括传递给语言模型的**指令**和**上下文**，代表用户期望模型实现任务。

- Prompt Engineering（提示工程）是开发和优化Prompt，让大模型输出更加有效的结果。
- Prompt Engineering 并不设计修改和训练模型，只关注模型的输入和输出



# 大模型应用：提示词工程

Prompt（提示）包括传递给语言模型的**指令**和**上下文**，代表用户期望模型实现任务。

- 任务提示：核心任务
- 上下文：任务描述 / 背景
- 输入数据：输入的自定义文本
- 输出格式

- Instructions
- Context
- Input data
- Output indicator

Classify the text into neutral, negative or positive

Text: I think the food was okay.

Sentiment:

# 大模型应用：提示词工程

Prompt（提示）包括传递给语言模型的**指令**和**上下文**，代表用户期望模型实现任务。

- 任务提示：核心任务
- 上下文：任务描述 / 背景
- 输入数据：输入的自定义文本
- 输出格式



对文本进行实体抽取。

案例1：今天很开心，去找一下北大的李华。

输出：北大-地点、李华-人名

输入：今天天气真不错，可以去百度科技园散步。

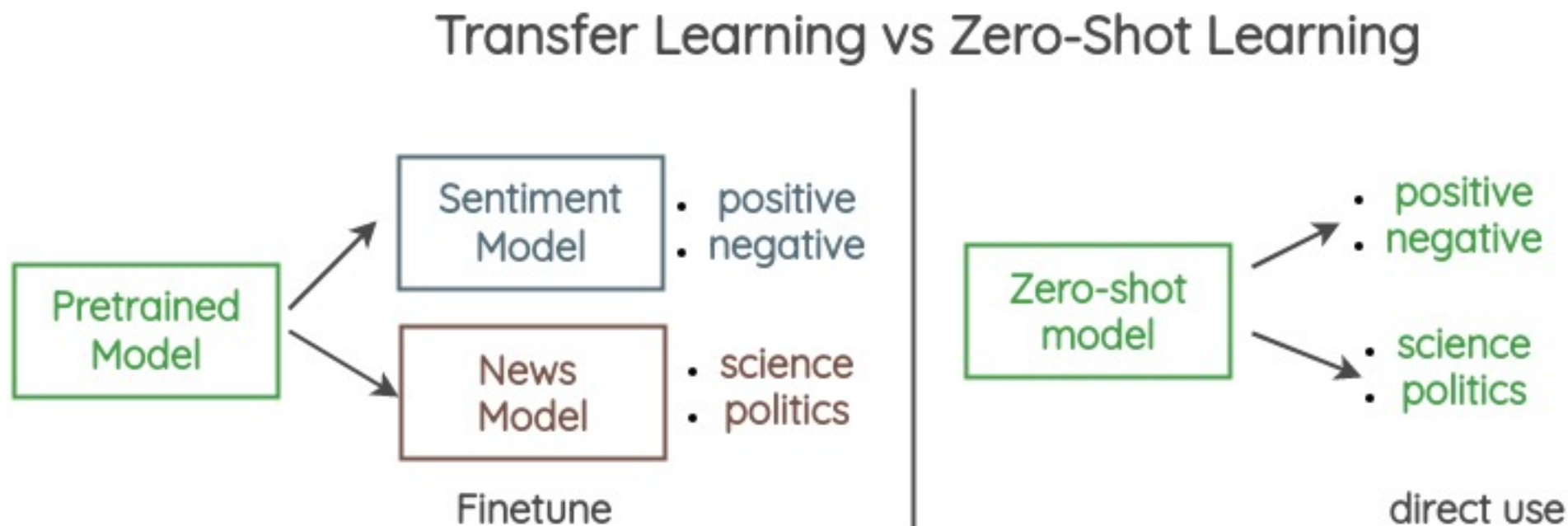


输出：百度科技园-地点

重新生成

# 大模型应用：提示词工程

**上下文学习 (In-context Learning)** 是大语言模型的一个关键能力。它指的是模型能够**仅根据输入中提供的上下文信息**（比如一些示例）来完成任务，而**无需更新模型的参数**。例如，你给模型一个“翻译：你好 -> Hello”的例子，然后给它一个“翻译：再见 -> ”的输入，模型就能理解并完成任务。这种学习方式是GPT-3等模型的强大之处。



# 大模型应用：提示词工程

- 明确表达任务的目的，建议直接提到具体任务的名字。如情感分类、文本翻译等。
- 给出正确的少量参考案例，包括输入和输出。增加模型对少样本 或 临界情况的建模能力。

如果Prompt效果不满意，如何调整？

- ✓ 通过对话，对模型之前输出的结果进行反馈；
- ✓ 调整Prompt，让任务更加清晰；
- ✓ 增加参考案例的输入和输出；