


人人懂AI之 从机器学习到大模型

零基础学AI

知其然并知其所以然

作者：刘军民



期望本书能成为AI技术爱好者的启蒙书籍。涉及机器学习、深度学习、神经网络等基本原理，以及大语言模型“知识茧房”问题探讨



aliyun.com

海量电子手册免费下载



阿里云瑶池数据库公众号

获取阿里云数据库最新信息

导论

AI 以极快的速度融入到每个人的日常，影响工作、学习、生活等方方面面。云计算、AI 等浪潮下，IT 从业者面临新挑战，也迎来新机遇。未来，AI 预计会成为像计算机语言一样的基础 IT 技能。

本书面向广大 IT 从业者及学生，作者将尽可能通俗易懂的把机器学习、深度学习、神经网络等基本原理讲解清楚，并分享大语言模型、知识库等当下很火爆的 AIGC 应用，探讨大语言模型“知识茧房”问题及解法。期望本书能成为 AI 技术爱好者的启蒙书籍、学习手册。希望人人都能了解 AI，知其然并知其一点所以然，看完后能有感而发：“原来 AI 是这么回事”，且能自己动手实践，构建自己的 AI 应用。

目录页

- 序 6
- AI 应用已无处不在 7
- AI 到底是什么 9
 - 尝试给 AI 一个定义 9
 - 机器学习训练及推理过程 12
 - AI 三要素：数据、算法、算力 13
 - 来一个 AI 的 “Hello world” 样例 14
 - 数据集和模型的数学表达与抽象 16
 - 损失函数(Cost Function) 17
 - 梯度下降 20
 - 多变量扩展 23
- 小结 26
- 讲解视频 28
- 深度学习及神经网络 29
 - 神经网络的由来 31
 - 回看线性回归 33
 - 激活函数 34
 - Softmax：多分类 38
 - 卷积：空间特征提取 39

多输入通道及多输出通道	43
特殊的卷积：1*1 卷积	45
卷积的难以解释性	45
池化/汇聚(Pooling)：Max Pooling、Average Pooling	46
全连接（Fully Connected）	48
深入理解经典 CNN	48
像搭积木一样构建神经网络	51
动手实践：搭建一个自己的神经网络-手写数字识别 CNN	53
更多经典 CNN	55
动手实践：在云上搭建深度学习 notebook 开发环境	57
讲解视频	60
AIGC 及大模型	61
硅基智能 vs 碳基智能：AIGC 带给我们的惊喜	61
大语言模型（LLM）	63
语言的数据特征	63
文本转向量（数字化）	64
应用开发新范式：对话即编程	67
LLM 在实际应用中的难点问题	69
LLM “知识茧房” 破解之法：RAG	71
知识库+LLM 智能问答系统的构建流程	73
Langchain 简介	76
系列动手实践	79

在云上从 0 开始搭建 Langchain-ChatGLM LLM 环境	79
从 0 开始搭建 LLM+知识库智能问答钉钉机器人	84
讲解视频	90
如何把 AI 技术融入到产品	91
产业界在干什么	91
学术界在干什么	91
AI 落地三问	94
附录	96
笔者简介	96
参考资料	96

序

AI 以极快的速度融入到每个人的日常，影响工作、学习、生活等方方面面。云计算、AI 等浪潮下，IT 从业者面临新挑战，也迎来新机遇。**未来，AI 预计会成为像计算机语言一样的基础 IT 技能。**笔者作为云计算行业从业者，对 AI 一直保持着强烈好奇心和关注。

本书面向广大 IT 从业者及学生，作者将尽可能通俗易懂的把机器学习、深度学习、神经网络等基本原理讲解清楚，并分享大语言模型、知识库等当下很火爆的 AIGC 应用，探讨大语言模型“知识茧房”问题及解法。期望本书能成为 AI 技术爱好者的启蒙书籍、学习手册。希望人人都能了解 AI，知其然并知其一点所以然，看完后能有感而发：“原来 AI 是这么回事”，且能自己动手实践，构建自己的 AI 应用。

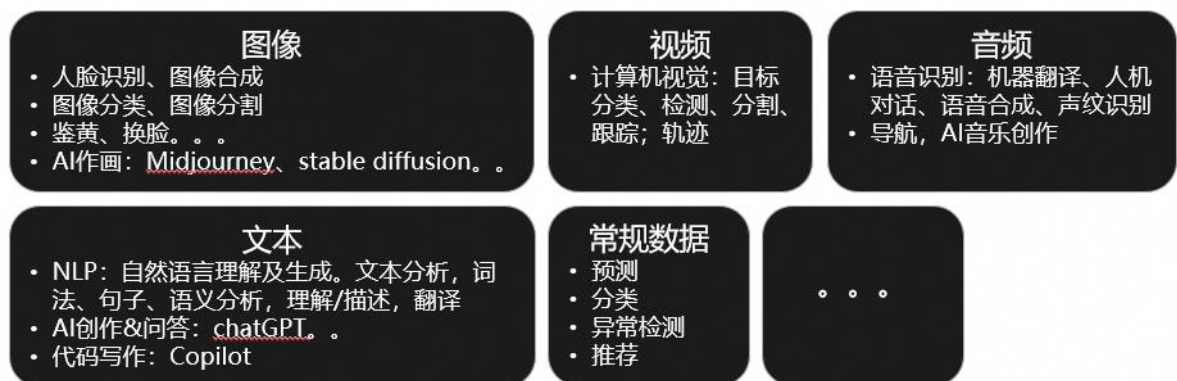
借此机会诚挚致谢吴恩达、周志华、李沐等学者以及斯坦福大学等机构推出的优秀 AI 课程，从中受益匪浅。本文也将引用其个别示例和图表，用于加深大家对 AI 原理的理解。

AI 应用已无处不在

AI 似乎已无处不在，早已深入到各行业：金融、制造、消费、医疗、教育、交通等等。随着 chatGPT 引爆的新一轮 AI 浪潮，使 AIGC 也进一步融入到每个人的工作、学习、生活等方方面面。随处可见的例子：图像/人脸识别、语音识别、自动驾驶、游戏对战、商品推荐、内容推荐、信用评估、风控、垃圾邮件识别等等。



数据类型角度：



比如图像领域，我们常见的人脸识别，车牌识别等。去上班的时候，都不需要带工卡，门禁系统就能自动识别员工，停车场能自动识别车牌，消费购物的时候可以直接刷脸等。视频领域，道路上、园区里随处可见的视频监控，包括现在的依然很火的自动驾驶，也是视频图像相关的 AI 应用。音频领域，语音识别，人机对话，智能音箱等。随着大模型的发展，文本领域的 AI 应用越来越火，包括现在的 ChatGPT，

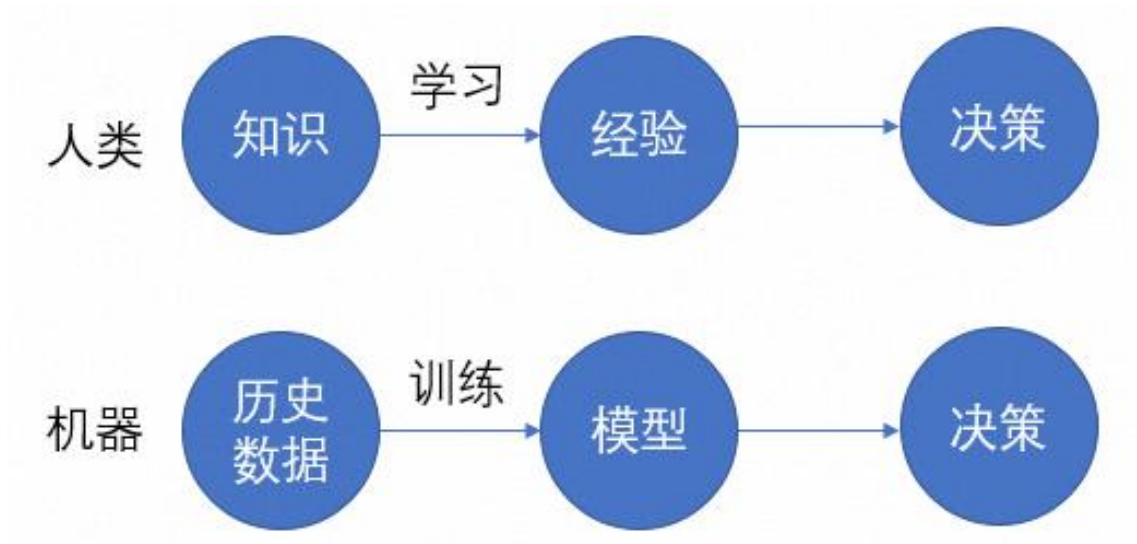
也是 NLP 应用。再比如，我们刷短视频或者网上购物，系统总是会推荐你很感兴趣的东西。以及其他方方面面的 AI 应用，和人们的工作、生活已密不可分，随处可见。

AI 到底是什么

尝试给 AI 一个定义

AI 无疑是当下最大热点话题之一，似乎人人都在谈论 AI，那 AI 到底是什么呢？耳熟能详的机器学习、神经网络又是什么呢？原理是什么呢？不知大家是否思考过这些问题。如果不探究 AI 的原理，就很容易陷入毫无根据的浮想联翩。比如，AI 是不是能炒股实现一夜暴富，是不是能自我进化成新物种，颠覆人类，控制人类等等。我们无需花太多时间放在这些问题的争论上，希望看完本书，可以带给大家新的感悟，形成自己对 AI 的认知。

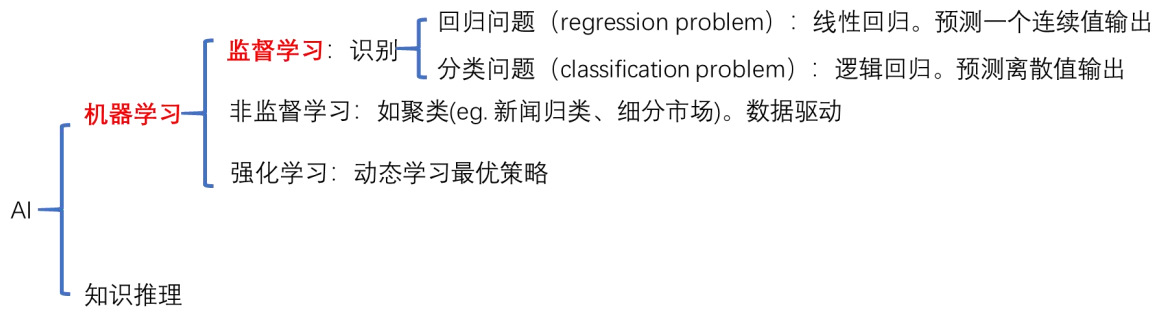
类比到人类自身认知世界的方式：通过对已有的知识、信息等不断的学习实践，积累成自己处理问题的知识和经验，再使用学习到的知识和经验针对新问题做决策判断。绝大部分 AI 技术也有些类似：从已有的数据样本（既有知识、信息）中使用数学、统计学等理论，通过不断的算法训练（后文会详细介绍训练过程和原理），得出 AI 模型（类比为处理问题的知识和经验，也可以直观的理解为一个函数），再使用该模型针对新的输入做推理决策。比如自然语言处理（NLP），数据样本可能是一些相关的文本语料、文章。图像领域，数据样本是大量的相关图片，模型训练好后，再给一个新的图片作为输入，就可以输出一个答案，或者是给出一个判断，比如说这个图片是只猫还是一只狗。



AI 的定义，在不同的资料上或许有不同的定义，我们也可以尝试给 AI 做一个自己的定义：**AI 是以数学、统计学为基础理论的工程实践**。从数据中通过暴力计算挖掘出规律（算法/模型训练），再使用规律（模型）预测结果（推理）。

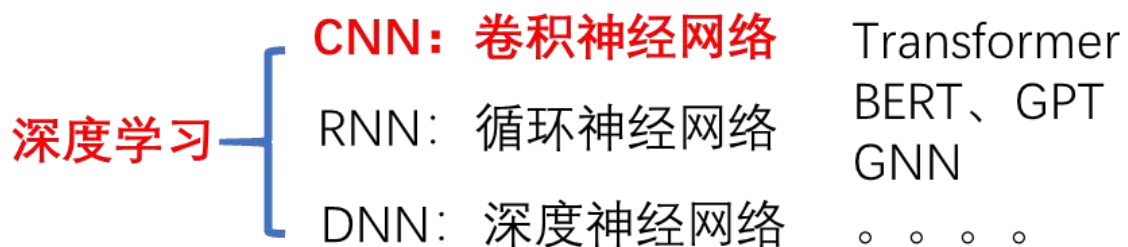
AI 的训练过程，可以简单理解为从一堆该场景关联的数据中，通过暴力计算，寻找一个合适的数学函数，该函数可以表征或拟合已知数据样本的规律。训练好后，再有新的输入，就可以用这个函数做一个预测（输出）。此外，AI 是以数学、统计学为基础理论的工程实践。回顾 AI 发展史，每次新的 AI 浪潮来袭，不仅仅是算法、理论上的进步，更是工程实践的进展，目前依然大火的 ChatGPT 更是工程实践上的大获成功。

AI 的分类：大致可分成机器学习和知识推理，目前大家所说的 AI，主要是指机器学习。机器学习又可以粗略的分成三类：监督学习，非监督学习，强化学习。其中监督学习发展最为迅速，应用最为广泛。



监督学习和非监督学习的主要区别：数据样本是有标注的，还是没有标注。举个例子，图片分类场景，给一个样本图片，并且标明这个图片上是一只猫，就称这个样本是有标注的数据。用这些标好的数据去训练 AI 模型，这就是监督学习，也是应用更广泛的一类，本文也将重点讲监督学习。而非监督学习，用于训练的数据样本无需标注。

监督学习又可分为两大类：第一类是回归问题（称为线性回归），第二类是分类问题（称为逻辑回归）。回归问题可以简单理解为，对应的 AI 模型可以预测连续的值，或直观理解为该函数的输出是由连续值构成的。举个例子，训练一个模型去预测房价，房价可以理解是一个有连续输出值的函数，而分类问题的输出是离散值，全部输出值是一些离散值组成的集合。比如，预测一张图片中的动物，是一只猫还是狗，输出就是动物所属的类别，而这些类别就是一个个离散值。



此外，还有一块重要内容：深度学习，属于机器学习里的一个分支，早期也称为模仿大脑的机器学习。很多大规模应用的近现代 AI 模型，基本上都和深度学习相关，

包括现在大火的 Transformer、BERT、GPT 等。深度学习又分成多个类别，比如常见的 CNN（卷积神经网络），在机器视觉领域应用广泛；RNN（循环神经网络），早期在 NLP 领域用的非常多；还有所谓的 DNN 深度神经网络等等。

机器学习训练及推理过程

接下来，可以从机器学习中的监督学习出发，继续探究 AI 的原理，看如何从数据样本中，通过模型训练找到一个合适的函数。

训练

Step1: 搜集数据、预处理、提取特征
Step2: 函数集合（依经验选择/试模型）、或开发定义新模型
Step3: 定义模型中函数的好坏
Step4: 找出最佳函数（及其最优参数）

推理：利用函数和新的输入得到预测结果

当需要用 AI 解决的问题（或场景）确定后，训练过程可粗略分为四大步：

Step1：搜集和该问题相关的数据，进行预处理，构建特征值，整理成数据样本，这是 AI 数据工程师的主要任务之一。比如，最终要训练出一个函数，那函数就有很多的输入变量 $x_1, x_2 \dots x_n$ ，每一个变量也称为一个特征。

Step2：函数集合（根据经验选择/试模型）、或开发定义新模型。处理一个问题，可能有多个函数（AI 中通常称为模型）适用，我们可以把这些函数想象成一个集合。最初，我们也不知道哪一个函数（或哪些函数的组合），最适合解决此问题。算法工程师根据自身经验选择可能适合的模型，或者采用最粗暴的方式，将各可能适合的模型逐个都训练一遍（这也是暴力计算的体现之一），逐个试效果。模型选定后，就可以用数据样本开始训练，得出该模型对应变量的参数。参数确定后，整个函数

也就确定了。

Step3: 定义模型的好坏。有一套方法论来定义模型效果的好坏，后文将做详细讲解。

Step4: 找出最佳函数（及其最优参数）。通过前三步，可得出最佳模型，或者说最佳函数。就可以拿这个函数去做推理：给定一个新输入，可以通过这个函数计算出结果。模型的推理效果也常称为泛化能力：也即从有限的训练数据中学习得出的模型，如何确保在未知的新输入数据上的预测能力、准确性。

AI 三要素：数据、算法、算力

数据：AI 是一门数据科学，或者说是一门从数据中发掘规律的科学。数据是 AI 模型的知识源头，如果数据本身就缺乏规律，没有规律可发掘；或者该场景存在数据规律，但收集的数据样本质量很差，那不管使用何种算法去训练，都很难取得效果。没有高质量数据，AI 将是无源之水，寸步难行。数据量、数据质量从根本上决定模型的推理质量。

算法：有了高质量数据之后，有各种各样的算法都可以训练出一个模型。算法的好坏也会直接决定模型推理的效果（泛化能力）。

算力：算力对于 AI 也是至关重要的，回顾前面的定义，我们特意强调了 AI 的训练过程就是通过不断的暴力计算，去找寻最优函数及其参数。尤其是现在已经进入深度神经网络、大模型时代，如果没有充足的算力，是难以训练出模型的。chatGPT 等 LLM（大语言模型）类人智能的诸多惊艳表现，放在十年前是难以想象的，因为缺

乏如此强悍的算力来完成模型训练。

此外，个人认为 AI 的核心，除了上述 3 要素，还有个点容易被人轻视，但至关重要，就是**应用场景**。AI 技术并非万能，并非每一类问题都适合使用 AI 来解决，找到合适的应用场景十分关键，如果一开始场景就选错了，后面做再多工作也是徒劳。此外，应用生态的构建、应用效果的持续优化，往往是比模型训练更具挑战的工作。

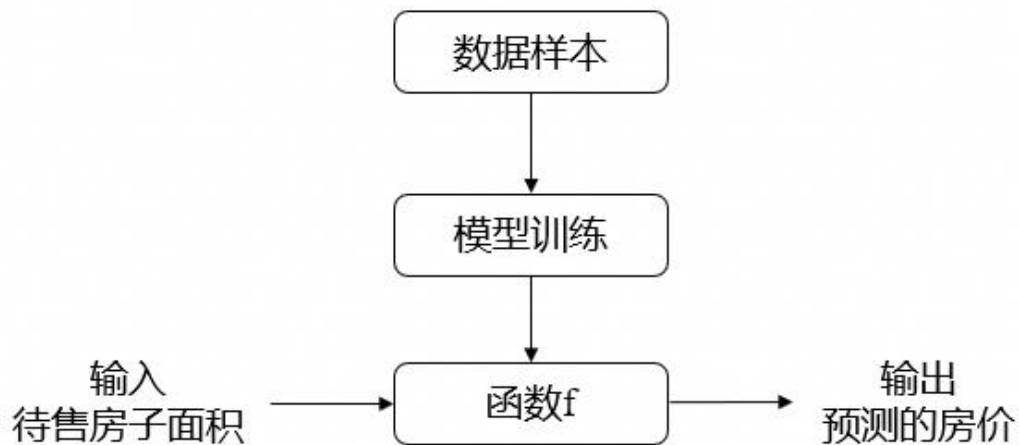
有资料显示，自 1970 年到 2020 年，数据样本有超过 1000 倍的增长，算力更有上万倍的增长。尤其是进入 21 世纪，CPU、GPU、云计算等算力的增长尤为迅猛，加上 2006 年后逐步完善的深度学习等算法上的创新突破。进一步引爆了 AI 这一上世纪 50 年代就已诞生的古老学科，AIGC（ChatGPT 等 LLM、文生图、文生代码等）、AGI 等有诸多类人智能的 AI 应用不断涌现。

来一个 AI 的 “Hello world” 样例

如果问学习 AI 的捷径是什么，建议是从 AI 的实现原理入手，分析原理，并动手实战，就如同习武先打通任督二脉（**原理+实践**）。接下来，就像学一门编程语言一样，先从一个监督学习里的回归问题的 “Hello world” 示例入手：已有一批已标注的数据样本，看如何通过 AI 训练，找到一个函数来表征这个数据样本集。

在此，引入吴恩达老师机器学习教学视频中的一个示例：房价预测。先将这个场景，抽象成一个**回归问题（线性回归）**：给定一个数据集，这个数据集由**“正确答案”组成**。在房价的例子中，数据样本就是一系列已成交房价相关的数据，数据集中每个样本包含了该套房子相关的信息，如房子的面积，房型，地段，楼层，朝向等等，以及该房子的正确价格（**实际售价**）。基于该数据集，训练出一个模型（函数），从而推算出更多的正确答案，比如预测一个待售房子的价格。

既然是“Hello world”示例，可以先做一个极简的假设，先看只有一个变量（也称为特征）的情况：假设房子的价格只跟面积相关。等全流程理顺后，后面再扩展其他变量就简单了，因为原理是一样的。

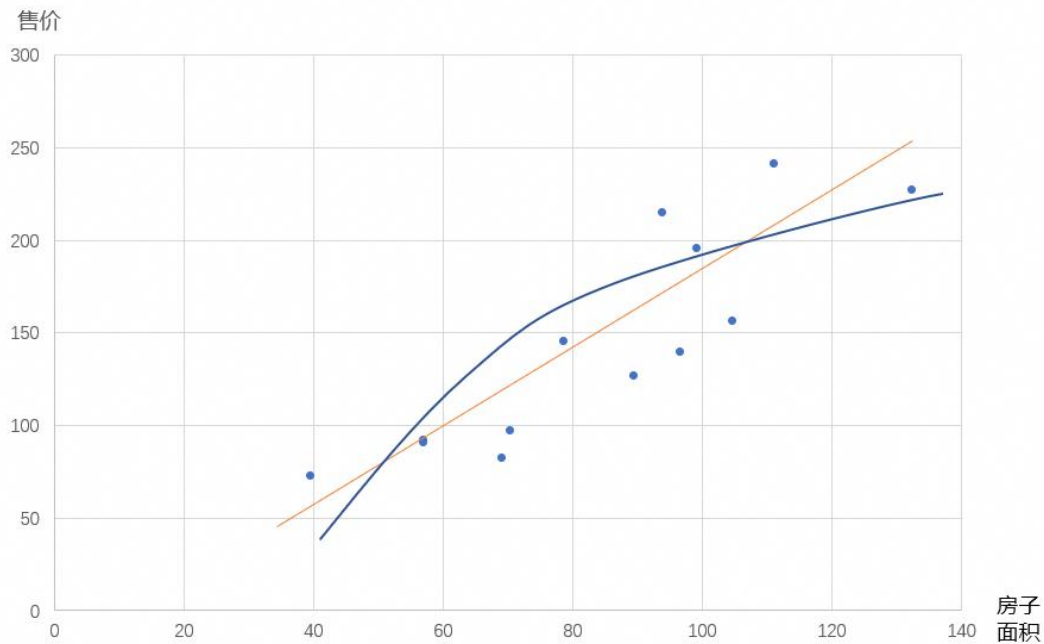


数据样本：输入是每套房子的面积，输出是该房子的售价（kaggle 等网站上有大量可用于练手的数据集）。接下来，就可以构造一系列函数来拟合这个数据集，如：

$y=ax+b$ //一次方函数

$y=ax^2+bx+c$ //二次方函数

。。。。。。 //其他更为复杂的多次方函数。这些可选择的函数，就构成了一个集合



比如 $y=ax+b$ 一次方函数，在坐标轴上画下来，就是一条直线。 $y=ax^2+bx+c$ 二次方及其他多次方函数，可能就是一条曲线。接下来的任务就变成了，如何从这些函数集中找到最匹配的函数，并确定其对应的参数。

数据集和模型的数学表达与抽象

x: LotArea(size in square feet)	y: SalePrice(\$)
8450	208500
9600	181500
11250	223500
9550	140000
14260	250000
...	...

为了方便数学计算和表述，可把数据集、模型做个数学表达：

m 代表训练集（从全量的数据样本中随机挑选出一定比例的数据，用于训练，称这批数据为训练集）中数据样本的总数量。

x 代表特征/输入变量。

y 代表目标变量/输出变量。

(x, y) 代表训练集中的实例。

$(x^{(i)}, y^{(i)})$ 代表第 i 个数据样本。

h 代表学习算法函数，也称为假设(hypothesis)，也即： $y = h(x)$ 。各函数模型（表达式），可相应的写成如下形式（其中 θ_0 是一个常数，也称为偏置）：

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

。 。 。 。 。 。

假如，我们根据自身经验从函数集中，先挑选出一个可能合适的函数，如，盲选一个最简单的，选择 $h_{\theta}(x) = \theta_0 + \theta_1 x$ ，再通过 AI 训练把该函数的各个参数 θ ，都确定下来，那也就表示这个函数就训练好了。就可以用它预测一个未销售的房子的价格，因为输入该房子的面积 x ，就可以得出房价 y 了。下一步，问题就变成，怎样把函数的各个参数 θ 都确定下来，使函数更精准的匹配已有训练集呢？

损失函数(Cost Function)

先假设模型已经训练完了，也即模型函数及其各参数值 θ 都已确定了。怎么去判断这个函数和训练集的匹配情况，也即该函数的好坏呢？最简单的方法：将训练集中每个数据样本的输入 $x^{(i)}$ 都代入到预测函数 $h_{\theta}(x)$ ，算出预测值，将预测值和该样本的实

际值 $y^{(i)}$ （真实售价）相减，并取绝对值，最后都加起来，就可以得到建模的误差了。
也即：

$$\text{建模误差} = \sum_{i=1}^m |h_{\theta}(x^{(i)}) - y^{(i)}|$$

建模误差(modeling error)：表示模型所预测的值与训练集中实际值之间的差距。如果模型很好，理论上预测值和实际值就是一样的，或者说至少是相近的。而建模误差越大，则说明预测值偏离实际值越多。

实际应用中，有更丰富的函数来表征模型函数和训练集的差异，称之为**损失函数**（有些书籍翻译为代价函数）。比如，建模误差中的绝对值在运算中通常是不好处理的，也难以求导数（后文可以看出，求导在 AI 训练中非常有用）。有一个最简单方式：使用平方，取差值的平方，和取绝对值的效果其实是完全一样的，都可以有效反馈预测值和实际值间的差异。于是，损失函数变成：

$$\text{损失函数} \rightarrow \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

取差值的平方还有个好处：不需要再处理那些差值的正值和负值了，因为平方之后，都是正数或 0。接下来，继续优化损失函数，有个影响因素需要考虑进来：样本数量 m 。比如训练集有 1 万条数据，则 m 就等于 1 万。假设有 10 万个数据样本，即便是同一个预测函数 $h_{\theta}(x)$ ，最后算出来的总差值，依然可能会放大 10 倍，这不是我们希望看到的。解决这个问题也很简单，直接除以 m 即可。这样的话，不管是 1 万个样本还是 10 万个样本，如果数据样本质量足够好，那最后得出来总差值都是差不多的。于是，损失函数变成：

$$\text{损失函数} \rightarrow \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

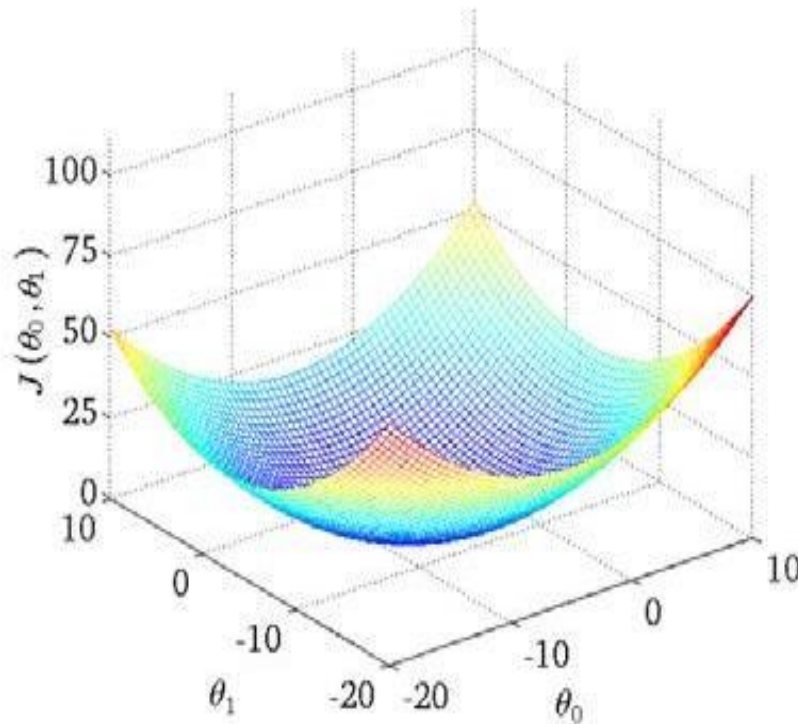
进一步，可以再乘一个 2 分之 1（方便后续训练时对损失函数做求导运算：平方函数求导后，平方数 2 会移下来，刚好和 1/2 可以相抵销）。只是乘以一个常数 1/2，对我们判断模型的好坏是没有任何影响的。另一个更常用的损失函数就定义出来了，用 $J(\theta)$ 表示：

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$J(\theta)$ 与建模误差的变化趋势是完全一致的，如果可以找到一组参数 θ 使 $J(\theta)$ 最小，则此时的建模误差也是最小的。如果损失函数值变得最小的话，那就初步认为这个模型是一个好模型。问题进一步变成：**找到最优的参数 θ ，使 $J(\theta)$ 最小**。即，使损失函数最小。例如，一次方函数，有两个 θ 参数：

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

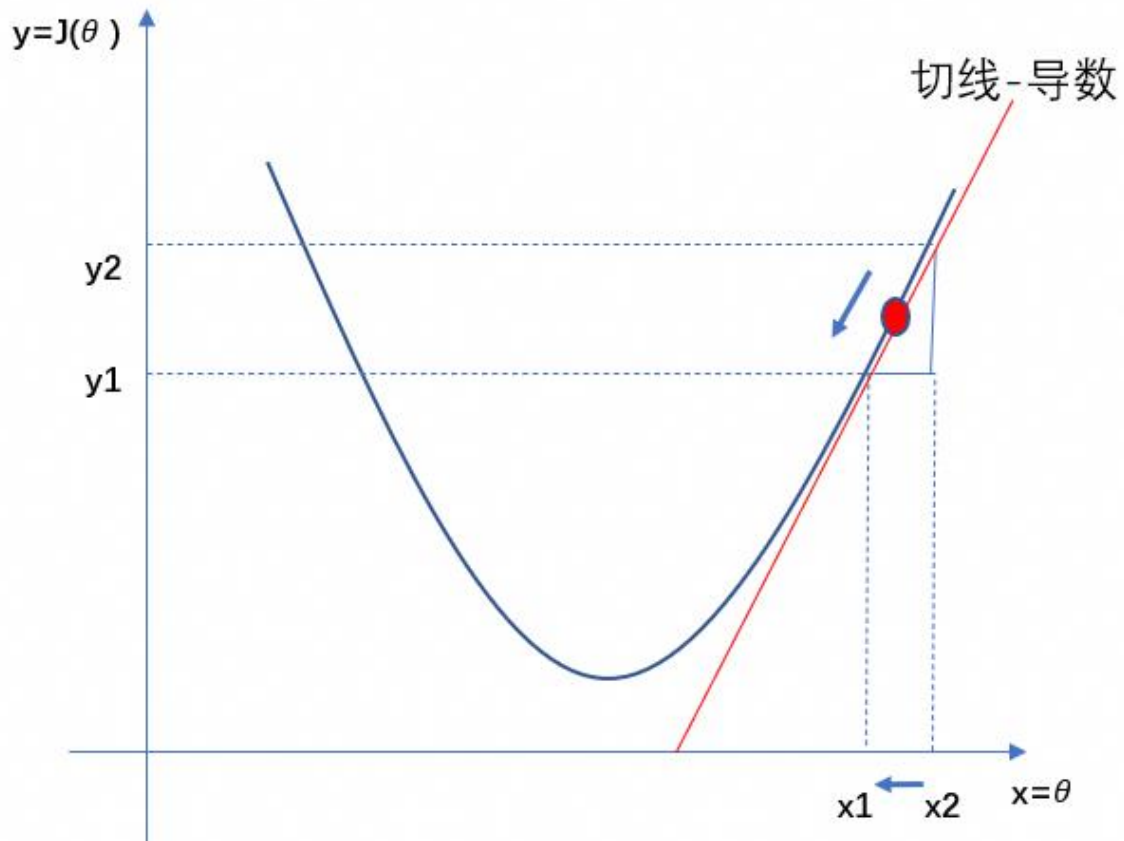
将 θ_0 和 θ_1 看做变量，绘制函数到坐标系中，三个坐标分别为 θ_0 、 θ_1 和 $J(\theta_0, \theta_1)$ ，可以看出在三维空间中，存在一组 (θ_0, θ_1) 使 $J(\theta_0, \theta_1)$ 最小。在实际应用中，可能很难找到一个全局最小值，大部分情况下是局部最小值。**问题变成，如何找到 $J(\theta_0, \theta_1)$ 的这个最低点（最小值）对应的 (θ_0, θ_1) 。**



梯度下降

梯度下降是用来求解一个函数最小值及其对应输入值的一个算法。算法的解题思路：**随机选择一个参数的组合**($\theta_0, \theta_1, \dots, \theta_n$) 作为起始值，计算损失函数，然后寻找下一个能让损失函数值**下降的参数组合**。一直重复该过程，直到找到局部最小值(local minimum)。由于没有尝试所有参数组合，所以并不能确定所得到的局部最小值是否是全局最小值(global minimum)，**选择不同的起始参数组合，有可能会找到不同的局部最小值。**

对于梯度下降，有个很形象的解释，结合前面的 $J(\theta_0, \theta_1)$ 的函数图片，想象成下山：随机的把你投到山上面的任何一个点，如何走到最低点。我们可以先只看其中一个变量（固定其余变量），比如先只看 θ_0 （或者先只看 θ_1 ）。找到此时所在的点，沿着 θ_0 坐标轴方向纵向切一刀。切出来后，是一条二维曲线，问题就变成了如何从当前所在的点，走到最低点。



导数/偏导数：

$$y' = (y_2 - y_1) / (x_2 - x_1) = \Delta y / \Delta x \quad (\Delta x \rightarrow 0)$$

$$y_1 = y_2 - y'(x_2 - x_1) = y_2 - y' \Delta x \quad (\text{导数切线指向最陡峭的方向})$$

可以使用高等数学中求导数/偏导数的方法，找到曲线的切线方向，如果始终沿着该点的切线方向一步一步走，就能走到最低点，而且理论上这个方向是可以最快的走到最低点。每走一步，下一步就沿着新点的切线方向继续往下走，直到最低点，用数学公式表示：

Gradient descent algorithm

```
repeat until convergence {
   $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ 
  (for  $j = 1$  and  $j = 0$ )
}
```

Linear Regression Model

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$j = 0 \text{ 时: } \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$j = 1 \text{ 时: } \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)})$$

则算法改写成:

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)})$$

}

直观理解梯度下降原理：随机初始化一个起始值， θ 每次更新都**减去偏导和放大系数 α （控制速度/步幅）的乘积**，便能走到下一个步进点，得到更小的 $J(\theta)$ 值（离最低点**又近了一步**），每次 θ 值更新后，就可以计算出新的 $J(\theta)$ 和 $h_{\theta}(x)$ ，代入到求偏导的公式中，又可以走到下一步。如此，反复循环的重复这个动作（这也是 AI 训练，暴力计算的体现之一），当 $J(\theta)$ 值不再明显变化（也即无更小值了。此时偏导接近 0，切

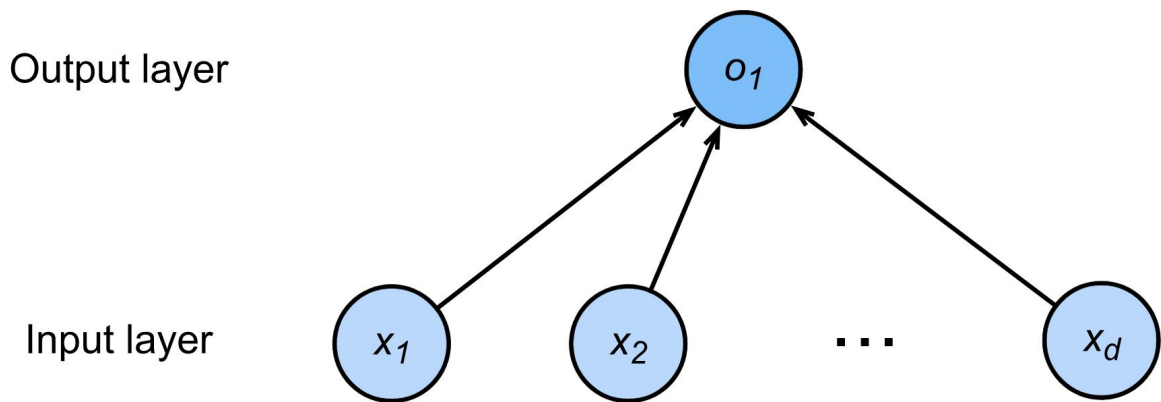
线方向与 x 轴平行或接近平行, **θ 值也不再变化**), 则意味着找到了 $J(\theta)$ 的最小值及对应的 θ 参数, 则训练结束 (或达到设置的最大训练次数时, 也会结束训练)。

进一步思考: α 在机器学习中是一个超参数, 称为学习率。从前面的理解不难看出: α 值的大小会控制每一步走多长 (步幅), 或者是走多快。 α 如果设置过小, 则每次只能前进一小步, 收敛速度慢, 训练耗时长, 需要大量的计算迭代才能达到最低点; α 如果设置过大, 则每次前进一大步。有可能 (大概率) 在接近最低点时会反复越过最低点, 而导致无法收敛。通常可尝试选不同的学习率: $\alpha = 0.01$ (常用), 0.03 , 0.1 , 0.3 , 1 , 3 , 10 。

至此, 机器学习训练的基本过程和原理也讲解完毕了。我们再回到房价预测这个例子上来, 通过上述训练过程后, 房价预测函数及其参数就确定了。这时候再输入一个新房子的面积 x , 就可以用这个函数计算出对应的售价 y , 从而实现对该房子的房价预测。

多变量扩展

前面的例子, 为了函数画图简便, 我们把输入变量做了极简的假设: 输入变量只有一个房子的面积, 函数也选了个极简的一次方函数。实际上, 房间数量、户型、朝向、地段、年份等等。。。。。有大量会影响房价的变量。数据样本也会是一个包含大量变量的数据集。而且最优的模型函数也通常不是一次方的简单函数, 而是有更为丰富和复杂的函数可选。但处理方法依然是大同小异的。



比如，多变量的扩展，有了前面的理解，其实就很好处理了，只是函数多了一些输入变量及其 θ 参数而已，依然可以采用梯度下降的方式做训练，找到 $J(\theta)$ 的最小值及对应的 θ 参数。区别在于，之前是 (θ_0, θ_1) ，现在是 $(\theta_0, \theta_1, \dots, \theta_n)$ ，每次迭代需要计算的 θ 多一些而已，计算量更大了，就这么个差别。还可以把 $(\theta_0, \theta_1, \dots, \theta_n)$ 和输入变量 $(1, x_1, \dots, x_n)$ 写成向量的方式，如， Θ 和 X 均写成 $(n+1) \times 1$ 的向量：

$$h_{\theta}(x) = \theta^T X = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\text{Repeat } \left\{ \begin{array}{l} \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots, \theta_n) \\ \end{array} \right\}$$

即：

$$\text{Repeat } \left\{ \begin{array}{l} \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \\ \end{array} \right\}$$

求导数后得到：

$$\begin{array}{l} \text{Repeat } \left\{ \begin{array}{l} \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}) \\ \text{(simultaneously update } \theta_j \\ \text{for } j=0, 1, \dots, n \text{)} \\ \end{array} \right\} \end{array}$$

当 $n \geq 1$ 时，

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

。 。 。 。 。 。

小结

假如你想用 AI 解决一个问题，并且这个问题也适合使用 AI 来解决。依然以监督学习为例，第一步需要尽可能收集与该问题相关的数据样本，且理论上数据样本越全面越具有代表性则越好。然后，需要对数据上做一些预处理。比如数据的质量参差不齐，有些数据样本本身就是错误的，或者缺失了一些关键信息，是无效数据，需要剔除。此外，还需要提炼输入特征，可能还涉及一些特征值的数字化处理（训练过程中只能处理数字化后的变量）。比如预测房价的例子，它在哪一个区，是以文字表述的，怎样转化成数字呢？比如我想分析深圳某 5 个区的房价，每个区可以使用一个数字来表示，比如，数字 1 表示福田区，数字 2 表示南山区等等。数字化的方法有很多，这只是举了个最简单的方式。最后，还需要采用随机取样等方式，按一定比例，将数据样本分成训练集、测试集等。

第二步：构造一个函数。机器学习的常用算法很多，如何从诸多算法里挑选出合适的算法，就需要大量的积累和学习。算法选好后，还需要选择一个合适的损失函数。损失函数本身也有很多种，前面例子中，求建模误差的平方，只是其中一个常用的损失函数。需要根据实际处理的具体问题选择适合的损失函数，并且最好先弄清楚损失函数的基本原理。

第三步就可以用训练样本集，结合算法模型和损失函数，使用梯度下降等方法训练模型。一步一步去逼近损失函数最小值及其对应的各参数 θ 。当求解到损失函数最小值时，对应的这些参数 θ ，就认为已找到了最优参数，该函数训练完毕。当然，也有可能运气不太好，随机初始化的起始参数，训练后只找到了局部最优解，需要设置不同的随机初始参数，重复训练多轮，寻找更优解。学习率等超参数设置不合理，

导致训练过程迟迟无法收敛。也有可能选择的函数模型本身不是最优的，导致推理效果不理想等等。这其中有大量的知识需要学习，比如超参数的调测，如何加快训练速度，提升算法效果等。目前业界也有一些使用简便的 **AutoML** 产品来简化训练和调优过程。总之，算法训练过程是个不断的分析、调测、一轮又一轮不断优化的过程，这也是 AI 训练是暴力计算的另一个体现。

总之，算法训练过程是个不断的分析、调测、一轮又一轮不断优化的过程，这也是 AI 训练是暴力计算的另一个体现。

第四步，可以用训练好的算法模型，去做推理，使用测试集做效果评测，这其中需处理过拟合、欠拟合等问题，可能也涉及算法模型的重新选择、反复训练，直到得到各项指标达标的算法，最后上线应用。

有了这些理解，逻辑回归、深度学习、神经网络等训练的大体步骤和基本原理都是相通的，只是有不同的特征向量、参数、函数、损失函数、训练方法等。

当然，AI 本就不是万能的，切忌在各种盲目的鼓吹声中失去理性判断。通过前文的学习，可以有个清晰的认知：AI 的本质是从现有的数据样本中发掘数据的规律，然后使用该规律针对新输入做输出。也有可能问题本身或者搜集的数据本身就不适合使用 AI 来解决，那即便是做再多的训练也是徒劳。遇到一个问题时，可以先从数据角度去分析思考，数据本身是否可能存在规律，问题本身是否能从获取的数据样本中找到关键规律，至少可以先有个理性的思考判断。

此外，找准合适的应用场景极其重要。回到文中房价预测的例子，实际应用中，一套房子的价格，即便是同属于一个社区的，也会因为楼盘的开发商、位置、年份、

房间的楼层、朝向等各种因素的影响。同一个楼盘的房间数量也是极其有限的，也即针对一套待售房子，有参考价值的数据样本数量是很有限的。通过前文的原理性讲解，可以看出，在数据样本数量稀少的情况下，是难以训练出高质量模型的。此外，哪怕是同一套房子，在不同的年度，可能因政府政策、经济发展等因素，而导致房价的巨大差异。每年的政府政策，大环境的经济发展往往起更大的作用，是影响房价的关键因素，而这些因素通常是难以体现在历史交易数据（样本）中的。所以，房价预测这个应用场景，从 AI 的原理上分析，本身是不太合适的。大费周章的训练一个 AI 模型，还不如直接参考近几个月该楼盘具有相似属性的房子的成交价，做个估算，来的更为直接和准确。虽然可能偏离实际应用，但作为一个学习示例，讲解原理是可以的。

有了这些基本原理的掌握，再遇到 AI 是不是能炒股实现一夜暴富，是不是能自我进化成新物种，颠覆人类，控制人类等天马行空的问题，相信读者可以从 AI 的原理角度展开，做思考分析，形成自己新的认知和答案了。

讲解视频

为进一步方便学习，录制了讲解视频，详见：

https://www.bilibili.com/video/BV1414y197xd/?vd_source=0c5a39664e27af522bfcd1e74b9780a8

深度学习及神经网络

接下来，我们可以学习一个更有意思的：神经网络和深度学习。假如把人想象成一个处理数据的智能机器，那么眼睛、耳朵、鼻子、嘴巴、皮肤等各个视觉、听觉、嗅觉、味觉、触觉等相关的器官，均是输入。大脑处理加工这些信息，进行知识积累和训练。其中有两类数据最为常见：**所谓眼观六路、耳听八方。目之所见是图像，耳之所听是声音、语言。**



AI 有两大应用广泛的领域分别与之对应：Computer Version（机器视觉/计算机视觉），NLP（自然语言处理）。图像和语音数据都是有显著空间和时间特征的：

- **观-图像**：空间特征（**平移不变性、局部相关性**） → CV
- **听-音频、语言、文本**：时序、上下文关联(语境、权重) → NLP

接下来，我们重点以图像处理来讲解神经网络。如何理解图像的平移不变性和局部

相关性空间特征呢？

平移不变性：比如给某人拍照，针对同一个人。把镜头挪一下，让这个人在照片中所处的位置做个变化（见下图，没找到合适的示意图：针对相同的背景图片，可以任意移动目标人像）。原先她可能在中央位置，现在偏左或者偏右了。如果使用 AI 模型去判断，比如人脸识别，理论上，都应该判断出是同一个人，这叫平移不变性。



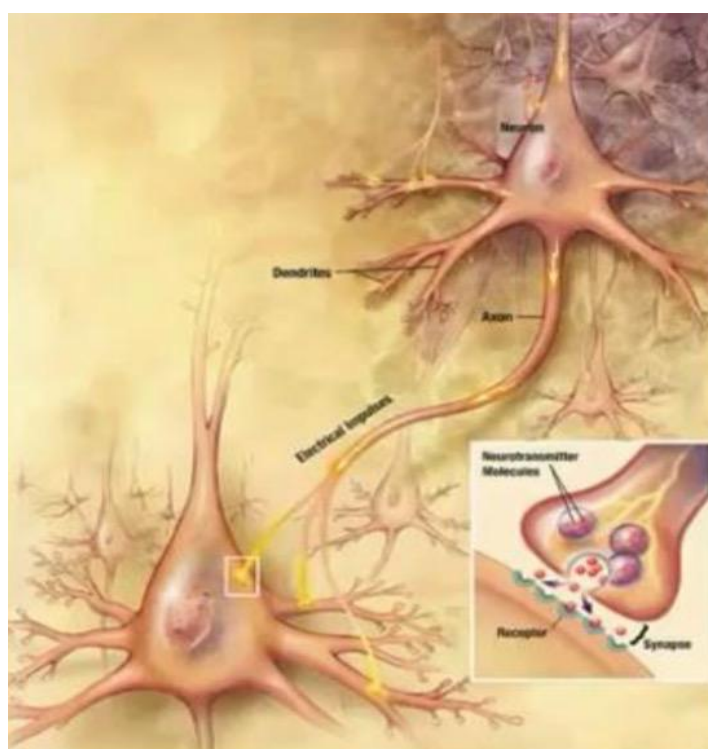
局部相关性：依然以人像为例，有眼睛、鼻子、嘴巴这些基本构造，而且彼此是局部相关的，比如眼睛的旁边有另外一只眼睛，附近是鼻子，往下是嘴巴。或者取图片中一小块图像，比如眼睛，本身也是局部相关的，睁开的眼睛是一个类似椭圆的形状，中间是圆形的眼球，也即该图像块中的像素点之间是彼此有关联的，或者有规律的。而且这个规律具有局部性。

前面讲解的线性回归、逻辑回归等传统机器学习，很难有效处理空间和时间特征。以图片为例，图片是由像素点构成，比如 RGB 格式的图片，每个像素点由 R（红）、G（绿）、B（蓝）三个颜色通道的值（0~255）来表示。可以简单理解为每张黑白图片就是一个二维矩阵（长、宽），彩色图片还涉及多个通道（多个二维矩阵）。假设，有一张 100*100 像素的黑白图片，相比现在动辄百万、千万像素的图片而言，

已经足够小了（100 乘 100 才一万像素）。使用之前的监督学习来处理，每一个像素点表示一个特征值，则会有 100×100 ，也即一万个特征。像素和像素之间还有些联系，两两特征间再做个组合，则会有 $10000 \times 2/2$ ，约 5 千万个特征。不难发现，传统机器学习已难以处理这么多的特征，训练不动，而且也很难提炼平移不变性、局部相关性规律，此时需要**神经网络**。

神经网络的由来

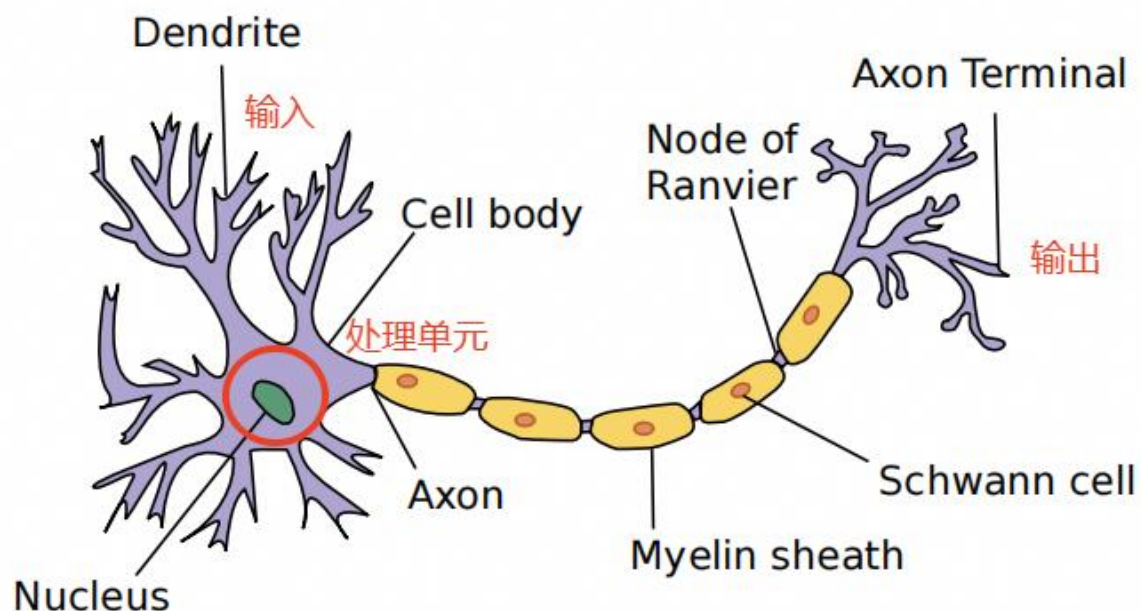
神经网络（Neural Network，缩写 NN）的命名，是类比于医学上的神经网络构成而来（不过实际上，当前大多数深度学习的研究和突破，其实**并没有直接从神经科学中获得太多灵感**）：



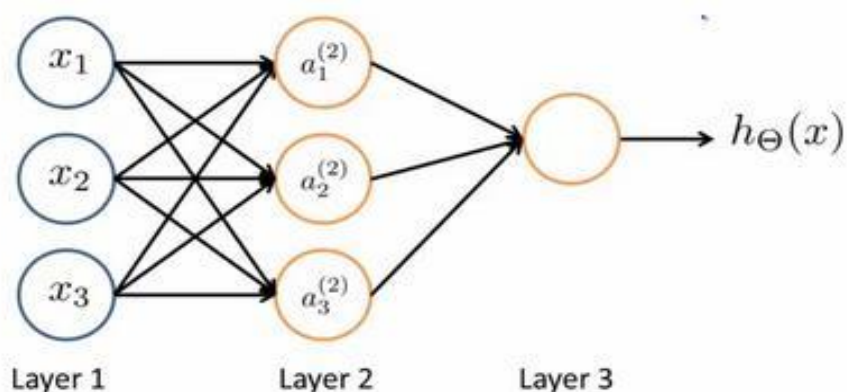
- ✓ 神经网络是大量神经元相互链接并通过电脉冲来交流的一个网络。
- ✓ 每一个神经元都可以被认为是一个**处理单元 / 神经核 (processing unit/Nucleus)**，它含有许多**输入/树突 (input/Dendrite)**，并且有一个**输出/**

轴突(output/Axon)。

- ✓ 神经元间传递消息：通过它的轴突，发送一段微弱电流给其他神经元 。



神经元：可以接收多个输入，汇集到处理单元运算，给出一个输出。这个输出，又可以和下一个神经元相连，作为下一个神经元的输入，由此可以形成一个庞大的神经网络。抽象成数学表达：

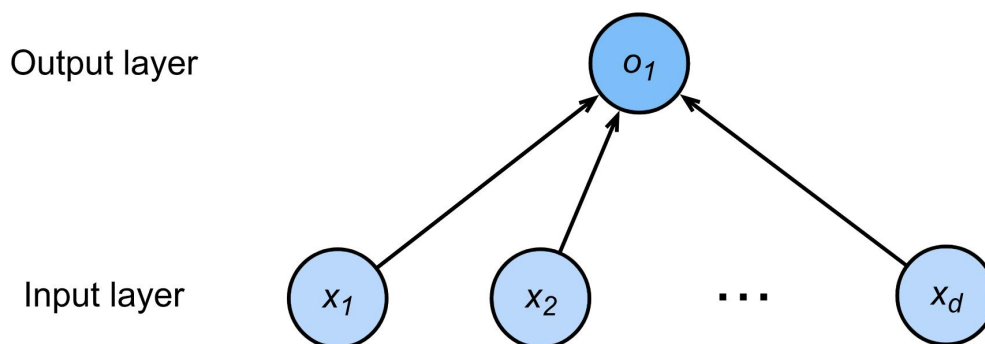


NN 构建： x_1, x_2, x_3 是输入单元(input units)，作为原始数据输入。 a_1, a_2, a_3 是中间单元（类比神经元），负责处理数据，然后传递到下一层。最后是输出单元，负

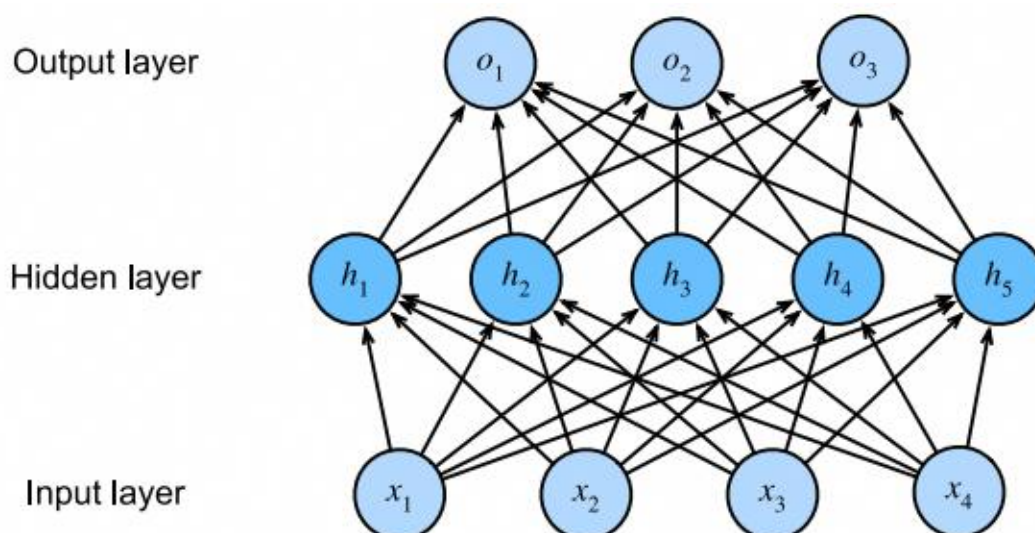
责计算 $h_{\theta}(x)$ 。第一层称为输入层(Input Layer)，最后一层称为输出层(Output Layer)，中间层全部称为隐藏层(Hidden Layers)，隐藏层通常不止一层，存在大量的隐藏层。

回看线性回归

我们再回到前面讲解的线性回归，回顾预测房价的示例，会发现线性回归其实就是一个单层神经网络：没有经过其他隐藏层，直接输入变量 x ，每个 x 对应一个 θ 参数，作用之后得出了 $h_{\theta}(x)$ ，相当于只有一个神经元的单层神经网络：



再回到神经网络，本质就是在单层神经网络的基础上，叠加多个神经元，并且一层层的往上叠加多层，每一层都输出到上面的层，直到生成最后的输出，就变成了一个庞大的神经网络。但拆开其中的每个神经元，其实就是之前学习过的线性回归：



可以把前 $L-1$ 层看作表示，把最后一层看作线性预测器。这种架构，也叫多层感知机（multilayer perceptron，缩写为 MLP）。

接下来，我们将分成几个小章节，分别介绍神经网络涉及的一些运算和基本概念，最后再将他们像搭积木一样组合起来，就能构建出实用的神经网络模型了。

激活函数

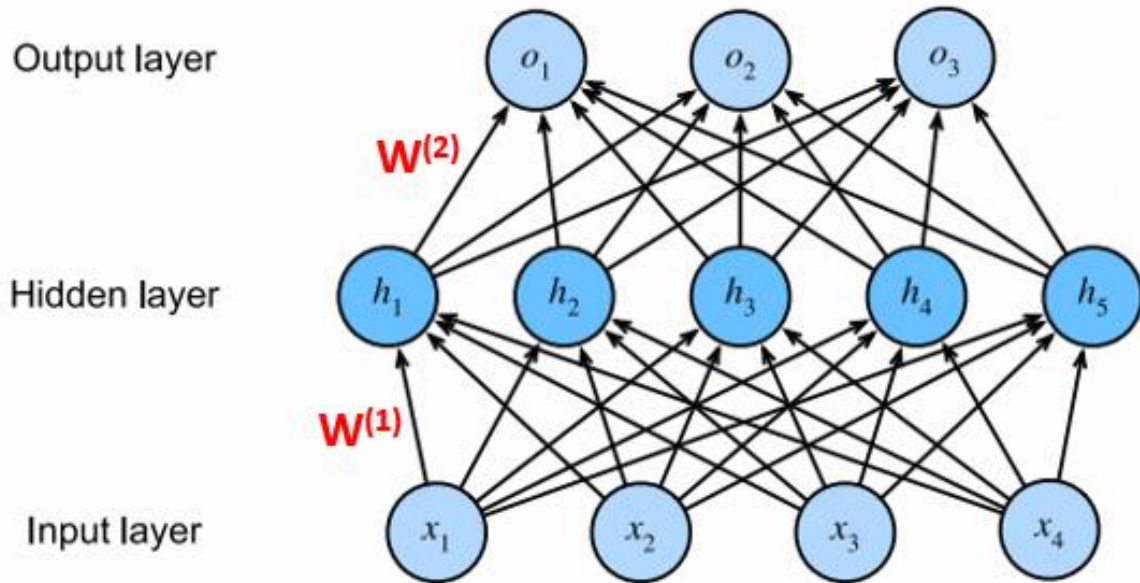
有了前面的理解，神经网络就是线性回归的叠加而成，尝试将网络用数学表达出来，以三层为例：

输入层：矩阵 $X \in \mathbb{R}^{n \times d}$ 来表示 n 个样本，其中每个样本具有 d 个输入(特征)。

隐藏层、输出层权重及偏置： $W^{(1)} \in \mathbb{R}^{d \times h}$ 和隐藏层偏置 $b^{(1)} \in \mathbb{R}^{1 \times h}$ 以及输出层权重 $W^{(2)} \in \mathbb{R}^{h \times q}$ 和输出层偏置 $b^{(2)} \in \mathbb{R}^{1 \times q}$ （偏置是个常变量，相当于前文中的 θ_0 ）

隐藏层输出： $H \in \mathbb{R}^{n \times h}$ 。

输出： $O \in \mathbb{R}^{n \times q}$



隐藏层输出： $H = XW^{(1)} + b^{(1)}$,

输出层： $O = HW^{(2)} + b^{(2)}$.

将隐藏层的输出 H ，代入到输出层（这其中使用到了线性代数中向量和矩阵相关的运算，读者可以动手做个公式的展开）：

$$O = (XW^{(1)} + b^{(1)})W^{(2)} + b^{(2)} = XW^{(1)}W^{(2)} + b^{(1)}W^{(2)} + b^{(2)} = XW + b$$

到这里，就会发现一个问题，虽然层次变多了，但最终的作用与一个单层的线性回归网络并无本质区别，依然可以用一个单层网络来表达，并未起到多层的作用。为了解决这个问题，引入了激活函数。

从前面的推导，不难发现，多层之所以没有起到作用，是因为每一层都是线性的，

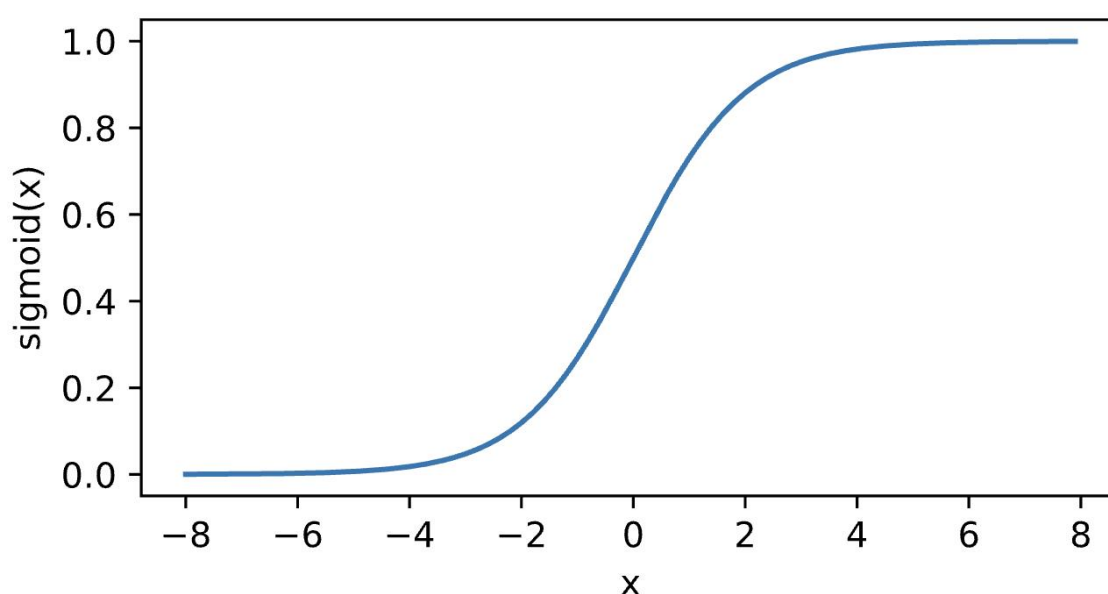
所以不管叠加多少层，最后得出来的网络依然是线性的，效果和单层无区别。如果要使多层网络有效，就需要打破线性关系，将线性变成非线性。比如输入层到第一个隐藏层，运算完成后，再增加一个非线性的激活处理：

$$H = \sigma(XW^{(1)} + b^{(1)}),$$

$$O = HW^{(2)} + b^{(2)}$$

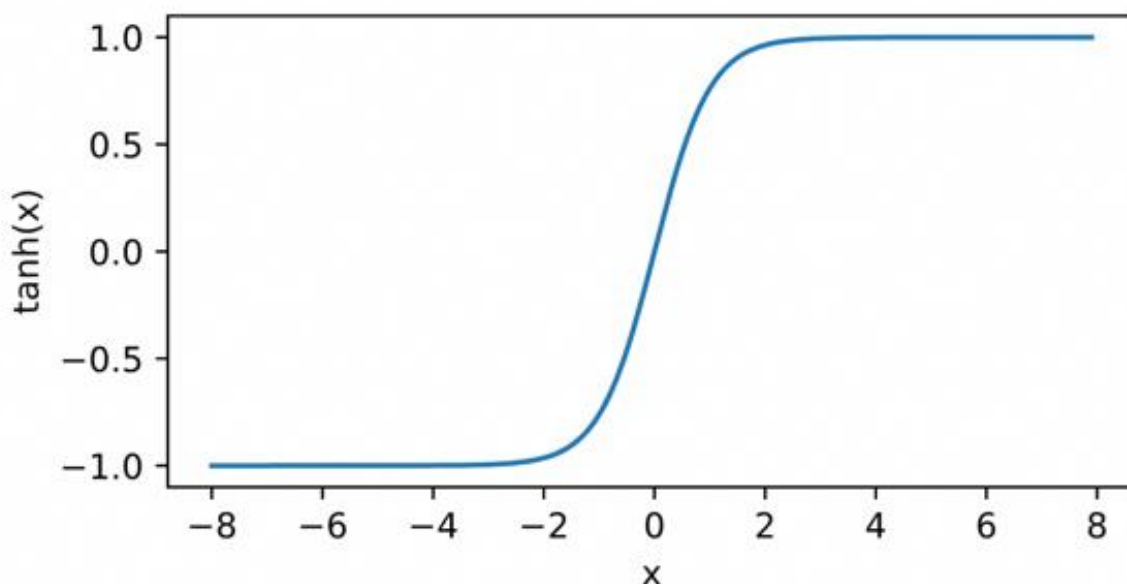
接下来，看看常用的激活函数，看如何打破线性为非线性。第一个 S 型 (sigmoid)：可以将任意输入值，映射到 (0, 1) 输出，非常方便概率问题的处理，应用广泛。比如分类问题，预测一张图片中是一只猫还是一只狗，最后输出值就是属于某个类别的概率，比如输出是猫的概率是 0.9，是狗的概率只有 0.05，那就认为这张图片是一只猫。

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



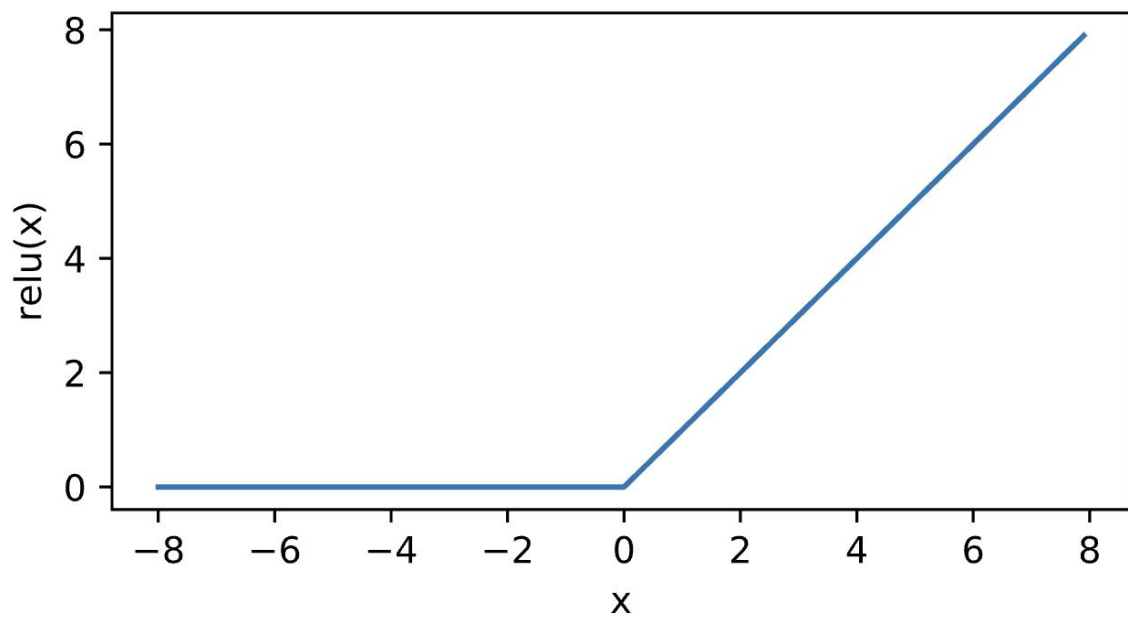
双曲正切（tanh）：可以将任意输入，映射到 $(-1, 1)$ 输出。

$$\tanh(x) = \frac{1 - \exp(-2x)}{1 + \exp(-2x)}$$

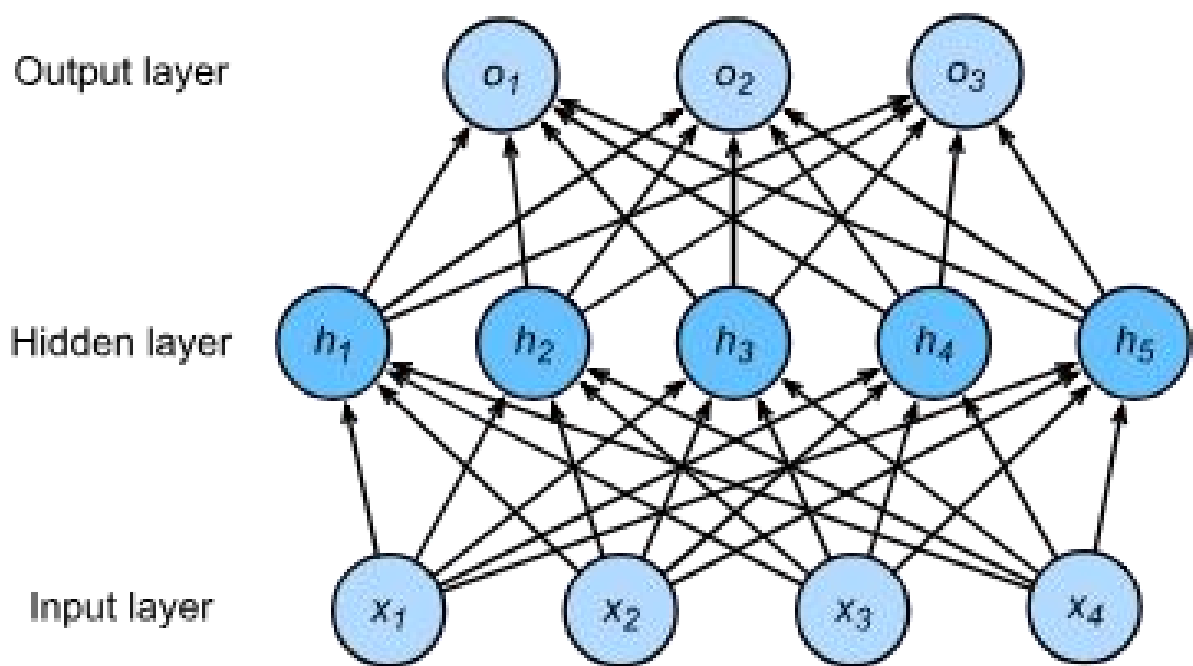


ReLU：线性修正单元。ReLU 激活函数，在神经网络中应用非常广泛，处理逻辑非常简单。只要小于 0，输出就是 0。大于零，输出与输入一致。直观上的理解，该函数可以将小于 0 的部分筛选掉，只留下大于零的部分，这样经过多层处理后，就可以提炼出关键特征。

$$\text{ReLU}(x) = \max(x, 0)$$



Softmax: 多分类



多分类问题是一种常见的问题，比如图片分类，见上图，假设最后输出有 3 类，比如有猫、狗、鸟 3 类。可以将图片按像素点的值作为输入 x ，训练一个神经网络，中

间可能有大量的隐藏层，最后一层有三个输出值 O_1 、 O_2 、 O_3 ，分别代表猫、狗、鸟 3 类。当模型训练完毕后，该网络的各个参数都已确定好了。再输入一张全新的图片，最后会输出 O_1 、 O_2 、 O_3 三个值，有个最简单的判断方法，比较 O_1 、 O_2 、 O_3 哪个值最大，则认为图片就是属于该类。

还有个更好的方式，将分类问题转变成概率问题，分别计算出每种类别的概率，选取概率最大的为最终的类别。比如，用 O_1 、 O_2 、 O_3 分别除以三者之和，就可以得出三个类别分别对应的概率：类别 1 的概率： $O_1/(O_1+O_2+O_3)$ ，类别 2 的概率： $O_2/(O_1+O_2+O_3)$ ，类别 3 的概率： $O_3/(O_1+O_2+O_3)$ 。因为涉及除法，就必须考虑分母不能为零的问题，使用 e 函数 $f(x)=e^x$ 做个处理即可，最终就变成了如下形式，这就是 Softmax 函数：

$$y_1, y_2, \dots, y_k = \text{softmax}(o_1, o_2, \dots, o_k)$$

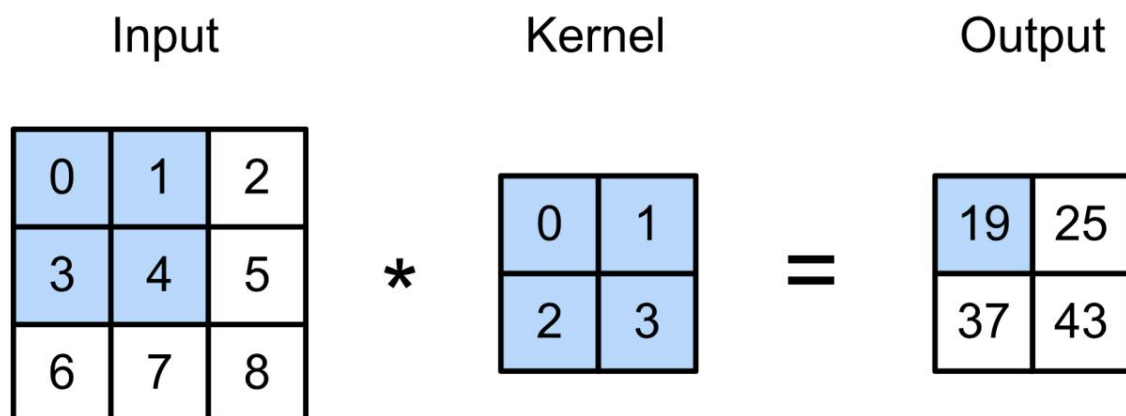
$$\hat{y} = \text{softmax}(\mathbf{o}) \quad \text{其中} \quad \hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}$$

softmax 特点：可看出所有的 j 总有 $0 \leq \hat{y}_j \leq 1$ ，非负并且各 \hat{y} 相加之和恒等于 1。这样一个多分类问题，最后的输出层，就变成了各类别的概率。

卷积：空间特征提取

估计对 AI 有过一些了解的读者，可能听说过卷积神经网络（缩写：CNN）。卷积是卷积神经网络的一大精髓。前面提到图像有两大空间特征：平移不变性和局部相关性。比如图片其实是一个由各像素点数值构成的二维矩阵（可能有多通道）。卷积中有个关键的概念，叫做卷积核，常用的卷积核有 $1*1$ ， $2*2$ ， $3*3$ ， $5*5$ 等几种，

卷积核可以直观理解为就是个长宽一样的小矩阵，以 2×2 的卷积核为例，由 4 个数值构成，如下图所示，



卷积核的运算：直观理解就是拿卷积核这个小矩阵逐列逐行按顺序一行行去扫描图片这个大矩阵，每扫到一个位置时，图片覆盖区域对应位置的数值和卷积核上对应的数值分别相乘（点积），最后再加总求和，得出该位置的一个数值，记录下来作为结果矩阵的数值。图片扫描完毕，将得出一个新的结果矩阵。比如扫到第一个初始位置时，计算结果是 19，其他位置类似，计算过程如下：

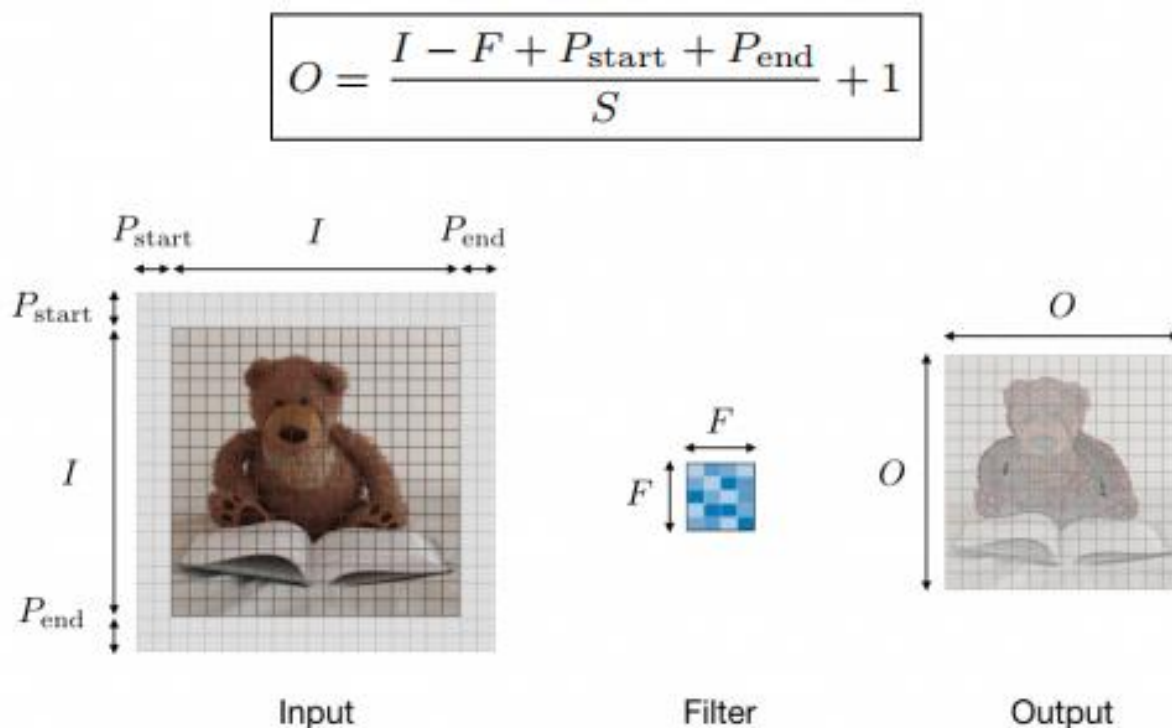
$$\begin{aligned} 0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 &= 19, \\ 1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 &= 25, \\ 3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 &= 37, \\ 4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 &= 43. \end{aligned}$$

从网上找到一个动图 ([链接](#))，可以直观的做个理解：用一个 3×3 卷积核去扫一个 5×5 的输入矩阵。最后会得到一个 3×3 的输出矩阵。

3_0	3_1	2_2	1	0
0_2	0_2	1_0	3	1
3_0	1_1	2_2	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

正常情况下，如果不对输入矩阵的行和列做全零填充，卷积之后，输出矩阵的大小会减小。可以想象一下，现在手上拿着个 3×3 的卷积核矩阵去扫这个 5×5 的输入矩阵，以列为例（行的理解类似），前面（即到 5-3 的位置）每走一步都有一个输出，一直扫到输入矩阵的末尾，所以输出矩阵的列数是 $(5-3) / 1 + 1 = 3$ 。其中 /1 是因为每次扫描的步幅是只向后走一位（实际应用时，针对大图片，为减少计算量，有些模型可能采用每次跳多步，则需要相应的除以步幅）；+1 是对应最后一位，就是 3×3 卷积运算到最后一位，作用完了之后，它只会得出来一个数值，需要再加 1。由此，再考虑到对输入图片行列首尾位置可能添加一些全零填充，可以推算出输出矩阵大小计算公式：



卷积的数学表达：

- **X**: $n_h \times n_w$ 输入矩阵
- **W**: $k_h \times k_w$ 核矩阵
- b: 偏差标量
- **Y**: $(n_h - k_h + 1) \times (n_w - k_w + 1)$ 输出矩阵

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

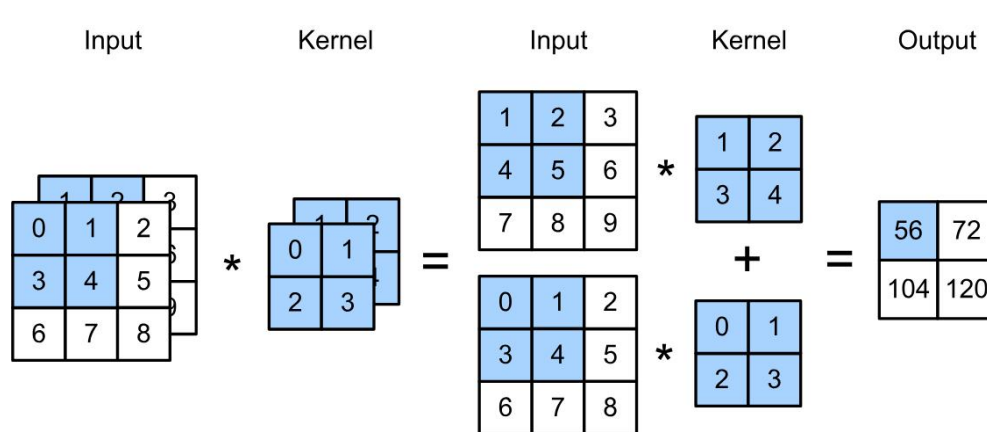
- **W** 和 b 是可学习的参数

卷积为什么会在图像处理上有效？笔者尝试从卷积的运算原理去理解：卷积可高效提取临近像素间的相关特征，卷积核及偏差标量是需要被训练的参数。前文提到过，图片特征是局部相关的，卷积就能处理这个问题。卷积核每次扫输入矩阵时，作用域是固定的小区域，并且是严格按行列顺序逐步扫描的，这样就可以有效提取到局

部相关性特征。此外，针对图像的平移不变性，卷积也是有效的。因为图片上的关键像素区域（比如识别狗，这只狗占据的那些像素区域称为关键像素区域，其余像素可以看做是背景杂质），不论处于图片中什么位置，理论上对于同一个卷积核，在这些关键像素区域处理后得出的运算结果是类似的。

多输入通道及多输出通道

比如 RGB 格式的彩色图片，有三个输入通道，每个像素由 R、G、B 三个数值表示。卷积核也对应变成了三个通道，可形象的称为 3-D 卷积核，卷积核的每个通道和其对应的输入通道上的数值矩阵相运算（点积），最后各通道对应的卷积结果再相加。以两个通道做示例，见下图：



示例：结果矩阵（输出矩阵）的第一位计算过程：

$$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) + (0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) = 56$$

数学表达式：

- 输入: $\mathbf{X}: c_i \times n_h \times n_w$
 - 卷积核: $\mathbf{W}: c_i \times k_h \times k_w$
 - 输出: $\mathbf{Y}: m_h \times m_w$
- $$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:} \star \mathbf{W}_{i,:}$$

输出通道也可以是多通道的，可以构建一系列的 3-D 卷积核和输入矩阵相作用，每个 3-D 卷积核将生成一个输出通道（输出矩阵）。在互相关运算中，每个输出通道先获取所有输入通道，再以对应该输出通道的卷积核计算出结果。

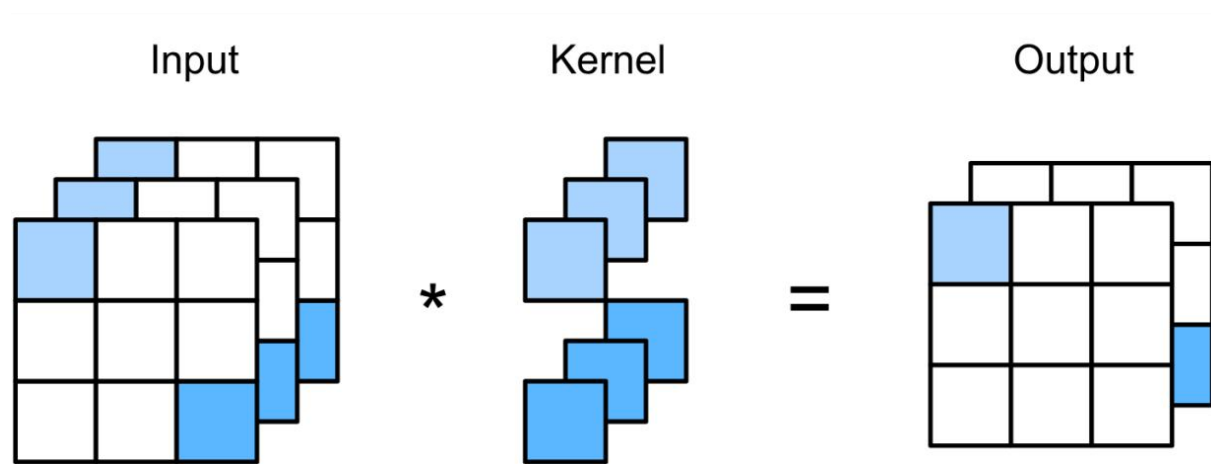
- 输入 $\mathbf{X}: c_i \times n_h \times n_w$
 - 卷积核 $\mathbf{W}: c_o \times c_i \times k_h \times k_w$
 - 输出 $\mathbf{Y}: c_o \times m_h \times m_w$
- $$\mathbf{Y}_{i,:} = \mathbf{X} \star \mathbf{W}_{i,:} \quad \text{for } i = 1, \dots, c_o$$

可以理解为，图像中可提取的特征是复杂多样的，比如颜色、轮廓线条等等。每个 3-D 卷积核的作用是有限的，只能提取到部分特征，需要大量的卷积核。比如，识别图片中是否有一只猫，第一个 3-D 卷积核提炼到，相关像素点是不是绿色的。第二个 3-D 卷积核提炼到，相关像素点是不是红色的。还有一些卷积核，可以提取黑白相间的线条，类似于轮廓等等。。。。。。为了提取图片里面丰富的特征，需要大量的 3-D 卷积核，对应多个输出通道。对于卷积的大致参数量，其实就是卷积核对应的参数量，即 $c_o \times c_i \times k_h \times k_w$



特殊的卷积：1*1 卷积





前面提到的常见的卷积，2 乘 2，3 乘 3，或者 5 乘 5 等，可以提取图片的特征。还有一个特殊的 1*1 卷积： $k_h=k_w=1$ 。这个卷积运算完后，发现图片的长宽并不会变化，但通道数会变化，因为一组卷积核对应一个输出通道。可以用 1*1 卷积去调整输出的通道数量。



卷积的难以解释性

卷积神经网络中有个不是特别好的地方：解释性不强。

比如模型训练完成后，有大量的卷积核，但这些卷积核为什么是这样子的，为什么是这些数值？这个卷积核的具体作用是什么？在大多数情况下，都是很难解释清楚的。仅有少数卷积核的作用，可以较直观的体现，比如维基百科上有些例子：

	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$		边缘检测
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$		锐化
(维基百科)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$		高斯模糊

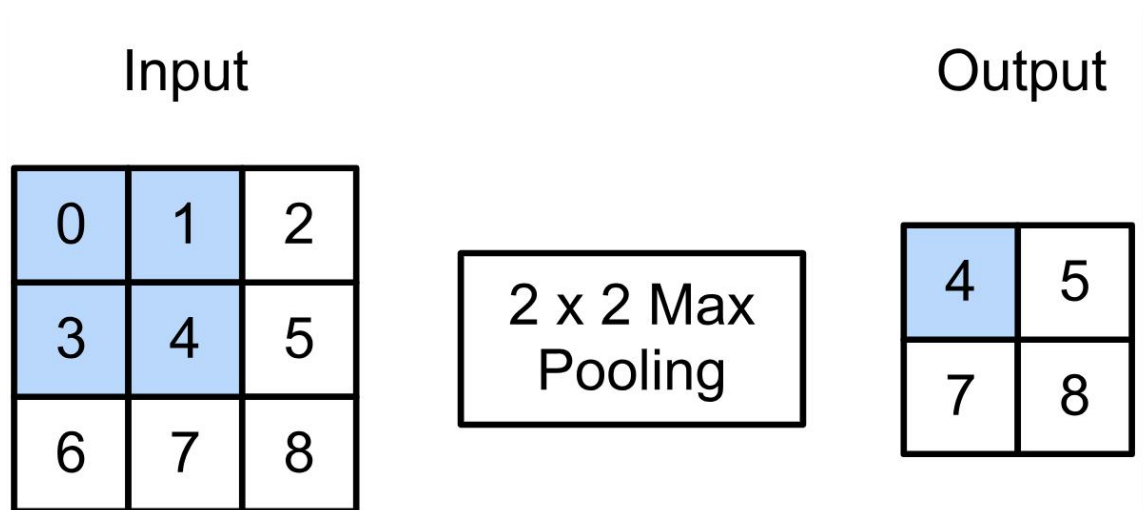
实际使用时，并不会人为构建具体的卷积核，卷积核的参数值都是训练出来的，而且通常可解释性较弱。有人尝试给 CNN 做出一些解释，并制作了一个网站，感兴趣的可以登录看看，CNN explainer:

<https://poloclub.github.io/cnn-explainer/>

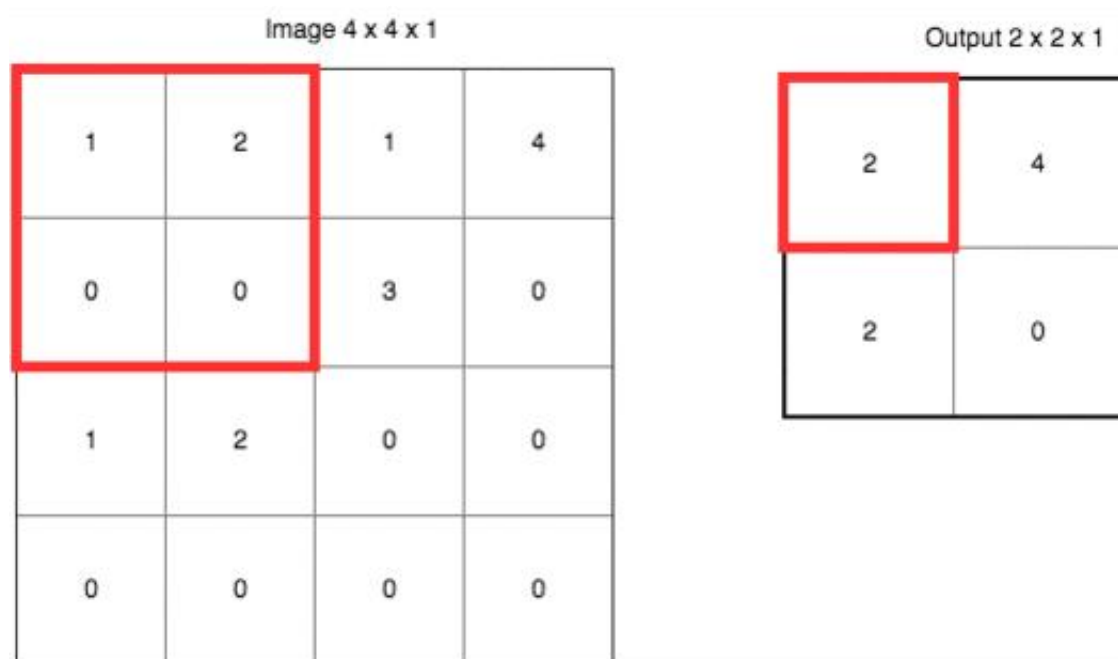
池化/汇聚(Pooling): Max Pooling、Average Pooling

还有一个比较常见的运算：池化（有些书上翻译成汇聚，英文名称是 Pooling）。主要有两种 Pooling 操作：Max Pooling（最大值池化）和 Average Pooling（平均值池化）。顾名思义，对比前文讲的卷积（先对应位置做两两点积，最后再相加），Pooling 操作很简洁：依然类似卷积一样逐行扫描输入矩阵，但只是取该 Pooling 矩阵内的最大值或者平均值（其中取最大值应用更为广泛，平均值在早前的神经网络

中有些应用)。此外,值得注意的是,每次滑动的步幅通常和 Pooling 矩阵的长度或宽度一致,比如 2×2 的 Pooling 操作,通常每次滑动的步幅也是 2,示例:



找到一个网上的动图 ([链接](#)), 可加深理解:

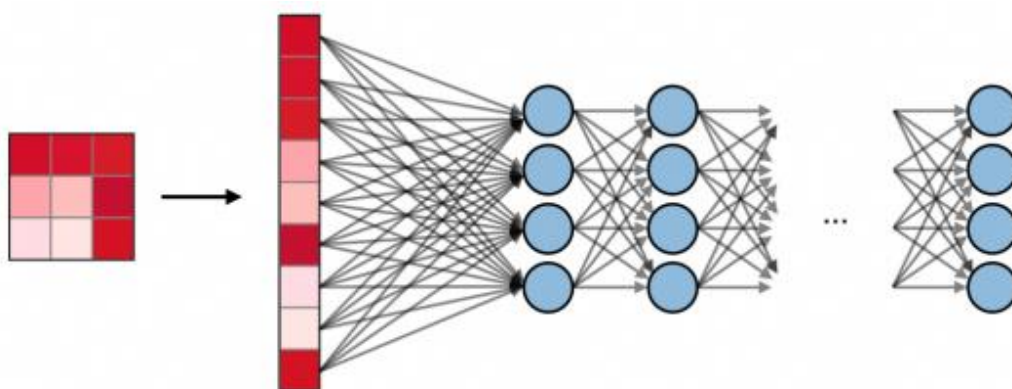


直观理解 Pooling 的作用: Max Pooling 实现了抓局部窗口内的关键信息, 把一些不重要的信息忽略掉, 对平移不变性可以做到更鲁棒, 过滤掉一些噪声的影响。

池化操作，可以将输入矩阵的维度降低，只留下关键信息，比如 2×2 的 Pooling，会使输入矩阵的长宽均减半，输出矩阵的维度可以降为输入矩阵的 $1/4$ 。

全连接（Fully Connected）

全连接常用于神经网络的最后几层。以图像处理为例，图像可以理解为多通道的二维矩阵，经过层层卷积、激活、池化等运算后，得到最后的输出矩阵，这时，可以把该矩阵按行的顺序展平为一个一维向量，与下层的神经元做全连接运算（每个输入值都和下一层的神经元相运算），而且可以构建多层的全连接（其实就是前文介绍的 MLP），示例：



最后一层得出最终的输出（可以看作一个向量），比如图片分类或者识别问题，又可以进一步使用前文介绍的 Softmax 运算，转换成一个 0-1 的概率值，用来表示该图片属于哪一类的概率。

深入理解经典 CNN

有了前面的这些基础知识（如卷积，Relu 激活，Max Pooling，Fully connected，Softmax 等），就可以进一步深入学习理解经典的 CNN 网络，如：LeNet、AlexNet、VGG、ResNets、GoogleLeNet 等。

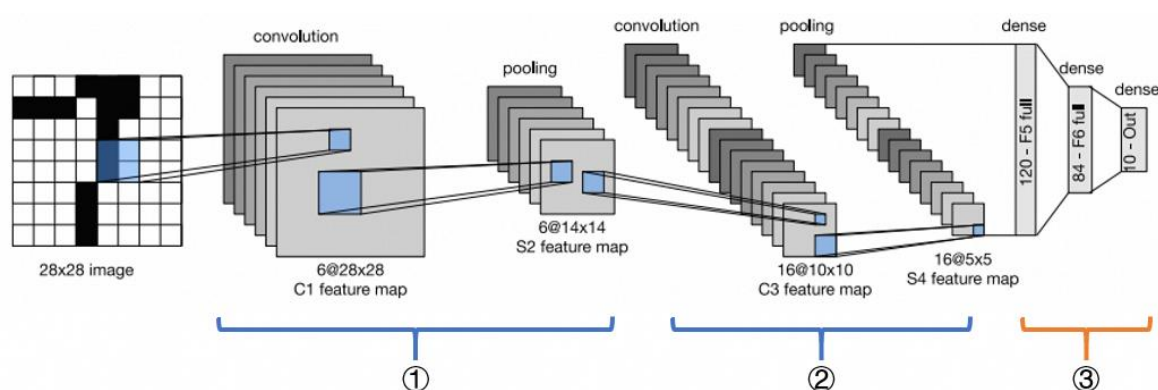
以 LeNet 为例，做个拆解剖析。LeNet 在 1989 年的时候就提出来了，广泛的应用在手写数字识别场景。比如早期八九十年代，发信件、邮件等，需要填邮编，写支票等场景，都涉及手写数字。LeNet 在这些场景，取得了很好的应用效果。



拿到一个 AI 训练任务后，可以先从数据样本集入手做个分析。以 MNIST 数据集为例，共 50,000 个训练数据，10,000 个测试数据。每个图像大小为 28*28，因为是黑白图片，不涉及彩色，输入通道只有一个。数字为 0~9，共 10 个，所以数据样本共 10 类，最终的输出也有 10 个（每个输出值关联一个数字）。

回到 LeNet 网络本身，LeNet（LeNet-5）由两大部分组成：

- **卷积编码器**：由两个卷积块组成（见①和②）。每个**卷积块**中的基本单元是一个卷积层（ 5×5 卷积核）、一个 sigmoid 激活函数和一个平均池化层。
- **全连接层（密集块）**：由三个全连接层组成（见③）。



卷积编码器：回想前文讲的卷积，可以有多个输入通道（如 RGB 彩色图片，原始图像有三个输入通道），也通常有多个输出通道。每一个输出通道都能提取到图像的一些规律或者特征。**LeNet** 第一卷积块只有 1 个输入通道（黑白图片对应一个输入通道），6 个输出通道。而第二个卷积块有 6 个输入通道（对应第一个卷积块 6 个输出通道，池化处理后输入到第二个卷积块），16 个输出通道。均选用了 5*5 的卷积核。

LeNet 的第一层卷积输出矩阵和输入矩阵大小一致，原因是对输入矩阵做了填充：在图像矩阵的边框位置，做全零填充，输入矩阵会变大。对原始图片矩阵在边框上做全零填充，从而更好的保留在边框位置上的像素输入。如果不做任何填充，则输出矩阵大小应该是 $(28-5)/1+1=24$ ，即 $24*24$ 。

由此，可以反推得到共填充了 4 行 4 列，也即前后各两列，上下各两行，真实的输入矩阵大小变成了 $28+4=32$ ，也即 $32*32$ ，才能使得输出矩阵大小保持在 $28*28$ 。第二卷积块中的输入矩阵为 $14*14$ ，不做填充，输出矩阵大小为 $(14-5+1)$ ，也即 $10*10$ 。

卷积之后, 选用了 **sigmoid** 激活函数做激活。之后做了一个平均池化, 每个 2×2 池化操作后, 都会将数据矩阵的数据量减少 4 倍 (直观上理解, 每 4 个输入值被处理成了一个输出值, 输出矩阵的数据量会变成输入矩阵的 $1/4$) , 可以直观的看到, 池化操作会大大减少计算量, 因为会直接将输入矩阵数倍的缩小。此外, 值得注意的是, **LeNet** 选用了平均值化, 而不是最大池化, 卷积神经网络发展到后面, 发现最大池化通常效果更好, 应用比平均池化也更多。

全连接层 (密集块) : 两层池化后, 得到 16 个 5×5 的输出矩阵。把这些矩阵按顺序展平为一个包含 $16 \times 5 \times 5$ 个元素的一维向量, 基于此做全连接。**LeNet** 做了三层全连接后, 得出最后的输出, 最后一层全连接由 10 个神经元组成, 分别对应 10 个数字 (0~9) 的输出。

像搭积木一样构建神经网络

对 **LeNet** 有了理解后, 可以像搭积木一样, 使用 **PyTorch** 将 **LeNet** 的代码写出来, 并可以更进一步搭建自己的 **CNN**。强烈建议搭建一个自己的开发环境, 并导入一些开源开放的示例代码, 深入学习并实际运行, 这样就能更清晰的理解了。本章节结尾处的动手实践部分, 已经给出了详细的指导书。**LeNet** 的关键代码如下 ([详见参考《动手学深度学习 PyTorch 版》](#)) :

```
net = nn.Sequential(  
    nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.Sigmoid(),  
    nn.AvgPool2d(kernel_size=2, stride=2),  
    nn.Conv2d(6, 16, kernel_size=5), nn.Sigmoid(),  
    nn.AvgPool2d(kernel_size=2, stride=2),
```



```
nn.Flatten(),  
nn.Linear(16 * 5 * 5, 120), nn.Sigmoid(),  
nn.Linear(120, 84), nn.Sigmoid(),  
nn.Linear(84, 10))
```

代码解读：第一层卷积采用卷积核大小为 5 的 2D 卷积，输入通道为 1，输入通道是 6。Padding 填充值为 2，也即共填充 4 行 4 列，前后各两列，上下各两行。随即进行 Sigmoid 激活。然后做 2*2 的平均池化。由此，可以计算出对应的参数量（激活和池化不涉及训练参数，主要是卷积涉及）：输出通道数*输入通道数*卷积矩阵参数，即 $6*1*5*5=150$ 。

第二层卷积采用卷积核大小为 5 的 2D 卷积，输入通道为 6，输入通道是 16。随即进行 Sigmoid 激活。然后做 2*2 的平均池化。对应的参数量为 $6*16*5*5=2400$ 。

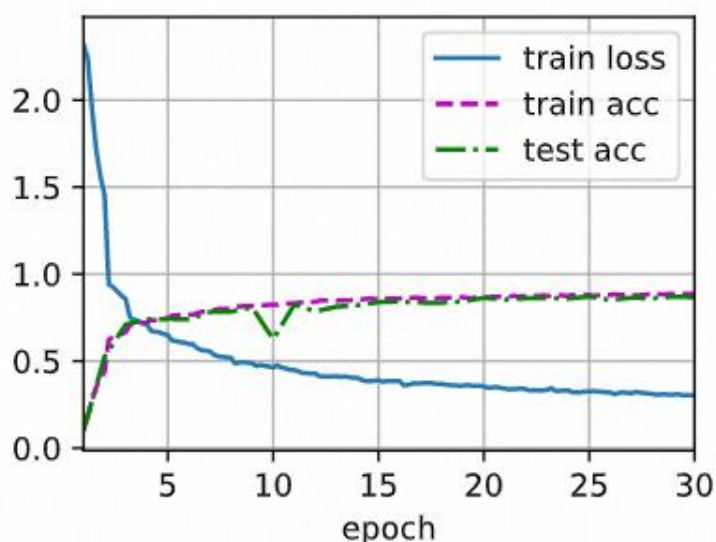
最后把输出矩阵展平，进行三次全连接运算，激活函数依然使用 Sigmoid。第一层全连接使用了 120 个参数。第二层全连接使用了 84 个参数。最后一层全连接参数只能选 10 了，因为是输出层，对应 0~9 共 10 个数字的输出，是个类别为 10 的分类问题。对应的大致参数量为 $120+84+10=214$ 。

将上述几部分的参数量相加，即可计算出 LeNet 的大致参数量级：2764。值得说明的是，实际参数量会稍多于该值。因为计算深度学习网络参数量时，通常为了简便，略去了卷积、全连接等涉及的偏差标量，偏差标量不影响深度学习网络的参数量量级。

算法训练基本情况：

```
lr, num_epochs = 0.9, 30
train_ch6(net, train_iter, test_iter, num_epochs, lr, d2l.try_gpu())
```

```
loss 0.304, train acc 0.886, test acc 0.867
8825.3 examples/sec on cpu
```



动手实践：搭建一个自己的神经网络-手写数字识别 CNN

学习到这里，可能有些比较爱动脑筋的人会问，LeNet 为什么只做两个卷积块，不能做更多的吗？为什么激活函数选 `sigmoid`，不能选其他的吗？为什么卷积核是 `5*5`，不能选 `3*3`，`7*7` 吗？为什么全连接层的参数是 `120`、`84`，不能选其他的参数吗？为什么只做三次全连接，不能做四层、五层等更多层吗？等等系列问题。其实这里面并没有严格的限制，还记得前文讲过 AI 的训练过程，其实是个不断的工程实践、暴力计算的过程吗？这些问题很多都涉及神经网络训练中的超参数，需结合自身经验不断测试验证，综合考虑模型最终的推理效果，训练成本等因素得出最佳实践。

所以，网上也戏称 AI 训练为炼丹，称从业者为 AI 炼丹师，对训练过程火候的控制，各项成分的配比等等，都需要日积月累不断的知识积累、最佳实践，方有小成。

LeNet 是很早期训练出来的 CNN，能满足当初的应用效果即可，但也并非完美。更进一步，我们可以根据学习到的 CNN 原理，对 LeNet 做一些改进，构造一个自己的且效果更好的手写数字识别 CNN。比如，因为输入矩阵比较小，手写数字需要提取的特征又相对较简单，卷积核可以尝试选择小一点的，比如 3×3 的，这样计算量更小，训练更快；前面提到过，Relu 激活其实在后面的 CNN 网络中应用更广，激活函数可以试试 Relu；平均池化可以换成最大池化，看看效果是否会更好。。。。。。等等，都可以动手尝试。

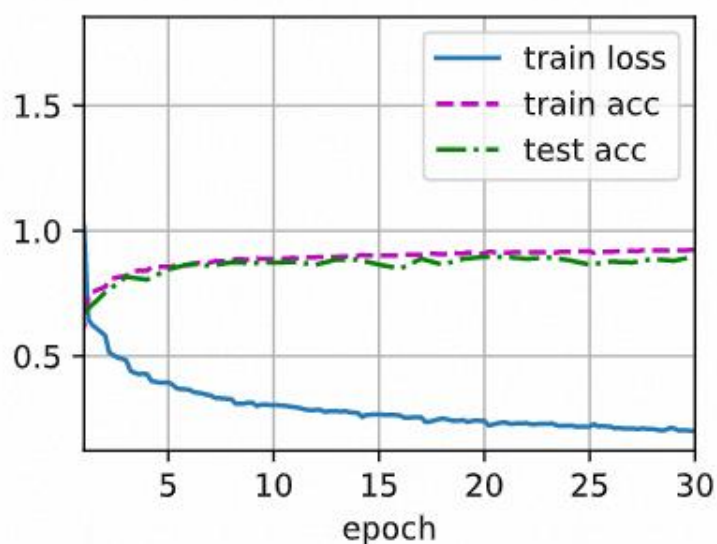
示例代码：

```
"""更换成更为通用的 3*3 卷积，ReLU 激活，最大池化层"""
net = nn.Sequential(
    nn.Conv2d(1, 6, kernel_size=3, padding=1), nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Conv2d(6, 16, kernel_size=3), nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2),
    nn.Flatten(),
    nn.Linear(16 * 6 * 6, 120), nn.ReLU(),
    nn.Linear(120, 84), nn.ReLU(),
    nn.Linear(84, 10))
```

这样的话，我们可以训练自己的 CNN 了。可以看到完整的训练过程，算法的收敛情况，以及算法效果（训练误差、泛化误差等）等。

```
lr, num_epochs = 0.1, 30  
train_ch6(net, train_iter, test_iter, num_epochs, lr, d2l.try_gpu())
```

```
loss 0.201, train acc 0.924, test acc 0.897  
15354.4 examples/sec on cpu
```



可以明显的看到，训练误差更小，收敛速度也更快。只是对 LeNet 做了为数不多的改动，就能得到更好的手写数字识别 CNN。当然我们现在是站在了巨人的肩膀上，吸取了前人的大量经验。人家是八九十年代，各种理论都尚不完善的情况下，就把 LeNet 做出来了，而且应用效果很好，具有开创性。

更多经典 CNN

有了前面的积累，我们可以学习和理解更多的经典 CNN，比如 AlexNet、VGG、ResNets、GoogleLeNet 等。

AlexNet 于 2012 年推出，并赢得该年 ImageNet 图像识别大赛冠军。引领 CV 领域重回 CNN，证明了深度神经网络卓有成效，再次掀起了深度神经网络热潮。展开看，可以看到是基于早期的 LeNet 做了扩展，这里截选了《动手学深度学习 PyTorch 版》中的一个对比图，见下：

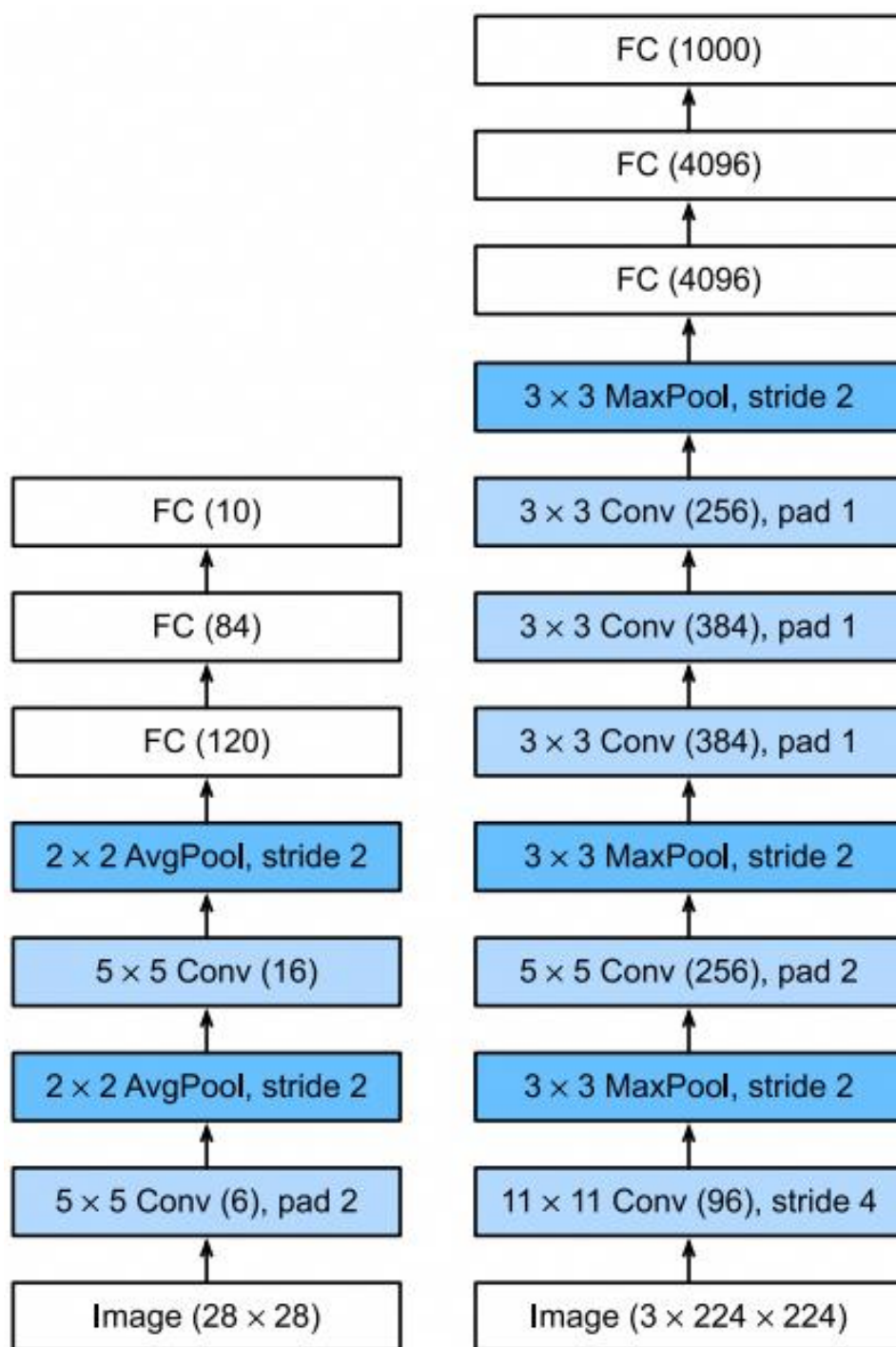


图7.1.2: 从LeNet (左) 到AlexNet (right)

主要差异:输入的图片变大了, 因为之前 LeNet 是针对 28×28 的手写数字黑白图片。

AlexNet 是针对 1000 类图像识别问题, 数据集使用了 ImageNet, 输入图片是 224×224 的 RGB 彩色图片。核心模块, 依然是卷积、激活、池化、全连接这些。

不难看出, AlexNet 其实就是更深更大的 LeNet: 采用了 8 层 CNN。激活函数使用

更简单的 ReLU。使用更有效的最大值汇聚。此外，采用了 GPU 做训练。

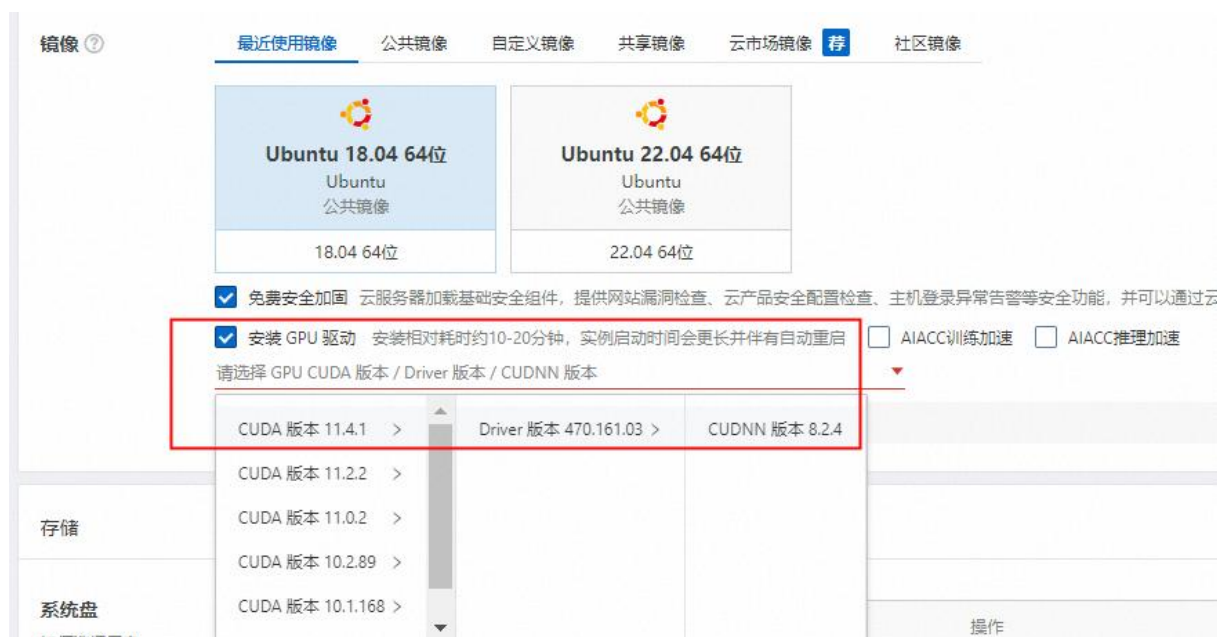
可以看到 AlexNet 的核心主要是把 CNN 做的更深了，效果就变的很好，一举夺得了 ImageNet 图像识别大赛冠军。后面的人受此启发，逐步将网络做的更深更大，神经网络开始真正进入了深度学习，出现了 VGG、ResNets、GoogleLeNet 等网络，网络的设计也走向了模块化，取得了更好的效果，直到后面准确率已经超越了人类。

动手实践：在云上搭建深度学习 notebook 开发环境

公共云提供了极大便利，让开发测试变得简单，计算资源便捷可得，随用随开。以阿里云上的 ECS 为例（有条件的可以选 GPU 实例），操作系统选了 Ubuntu 18.04 64 位，搭建 pytorch 开发环境，并安装《动手学深度学习 PyTorch 版》学习包（该学习包含有大量示例代码，且配套有教程，适合入门学习）：

Step1：创建 ECS 实例，设置用户名及登录密码（使用 ecs-user 登录，如果使用 root 安装，后续部分安装包会提示有权限风险）。

注意：如果选择 GPU 实例，需在**实例创建时**选择安装 **cuda 11.4**。



Step2: 设置安全组配置，配置出方向端口 22（用于远程 ssh 登录），在源 IP 中加入本机 IP（可以在百度中搜“本机 IP”，即可获取）。

Step3: ssh 到云 ECS。

Step4: 在 ECS 上安装相关软件包：

- a. `sudo apt-get update`
- b. 安装 gcc 编译器(可选, 系统中通常已有): `sudo apt install build-essential`
- c. 安装 Python, 以 3.8 为例: `sudo apt install python3.8`
- d. 安装 miniconda:

(1) 下载安装包: 注意 miniconda 包需选择对应 python 版本的包

(2) `wget`

`https://repo.anaconda.com/miniconda/Miniconda3-py38_23.3.1-0-Linux-x86_64.sh`

(3) 运行安装脚本, 并初始化: `bash Miniconda3-py38_23.3.1-0-Linux-x86_64.sh`

(4) (可在 2 中完成) 初始化终端 Shell, 以便运行 conda。


```
~/miniconda3/bin/conda init
```

(5) 初始化完成后，运行 bash 命令，即可进入 conda 环境：bash

(6) 创建一个新的环境（可选）：conda create --name aigc python=3.8 -y

(7) 激活 aigc 环境（可选）：conda activate aigc

e. 安装 jupyter: pip install jupyter

f. 安装 pyTorch: conda install pytorch torchvision torchaudio cpuonly -c pytorch

g. 安装 d2l 包: pip install d2l==0.17.6

Step5: 安装解压工具及 git 工具: sudo apt install unzip 安装 git: sudo apt install git

Step6: 下载 d2l notebook 示例代码，并解压：

```
mkdir d2l-zh && cd d2l-zh      wget https://zh-v2.d2l.ai/d2l-zh-2.0.0.zip
```

```
unzip d2l-zh-2.0.0.zip
```

Step7: 本地便携机上执行如下命令，将云 ECS 的 8888 端口映射到本地：

```
ssh -L8888:localhost:8888 ecs-user@47.106.127.176
```

Step8: 在 ECS 上运行 jupyter: jupyter notebook

Step9: 在本地浏览器登录 jupyter（填写运行 notebook 时，展示的正确 URL）：

```
http://localhost:8888/?token=rWxo4Uo\*\*\*
```


环境安装参考资料: https://zh.d2l.ai/chapter_installation/index.html

讲解视频

为进一步方便学习, 录制了讲解视频, 详见:

https://www.bilibili.com/video/BV17oN3eAEbN/?vd_source=0c5a39664e27af522bfcd1e74b9780a8

AIGC 及大模型

硅基智能 vs 碳基智能：AIGC 带给我们的惊喜

此前 AI 的发展通常局限在特定领域的特定问题，ChatGPT 等 LLM 在文本摘要/概括、推断、文本转换、文本扩展等多种任务上均效果惊艳。AIGC 在智能问答、文生图、文生视频等场景迅速迎来火爆应用，掀起了新一轮 AI 热潮。企业、科研机构等纷纷入场，引发了全球 AIGC 大战，各种各样的 LLM、文生图等模型如雨后春笋般涌现，并在电商、泛娱乐、金融、政企等行业应用开来，带来了新变革：

- **搜索推荐**：搜素材转变为直接给答案。LLM 具备素材理解、加工、内容生成的颠覆性革新能力，正在重塑搜索行业。
- **人机交互、智能问答**：需要死记硬背、有应答框架的行业将受剧烈冲击，如信息咨询、顾问、客服、翻译、主播。游戏 NPC、元宇宙、机器人、虚拟人、数字克隆人等。
- **内容创作**：有规范模板、重复性强的工作，将迎来变革，如代码开发，DBA，图像/视频类的设计（如短视频、服装设计、人物风景画作、插画、海报等），辅助写邮件、摘要、文案、提纲、撰稿，辅助词曲、新闻、剧本创作、影视频、短视频创作等。
- **其他无限可能**：Depends Your Imagination。AIGC 各细分领域还在日新月异的发展，不断涌现颠覆式的新应用，大家可以拭目以待。比如，近期 OpenAI 发布了 Sora，在文生视频领域取得了新突破，在短视频、影视制作、元宇宙等领域带来新的冲击和想象空间。未来，借助 AIGC，比如文生文、文生代码、文生图、文生视频、文生 SQL 等。人人都可成为内容制造者、应用开发者。普通人也能拓展自己的职业，比如成为插画师、程序员、短视频大 V、作家、编剧、导演等等。设想下，未来借助文生文，普通人也可以编写精彩的故事剧本，结合文生图，文字又能生成精美的故事插画，也能成为作家、漫画家。更进一步，结

合文生视频，故事剧本能生成视频大片，甚至于，可以自定义视频影片中的男女主角形象，普通人也能成为男一号、女一号，打造自己的专属大片。以前，我们以写日记、博客等方式来记录生活和感悟，未来，个人的思想表达不再局限于文字，可以更进一步创作自己专属的画作、动画片、短视频、影片。随着 AIGC 技术的不断成熟，使用门槛的不断降低，这些都将不再是梦想，正在快速的实现。无论是工作、生活、学习等方方面面，AIGC 都正在给我们打开一扇大门。**创意将变的更有价值，且更具感染力和冲击力。文学、艺术、创意等在人的一生中将发挥更大的影响力**，教育行业或将很快迎来相应的调整，或将文学、艺术等当作学生的基础素养来学习和培养。

AIGC、大模型的蓬勃发展带来的惊喜和想象空间远不止于此。我们尝试站在人类智能发展这个更高的视角去看这个事情。人类的智能由一个个人类个体构成，往往是绝大部分个体对整体智能的影响是微乎其微、可忽略不计的。目前的知识体系主要源自历史长河中少数个体的突出贡献，逐步积累叠加、缓慢突破而成。

但发展过程中碳基生命有个重大缺陷：相比宇宙的历史长河，人类个体的生命仅仅数十年，太过短暂。且生命个体诞生后，知识几乎是空白的，也即知识无法由后代直接继承，每个新生个体需要重新开始一点点学习。

以一个程序员视角看人类智能的发展，有点近似一个死循环：一个生命个体出生后，类似一个近乎全 0 的初始化状态，然后需要耗费十数年甚至几十年时间不断的学习积累，才能掌握前人的一部分智慧，极少数的天赋异禀者可能会有一些突破，然后个体又会走到生命尽头。由诞生的新生生命重新清零，回到初始化状态，在大量的个体、不断的循环中取得一点一滴量变，进而到质变的向前突破。未来的硅基智能带来的想象力在于知识可以继承，不断向前迭代发展，大模型似乎隐约让人看到了实现 AGI（通用人工智能）的希望。

不过话说回来，目前的大模型虽然在很多场景有惊艳表现，但离能自我理解、发现、分析、甚至独立思考创造、具备认知智能甚至自我意识的 AGI 还相距甚远，一切才刚刚拉开序幕。

大语言模型（LLM）

接下来，将围绕 LLM、知识库+LLM 等几大热门应用讲解基本原理。针对 LLM 在某些场景上存在胡说八道的现象，探讨 LLM 存在的“知识茧房”缺陷，思考解决方法。详细介绍知识库的构建原理、系统架构等。并提供两个实战演练（知识库+LLM、钉钉智能问答机器人）的详尽实操指导书，使读者可以零基础构建自己专属的智能问答应用。

语言的数据特征

前面主要讲了图像、视觉相关的内容。还有一块和语言相关的也很重要，如文本、音频等，与之关联的是自然语言处理（英文简称为 NLP）。语言的数据特征是：时序、上下文关联(语境、权重)。比如，说话是一个一个字词往后说的，语句文章等的字词间都是有时间线先后顺序的，这叫时序特征。字词语句在一段文字中，存在上下文关联，相同的字词语句在不同的上下文语境中可能出现完全不同的意思。比如，网上曾流行的一个段子。一个女生跟一个男生约会，女生跟男生说：如果你到了我还没到，你就等着吧。如果我到了你还没到，你就等着吧。这里面的“你就等着吧”，虽然是一模一样的词句，但意思却截然不同。另外，很多语言都存在字词多音多义，疑问反问等多种修辞手法，也会导致相同的词句本身就存在多种含义。各字词在语言中的权重，重要程度也不相同，有所谓的关键字词，核心词句。这些问题都需要 NLP 算法来解决和处理。除了近期大火的大语言模型，NLP 领域前期也诞生了系列的深度神经网络，如 RNN、GRU、LSTM 等。

文本转向量（数字化）

AI 算法训练只能处理数字化后的数据，那语言中的字、词、句、段落等怎么转成数字呢？这个问题本身就是一个专门的技术领域。我们只介绍两个基础的方法，让大家有个认识：One-Hot 编码、Embedding 算法。

One-Hot 编码原理：对于自然语言，输入是一段文本，在中文里就是由一个个字、词标点符号组成，这些最基本的不再拆解的字词、标点符号、短语、前后缀等元素就称为词元，英文名称叫 Token。One-Hot 编码的基本思路是将句子拆解成词元，针对词元构建一个词表（有些书上也称作字典），然后借助词表中的每个词元所处的位置编号来表示该词元。任何语言，基本词元数量总是有限的，词元本身就能构建一个词表。比如汉语，汉字肯定是有数量上限的。将文本先按一定规则拆解成词元（或叫 token），就能从词表中查到这些词元对应的位置编码了，以该词元在词表中的位置编码表示该词元，也就实现了词元数字化，进而实现整个句子、文本的数字化。引用网上的一个示例（详见[链接](#)），给定文本：我们相信 AI 可以让世界变得更美好。先拆解成词元（存在多种拆解方式）：

- 按字 Token 化：我/们/相/信/A/I/可/以/让/世/界/变/得/更/美/好/。
- 按词 Token 化：我们/相信/AI/可以/让/世界/变得/更/美好/。
- 按 Bi-Gram Token 化：我们/们相/相信/信 A/AI/I 可/可以/以让/让世/世界/界变/变得/得更/更美/美好/好。

构建一个包含全部 Token 的词表，每一个 Token 都有固定大小的 N（N 是词表的大小）个特征，比如构建了一个有 4 万个词元的词表，N 就等于 4 万。除了该 Token 在词表中的位置值为 1，其余都为 0。如此，每个 Token 都可表示为一个向量，比如「我」：[1,0,...0]，向量的长度为词表大小 N。（说明：实际应用不会简单的用

1/0 来表示，因为每个字在句子中的作用是不一样的，会给不同的 Token 赋予不同的权重）。

对于一段文本，可将每个 Token 结合起来，采用拼接、求和、逐元素相乘、取平均值或最大值等方法。这样，对于任意文本，都能将其表示为固定大小的 N 维向量。就实现了文本转成数字了。

稍加思考，会发现 One-Hot 编码这种处理方式过于简单，存在诸多固有缺陷：数据维度过高（N 维），向量过于稀疏（过多的 0）。向量之间缺乏语义上的交互（语义鸿沟）：比如“我爱吃苹果”、“我爱用苹果”，前者是水果，后者可能是手机，也可能是电脑等，无法判断。此外，One-Hot 编码难以处理文本中词元的顺序。把句子中的字词调换顺序，比如主语和宾语调换顺序，如“一条狗咬了他”和“他咬了一条狗”，“我喜欢你”和“你喜欢我”句子中的基本字词完全一样，One-Hot 编码会得出相同的表示，实际却是完全不同的意思。

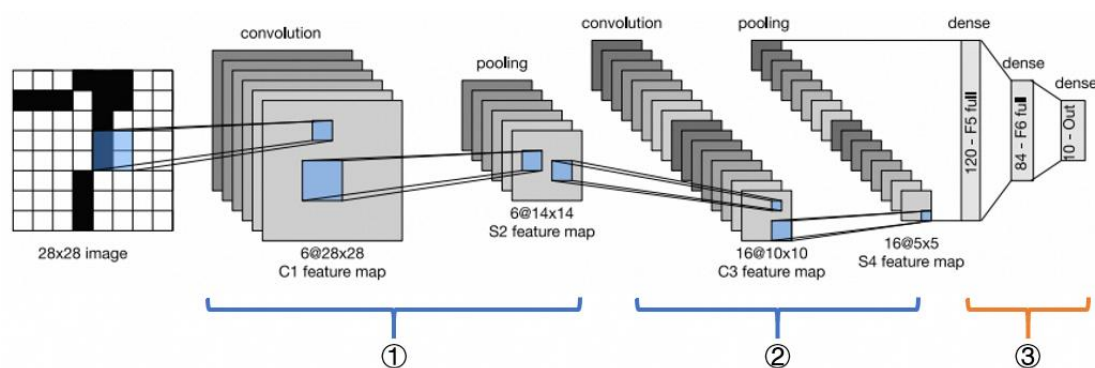
Embedding 算法：其本身就是个 AI 模型，采用深度神经网络等方法，使用样本数据中的上下文关系，训练出一个模型。把 Token 特征固定在维度 D（D 远小于 N，如 256、768 等），这 D 个数字便构成了一个 Token 表示。随着 NLP 的不断发展，Embedding 不仅是针对单个 Token 的向量化，给大语言模型输入任意一句话、数段话，都可以输出一个固定长度的向量化表示。

Embedding 是用一组稠密向量来表示一段文本（可以是字、词、句、段等），得到这个表示后，就可以进一步做相似匹配等任务（如：知识库、QA 机器人、相似内容推荐等）。值得注意的是，Embedding 模型本身就是要训练的，而且通常和应用场景对应的语料强相关，很难有一个适用于全部场景的模型。比如，训练了一个模型，在金融场景好用，在政务场景可能就效果欠佳。在中文场景有用，可能英语场景就

不行了。此外，这个稠密向量本身各维上的数值是不具有可解释性的，比如输出是 256 维的向量，对应 256 个数字来表示对应的输入文本，每个数字都有一些特殊含义，但具体是什么含义，本身是难以解释的。可以简单的把 Embedding 模型当作一个黑盒，输入文本，模型可以输出一个固定大小的向量来表征该输入的文本。

值得一提的是，上文描述的文本转成向量，针对向量的存取，LLM 还带火了一个数据库类型：向量数据库。熟悉 LLM 的人，可能经常会听到知识库、RAG（检索增强生成）等名词，其核心就是将已有的知识文本做拆分，再转向量，形成知识库，针对输入的问题，也转成向量，再到知识库中做向量相似检索，就能匹配出和该问题相关的文本了。

讲到这里，可能对向量化依然有些不太理解。向量化其实无论是 NLP 领域还是图像领域，都是相通的。我们可以再看一个图像领域的例子。回想前文针对图像的深度学习讲解，其中详细拆解了一个经典的卷积神经网络：LeNet，如下：



我们把最后一层（输出层）去掉，输出层的前一层是含 84 个参数的全连接，对应 84 个输出值，其实这 84 个输出值就可以理解为一个向量，用于表征这副输入图像的特征。我们可以训练一个 CNN，如果算法的效果训练的很好，理论上，各个相同的手写数字图像所输出的向量相似度很高（相似度算法如欧式距离、cosine 相似度等，读者可以自行搜索相似度的常用算法原理），而对于不同的手写数字图像所输

出的向量，相似度就不高。这种方法在人脸识别等场景使用广泛。

应用开发新范式：对话即编程

AIGC 对传统 IT 行业带来了系列革新，其中之一是用户输入需求描述（常称为提示词，英文简称 **prompt**），只需将自己的需求或问题描述清楚，模型将直接给出高质量的答案。这将给应用开发提供便利，大大降低应用开发的技术门槛。以 ChatGPT 举例：

```
import openai
import os
from dotenv import load_dotenv, find_dotenv
load_dotenv(find_dotenv()) # read local .env file
openai.api_key = os.getenv('OPENAI_API_KEY')
def get_completion(prompt, model="gpt-3.5-turbo"):
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0, # this is the degree of randomness of the model's
output
    )
    return response.choices[0].message["content"]
prompt = f"""
请帮我写一份中文邮件，告知部门全员在 AIGC 浪潮下，唯有自我变革，才能开创未来。
"""
response = get_completion(prompt)
print(response)
```

如下是 ChatGPT 给出的答案示例：

亲爱的部门全员，

在当前 AIGC（人工智能、物联网、大数据、云计算）浪潮下，我们的工作和生活方式都将发生巨大变革。唯有自我变革，我们才能在这个激动人心的时代中开创未来。

我们需要不断学习和适应新技术，不断提升自己的能力和素质，不断挑战传统的思维方式和模式。只有勇于变革，才能在激烈的竞争中立于不败之地。

让我们共同努力，相互鼓励，共同成长，共同开创未来！

祝好！

我们简单对应用代码做个 **review**，发现十分简洁，真正需要使用者填写的是“**prompt**”部分，也即把自己想要模型解决的问题，把自己的需求描述清楚即可。其余部分的代码，主要是导入对应的模块，按照接口调用的格式填写好相对固定的接口参数。基本上 0 代码基础的人，也能快速完成应用开发。因为其中最为复杂的部分，已交给 AI 模型了。设想下，如果要自己编程实现一个邮件内容生成的程序，那得多么的复杂，门槛何其高。

此外，业界也有大量的文生代码大模型陆续发布，GitHub Copilot 作为其中的典型代表，一经推出就有惊艳表现，甚至一度引发了码农们的恐慌。这就是 AIGC 的魅力之一，使用者仅需把自己的需求，用自然语言描述出来（也即写好 **prompt**），其余的复杂部分交给 AI 模型，由 AI 模型给出高质量答案或代码。惊奇的发现，自然语言也能成为编程语言，跟 AI 模型使用自然语言对话，就实现了复杂的编程，能开发

复杂的应用。可以预见的是，未来编程的门槛会持续降低。

AIGC 是一把双刃剑，在某些方面正在颠覆和影响我们的工作和生活，避免被利刃所伤的最好办法是：成为掌控利刃之人。积极的把它当做提效工具用起来，掌握它，让它成为提高效率的工具和帮手。

LLM 在实际应用中的难点问题

ChatGPT 是火热的 LLM 典型代表，再次掀起了 NLP、AIGC 热潮。本文不去介绍 ChatGPT、Transformer 等的原理。重点从 LLM 的应用视角，分析 LLM 的固有缺陷，并思考解决方法。先看看 LLM 的几大常用功能，比如：

- **智能问答**：应用很广的智能聊天机器人，输入问题，让 LLM 给出答案，多轮互动对话等。
- **摘要/概括**：输入一段话、一篇短文甚至是一篇技术论文，LLM 可以对齐进行摘要和概括，可以精炼出关键内容内容。
- **推断**：比如输入一段话，判断说话的这个人开心还是愤怒。
- **文本转换**：比如翻译，输入一段中文，转换成英文等。
- **文本扩展**：比如现在很多网上用 AIGC 来辅助写故事、文章、邮件、诗词等。用户只需要写好一个故事框架，用 AIGC 去做内容的扩展，写出一个完整的故事等。

ChatGPT 等 LLM，只需要输入提示语句和问题描述（prompt），模型就能给出类人智能的答复，表现惊艳，在智能问答等场景迅速迎来火爆应用。但随着应用的不断深入，大家发现不论是哪家的 LLM，给出的答案，时常出现胡说八道的问题。究其原因，回到 AI 的数据本质，LLM 也是采用大量数据样本做训练得出的，模型一旦训练完成，对于数据样本外的知识一无所知，我们给这个现象取个非常形象的名字：

知识茧房。 LLM 的知识局限在所训练的数据样本中，类似被包裹在一个茧房里面一样。由于缺乏茧房之外的知识，所以面对需要额外知识的问题，则只能根据模型已有逻辑盲目的推测一个答案，问答效果自然很难保障，甚至驴唇不对马嘴。

以企业构建智能问答系统为例，企业的私域知识因涉密、商业价值等原因，不对外开放，无论哪家的 LLM，通常都缺乏企业私域知识。在涉及私域知识的问答中，通用 LLM 很难取得很好的效果。笔者以数据库产品介绍为例，做一个简单测试。采用清华大学的开源 ChatGLM-6B 搭建一套智能问答系统。输入一个简单问题：请介绍阿里云 RDS 数据库。此问题还并非完全是私域知识问题，因为阿里云 RDS 数据库在公网上已发布了大量资料，其他厂商也都有类似产品及资料。看问答表现：



请介绍阿里云rds数据库

AI正在回复:

阿里云RDS(Amazon Relational Database Service)是阿里云提供的一种关系型数据库服务，支持多种类型的关系型数据库，包括MySQL、PostgreSQL、Oracle、SQL Server、Microsoft SQL Server等。

RDS通过云原生架构实现，具有高可用性、可扩展性、安全性、易用性等特点，支持自动扩缩容、备份和恢复等高级功能，可以方便地部署和管理数据库。同时，RDS还提供了丰富的数据库扩展和优化工具，如数据库性能分析工具、数据库安全工具等，帮助用户更好地管理和维护数据库。

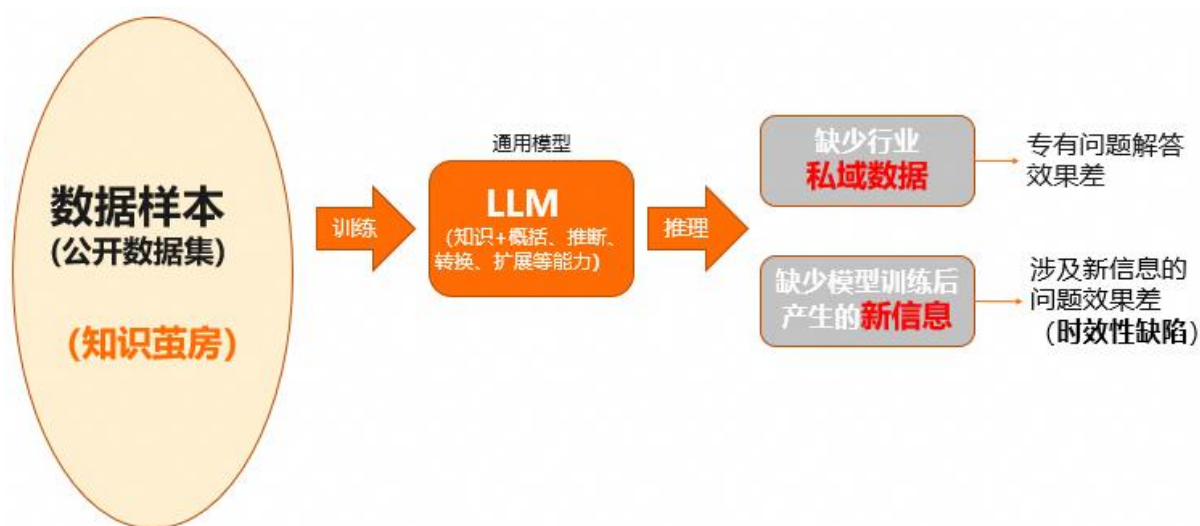
使用阿里云RDS，用户可以通过API接口或者SSH/Telnet等方式连接到数据库，进行数据的存储、管理和查询。同时，阿里云还提供了丰富的数据库管理工具，如DB Manager、DB Designer等，帮助用户更好地管理和维护数据库。

不难发现，答案存在明显错误，如：

- 阿里云 RDS 的产品名称里不应该出现"amazon"。
- 阿里云数据库没有名字为“Oracle”、“DB Manager”和“DB Designer”等服务。

如果输入问题是 Tair、ADB、Lindorm 等阿里云独有的数据库，则给出的答案更是问题百出。也即问题越涉及私域知识，则 LLM 给出的答案通常就更加离谱。读者可自行做个测试。

此外，LLM 的知识茧房问题还表现在，LLM 的知识是有时效性缺陷的，比如在四月份完成了 LLM 训练，到六月份，这两个月产生了大量新信息、新数据、新知识，无论 LLM 的模型参数做多大，哪怕有一天训练了一个超级 LLM，把已有的知识都纳入了，这个问题依然是存在的。因为随着时间的流逝，总是不断有新知识新信息产生，知识和信息本身是难以穷尽的。



LLM “知识茧房” 破解之法：RAG

解决方法通常有如下几种：一、让 LLM 补充目前不具备的知识。如直接做训练，或微调训练。二、将私域知识加工成知识库，结合大模型构建问答系统。

如果用私域数据，训练一个企业独有的 LLM，存在诸多挑战：成本过高，动辄数百

万美金起步，且需要专业技术团队。另一方面，企业的私域数据通常是很有限制的，即便是做微调训练，针对已使用大量语料训练且动辄百亿参数量的大模型，较难出效果。

当下更有效便捷的处理方式是，将企业私域数据通过文本处理、分割、向量化等步骤，加工成企业知识库。结合语义检索和 LLM，可构建一个高效、智能、精准的问答系统。相比传统问答系统，该系统具备更高的智能化和自适应性，不仅可以实现基本的问答功能，还能够对问题进行智能化分析、归纳和推理，提供更加精准和全面的答案，而且具备 LLM 的知识概括、推断、转换、扩展等能力。

下图依然采用清华大学的开源 ChatGLM-6B，仅灌入一篇产品文档构建知识库，相同问题，使用相同的 LLM，答案精准，且更简洁专业。

已加载知识库aliyun zhuanyouyunqiyeban V3.12.0 yunshujukuRDS chanpinjianjie 20210730_FAISS_20230522_203443，请开始提问

请介绍阿里云rds数据库

阿里云RDS(Relational Database Service)是阿里云推出的一种关系型数据库服务，具有高可靠性、可弹性伸缩、高性能、高可用等特点，适用于各种应用场景，例如企业级数据库、金融级数据库等。

阿里云RDS支持MySQL、SQL Server、PolarDB、PostgreSQL等多种数据库引擎，可以方便地创建出适合自己应用场景的数据库实例。同时，阿里云RDS还提供了完整的数据库管理工具，包括数据库备份、恢复、监控、迁移等解决方案，彻底解决了数据库运维的烦恼。

阿里云RDS还支持单可用区部署，保证实例在

▶ 出处 [1] 阿里云 专有云企业版 V3.12.0 云数据库RDS 产品简介 20210730.pdf

▶ 出处 [2] 阿里云 专有云企业版 V3.12.0 云数据库RDS 产品简介 20210730.pdf

知识库是企业私域知识问答系统的核心部分，它包含了企业的专属知识和最新信息，这些通常是大语言模型在训练时无法获取的信息。LLM 结合知识库，可破解“知识茧房”困局。其工作原理是，先使用相似检索的方式，将和问题相关的知识检索出来（可能涉及多条，而且这些知识可能散落在多个文档的不同地方）。然后作为提示

词（prompt）的一部分，输入给 LLM，结合 LLM 的概括、推理、扩展等能力，由 LLM 加工成最终的问题答案，返回给用户。Prompt 模板示例：

```
PROMPT_TEMPLATE = """已知信息：
```

```
{context}
```

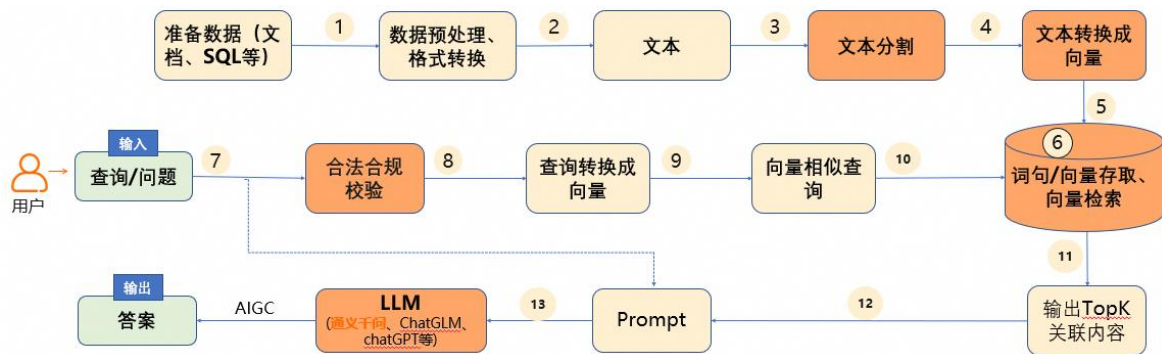
```
根据上述已知信息，简洁和专业的回答问题。如果无法从中得到答案，请说“根据已知信息无法回答该问题”，禁止在答案中添加编造成分，答案请使用中文。 问题是：
```

```
{question}"""
```

其中，问题 **question** 是用户问的问题，而已知信息 **context** 就是知识库中检索出来与该问题相关的信息。也即输入给 LLM 的 **prompt** 中，除了问题之外，还输入了该问题相关的知识，让 LLM 结合这些知识做归纳，给出最终答案。当然这仅仅是一个示例，像 chatGPT 等 LLM，甚至还支持给 LLM 设置角色，给角色定义一些性格，比如扮演一个说话谦虚礼貌的客服，或者模仿某个名人的说话方式等等，各种花样玩法和有意思的应用，读者可以尝试。

知识库+LLM 智能问答系统的构建流程

笔者在阿里云数据库团队供职，曾和阿里云数据库 NoSQL 团队共同规划设计过企业知识库+LLM 产品。企业的原始数据通常以 PDF、docx、文本、网页等格式存在，将这些数据加工成企业知识库，通常涉及如下步骤：



- 1) 数据工程师准备好私域源数据（文档、图片、JSON、SQL 等）。
- 2) 对源数据做预处理，统一转换成文本格式。
- 3) 引入或开发文本分割算法，将文本做句段的分割，分割成大量 text chunks。
- 4) 引入或开发文本转向量算法，将各个 text chunk 转换成对应的向量，即前文介绍的文本转向量的 embedding 过程。
- 5) 将转换后的各个向量，及对应的数据信息存放到向量数据库。
- 6) 部署和维护一套向量数据库。
- 7) 用户输入查询问题。
- 8) 引入或开发合法合规校验系统，校验用户问题的合规性。
- 9) 将查询问题转换成向量。
- 10) 调用向量数据库查询接口，做语义检索（向量相似查询）。

11) 返回检索到的 topK（比如选 top5、top10 等）关联内容。

12) 用户将 topK 内容以及用户问题，填入到 prompt 模板。

13) 输入 prompt 给 LLM，LLM 返回答案。

上述过程，大致列举了知识库和 LLM 的使用和构建过程。若要上线生产，还需结合实际应用场景，构建更多的能力，如：

- 知识库数据新增和更新的处理。需要具备数据订阅，流处理能力，用于知识库的全量数据、增量数据、数据更新的持续加工。
- 文本预处理、文本分割、embedding 等算法的开发和引入，结合业务进行迭代优化。
- 要提高检索知识库的准确率，往往需要结合向量检索和传统基于关键词的文本检索，构建多路召回能力。
- LLM 的选用、部署、维护等。
- CPU/GPU 机器的资源管理，资源扩缩容等。
- 监控告警；系统 HA，灾备；权限管理；数据加密；安全审计等。
- 需要引入数据存储系统，除了存放企业知识库原始语料外，还涉及用户信息、对话记录、配置信息、登录信息、用户行为等数据的处理，要求该系统支持处理结构化和非结构化数据。此外，针对线上系统，为提高性能和并发，还可能涉及数据缓存。
- 为提高问答准确性，需设计用户对答案满意度的反馈机制，并以此提升问答质量。

此外，即便是有了知识库的加持，能提升私域知识问答的效果，但依然面临诸多挑

战，比如：实际使用时，用户问的问题可能是五花八门的。一个问题的答案，可能不仅涉及知识库，可能涉及到查询其他系统，如公司的数据库系统，网上的公开信息等等。可能需要设计一个前置系统，先将问题做个分类，根据不同类型的问题构建不同的处理方式。决定问答质量除了 LLM 本身能力外，还有几大关键因素：

- 数据格式转换成文本：原始语料的格式是多种多样的，笔者访谈的用户中，除了常见的 PDF、docx、文本这些文件外，还有些数据是存放在数据库中的，还有网页方式的，而且经常性变动、更新。有些用户的 PDF 文档中还存在大量图片，图表等，这些使得语料转文本这一步就很有挑战。
- 文本分割算法：文本分割的效果会对后续的分割后的文本转向量，以及相似检索产生直接影响，并最终影响问答质量。也有些开源算法，比如 BERT ([详见](#)) 等。
- 文本转向量 embedding 算法：这个算法尤为关键，因为知识库的核心就是使用向量相似检索的方式来获取知识库中与问题相关的内容。而前文也提到过，很难有一个适用于全部场景的 embedding 模型，该模型的训练本身就充满了技术挑战。比如，较常见的有 text2vec ([详见](#)) 等。笔者在实际用户访谈中，发现有些比较有意思的案例，比如，用户的问题及答案，并非都能加工成一问一答的简洁方式（这种方式下，直接使用相似检索其实反而是最简洁的，效果也很好），大多数情况下，用户的原始数据，都是些各种各样的文档，这种情况下，时不时就发现相似匹配失效的问题，用户问题和检索出来的关联内容不相干，其中原因之一就是，问题和关联内容，转成向量后，向量之间丢失了相似性。关联内容中可能就没有这个问题相关文字的直接描述。

最后，还需要开发一套系统，能将这一系列的数据加工和使用流程做统一编排。对外提供标准接口，供应用开发者便捷使用。有些业界使用较广的开源框架，如 Langchain。

Langchain 简介

Langchain 可以理解为是一个知识向量化的数据加工平台/框架 ([详见](#))，集成了文

档格式转换、分割、转向量等系列流程, 实现了知识库+LLM 的完整方案。Langchain 实现的功能:

- 1) 支持用户导入私域知识 (如.txt、.md、.docx、.pdf 等多种文档格式)。
- 2) 知识库自动加工: 文本分割、向量转换、向量存取等由平台自动完成。
- 3) 支持全量数据持续导入 (每一篇文档为一个知识库, 问答时支持选择知识库)。暂不支持增量, 暂不支持文档内的数据更新。
- 4) 系统自动完成查询问题向量转换、语义检索。并将问题和语义检索得到的 topK 关联内容, 填入 prompt, 调用 LLM 得到答案, 返回给用户。
- 5) 目前兼容的 embedding 算法和 LLM, 可查看代码, 配置文件中的如下变量:

```
embedding_model_dict = {  
    "ernie-tiny": "nghuyong/ernie-3.0-nano-zh",  
    "ernie-base": "nghuyong/ernie-3.0-base-zh",  
    "text2vec": "GanymedeNil/text2vec-large-chinese"}  
llm_model_dict: chatglm-6b-int4-qe、chatglm-6b-int4、chatglm-6b-int8、  
chatglm-6b、chatyuan、moss。
```

其中 ChatGLM-6B 是清华大学的开源模型, 可以处理自然语言文本数据, 并能进行语言生成, 在中国开源社区用的较广。

- 6) 向量检索、存储: FAISS。

使用 Langchain+chatGLM 构建知识库方案, 建议读者可以参考“系列动手实践”中对应的实战指导书, 自行安装使用一遍。操作界面示例:



Langchain 的优点：

- 实现了文档导入、语料分割、向量转换、向量存取、向量检索等的服务化流程。用户可以快速搭建一个测试环境。
- 是一套开源框架，可自己开发集成自有的算法和 LLM，实现一些定制化的功能。
- 缺点：
 - 比较适合早期的调研、测试使用。集成的 embedding 算法、LLM 基本上都是些开源产品，效果难以保障。用的多了，会发现能用但不好用。
 - 组件非常多，且都是些开源组件，出现问题难定位，效果调优困难，维护困难。
 - 目前只支持文档全量导入，一个文档对应一个知识库。不支持一个知识库包含多个文本，不支持知识库的增量数据导入，以及数据更新等。
 - 向量数据库使用的是开源 FAISS，难以直接使用在生产环境。

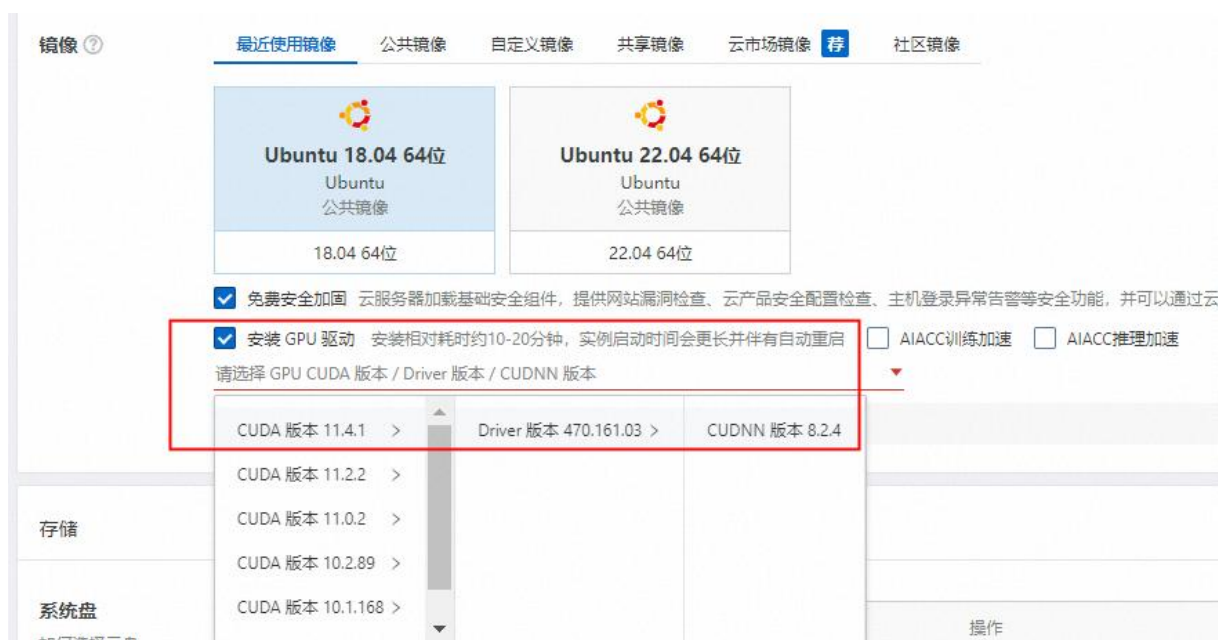
如果要进一步了解 LLM、Langchain+大模型构建带私域知识能力的智能问答，强烈建议读者亲手搭建一套。尤其有了公有云，在云上搭建这些系统现在变的非常简便，仅需一两个小时就能搭建完成。详细的搭建指导书已经放在了系列动手实践章节了。

系列动手实践

在云上从 0 开始搭建 Langchain-ChatGLM LLM 环境

以在 Ubuntu 18.04 64 位系统搭建 LLM 环境，安装《[langchain-ChatGLM](#)》为例：

- 1) 创建 GPU 实例（显存 8GB 以上，磁盘至少 40G），并在实例创建时选择安装 cuda 11.4。设置用户名及登录密码。



- 2) 设置安全组配置，配置出方向端口 22，并在源 IP 中加入本机 IP。
- 3) Ssh 到云 ECS。如果是 root 登录，系统会提示正在安装 CUDA，待安装完毕，查看 GPU 详情（若命令无法正常运行，则说明 CUDA 没安装好）：`nvidia-smi`

```

root@aigc:~# nvidia-smi
Thu May 18 13:34:31 2023

+-----+
| NVIDIA-SMI 470.161.03    Driver Version: 470.161.03    CUDA Version: 11.4    |
+-----+-----+
| GPU   Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|=====+=====+
|  0   Tesla P100-PCIE...    On         | 00000000:00:08.0 Off |             0        |
| N/A   30C    P0      26W / 250W | 0MiB / 16280MiB |      0%    Default  |
|=====+=====+
|
+-----+
| Processes:                                                       GPU Memory |
|  GPU   GI    CI          PID    Type   Process name                  Usage   |
|=====+=====+
| No running processes found                                     |
+-----+

```

4) 在 ECS 上安装相关软件包：

a. `sudo apt-get update`

b. 安装 gcc 编译器(可选，系统中通常已有)：`sudo apt install build-essential`

c. 安装 Python，以 3.8 为例：`sudo apt install python3.8`

d. 安装 miniconda：

(1) 下载安装包：注意 miniconda 包需选择对应 python 版本的包

`wget`

`https://repo.anaconda.com/miniconda/Miniconda3-py38_23.3.1-0-Linux-x86_64.sh`

(2) 运行安装脚本，并初始化：`bash`

Miniconda3-py38_23.3.1-0-Linux-x86_64.sh

(3) （可在 2 中完成）初始化终端 Shell，以便运行 conda。

```
~/miniconda3/bin/conda init
```

(4) 初始化完成后，运行 bash 命令，即可进入 conda 环境：bash

(5) 创建一个新的环境（可选）：conda create --name aigc python=3.8
-y

(6) 激活 aigc 环境（可选）：conda activate aigc

5) 安装 git：sudo apt install git

6) git clone langchain-ChatGLM，并安装软件依赖：

```
git clone https://github.com/imClumsyPanda/langchain-ChatGLM.git
cd langchain-ChatGLM && pip install -r requirements.txt
pip install fastapi uvicorn
conda install nltk          # langchain 依赖的语言工具包
```

7) 将 chatglm-6b 模型及 Embedding 模型，下载到本地（可从本地加载模型）：

安装 git-lfs：apt install git-lfs

git-lfs clone <https://huggingface.co/THUDM/chatglm-6b>

下载 Embedding 模型

```
$ git clone https://huggingface.co/GanymedeNil/text2vec-large-chinese  
/your_path/text2vec
```

模型需要更新时，可打开模型所在文件夹后拉取最新模型文件/代码

```
$ git lfs pull
```

8) 更新 config/model_config.py 中的模型路径为本地路径。

```
llm_model_dict 变量中 "chatglm-6b" : "pretrained_model_name":  
"/home/fsnode/chatglm-6b"
```

```
embedding_model_dict 变量中 "text2vec" : "text2vec":  
"/home/fsnode/text2vec-large-chinese"
```

9) 导入自有的文档到 ecs，目前主要支持 md、txt、pdf、jpg 格式。

10) 运行 CLI 或 web DEMO:

```
cli_demo: python cli_demo.py
```

webui: 如需要公网访问，还需要将 webui.py 中修改: share=True

a. 安装 gradio : pip install gradio

b. 在 ECS 实例安全组的入方向添加安全组规则，并放行 8501 端口（默认的

公网服务端口，也可以自定义）。



c. 在 ECS 上运行脚本：python webui.py

d. 脚本运行成功后，会展示生成的公网链接。在本机浏览器中输入该链接，就可以使用了。

已加载知识库aliyun zhuanyouyunqiyeban V3.12.0 yunshujukuRDS chanpinjianjie 20210730_FAISS_20230522_203443， 请开始提问

请介绍阿里云rds数据库

阿里云RDS(Relational Database Service)是阿里云推出的一种关系型数据库服务，具有高可靠性、可弹性伸缩、高性能、高可用等特点，适用于各种应用场景，例如企业级数据库、金融级数据库等。

阿里云RDS支持MySQL、SQL Server、PolarDB、PostgreSQL等多种数据库引擎，可以方便地创建出适合自己应用场景的数据库实例。同时，阿里云RDS还提供了完整的数据库管理工具，包括数据库备份、恢复、监控、迁移等解决方案，彻底解决了数据库运维的烦恼。

阿里云RDS还支持单可用区部署，保证实例在

- ▶ 出处 [1] 阿里云 专有云企业版 V3.12.0 云数据库RDS 产品简介 20210730.pdf
- ▶ 出处 [2] 阿里云 专有云企业版 V3.12.0 云数据库RDS 产品简介 20210730.pdf
- ▶ 出处 [3] 阿里云 专有云企业版 V3.12.0 云数据库RDS 产品简介 20210730.pdf
- ▶ 出处 [4] 阿里云 专有云企业版 V3.12.0 云数据库RDS 产品简介 20210730.pdf

此外，如果不想修改安全组端口，也无需公网访问。可以把模型的本地服务端口 7860

直接映射到自己的本地便携上，登录使用：

A、本地便携机上执行如下命令，将云 ECS 的 7860 端口映射到本地（IP 及用户名填实际的）：

```
ssh -L7860:localhost:7860 ecs-user@ecs 公网 IP
```

B、在本地浏览器登录 web 界面：<http://127.0.0.1:7860>

从 0 开始搭建 LLM+知识库智能问答钉钉机器人

以阿里云为例，描述在云上从 0 开始构建个人/企业专属，具备私域知识库+LLM 智能问答钉钉机器人。知识库技术方案使用了 Lindorm AI 数据服务平台知识库能力，LLM 使用了开源 ChatGLM2-6B，读者也可以更换成自己的知识库产品。

一、开通和配置 ECS

- 1) 创建 ECS 实例（以 Ubuntu 系统为例），作为登录数据库的客户端。设置 ECS 用户名及登录密码（使用 ecs-user 登录，如果使用 root 安装，后续部分安装包会提示有权限风险）。
- 2) 设置安全组配置，配置出方向端口 22，并在源 IP 中加入本机 IP。



3) 本机 ssh 到云 ECS。

二、构建自己的知识库+LLM

- 1) 本例以阿里云 Lindorm 数据库提供的知识库方案为例，读者也可以更换成自己的方案。根据产品[指导书](#)，开通 Lindorm 知识库+LLM 实例，并根据指导书导入或 SQL 方式写入自己的知识文档，完成知识库的构建。该方案安装部署 LLM 非常简便，仅需一条简单的 SQL 命令即可完成安装。
- 2) 将 ECS 客户端 IP 加入到 Lindorm 数据库白名单，确保 ECS 可以正常访问 Lindorm 数据库。可以在 ECS 上安装 Lindorm CLI [客户端](#)进行测试。
- 3) 在 ECS 上安装 phoenixdb（用于连接数据库）：`pip install phoenixdb==1.2.0`
- 4) 注意：有些系统版本在安装时，可能会报错 “subprocess.CalledProcessError: Command 'krb5-config --libs gssapi' returned non-zero exit status 127”，原因是缺乏安装依赖，需执行如下命令后再安装：`sudo apt-get -y install gcc libkrb5-dev`

三、申请钉钉机器人

- 1) 在钉钉上申请钉钉机器人：选择一个群，点击右上角的“群设置”按钮，选择“机器人”进行添加。



- 2) 根据文档说明，完成机器人申请：



- 3) 机器人申请成功后，完成基础设置。其中消息接受模式选择“HTTP 模式”；配置消息接受地址：将 IP 设置为前面申请的 ECS 大网 IP，服务端口号设置为自定义好的端口号(本文以 5000 为例,后面的钉钉 post 程序中 also 需配置为该端口)。



四、编写钉钉机器人 post 程序

说明：该程序运行在 ECS 上，用于响应钉钉机器人从钉钉上获取的信息，进行应答。

- 1) 在 ECS 上安装 dingtalkchatbot、fastapi、uvicorn：

```
pip install dingtalkchatbot fastapi uvicorn
```
- 2) 编写钉钉机器人服务端程序（Python 脚本）：

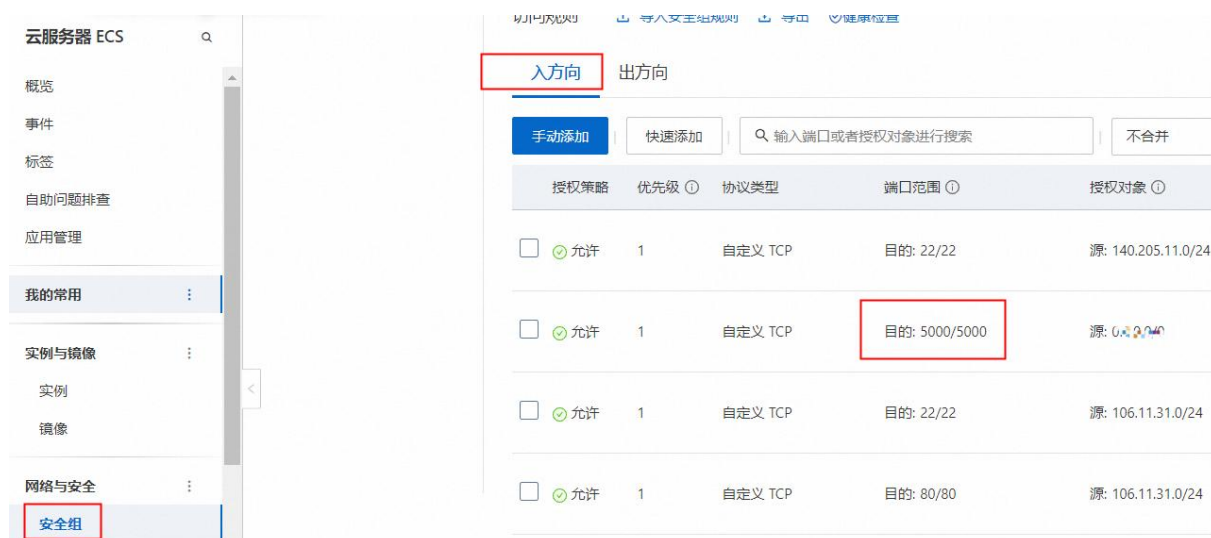
https://code.alibaba-inc.com/pangshaoting.pst/PST/blob/master/dingtalk4li_ndorm.py

代码详解：

Lindorm 提供了类 JDBC 的连接和使用方式,使用起来非常简便。对于知识库和 LLM 的调用,直接使用 SQL 查询方式即可返回 LLM 的问答结果,将返回的内容进行格式化整理即可:

```
sql_select = "select ai_infer('%s', '%s')" % (LLM_MODEL, prompt)
statement.execute(sql_select)
response = statement.fetchall()
msg = response[0][0]      #查询返回是 list 格式, 需要取 list[0][0],转换为 str 格式
```

- 3) 配置服务端口 (本文设置为 5000), 配置安全组, 放通入方向 IP。初始调试的时候, 可以配置为 0.0.0.0/0, 以免因为网络配置导致程序不通。调测成功后再修改为钉钉机器人服务器对应的 IP (见步骤 5)。



4) 运行脚本: `nohup python3 dingtalkpost.py &`

5) 安全组入方向 IP 获取方法: 上述步骤使用 `nohup` 方式运行脚本后, 会在该脚本所在目录下生成一个 “`nohup.out`” 日志文件, 该文件内会记录调用钉钉机器人的源端 IP, 比如:

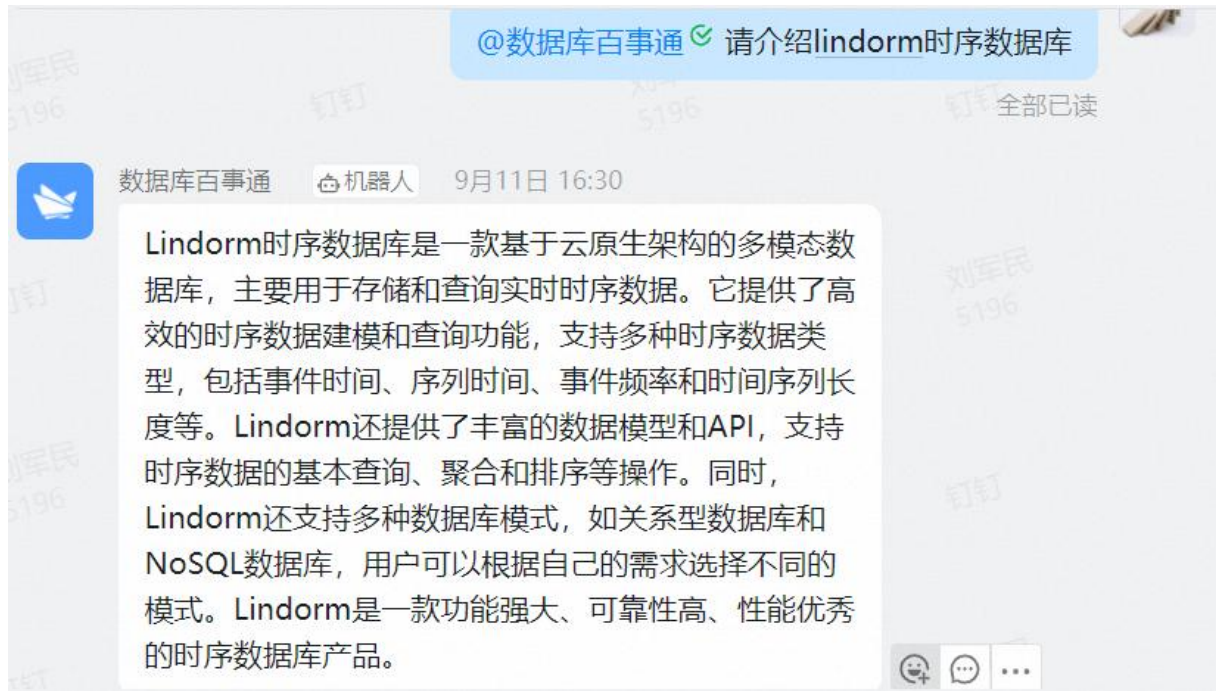
INFO: 59.82.84.108:15922 - "POST /generate HTTP/1.1" 200 OK

持续使用一段时间后, 通过分析该日志文件中的源端 IP, 即可获取到调用钉钉机器人的 IP 信息, 从而配置安全组。如上示例, 可配置 `59.82.0.0/16`, 见:



五、效果展示

在钉钉群中添加所申请的机器人, 并@该机器人进行智能问答。则可以回答涉及私域知识的问题。



讲解视频

为进一步方便学习，录制了讲解视频，详见：

https://www.bilibili.com/video/BV1BRN3eYEGp/?vd_source=0c5a39664e27af522bfcd1e74b9780a8

如何把 AI 技术融入到产品

通过前文的学习，相信大家对 AI 已经有一个初步的理性认知了。现在假设我们是一个产品经理，想用 AI 去干一些事情，发掘一些应用场景，将 AI 融入到自己的产品中，那该从哪些方面去思考呢？或许可以先看看产业界、学术界在干什么，并结合 AI 的三要素：数据、算法、算力等视角评估分析应用场景，寻找启发。

产业界在干什么

当不清楚 AI 具体能做什么的时候，可以先看看 AI 产业界在干什么。产业界在持续探索新领域，有些盐碱地已经洗过了，可以做一个参考。比如《中国人工智能行业全景图谱》等资料，会持续更新中国 AI 行业的一些新进展、主要的厂商、投资情况等信息。围绕 AI 算法、算力、数据，以及应用等几大方面对厂商进行了分类：



学术界在干什么

还有一个角度，AI 学术界在研究什么。如果一个领域，都没有任何人去研究，那就可能就得仔细斟酌，是不是选错方向了。这方面可以读一读权威机构、高校发布的一

些相关资料。比如，斯坦福大学发布的 AI 指数报告：《Artificial Intelligence Index Report 2023(斯坦福 2023 年 [AI 指数报告](#))》，[中文链接](#)。该报告覆盖 AI 的多个方面，是一个总括性的报告。由斯坦福李飞飞教授主导发起，每年发布一份，2017 首发。2023 报告中的部分观点如下：

- 1) 2014 年之前，大多数重要的机器学习模型都是由学术界发布的。而此后，工业界开始超越学术界。2022 年，工业界生产了 32 个重要的机器学习模型，而学术界只生产了 3 个。
- 2) AI 学术界主要研究领域：模式识别、机器学习、机器视觉、算法、数据挖掘、NLP、控制理论、人机接口等。

Number of AI Publications by Field of Study (Excluding Other AI), 2010–21

Source: Center for Security and Emerging Technology, 2022 | Chart: 2023 AI Index Report

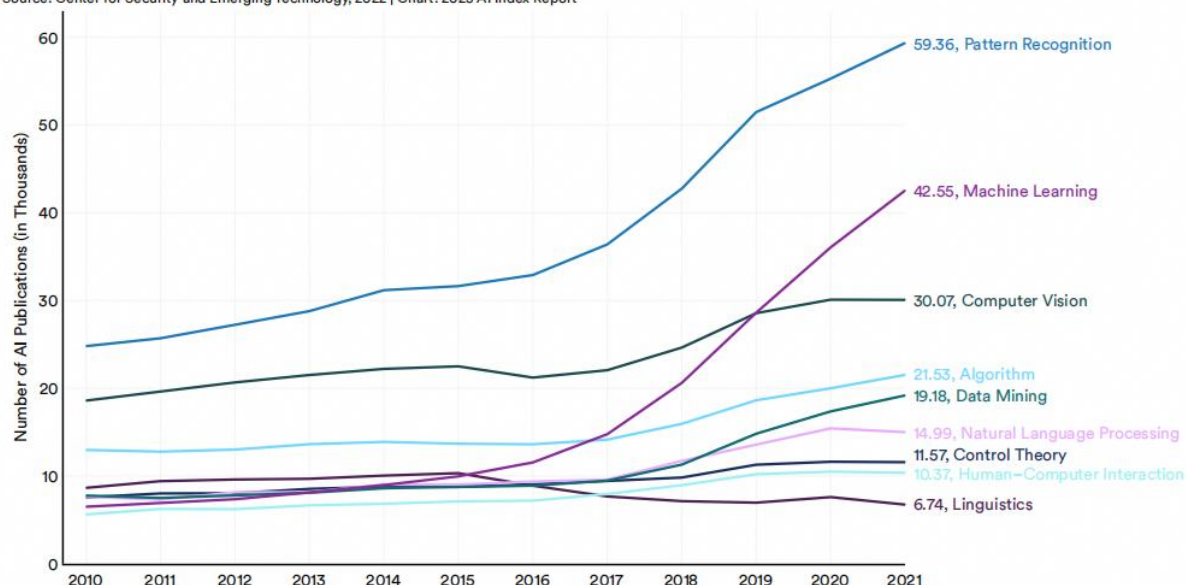
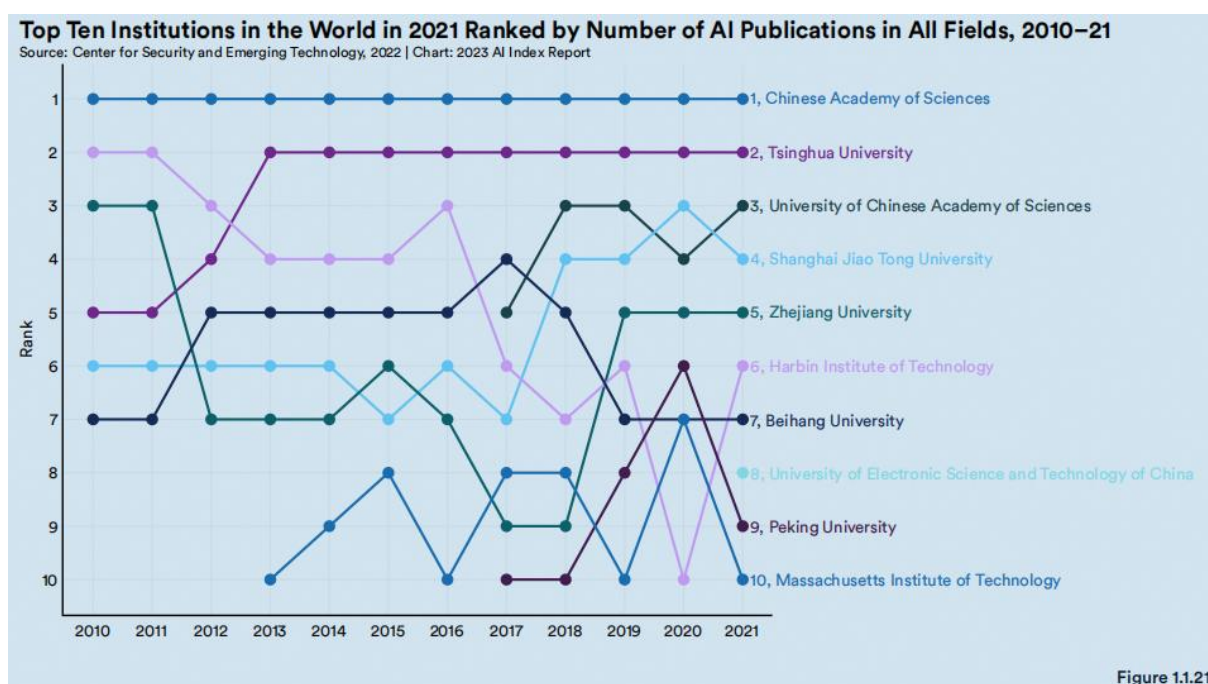


Figure 1.1.3

- 3) AI 论文发表数量前十的机构，中国占了前 9 名（麻省理工第 10），主要有：中科院、清华、中科大、上海交大、浙大等。特别强调，尤其是机器视觉领域，前十发版机构均来自中国，排名第一的中科院达 562 篇。



AI 领域的主要发展依然集中在**机器视觉和语言**，细分领域见下：

概述	72
章强调了	73
2.1 2022 年的新事物:时间线	74
2.2 计算机视觉-图像	81
图像分类	81
ImageNet	81
人脸检测与识别	82
国家标准与技术研究所人脸识别供应商测试 (FRVT)	83
Deepfake 检测	84
Celeb-DF	84 年
人体姿态估计	85
MPII	85
语义分割	86
Cityscapes 挑战，像素级语义标注任务	86
医学图像分割	87
Kvasir-SEG	87 年
物体检测	88
背景中的常见物体(COCO)	88
图像生成	89
CIFAR-10 和 STL-10	89

视觉推理	92
视觉问答(VQA)挑战	92
视觉常识推理(VCR)	95
2.3 计算机视觉-视频	96
活动识别	96
Kinetics-400, Kinetics-600, Kinetics-700	96
2.4 语言	99
英语理解能力	99
超强力胶水	99 年
需要逻辑推理的阅读理解数据集(ReClor)	100
arXiv 和 PubMed	104

自然语言推理	105	2.7 硬件	115
溯因自然语言推理(aNLI)	105	MLPerf 训练时间	115
情绪分析	106	MLPerf 推理	117
SST-5 细粒度分类	106	gpu 的发展趋势	118
多任务语言理解	107	2.8 环境	120
大规模多任务语言理解(MMLU)	107	选择大型语言模型的环境影响	120
机器翻译(MT)	108	利用 AI 优化能源使用	122
商用 MT 系统数量	108	2.9 科学 AI	123
2.5 讲话	109	通过学习等离子体控制加速聚变科学	123
语音识别	109	利用字母数发现矩阵运算的新算法	123
VoxCeleb	109	用深度强化学习设计算术电路	124
耳语	110	用生成式 AI 解锁从头抗体设计	124
2.6 强化学习	112	访问公共数据	
强化学习环境	112		
Procgen	112		
基准饱和	114		

更多详细信息可以查阅原版资料。此外，国内高校也有一些学术资料发布，如清华大学发布了《AIGC 发展研究报告 1.0 版》等系列资料。更进一步，想熟悉具体领域的最新学术进展，也可以阅读该领域的相关论文。

AI 落地三问

从前面的学习中，可以看到 AI 的几大要素：数据、算法、算力，以及应用场景分析。可以从这些要素入手，先问自己三个问题：

问题一：场景。你想用 AI 解决什么问题？AI 不是万能的。即便强如大模型，依然有大量事情干不了或者效果差，更何况一些传统的机器学习了。一定要想清楚要解决什么问题，这个问题用 AI 靠不靠谱？是不是能够从已有的数据里面找到规律。可以

粗浅的认为 AI 是以统计学、概率论等为基础理论，提取数据中的规律，进而对新的输入做推理。如果数据样本中本身就缺乏关键规律，那想用 AI 来解决这个问题，这条路大概率一开始就走错了。

问题二：数据。目前绝大多数 AI 都需要基于已有的数据样本做训练，训练过程可以理解为提取数据中的规律的过程。第二个问题依然和数据强相关：用于训练的数据样本在哪里？有没有高质量的标注数据？如果数据样本本身就是错误的，或者根本就没什么有效数据，如果不解决高质量数据样本问题，AI 也是很难起作用的。

问题三：算法&算力。研发团队和资源在哪里？算法主要靠研发团队，算法也是决定最终应用效果的核心因素，这就要求有一个高水准的 AI 研发团队。此外，还需要考虑是否有足够的算力资源。

附录

作者简介

刘军民：先后供职于华为、阿里云。目前在阿里云数据库事业部担任产品经理，潜心数据库产品规划和设计。同时，对 AI 等有强烈好奇心、求知欲，喜欢钻研新技术。读者若发现文中刊误，或有其他建议，欢迎发送至邮箱：careerlife@qq.com。

参考资料

Coursera：吴恩达讲《机器学习》，网址：<https://www.coursera.org>

《动手学深度学习 PyTorch 版》：该书有纸质印刷版，也有配套的电子版：
https://zh.d2l.ai/chapter_introduction/index.html

LeNet 参考：

https://github.com/d2l-ai/d2l-zh/blob/master/chapter_convolutional-neural-networks/lenet.md

nn.Conv2D：

<https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d>

清华大学：《AIGC 发展研究报告 1.0 版》

Artificial Intelligence Index Report 2023(斯坦福 2023 年 [AI 指数报告](#))，中文

《中国人工智能行业全景图谱》

Langchain+ChatGLM:

<https://github.com/imClumsyPanda/langchain-ChatGLM>

BERT 文本分割:

https://modelscope.cn/models/iic/nlp_bert_document-segmentation_chinese-base/summary

text2vec: <https://github.com/shibing624/text2vec>

Langchain-Chatchat

:

<https://github.com/chatchat-space/Langchain-Chatchat>