

看图学大模型

看图学

2024 年 3 月 17 日

当前版本为 v0.20240317, 还在更新中, 目前维持每周更新。

本电子书目前进度完成不到 10%, 可以关注下方公众号回复”看图学大模型”来获取最新版。



看图学

微信扫描二维码, 关注我的公众号

目前 PDF 排版等还略有问题，Latex 还需要略微调整。

目录

1 Chapter 1: 大模型发展史	2
1.1 AI 的起源	2
1.2 符号派、链接派，相爱又相杀	18
1.3 从 one-hot 到 ChatGPT	18
2 Chapter 2: 主流大模型的模型架构和细节分析	19
2.1 Position Embedding	19
3 Chapter 3: 大语言模型 pipeline	25
3.1 Stage 1: Pretrain	25
3.2 Stage 2: SFT	25
3.3 Stage 3: Alignment	25
3.4 Stage 0: 数据处理	25
4 Chapter 4: 大语言模型训练	25
4.1 显卡和模型训练基础知识	25
4.2 多卡并行训练	25
4.3 当前流行训练框架	25
5 Chapter 5: 大语言模型推理	26
5.1 KV Cache	26
6 Chapter 6: MOE, 多模态等	30
7 Chapter 7: 大语言模型评估	30
8 Chapter 8: 大语言模型应用	30
8.1 Prompt Engineering	30
8.2 Agent	30

1 Chapter 1: 大模型发展史

1.1 AI 的起源

翻一翻历史其实是蛮有意思的一件事情，我们可以看到很多现在与历史上一些相似的瞬间，仿佛 Yesterday Once More。

AI 的发展到今天差不多也有 70 多年的历史了，让我们来回顾一下这段跌宕起伏的历史。

1.1.1 机器能不能思考?

1900 年的科幻小说《绿野仙踪》里，有个铁皮人，这可能是早期人们对人工智能机器人的一种幻想。



50 年后，图灵认为人类通过信息和推理来解决问题和做出决策，那机器能不能做同样的事情？1950 年，图灵发表了论文《计算机械和智能》，在里面讨论了如何构建智能机器以及如何测试机器的智能，也就是大家都熟知的“图灵实验”，开始把幻想映射到现实。

但是图灵提出设想后并没有做出一台智能机器。原因有很多，但是主要的原因可能有两个。

一个是图灵在研究人工智能的时候，冯诺依曼正在研究计算机体系结构，所以那个时候计算机还没有使用冯诺依曼结构，自然也就缺乏一个智能体的关键结构：记忆。那个时候的计算机只存储数据，并不存储命令。也就是说你可以告诉计算机要干啥，但是它并没有记住自己干了什么。是不是感觉跟现在的大模型有点像？

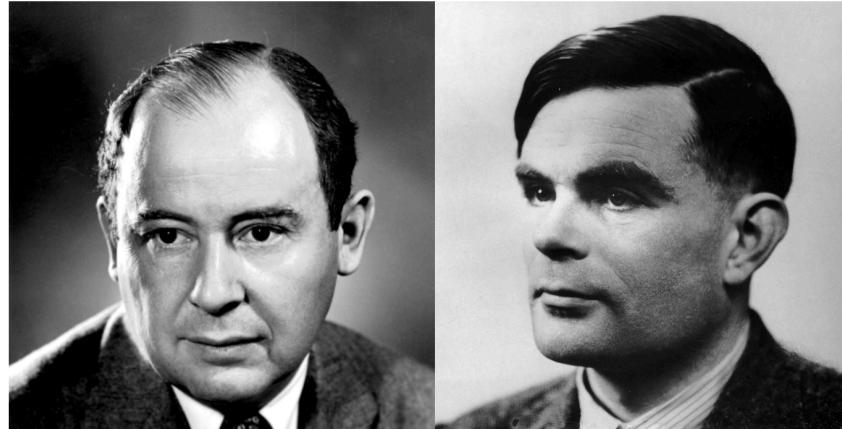
再一个，那个时候计算机是真的贵，当时租用一台计算机要花 20 万美元，只有高科技公司和大学才有机会使用。现在大模型训练成本也高的离谱。

相似瞬间

历史	现在	启发
当时的计算机只存储了数据，没有存储指令，没有记忆	大模型也可以认为是存储了数据，目前也没有记忆	Agent 目前是外挂记忆，后续大模型会不会内置一个指令存储单元
成本高，计算机租金每月 20 万美元	大模型训练成本高，GPT3 最开始训练一次几千万美元	成本在当时是瓶颈，摩尔定律会解决这个瓶颈

这段历史中出现了两个大佬。

冯诺依曼被人们称作“计算机之父”，“博弈论之父”。图灵被人们称作“计算机科学之父”，“人工智能之父”。



冯诺依曼
计算机之父
博弈论之父

艾伦图灵
“计算机科学之父”
“人工智能之父”

冯诺依曼比图灵大 10 岁，这两个都是天才级别的人物。

1.1.2 达特矛斯会议：AI 一大

目前大家普遍认为是 AI 的起源是 1956 年的达特矛斯会议。因为这次会议上，人工智能研究的先行者聚集在了一起，搞了个为期 2 个月的头脑风暴。第一次提出了“人工智能”这个词，虽然后来考证可能是麦卡锡把维纳的 Cybernetics 换了个名字。

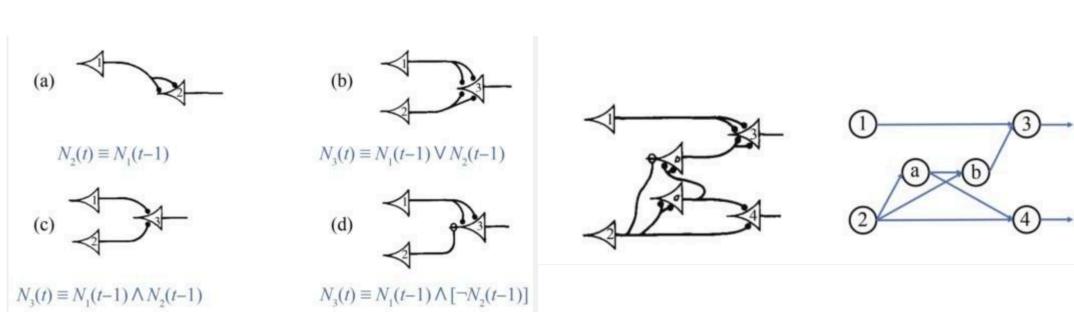
当时参加会议的人，大都是某学科的开山鼻祖，混的差一点的也都是几年后的图灵奖获得者。

这帮人之间也有千丝万缕的关系，这也告诉我们一个道理，你要想成为大佬，先要接近大佬。

我们按时间线来讲一下参加这次会议的人是怎么凑到一起的。

1936 年，图灵在论文《论可计算数及其在判定问题上的应用》中，提出了图灵机理论。

1943 年，神经生理学家沃伦·麦卡洛克 (Warren McCulloch) 和数学家沃尔特·皮茨 (Walter Pitts) 参考了图灵机里面的一些思想，开发了神经元基本模型，将神经网络的概念引入计算机领域，被认为人工智能第一项工作。¹



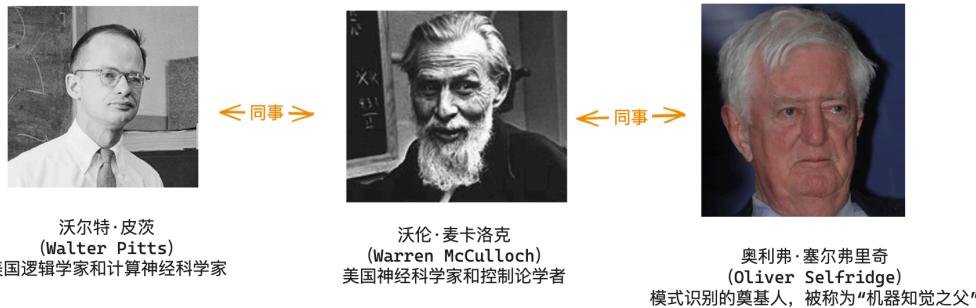
¹<https://www.cs.cmu.edu/~epxing/Class/10715/reading/McCulloch.and.Pitts.pdf>

上图为论文中的神经元结构，右侧的图甚至有点像后来的 GRU 和 LSTM。

皮茨是个天才，12岁时，3天读完了罗素的《数学原理》，还给罗素写了封信，指出了其中的几处错误。这个罗素就是提出“罗素悖论”的那个，看到信后邀请他去剑桥读他的研究生，根本没想到是个12岁的小屁孩，结果这个小屁孩没钱，十动然拒，罗素痛失天才弟子。皮茨由于都是自学成才，高中都没毕业，没学位也让他四处碰壁。但是他实在过于优秀，很多大佬给他站台。曾跟随芝加哥大学的逻辑学家卡尔纳普（Rudolf Carnap）学习，卡尔纳普无论是在学习还是生活都给了皮茨很大的帮助。后来皮茨被沃伦·麦卡洛克（Warren McCulloch）推荐给控制论之父诺伯特·维纳（Norbert Wiener），维纳发现了皮茨的潜力，直接收了他作为博士生，要注意皮茨连高中和本科学位都没有啊。沃伦·麦卡洛克（Warren McCulloch）本来在芝加哥大学好好的当教授，可能是比较崇拜维纳，辞去教授去MIT当了个研究员，就是为了和维纳一起工作。

1945年以后，奥利弗·塞尔弗里奇（Oliver Selfridge）在MIT读研究生，他的导师就是上面提到的诺伯特·维纳（Norbert Wiener）。这个时候他与沃尔特·皮茨（Walter Pitts）和沃伦·麦卡洛克（Warren McCulloch）在一起工作，研究人类智能。²

q



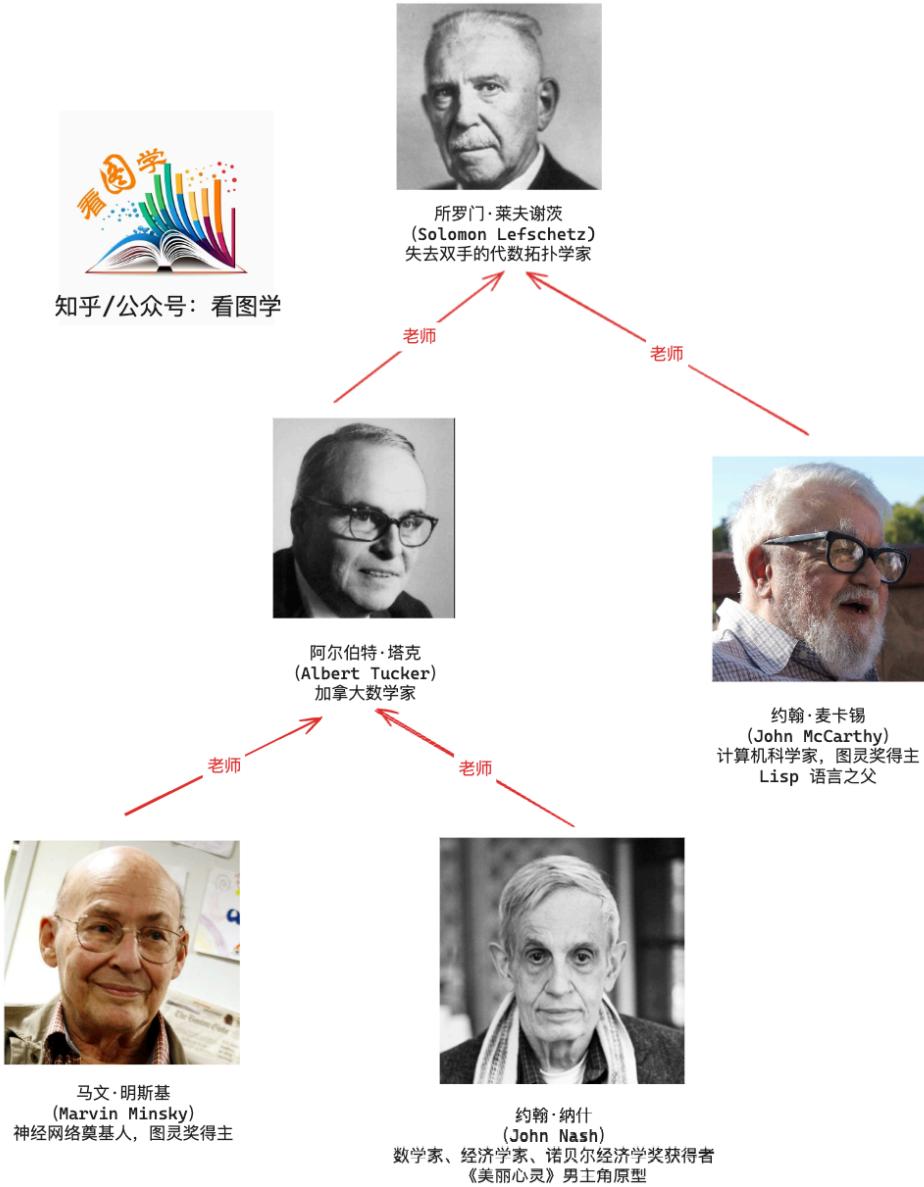
1951年，塞尔弗里奇加入了麻省理工学院的林肯实验室，很快成为一个研究通信技术、模式识别、指挥和控制以及交互式计算的团队的组长。折腾一些机器识别摩尔斯电码的早期工作。塞尔弗里奇可以认为模式识别的奠基人，他写了第一个可工作的AI程序，后来被人们称为“机器知觉之父”。那一年的夏天，他招了伙计，这个人叫马文·明斯基（Marvin Minsky）。

²<https://vineyardgazette.com/obituaries/2009/01/29/oliver-selfridge-computer-pioneer-loved-chappy>



马文·明斯基 (Marvin Minsky) 呢，在这一年搞出了第一部能自我学习的人工神经网络机器，SNARC。他奠定了人工神经网络的研究基础，然而造化弄人，在不久的将来他却亲自给人工神经网络判了死刑，导致很长一段时间神经网络变成了冷板凳。

马文·明斯基的导师是塔克 (Tucker)，塔克还有个学生就是电影《美丽心灵》的主角约翰纳什 (John Nash)。所以纳什是明斯基的师兄。塔克的导师是失去双手的代数拓扑学家所罗门·莱夫谢茨 (Lefschetz)。莱夫谢茨还有个学生，叫约翰·麦卡锡 (John McCarthy)。从师承上讲约翰·麦卡锡是马文·明斯基和纳什的师叔，算是老相识。



马文·明斯基的博士论文是神经网络，但是他当时是数学系的学生，博士委员会看了就有点迷糊，神经网络算是数学么？当时冯诺依曼也是博士委员会的，说虽然现在可能不算，但是不久之后就是数学了。所以马文·明斯基的博士答辩通过，冯诺依曼是大恩人。

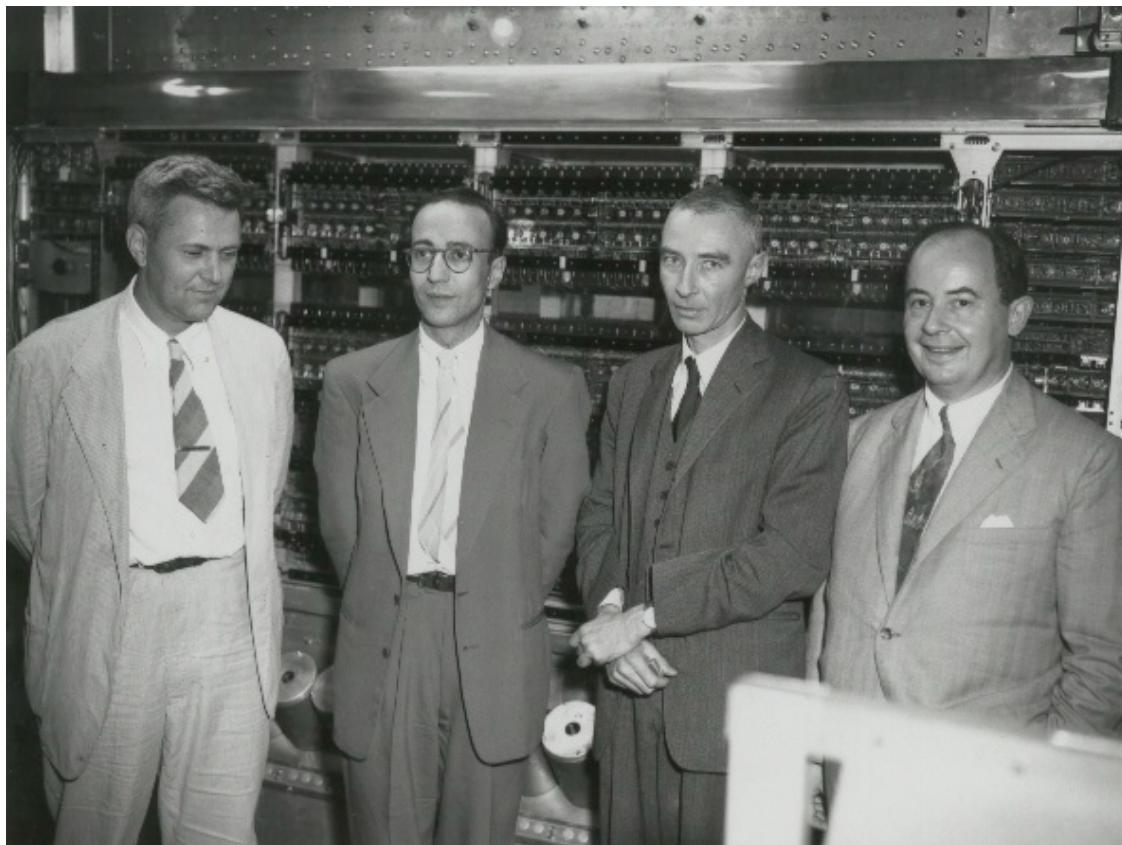
约翰·麦卡锡是个共产主义战士。他是达特茅斯会议的发起人。他是怎么到达特茅斯学院的呢？当时达特茅斯数学系的系主任是克门尼 (Kemeny)。克门尼是图灵的师弟，都师从普林斯顿逻辑学家丘奇 (Church)，他战时和物理学家费曼一起工作，还一度当过爱因斯坦的数学助理，就问牛不牛逼。当时达特茅斯数学系一下子退休了 4 个教授，克门尼压力山大，人都走光了，工作还怎么展开。就去普林斯顿找了 4 个博士，其中一个就是麦卡锡。克门尼和麦卡锡合作还是很愉快的，一起琢磨出了计算机的分时系统，克门尼创造了 Basic 编程语言，而麦卡锡则创造了 Lisp。Lisp 现在可能很多人没听说，但是 Lisp 的设计理念可以说是影响了后续所有的编程语言，我现在因为使用 Emacs 还偶尔用一下。

1953年夏天，麦卡锡和明斯基都在贝尔实验室为克劳德·香农（Claude Shannon）打工。香农是信息论的创始人，和图灵、冯诺依曼是同一个级别的，就不多介绍了。这里有个段子就是香农作为学生的时候，天天问诺伯特·维纳（Norbert Wiener）问题，借鉴他的思想。导致后来维纳拒绝跟香农见面，说：“香农就是来挖我脑浆子的”。后来被认为是创立了信息论的香农的硕士论文《通信的数学原理》中，香农自己都说：“Credit should also be given to Professor N. Wiener”，也就是荣誉也当属于维纳教授。³

Acknowledgments

The writer is indebted to his colleagues at the Laboratories, particularly to Dr. H. W. Bode, Dr. J. R. Pierce, Dr. B. McMillan, and Dr. B. M. Oliver for many helpful suggestions and criticisms during the course of this work. Credit should also be given to Professor N. Wiener, whose elegant solution of the problems of filtering and prediction of stationary ensembles has considerably influenced the writer's thinking in this field.

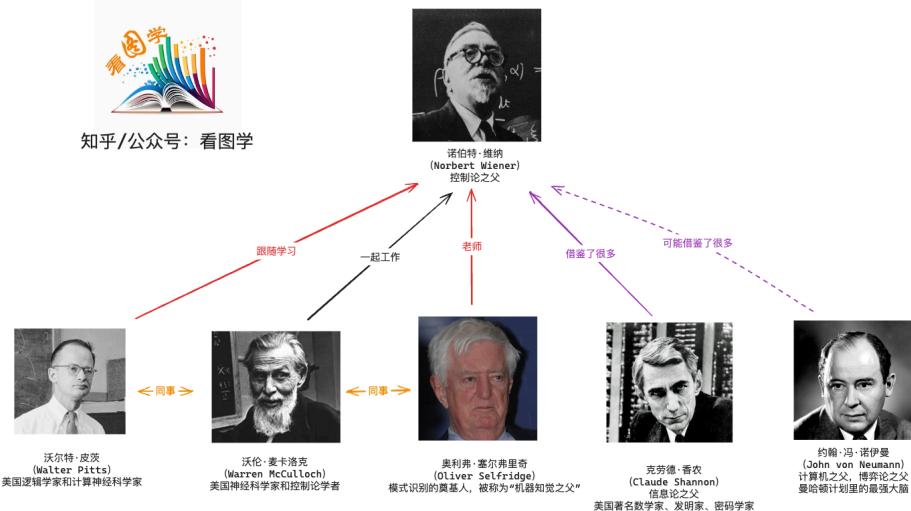
在一本书《维纳传：信息时代的隐秘英雄》曾经提到，冯诺依曼也经常参加维纳的控制论相关的会议，而且是“毫无保留地或者几乎毫无保留地把自己对计算机和自动机器的想法告诉了冯·诺依曼”，而且还派了朱利安·毕格罗（Julian Bigelow）给冯诺依曼当助手。所以有人认为冯诺依曼架构应该叫做维纳-冯诺依曼架构。维纳对后世的贡献很有可能被低估了。



³https://pure.mpg.de/rest/items/item_2383164/component/file_2383163/content

从左到右依次是朱利安·毕格罗、赫尔曼·戈德斯汀、罗伯特·奥本海默(原子弹之父)和约翰·冯·诺依曼

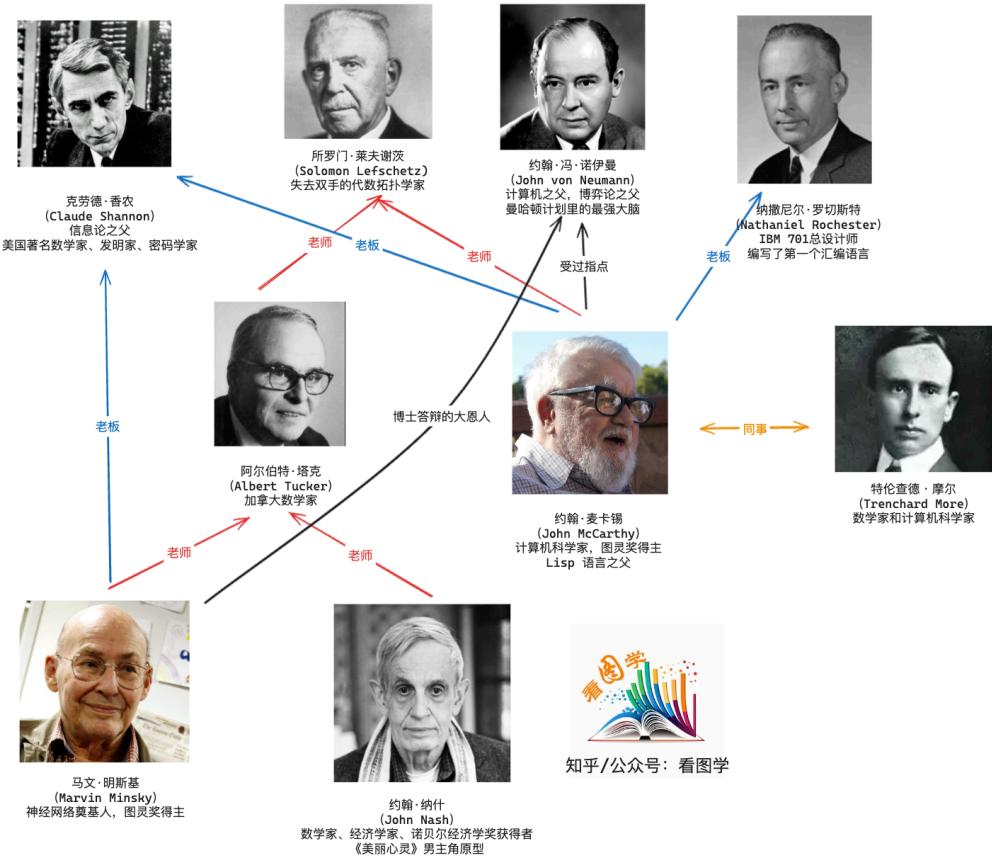
维纳的影响力在当时是巨大的。



1954年，一个叫艾伦·纽厄尔(Allen Newell)的年轻人在一次研讨会上听到奥利弗·塞尔弗里奇(Oliver Selfridge)描述“运行一个计算机程序，学会了识别字母和其他模式”，也开始做人工智能相关的工作，但是他与他的博导司马贺(Herbert Simon)后来开辟了一条全新的道路。

1955年夏天，麦卡锡又跑去IBM打工了。为啥麦卡锡老在夏天跑出去打工？因为那个时候美国的教授只发9个月工资，你要是没有科研经费，就得趁着暑假出去打工，年轻教授真是不容易。打工的老板是纳撒尼尔·罗切斯特(Nathaniel Rochester)，这个老板对神经网络很感兴趣。麦卡锡一看，机会来了，有现老板和前老板站台，再加上前同事明斯基，4个人决定搞一次人工智能的会议。

麦卡锡之所以拉上另外3个人，是因为要申请会议经费，另外三个名气当时都很大。很明显，麦卡锡是达特茅斯会议的KOL。



1955 年还发生了一件事，那就是在美国西部计算机联合大会上，前面提到的奥利弗·塞尔弗里奇 (Oliver Selfridge) 发表了一篇模式识别的文章，他之前都跟神经网络的大牛们混在一起，所以自然是研究神经网络。但是当时研究智能还有另外一派，那就是模拟人的心智。艾伦·纽厄尔 (Allen Newell) 作为这一方向的研究者在会上探讨了计算机下棋。

讨论会的主持人是前面讲对最早提出人工神经元对沃尔特·皮茨 (Walter Pitts)，他最后总结时说：“(一派人) 企图模拟神经系统，而纽厄尔则企图模拟心智 (mind) ……但殊途同归。”

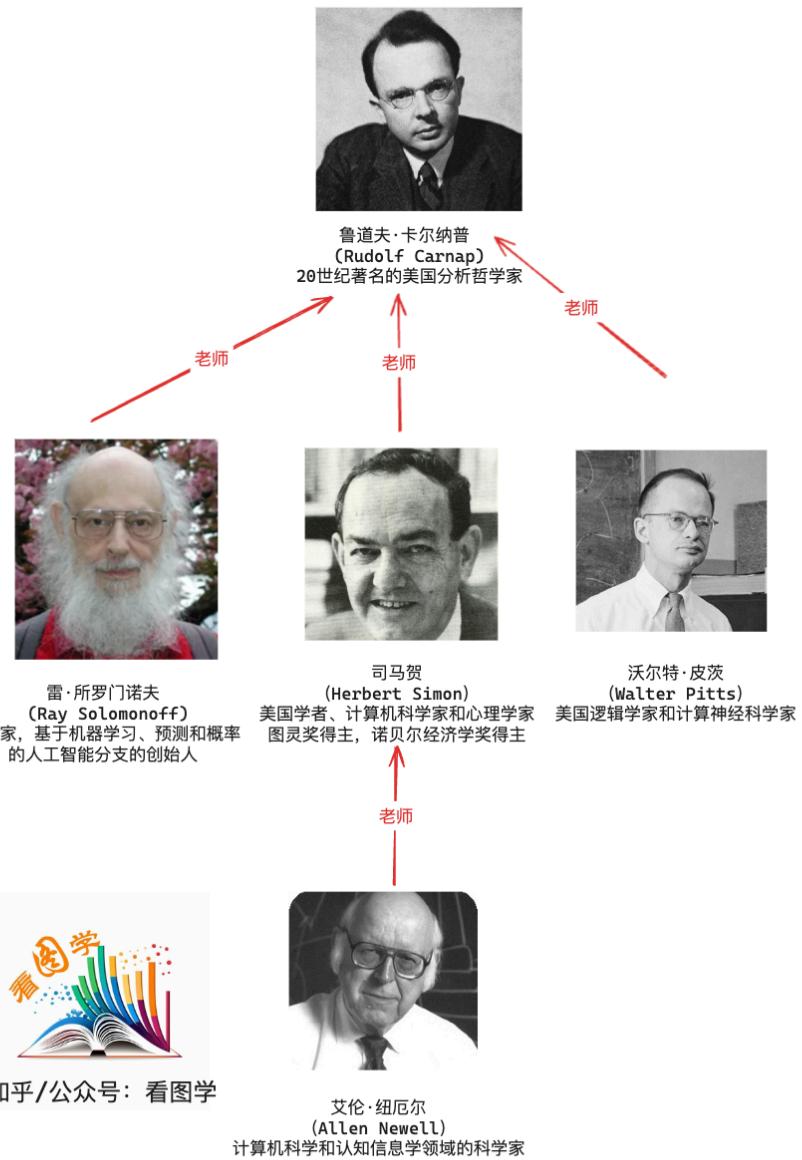
但是从后面的 AI 的发展来看，这两派丝毫没有同归的意思，反而想要同归于尽。

艾伦·纽厄尔 (Allen Newell) 的博导是司马贺 (Herbert Simon)，这两个人是后来人工智能符号派的代表。这两个人后来和第一届图灵奖获得者阿兰·珀里思 (Alan Perlis) 一起创办了卡内基梅隆大学的计算机系，硬是把一个三本提升到了世界一流学校。



司马贺（Herbert Simon）对中国大陆学术界有较深影响。“乒乓外交”打破了中美坚冰后的 1972 年 7 月，赫伯特·西蒙作为美国计算机科学代表团成员首次访问中国，后多次访华交流讲学及合作研究。其中文名字司马贺，即是他 1980 年作为美国心理学代表团成员第二次访华时所起，其本人 70 多岁的年龄开始学习汉语。1994 年当选为中国科学院外籍院士。

司马贺在芝加哥大学求学时，也曾经跟皮茨的老师卡尔纳普（Rudolf Carnap）学习，受他的影响开启了研究人工智能之路。所以虽然后面的人工智能分成了连接派和符号派，但是他们的启蒙老师都是卡尔纳普（Rudolf Carnap）。



终于，在 1956 年，麦卡锡筹备的会议开始了，这次会议名就叫：达特茅斯夏季人工智能研究计划（Dartmouth Summer Research Project on Artificial Intelligence）。

下面这张照片，展示了本次参会的主要 7 个人。



Nathaniel Rochester Marvin L. Minsky John McCarthy
Oliver G. Selfridge Ray Solomonoff Trenchard More Claude E. Shannon

August 1956

当时还有很多人也参加了。比如同样受到卡尔纳普（Rudolf Carnap）影响的所罗门诺夫（Solomonoff），还有两个是来自 IBM 的撒缪尔（Arthur Samuel），达特茅斯的教授摩尔（Trenchard More）。

所以后来有些文献说主要有 10 个人参加。这 10 个人有时候被称为 AI 之父。

2016.4
3
40.

a medal". All of them are heroes. But did this workshop leave out some

1956 Dartmouth Conference: The Founding Fathers of AI



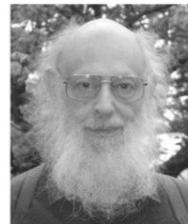
John McCarthy



Marvin Minsky



Claude Shannon



Ray Solomonoff



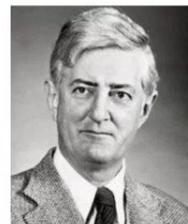
Alan Newell



Herbert Simon



Arthur Samuel



Oliver Selfridge



Nathaniel Rochester



Trenchard More

after him?

麦卡锡后来把参会名单弄丢了，好在所罗门诺夫（Solomonoff）全程参与了两个月的会议，他有个好习惯是记笔记。他的笔记中，记录了 20 个人参加了这次会议，比如纳什。

T.M.

from 196

People at Summer research project.

Solomonoff

Maurice Wilsky MIT Lincoln

John McCarthy IBM, Dartmouth

Claude Shannon MIT, Bell

French More IBM, MIT

Mat Rochester IBM Poughkeepsie

Oliver Selfridge MIT Lincoln

Julian Bigelow IAS

W. Ross Ashby Barnwood house (?)

W.S. McCulloch, MIT, RLE

Abraham Robinson Montreal logic

Tom Etter

John Nash MIT

David Sayre IBM New York

Samuels (IBM) non checkers

Shoulders MIT RLE or Lincoln components man

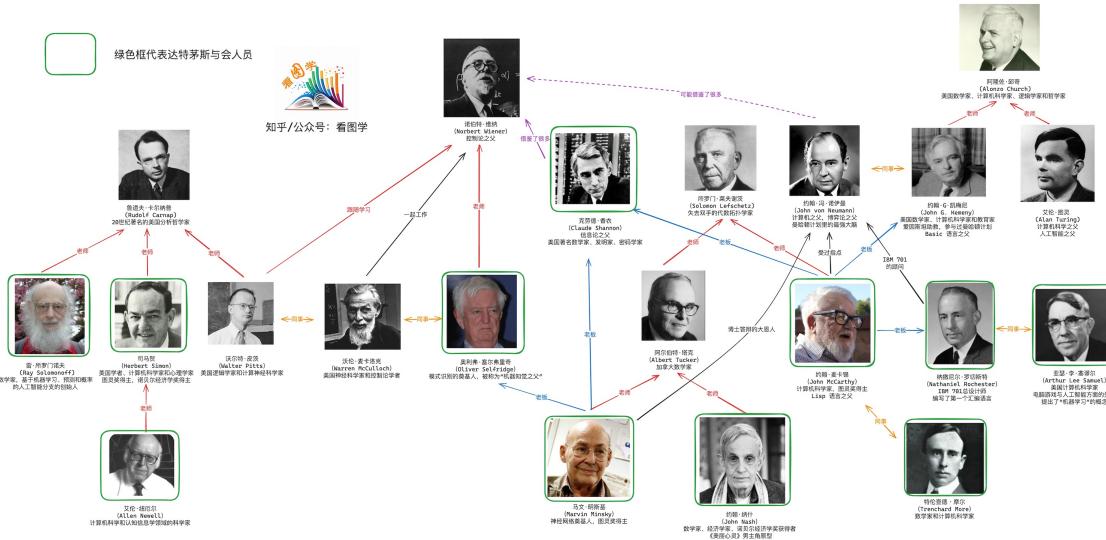
... (with Shoulders)

Alex Bernstein IBM (New York) chess

Herbert Simon: U of Pa (?)

Allen Newell: Rand

上面提到的人建立个关系图谱，大概是下面这个样子。



1.1.3 AI 一大都研究了什么？

会议的主要议题如下：

1. 自动计算机

如果一台机器可以做一项工作，则一台可编程自动计算机能用来模拟这台机器。现有计算机的速度和内存容量可能不足以模拟人脑的许多高级功能，但主要障碍不是缺乏机器容量，而是我们无法编写充分利用我们所拥有优势的程序。

作者注：这个目标似乎已经实现，目前图像识别，语音识别等都已经做的不错了。

2. 如何使用语言对计算机进行编程

可以推测，人类思想的很大一部分包括根据推理规则和猜想规则来操控单词。从这个角度来看，形成包括承认一个新词和一些规则的概括，其中含有它的句子暗示并被其他人暗示。这个想法从未如此精确地制定，也没有制定出实例。

作者注：站在现在的角度看，这就是 NLP 发展的一个基本思想，如何表示一个单词？大佬们 70 年前就已经尝试从 NLP 这个领域来突破 AI 了。

3. 神经网络

如何安排一组（假设的）神经元以形成概念。乌特利·拉什夫斯基 (Uttley, Rashevsky) 和他的团队，法利 (Farley) 和克拉克 (Clark)，皮茨 (Pitts) 和麦卡洛克 (McCulloch)，明斯基 (Minsky)，罗切斯特 (Rochester) 和霍兰德 (Holland) 等人在这个问题上做了大量的理论和实验工作。已经获得了部分结果，但问题是需要更多的理论工作。

作者注：最最早期的神经网络的探索。

4. 计算大小的理论

如果给出一个定义明确的问题（可以用机械方式测试提出的答案是否是有效答案），解决问题的方法是按顺序尝试所有可能的答案。这种方法效率低，要排除它，必须有一些计算效率的标准。一些考虑将表明，为了测量计算的效率，有必要手头有一种测量计算装置复杂性的方法，如果有一个具有功能复杂性的理论，则可以这样做。香农 (Shannon) 和麦卡锡 (McCarthy) 也获得了关于这个问题的部分结果。

作者注：这个属于计算机科学中的算法理论的范畴了。

5. 自我改进

可能真正智能的机器将开展可以最好地描述为自我改进的活动。已经提出了一些这样做的方案，值得进一步研究。这个问题似乎也可以抽象地进行研究。

作者注：已经是机器学习的基本套路了。

6. 抽象

许多类型的“抽象”可以明确定义，而其他几个则不那么明显。直接尝试对这些进行分类并描述从感官数据和其他数据形成抽象的机器方法似乎是值得的。

作者注：如何数据中归纳出知识，也可以归到机器学习。

7. 随机性和创造力

一个相当有吸引力但又不完全不完整的猜想是，创造性思维和缺乏想象力的能力思维之间的区别在于注入一些随机性。随机性必须由直觉引导才能有效。换句话说，受过教育的猜测或预感包括在其他有序思维中的受控随机性。

作者注：这个就有点 AGI 的味道了。当时是通过加入随机性来提高创造性，现在可以通过 Temperature 来调整。那个时候的大牛们是真的没想到后来 GPT “涌现”出来的智能。

1.1.4 AI 一大的影响

从上面可以看出，大佬们在会上还是讨论了很多东西，而且很多东西站在现在来看也并没有过时。这次会议上关于 AI 的研究方向已经隐隐划分为两派：

1. 符号派。他们开创性的认为：知识可以由一组规则来表示，而计算机程序可以使用逻辑来操控这些知识。这一派参会的主要代表人物是：艾伦·纽厄尔 (Allen Newell)、司马贺 (Herbert Simon)，约翰·麦卡锡 (John McCarthy)，马文·明斯基 (Marvin Minsky)。明斯基早期应该是连接派的，因为他的博士论文就是研究神经网络，后来给神经网络死刑，然后叛变成符号派了。在这个会上，符号派的代表纽厄尔和司马贺公布了一款程序“逻辑理论家”(Logic Theorist)，这个程序可以证明怀特海和罗素《数学原理》中命题逻辑部分的一个很大子集。是整个会议最令人印象深刻的。此时此刻符号派稳压连接派一头。
2. 连接派。连接派就是通过一种刻画人类大脑中神经元之间相互连接的机制，来模拟人类行为。当时在会上只是讨论，当时还没有特别亮眼的表现，提到了皮茨 (Pitts) 和麦卡洛克 (McCulloch)。但是会后没多久，神经网络也有了突破。

有意思的是，鲁道夫·卡尔纳普 (Rudolf Carnap) 的两个学生，司马贺与皮兹，分别成了符号派和连接派最早期的代表，还有个学生雷·所罗门诺夫 (Ray Solomonoff) 则在机器学习领域有开创性的贡献，从这个角度来看，AI 的起源，鲁道夫·卡尔纳普是有巨大贡献的。

还有一个流派也隐藏在上面的图中，那就是起源于维纳的控制论，也就是后来的行为主义。采用“感知-动作”来研究，通过环境反馈和智能行为之间的因果去探寻智能，后来就发展除了强化学习和现在正火的 Agent 。

维纳对 AI 的贡献甚至比想象中的还要大，以至于 Michael Jordan (2018) 曾讽刺的说到：“维纳提出的方法却披着麦卡锡发明的术语的外衣”。维纳提出的 Cybernetics，被翻译成了控制论，但实际上“机械大脑论”可能更符合这个词的本意。只不过当时维纳突然与自己的学生决裂，导致年轻学生们都想“离维纳的控制论越远越好”。麦卡锡当时就是这么想的，他曾在“控制论”和“自动机”之间徘徊不定，最后选择了“人工智能”。而且当时计算机的三大“之父”，除了图灵以外，香农、冯诺依曼都深深的受到维纳的影响。图灵是真的天才，自己独立搞了一套。所以维纳可能是活成了 AI 的里子。

AI 的故事到现在才刚刚开始，后续几年，符号派和连接派的内战就要开始了，我们下次再讲。

1.2 符号派、链接派，相爱又相杀

1.3 从 one-hot 到 ChatGPT

2 Chapter 2: 主流大模型的模型架构和细节分析

2.1 Position Embedding

从 Luong Attention 与 Bahdanau Attention 演变为 Transformers 的 Scaled Dot-Product Attention 后，出现了一个问题，那就是 token 的位置信息丢失了。

基于传统的 RNN 结构的 Attention 由于时刻 t 的隐层计算依赖时刻 $t - 1$ 的隐层，所以位置信息理论上是可以传递的。

Transformers 的 Attention 丢弃了 RNN 的结构之后，带来的好处就是可以并行计算了，但是信息通过时间/位置的传递的特性也就丢失了。

然而位置信息又很重要。

比如曾国藩写周报，如果写“臣屡战屡败”，结果可能是拖出去斩首。如果写“臣屡败屡战”，结果可能是忠勇无双有赏赐。

研究人员自然是想既要有要，所以就想办法从别的地方把位置信息加进去。我们来研究下 Position Embedding 的发展史。

2.1.1 朴素的想法

一个很直接的方法，就是直接把位置输入进去。这样做有两个缺点：

1. 泛化不好，没见过的位置模型处理不好
2. 这样模型的权重存在很大的数字，神经网络不喜欢大数字，影响训练的稳定性。

那如果是将位置做个归一化呢？也很难，因为无法找到一个归一化的标准。

2.1.2 绝对位置编码 (sinusoidal PE)

在《Attention is All you Need》的论文中，单独设置了一个绝对位置编码，叫 sinusoidal 位置编码，这个编码会和输入的词向量相加。文中的编码函数如下：

$$PE_{(t,2i)} = \sin\left(\frac{t}{10000^{2i/d}}\right)$$

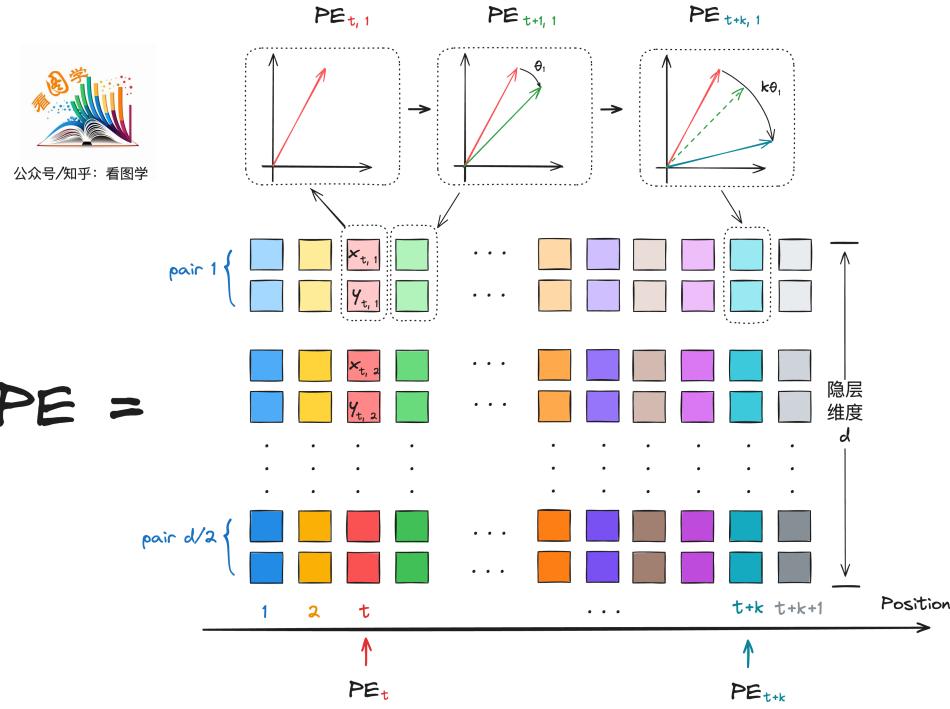
$$PE_{(t,2i+1)} = \cos\left(\frac{t}{10000^{2i/d}}\right)$$

其中 t 代表了输入的位置。原文用了 pos 来表示，为了后续推导公式的方便，将 pos 统一改写为 t 。
 i 则是 Position Embedding 的向量下标，向量长度为 d , i 的取值范围是 $[0, \frac{d}{2})$

这个函数看起来奇怪又有些恐怖，到底是什么鬼？

我们先看下结论：用图示来解释一下这个函数到底有什么特性。

Sinusoidal 绝对位置编码中，分组后，相对位置之间的线性变换等价于【顺时针旋转】



通俗一点来说，这个函数就是根据向量的下标两两配对，分成多组，每一组的二维向量根据 **位置信息** 进行 **顺时针旋转**，旋转的角度跟相对位置是线性关系。

论文中对这个函数进行了解释：“We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k , $PE_{(pos+k)}$ can be represented as a linear function of PE_{pos} .

用学术一点的说法，就是该函数是相对位置 k 的一个线性变换，也就是符合这么一个特性：

$$PE_{t+k} = f(PE_t, k)$$

其中， f 在这里是进行了顺时针旋转，旋转的角度是相对位置 k 的线性函数。

1. 下面是详细的证明：

为了方便，我们把原函数进行一些简化。原函数为：

$$PE_{(t,2i)} = \sin\left(\frac{t}{10000^{2i/d}}\right)$$

$$PE_{(t,2i+1)} = \cos\left(\frac{t}{10000^{2i/d}}\right)$$

由于函数对向量进行了两两分组，我们用 j 来表示分组 $PE_{(t,2i)}, PE_{(2i+1)}$ ，则有

$$PE_{(t,j)} = \begin{cases} \sin(\theta_j \cdot t), & \text{if } j = 2i/2 \\ \cos(\theta_j \cdot t), & \text{if } j = (2i + 1)/2 \end{cases}$$

其中

$$\theta_j = \frac{1}{10000^{j/d}}$$

, 则 Position Embedding 则可以表示为:

$$PE_t = \begin{bmatrix} \sin(\theta_1 \cdot t) \\ \cos(\theta_1 \cdot t) \\ \sin(\theta_2 \cdot t) \\ \cos(\theta_2 \cdot t) \\ \vdots \\ \sin(\theta_{d/2} \cdot t) \\ \cos(\theta_{d/2} \cdot t) \end{bmatrix}_{d \times 1}$$

对于第 j 个 pair $PE_{t,j}$ 来说, 如果 $PE_{t+k,j}$ 是 $PE_{t,j}$ 的线性变换, 则存在一个矩阵 $M \in \mathbb{R}^{2 \times 2}$ 使得:

$$M \cdot PE_{t,j} = PE_{t+k,j}$$

也就是

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} \cdot \begin{bmatrix} \sin(\theta_j \cdot t) \\ \cos(\theta_j \cdot t) \end{bmatrix} = \begin{bmatrix} \sin(\theta_j \cdot (t+k)) \\ \cos(\theta_j \cdot (t+k)) \end{bmatrix}$$

根据三角函数求和的公式:

$$\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta$$

$$\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta$$

将右侧展开, 左侧根据矩阵乘法展开, 则有

$$\begin{aligned} \begin{bmatrix} u_1 \times \sin(\theta_j \cdot t) + v_1 \times \cos(\theta_j \cdot t) \\ u_2 \times \sin(\theta_j \cdot t) + v_2 \times \cos(\theta_j \cdot t) \end{bmatrix} &= \begin{bmatrix} \sin(\theta_j \cdot t) \cos(\theta_j \cdot k) + \cos(\theta_j \cdot t) \sin(\theta_j \cdot k) \\ \cos(\theta_j \cdot t) \cos(\theta_j \cdot k) - \sin(\theta_j \cdot t) \sin(\theta_j \cdot k) \end{bmatrix} \\ &= \begin{bmatrix} \sin(\theta_j \cdot t) \cos(\theta_j \cdot k) + \cos(\theta_j \cdot t) \sin(\theta_j \cdot k) \\ -\sin(\theta_j \cdot t) \sin(\theta_j \cdot k) + \cos(\theta_j \cdot t) \cos(\theta_j \cdot k) \end{bmatrix} \end{aligned}$$

解这个方程会得到 (根据上面公式对号入座就可以):

$$M = \begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta_j \cdot k) & \sin(\theta_j \cdot k) \\ -\sin(\theta_j \cdot k) & \cos(\theta_j \cdot k) \end{bmatrix}$$

学过线代的看着这个矩阵是不是有点眼熟? 没错, 这个矩阵和逆时针旋转矩阵

$$\begin{bmatrix} \cos(\theta_j \cdot k) & -\sin(\theta_j \cdot k) \\ \sin(\theta_j \cdot k) & \cos(\theta_j \cdot k) \end{bmatrix}$$

非常像。

这两个矩阵什么联系呢？其实也很简单，上面的是顺时针旋转矩阵，下面的是逆时针旋转矩阵。

比如我顺时针旋转了 θ ，就相当于逆时针旋转了 $-\theta$ ，带入逆时针旋转矩阵有：

$$R(-\theta) = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}$$

所以，我们就知道了这个论文中提到的线性变换 $PE_{t+k} = f(PE_t, k)$ 就是顺时针旋转，旋转的角度为 $\theta_j \cdot k$ ，和相对位置 k 是线性关系。

对于所有 pair 来说， PE_{t+k} 相对于 PE_t 的顺时针旋转，写成矩阵形式为：

$$\begin{bmatrix} \cos(\theta_1 \cdot k) & \sin(\theta_1 \cdot k) & 0 & 0 & \cdots & 0 & 0 \\ -\sin(\theta_1 \cdot k) & \cos(\theta_1 \cdot k) & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos(\theta_2 \cdot k) & \sin(\theta_2 \cdot k) & \cdots & 0 & 0 \\ 0 & 0 & -\sin(\theta_2 \cdot k) & \cos(\theta_2 \cdot k) & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & \cos(\theta_{d/2} \cdot k) & \sin(\theta_{d/2} \cdot k) \\ 0 & 0 & 0 & 0 & \cdots & -\sin(\theta_{d/2} \cdot k) & \cos(\theta_{d/2} \cdot k) \end{bmatrix} \begin{bmatrix} \sin(\theta_1 \cdot t) \\ \cos(\theta_1 \cdot t) \\ \vdots \\ \sin(\theta_{d/2} \cdot t) \\ \cos(\theta_{d/2} \cdot t) \end{bmatrix}$$

不得不说这个设计还是很精彩的，虽然还是写死的绝对位置编码，但是却巧妙的捕捉到了相对位置关系。

下面是一些疑问和思考：

- 为什么要分组？不分组然后向量整体旋转行不行？理论上也可以，但是这样做的话上面的矩阵就变成了一个稠密矩阵，影响运算速度。
- 为什么随着下标的增加，旋转的角度越来越小？这个我也没有很严谨的证明过，感觉跟时钟系统有点像吧，时针分针秒针分别表示不同粒度的时间，在钟表上顺时针旋转。这里既然也是旋转，那用不同的颗粒度应该能捕获更细粒度的位置信息。

需要注意的是，《Attention is All your Need》中，词向量和绝对位置向量采用的是相加的操作，然后再进行线性变换和 Attention 操作。具体来说就是：

$$\mathbf{q}_i = (\mathbf{x}_i + \mathbf{p}_i)\mathbf{W}_Q$$

$$\mathbf{k}_j = (\mathbf{x}_j + \mathbf{p}_j)\mathbf{W}_K$$

$$\mathbf{v}_j = (\mathbf{x}_j + \mathbf{p}_j)\mathbf{W}_V$$

$$a_{i,j} = \text{softmax}(\mathbf{q}_i \mathbf{k}_j^\top)$$

$$\mathbf{o}_i = \sum_j a_{i,j} \mathbf{v}_j$$

比如将 $q_i k_j^\top$ 完全展开后

$$\mathbf{q}_i \mathbf{k}_j^\top = \mathbf{x}_i \mathbf{W}_Q \mathbf{W}_K^\top \mathbf{x}_j^\top + \mathbf{x}_i \mathbf{W}_Q \mathbf{W}_K^\top \mathbf{p}_j^\top + \mathbf{p}_i \mathbf{W}_Q \mathbf{W}_K^\top \mathbf{x}_j^\top + \mathbf{p}_i \mathbf{W}_Q \mathbf{W}_K^\top \mathbf{p}_j^\top$$

上面有颜色的部分代表了位置编码，有的将其变成可训练的参数，有的甚至把中间两项都去掉了。总之，后续的 Position Embedding 很多进展都是在 x_i, p_i 还有他们之间的组合关系上做文章。

苏剑林对 sinusoidal 绝对位置编码进行了改造，提出了一种更优雅和兼具外推性的编码方式，就是旋转位置编码 (RoPE)。

2.1.3 旋转位置编码 (RoPE)

绝对位置编码采用的是词向量与 PE 相加，然后再做线性变换的方式。公式如下：

$$\mathbf{q}_t = (\mathbf{x}_t + \mathbf{p}_t) \mathbf{W}_Q$$

而 RoPE 则更简单直接，丢弃了绝对位置编码 PE，词向量做线性变换后，按照前面说的 PE 分组方式，直接进行旋转。

这个论文已经画的十分清楚了：

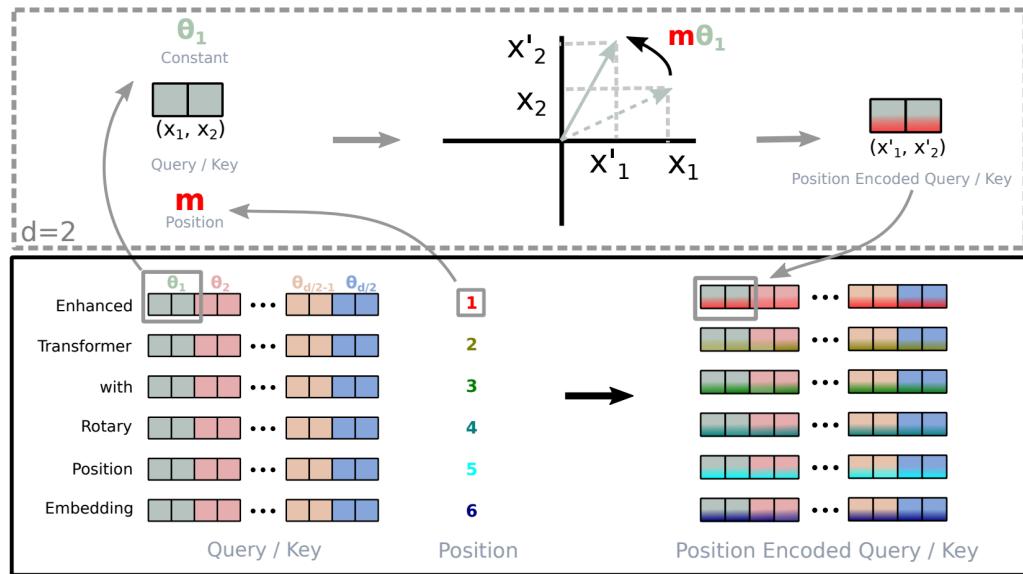


Figure 1: Implementation of Rotary Position Embedding(RoPE).

下面是详细的公式推导，去掉复数领域，只考虑旋转矩阵的一些证明。

假设 $\mathbf{q}_t = \mathbf{x}_t \mathbf{W}_Q$ ，仿照绝对位置编码进行两两分组，则

$$q_t = \begin{bmatrix} q_{t,1}^{(1)} \\ q_{t,1}^{(2)} \\ q_{t,2}^{(1)} \\ q_{t,2}^{(2)} \\ \vdots \\ q_{t,d/2}^{(1)} \\ q_{t,d/2}^{(2)} \end{bmatrix}_{d \times 1}$$

对于一个分组 $q_{t,j}$ 来说，旋转后的 $RoPE(q_{t,j})$ 为：

$$RoPE(q_{t,j}) = R(\theta_j \cdot t)(q_{t,j}) = \begin{bmatrix} \cos(\theta_j \cdot t) & -\sin(\theta_j \cdot t) \\ \sin(\theta_j \cdot t) & \cos(\theta_j \cdot t) \end{bmatrix} \begin{bmatrix} q_{t,j}^{(1)} \\ q_{t,j}^{(2)} \end{bmatrix}$$

写成矩阵形式为：

$$RoPE(q_t) = R(\theta \cdot t)(q_t) = \begin{bmatrix} \cos(\theta_1 \cdot t) & -\sin(\theta_1 \cdot t) & 0 & 0 & \cdots & 0 & 0 & q_{t,1}^{(1)} \\ \sin(\theta_1 \cdot t) & \cos(\theta_1 \cdot t) & 0 & 0 & \cdots & 0 & 0 & q_{t,1}^{(2)} \\ 0 & 0 & \cos(\theta_2 \cdot t) & -\sin(\theta_2 \cdot t) & \cdots & 0 & 0 & q_{t,2}^{(1)} \\ 0 & 0 & \sin(\theta_2 \cdot t) & \cos(\theta_2 \cdot t) & \cdots & 0 & 0 & q_{t,2}^{(2)} \\ 0 & 0 & 0 & 0 & \ddots & 0 & 0 & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos(\theta_{d/2} \cdot t) & -\sin(\theta_{d/2} \cdot t) & q_{t,d/2}^{(1)} \\ 0 & 0 & 0 & 0 & \cdots & \sin(\theta_{d/2} \cdot t) & \cos(\theta_{d/2} \cdot t) & q_{t,d/2}^{(2)} \end{bmatrix}$$

RoPE 这么做的好处是，当 q 和 k 进行 attention 操作后，依然可以保留相对位置，这样就很好的完成了位置编码的任务，而且旋转位置编码就像时钟一样可以无限的旋转，具备很好的外推性。

$$RoPE(q_{m,j}), RoPE(k_{n,j}) = RoPE(q_{m-n,j}), RoPE(k_{0,j})$$

证明如下：

假设位置 m 的 pair $q_{m,j}$ 和位置 n 的 pair $k_{n,j}$ 取点积操作计算 attention，则有：

$$\begin{aligned}
RoPE(q_{m,j}), RoPE(k_{n,j}) &= \begin{bmatrix} \cos(\theta_j.m) & -\sin(\theta_j.m) \\ \sin(\theta_j.m) & \cos(\theta_j.m) \end{bmatrix} \begin{bmatrix} q_{m,j}^{(1)} \\ q_{m,j}^{(2)} \end{bmatrix}, \begin{bmatrix} \cos(\theta_j.n) & -\sin(\theta_j.n) \\ \sin(\theta_j.n) & \cos(\theta_j.n) \end{bmatrix} \begin{bmatrix} k_{n,j}^{(1)} \\ k_{n,j}^{(2)} \end{bmatrix} \\
&= \begin{bmatrix} q_{m,j}^{(1)} \cos(\theta_j.m) - q_{m,j}^{(2)} \sin(\theta_j.m) \\ q_{m,j}^{(1)} \sin(\theta_j.m) + q_{m,j}^{(2)} \cos(\theta_j.m) \end{bmatrix}, \begin{bmatrix} k_{n,j}^{(1)} \cos(\theta_j.n) - k_{n,j}^{(2)} \sin(\theta_j.n) \\ k_{n,j}^{(1)} \sin(\theta_j.n) + k_{n,j}^{(2)} \cos(\theta_j.n) \end{bmatrix} \\
&= (q_{m,j}^{(1)} \cos(\theta_j.m) - q_{m,j}^{(2)} \sin(\theta_j.m)) \times (k_{n,j}^{(1)} \cos(\theta_j.n) - k_{n,j}^{(2)} \sin(\theta_j.n)) \\
&\quad + (q_{m,j}^{(1)} \sin(\theta_j.m) + q_{m,j}^{(2)} \cos(\theta_j.m)) \times (k_{n,j}^{(1)} \sin(\theta_j.n) + k_{n,j}^{(2)} \cos(\theta_j.n)) \\
&= q_{m,j}^{(1)} k_{n,j}^{(1)} (\cos(\theta_j.m) \cos(\theta_j.n) + \sin(\theta_j.m) \sin(\theta_j.n)) \\
&\quad + q_{m,j}^{(1)} k_{n,j}^{(2)} (-\cos(\theta_j.m) \sin(\theta_j.n) + \sin(\theta_j.m) \cos(\theta_j.n)) \\
&\quad + q_{m,j}^{(2)} k_{n,j}^{(1)} (-\sin(\theta_j.m) \cos(\theta_j.n) + \cos(\theta_j.m) \sin(\theta_j.n)) \\
&\quad + q_{m,j}^{(2)} k_{n,j}^{(2)} (\sin(\theta_j.m) \sin(\theta_j.n) + \cos(\theta_j.m) \cos(\theta_j.n)) \\
&= q_{m,j}^{(1)} k_{n,j}^{(1)} \cos(\theta_j.(m-n)) + q_{m,j}^{(1)} k_{n,j}^{(2)} \sin(\theta_j.(m-n)) \\
&\quad + q_{m,j}^{(2)} k_{n,j}^{(1)} \sin(\theta_j.(m-n)) + q_{m,j}^{(2)} k_{n,j}^{(2)} \sin(\theta_j.(m-n)) \\
&= (q_{m,j}^{(1)} \cos(\theta_j.(m-n)) - q_{m,j}^{(2)} \sin(\theta_j.(m-n))) \times k_{n,j}^{(1)} \\
&\quad + (q_{m,j}^{(1)} \sin(\theta_j.(m-n)) + q_{m,j}^{(2)} \sin(\theta_j.(m-n))) \times k_{n,j}^{(2)} \\
&= RoPE(q_{m-n,j}), RoPE(k_{0,j})
\end{aligned}$$

2.1.4 其他编码方式

目前主流就是 RoPE，其他的方式后面慢慢更新。

3 Chapter 3: 大语言模型 pipeline

3.1 Stage 1: Pretrain

3.1.1 Continue Pretrain

3.2 Stage 2: SFT

- SFT 是否必须存在

3.3 Stage 3: Alginment

3.4 Stage 0: 数据处理

4 Chapter 4: 大语言模型训练

4.1 显卡和模型训练基础知识

4.2 多卡并行训练

4.3 当前流行训练框架

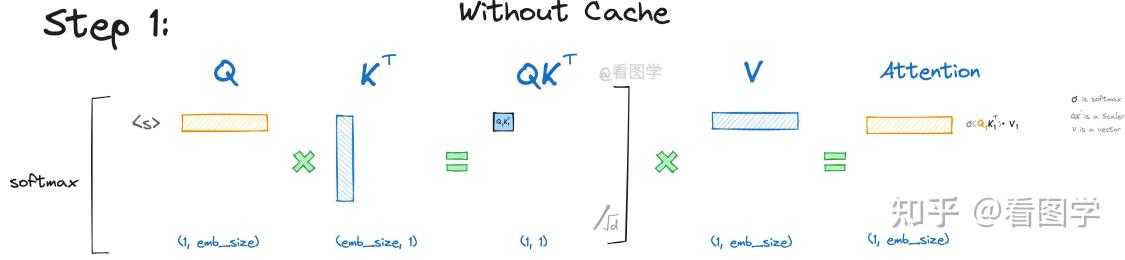
5 Chapter 5: 大语言模型推理

5.1 KV Cache

KV Cache 是 Transformer 标配的推理加速功能，transformer 官方 use_cache 这个参数默认是 True，但是它只能用于 Decoder 架构的模型，这是因为 Decoder 有 Causal Mask，在推理的时候前面已经生成的字符不需要与后面的字符产生 attention，从而使得前面已经计算的 K 和 V 可以缓存起来。

我们先看一下不使用 KV Cache 的推理过程。假设模型最终生成了“遥遥领先”4 个字。

当模型生成第一个“遥”字时，input=“<s>”，“<s>”是起始字符。Attention 的计算如下：

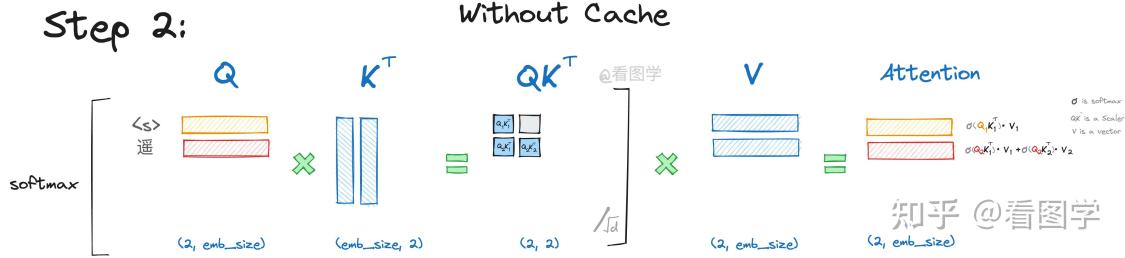


为了看上去方便，我们暂时忽略 scale 项 \sqrt{d} ，但是要注意这个 scale 面试时经常考。

如上图所示，最终 Attention 的计算公式如下，(softmaxed 表示已经按行进行了 softmax)：

$$Att_1(Q, K, V) = \text{softmax}(Q_1 K_1^T) \vec{V}_1 = \text{softmaxed}(Q_1 K_1^T) \vec{V}_1$$

当模型生成第二个“遥”字时，input=“<s> 遥”Attention 的计算如下：



当 QK^T 变为矩阵时，softmax 会针对 行进行计算。写详细一点如下，softmaxed 表示已经按行进行了 softmax。

$$\begin{aligned}
 Att_{step2}(Q, K, V) &= \text{softmax} \left(\begin{bmatrix} Q_1 K_1^T & -\infty \\ Q_2 K_1^T & Q_2 K_2^T \end{bmatrix} \right) \begin{bmatrix} \vec{V}_1 \\ \vec{V}_2 \end{bmatrix} \\
 &= \left(\begin{bmatrix} \text{softmax}(Q_1 K_1^T) & \text{softmax}(-\infty) \\ \text{softmax}(Q_2 K_1^T) & \text{softmax}(Q_2 K_2^T) \end{bmatrix} \right) \begin{bmatrix} \vec{V}_1 \\ \vec{V}_2 \end{bmatrix} \\
 &\stackrel{\$\$}{=} \left(\begin{bmatrix} \text{softmax}(Q_1 K_1^T) & 0 \\ \text{softmax}(Q_2 K_1^T) & \text{softmax}(Q_2 K_2^T) \end{bmatrix} \right) \begin{bmatrix} \vec{V}_1 \\ \vec{V}_2 \end{bmatrix} \\
 &= \left(\begin{bmatrix} \text{softmax}(Q_1 K_1^T) \times \vec{V}_1 + 0 \times \vec{V}_2 \\ \text{softmax}(Q_2 K_1^T) \times \vec{V}_1 + \text{softmax}(Q_2 K_2^T) \times \vec{V}_2 \end{bmatrix} \right) \\
 &= \left(\begin{bmatrix} \text{softmax}(Q_1 K_1^T) \times \vec{V}_1 \\ \text{softmax}(Q_2 K_1^T) \times \vec{V}_1 + \text{softmax}(Q_2 K_2^T) \times \vec{V}_2 \end{bmatrix} \right)
 \end{aligned}$$

假设 $\text{Att}_1(Q, K, V)$ 表示 Attention 的第一行， $\text{Att}_2(Q, K, V)$ 表示 Attention 的第二行，则根据上面推导，

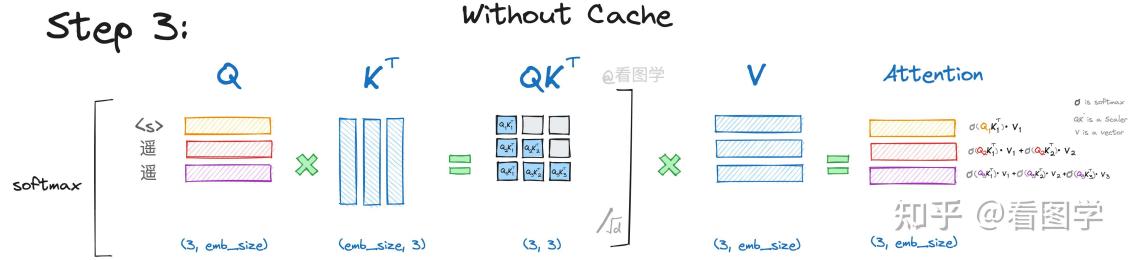
其计算公式为：

$$\begin{aligned} \text{Att}_1(Q, K, V) &= \text{softmax}(Q_1 K_1^T) \vec{V}_1 \\ \text{Att}_2(Q, K, V) &= \text{softmax}(Q_2 K_1^T) \vec{V}_1 + \text{softmax}(Q_2 K_2^T) \vec{V}_2 \end{aligned} \quad \$\$$$

你会发现，由于 $Q_1 K_2^T$ 这个值会 mask 掉

- Q_1 在第二步参与的计算与第一步是一样的，而且第二步生成的 V_1 也仅仅依赖于 Q_1 ，与 Q_2 毫无关系。
- V_2 的计算也仅仅依赖于 Q_2 ，与 Q_1 毫无关系。

当模型生成第三个“领”字时，input="<s> 遥遥" Attention 的计算如下：

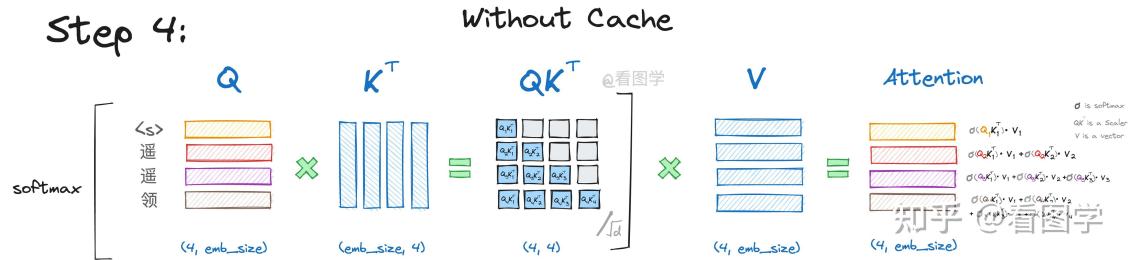


详细的推导参考第二步，其计算公式为：

$$\begin{aligned} \text{Att}_1(Q, K, V) &= \text{softmax}(Q_1 K_1^T) \vec{V}_1 \\ \text{Att}_2(Q, K, V) &= \text{softmax}(Q_2 K_1^T) \vec{V}_1 + \text{softmax}(Q_2 K_2^T) \vec{V}_2 \\ \text{Att}_3(Q, K, V) &= \text{softmax}(Q_3 K_1^T) \vec{V}_1 + \text{softmax}(Q_3 K_2^T) \vec{V}_2 + \text{softmax}(Q_3 K_3^T) \vec{V}_3 \end{aligned}$$

同样的， Att_k 只与 Q_k 有关。

当模型生成第四个“先”字时，input="<s> 遥遥领" Attention 的计算如下：



$$\begin{aligned} \text{Att}_1(Q, K, V) &= \text{softmax}(Q_1 K_1^T) \vec{V}_1 \\ \text{Att}_2(Q, K, V) &= \text{softmax}(Q_2 K_1^T) \vec{V}_1 + \text{softmax}(Q_2 K_2^T) \vec{V}_2 \\ \text{Att}_3(Q, K, V) &= \text{softmax}(Q_3 K_1^T) \vec{V}_1 + \text{softmax}(Q_3 K_2^T) \vec{V}_2 + \text{softmax}(Q_3 K_3^T) \vec{V}_3 \\ \text{Att}_4(Q, K, V) &= \text{softmax}(Q_4 K_1^T) \vec{V}_1 + \text{softmax}(Q_4 K_2^T) \vec{V}_2 + \text{softmax}(Q_4 K_3^T) \vec{V}_3 + \text{softmax}(Q_4 K_4^T) \vec{V}_4 \end{aligned}$$

和之前类似，不再赘述。

看上面图和公式，我们可以得出结论：

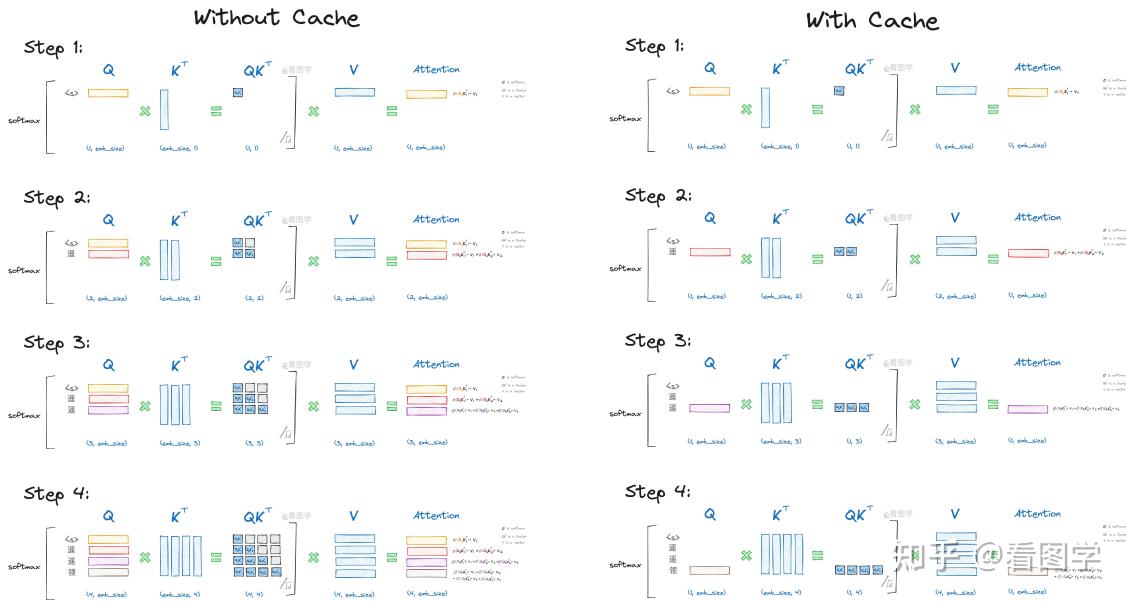
1. 当前计算方式存在大量冗余计算。
2. Att_k 只与 Q_k 有关。
3. 推理第 x_k 个字符的时候只需要输入字符 x_{k-1} 即可。

我们每一步其实之需要根据 Q_k 计算 Att_k 就可以，之前已经计算的 Attention 完全不需要重新计算。但是 K 和 V 是全程参与计算的，所以这里我们需要把每一步的 K, V 缓存起来。所以说叫 KV Cache 好像有点不太对，因为 KV 本来就需要全程计算，可能叫增量 KV 计算会更好理解。

下面 4 张图展示了使用 KV Cache 和不使用的对比。

推理加速KV Cache 示意图

看图学



下面是 gpt 里面 KV Cache 的实现。其实明白了原理后代码实现简单的不得了，就是 concat 操作而已。

https://github.com/huggingface/transformers/blob/main/src/transformers/models/gpt2/modeling_gpt2.py#L318C1-L331C97

```

if layer_past is not None:
    past_key, past_value = layer_past
    key = torch.cat((past_key, key), dim=-2)
    value = torch.cat((past_value, value), dim=-2)

if use_cache is True:
    present = (key, value)
else:
    present = None

if self.reorder_and_upcast_attn:
    attn_output, attn_weights = self._upcast_and_reordered_attn(

```

```
    query, key, value, attention_mask, head_mask)
else:
    attn_output, attn_weights = self._attn(query, key, value,
        attention_mask, head_mask)
```

最后需要注意当 sequence 特别长的时候，KV Cache 其实还是个 Memory 刺客。

比如 batch_size=32, head=32, layer=32, dim_size=4096, seq_length=2048, float32 类型，则需要占用的显存为 $2 * 32 * 4096 * 2048 * 32 * 4 / 1024/1024/1024 = 64G$ 。

针对内存的问题，学者们研发出了 Page Attention 之类的方法来优化。

6 Chapter 6: MOE, 多模态等

7 Chapter 7: 大语言模型评估

8 Chapter 8: 大语言模型应用

8.1 Prompt Engineering

8.2 Agent