

# AI 前线

2018年03月刊

AI - FRONT



关注落地技术，探寻AI应用场景



# QCon

## 全球软件开发大会

北京·国际会议中心

演讲：2018年4月20-22日

培训：2018年4月23-24日

### 大会部分日程

时间	专题	议题	讲师
主题演讲	编程语言	Shaping the future with Java, Faster	Georges Saab Oracle / Java平台事业群VP
	架构设计	拥抱变化：演进式架构	Neal Ford ThoughtWorks 总监， 《卓有成效的程序员》作者
	流处理	Apache Kafka的过去，现在，和未来	Jun Rao Confluent / 联合创始人， Kafka作者之一
	职业成长	产品经理的发现和成长	俞军 滴滴 / 产品高级副总裁
了解更多主题演讲 >>			
4月20日	数据库	MySQL的Docker容器化大规模实践	王晓波 同程艺龙 / 机票事业群CTO, TGO会员
	Java	GraalVM及其生态系统	郑雨迪 Oracle Labs / 高级研究员
	DevOps	从标准到落地：数据驱动的风险防范体系建设	华明 滴滴出行 / 运维架构师
	性能优化	分布式计算系统的性能优化	张建伟 百度 / 技术经理
	开源	OpenResty十年开源的历程和思考	温铭 OpenResty Inc. / 合伙人
	质量建设	手Q性能优化的大数据实战	谭力 腾讯科技 / 测试开发高级工程师
	管理创新	培育创新生态系统	吴穹 Agilean咨询公司 首席咨询顾问，软件工程专家

以上仅为大会首日部分演讲，识别二维码了解更多内容

**9折** 报名倒计时，立减**680元**  
 团购享受更多优惠

访问官网获取更多前沿技术趋势

2018.qconbeijing.com

如有任何问题，欢迎咨询

电话：15110019061，微信：qcon-0410





# 极客时间

重拾极客精神·提升技术认知

## 「专栏订阅」

每天 10 分钟，邀请顶级技术专家，探究技术本质，解读科技动态。

## 「极客新闻」

每天早上 8 点，朝闻技术天下事。

## 「热点专题」

最前沿的专题，最独特的视角，最风趣的解读。

## 「二叉树视频」

一档属于技术人的直播和短视频节目，记录与时代并行的技术人。



关注我，获取更多干货



# 卷首语

## 停下来，并不是一件坏事

作者 陈思

2018 年 3 月 14 日晚，河南焦作车主薛先生驾驶奔驰 200L 轿车开启定速巡航，后发现车辆失控，只能以 120 公里 / 时的速度继续飞驰，在交警和奔驰售后方面操作下，奔驰车在失控近一小时、约一百公里后，终于安全停下。

2018年3月19日，Uber的一辆自动驾驶汽车撞死了美国亚利桑那州一名正在过马路的49岁女性伊莱恩·赫兹伯格（Elaine Herzberg），这位不幸的女性在被送往医院后因伤死亡，这是自动驾驶汽车首例发生在公共道路上的致死案。事件发生后，Uber方面立即停止了所有自动驾驶车辆的测试。

2018年3月18日，美国知名社交媒体Facebook被曝出其合作机构剑桥分析对Facebook的数据使用是“不道德的实验”。剑桥分析被指，在未经用户同意的情况下，利用在Facebook上获得的5000万用户的个人资料数据，来创建档案、并在2016总统大选期间针对这些人进行定向宣传。

.....

对于全球的人工智能研究者和企业来说，最近这段时间接连被爆出的新闻对这项技术一点儿都不友好，甚至让本就饱受质疑的人工智能再次成为了众矢之的：人们更加不愿意相信自动驾驶车辆，因为没有人愿意在高速行驶的车流中将自己的声明交给一个没有人控制的机器；人们宁愿放弃社交软件，因为起码这样自己的个人信息安全不会受到威胁。

人们在面对任何新型科技或者发明的时候，总会抱有质疑和否定的态度，而科技工作者需要做的正是让人们相信它是无害的，或者“基本上”是无害的。

让我们站在今天的时间节点简单回顾一下我们最常用的两款电子产品的发展史：

- 从1946年“埃尼阿克”计算机出现，到第一代个人计算机于1984年问世，花费了38年；
- 从1875第一部电话诞生，到1973第一台移动电话出现，花费了98年；
- 从第一台移动电话，到1993年第一台全球公认的智能手机“IBM Simon”诞生，花费了20年；
- 智能手机从2008年逐渐开始普及，到今天成为最日常使用的电子产品，花费了10年；
- 人工智能技术从提出理论到被称作“元年”的2016年，花费了半个多世纪；
- 而AI技术从“元年”到AI科技公司林立的2017年，只用了1年时间。

在“埃尼阿克”诞生的时候，没有人相信未来这种叫做“电脑”的东西能走进千家万户，同样的，在人工智能的概念第一次出现的时候，人们更多的是认为它只能够存在于科学家们的实验室当中，可是，人工智能时代就这么来了，仅仅一年的时间，它来的太快了。

2016年，AlphaGo证明了人工智能可以战胜人类，于是从哪一年开始，一些从没有听过的、或者只存在于电影当中的名词开始出现在我们的

日常生活里：机器学习、深度学习、人脸识别、自动驾驶……在大多数人都还没有准备好的时候，人工智能就这么突然降临了，一时间，我们看到谷歌、百度、亚马逊等等科技巨头们的人工智能研究项目在飞速进行着；我们看到一些从没有听过、没有见过的人工智能科技公司遍地都是，就像一位AI领域的从业者所说：以前谈人工智能，人家说你脑子不正常，现在如果不谈人工智能，那你的脑子才不正常。

从一个陌生的概念发展的人们日常谈论的话题，从深藏实验室的研究项目到鳞次栉比的科技公司，这一切只发生在一年的时间里。

这样快到有些疯狂的发展速度，对于这项技术真的好吗？

有了前文提到的那些案例，以及更多没有被媒体曝光的问题，相信有人提出这样的问题也不会奇怪，不可否认的是，目前的技术水平完全有能力让AI飞速发展，但也同样有能力让某些别有用心的企业或个人拿去，当作敛财和害人的工具。

2018年3月28日，在英伟达GTC 2018大会上，英伟达CEO黄仁勋对Uber自动驾驶事故发表了一些感想，在发言中他说了这样一段话：“这显然是一个重要的时刻，你应该停下来从中吸取教训，毫无疑问，行业中的每个人都应该停下来。”

当发展缓慢的时候，我们需要审视自我，当发展迅速的时候，我们更需要审视自身，有时候停下来并不一定是件坏事。

愿身处人工智能浪潮下的每一个人都能够保持热爱、保持冷静。

# AI 前线

InfoQ 中文站 AI 月刊 2018 年 3 月

## 生态评论

8 人工智能发展神速？37 年前的尘封档案告诉你并没有

## 重磅访谈

22 想成为教育领域的阿里云，要分几步走？

## 落地实践

36 阿里巴巴最新实践：TVM+TensorFlow 提高神经机器翻译性能

## 推荐阅读

46 推特爆款：谷歌大脑工程师的深度强化学习劝退文

73 传奇工程师卡马克入坑 AI：徒手一周实现反向传播和 CNN

## 精选论文导读

82 JeffDean 又用深度学习搞事情：这次要颠覆整个计算机系统结构设计



# 人工智能发展神速？37 年前的尘封档案告诉你并没有

作者 | Jeremy Bernstein

译者 & 编辑 | Debra

近日，加州大学伯克利分校电子工程与计算机科学系助理教授 moritz hardt 发推吐槽关于 Perceptron 的发展进程，他毫不客气地评价道，“2018 年了，Perceptron 仍然是老生常谈，还是之前的基础算法”。



关注

It's 2018 and they still say that about  
essentially the same algorithm.

翻译自英语

## 10.2 The Perceptron

The New York Times wrote in 1958 that the Perceptron [Ros58] was:

*the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.*

So, let's see.

**Definition 10.2** (Perceptron). Given labeled points  $((x_1, y_1), \dots, (x_m, y_m)) \in (\mathbb{R}^n \times \{-1, 1\})^m$ , and initial point  $w_0 \in \mathbb{R}^n$ , the Perceptron is the following algorithm. For  $i_l \in \{1, \dots, m\}$  selected uniformly at random,

$$w_{l+1} = w_l(1 - \gamma) + \eta \begin{cases} y_{i_l} x_{i_l} & \text{if } y_{i_l} \langle w_l, x_{i_l} \rangle < 1 \\ 0 & \text{otherwise} \end{cases}$$

上午9:42 · 2018年2月28日

29 转推 137 喜欢



4 29 137

《MIT 科技评论》AI 高级编辑 Will Knight 随之转推表示同意，称“历史总是惊人的相似”。

**will knight** @willknight · 4小时前  
yes, it increasingly feels like history is repeating

**Moritz Hardt** @mrtz  
It's 2018 and they still say that about essentially the same algorithm.

Marvin Minsky in 1958 that the Perceptron [Bodenhamer et al. 1998] (neural computer that like Nagel expects about itself and be conscious of its existence) can learn and be conscious of its existence. Given labeled points  $((x_1, y_1), \dots, (x_n, y_n))$  and point  $x_0 \in \mathbb{R}^d$ , the Perceptron is the uniformly at random:

$$w_t(1 - \gamma) + \eta \begin{cases} y_i x_i & \text{if } y_i \neq \langle w_t, x_i \rangle \\ 0 & \text{otherwise} \end{cases}$$

翻译自英语

在他们看来，人工智能相关的基本概念和研究进展似乎过于缓慢。近日，AI 前线从《纽约客》挖掘到 1981 年的一篇对人工智能之父马文·明斯基的人物专访，我们得以拂去历史的灰尘，一窥在 20 世纪 80 年代，初步萌芽的人工智能研究究竟是什么面貌，以及这位 AI 界的领袖对 1981 年之前人工智能发展历程的回顾。

1979 年 7 月，一款名为 BKG 9.8 的计算机程序在蒙特卡洛举行的世界西洋双陆棋锦标赛中夺得冠军。这款程序的发明者是匹兹堡卡内基梅隆大学的计算机科学教授 Hans Berliner，它在卡内基梅隆的一台大型计算机上运行，并通过卫星连接到蒙特卡洛的一个机器人上。这个名为 Gammonoid 的机器人胸前有一个西洋棋显示屏，可以显示它自己，以及其意大利对手 Luigi Villa 的动作。Luigi Villa 在短时间内击败了所有人类挑战者，赢得了与 Gammonoid 对弈的权利。竞赛的奖励是五千美元，Gammonoid 最终以 7:1 赢得了比赛。一开始，几乎所有人认为它会输，其创建者 Berliner 还曾在《科学美国人》中撰文，描述了比赛之前人们对 Gammonoid 是如何嗤之以鼻的。

然而结果让人大跌眼镜。

我其实对计算机其实一知半解。20 世纪 50 年代计算机普及以来，他是典型的接受计算机教育的一代人，可以操作最新的可编程随身计算机，懂得基本的编程语言。但作为理论物理学家，我避免了在大型机器上

运行的物理问题。尽管如此，多年来我已经阅读了很多关于新的计算机革命和微处理器时代的书籍：一个微处理器可以将包含数千种元素的电路装入计算机硅晶芯片的时代，它小到可以插入针孔中去；机器的操作以十亿分之一秒为计量单位；而且，由于电磁信号仅能够以光速传播，从而导致机器性能受到限制的事实开始体现出来。关于这个问题和它的含义有太多的书籍和文章，很难区分谁对谁错，但是在所有这些计算机文献中，我一直很欣赏从马文·明斯基那里读到的东西，其自 1974 年以来一直是麻省理工学院的教授。在一篇题为《物质、精神和模型》(Matter, Mind, and Models) 的论文中，明斯基对自由意志的评论如下：

如果一个人完全理解一台机器或一个程序，他就没有将“意志”归因于它的冲动。如果一个人不能很好地理解，他必须提供一个不完整的解释模型。我们日常的高级人类活动的直觉模型相当不完整，我们非正式解释中的许多概念都经不起仔细推敲。自由意志或意志就是这样一种观念：人们无法解释它与随机随想的不同之处，但强烈地认为它确实如此。我猜想这个想法起源于强大的原始防御机制。简单地说，在童年时代，我们逐渐学会各种形式的侵略和强迫，并对之产生厌恶心理，无论我们是屈服亦或抵抗。等长大一些，当我们被告知我们的行为应当受到诸如此类的一套规矩的“控制”时，我们将这一事实与其他识别的强制性行为一起植入我们的模型中（不恰当地）。我们抵制这种来自所有人的“强迫”感。尽管抵抗在逻辑上是徒劳的，但它们会留在我们的记忆中，并且因具有缺陷性的解释而合理化，因为这在情感上是不可接受的。

在文章的后面，明斯基写道：

当构建智能机器时，我们不难发现，它们对于心智、意识、自由意志等方面信念和人类一样困惑和固执。所有这些问题都指向解释自我模型各部分之间复杂的相互作用。一个人或一台机器对这些事情的信念不会告诉我们关于这个人或机器的任何信息，而仅会告诉我们关于模型自身的东西。

我知道明斯基已经三十多年了，但在 20 世纪 40 年代末才第一次见

到他本人。他在哈佛具体干什么我不太清楚，他跟着作曲家 Irving Fine 学习音乐作曲，身为本科生却已经在心理学院和生物学院各有一间自己的实验室，而且还完成了一篇后来证明是闪着智慧光辉的拓扑学论文。但所有这些兴趣之中，他对人类的思维研究工作最为严肃。当他还是一名学生时，至少在他看来，在这个世界上或者在科学的世界里只有三个有趣的问题。“遗传学似乎很有趣，因为没有人知道它是如何工作的，”他说道。

“但我不确定它是否深刻。物理学问题似乎是深刻但可以解决的。从事物理研究可能是个不错的选择。但是智力问题看起来绝对无比深刻。除此之外我不认为还有任何其他值得做的事情。”

## 明斯基其人：幽默，笑容爽朗，酷爱音乐

在后来的几年里，我没有和明斯基保持联系，但大约一年前，当他意识到技术方面的新事物在逐渐吞没我们之后，决定去找他请教。他已经进入了一个现在被称为人工智能，或者 A. I. 的领域。在那之前，它甚至没有正式的名称。（“人工智能”一词通常被认为是由明斯基在麻省理工学院的前同事 John McCarthy 提出的，他是麻省理工学院的一名数学家，现在是斯坦福大学计算机科学教授，他在 19 世纪 50 年代中期创造了这个词来描述某些机器做一些人们称之为智能（intelligent）的事情。1958 年，麦卡锡和明斯基在麻省理工学院创建了人工智能小组。在谈话中，明斯基非常健谈，颇具幽默感，带着灿烂的笑容。明斯基是我遇到过的头脑最清晰的人之一，他能够用简单的语言阐明最复杂的想法。我们在他 M. I. T. 的办公室和波士顿附近的家中进行。他的妻子 Gloria Rudisch 是波士顿当地一位著名儿科医生，他们和 18 岁的双胞胎孩子朱莉和亨利，住在一幢宽敞的房子里。Minsky 最年长的孩子玛格丽特，二十三岁，毕业于 M. I. T.，现在正在研究航天和设计家用电脑的教育计划。

他家里的乐器装饰品透露出他对于音乐的热爱，杂乱的书房里有一台计算机终端。这个国家的 AI 研究人员可以通过这个他们于 1969 年创建

的网络进行交流。他还用一台被记者误以为音响的机器向我演示如何生成复杂的音乐。明斯基告诉他，几年前，明斯基把一箱计算机模块带回家用于构建逻辑电路。他在调试电路时遇到了麻烦，因为他没有示波器——一种能够在屏幕上显示电路行为的仪器，而且他发现如果他非常快速地运行计算电路并将其连接到扬声器，他可以通过声音来分辨是否有错误发生。

“我将几个扬声器连接到电路上，通过声音我可以判断触发器是否已经无效。”触发器是一种可以找到两个稳定位置之一的电子元件。

有一天，他的朋友，一位 M. I. T. 的计算机科学教授 Edward Fredkin 来拜访时，对这个物件十分感兴趣，后来创立了一家公司把这个机器当作玩具来卖。

## 创建 M.I.T. 人工智能实验室

明斯基在 M. I. T. 人工智能实验室的办公室同样拥挤不堪。这里有一个机器人塑料雕像，还有不可或缺的计算机终端。该实验室拥有自己的大型计算机，多年来，该计算机上运行了任何人所能想到的几乎所有编程。它可以打开实验室的门，召唤大楼内的电梯；它装有机械手臂，还有特殊的电视摄像机，模拟视觉和无线电发射机，以操作遥控机器人。它曾经赢得过象棋比赛，上面还放着得来的奖杯。最初，该实验室位于一栋设有第二次世界大战电子实验室的摇摇晃晃的建筑物中，但自 1963 年以来，它已被安置在一栋俯瞰技术广场的现代化九层建筑的第三层楼上，与 M. I. T 校园隔街相望。大约有一百人在这里工作，其中包括七位教授，其中大多数是明斯基的前学生；约二十五名研究生；以及明斯基亲切地称之为黑客的一群人。这些黑客大多是进入 M. I. T.，并迷恋计算机的一群人。有些人从来不打算拿到学士学位，但有几个人获得了高等学位。

有一天，明斯基带我参观了 A. I. 实验室，并告诉了我它的发展历史。当他和麦卡锡成立人工智能小组之初，小组仅有他们两人和几个学生。大约一年后，当 Minsky 和 McCarthy 在 M. I. T. 的走廊里讲话时恰巧被当时学校的电子研究实验室主任 Jerome Wiesner 偶遇，交流之后三

人感觉兴趣相投。麦卡锡当时正在启动一个计算机分时系统，并且还创造了一种新的非常复杂的计算机语言，而明斯基开始尝试让计算机去做非数字任务，比如近似推理。 Jerome 向他们提供了所需的资金。几年来，他们从未写过研究计划。然而自此事情发生了变化，提出书面申请后，实验室每年可以从各个政府机构那里获得 250 万美元的资金支持。1968 年，当该小组正式成为人工智能实验室时，明斯基成为其董事，直到 1973 年，他一直担任这项工作，后来他厌倦了编写资金申请提案，将董事职位转交给他以前的学生之一 Patrick Winston。

## 第一台微型人工智能计算机

在实验室，我注意到一幅约 6 英尺的巨幅画像，乍看上去就像是街道规划图，然而明斯基告诉我这其实是计算机芯片的线路图，它是专为人工智能工作而设计的第一台微型计算机的重要组成部分。这台电脑是由 Gerald Sussman 和他的一些学生设计的，Sussman 曾是明斯基的学生，现在是 M. I. T. 的电气工程教授。在三楼，我看到了实际的芯片。它的面积不到半英寸，比其电路图大约小十万倍，我们不得不把它放在显微镜下才能看到电路线。计算机芯片上有两条电路线交叉的晶体管，每个晶体管的约为 7 微米，相当于红色血细胞的大小。下一代晶体管将只有其四分之一大小。

在三楼，他还向我展示了一台由他设计和建造的计算机。1970 年，他坚信可以生成自动视觉显示的计算机是很有价值的教辅工具。“所以我设计的这台电脑能够在屏幕上每秒制作 200 万个点，可以产生足够逼真的动画效果。”相比之下，典型的爱好电脑每秒只能画几千个点。明斯基称他的计算机为 2500，寓意他认为它所在的学校价值 2500 美元。一年来，他沉浸在设计中，并像阅读小说一样学会看电路图，了解了两百多种电脑芯片的工作原理。他的工作得到了斯坦福大学人工智能实验室的帮助，该实验室是 John McCarthy 于 1963 年成立的，开发了一些程序，可以自动分析短路和有其他缺陷的电路图。明斯基在自己的电脑控制台上

使用这些程序，在办公室里设计了这台机器。它所需的三百个芯片从德州仪器订购，光电路图就花费了二十四页图纸。“连接电脑曾经是一项艰巨的任务，”明斯基指出。“但所幸我们可以把整个东西放在磁带上，这样就可以通过自动布线机来读取信息。完成之后，我们必须插入三百个芯片并连接电源，键盘和电视屏幕。这并不是那么容易，但它证明了一小撮人和一个有用的计算机设计程序可以比一个大型工业设计部门做得更好。”

而此时，来自南非的数学家 Seymour Papert 加入到 AI 实验室。他发明了一种孩子们可能会喜欢的机器语言——logo。明斯基向我展示了如何使用它让机器在其显示屏上绘制各种多边形，并使它们像螺旋桨一样旋转。他有一段时间没有使用该程序，期间显示器还因数据不足而暂停。

20 世纪 70 年代初，Minsky 和 Papert 联合创办了一家公司来销售这种机器，但是最后因为资金断裂而破产。在过去的一年中，从事 logo 语言工作的人们设法找到了将其编程到家用电脑中的方法，Minsky 和 Papert 再次尝试将它推广到儿童中，因为机器现在已经变得足够便宜，供学校使用 购买。明斯基认为，几年后，他们应该变得和原来的 2500 一样强大。

## 郁闷的中小学生涯，快乐的哈佛时光

明斯基于 1927 年 8 月 9 日在纽约出生。

和所有数学天才一样，明斯基并没有数学方面的背景，他的父亲是一名眼科医生，母亲是一名犹太复国主义狂热者，上有一个建筑师和画家的姐姐，下有一个身为疾病控制咨询顾问的妹妹。

但他的智力水平在 5 岁时经历的一次智力测试中就得以体现，也因此根据测试结果进了一个公立天才儿童实验班。

在小学、初中、高中阶段，这位数学天才也和普通人一样经历过校园欺凌和嘲笑，甚至还因为字写的难看被要求留级，父母觉得理由不合理让他匆忙转学。

在菲尔德斯顿高中（Fieldston），明斯基对科学的兴趣开始萌

芽，并于 1941 年进入布朗克斯科技高中（Bronx High School of Science），这是一所培养对科学感兴趣的年轻人的学校，该校曾培养了 1979 年诺贝尔物理学奖两位得主 Steven Weinberg 和 Sheldon Glashow。在这里，他接触到了计算机先驱 Russell Kirsch，哈佛的应用数学和信息资源教授 Anthony Oettinger，以及人工智能先驱 Frank Rosenblatt，他发明了 Perceptron，但不幸于 1971 年在一次沉船事故中丧生。

1945 年，明斯基应征参加美国海军，他在五大湖海军训练中心练就了高超的射击技巧，并在结束服役后于 1946 年 9 月进入哈佛。

在这里，他学会了高等微积分，同时对社会学和心理学、神经学非常感兴趣，并提出了一些学习机器学习过程的理论。

他还做了一个有趣的实验项目，用一台仪器连接上小龙虾的爪，通过刺激特定神经纤维来控制爪的张合来抓取物体。这激发了他对机器人工具的兴趣，在这个领域尝试为手术等建立更好的显微操作器。虽然在这个领域几十年来一直没有取得太多的进展，他还是决心坚持下去。

提起大学时光，明斯基称给他留下最深刻印象的同学是只比他大几岁的数学家 Gleason，他用了几年时间解开了当时被认为是世界上最难解的难题之一——希尔伯特第五问题，这也让明斯基第一次意识到数学是一个可以跨越几乎所有困难的阶梯。

结束了在哈佛的快乐时光后，明斯基来到普林斯顿数学系，导师是 Solomon Lefschetz。

## 第一台学习机器

在这里，他遇到了志同道合的电子系同窗 Dean Edmonds，并说服哈佛的 George Miller 从伦敦海军研究局处获得资金，进行了研发具有学习能力的电子学习机器的项目。“这台机器有三百个电子管和很多电机，一些我们自己加工的自动电动离合器。机器的存储器存储在其控制旋钮的位置（其中 40 个），当机器学习时，它使用离合器来调整旋钮。我们用

一架 B-24 轰炸机的自动操纵装置来移动离合器。“明斯基讲道。

毫无疑问，明斯基的机器是世界上第一台电子学习机器之一，也许是第一台。然而，这台机器除了神经元和突触及其内部记忆回路之外，许多网络都是随机连接的，因此无法进行预测。一个“老鼠”会在网络中的某个点产生，然后开始学习到某个特定终点的路径。首先，它会随机进行，然后通过使机器更容易再次进行选择来增强正确的选择，从而增加其执行的可行性。通过灯光布置，实验者可以跟踪老鼠的进展。“事实证明，由于我们设计中的问题，我们可以将两三只老鼠放在同一个迷宫并追踪它们，”明斯基告诉说道。“这些老鼠实际上是相互影响的。如果其中一只找到了一条好路径，其他老鼠会跟上它。我们惊讶于它的小神经系统可能会同时发生多项活动。由于布线是随机的，这反而带给系统一种故障安全特性，其中一个神经元停止工作也不会产生很大影响。而且，由于有近三百个电子管和我们焊接的数千个连接，到处都可能出现问题。”

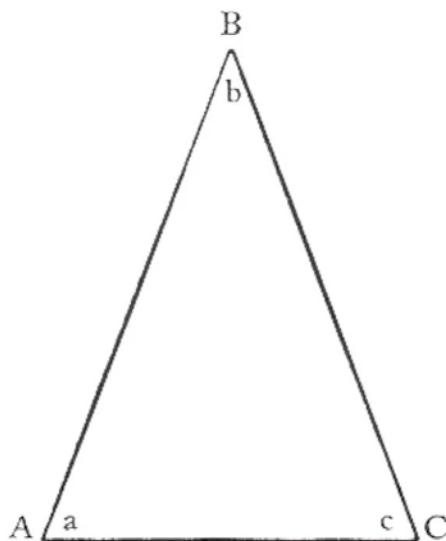
由于这台机器几乎是操作性条件反射的，而且不能进行预测，学习能力有限，它在当时并没有引起多大的重视。然后，我对机器进行了二次记忆的改动，这样就可以进行预测。当机器处理新出现的情况时可以搜索记忆，如果过去的记忆中有相关的“不愉快经历”，系统会学会尝试另外的路径。“我曾经天真地以为，如果建立一个足够大，足够多回路存储器的网络，这台机器就可以具有想象力。这在之后成为一个研究方向，被称为自组织随机网络。但我没有能力构建这样的网络。“当宾夕法尼亚大学莫尔电气工程学院已经创造出第一台电子数字计算机时，明斯基却没有在计算机上模拟运行他的学习机器。据他所说是出于两点考虑：第一，担心机器学习的复杂性；第二，认为他的机器对于研究任何方式的学习来说都还不够大。尽管如此，他还是凭借一篇神经系统学习方式的论文获得了 Ph. D.。

## 在哈佛成立人工智能团队

之后，明斯基在哈佛开始建立自己的人工智能团队，共三十多人。

两年后，即 1956 年，正值历史上的大事件——达特茅斯人工智能夏季研究项目启动。此前一年，明斯基和他的三位同事——他在普林斯顿的同窗 John McCarthy、IBM 实验室的信息研究 bu 部门经理 Nathaniel Rochester 和贝尔电话实验室的数学家 Claude Shannon，一起向洛克菲勒基金会提出应该召开一次被麦卡锡称之为“人工智能”的会议。洛克菲勒基金会认为这个提议十分有趣，并为这次会议赞助 7500 美元资金。不用多说，之后这些参加会议的人都成了人工智能领域的大人物。

明斯基说道，当时让他感到震惊的有两件事，一是 IBM 公司的工程师当时已经制作出了 701 计算机，已经具有数百个神经；另一件事是 1959 年，IMB 的年轻物理学家第一次用计算机证明了一条计算机定理。



之后，明斯基沉迷于用机器学习证明一些数学原理问题，并成功地用机器学习的毒液逻辑证明了上图中简单的 BA 与 BC 相等的问题。

## 足够复杂的计算机语言

在这之前几个月，兰德公司和卡内基技术学院的 Allen Newell、J. C. Shaw 和 Herbert Simon 已经发明了一种被称作 I.P.L. (information-processing language) 的语言，John McCarthy

提出将之与 IBM 的 fortran 语言相结合，创造出一种可以编写几何定理的语言，Gelernter 付诸实践，创造出 flpl 语言。几年之后，麦卡锡又结合 I.P.L. 和 flpl，以及其他人的思想发明了新语言——lisp (list-processing)，成为下一代计算机研究语言。1959 年，他的这一语言已经可以实际应用。

1957 年，明斯基加入 MIT 林肯实验室，与研究计算机图像识别第一人 Oliver Selfridge 成为同事，并于同年与麦卡锡成立 AI 小组。麦卡锡在之后的四年里对计算机科学的研究一定程度上奠定了 AI 领域的基础，其中一项就是计算机分时系统，即后来被普遍使用的多人可以同时使用一台计算机处理任务的方法。但在当时，几乎没有认为这是一件重要的事情，很多人分不清分时和多任务处理的差异，他们花了很长时间才说服老同学的公司 Digital Equipment Corporation 采用了分时处理器，不久，他们有了第一台商用分时计算机，Digital Equipment Corporation 也成为最大的计算机公司之一。然后我们决定分时共享 M.I.T. 的计算机。效果出人意料地好，因此 M.I.T. 长期每年从国防部高级研究计划局的计算机科学研究中心获得三百万美元资金。”

## 晶体管问世

像明斯基和麦卡锡这样的人在计算机革命中发挥如此大的作用，部分原因是由于晶体管的发明，以及高级计算机语言的发展，即使是年幼的孩子学习也几乎没有问题。晶体管是由 John Bardeen Walter H. Brattain 和 William Shockley 于 1948 年发明的。自问世以来，晶体管已经以许多不同的方式发展。

1954 年，市场上出现第一台晶体管收音机，它们是由印第安纳波利斯的工业发展工程联合公司制造的，在商业上取得巨大的成功。1959 年，飞兆半导体公司开发出第一款集成电路。这款电路中，晶体管中加入硅芯片，被这些晶体管通过像铝这样的导电材料彼此连接，因为铝比铜更容易附着到硅上。1961 年，数字设备公司销售了第一台小型计算机，并

于 1963 年制造出带有半导体元件的电子袖珍计算器，但直到 1970 年代才开始批量生产，费用随之大幅下降。

1979 年，根据罗切斯特大学计算机科学系主任 Jerome A. Feldman 统计，仅在美国，当时就有超过 150 多种用于各种用途的编程语言。对于简单的数值计算，大多数语言几乎同样适用；实际上，1963–64 年由达特茅斯的一个小组开发的 basic（初学者通用符号指令代码）是小型家用计算机中使用最广泛的语言，可以完成家用电脑所需的几乎所有任务。

（大多数人似乎就是用电脑玩游戏，而游戏的程序是现成的）。这些小型计算机的内存非常小，最多只有 6508 比特，所以最先进的计算机语言不能得到充分利用。有人开始察觉到人工智能领域所需的程序复杂性要比这些大得多。对于这些程序，fortran 和 basic 都不够复杂。当 fortran 问世时，每一比特的计算机内存花费超过一美元。今天，人们以 6 美元左右的价格可以购买 65000 比特内存的电路芯片，价格下降约一万倍。下一代个人电脑应该可以为用户提供最先进的计算机语言。但明斯基认为，将来最有用的语言将是人工智能编写的程序。这个想法是让每个人，不仅是程序员，用非正式的术语来描述他想要一个程序做什么，或仅仅通过展示程序编写程序的一些例子，让计算机编写一个程序来完成所描述的任务。这个过程比聘用专业程序员要便宜得多。

## 国际象棋编程

在 20 世纪 50 年代，已经有很多人在研究国际象棋编程，包括对人工智能先驱人物阿兰·图灵。当时最好的国际象棋程序是 Belle，由贝尔实验室的 Ken Thompson 和 Joe Condon 创建，其后是西北大学的 David Slate 和 Lawrence Atkin 发明的 Chess 4.9，Belle 的得分为 2,200，Chess 4.9 的得分为 2,050。总的来说，这些程序并不真正模仿人类象棋大师，人类的国际象棋大师可以或多或少地了解一个位置的总体结构，然后分析数步三或四步棋路。

## 第一台模式识别计算机

1959 年，明斯基的高中同学 Frank Rosenblatt 发明了第一台能够处理复杂模式识别的计算机。

## AI 小组的第一台机器人

1963 年，南非的 Seymour Papert 来到 MIT，明斯基和他开启了一项研究人类意识、儿童心理学、实验机器人、计算理论的项目，并计划制造一台能够处理复杂问题的“智能”计算机。这一研究就是十多年，在第十个年头，明斯基改变了事事亲为的做法，用资金大力招募人才，招揽了一大批来自各个领域的人才来进行这项研究。

研究过程遇到很多问题，与此同时，明斯基和他的同事们参与了另外两个项目：计算机语言学和机器人。他们在计算机上尝试的最早的非数字项目之一是语言翻译。结果一言难尽，部分原因是因为机器对语法的理解还不够，以及词语固有的模糊性，简单的逐字翻译可能会得出荒谬的结果。

明斯基对机器人项目一直兴趣颇深。1962 年，跟着 Minsky 和 Claude Shannon 学习的 Henry Ernst 发明了人工智能小组的第一台电脑控制机器人。这是一个带有肩膀，肘部和手爪的机械臂，基本上就是用来远程操纵放射性材料的手臂，可以放置和抓取物体。

## 人工智能的定义

当时，人工智能领域内外对“人工智能”的定义发生激烈的争论。该领域工作人员最普遍接受的观点是，人工智能是生产并输出与人类头脑相似甚至最终无法区分的结果的机器。终极人工智能可以执行所有的认知功能。那么，“机器”又是什么呢？几乎所有的这个领域的工作者似乎都认为机器就是某种数字计算机。

在这方面，存在着一个自 19 世纪以来就广为人知的观点，即虽然真

正的计算机可能有无数种，但理论上只有一种。这个概念来源于阿兰·图灵，他构思了一种他称之为抽象通用计算机的东西，原则上它可以被编程模仿所有其他计算机。这台通用计算机可以执行所有计算机模型可以执行的所有操作。

人工智能的目标之一是建立一台计算机，它与人类的智力不相上下，人类能用思维玩象棋和跳棋、做数学研究、写音乐和读书，理想的机器也必须能够做到所有这些事情，至少达到和人类一样的水平。显然，制造这样一台机器是一项艰巨的任务，也许是不可能的。和面临所有非常复杂问题的科学家一样，从事人工智能工作的人们一直在试图各个击破完成这项任务，因此，他们制造机器，包括硬件和必要的程序，“理解”报纸内容、识别图像。不可否认，那些机器已经可以不同程度地完成这些事情，但问题在于这意味着什么。我们是否正在接近更好地理解人的思维？即使我们能够建造一个人形机器，也会有人认为它并不真正理解自己在做什么，它只是在模拟信息，并且永远不可能超越现实。

对此，明斯基认为，“我相信，对于‘智能’的理解，应该是拥有人类思维中可以理解特定事物，以及可以理解为什么了解这些事物的能力。如果你想学习一些知识，最重要的是要知道你的大脑中哪个部分擅长学习那种东西。我不太注重统一的一般理论。我正在寻找一种可以解释大脑的各部分是如何拥有足够的理解能力去解决所面临所有问题的理论。我有兴趣研究各种简单的学习机器的方法，其中每一种方法都以互相学习彼此擅长解决的问题为主题。最后，我希望它们能够形成一个闭环，让所有的部分都能找到优化本身的方法。至少，这是我的一个梦想。”



## 想成为教育领域的阿里云，要分几步走？

采访 & 撰稿 | Natalie

在刚刚结束的两会第三场全国代表通道中，科大讯飞董事长刘庆峰在回答记者提问时，十分激动地表示：人工智能正在改变教育，让学习变得越来越有趣，科大讯飞的语音产品已经进入多个省市的学校，帮助众多学生取得更好的成绩。

这两年，教育可以说是 AI 落地场景中最为热闹的场景之一，不同公司分别从各种角度切入市场。有的公司以拍照搜题打开市场、更专注数理化科目，比如我们上一期采访的学霸君；有的公司更关注语音测评，比如科大讯飞，和我们今天文章的主角先声教育。

但只是侧重不同科目，足以在竞争激烈的 AI 教育市场圈地为王吗？这到底是一个怎么样的市场，小公司又该如何突破？

### 热闹的 AI+ 教育市场，千篇一律的“个性化教学”

2017 年中国家庭教育消费白皮书指出，教育支出占家庭年收入的 20% 以上。为了不让孩子输在起跑线上，中国家长们非常舍得在孩子的教育上花钱，各式教育辅导机构层出不穷。



这两年，AI+ 的大风向教育行业，在线教育市场的膨胀速度越发惊人，而各大投资机构更是闻风起舞。前百度研究院院长林元庆在创办自己的 AI 公司时也将教育行业列入重点领域，他认为，相对于其他行业来说，教育行业的业务场景可能是理解起来难度最低的，同时它涉及到的 AI 技术相对来说较为全面（考验综合能力而非单点技术）。

《2017 年教育行业蓝皮书》显示，截至目前，中国在线教育机构已达 4.5 万家，涵盖外语学习、K12 教育、应试学习、职业教育、语言教育等方面。2017 年教育领域的融资数量超过 450 起，入场的投资机构近 200 家，其中不乏顶级投资机构：红杉、经纬、华平、GGV、招商局、远翼资本、IDG、鼎晖、云锋基金等。而其中覆盖时间长、用户规模大（超 1.8 亿）且是刚性需求的 K12 (kindergarten through twelfth grade, 即幼儿园到高中毕业)，是最有爆发力的细分赛道之一。2017 年，K12 领域 AI+ 创业公司的融资堪称数字接力赛，亿级美元融资层出不穷：学霸君 2017 年 1 月获得 1 亿美元 C 轮融资；猿辅导 5 月获得 1.2 亿美元 E 轮融资；8 月份作业帮获得 1.5 亿美元 C 轮融资；不到一周，VIPKID 就刷新了前者记录，获得 2 亿美元 D 轮融资。

在这个热闹非凡但也明显有些拥挤的 AI+ 教育市场中，各家公司到底做得怎么样？我尝试翻阅了几家比较知名的 AI+ 教育公司的官网，发

现公司介绍中出现得最多的不外乎“1 对 1 辅导”、“搜题”、“自动批改”、“个性化试题”、“自适应学习”，虽然有的公司主打数理化，有的公司主打英语，但看上去宣传语都大同小异。这不免让人困惑，如果大家做的事情都差不多，这么多公司集体涌入这个市场的意义在哪里？那么多 AI+ 教育公司之间到底有何差异？初创公司如何脱颖而出？

带着这些疑问，我接触到了秦龙，先声教育 CTO。

秦龙是一个对自己正在做的事情充满了热情和信心的典型技术人，当他笃定地说出：“我们把自己定位为教育领域的阿里云。”时，我更加好奇了：这家成立不到两年，名称中不提“智能”、连“科技”都没有的公司，打算怎么实现这个看上去很不简单的目标？

## 从 Duolingo 到先声教育

在联合创办先声教育之前，秦龙是 Duolingo 的资深技术专家。在 Duolingo 的工作经历对秦龙回国创业有着深远的影响。Duolingo 是一个提供免费语言学习课程的平台，在全球拥有超过 2 亿用户。Duolingo 从创立伊始就以 free education 为基础，这是所有员工认同的一个准则，也是公司文化最重要的一部分。在 Duolingo 的经历让秦龙看到，高质量的免费教育是如何帮助到普通人，让很多人的不可能成为可能。

国内的学生对于英语词汇和语法的学习非常充足，但是对于听说和写作能力的锻炼还非常少。而且这方面的资源特别匮乏，大部分学校和家庭无法创造足够的机会和场景帮助学生锻炼听说和写作能力。因此，秦龙决定回到国内，通过技术改变这一现状。

秦龙告诉我们，先声教育创办的初衷是通过人工智能技术为中国绝大多数家庭提供最好的教育。目前国内存在越来越严重的教育资源分配不均衡的问题，先声教育希望通过技术减轻这一问题对于大多数学生的影响。

而之所以选择语音识别和语音测评作为切入点，是因为一方面语音技术经过多年的发展，已经逐渐被大众所接受；另一方面，英语口语确实对很多学生是个很大的问题，而且学生想在口语方面得到好的指导非常困

难，好的资源非常匮乏。语音测评恰好就是教育场景中能够和 AI 结合得非常好的一项应用，这也就成了先声教育选择的切入点。

## 先声教育背后的 AI 技术支撑

### 以智能测评为切入点

先把相对成熟的技术与教育场景的结合做好

根据对于市场和技术成熟度的分析，先声教育首选的方向是智能测评，这主要是为了提高老师和学生在英语学习过程中作业环节的效率，一方面把老师从繁重的作业批改工作中解放出来，另一方面，提高学生做作业的效率和效果，可以使学生在第一时间得到作业结果的反馈。

最早上线智能语音测评技术主要出于两方面考虑：

- 语音测评技术相对更加成熟；
- 语音测评在教育中是高频应用，并且已经逐渐被老师和学生所接受。

先声教育的很多合作客户主要面向 K12 市场，因此技术上需要解决的问题包括：

1. 语音识别和测评的用户年龄跨度很大，从小学到高中，尤其是小学和初中的用户，他们的声道发育还未完成，也就是我们常说的尚未完成变声阶段，音质和成人有很大的差别。
2. 处理噪音和作弊的问题。有些用户在很嘈杂的环境下使用，有些用户的麦克风比较差，或者还有的学生想作弊，比如说中文或者是弄出来其他声音，需要确保在不同条件下测评的准确性。

同智能语音测评类似，智能写作批改也属于智能测评的一部分。相比语音测评，英文写作的频率没有那么高，但是对于老师和学生来说存在与语音测评同样的痛点。而先声教育在智能写作批改中一项比较大的突破就是测评不需要任何范文。

目前一些主流的写作批改技术，是建立在对于需要批改的作文题目有

大量从低分到高分平均分布的范文作为训练数据的。但这个要求会严重限制老师和学生的使用，老师只能在已有的题目中挑选或者需要想办法找到大量范文。为了解决这一问题，先声教育通过大量的数据训练出一个常规打分模型，同时再训练另一个模型来判断学生提交的作文是否是题目所要求的内容。如此一来，不需要用户提供任何范文，就可以对作文题目进行打分。

对话机器人和情感识别是为了解决用户没有实际英语使用场景的问题，为用户打造一个纯英语环境，通过浸入式的语言学习环境，帮助学生提高英语水平。自适应学习则是为了进一步提高学生的学习效率，将时间使用在最需要提高的知识点上。目前先声教育这两方面的技术仍处于研发阶段。

## 基础平台架构

目前先声教育的语音测评模型和作文批改模型训练使用的是自己搭建的计算平台，主要基于 GPU 和 TensorFlow。

云端部署主要涉及以下工作：

根据访问量动态部署服务器的数量，以更高效地利用计算资源，每天最高峰和最低谷服务器数量相差 10 倍。

在语音测评过程中会保存大量的音频文件，对于这些音频文件，一方面要让用户能够快速获取，比如用做录音的回放；另一方面也需要将文件转移到一个能长久保存的地方。为了实现这两个目标，采用了 CDN、Redis 等服务。

测评的同时会产生大量的日志文件，为了可以更好地读取和分析日志，先声教育还搭建了一个 Elasticsearch 集群。

在客户端，先声教育提供支持不同开发环境的 SDK，覆盖 iOS、Android、Web、Linux & Windows Server 和微信端，客户通过调用 SDK 接口进行语音或者写作的云端测评。目前 SDK 支持离线调用和在线调用两种方式，当用户的设备无法连接网络时，SDK 自动切换到离线调用，不

影响用户体验。

## 智能口语评测：以不变应万变

好的评测 = 双指标 + 多维度

口语测评主要关注两方面的指标，一个是评分的准确性，一个是反馈信息的全面性和有效性。对于评分的准确性，通常会使用 Pearson 相关度来考量，具体地说，就是计算人工专家打分和机器打分的相关度，相关度越接近 1 说明评分越准确。目前先声教育的语音测评相关度可以达到 0.93 以上。

对于学生而言，评分不是终点，通过学习提高英语口语能力才是最终目的。为了帮助学生达到这个目标，先声教育的智能语音测评技术在提供准确打分的同时，还会提供非常详尽的共计 15 个维度的教学场景反馈，使学生从错误中得到成长。

“我们既提供整个句子或者篇章的总体打分，同时也提供包括完整度、流利度、准确度在内的每个维度的分数。我们还会反馈小到音素级别的音素发音分、音素检错信息，单词发音分、单词重音检错，句子语调、语速、停顿、重复、遗留单词等信息。”

秦龙告诉我们，发音的准确性仅仅是英语口语的一个方面，韵律度和节奏感也是非常重要的。因此先声教育提供的很多反馈信息，都是为了提高学生说英语的节奏感的，比如语速、停



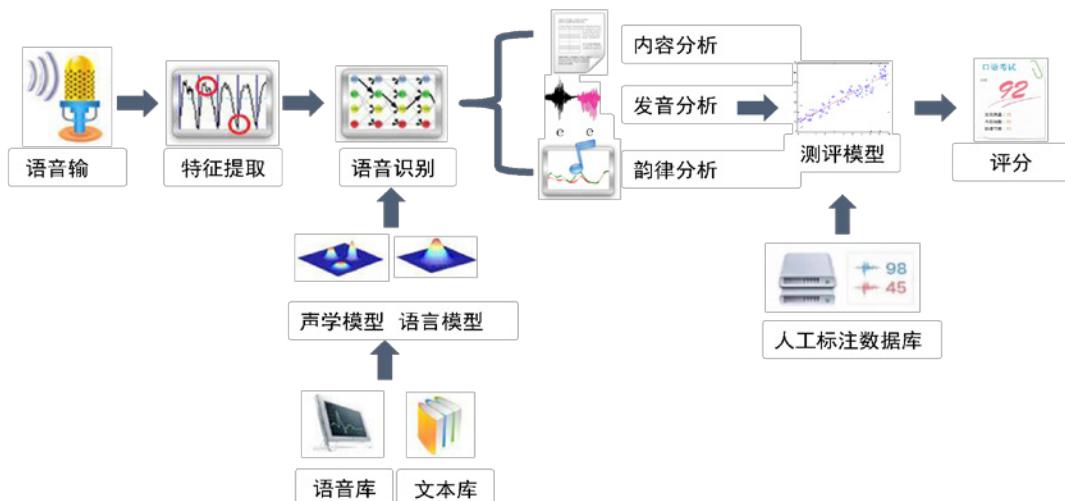
### 语音测评技术演示

顿、重读、语调等。

### 选择效果最好的算法模型

口语测评一般包括两个环节，首先通过语音识别对用户的录音进行分析，提取一些声学和韵律等各方面的特征，然后再通过训练好的测评模型估计用户的评分。

在语音识别部分，先声教育使用的技术框架和主流的语音识别系统基本一致，目前使用最多的是基于回归神经网络（LSTM）的声学模型。语音信号是一个时序序列，而且我们对一个词 / 音素的发音会受到这个词所处的上下文影响。深度回归神经网络，尤其是长短记忆门网络（LSTM），很好地通过网络结构的设计，不是仅仅根据当前输入进行计算，而是对当前输入的前后状态同时进行考虑。同时，LSTM 对语音信号间大跨度的依赖性有很好的建模，非常适合语音识别和测评的应用。



### 语音测评原理

语音测评模型一般使用回归模型估计评分，从简单的线性回归，到 boosting，到 SVM，到神经网络，都有使用。如何从众多模型中做出最佳选择，这主要取决于训练数据的多少。同时，在部分使用场景，比如当在用户的手机或者平板电脑上做离线测评时，会根据设备的计算能力调整模

型的复杂度。

### 低延迟、高准确性，但并不完美

目前先声教育智能口语评测的实时性已经达到比较高的水平，在线测评的延时为毫秒级，主要的时间都是用于语音的传输。为了提高实时性，先声教育在传输语音时，一般使用 WebSocket 协议，也就是常说的边录音边传输边测评，等用户的录音结束了，系统的测评也随之结束。如果使用离线测评，基本感觉不到任何延时，测评就已经完成。

从评分的准确性上来说，目前先声教育的语音测评技术已经达到甚至超过了人工评分的表现。对于人类专家来说，多个专家的评分相关度一般在 0.7-0.8 之间，而先声教育的智能语音测评准确性为 0.93。这其中的原因包括：1. 每个人测评的严格程度往往不同，有的人比较严格，而有的比较宽松。2. 即使对于同一个人，也会受到时间、心情等因素影响，造成评分幅度的变化。

当然，先声教育的语音测评系统也并不完美。

秦龙坦诚：“目前的测评系统对于极端情况的测评还不够精细，对于口语特别差以及口语特别好的情况，评分的精细度有时还不能满足要求。比如满分 10 分的时候，9 分和 10 分语音的测评有时候还是会出现偏差。这方面可以通过搜集更多的低分和高分的语音数据，然后优化评分模型去解决。”

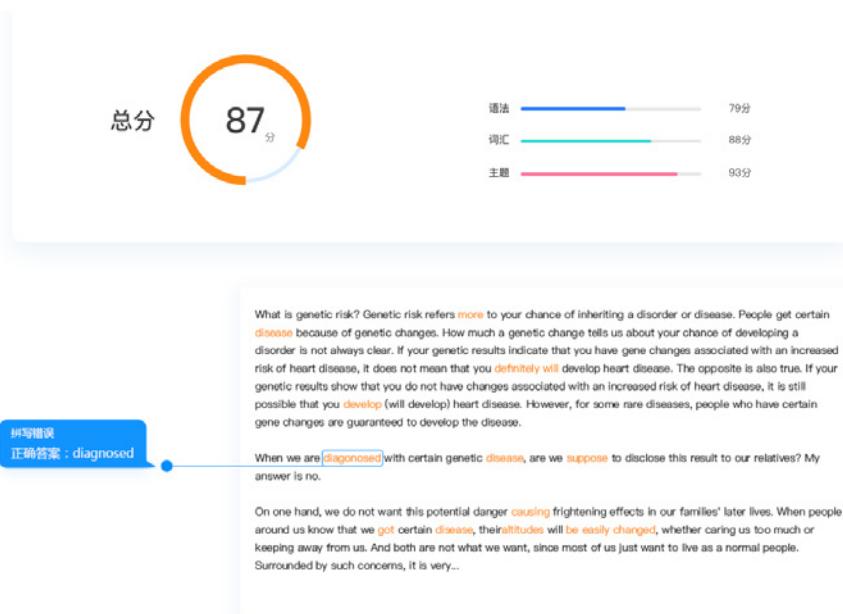
“从场景上来说，目前的语音测评主要是针对某一点进行评估，比如某句话学生的表现如何。这样有两方面的缺失，一个是对于同一个学生，跨时间的数据追踪；另外一方面，我们还需要对学生的口语水平做一个全面的评估。通过不断积累同一用户的更多数据，持续地全面评估用户的口语水平。这将是我们下面工作的一个重点。”

### 智能写作批改：无需大量范文

目前一种比较常用的作文批改方法需要大量的从低分到高分的范文，在批改新的作文的时候，使用同 KNN (K nearest neighbors) 非常接近

的思想，计算同已有范文的距离，然后根据距离加权得到作文的评分。这一方法的主要问题就是需要大量的范文作为评分前提，而这在实际应用中往往是比较困难的。

先声教育的作文批改没有采用这个技术方案，而是将作文拆解为写作质量（包括语法、词汇正确性等）和主题吻合度分别进行评分。当前的做法是训练一个模型来判断写作的质量，同时训练另外一个模型来判断文章是否点题。在训练第一评分模型的时候，使用来自所有作文题目的数据训练一个深度神经网络，这个模型的作用是判断一篇文章的好坏，主要基于词汇的使用、语法、文章结构等判断维度。训练主题模型的时候，则使用作文的题目和学生的写作一起训练。评分模型和主题模型输出的后验概率一起被作为一个 softmax 函数的输入，而输出为文章最终的得分。

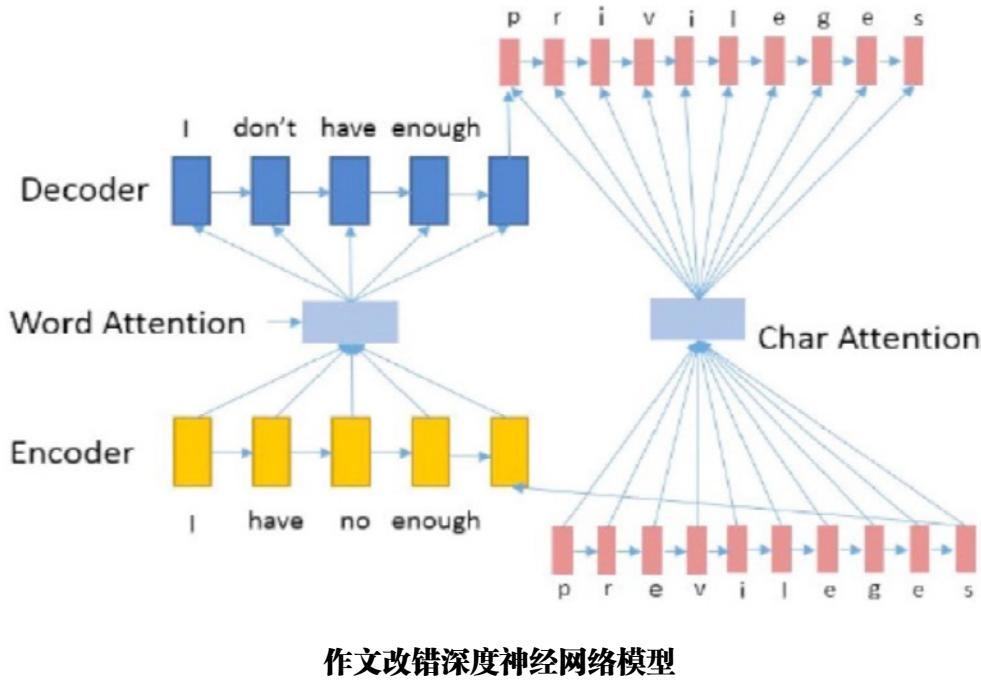


### 写作批改技术演示

但同一篇文章，放到不同的年级，可能评分结果就会有所不同。如果在智能写作批改中，需要按不同的年级、不同的打分标准，分别用不同的数据进行单独的模型训练，未免太过低效。

针对这一问题，先声教育目前的做法是将学段作为模型的输入参数之一，然后使用不同学段的数据一起训练评分模型。在有足够的数据的

时候，可以针对不同的学段分别训练独立的评分模型进行打分。秦龙解释到：“当然，也不是说不同学段的数据不能复用。在训练我们的改错模型时，我们就会使用所有的数据一起建模，这个模型用来寻找语法和用词的错误。在这个场景下，是否错误的标准是唯一的，所以不会有学段的限制。”



### 情感计算和自适应学习

除了有条不紊地迭代已经成熟的语音测评系统，升级日渐成熟的写作批改系统，先声教育也在尝试更多，比如情感计算和自适应学习系统。

目前在这两个方面，先声教育都还处于起步阶段。对于情感计算，先声教育主要是从语音信号和文本两个方面来做情感分类。语音信号主要的特征来自韵律，比如语音的基音频率的均值和方差、语音的能量等。对于文本来说，则需要从词汇、短语和主题三个方面来判断。

秦龙指出：“目前公开的一些用于情感情识别的语音和文本数据往往是网页或者是商品的评价数据。这些数据库标注的情感标签并不一致，没有统一的标准。另外，我们需要的是针对 K12 教育领域的情感标注数据。

比如枯燥或者走神，并不是一个常用的情感分类，但对于教育而言却是非常重要的。”

传统的自适应学习主要是使用基于贝叶斯模型的知识追踪方法。但是这一模型比较简单，无法利用大量的训练数据。最近几年，有一些学者开始尝试使用深度学习的方法去做知识追踪。这种方法需要大量的训练数据来训练深度神经网络，然后通过神经网络自动的去抽象出来每个题目所对应的知识点。但是这种方法的有效性目前学界还存在争议，而且在工业界也没有被验证。

人类对于知识点的记忆是有半衰期的，学生对于知识点的掌握程度不仅取决于历史上考察该知识点的习题的对错，也取决于学生上一次复习该知识点的时间。因此，先声教育目前正在尝试基于贝叶斯知识追踪模型和半衰期回归模型相结合的方法。同时，秦龙认为，要达到很好的自适应学习效果，需要算法、内容和产品的高度结合，因此还需要与用户进行更多的沟通和合作。

## 先声教育的底气

先声教育目前的商业模式以 ToB 为主，剑指“教育领域的阿里云”。如今他们已有的语音测评和写作批改技术就是以云计算的形式提供给合作伙伴，因此这个目标与先声教育作为教育领域人工智能技术服务商的身份倒也吻合。秦龙告诉我们，他们希望可以像阿里云一样成功，通过向教育企业提供人工智能技术服务，提高教育能效，从而使客户可以更好地为学生和老师服务。

那么先声教育想做“教育领域阿里云”的底气源自于哪里？

据秦龙介绍，目前先声教育语音测评云服务已经拥有超过 20 家企业合作伙伴，覆盖了从线下培训机构、到在线教育企业、到智能教育硬件厂商等多个行业，包括美联英语、纳米盒、步步高等。云平台日均调用量超过 1000 万次，每秒能够同时处理超过 1 万个并发测评请求，每天处理的全国各地各个年龄段的用户的语音数据超过 1 万小时。

这样的一组调用量数据可以说是相当不错了。这背后当然离不开前面提到的一系列技术服务，秦龙透露：“我们希望能够做到，客户需要什么 AI 技术，我们就能够提供相应的服务。在语音测评和写作批改方面，我们已经可以向用户提供最好的技术。”

在提供最好的技术的同时，秦龙认为，企业还需要成为合格的“AI 技术产品经理”。“我们要教育客户如何最好地使用我们的技术，这有点类似云服务厂商的架构师会帮助客户搭建最好的后台架构，在这里，我们除了要帮助用户建立一个好的架构以外，我们还希望理解客户的产品需求和使用场景，从产品层面去帮助用户。”

最后但也是最核心的一点，是数据闭环。数据是 AI 非常重要的基础，尤其在医学 AI 领域，精确标注的数据可以说是相关创业公司的核心竞争力，这一点在教育 AI 领域也是如此。实际上，秦龙表示，在绝大多数场景，数据对于 AI 技术都是至关重要的。

目前先声教育的数据主要是来源于其云平台的用户。通过从云平台每个客户收集到的语音和写作数据，先声教育能够重新训练评分模型，然后使用更新后的更加准确的模型服务所有的客户。每个客户的原始数据只对客户本身开放，而训练出来的统计模型可以用来为所有客户服务。

“通过这样的合作，能更好地优化核心算法，由客户提供使用场景，我们搜集更多的数据，形成了一个从算法到场景到数据的闭环，从而保证我们可以不断的优化测评系统的性能。”秦龙说。

基于 ToB 的商业模式，先声教育可以接触不同类型的客户，比如培训机构、在线学习企业、智能硬件厂商等，而这些客户又覆盖各种各样的学习场景，拥有各个学段各个地区的学用户，进而帮助先声教育搜集到非常多样化的语音和写作数据，这些都是训练一个高精度的测评模型不可或缺的原料。同时，针对特定的问题，也可以很容易地找到大量的对应数据，从而使这些问题的优化和解决变得更加容易。“海量覆盖各个场景、地区、学段的数据越来越成为我们的核心竞争力。”

目前先声教育以英语为主攻方向，对于其他学科还没有明确的计划。

对于秦龙来说，每个学科所面临的问题是不同的，也没有难易之分，在对所面临的场景和问题的细致分析以后，一定有很多问题都是可以通过 AI 技术来解决的。

关于下一步公司的技术布局和发展计划，秦龙也做了颇多考虑。

“我们把自己定位为教育领域的阿里云，那么，我们就不可能仅仅是提供语音测评或是写作批改，这样 1-2 个人工智能技术服务。我们需要的是向客户提供一整套的人工智能解决方案。我们需要通过不同的技术解决客户各种各样的痛点，同客户紧密合作。因此，我们会逐步研发和上线更多的人工智能技术，比如今年我们的重点就是自适应学习和对话技术。”

有了最好的核心技术和服务闭环，接下来最重要的事情就是如何基于客户需求打造更多 AI 技术，并从后台架构以及产品两个层面更好地帮助客户。

## 征途才刚刚起步

2012-2015 年主要是互联网 + 教育的阶段，涌现了很多 O2O 的教育公司，解决了从线下到线上的问题。而 2016 年到现在则是 AI+ 教育的阶段，解决的是通过人工智能技术提高教育能效的问题。实际上，只有过去比较成熟的产品才会想到通过 AI 技术来打造更好的效果。

在秦龙看来，目前 AI+ 教育还处于早期摸索阶段，虽然在某些问题上已经有了比较成功的应用，但是离“革命”还有很多工作要做。AI 的产业化，AI 和商业或者传统行业的结合才刚刚起步。但与此同时，秦龙也认为 AI 技术本身还有很大的发展空间，未来和产业的结合一定会对我们的生活产生巨大的改变。

“如今很多教育企业，包括初创企业，已经越来越多将 AI 技术的使用作为产品的最基本功能了。或者说，AI 技术越来越成为教育产品的标配。这是一个很强烈的趋势，我觉得就像云服务取代传统的自建机房服务器一样，越来越多的企业会引入人工智能技术。”

但未来 AI 充分发展之后，是不是老师的角色就会被 AI 替代了呢？秦龙的答案是 No！“未来 AI+ 教育，我不认为是简单的替代或者颠覆老师或者学校，我认为更多的是 AI+ 老师的形式。老师从知识的传授者的角色慢慢变成一个教学设计者、一个监督的角色，更多的工作会交给 AI 去做，老师会更注重对于学生人格和思想的培养。”

先声教育作为一家垂直于 K12 领域的人工智能技术服务商，难免会面对“AI 公司就是外包公司”的调侃，对此秦龙到并不在意。他认为，“是否外包公司与做不做 AI 没有绝对关系，如果一个公司没有自己确定的发展方向，客户要什么就做什么，或者说每个客户需求都是不同的，那这就很有外包的特征。AI 公司要能够提供一个标准化的解决方案，多配置少定制。”

最后，我又向秦龙追问了一个问题：

**您觉得先声教育距离“教育领域的阿里云”这个目标还有多远？**

秦龙斟酌再三给出回答：“总的来说还是起步阶段。我们算是走通了技术输出的模式，现在做的是需要提供更多技术，并且每个技术要更加深入。”



策划 & 编译 | Natalie

本文是阿里巴巴 PAI-Blade 团队发表于 TVM 的最新博文，文中阐述了如何将 TVM 引入 TensorFlow，使 TensorFlow 中的 batchmul 速度提高 13 倍，同时将端到端神经机器翻译性能提高 1.7 倍。AI 前线对原文进行了编译。

TVM 是由华盛顿大学在读博士陈天奇等人提出的深度学习自动代码生成方法，该技术能自动为大多数计算硬件生成可部署优化代码，其性能可与当前最优的供应商提供的优化计算库相比，且可以适应新型专用加速器后端。



Tianqi Chen  
@tqchenml

正在关注



Nice blog on using TVM stack to speed up  
batchmul in @TensorFlow for 13x and end to  
end neural machine translation for 1.7x  
[tvmlang.org/2018/03/23/nmt...](http://tvmlang.org/2018/03/23/nmt...)

© 翻译自英语

上午8:53 - 2018年3月23日

11 转推 42 喜欢



## 背景

神经机器翻译（NMT）是一种端到端的自动翻译方法，有可能克服传统短语翻译系统的不足。最近，阿里巴巴集团正在尝试将 NMT 服务部署到全球电子商务业务场景中。

目前，我们正在尝试开发 Transformer<sup>[1]</sup>作为我们 NMT 系统的核心组件，因为它比基于 RNN/LSTM 的经典模型更有利于高效的离线训练，具有同等（甚至更高）的精确度。虽然 Transformer 打破了时间步长之间的依赖关系，因此对于离线训练阶段非常友好，但是对于在线推理来说，就没有那么高效了。在我们的生产环境中，初始版本的 Transformer 推理速度比 LSTM 版本慢大约 1.5 倍到 2 倍。为了提高推理性能，我们已经尝试进行了多种优化，如图级别 op 融合、循环不变式节点运动<sup>[3]</sup>。我们发现批量 matmul 是 Transformer 中的一个性能关键点，而当前 cuBLAS 的实现并未进行很好的优化。

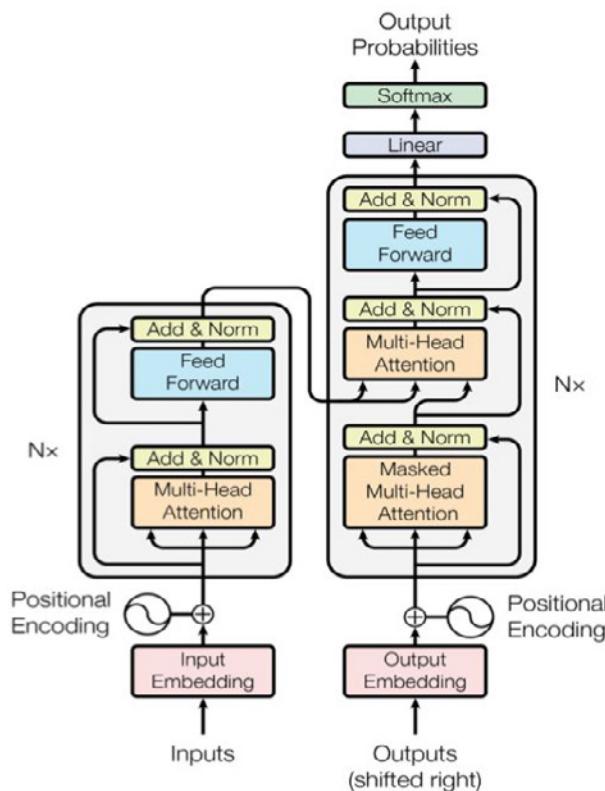
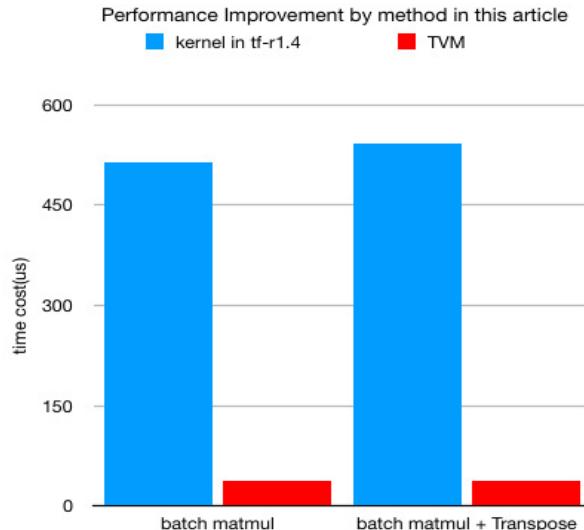


Figure 1: The Transformer - model architecture.

我们的实践结果表明，通过 TVM 生成的内核（优化了 schedule）可以使批量 matmul 计算速度提高至少 13 倍，并且在启用了算子融合的情况下还将进一步提高速度。



## 批量 Matmul

### 为什么要使用批量 matmul

在 Transformer 中，批量 matmul 被广泛应用于多头 attention (multi-head attention) 的计算。利用批量 matmul，attention 层中的多头可以并行运行，有助于提高硬件的计算效率。

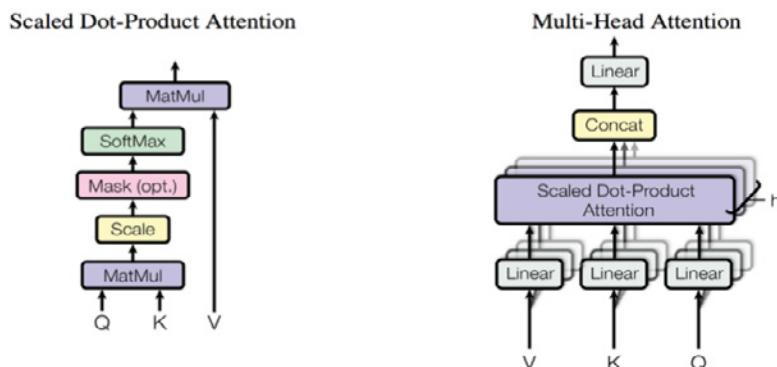


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

我们在推理阶段对 Transformer 模型进行了深入的剖析，结果表明批量 matmul 计算占 GPU 内核执行时间的 30% 左右。我们利用 nvprof<sup>[2]</sup>对 cuBLAS 的批量 matmul 内核进行了第一性原理分析，结果表明当前的实现效果很差，同时我们还发现了一些有趣的现象。

## 什么是批量 matmul

通常，批量 matmul 计算是指在一批矩阵上执行矩阵 - 矩阵乘法。其中 batch 被认为是“均匀的”，即所有实例对于它们各自的 A、B 和 C 矩阵具有相同的维度 (M、N、K)、前导维度 (lda、ldb、ldc) 和转换率。

批量 matmul 计算可以更具体地描述如下：

```
void BatchedGemm(input A, input B, output C, M, N, K, batch_dimension) {
    for (int i = 0; i < batch_dimension; ++i) {
        DoGemm(A[i], B[i], C[i], M, K, N)
    }
}
```

## 批量 matmul 的 shape

在语言翻译任务中，批量 matmul 的 shape 明显小于其他工作负载下的正常 matmul 计算。Transformer 的 shape 与输入句子的长度和解码器步骤的长度有关。通常会小于 30。

当给定推理的 batch size 时，批量维度 (batch dimension) 是一个固定值。例如，如果将 batch size 设定为 16，beam 大小为 4，则批量维度为 16\*4\*#head (多头 attention 中的头数，通常为 8)，矩阵 M、K、N 的 shape 在 [1, 最大解码长度] 或 [1, 最大编码长度] 的范围内。

## cuBLAS 批量 matmul 的性能问题

首先，我们对批量 matmul 内核进行了 FLOPs 理论分析。结果相当有趣：所有批量 matmul 的计算强度都非常有限（小于 1TFLOPs）。

接着我们利用 nvprof 软件对不同 shape 的批量 matmul 的 cuBLAS 进行了分析。下表显示了使用 CUDA 8.0 在 NVIDIA M40 GPU 上得到的一些指标。

input shape [batch, M, N, K]	kernel	theoretical FLOPs	nvprof observed FLOPs	theoretical FLOPs / observed FLOPs
[512, 17, 17, 128]	maxwell_sgemmBatched_128x128_raggedMn_tn	18939904	2155872256	0.87%
[512, 1, 17, 128]	maxwell_sgemmBatched_128x128_raggedMn_tn	1114112	2155872256	0.052%
[512, 17, 1, 128]	maxwell_sgemmBatched_128x128_raggedMn_tn	1114112	2155872256	0.052%
[512, 30, 30, 128]	maxwell_sgemmBatched_128x128_raggedMn_tn	58982400	2155872256	2.74%

即使使用不同的 shape (改变 M, N, K), 所有的 Maxwell\_sgembatched\_128x128\_raggedMn\_tn 调用都会执行相同数量的 FLOP, 而且实际值比理论值大得多。可以推断, 所有这些不同的 shape 可能都被填充成了特定的 shape。在所有这些 shape 中, 即使在最好的情况下, 理论 FLOP 仍仅占实际执行 FLOP 的 2.74%, 因此大部分计算相当冗余。类似地, 调用另一个 cuBLAS 内核的 Maxwell\_sgembatched\_64x64\_raggedMn\_tn 出现了同样的现象。

显而易见, cuBLAS 的批量 matmul 实现效率很低。因此, 我们使用 TVM 为 NMT 工作负载生成高效的批量 matmul 内核。

## 批量 matmul 计算

在 TVM 中, 通常批量 matmul 计算可以声明为:

```
# computation representation

A = tvm.placeholder((batch, M, K), name='A')

B = tvm.placeholder((batch, K, N), name='B')

k = tvm.reduce_axis((0, K), 'k')

C = tvm.compute((batch, M, N),

    lambda b, y, x: tvm.sum(A[b, y, k] * B[b, k, x], axis = k),

    name = 'C')
```

## Schedule 优化

在对批量 matmul 计算进行声明之后，我们需要仔细设计Schedule，以充分挖掘其性能潜力。

调参：块 / 线程数

```
# thread indices

block_y = tvm.thread_axis("blockIdx.y")
block_x = tvm.thread_axis("blockIdx.x")
thread_y = tvm.thread_axis((0, num_thread_y), "threadIdx.y")
thread_x = tvm.thread_axis((0, num_thread_x), "threadIdx.x")
thread_yz = tvm.thread_axis((0, vthread_y), "vthread", name="vy")
thread_xz = tvm.thread_axis((0, vthread_x), "vthread", name="vx")

# block partitioning

BB, FF, MM, PP = s[C].op.axis
BBFF = s[C].fuse(BB, FF)
MMPP = s[C].fuse(MM, PP)
by, ty_block = s[C].split(BBFF, factor = num_thread_y * vthread_y)
bx, tx_block = s[C].split(MMPP, factor = num_thread_x * vthread_x)
s[C].bind(by, block_y)
s[C].bind(bx, block_x)
vty, ty = s[C].split(ty_block, nparts = vthread_y)
vtx, tx = s[C].split(tx_block, nparts = vthread_x)
s[C].reorder(by, bx, vty, vtx, ty, tx)
s[C].reorder(by, bx, ty, tx)
s[C].bind(ty, thread_y)
s[C].bind(tx, thread_x)
s[C].bind(vty, thread_yz)
s[C].bind(vtx, thread_xz)
```

我们融合了批量 matmul 的外部维度，即 op 维度的 BB 和 FF，在批量 matmul 计算中通常称为“批量”维度。然后我们用因子 (number \_ thread \* vthread) 分割外部维度和内部维度。

批量 matmul 中不需要 stridged 模式，因此虚拟线程数（vthread\_y 和 vthread\_x）都设置为 1。

### 寻找 number\_thread 的最佳组合

下图是使用 CUDA8.0 在 NVIDIA M40 GPU 上得到的结果。

Input Shape [batch,features,M,N,K]	num_thread_y, num_thread_x	num_vthread_y, num_vthread_x	Time(us)
[64, 8, 1, 17, 128]	8, 1	32, 1	37.62
[64, 8, 1, 17, 128]	4, 1	32, 1	39.30
[64, 8, 1, 17, 128]	1, 1	32, 1	38.82
[64, 8, 1, 17, 128]	1, 1	256, 1	41.95
[64, 8, 1, 17, 128]	32, 1	1, 1	94.61

根据以往的经验，寻找 num\_thread\_y 和 num\_thread\_x 最佳组合的方法是暴力搜索。通过暴力搜索，可以找到当前形状（shape）的最佳组合，我们在当前计算中得到的最佳组合为 num\_thread\_y=8, num\_thread\_x=32。

## 将批量 matmul 与其他运算融合

当前的“黑盒”cuBLAS 库调用通常会被作为常用“op 融合”优化策略的边界。然而，利用生成的高效批量 matmul 内核，可以很容易地打破这一融合边界，不仅可以融合 element-wise 运算，从而进一步提高性能。

从计算图中可以看出，批量 matmul 之后总是会进行广播加法运算或转置运算。通过将“加法”或“转置”运算与批量 matmul 融合，可以减少内核启动开销和冗余内存访问时间。

批量 matmul 和广播加法融合计算可声明如下：

```
# computation representation
A = tvm.placeholder((batch_size, features, M, K), name='A')
# the shape of B is (N, K) other than (K, N) is because B is transposed in this
# fusion pattern
B = tvm.placeholder((batch_size, features, N, K), name='B')
ENTER = tvm.placeholder((batch_size, 1, M, N), name = 'ENTER')
k = tvm.reduce_axis((0, K), 'k')
C = tvm.compute(
```

```
(batch_size, features, M, N),  
lambda yb, yf, m, x: tvm.sum(A[yb, yf, m, k] * B[yb, yf, x, k], axis = k),  
name = 'C')  
  
D = topi.broadcast_add(C, ENTER)
```

批量 matmul 和转置融合计算可声明如下：

```
# computation representation  
  
A = tvm.placeholder((batch_size, features, M, K), name='A')  
B = tvm.placeholder((batch_size, features, K, N), name='B')  
k = tvm.reduce_axis((0, K), 'k')  
  
C = tvm.compute(  
    (batch_size, M, features, N),  
    lambda yb, m, yf, x: tvm.sum(A[yb, yf, m, k] * B[yb, yf, k, x], axis = k),  
    name = 'C')
```

## 融合后的内核性能

我们选择 [batch=64, heads=8, M=1, N=17, K=128] 的形状来测试前文生成的代码的性能。我们将序列长度设置为 17，因为这个值是我们生产场景中的平均输入长度。

测试结果如下：

- tf-r1.4 BatchMatmul: 513.9 us
- tf-r1.4 BatchMatmul+Transpose (separate): 541.9 us
- TVM BatchMatmul: 37.62 us
- TVM BatchMatmul+Transpose (fused): 38.39 us

内核融合优化进一步将速度提高了 1.7 倍。

## 与 TensorFlow 集成

在我们的工作负载中，批量 matmul 的输入形状是有限的，并且易于预先枚举。利用这些预定义的形状，我们可以提前生成高度优化的 CUDA 内核（固定形状计算可以带来最佳的优化潜力）。同时，我们还将生成适用于大多数形状的通用批量 matmul 内核，为没有提前生成内核的形状提供回退机制。

我们将生成的针对特定形状的高效内核和回退机制集成到了 TensorFlow 框架中。我们开发了融合 op，如 BatchMatMulTranspose 或 BatchMatMulAdd，以使用 TVM 的运行时 API 为特定输入形状启动特定的生成内核，或调用回退内核。通过执行图形优化通道，可以用融合 op 自动替换原始的批量 matmul+ 加法 / 转置模式。同时，通过结合更激进的图形优化通道，我们尝试利用 TVM 为长尾操作模式生成更高效的融合内核，以进一步提高端到端性能。

## 总结

在阿里巴巴内部，我们发现 TVM 是一个非常高效的工具，我们可以用它开发高性能的 GPU 内核以满足内部需求。

本文以 NMT Transformer 模型为例，对 TVM 的优化策略进行了详细说明。首先，通过第一性原理分析找出 Transformer 模型的关键问题。然后使用 TVM 生成高度优化的 CUDA 内核来替换 cuBLAS 版本（此时已经得到了 13 倍的加速效果）。接下来，我们利用 TVM 的内核融合机制来融合批量 matmul 的前 / 后操作，从而带来进一步的性能改善（性能进一步提高 1.7 倍）。在此基础上，我们开发了一个图形优化通道，自动用 TVM 融合内核替换原有的计算模式，保证优化过程对最终用户透明。作为 AI 基础设施提供商，我们发现透明度对于推广优化策略的应用非常重要。

最后，所有这些优化都以松散耦合的方式集成到了 TensorFlow 中，展示了将 TVM 与不同深度学习框架集成的一种可能方式。此外，我们目前还在进行将 TVM 整合为 TensorFlow 代码生成后端的工作，希望今后能够与社区分享更多成果。

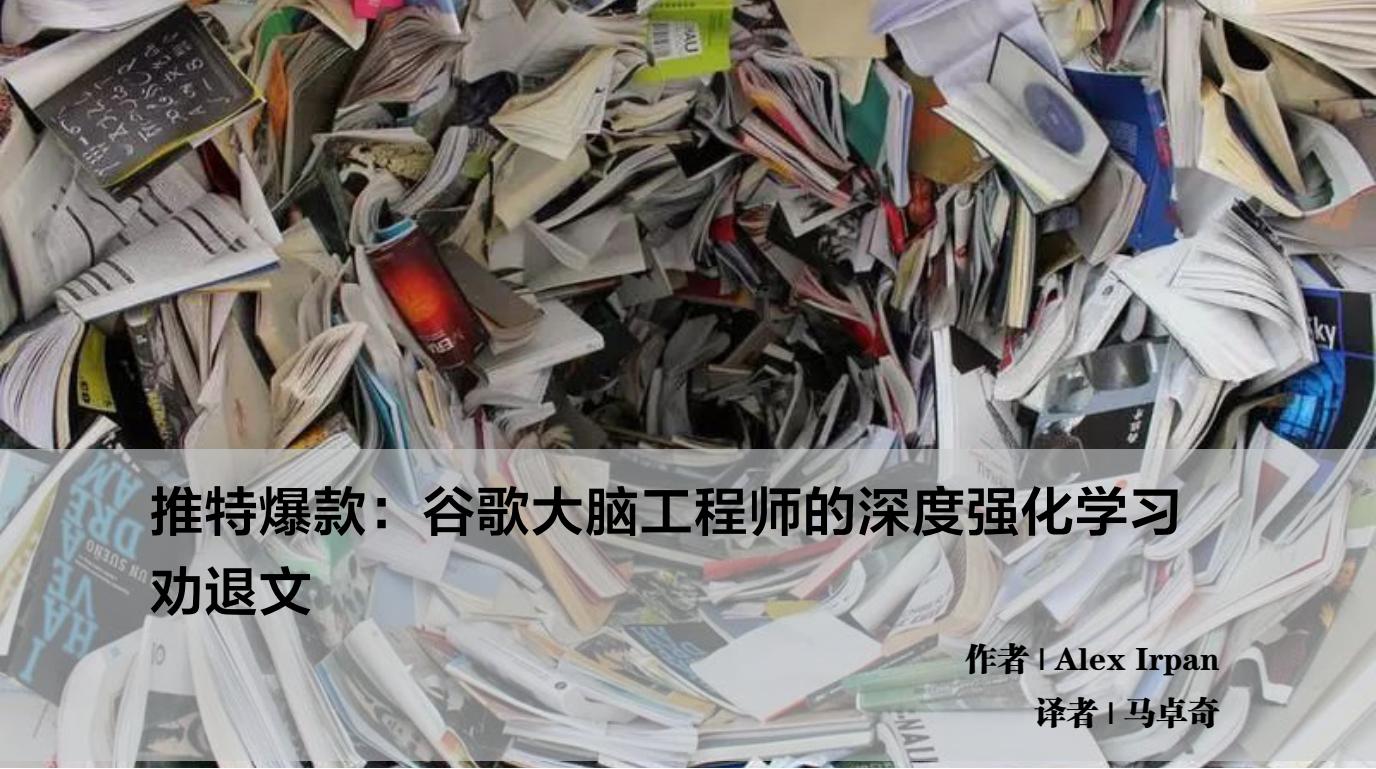
## 资源

- TVM implementation of fused batch matmul + transpose computation [项目代码](#)

- 
- [PAI 团队中文介绍](#)

## 参考文献

- [1] [Attention is All You Need](#)
- [2] [nvprof is Your Handy Universal GPU Profiler](#)
- [3] [Add Loop Invariant Node Motion Optimization in GraphOptimizer](#)



# 推特爆款：谷歌大脑工程师的深度强化学习 劝退文

作者 | Alex Irpan

译者 | 马卓奇

深度强化学习可以说是人工智能领域现在最热门的方向，它之所以声名大振，与 DeepMind 团队用它在 AlphaGo 和 AlphaZero 上大获成功脱不开关系。

强化学习本身是一个非常通用的人工智能范式，在直觉上让人觉得非常适合用来模拟各种决策任务。当它和深度神经网络这种只要给足Layer 和神经元，可以逼近任何函数的非线性函数近似模型结合在一起，那感觉就是要上天啊！

但本文作者 Alex Irpan 想要告诉大家，深度强化学习是个大坑，别着急进坑！它的成功案例其实不算很多，但每个都太有名了，导致不了解的人对它产生了很大的错觉，高估能力而低估了难度。Alex Irpan 目前是谷歌大脑机器人团队的软件工程师，他从伯克利获得计算机科学本科学位，本科时曾在伯克利人工智能实验室（Berkeley AI Research (BAIR) Lab）参与科研，导师是深度强化学习大牛 Pieter Abbeel。

这篇文章在推特上非常火爆。知乎网友 Frankenstein 在看完英文

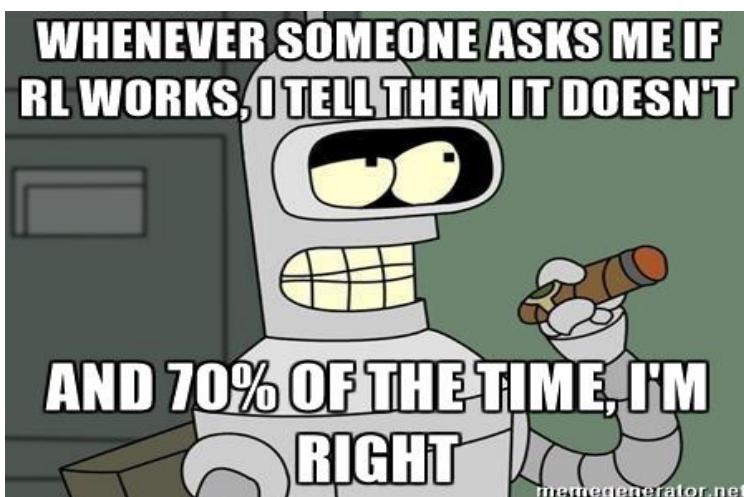
原文后表示“读完大有如不见此文，万古如长夜之感。心里激动，一时难以自抑。这篇文章是我入坑以来看到的深度强化学习方面最好的阶段性总结，强烈建议应该作为深度强化学习的第一课，看完以后大家再慎重考虑到底要不要入坑。”

AI 前线觉得很有必要将这篇文章介绍给更多国内的小伙伴。虽然标题写着劝退文，但并非真的要完全劝退大家，而是希望大家更加冷静地看待目前深度强化学习的研究进展。

## 写在前面

有一次在 Facebook 上，我发表了这样一段话：

任何时候，如果有人问我强化学习能不能解决他们的问题，我都会告诉他们，不行。而且 70% 的情况下我都是对的。



深度强化学习被大量的炒作重重包围。这并不是无缘无故的！强化学习是很奇妙的一种范式，原则上，一个鲁棒且高性能的 RL 系统应该是可以完成一切任务的。将这种范式与深度学习的经验力量相结合明显是最佳搭配。深度 RL 看起来是最接近通用人工智能（Artificial general intelligence, AGI）的系统之一，而打造 AGI 这一梦想已经消耗了数十亿美元的资金。

不幸的是，**目前深度强化学习仍然不起作用。**

当然，我相信它是有用的。如果不相信强化学习，我也不会研究它。但是研究过程中也出现了很多问题，而且大部分都很难解决。智能体 agent 光鲜亮丽的演示背后，是大家看不到的无数鲜血、汗水和泪水。

我已经看到过很多次人们被这个领域的研究所诱惑。他们只不过第一次尝试深度强化学习，并且刚好没遇到问题，于是就低估了深度 RL 的难度。如果不经历失败，他们不会知道这到底有多难。深度 RL 会持续不断地打击他们，直到他们学会如何设定符合现实的研究期望。

这不是任何人的错，而是一个系统性问题。如果结果是好的，自然什么都好说，但如果结果不太好，还想写出同样精彩的故事就没那么简单了。问题是，负面的结果才是研究人员最经常遇到的。在某些方面，负面案例其实比正面案例更重要。

在后面的内容中，我将解释为什么深度 RL 没有起到作用，什么情况下它能起到作用，以及它如何在未来变得更可靠。我并不是为了让人们停止研究深度 RL 才写这篇文章。我这样做是因为我相信如果大家一起找到问题所在，就更容易在这些问题上取得进展。如果大家聚在一起讨论这些问题，而不是每个人自己一次又一次地掉进同样的坑里，就能更容易取得一致的意见。

我希望看到深度 RL 研究有更好的发展，有更多的人进入这个领域。但我也希望人们了解他们即将要面临的到底是什么。

在本文中，“强化学习”和“深度强化学习”相互替代使用。我批判的是深度强化学习的经验主义行为，并不是批判强化学习整体。我引用的论文通常用深度神经网络作为 agent。虽然经验批判主义可能适用于线性 RL 或 tabular RL，但我不确定它们是否能推广到更小的问题。在巨大、复杂、高维的环境下，良好的函数逼近是十分必要的，RL 在这种环境下的良好应用前景推动了对深度 RL 的炒作。这种炒作是目前需要解决的问题之一。

这篇文章的走向是从悲观到乐观，虽然文章很长，但是如果你耐心阅读，我相信你一定会有所收获。

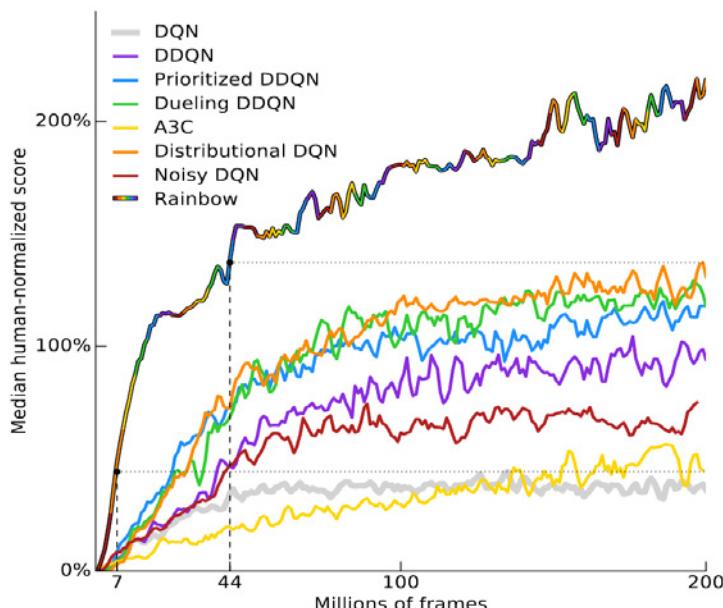
下面首先介绍深度 RL 的失败案例。

## 深度强化学习的样本效率可能极低

深度强化学习最经典的实验基准是 Atari 游戏。在著名的深度 Q 网络 (Deep Q-Networks) 论文中, 如果你将 Q 学习、大小合适的神经网络和一些优化技巧结合, 就可以让网络在一些 Atari 游戏中达到甚至超越人类水平。

Atari 游戏的运行速度是每秒 60 帧, 你能估算出最先进的 DQN 想要达到人类的水平, 需要多少帧吗?

这个问题的答案取决于游戏的种类, 我们来看看 Deepmind 最近的一篇论文, Rainbow DQN (Hessel et al, 2017)。这篇论文用模型简化法研究了对于原始 DQN 结构的几种改进, 证明了将所有的改进结合起来可以达到最好的结果。在 57 种 Atari 游戏测试中, 该模型在 40 种游戏中超过了人类的表现。下图展示了实验结果。



Y 轴是“人类标准化分数中值”, 是通过训练 57 个 DQN (每一个 Atari 游戏训练一个模型), 并以人类表现为 100%, 对 agent 的表现进行标准化得到的。以此为基础绘制了 57 场游戏表现的中位数。在 1800

万帧时，RainbowDQN 的表现超过了 100% 的阈值。这相当于大约 83 小时的游戏体验，再加上训练模型所需的时间，而大多数人玩通一个 Atari 游戏只需要几分钟。

请注意，1800 万帧其实已经很少了，考虑到之前的方法 (Distributional DQN, Bellemare et al, 2017) 需要 7000 万帧才能达到与人类相当的水平，是 RainbowDQN 所需时间的 4 倍。至于 (Nature DQN, Mnih et al, 2015)，它永远无法达到 100% 的性能，即使经过 2 亿帧之后。

规划谬误论认为完成某件事花费的时间通常比你想象的要长。强化学习有它自己的规划谬误——学习一个 policy (策略) 需要的样本通常比你以为的更多。

这个问题不只在 Atari 游戏中出现。第二常见的基准实验是 MuJoCo，是设置在 MuJoCo 物理模拟器中的一组任务。在这些任务中，输入状态通常是一些模拟机器人每个关节的位置和速度。即使不用解决视觉问题，学习这些基准也需要 10~5 到 10~7 个步骤。对于这样一个简单的环境，这一数字可以说是很大的经验量了。

DeepMind 的跑酷论文 (Heess et al, 2017)，用 64 个机器训练了超过 100 个小时来学习 policy。论文中没有说明“机器”是什么，但我认为应该是指 1 个 CPU。

论文结果很酷。刚看到这篇论文时，我没想到深度 RL 竟然可以学习这些奔跑的步伐。

不过与此同时，需要 6400 个 CPU 时的事实让人有点沮丧。并不是因为我没想到它需要这么多的时间，而是深度 RL 仍然比实际应用级别的样本效率低了几个数量级。

这里有一个明显的对比：如果我们直接忽略样本效率呢？有几种情况很容易获取经验，比如游戏。但是，在这一点不成立的情况下，RL 面临着一场艰苦的战斗，而且不幸的是，现实世界中的大多数情况就属于这类。

如果你只关心最终的性能，许多问题更适合用其他方法解决。

在寻找任何研究问题的解决方案时，通常会在不同的目标之间进行权衡。优化的目标可以是一个真正好的解决方案，也可以是良好的研究贡献。最好的情况是获得一个好的解决方案的同时需要做出好的研究贡献，但是很难找到符合这个标准并且可行的问题。

如果纯粹为了获得好的表现，深度 RL 的记录可能就没那么光鲜了，因为它总是被其他方法打败。这是一个 MuJoCo 机器人的视频，用在线轨迹优化（online trajectory optimization）控制。正确的动作是在近乎实时、在线，而且没有离线训练的情况下计算的。而且它运行在 2012 年的硬件上（Tassa et al, IROS 2012）。

我认为这篇论文可以和那篇跑酷论文相提并论。那么两者之间的区别在哪里呢？

区别在于，Tassa 等人利用模型预测控制，可以用真实世界模型（物理模型）来进行规划。而无模型的 RL 不进行这样的规划，因此难度更大。另一方面，如果对模型进行规划有所助益，那为什么要把问题复杂化，去训练一个 RL 策略呢？

同理，用现成的蒙特卡洛树搜索（Monte Carlo Tree Search, MCTS）就可以很容易地超越 DQN 在 Atari 游戏中的表现。下面是 Guo et al, NIPS 2014 论文中给出的基准数字。他们把一个训练好的 DQN 的分数和 UCT agent（UCT 就是我们现在使用的标准 MCTS）的分数做对比。

Agent	<i>B.Rider</i>	<i>Breakout</i>	<i>Enduro</i>	<i>Pong</i>	<i>Q*bert</i>	<i>Seaquest</i>	<i>S.Invaders</i>
DQN	4092	168	470	20	1952	1705	581
-best	5184	225	661	21	4500	1740	1075

这并不是一个公平的对比，因为 DQN 没有搜索步骤，而 MCTS 能够用真实模型（Atari 模拟器）进行搜索。然而有时候人们不在乎对比是否公平，人们只在乎它的有效性。（如果你对 UCT 的全面评估感兴趣，

可以看看原始的 (Arcade Learning Environment, Bellemare et al, JAIR 2013) 论文附录。

强化学习理论上可以用于任何事物，包括模型未知的环境。然而，这种普遍性是有代价的：算法很难利用特定信息来帮助学习，这迫使人们使用多到吓人的样本来学习本来通过硬编码就可以得到的规律。

无数经验表明，除了极少数情况，特定域的算法均比强化学习的表现更快更好。如果你只是为了深度 RL 而研究深度 RL，这不成问题，但是每当我把 RL 与其他任何方法进行对比时，都会觉得十分沮丧，无一例外。我非常喜欢 AlphaGo 的原因之一就是因为它代表了深度 RL 的一次毋庸置疑的胜利，而这样的胜利并不常见。

这一点让我越来越难给外行解释为什么我研究的问题又酷又难又有趣，因为他们往往没有背景或经验来理解为什么这些问题这么难。人们所认为的深度 RL 能做什么，和它真正能做什么之间横亘着一个巨大的“解释鸿沟”。我现在研究的是机器人。当你提到机器人时，大多数人们想到的公司是波士顿动力 (Boston Dynamics)。

他们可不使用强化学习。如果你看看该公司的论文，你会发现论文的关键字是时变 LQR, QP 求解以及凸优化。换句话说，他们主要用的还是经典的机器人技术，而实际结果证明，只要用对了这些经典技术，它们就可以很好地发挥作用。

## 强化学习通常需要奖励函数

强化学习一般假定存在奖励函数。通常是直接给定的，或者是离线手动调整，然后在学习过程中不断修正。我说“通常”是因为也有例外，例如模仿学习或逆 RL，但大多数 RL 方法把奖励函数奉为神谕。

更重要的是，想让 RL 做正确的事，奖励函数必须知道到底你想要什么。RL 有过拟合奖励函数的倾向，会导致意想不到的结果。这就是为什么 Atari 游戏是一个很好的基准。不仅是因为很容易从中获得大量的样本，而且是因为每场比赛的目标都是尽可能地得分，所以永远不用担心该

如何定义奖励函数，而且每个系统都用相同的奖励函数。

这也是 MuJoCo 任务很受欢迎的原因。因为它们在模拟器中运行，所以你清楚地知道所有的对象状态（state），这使得奖励函数的设计容易许多。

在 Reacher 任务中，需要控制一个连接到中心点的两段臂，任务目标是移动手臂到指定位置。下面是一个成功学习出策略的[视频](#)。

因为所有的位置都是已知的，所以奖励函数可以定义为从手臂末端到目标的距离，再加上一个小的控制损失。理论上来说，在现实世界中，如果你有足够的传感器来获取环境中精确的位置，你也可以这样做。

就其本身而言，需要奖励函数不算什么，除非……

## 奖励函数设计难

增加一个奖励函数并不难。困难的是如何设计一个奖励函数，让它能鼓励你想要的行为，同时仍然是可学习的。

在 HalfCheetah 环境中，有一个双足机器人被限制在一个垂直平面内，只能向前跑或向后跑。

[视频](#)。

任务目标是学习跑步的步态。奖励函数是 HalfCheetah 的速度。

这是一个计划奖励函数（shaped reward），也就是说在状态（state）离目标（goal）越来越近时，它增加奖励。另一种是稀疏奖励（sparse reward），即达到目标状态才给予奖励，而其他状态都没有奖励。计划奖励往往更容易学习，因为即使策略没有找到问题的完整解决方案，系统也会提供正反馈。

不幸的是，计划奖励可能会影响学习，导致最终的行为与你想要的结果不匹配。一个很好的例子就是赛龙舟游戏，（详情参考 OpenAI 博客），预定的目标是完成比赛。可以想象，在给定的时间内完成比赛，稀疏奖励函数会 +1 奖励，而其他情况都是 0 奖励。

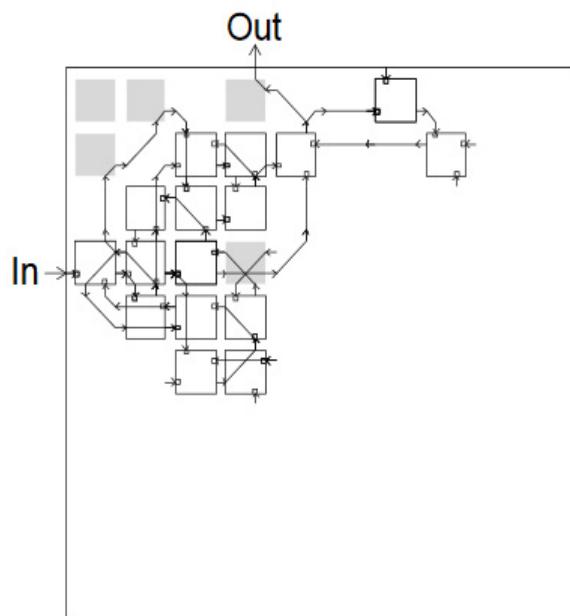
但游戏给出的奖励是只要到达关卡点就会加分，并且收集道具也会

加分，而且收集道具给的分数比完成比赛得到的还多。在这样的奖励函数下，RL 系统的 agent 即使没有完成比赛，也能拿到最高得分。因此也导致了许多意想不到的行为，例如 agent 会撞毁船、着火、甚至走反方向，但它获得的分数比单纯完成游戏还要高。

### 视频。

RL 算法是一个连续统一体，他们可以或多或少地了解他们所处的环境。最常见的无模型强化学习（model-free RL），几乎与黑盒优化相同。这些方法只有在 MDP（马尔科夫决策过程）中才能求解出最优策略。agent 只知道哪个给了 +1 奖励，哪一个没有，剩下的都需要自己学习。和黑盒优化一样，问题在于对 agent 来说，任何给予 +1 奖励的都是好的，即使 +1 的奖励不是因为正确的理由而得到的。

一个经典的非 RL 例子是应用遗传算法设计电路的时候，得到的电路中出现了一个无连接的逻辑门。



**Fig. 7.** The functional part of the circuit. Cells not drawn here can be clamped to constant values without affecting the circuit's behaviour — see main text.

所有的灰色单元都要有正确的行为，包括左上角的，即使它没有跟任何单元连接起来。（来自 “An Evolved Circuit, Intrinsic in

Silicon, Entwined with Physics” )

更多的例子可以参考 Salesforce 2017 年的博客，它们的目标是文本总结。它们的基准模型用监督学习训练，然后用自动度量方法 ROUGE 对总结效果进行评估。ROUGE 是不可微的，但是 RL 可以解决不可微的奖励，所以他们尝试直接用 RL 来优化 ROUGE。得到了很高的 ROUGE 分数，但是实际效果并不是很好。下面是一个例子：

*Button was denied his 100th race for McLaren after an ERS prevented him from making it to the start-line. It capped a miserable weekend for the Briton. Button has out-qualified. Finished ahead of Nico Rosberg at Bahrain. Lewis Hamilton has. In 11 races. . The race. To lead 2,000 laps. . In. . . And.*

所以尽管 RL 模型得到了最高的 ROUGE 评分：

Model	ROUGE-1	ROUGE-L
Nallapati et al. 2016 (abstractive)	35.46	32.65
Nallapati et al. 2017 (extractive baseline)	39.2	35.5
Nallapati et al. 2017 (extractive)	39.6	35.3
See et al. 2017 (abstractive)	39.53*	36.38*
<b>Our model (RL only)</b>	<b>41.16</b>	<b>39.08</b>
<b>Our model (supervised+RL)</b>	39.87	36.90

他们最后还是用了另外一个模型。

还有另一个有趣的例子：Popov et al, 2017 的论文，也被称为“乐高积木论文”。作者用分布式 DDPG 来学习抓握策略，目标是抓住红色块，并把它堆在蓝色块的上面。

他们完成了这个任务，但中途遇到了一个彻底失败的案例。他们对于初始的抬升动作，根据红块的高度给出奖励，奖励函数由红块底面的 z 轴坐标定义。其中一种失败模式是，策略学习到的行为是把红块倒转过来，而不是捡起来。

[视频。](#)

显然这并不是研究人员预期的解决方案，但是 RL 才不在乎。从强化学习的角度来看，它因为翻转一个块而得到奖励，所以它就会将翻转进行到底。

解决这个问题的一种方法是给予稀疏奖励，只有机器人把这个块堆叠起来后才给予奖励。有时候这样能行得通，因为稀疏奖励是可学习的。但通常情况下却是没用的，因为缺乏正向强化反而会把所有事情复杂化。

解决这个问题的另一种方法是仔细地调整奖励函数，添加新的奖励条件，调整现有的奖励系数，直到你想学习的行为在 RL 算法中出现。这种方法有可能使 RL 研究取得胜利，但却是一次非常没有成就感的战斗，我从来没觉得从中学到过什么。

下面给出了“乐高积木”论文的奖励函数之一。

$$r(b_z^{(1)}, s^P, s^{B1}, s^{B2}) = \begin{cases} 1 & \text{if } \text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0.25 & \text{if } \neg \text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \wedge \text{grasp}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0.125 & \text{if } \neg (\text{stack}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \vee \text{grasp}(b_z^{(1)}, s^P, s^{B1}, s^{B2})) \wedge \text{reach}(b_z^{(1)}, s^P, s^{B1}, s^{B2}) \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

我不知道他们到底花了多少时间来设计这个奖励函数，但是根据术语的数量和不同系数的数量，我猜是“相当多”。

在与其他 RL 研究人员的交流中，我听到了一些趣闻，是由于奖励函数定义不当而造成的新奇行为。

- 有位同事正在教一个 agent 通过一个房间。如果 agent 离开边界，这一事件就终止了。他没有对事件的这种结束方式加任何惩罚。结果最后学到的 policy 是自我毁灭式的，因为消极的奖励太丰富了，而积极的奖励很难获取，在 agent 看来，快速死亡以 0 奖励结束，比长时间活动可能造成负奖励更可取。
- 一位朋友正在训练一个模拟机械手臂伸到桌子上方的某个点。这个点是根据桌子定义的，而桌子没有固定在任何东西上。agent 学会了重重地拍桌子，使桌子翻倒，这样一来它也算移动到目标点了。而目标点正好落到了机械臂的末端。

- 一位研究人员在使用 RL 训练模拟机器手臂拿起锤子并把钉子钉进去。最初，奖励定义为钉子被钉到洞里的深度。结果，机器人没有拿起锤子，而是用自己的四肢把钉子钉进去了。于是研究人员增加了奖励项，鼓励机器人拿起锤子，然后重新训练策略。虽然他们得到了拿起锤子的策略，但是机器人只是把锤子扔在了钉子上而不是使用锤子。

诚然，这些都是“听别人说的”，但是这些行为听起来都很可信。我已经在 RL 的实验中失败了太多次，所以我不会再怀疑这一点。

我知道有人喜欢用回形针优化器的故事来危言耸听。他们总喜欢猜测一些特别失调的 AGI 来编造这样的故事。现在每天都有很多真实发生的失败案例，我们没有理由去凭空想象那样的故事。

## 即使奖励函数设计得很好，也很难避免局部最优解

前文给出的 RL 例子有时被称为“奖励作弊”。正常情况下，AI 会通过正常的手段最大化奖励函数。然而，有些情况下，AI 可以通过作弊的手段，直接把奖励函数调到最大值。有时一个聪明的、现成的解决方案比奖励函数设计者所预期的答案能得到更多的奖励。

奖励作弊是一个例外，更常见的情况是由于探索 - 利用权衡不当而导致的局部最优解。

下面是一个归一化优势函数的[实现视频](#)，在 HalfCheetah 环境中学

习。在局外人看来，这个机器人太蠢了。但是这是因为我们站在第三方的视角，而且我们知道用脚跑步要好得多。但是 RL 并不知道啊！它收到状态 (state)，然后做出行动 (action)，然后知道它得到了正向奖励，就是这样。

我猜测学习过程发生了以下几件事：

- 在随机探索阶段，策略发现向前倒比静止站着要好。
- 于是策略记住了这个动作，然后持续向前倒。

- 向前倒之后，策略了解到，如果一次性施加很大的力，它会做一个后空翻，并且得到奖励。
- 于是它试了一下后空翻，很自信这是一个好主意，然后策略牢牢记住了要多做后空翻。
- 当策略连续后空翻时，哪种方法对策略来说更简单呢？学习纠正自己然后按“标准方法”跑还是学会躺着跑呢？我想应该是第二种更简单吧。

下面是另外一个[失败示例](#)，来自 Reacher 项目。

在这个过程中，初始随机权重倾向于输出高度正或高度负的动作输出。这使得大多数动作尽可能输出最大或最小加速度。超高速旋转非常容易：只需要每个关节输出高强度的力。一旦机器人开始行走，就很难以一种有意义的方式脱离这一策略——想要脱离这个策略，必须采取几个探索步骤来阻止混乱的旋转。这当然是可能的，但在这次试验中并没有发生。

这些都是一直以来困扰强化学习的经典的探索 - 利用问题。由于数据来自于当前的策略，如果当前的策略探索太多，得到的是垃圾数据，那就什么也学不到；如果利用太多，智能体也无法学到最优行为。

有几个不错的想法可以解决这个问题——内在动机、好奇心驱动探索、基于计数的探索等等。这些方法中有许多是在 80 年代或更早的时候提出的，其中有几个在深度学习模型中重新得到应用。然而，据我所知，没有一个方法能在所有环境中持续奏效，他们总是有时有用，有时没用。如果有一个到处都能奏效的探索技巧，那就太好了，但我认为这种万能技术在很长一段时间内都不能实现。不是因为人们没有尝试，而是因为探索 - 利用困境真的太难了。

引用维基百科的一段话：

“最初二战中盟军的科学家认为，这个问题太棘手了，应该丢给德国，让德国科学家也浪费自己的时间。”（参考：[Q-Learning for Bandit Problems](#), Duff 1995）

我将深度 RL 想象成一个恶魔，故意曲解你的奖励函数，然后积极寻

找最懒的可行局部最优解。

## 当深度 RL 有效时，它可能过拟合环境中奇怪的模式

“深度 RL 广受欢迎的原因是它是机器学习中唯一允许在测试集上进行训练的研究领域”

强化学习的好处是：如果你想在一个环境中表现很好，可以随意的过拟合。缺点是，如果你想推广到其他环境中，可能表现会很差。

DQN 可以解决很多雅达利 (Atari) 游戏，但它是通过对单一目标进行集中学习来完成这项任务的——它只在某一个游戏的表现很好。但最后学到的模式无法推广到其他游戏，因为它没有经过这种训练。你可以通过微调训练好的 DQN，让它去玩新的雅达利游戏 (Progressive Neural Networks, Rusu et al, 2016)，但也不能保证它能够迁移，而且人们通常不期望它能迁移。

原则上在各种各样的环境下进行训练应该能使这些问题消失。在某些情况下，可以轻易得到这样的环境。例如导航，可以在环境中随机采样目标位置，并使用通用值函数来泛化 (Universal Value Function Approximators, Schaul et al, ICML 2015)。这是一项很有前途的研究，然而我不认为深度 RL 的泛化能力足以处理不同的任务。OpenAI Universe 试图用分布式来解决这个任务，但据我所知，目前他们也没有太多进展。

除非 RL 系统达到了那样的泛化能力，否则 agent 所学习到的策略应用范围会一直受限。举一个例子，想一想这篇论文：Can Deep RL Solve Erdos-Selfridge-Spencer Games? Raghu et al, 2017。我们研究了一个双玩家组合游戏，玩家的最佳发挥有封闭形式的解析解。在第一个实验中，我们固定了玩家 1 的行为，然后用 RL 训练了玩家 2。这样可以将玩家 1 的行为视为环境的一部分。在玩家 1 的表现最优时训练玩家 2，我们发现 RL 可以达到高性能。但是当我们针对非最佳玩家 1 部署相同的策略时，它的性能下降了，因为它不能泛化到非最佳对手的情况。

Lanctot et al, NIPS 2017 中给出了类似的结果。有两个 agent 在

玩激光枪战。这两个 agent 采用多智能体强化学习进行训练。为了测试泛化能力，他们用 5 个随机种子进行训练。下面是互相训练的 agent 的视频。

可以看到，他们学会了朝对方走并且射击对方。然后他们从一个实验中取出玩家 1，从另一个实验中取出玩家 2，让他们互相对抗。如果学到的策略可以泛化，应该能看到与之前类似的行为。

但是并没有。

[视频](#)。

这似乎是多智能体 RL 的主题。当 agent 相互训练时，会产生一种协同进化。agent 变得特别擅长攻击对方，但当他们对付一个之前没有遇到过的玩家时，表现就会下降。需要指出的是，这些视频之间唯一的区别是随机种子。它们都有同样的学习算法，相同的超参数，但是发散行为完全来自于初始条件下的随机性。

话虽如此，竞争性自我游戏环境产生了一些很好的结果，似乎跟这一点互相矛盾。OpenAI 有一篇文章介绍了他们在该领域的工作。自我竞争也是 AlphaGo 和 AlphaZero 算法的重要组成部分。我认为，如果 agent 以同样的速度学习，他们可以不断地挑战对方，加快彼此的学习，但是如果其中一个学得更快，它会过度利用较弱的选手，导致过拟合。当你将环境从对称的自我竞争放松到一般的多智能体时，更难确保学习能以同样的速度进行。

## 即使忽略泛化问题，最终结果也可能不稳定而且难复现

几乎每一个 ML 算法都有超参数，它们会影响学习系统的行为。通常，超参数都是人工挑选的，或者是随机搜索得到的。

监督学习是稳定的，有固定的数据集和真实值目标。如果你稍微改变超参数，算法的表现不会改变很多。并不是所有的超参数都表现良好，但有了过去几年发现的所有经验技巧，许多超参数在训练中可以进行优化。这些优化方向是非常重要的。

目前，深度 RL 是不稳定的，是一项十分麻烦的研究。

当我开始在 Google Brain 工作的时候，我做的第一件事就是从标准化优势函数（NAF）论文中实现这个算法。我想大概只需要 2-3 个星期。我熟悉 Theano（可以很好地迁移到 tensorflow），有一些深度 RL 经验，而且 NAF 论文的第一作者正在 Google Brain 实习，所以我有问题可以直接请教他。

然而由于一些软件错误，复现结果花了 6 个星期。问题在于，为什么花了这么久才找到这些错误？

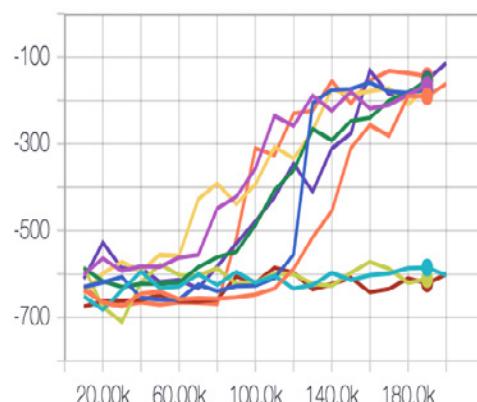
要回答这个问题，我们首先可以看看 OpenAI GYM 的最简单的连续控制的任务：摆任务。在这个任务中，有一个摆，固定在一个点上，重力作用在钟摆上。输入状态是三维的。动作空间是一维的，即施加扭矩的量。我们的目标是完美地平衡摆锤，让其垂直向上。

这是一个小问题，好的奖励函数让它变得更容易。奖励是用钟摆的角度来定义的。将钟摆向垂直方向摆动的动作不仅给予奖励，还增加奖励。奖励空间基本上是凹的。

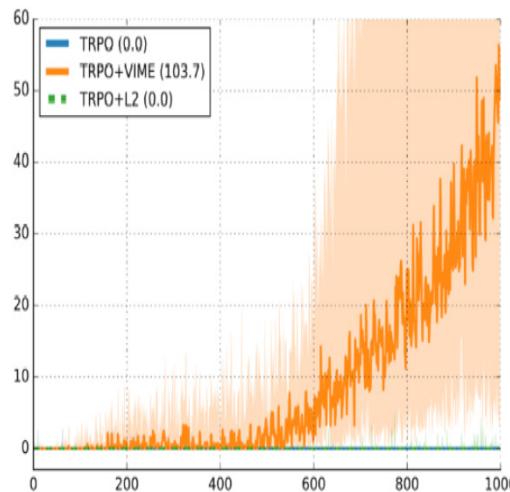
下面是一个基本有效的策略的[视频](#)。虽然该策略没有直接平衡到垂直位置，但它输出了抵消重力所需的精确扭矩。

这是表现性能曲线，在我改正所有的错误之后。每一条曲线都是 10 次独立运行中的 1 次奖励曲线。相同的超参数，唯一不同的是随机种子。

episode\_reward/test



10 次实验中有 7 次运行成功，3 次失败，30% 的失败率。下面是其他已发表工作中的曲线图，Variational Information Maximizing Exploration, Houthooft et al, NIPS 2016。环境是 HalfCheetah。我们将奖励函数修改的更稀疏了一些，但这些细节并不太重要。Y 轴是每一段的奖励，X 轴是时间间隔，使用的算法是 TRPO。

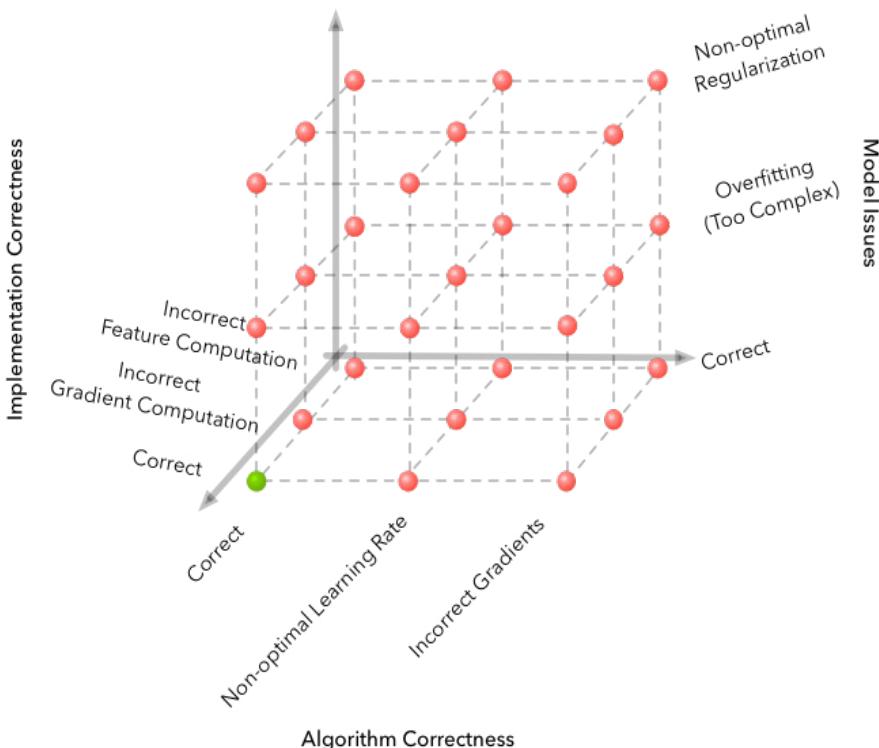


较深的曲线是 10 个随机种子表现性能的中值，阴影区域是第 25 到第 75 个百分点。这个曲线是支持论文算法的一个很好的论据。但另一方面，第 25 百分点离 0 奖励十分接近。这意味着大约 25% 的实验是失败的，只是因为随机种子。

监督学习也会有变化，但变化不会这么大。如果有监督学习的代码运行结果有 30% 低于随机结果，我会很自信肯定是数据加载或训练过程哪里出错了。如果我的强化学习代码比不过随机结果，我不知道是有地方出错了，还是超参数不对，或者我只是运气不好。

这张照片来自《Why is Machine Learning “Hard”？（为什么机器学习这么“难”？）》。核心论点是机器学习给失败情况空间增加了更多维度，从而成倍地增加了导致失败的原因。深度 RL 又在此基础上增加了一个新维度：随机性。你唯一能解决随机问题的方法是通过在这个问题上投入足够的实验来抵消噪音。

当你的训练算法样本效率低下且不稳定时，它会严重降低你的研究速



度。也许只需要 100 万步。但是当你给这个数字乘上 5 个随机种子，再乘上超参数调整，你需要不计其数的计算才能验证所提假设的有效性。

或许这些能让你觉得好受些：我研究手头的工作已经好一段时间了，我花了 6 个星期的时间，让一个从头开始的策略梯度在一系列 RL 问题上获得了 50% 的成功率。我有一个 GPU 集群供我使用，还有一些每天一起吃午饭的朋友可以一起交流，他们在该领域已经工作了几年了。

另外，我们从监督学习中学到的 CNN 设计似乎不适用于强化学习，因为强化学习的瓶颈在于信用分配或监管比特率，而不是表示能力不足。ResNet、batchnorm，或者是很深的神经网络在这里都无法发挥作用。

[监督学习] 是主动想要发挥作用的。即使你搞砸了，你也会得到一些非随机的反馈。RL 是必须被迫发挥作用的。如果你搞砸了什么，或者调整得不够好，你极有可能得到比随机情况更糟糕的策略。即使一切都很好，也有 30% 的可能得到一个很差的策略，没什么理由。

长话短说，你的失败更多是由于深度 RL 太难了，而不是因为“设计神经网络”的困难。

—— Andrey Karpathy 在 Hacker News 上发表的博客

随机种子的不稳定性就像是煤矿中的金丝雀。如果纯粹的随机性足以

导致每次实验结果之间如此大的差异，那么可以想象代码中实际的差异该有多大。

不过我们不需要去想这个问题，因为 Deep Reinforcement Learning That Matters, Henderson et al, AAAI 2018这篇论文已经给出了结论：

- 给奖励函数乘以一个常数可以导致表现性能出现巨大的偏差。
- 5 个随机种子可能不足以证明结果的显著性，因为通过仔细的选择，你可以得到不重叠的置信区间。
- 相同算法的不同实现对同一任务有不同的表现，即使是使用相同的超参数。

我的理论是，RL 对初始化和训练过程的动态非常敏感，因为数据都是在线收集的，而唯一的监督是一个标量奖励。一个在好的训练样例上得到的策略会比一个没有好的训练样例的策略能更快地引导自己。一个不能及时发现好的训练样例的策略会崩溃，什么也学不到，因为它相信它尝试的任何方向都会失败。

## 那么 RL 已经成功做到的事呢？

深度强化学习确实做了一些非常酷的事情。DQN 虽然是旧新闻，但当时绝对轰动。单个模型能够直接从原始像素中学习，而不需要对每个游戏单独进行调优。而且 AlphaGo 和 AlphaZero 也是非常令人印象深刻的成就。

然而，在这些成功之外，很难找到深度 RL 在真实世界创造实际价值的案例。

我试图思考深度 RL 在现实世界的用途，却发现非常困难。我想在推荐系统中找到些东西，但那些算法仍然以协同过滤（collaborative filtering）和上下文赌博机（contextual bandits）为主。

到最后，我能找到的最好的是两个谷歌项目：降低数据中心能耗，以及最近发布的 AutoML Vision。

我知道奥迪正在做一些深度 RL 相关的项目，他们在 NIPS 上展

示了自动驾驶的 RC 汽车，声称使用了深度 RL。我知道有一些很棒的工作，优化大型 Tensorflow 图的设备放置 (Mirhoseini et al, ICML 2017)；Salesforce 的文本摘要模型。融资公司肯定在用 RL 做实验，但目前还没有确切的证据。Facebook 已经在聊天机器人中用深度 RL 完成了一些很棒的工作。每个互联网公司应该都想过在广告服务模型中加入 RL，但是出于商业保密，我们并不知情。

在我看来，或许深度 RL 是一个不够强大，不能广泛应用的研究课题，也可能它是可用的，只是应用了它的人没有公开。我认为前者更有可能。

如果你问我图像分类的问题，我会告诉你预训练的 ImageNet 模型。对于图像分类问题，我们处在一个连电视剧《硅谷》的制作团队都能开发出识别热狗的软件的世界。但在深度 RL 领域几乎是不可能发生同样的事情。

## 鉴于目前的限制，什么时候深度 RL 能真正起效？

首先，真的很难说。想用 RL 解决所有事情，等同于试图用同样的方法来解决非常不同的环境下的问题。它不能一直起作用才应该是正常的。

话虽如此，我们可以从目前强化学习成功的项目列表中得出结论。这些项目中，深度 RL 或是学习了一些质量极优的行为，或是学习到了比以前更好的行为。

这是我目前的列表：

- 在前面的章节中提到的项目：DQN，AlphaGo，AlphaZero，跑酷机器人，减少能耗中心使用，以及利用神经架构搜索技术（Neural Architecture Search）的 AutoML。
- OpenAI 的 Dota2 1v1 影魔（Shadow Fiend）机器人，在一个简单的对战设置中打败了顶级职业选手。
- 任天堂明星大乱斗（Super Smash Brothers Melee）机器人，在 1v1 的 Falcon ditto 游戏中打败了职业选手。（Firoiu et al, 2017）

- 机器学习最近在玩法不限注德州扑克中击败了职业玩家。 Libratus (Brown et al, IJCAI 201) 和 DeepStack (Moravík et al, 2017) 都做到了这一点。有些人认为这是用深度 RL 完成的。但这两篇论文都不是，他们用的是虚拟遗憾最小化 (counterfactual regret minimization) 和迭代求解子博弈的方法。

从这个列表中，我们可以看出使学习更容易的一般属性。下面的属性都不是学习必需的，但是满足的特性越多绝对是越好的：

- 很容易产生近乎无限量的经验。为什么这一点有用应该是很清楚的。拥有的数据越多，学习问题就越容易。这适用于 Atari 游戏，围棋、象棋、将棋，以及跑酷机器人的模拟环境。它似乎也适用于能耗中心的项目，因为在以前的工作中 (Gao, 2014)，结果表明神经网络预测能源效率的准确度很高。这正是你训练 RL 系统想要的模拟模型。
- 问题可以简化成更简单的形式。我在深度 RL 看到的常见错误是想法太大。强化学习可以做任何事情！这并不意味着你必须同时完成每件事。

OpenAI 的 Dota2 机器人只玩了初级游戏，只在 1V1 对线设置中玩了影魔对影魔，使用硬编码的项目结构，并且调用了 Dota2 API 来避免解决感知问题。SSBM 机器人达到了超人的表现，但它只是在 1v1 比赛中，与 Captain Falcon 在 Battlefield 上，而且不限时。

这不是在讽刺这些机器人。如果你连一个简单的问题都不知道是否能解决，为什么还要研究一个困难的问题呢？所有研究的趋势都是先演示最小的概念证明，然后再推广它。OpenAI 在扩展他们的 Dota2 的研究工作，也在将 SSBM 机器人扩展到其他角色。

- 有一个方法可以将自我博弈引入学习过程。这是 AlphaGo，AlphaZero，Dota2 影魔机器人，以及 SSBM Falcon 机器人的一个组成部分。自我博弈，指的是比赛有竞争性，两个玩家都可以

由同一个 agent 控制。到目前为止，这种设置似乎是最稳定且表现良好的。

- 有一种简单的方式来定义一个可学习的、不可竞争的奖励。两个玩家的游戏设置这样的奖励：赢的 +1，输的 -1。最初的神经架构搜索论文（Zoph et al, ICLR 2017）使用了这个方法验证训练模型的精度。任何时候你引入计划奖励，你都会引入学习出优化错误目标的非最佳策略的可能。

如果你想进一步了解如何设计一个好的奖励函数，可以搜索“合适的评分规则（proper scoring rule）”。Terrence Tao 的博客中介绍了一个可行的例子。

而对于可学习性，除了多次尝试看看是否有效，我没有别的建议。

- 如果需要定义奖励，至少应该形式丰富。在 Dota2 中，最后的补刀（杀死小怪时触发），以及治愈（每次攻击或杀死目标后出发）会有奖励。这些奖励信号频率很高。对于 SSBM 机器人，造成伤害时会给予奖励，每次成功的攻击都会给出信号。行动与结果之间的延迟越短，反馈环形成越快，强化学习就越容易找到高回报的途径。

## 案例研究：神经架构搜索

我们可以结合一些原理来分析神经架构搜索的成功。从最初 ICLR 2017 版，在 12800 个样例后，深度 RL 能够设计先进的神经网络结构。诚然，每个样例都需要训练一个收敛的神经网络，但这仍然是非常有效的。

如上所述，奖励是验证准确性。这是一个非常丰富的奖励信号——即使一个神经网络设计的结果只将精度从 70% 增加到 71%，RL 仍然会考虑这一点。此外，有证据表明，深度学习的超参数接近线性无关。（这是基于经验得出的结论，见 Hyperparameter Optimization: A Spectral Approach (Hazan et al, 2017)）NAS 不完全是调整超参数，但我认为神

经网络设计决策与调整超参数类似。这对学习来说是个好消息，因为决策和性能之间的相关性很强。最后，奖励不仅很丰富，它也体现了我们训练模型时关心的是什么。

这些点的结合让我明白了为什么它“只”需要大约 12800 个训练好的网络来学习更好的网络，而在其他环境中需要数以百万计的样例。这个问题的许多方面都对 RL 有利。

简而言之：深度 RL 目前还不能即插即用。

## 未来发展

有一句老话——每个研究者都在学习如何讨厌他们的研究领域。尽管如此，研究人员还是会继续努力，因为他们太喜欢这些问题了。

这也是我对深度强化学习的大致感受。尽管我有保留意见，但我认为人们应该用 RL 去尝试不同的问题，包括那些可能行不通的问题。否则我们还有其他办法让 RL 变得更好吗？

我觉得深度 RL 的有效性只是时间问题。当深度 RL 足够鲁棒，能更广泛的使用时，它能做到非常有趣的事情。问题在于它该如何走到那一步。

下面我介绍了一些我认为的未来发展趋势。未来依然基于进一步的研究，因此我提供了这些研究领域相关论文的引用。

**局部最优已经足够好：**认为人类在任何事情上都已经做到最好了是很傲慢的想法。我认为与其他任何物种相比，我们仅仅刚好步入了文明阶段。同样，只要 RL 解决方案的局部最优解优于人类基准，它不必达到全局最优解。

**硬件解决一切：**我知道有些人认为最能影响人工智能的事情就是提升硬件。就我个人而言，我对于硬件能解决一切问题的观点持怀疑态度，但这一点肯定是很重要的。运行的速度越快，对样本效率的担心就越少，越容易对探索问题暴力解决。

**添加更多的学习信号：**稀疏奖励很难学习，因为得到的信息太少

了。也许我们可以变换积极奖励 (Hindsight Experience Replay, Andrychowicz et al, NIPS 2017)，定义辅助任务 (UNREAL, Jaderberg et al, NIPS 2016)，或用自我监督学习引导构建良好的真实世界模型。

**基于模型的学习提升样本效率：**我是这样描述基于模型的 RL 的：“每个人都想做，但很多人不知道。”原则上，一个好的模型能修正许多问题。就像 AlphaGo 一样，一个好模型让它更容易学习出解决方案。好的模型能迁移到新的任务，基于模型的方法也能使用较少的样本。

问题在于学习一个好的模型很难。我认为低维状态模型有时有效，图像模型通常太难。但是，如果这一点变得容易了，就能产生一些有趣的事情。

Dyna (Sutton, 1991) 和 Dyna-2 (Silver et al., ICML 2008) 是这个领域的经典论文。对于将基于模型学习和深度网络结合起来的研究领域，我推荐几篇 Berkeley 机器人实验室最近发表的几篇论文： Neural Network Dynamics for Model-Based Deep RL with Model-Free Fine-Tuning (Nagabandi et al, 2017)， Self-Supervised Visual Planning with Temporal Skip Connections (Ebert et al, CoRL 2017)。 Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning (Chebotar et al, ICML 2017)。 Deep Spatial Autoencoders for Visuomotor Learning (Finn et al, ICRA 2016)，以及 Guided Policy Search (Levine et al, JMLR 2016)。

**用强化学习进行调优：**第一篇 AlphaGo 论文就是采用监督学习，然后在其基础上用 RL 进行微调。这是一个很好的方法，因为它使用一个更快但不那么强大的方法来加快初始学习。该方法在其他环境下也有效，例如 Sequence Tutor (Jaques et al, ICML 2017)。可以将其视作以一个合理的先验条件作为 RL 过程的开始，而不是用一个随机的条件，或者用其他方法来解决学习先验条件的问题。

**可学习的奖励函数：**ML 的前景在于我们可以用数据来学习比人类的设计更好的东西。如果奖励函数的设计这么难，为什么不用 ML 来学习更

好奖励函数呢？模仿学习和逆向增强学习都显示了奖励函数可以用人为演示和人为评分来隐式定义。

逆RL和模仿学习领域的优秀论文：Algorithms for Inverse Reinforcement Learning (Ng and Russell, ICML 2000), Apprenticeship Learning via Inverse Reinforcement Learning (Abbeel and Ng, ICML 2004)，以及 DAgger (Ross, Gordon, and Bagnell, AISTATS 2011)。

最近也有一些工作将这种想法推广到深度学习：Guided Cost Learning (Finn et al, ICML 2016)，Time-Contrastive Networks (Sermanet et al, 2017)，和 Learning From Human Preferences (Christiano et al, NIPS 2017)。（Human Preferences 论文尤其证明从人为评分中学到的奖励函数比客观定义的奖励函数更适合于学习。）

对于不使用深度学习的长期研究，我推荐阅读：Inverse Reward Design (Hadfield-Menell et al, NIPS 2017) 和 Learning Robot Objectives from Physical Human Interaction (Bajcsy et al, CoRL 2017)。

**迁移学习拯救一切：**迁移学习的前景在于你可以利用从以前的任务中学到的知识来加速学习新知识。我认为这绝对是未来的发展趋势，任务学习足够强大，可以解决几个不同的任务。如果根本就没有学习，也无法进行迁移学习，而且给定任务 A 和 B，很难预测 A 是否可以迁移到 B。

该领域推荐阅读的论文：Universal Value Function Approximators (Schaul et al, ICML 2015)，Distral (Whye Teh et al, NIPS 2017)，和 Overcoming Catastrophic Forgetting (Kirkpatrick et al, PNAS 2017)。之前的工作可以阅读 Horde (Sutton et al, AAMAS 2011)。

机器人在 sim-real 迁移（模拟任务和真实任务之间的迁移学习）中取得了很大的进步。推荐阅读论文：Domain Randomization (Tobin et al, IROS 2017)，Sim-to-Real Robot Learning with Progressive Nets (Rusu et al, CoRL 2017)，和 GraspGAN (Bousmalis et al, 2017)。

**好的先验条件可以减少学习时间：**这一点和之前的几点之间有紧密联系。有一种观点认为，迁移学习是利用过去的经验为其他任务学习建立一

个良好的先验。RL 算法的设计适用于任何马尔可夫决策过程。如果我们承认，我们的解决方案只在很小一部分环境中表现较好，我们应该能够以一种有效的方式利用共享结构来解决这些环境中的问题。

Pieter Abbeel 在演讲中喜欢提到的一点是，深度 RL 只需要解决我们希望在现实世界中需要解决的任务。应该有一种现实世界的先验，能够让我们快速学习新任务，代价是非现实任务中较慢的学习成本，但这是非常值得的。

困难在于，这样的真实世界先验是很难设计出来的，但依然是可能的。我对元学习（meta learning）最近的工作很有兴趣，因为它提供了数据驱动的方法来生成合理的先验条件。例如，如果我想用 RL 设计仓库导航，我会用元学习来学习一个好的导航先验，然后根据特定仓库机器人的要求对先验进行调优。

最近这方面的工作可以在 BAIR 的博客中找到。

**更难的环境反而可能更简单：**我们从 DeepMind 跑酷论文中受启发最大的一点就是，如果通过添加几个任务变体让任务变得很难，你其实让学习变得更加容易了，因为策略不能过度拟合到任何一个设置而在其他设置下不损失性能。我们在域随机化论文中已经看到了类似的事情，甚至在 ImageNet 中：在 ImageNet 上训练的模型比在 CIFAR-100 上训练的模型推广性更强。正如我上面所说的，也许我们离更通用的 RL 只有一个“ImageNet 控制”的距离。

从环境角度来说，有许多选择。OpenAI Gym 目前用户最多，但是 Arcade Learning Environment、Roboschool、DeepMind Lab、DeepMind Control Suite 以及 ELF 都是很好的环境。

最后，虽然从研究的角度来看深度 RL 并不令人满意，但是它的经验问题对于实际用途可能并没有影响。假设一个融资公司使用深度 RL。他们用美国股票市场过去的数据训练了一个交易 agent，使用 3 个随机种子。在现场的 A / B 测试，一个收入减少了 2%，一个相同，另一个收入多出 2%。在这个假设中，复现性不影响结果，你只要部署收入多出 2%

的模型的就好。交易 agent 可能只在美国的表现性能好，没关系，如果在全球市场推广效果不佳，不要部署在那里就好。取得非凡的成就和别人能复制的成功之间还有较大的差距，可能前一点更值得重点关注。

## 我们现在走到哪一步了

从许多角度来说，我对深度 RL 当前的状态都感到很烦恼。然而，人们对它有着强烈的研究兴趣，可谓是前所未见。Andrew Ng 在他的 Nuts and Bolts of Applying Deep Learning（螺母和螺栓的应用深度学习）的谈话中很好的总结了我的想法，短期很悲观，但长期的乐观也平衡了这一点。深度 RL 现在有点混乱，但我仍然相信它是未来。

也就是说，下一次有人再问我强化学习是否可以解决他们的问题时，我仍然会告诉他们：不，不行。但我也会告诉他们等几年后再问我，到那时，也许它就可以了。



# 传奇工程师卡马克入坑 AI：徒手一周实现反向传播和 CNN

撰稿 | Natalie、Vincent

有这么一个大牛程序员，他在几乎没有接触过神经网络的情况下，仅用了一周时间，在几乎是最基础且受限的编程环境下，从零开始徒手撸码，实现了反向传播和 CNN。

今年，这位程序员已经 48 岁了，他叫：约翰 · 卡马克。

约翰 · 卡马克是何方神圣？

## 谁是约翰 · 卡马克？

他是一位集传奇工程师、大神、疯狂程序员、黑客之神、第一人称射击游戏之父、业界活化石、一代玄学码神所有称号为一身的老牌程序员，一举一动都牵动人心。

约翰 · D · 卡马克二世（John D. Carmack II，出生于 1970 年 8 月 20 日），是美国的电玩游戏程序员、id Software 的创始人之一，id 是一家专门开发电子游戏、电视游戏的公司，成立于 1991 年。

至于 id Software 这家公司都制作过什么游戏呢？说几个你应该就知道了：《CS（反恐精英）》、《半条命》、《毁灭战士》都出自这家公司。



怎么样，对这位卡马克先生多少有些了解了吧？

当然，对技术大牛的一切不提技术水平的吹捧都是要流氓！——沃茨·基硕德，那我们就来说一说这位卡马克大神的技术水平。

卡马克最让人咋舌的冒险就是涉足了第一人称射击游戏领域。他的编程能力得以毫无保留地展现，随后的《德军总部 3D》（Wolfenstein 3D）、《毁灭战士》（Doom）和《雷神之锤》（Quake）就是最好的佐证。这些游戏和它们的后续版本都获取了巨大的成功。

卡马克喜欢在电脑图像领域尝试新的技术，比如他在 Doom 上第一次使用了二叉树分区技术，表面缓存技术则在 Quake 中第一次出现。还有就是后来在 Doom3 里面使用的“卡马克反转”（即 shadow volume 的 z-fail 方法。事实上并不是卡马克首先创新了这个技术，他在后来独立研究出来）。

卡马克创造的游戏引擎被用来制作其他的第一人称射击游戏，比如《半条命》（Half-life）和《荣誉勋章》（Medal of Honor）。

在 2007 年苹果公司全球软件开发者年会上，卡马克宣布了 id Tech 5，它实际上消除了过去对美工和设计人员的纹理内存限制，允许在像素级别上对整个游戏世界实现独特的定制设计，并提供了几乎无限的视觉真实性。“该技术可以允许”广袤的户外场景，而室内场景则具有前所未见的艺术细节。

2013 年的 QuakeCon，卡马克表示对函数式编程很感兴趣。他在 Twitter 上表示了“已经学习 Haskell 一年”，“学习 SICP 和尝试使用 Scheme 中”，并且表示正在用 Haskell 重写德军总部。与此同时，卡马克建议其他游戏开发者尝试函数式编程。

除了游戏领域，卡马克还是个火箭爱好者，并成立了名为犰狳宇航（Armadillo Aerospace）的私人研发团队。

**总结起来，这位卡马克大神就是：特别能创造、特别能折腾还特别聪明。在 AI 大火的今天，他又把自己的“折腾精神”发挥得淋漓尽致。**

## 大神的一周编程实践：徒手实现反向传播与 CNN

几天之前，卡马克大神在 Facebook 上发表了一篇文章，总结了一下自己如何徒手实现反向传播与 CNN 的事情。以下内容编译自卡马克的自述文章：

间隔了好几年，我终于又可以进行我的一周编程实践了，在编程的世界里我可以在隐士模式下工作，远离日常的工作压力。过去几年，我的妻子一直慷慨地为我打造这种环境，但我一般不善于在工作中休息。

随着 Oculus（卡马克目前所在公司）工作步伐的改变，我打算从头开始编写一些 C ++ 代码来实现神经网络，而且我想用严格的 OpenBSD 系统来实现。有人可能会说这是一个非常随意且不太靠谱的选择，但事实证明这是行得通的。

尽管我没有真正使用过它，但我一直很喜欢 OpenBSD——一个相对简单且足够自用的系统，它具有紧凑的图形界面，并且重视质量和工艺。Linux 什么都好，但图形界面不够紧凑。

我不是 Unix 极客。各种系统我都可以用，但我最喜欢在 Windows 上使用 Visual Studio 进行开发。我认为在老式的 Unix 风格下完成一周的沉浸式工作会很有趣，即使这意味着工作速度要慢一些。这是一次复古计算的冒险——使用 fvwm 和 vi。不是 vim，实际上是 BSD vi。

其实到最后，我也并没有真正全面地探索这个系统，95% 的时间

都花在基本的 `vi/make/gdb` 操作中。我喜欢那些操作手册页面，因为我试图在自带的系统中做所有事情，而不诉诸于互联网搜索。阅读诸如 Tektronix 终端等已有 30 多年历史的事物的参考手册是一件很有意思的事情。

我有点意外，C++ 的支持做得不是很好。G++ 不支持 C++ 11，并且 LLVM C++ 不能很好地与 gdb 配合使用。Gdb 也让我踩了不少坑，我同样怀疑是由于 C++ 的问题。我知道你可以获得更新的版本，但我坚持使用基础系统。

事后看来，我还不如完全复古，干脆在 ANSI C 中做所有事情。和许多老程序员一样，有很长一段时间，我一直在琢磨“也许 C++ 并不像我们想象的那么好”。我仍然喜欢 C++ 的很多方面，但对于我来说用普通的 C 语言来构建小型项目并不困难。

也许下次我再进行一周编程实践时，我会尝试完整的 emacs，这是另一个我还没怎么接触过的主流文化。

我对大多数机器学习算法已经有比较基本的了解，并且我已经完成了一些线性分类器和决策树的工作，但出于某种原因，我从未使用过神经网络。在某种程度上，我怀疑是深度学习太过流行导致那个不愿意人云亦云的内在的我感到抵触。我仍然持有一点反思性的偏见，反对“把所有的东西都扔在 NN（神经网络）上，让它自己整理出来！”

为了彻底贯彻我这次复古主题的精神，我打印了几篇 Yann LeCun 的旧论文，并打算完全脱离互联网去完成所有事情，这就好像我被困在了某个山间的小屋里，但最后我还是在 YouTube 上看了很多斯坦福 CS231N 课程视频，并发现它们非常有价值。我一般很少看课程视频，因为这通常让我觉得时间花的不值，但在我“隐退编程”的这段时间里看这些视频感觉还是很棒的！

我不认为我有什么特别的洞察力来为神经网络添砖加瓦，但对我来说这是非常高效的一周，充分将“书本知识”转化为真实体验。

我采用了一种我经常使用的模式：先写出一段粗糙且不怎么优美的代

码，初步得到结果，然后用从视频课程学到的东西再写出一段全新且更优美的代码，这样以来两份代码可以并存和交叉检查。

我一开始尝试实现反向传播，结果两次都做错了，数值微分比较至关重要！有趣的是，即使在各个部分都出现错误的情况下，训练仍然能够进行——只要大多数时候符号正确，通常就会取得进展。

我对我的多层神经网络代码非常满意，它已经可以在我未来的工作中直接使用。是的，对于很多重要的事情我一般都使用一个已有的库，但是在很多时候，哪怕只有一个.cpp 和.h 文件是你自己写出来的，还是会方便许多。

我的 CNN 代码还很粗糙但已经凑合能用了，我可能还会再用一两天的时间来完成一个更干净而灵活的实现。

有一件事我觉得很有趣，在加入任何卷积之前，用我的初始 NN 基于 MNIST 进行测试，我得到的结果明显好于 LeCun 98 年的论文中报告的用于比较的非卷积 NN——我使用了包含 100 个节点的单个隐藏层，在测试集上的错误率大约为 2%，而 LeCun 论文中使用了包含更多节点和更深层的网络错误率却是 3%。我将其归因于现代最佳实践——ReLU、Softmax 和更好的初始化过程。

我认为这是关于神经网络工作的最有趣的事情之一：它非常简单，突破性的进步通常只需要几行代码即可表达出来。这感觉和图形世界中的光线跟踪有一些相似之处，只要你拥有数据并且对运行时间有足够的耐心，你就可以很快地实现基于物理的光传输光线跟踪器，并生成最先进的图像。

通过探索一系列训练参数，我对过度训练 / 泛化 / 正则化有了更好的理解。在我不得不回家的前一天晚上，我不再修改架构，只是玩超参数。“训练！”简直比“编译！”更糟糕，更难让人保持专注。

现在，我要开始睁大眼睛寻找新的工作机会了，我迫不及待地想把我学到的新技能用起来！

我有点担心明天进入办公室时，我的邮箱和工作区将会变成什么样子。

之后，大神 Yann LeCun 也回复了卡马克：

欢迎入坑，约翰！在 OpenBSD 上用 vi 来完成这件事属英雄所为！每个人第一次尝试的时候都会遇到梯度错误。

在过去的 35 年里，我也做过很多次类似的事情。我的第一个反向传播模拟器是在 PDP11 的 FORTRAN 中编写的（大约在 1984 年）。第二个是在 Pascal 上的 Pr1me OS（大约在 1986 年，使用类似 Emacs 的编辑器）。第三个是由 Leon Bottou 和我在 C 中使用 emacs / gcc / make 在我们的 Amiga 1000s 上编写的（1987 年），我们写了一个 lisp 解释器用作交互式前端语言。当我在 1987 年搬到多伦多时，我把这个东西移植到了 Sun OS (BSD Unix) 上。直到 2011 年左右，我们一直使用这个系统及其后继者（称为 Lush），2011 年之后我们才切换到 Torch7。但在 2010 年，我开始编写一个名为 EBLearn 的 C ++ 深度学习框架，由 Pierre Sermanet 和 Soumith Chintala 完成并维护。

在我们 1998 年的论文中，MNIST 上的全连接网络的错误率是次优的，因为我们使用了最小平方损失（对于标记噪声往往更加鲁棒），而不是交叉熵，利用交叉熵和更大的网络 (>1000 个隐藏单元)，错误率可以下降到 1.6% 左右。

对于不太了解技术的读者，大概会产生一种“神仙聊天”的感觉，每一个字你都认识，但是你就是看不懂，所以我们贴心地为大家解释了一个简约版：

首先，大神卡马克牛逼在哪儿了呢？看下知乎网友 wsivoky 的总结：

- 48 岁仍然在学习新技术、编写代码；
- 喜欢受限的开发环境（Twitter 上曾说过，受限环境有益）；
- vi&emacs 都用（注意是 bsd vi，不是 vim，前者是 bill joy 当初开发的第一版也是第一个 screen editor）；
- 使用 OpenBSD，但喜欢 Windows Visual Studio；
- 用 makefile&gdb；

- 使用 C++，同时也吐槽 C++；
- 不用 Google Search，完全看 man (Tektronix terminal 是 bill joy 编写 vi 时代的显示器，更早时候是靠纸张输出。)；
- 对 Deep Learning 如此流行持保留态度；
- 打印了 MNIST 早期论文并尝试实现；
- 最终忍不住看了 cs231n；
- 使用自己喜欢的开发方式：先实现后优化；
- 实现反向传播容易出错；
- 对 CNN 的效果很满意 (Tensor 操作也是“from-scratch-in-C++”CPU 实现的？)；
- 对 NN 代码之少却强大感到兴奋（看到纯手工撸的 NN 运行结果之神奇，是很持久的快感）；
- 编写算法实现对过拟合、正则化、调参有了更好的理解。

再简约一点儿，大概就是这样。

**卡神：**反向传播和 CNN 这东西之前没搞过，那既然如此就自己动手试试吧。一个礼拜之后发现：哎呀~ 神经网络这玩意儿还挺有意思，感觉入坑了。

**乐村儿：**大兄弟你终于入坑了，来来来，我跟你说，用哥这方法你还有上升空间，以后咱们可以经常交流经验。

## 成为大神，从你做起！

卡马克的一周编程实践一出，迅速在国内外程序员圈子里引发骚动。国外程序员论坛 Reddit 的 MachineLearning 板块下，卡马克一周



编程实践的话题受关注度 364，共收获 53 则留言

知乎上也很快有网友发布了相关问题，截止发稿时间，已有 685 人

关注, 浏览次数 25991

程序员 编程 计算机 机器学习 约翰·卡马克 (John Carmack)

关注者  
685

被浏览  
25,991

### 如何看待传奇工程师 John Carmack 的一周编程实践?

估计要火, 原文发在 fb 上: [facebook.com/permalink....](https://facebook.com/permalink....) [图片]...显示全部 ▾

卡马克 Facebook 下的留言大多是这样的:



其实大家为之震动的并非卡马克入坑 AI 这件事本身（当然大神神乎其技的编程水平也确实让人顶礼膜拜），而是卡马克作为一位诸多成就加身的老牌程序员兼 Oculus CTO，仍然狂热地爱着编程这件事本身，并且保持着对一切新鲜事物感到好奇的赤子之心。

当卡马克对 AI、神经网络、深度学习产生兴趣，决定探索一下这些新技术时，他没有直接安装 TensorFlow 或 PyTorch，而是花了一个星期的时间，通过逐一编写各个功能模块代码，并进行了 MNIST 实验，从头开始实现 CNN 和反向传播。他没有去学习最新的术语，但却从中获得了最有价值的知识。这里不争论 scratching 到底好还是不好，但手撸新技术确实是学习的一种好途径。面对新技术，不纠结要不要尝试、不犹豫会不会太难，而是撸起袖子动手干，难怪乎知乎网友将卡马克称作“老程序员的标杆”。

这是一个最好的时代，技术日新月异，热点一年一个，对技术人才的需求越来越大。这是一个最坏的时代，深陷技术漩涡的程序员们成为了最容易焦虑的群体，今天是 AI 火了我要不要转行，明天是新技术太多学不

动了，后天是 35 岁程序员中年危机了。其实哪里有那么多可焦虑的，有时间焦虑不如多撸几行代码。互联网技术更新确实快，这要求程序员必须终身学习，但这是选择了这个职业的宿命。

卡马克告诉你：忘记中年危机吧，想想我们应该做些什么，才不辜负这美好的时代！

## 写在最后

这里引用知乎网友姚钢强的回答作为结语。

读过《DOOM 启世录》 两遍，John Carmack 说过：

在信息时代，客观障碍已不复存在，所谓障碍都是主观上的。如果你想动手开发什么全新的技术，你不需要几百万美元的资金，你只需要在冰箱里放满比萨和可乐，再有一台便宜的计算机，和为之献身的决心。我们在地板上睡过，我们从河水中趟过。

他切实做到了，与君共勉。

## 参考资料

1. <https://zh.wikipedia.org/wiki/%E7%B4%84%E7%BF%B0%C2%B7%E5%8D%A1%E9%A6%AC%E5%85%8B>
2. [https://www.reddit.com/r/MachineLearning/comments/82mqtw/d\\_john\\_carmacks\\_1week\\_experience\\_learning\\_neural/](https://www.reddit.com/r/MachineLearning/comments/82mqtw/d_john_carmacks_1week_experience_learning_neural/)
3. <https://www.zhihu.com/question/268230219>
4. [https://www.facebook.com/permalink.php?story\\_fbid=2110408722526967&id=100006735798590](https://www.facebook.com/permalink.php?story_fbid=2110408722526967&id=100006735798590)



# JeffDean 又用深度学习搞事情：这次要颠覆整个计算机系统结构设计

作者 | Milad Hashemi 等

编译 | Rays

上个月，Jeff Dean、Michael I. Jordan、李飞飞、LeCun 等大牛发起的系统机器学习会议 SysML 2018 在斯坦福闭幕。这是一个专门针对系统和机器学习交叉研究领域的会议，目的是弥补当前关于 AI/ML 的讨论大多偏向算法和应用而关于底层的基础设施却少有讨论造成的缺口。

在会议主题演讲中，Jeff Dean提到了去年谷歌发布的研究论文《用 ML 模型替代数据库索引结构》（The Case for Learned Index Structures，去年在技术圈引起轰动，详见 AI 前线首发报道），其关键点就是用计算换内存。未来基于摩尔定律的 CPU 在本质上将不复存在，利用神经网络取代分支重索引结构，数据库就可以从硬件发展趋势中受益。Jeff Dean 认为，这代表了一个非常有前景且十分有趣的方向：传统系统开发中，使用 ML 的视角，就能发现很多新的应用。除了数据库，ML 还能使用在系统的哪些方面？Jeff Dean 表示，一个很大的机会是启发式方法，计算机系统里大量应用启发式方法，因此，ML 在用到启发式方法的地方都有机会带来改变。

今天 AI 前线为大家带来的是《用 ML 模型替代数据库索引结构》研究

的续篇，这一次谷歌研究人员尝试用深度学习解决冯·诺依曼结构内存性能瓶颈，并取得了一定成果。Jeff Dean 在推特评论表示，未来在计算机系统的众多领域，类似研究工作将不断增多。

该论文是将神经网络应用于计算机结构设计中的一项开创性工作。如何将机器学习应用于计算机硬件结构领域，目前依然鲜有研究。这篇论文展示了深度学习在解决冯·诺依曼架构计算机内存性能瓶颈问题中的应用。论文关注的是学习内存访问模式这一关键问题，意在构建一种准确高效的内存预期器。研究提出将预期策略视为 NLP 中的 n-gram 模型，并使用 RNN 模型完全替代基于表的传统预取。在基准测试集上的实验表明，基于神经网络的模型在预测内存访问模式上可稳定地给出更优的准确率和召回率。

Jeff Dean @JeffDean · 3月7日  
@Miles\_Brundage is right: this sort of work is going to be a growing trend across many field of computer systems. This paper is joint work by many people across Google (and students interning at Google from Stanford and UCSC).

Miles Brundage @Miles\_Brundage  
"Learning Memory Access Patterns," Hashemi et al.: arxiv.org/abs/1803.02329

NNs for prefetching. ~A sequel to "The Case for Learned Index Structures" as foreshadowed by @JeffDean at NIPS ("Learning in the core of all of our...")

翻译自英语

5 19

Miles Brundage  
@Miles\_Brundage

正在关注

"Learning Memory Access Patterns," Hashemi et al.: [arxiv.org/abs/1803.02329](https://arxiv.org/abs/1803.02329)

NNs for prefetching. ~A sequel to "The Case for Learned Index Structures" as foreshadowed by @JeffDean at NIPS ("Learning in the core of all of our computer systems will make them better/more adaptive").

翻译自英语

以下内容为AI前线根据论文编译整理而成。

在计算机结构的设计中，很多问题涉及使用预测和启发式规则，内存预取就是其中的一个经典问题。内存预取将指令和数据在被微处理器实际使用前，预先置于存取速度更快的内存中。因为在冯·诺伊曼结构计算机中，微处理器的运算速度要比内存访问速度高出数个数量级，因此内存预取是一个关键的瓶颈问题，也被称为“内存墙”（Memory Wall）。有统计表明，计算机中 50% 的计算周期是在等待数据加载到内存中。为解决内存墙的问题，微处理器使用了层次结构的内存，即在结构上采用多级缓存设计，让规模更小但是访问速度更快的缓存更接近于处理器。为降低内存读取的延迟，需要预测何时以及哪些数据应预取到缓存中。因此，提高性能的一个关键问题，是如何给出有效的内存访问模式预测。

在现代处理器中，使用了一些预测方法。很多预测器是基于记录表实现的。记录表是一种内存数组结构，记录了未来事件与以往历史信息的关联性。但是相比起传统的单机工作负载，现代数据中心的工作负载规模呈现指数级的增长，这对内存访问模式预测提出了严峻挑战。预测的准确性会随预测表规模的增大而显著下降。简单地扩展预测表的规模，需在硬件实现上付出很大的代价。

神经网络技术已在 NLP 和文本理解取得了很好的效果，它为解决序列预测问题提供了一种强大的技术。例如，在 SPARC T4 处理器中，已经部署了一种简单的感知机，实现对分支处理指令的预测。但是，对于微处理器架构设计而言，如何有效地引入序列学习算法依然是一个开放的问题。

本文的研究考虑在微处理架构中引入基于序列的神经网络模型，意在解决内存墙所提出挑战，将序列学习应用到内存预取这一难题上。内存预取问题，在本质上可以看成是一种回归问题。但是该问题的输出空间很大，并且非常稀疏，使用标准的回归模型难以给出好的效果。本文作者们考虑使用一些图像和语言生成上的最新研究成果，例如 PixelRNN 和 Wavenet。这些研究成果对搜索空间做离散化，使得内存预取模型更近似于神经语言模型，并依此作为构建神经网络内存预取器的出发点。本文研究所提出的神经网络模

型，可以成功地建模问题的输出空间，实现一种神经网络内存预取，并给出了显著优于现有传统硬件预取器的性能。此外，论文中使用的 RNN 可以根据应用的内存访问轨迹辨别应用的底层语义信息，给出的结果更具可解释性。

## 背景知识

### 内存预取器 (Prefetcher)

内存预取器是一种计算机硬件结构，它使用历史内存访问记录预测将来的内存访问模式。现有的内存预取器大体上可分为两类。一类称为“步长预取器” (stride prefetcher)，另一类称为“关联预取器” (correlation prefetcher)。现代处理器中通常使用的是步长预取器，序列中固定采用稳定的、可重复的地址差值（即序列中相邻两个内存地址间的差值）。例如，给定一个内存访问模式 (0, 4, 8, 12)，每次访问的地址差值为 4，那么预取器可以由此预测此后的地址访问模式为 (16, 20, 24)。

关联预取器可预测地址差值不固定的重复模式。它使用了一个大的记录表，保持过往的内存访问历史信息。相比于步长预取器，关联预取器可以更好地预测非规则变化的模式。典型的管理预取器包括 Markov 预取器、GHB 预取器，以及一些使用大规模内存中结构的最新研究预取器。关联预取器需要具备一张大规模的记录表，该表的实现代价大，因此通常并未实现在多核处理器中。

### 递归神经网络 (RNN)

深度学习适用于很多的序列预测问题，例如语言识别和自然语言处理。其中，RNN 可对长距离依赖建模。LSTM 作为一种广为使用的 RNN 变体，通过以加法方式而非乘法方式传播内部状态，解决了标准 RNN 的训练问题。LSTM 由隐含状态、元胞状态、输入门、输出门和记忆门组成，分别表示了需要存储的信息，以及在下一个时间步 (timestep) 中传播的信息。给定时间步和输入，LSTM 的状态可以使用如下过程计算：

1. 计算输入门、输出门和记忆门

2. 更新胞元状态
3. 计算 LSTM 隐含状态（输出）

其中，表示当前的输入和前一个隐含状态的级联（concatenation），表示点乘操作（element-wise multiplication），是 Sigmoid 非线性函数。上面给出的过程表示了单个 LSTM 层的处理形式，是该层的权重，是偏差。LSTM 层可以进行堆叠，在时间步 N 的 LSTM 层的输出，可以作为另一个 LSTM 层的输入，这类似于前向回馈神经网络中的多层结构。这使得模型在使用相对较少额外参数的情况下，增大了灵活性。

## 问题定义

### 将预取作为预测问题

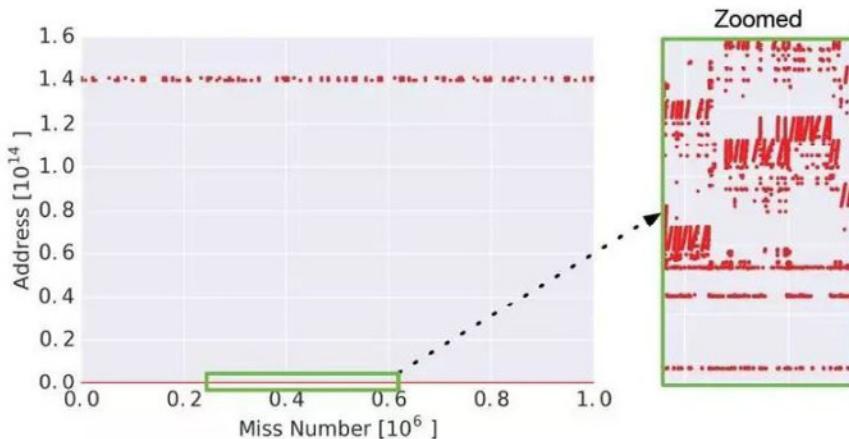
预取可以看成是对未来内存访问模式的预测过程。如果数据并不在缓存中（即缓存未命中），预期器需要根据历史访问情况，到主存中去获取数据。每条内存地址由内存操作指令（load/store）生成。内存操作指令是计算机指令集中的一个子集，实现对计算机系统中具有地址内存的操作。

很多硬件厂商建议在做出预期决策时使用两个特征。一个是截至目前缓存未命中的地址序列，另一个是指令地址序列，也称为程序计数器（Program counter，PC），即关联到生成每条缓存未命中地址的指令。

每条 PC 可以唯一标识一条指令，指令是对给定代码文件中的特定函数编译生成的。PC 序列可告知模型应用代码的控制流模式，而缓存未命中地址序列可告知模型需要下一步预期的地址情况。在现代计算机系统中，上述特性均使用 64 位整数表示。

初始模型可以在每个时间步使用两个输入特征，即在步生成的缓存未命中地址，以及由 PC 预测的 N+1 步缓存未命中情况。但是其中存在一个问题，应用的地址空间是非常稀疏的。对于此问题的训练数据而言，缓存未命中的空间复杂度为量级，其中只有的缓存未命中地址出现在物理地址空间中。图 1 显示了使用标准 SPEC CPU2006 基准测试集 omnetpp 时的缓存情

况，红色点表示了缓存未命中地址。这一空间范围宽泛，具有重度多模态本质，这对传统的时序回归模型提出了挑战。例如，虽然神经网络非常适用于正则化输入，但是对这样的数据做正则化时，有限精度的浮点数表示将导致显著的信息丢失。该问题影响了对输入层和输出层的建模。在本文中，作者对此给出了解决方法。



**图 1 数据集 omnetpp 上的缓存未命中情况。图中多尺度地展示了稀疏访问模式将预取作为分类问题**

在本文作者提出的解决方法中，考虑将整个地址空间视为一个离散的大规模词汇表，并对词汇表执行分类操作。这类似于 NLP 中对随后出现的字符或单词的预测问题。但是，该空间是极度稀疏的，并且其中部分地址比其它地址更频繁地访问。对于 RNN 模型，是可以管理这样规模的有效词汇表的。此外，相比于单模态回归技术，通过生成多模态输出，模型处理可以变得更加灵活。在图像处理和语音生成领域中，已有一些研究成功地将输出输出作为预测问题而非回归问题。

但是，预期问题存在着种可能的 softmax 目标，因此需要给出一种量化 (quantization) 模式。更重要的是，预取必须完全准确才能发挥作用。对于通常的 64 个字节情况应该如此，对于通常是 4096 个字节的页面也应如此。在页面层级上的预测，将会给出中可能的目标。为避免对超过个值应

用 softmax，一些研究提出应用非线性量化模式将空间降维到 256 类。但是这些解决方法这并不适用于本文中的问题。因为这些方法降低了地址的分辨率，使得预取无法获得所需的高精度。

由于动态边界效应，例如地址空间布局随机化（ASLR）问题，同一程序的多轮运行可能会访问不同的原始地址。但是，同一程序对于一个给定的布局将给出一致的行为。由此，一种可行的策略是预测地址差值，而非直接预测地址本身。如表 1 所示，地址差值的可能出现情况，要比地址的可能出现情况呈空间指级别降低。

**表 1 程序追踪数据集统计情况，“M”表示“百万”**

Dataset	# Misses	# PC	# Addrs	# Deltas	# Addrs 50% mass	# Deltas 50% mass
gems	500M	3278	13.11M	2.47M	4.28M	18
astar	500M	211	0.53M	1.77M	0.06M	15
bwaves	491M	893	14.20M	3.67M	3.03M	2
lbm	500M	55	6.60M	709	3.06M	9
leslie3d	500M	2554	1.23M	0.03M	0.23M	15
libquantum	470M	46	0.52M	30	0.26M	1
mcf	500M	174	27.41M	30.82M	0.07M	0.09M
milc	500M	898	3.74M	9.68M	0.87M	46
omnetpp	449M	976	0.71M	5.01M	0.12M	4613
soplex	500M	1218	3.49M	5.27M	1.04M	10
sphinx	283M	693	0.21M	0.37M	0.03M	3
websearch	500M	54600	77.76M	96.41M	0.33M	5186

## 模型描述

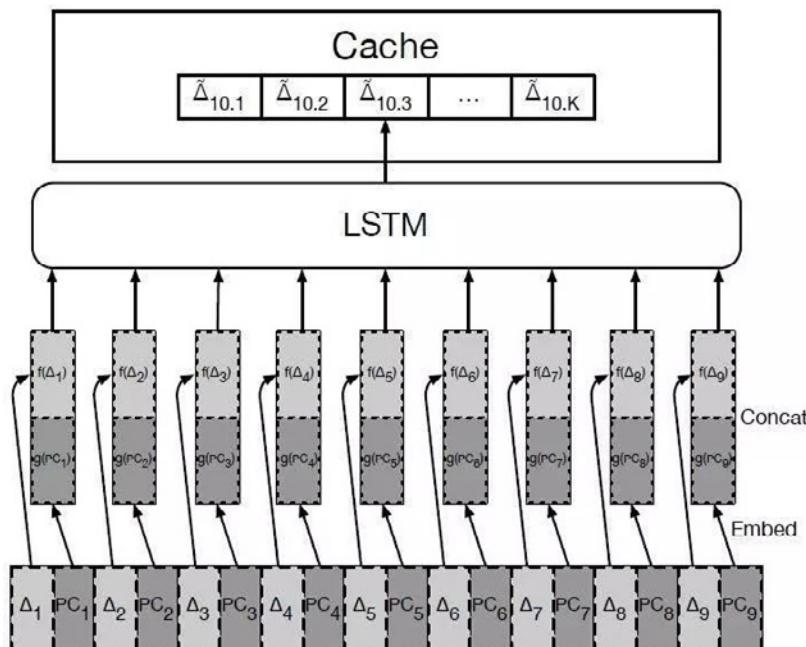
论文给出了两种基于 LSTM 的预取模型。第一种模型称为“嵌入 LSTM”模型，它类似于标准的语言模型。第二种模型利用内存访问空间的结构去降低词汇表的规模，降低了模型占用的内存量。

### 嵌入 LSTM 模型

该模型限制输出词汇表的规模，仅建模最频繁出现的地址间隔。根据表 1，要获得最优 50% 的准确性，所需的词汇表规模仅为量级乃至更小。这样规模的词汇表完全处于标准语言模型可解决的能力范围内。基于此，论文提出的第一种模型对输出词汇表规模做了限制，选用了其中 50000 个最频繁出现的唯一地址差值。对于输入词汇表，模型考虑在词汇表中出现了至少 10 次的所有地址差值。要进一步扩展词汇表，无论在统计学还是计数上

都是有挑战的。这些挑战可以采用层次 softmax 类方法解决，有待进一步研究。

嵌入 LSTM 的结构如图 2 所示。模型将输入和输出地址差值以分类表示（即独热编码）。在时间步，分别嵌入输入和，并将嵌入词汇级联起来，构成两层 LSTM 的输入。进而，LSTM 对地址差值词汇做分类，并且选取个最高概率地址差值，用于预取。分类方法可直接给出预测概率，这对传统硬件是另一个优点。



**图 2 嵌入 LSTM 模型结构。其中 和 分别表示嵌入函数**

实际实现中，预取器会返回多个预测值，因此需要从中做出权衡。虽然更多的预测值增加了缓存命中下一个时间步的可能性，但是有可能会从缓存中移除其它一些有用的项。论文选取在每个时间步预取 LSTM 的头部 10 个预测值。这里还有其它一些可能的做法，例如使用使用 beam-search 预测之后个时间步，或是通过直接学习先于 LSTM 的一次前向操作给出对到时间步的预测。

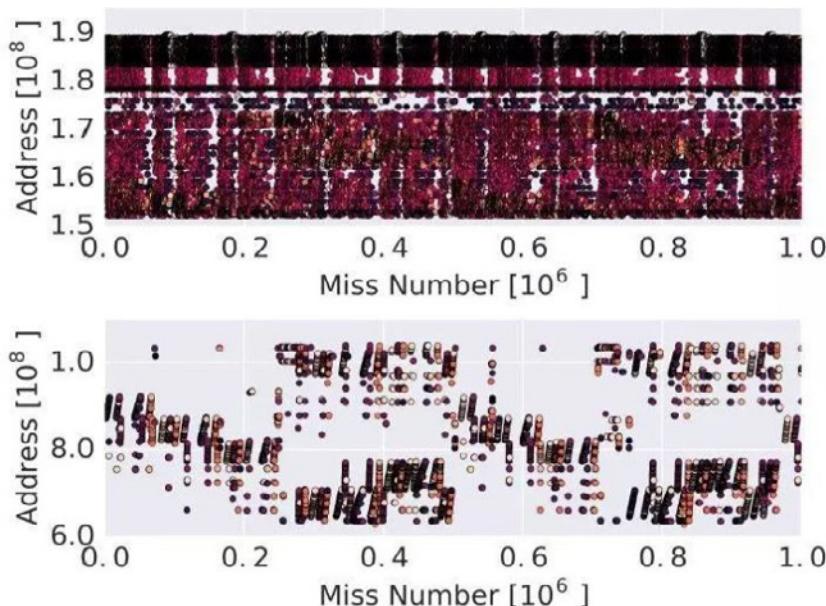
嵌入 LSTM 模型存在一些局限性。首先，大型的词汇表增加了模型的计算复杂度和存储量。其次，截取部分词汇表无疑为模型准确性设置了一个上

限。最后，不易处理一下罕见发生的地址差值，因为这些差值很少在训练集中出现。该问题在 NLP 领域称为“罕见词汇问题”。

## 聚类与 LSTM 的组合模型

假定大多数地址间感兴趣操作发生在本地地址空间中。例如，结构体和数组的数据结构通常使用持续的数据块存储，并会被指令重复访问。论文在模型设计中引入了上述理念，对本地上下文做细致建模。与之不同的是，在嵌入 LSTM 模型中不仅考虑了本地上下文，而且引入了全局上下文。

通过查看更窄区域的地址空间，可以发现其中的确存在着丰富的本地上下文。为此，论文从 omnetpp 中取出了部分地址集，并使用 K-Means 聚类为 6 个不同的区域。图 3 中展示了其中的两个聚类。



**图 3 展示了基准测试数据集 omnetpp 的六个 K-Means 聚类中的两个聚类。内存访问按照生成访问的 PC 分别标记颜色**

为达到对本地地址空间区域建模的相对准确性，论文对原始地址空间进行 K-Means 聚类，将数据分区为多个聚类，并对计算每个聚类内的地址差值。图 4a 是对上例的可视化。该方法具有一个显著的优点，就是聚类内的

地址差值集要显著地小于全局词汇表中的差值，这缓解了嵌入 LSTM 模型中存在的一些问题。

为进一步降低模型的规模，论文使用了多任务 LSTM 对所有距离建模，即对每个聚类独立使用一个 LSTM 建模。聚类 ID 也添加为模型的特性，这为每个 LSTM 提供了一组不同的偏差。

将地址空间分区为更小的区域，意味着每个聚类内地址组使用的幅度量级大体上相同。因此，所生成的地址差值可以有效地正则化为适用于 LSTM 的真实输入值，不再需要维护一个大型的嵌入矩阵，进而进一步降低了模型的规模。更重要的是，下一个地址差值预测的问题可作为一个分类问题，而回归模型在现实中通常不够精确。

该模型解决了一些嵌入 LSTM 中存在的问题。预处理步骤（即对地址空间的聚类）是模型中需要权衡考虑的一个因素。此外，该模型只对本地上下文建模，也可以对程序访问的不同地址空间区域进行动态建模。

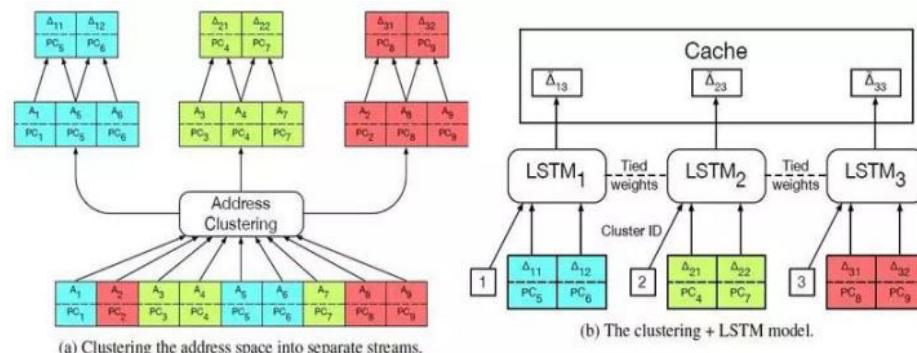


图 4 组合聚类和 LSTM 模型的结构，以及数据处理

## 实验

一个有效的内存预取器，应该具备对缓存未命中情况的准确预测能力。因此，论文的实验主要针对如何测定预取器的有效性。

### 实验数据的收集

实验中所使用的数据主要是程序的动态追踪数据，其中包含程序计算的

一个内存地址序列。追踪数据使用动态工具 Pin 捕获。该工具附着在进程上，并以文件形式给出被测量应用访问内存的“PC，虚地址”元组。原始访问追踪数据主要包含了堆栈访问等命中内存的方法。对于研究中关注的预测缓存未命中情况，实验中使用了一个模拟 Intel Broadwell 微处理器的模拟器，获取缓存未命中情况。

实验程序使用的是对内存做密集访问的 SPEC CPU2006 应用。该基准测试集用于算机系统的性能做完全评估。但是，与现代数据中心工作负载相比，SPEC CPU2006 依然是一个小规模的工作集。因此在论文的实验中，还添加了 Google 的 Web 搜索工作负载。

实验中将内存访问数据集分为训练集和测试集，其中 70% 用于训练，30% 用于测试。在每个数据集上独立训练每个 LSTM。嵌入 LSTM 使用 ADAM 训练，聚类 LSTM 使用 Adagrad 训练。使用的超参数参见原文附录。

实验测试的度量包括准确率和召回率。在实验中，对每个模型做出 10 次预测。如果在头部 10 次预测中给出了完全正确的地址差值，就认为生成了一次准确预测。

## 模型对比情况

实验将基于 LSTM 的预取器与两种硬件预取器进行了对比。第一种是标准流预取器。为保持机器学习预取器与传统预取器的一致性，实验中模拟了支持多至 10 个并发流的硬件结构。第二种是 GHB PC/DC 预期器。这是一种关联预取器，它使用了两个表。一个表用于存储 PC，并将所存储的 PC 作为执行第二个表的指针。第二个表存储了地址差值的历史信息。在每次访问时，GHB 预取器跳转到第二个表，预取历史记录的地址差值。关联预取器对于复杂内存访问模式表现很好，具有比流预取器更低的召回率。

图 5 给出了不同预取器在各种基准测试数据集上的对比情况。尽管流预取器由于其动态词汇表特性可以获得最高的召回率，但是 LSTM 模型整体表现更好，尤其是准确率。

通过对嵌入 LSTM 模型和聚类与 LSTM 的组合模型，可以看到两种模

型间的准确率不相上下。后者趋向于给出更好的召回率。当然，组合更多的模型将会给出更好的结果，这有待未来的工作去探索。

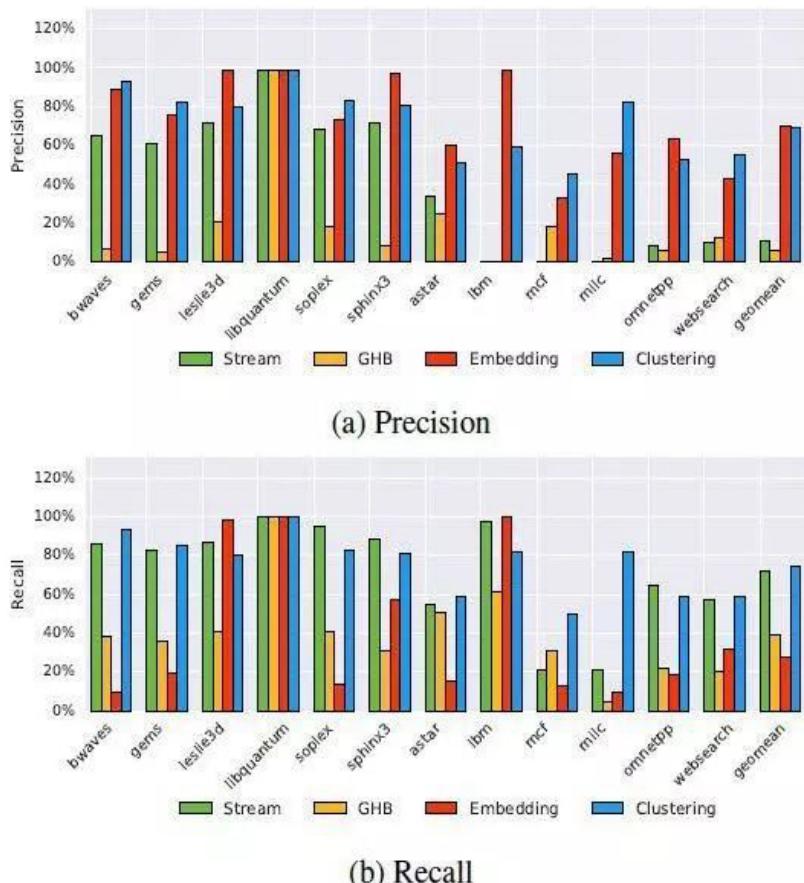


图 5 传统预取器和 LSTM 预取器的准确率和召回率对比图。其中，“Geomean”表示几何平均值

## 对比 PC 的预测情况

该实验从嵌入 LSTM 模型的输入中分别移除或 PC。实验设计意在确定在不同输入模组中所包含的相对信息内容。

实验结果如图 6 所示。从图中可以看出，PC 和地址差值中包含有大量预测信息。地址差值序列中的预测信息可以提高准确率，而 PC 序列有助于提高召回率。

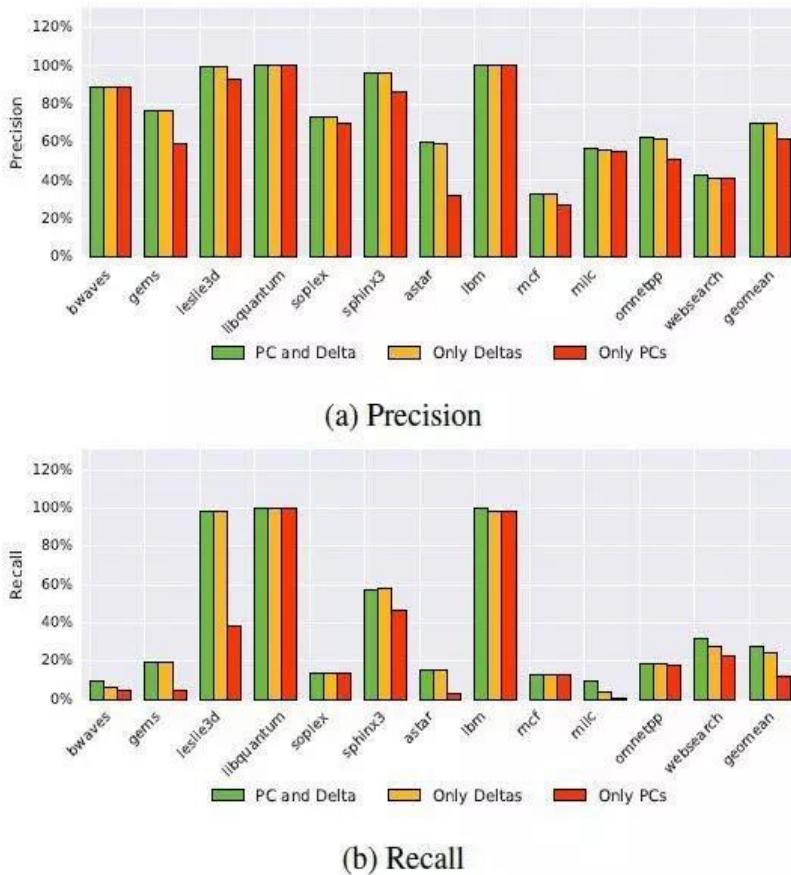


图 6 不同输入模组的嵌入 LSTM 模型，准确率和召回率的对比图

## 解释程序的语义

相比于基于查找表的预期器，模式学习模型的一个主要优点是可以通过审视模型获取数据的内涵。图 7 展示了级联串在 mcf 上嵌入情况的 t-SNE 可视化。

图中的一个聚类是由一组具有统一代码的重复指令组成，这是由于编译器对循环体展开所导致。还有一个聚类仅由指针解除引用（pointer dereference）组成，这是由于程序遍历链接列表所导致的。在 omnetpp 中，对数据结构的插入和移除操作映射为同一个聚类，而数据比较操作则映射在不同聚类中。例子所使用的代码参见原文附录。

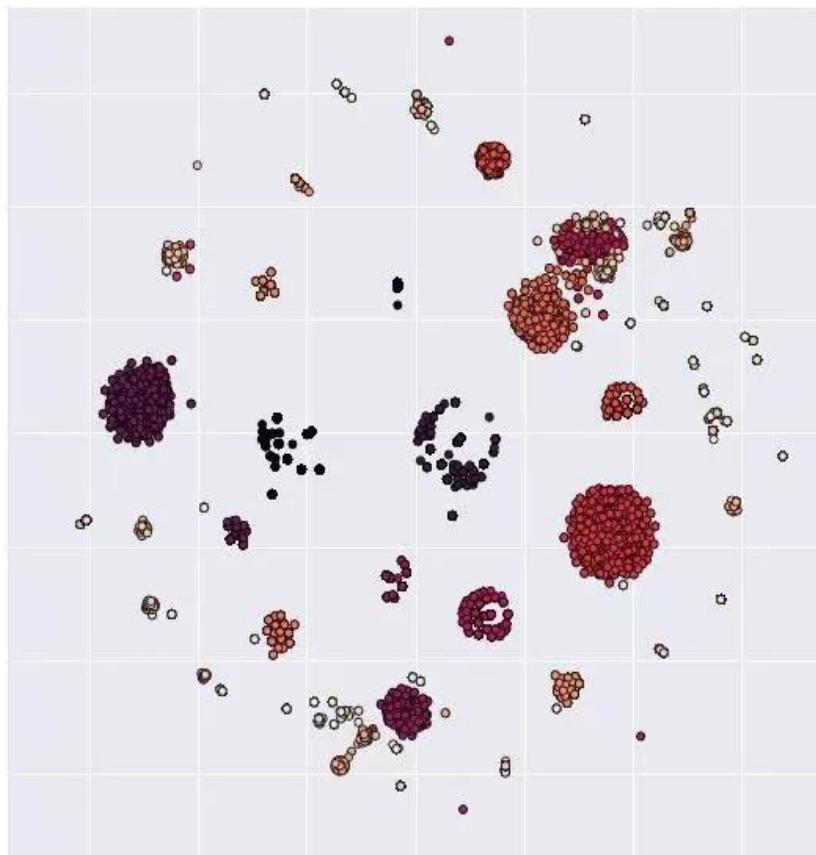


图 7 级联串在 mcf 上嵌入情况的 t-SNE 可视化，图中按指令展示为不同的颜色

## 结论及进一步工作

对应用程序行为的学习和预测，可在解决计算机架构中的控制和数据并发问题中发挥重要作用。传统方法是使用基于表的预测器，但是当扩展到数据密集不规则工作负载时，实现此类方法的代价过大。

本文介绍了两者基于神经网络的预取模型，它们给出了比基于表的传统方法显著高的准确率和召回率。在实验中，论文对模型做离线训练并在线测试，使用准确率和召回率评估了模型。实验表明，论文提出的预取器模型可以改进缓存未命中的分布。当然，还有其它一些在不增加预取器的计算和内存负担条件下改进预期器的方法。在进一步的研究中，可以考虑基于内存命中和未命中数据训练模型，但是这将显著改变数据集的分布和规模。

时效性也是研究预期其中一个考虑的重要因素。对于 RNN 预取器而言，预期过早会导致处理器未使用缓存数据，预取过迟会则对性能影响甚微，因为延迟访问内存的代价已经付出。一个简单的启发式规则是采用类似于流预期器的行为，预先对多个时间步做出预测，而非仅是下一个时间步。

最后一点，对应用性能的影响可以评估 RNN 预取器的有效性。在理想情况下，RNN 将会直接优化应用性能。一个考虑是，使用强化学习技术作为动态环境中 RNN 的训练方法。

当然，内存预取并非计算机系统中唯一需要给出预测执行的领域。只要涉及分支行为，就需要做出预测。分支算法用于重定向控制流的地址，高速缓存替换算法在需要做出替换决定时预测从高速缓存的最佳替换处。如果采用机器学习系统代替传统微体系结构中的启发式规则，可以通过对此类系统审视，更好地理解系统的行为。论文中的 t-SNE 实验仅给出了一些表面上的观察，展示了内存访问模式是程序行为的一种表示。这表明，利用最新研究的 RNN 系统，为计算机系统结构研究提供了大量的机会。



全球技术领导力峰会

BEIJING · 2018

聚变

| 加入 TGO 鲲鹏会与 500 位 CTO 共同成长  
成为 TGO 会员，尊享 0 元购票 >>

## 大会简介

GTLC全球技术领导力峰会是由极客邦旗下TGO鲲鹏会主办的高端技术领导人盛会，大会倾力策划优质分享与活动，以社交为核心，为技术领导者提供学习交流、提升视野与拓展人脉的平台。

本届GTLC将邀请互联网及传统行业权威技术领袖，面向CTO、技术VP、技术团队LEADER、技术项目负责人等对领导力感兴趣的技术人，以大会演讲、深度培训、高端社交等多种形式，从不同领域、不同方向，分享他们关于技术、行业、商业、投资、领导力的实践与蜕变经验。

## 大会主题

技术影响力

技术规划与选型

新热技术落地评估

组织与管理

洞察力与决策力

商业运营与VC思维



扫码了解更多