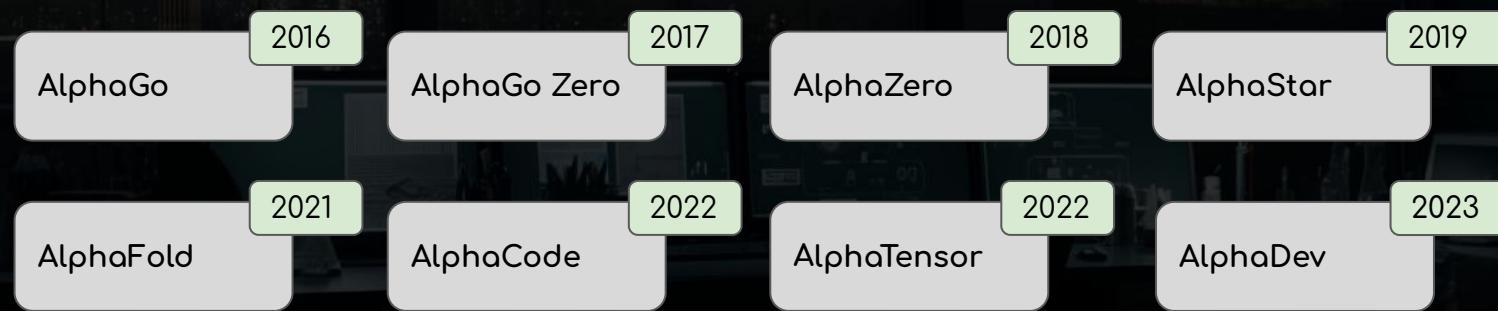


浅谈 DeepMind 的 8 篇“Alpha”冠名论文



少剑

hululu.zhu@gmail.com

07/09/2023

关于少剑同学

博士实习 + 毕业后在 Google + DeepMind 工作超过 11 年，共 8 个部门



平时喜欢做一些人工智能(AI) 的科普。

- 因为传递和分享知识是一种快乐，也是一种信仰。

声明

这个讲座里所有的内容均基于 DeepMind 公开的论文、博客文章、社交媒体讨论和公开演示。和少剑的本职工作无直接联系。

本幻灯片中的所有观点均只是少剑他的个人观点，与 Google DeepMind 无关

做这次 Alpha 冠名论文讲座的动机

- DeepMind 和 Google 的母公司叫 Alphabet, 所以用“Alpha”来冠名的论文, 我个人感觉分量比较重比较酷一些
- 很多人都听说过 AlphaGo, AlphaFold 等研究, 或者最近的 AlphaTensor 和 AlphaDev, 但很少人了解背后具体问题的抽象, 和基于AI的大致解决方案
- 沿着 Alpha 论文系列的时间线, 我们或许也可以看到一点 DeepMind 在选择研究问题和AI技术上的思路
- 个人没看到有谁做过完整的所有 Alpha 冠名论文的科普讲座

那么, 我就试试看

那我们开始把，先来看一下 8 篇论文

- 2016 AlphaGo 下围棋(使用人类棋局)
- 2017 AlphaGo Zero 下围棋(不用人类棋局)
- 2018 AlphaZero 玩多种棋类
- 2019 AlphaStar 玩转星际争霸2
- 2021 AlphaFold 蛋白质预测
- 2022 AlphaCode 解计算机竞赛题
- 2022 AlphaTensor 寻找更快的矩阵乘法的算法
- 2023 AlphaDev 提高排序算法算法速度

那我们开始把，先来看一下 8 篇论文

- 2016 AlphaGo 下围棋(使用人类棋局)
- 2017 AlphaGo Zero 下围棋(不用人类棋局)
- 2018 AlphaZero 玩多种棋类
- 2019 AlphaStar 玩转星际争霸2
- 2021 AlphaFold 蛋白质预测
- 2022 AlphaCode 解计算机竞赛题
- 2022 AlphaTensor 寻找更快的矩阵乘法的算法
- 2023 AlphaDev 提高排序算法算法速度

“不务正业”的游戏公司

那我们开始把，先来看一下 8 篇论文

- 2016 AlphaGo 下围棋(使用人类棋局)
- 2017 AlphaGo Zero 下围棋(不用人类棋局)
- 2018 AlphaZero 玩多种棋类
- 2019 AlphaStar 玩转星际争霸2
- 2021 AlphaFold 蛋白质预测
- 2022 AlphaCode 解计算机竞赛题
- 2022 AlphaTensor 寻找更快的矩阵乘法的算法
- 2023 AlphaDev 提高排序算法算法速度

“改邪归正”的科研公司

AlphaGo - 为什么下好围棋很难？

理性的解释

- 19×19的棋盘，每一步有几百个落子点，每一步都引发连锁反应
- 所以，理论上所有围棋的局面数量超过 10^{170} ！
- 围棋的局势评估非常困难！（反正我不会）
- 围棋要兼顾长期的战略规划，和灵活的局部战术



AlphaGo - 为什么下好围棋很难？

理性的解释

- 19×19的棋盘，每一步有几百个落子点，每一步都引发连锁反应
- 所以，理论上所有围棋的局面数量超过 10^{170} ！
- 围棋的局势评估非常困难！（反正我不会）
- 围棋要兼顾长期的战略规划，和灵活的局部战术

感性的解释

- 摘自金庸《天龙八部》“玲珑棋局”

“这个玲珑变幻百端，因人而施，爱财者因贪失误，易怒者由愤坏事。段誉之败，在于爱心太重，不肯弃子；慕容复之失，由于执着权势，勇于弃子，却说什么也不肯失势。段延庆生平第一恨事，乃是残废之后，不得不抛开本门正宗武功，改习旁门左道的邪术，一到全神贯注之时，外魔入侵，竟尔心神荡漾，难以自制。”

1/8: AlphaGo - 整体思路: 从模仿到超越

模仿

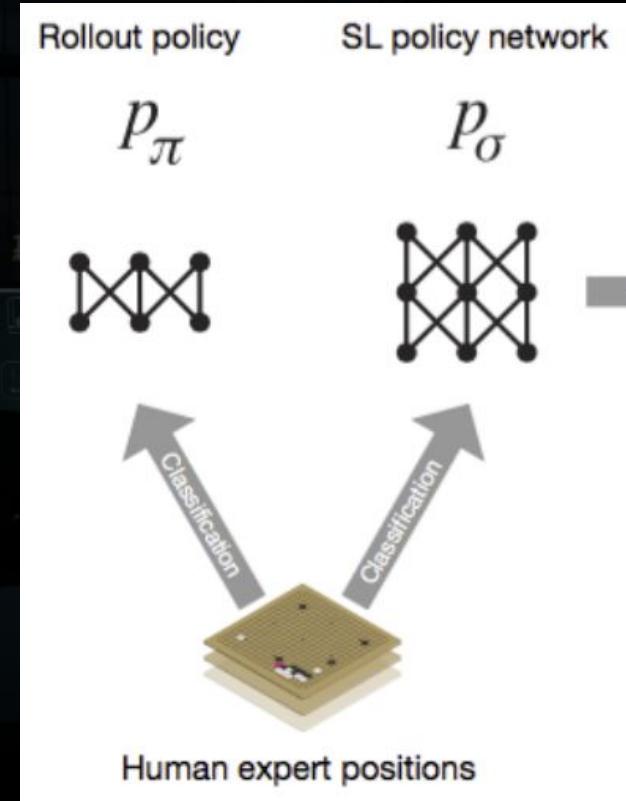
- 模仿高手(或准高手)的对局
 - 我也能“照虎画猫”

超越

- 如何评估当前局势?
 - 我要把握“战略”
- 如何选取最佳的几个落子点?
 - 我要钻研“专属技巧”
- 如何推演未来棋局变化实现最佳落子的判断?
 - 平衡战略和战术, 起飞!

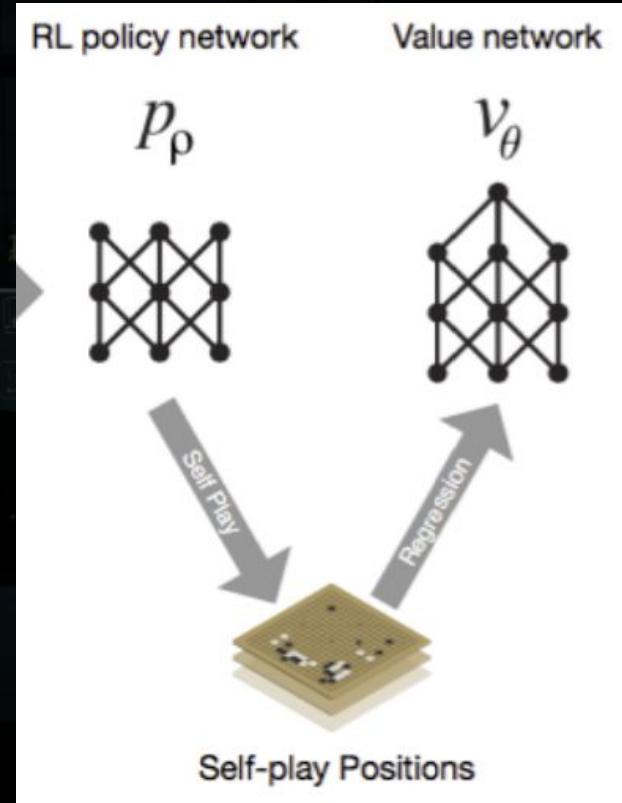
1/8: AlphaGo - 模仿

- 用下面的数据训练 AI
 - 输入数据: 几十万局欧美高手(亚洲准高手水平)的历史对战棋局
 - AI预测: 高手的下一步操作
- 具体技术实现
 - 上面一样的训练方法, 但训练 2 个AI
 - Rollout Policy
一个高速小型AI, 用来大规模快速推演, 但模仿能力有限
 - SL Policy
一个低速大型AI, 模仿更到位



1/8: AlphaGo - 超越

- 具体技术实现
 - 在低速模仿SL Policy的基础上训练新的RL Policy, 需要在SL(单纯模仿)基础上进化提高
 - 同时, 增加局势判断AI (Value network)来用0.0-1.0来判断局势
 - Value network来做全局的判断, 类似于“战略”, RL Policy则通过自我博弈提高单步(战术技巧)的提高
 - 最终目标是实现全局的最优(Value network的预测值越高, 越接近胜利)
- 自我博弈几亿盘, 麻雀变凤凰



1/8: AlphaGo - MCTS 整合下的模型推理

MCTS ?

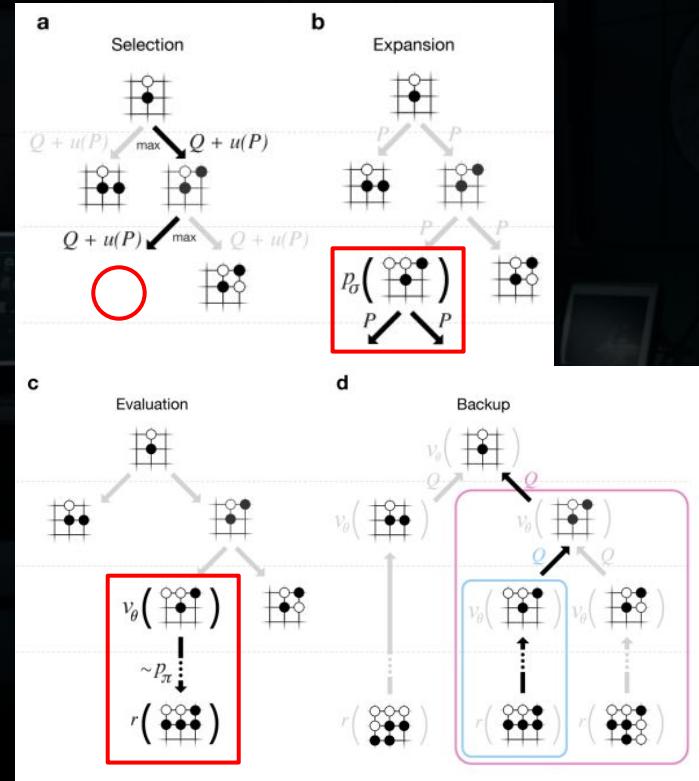
- 蒙特卡洛树搜索(Monte Carlo Tree Search)大量的随机对局来估计判断
- 前面训练完的快速模仿, 低速模仿网络以及评估网络会在这里用MCTS的方式整合后使用

a 选择 Selection: 选择一个起点位置。

b 扩展 Expansion: 从起点扩展到多子节点

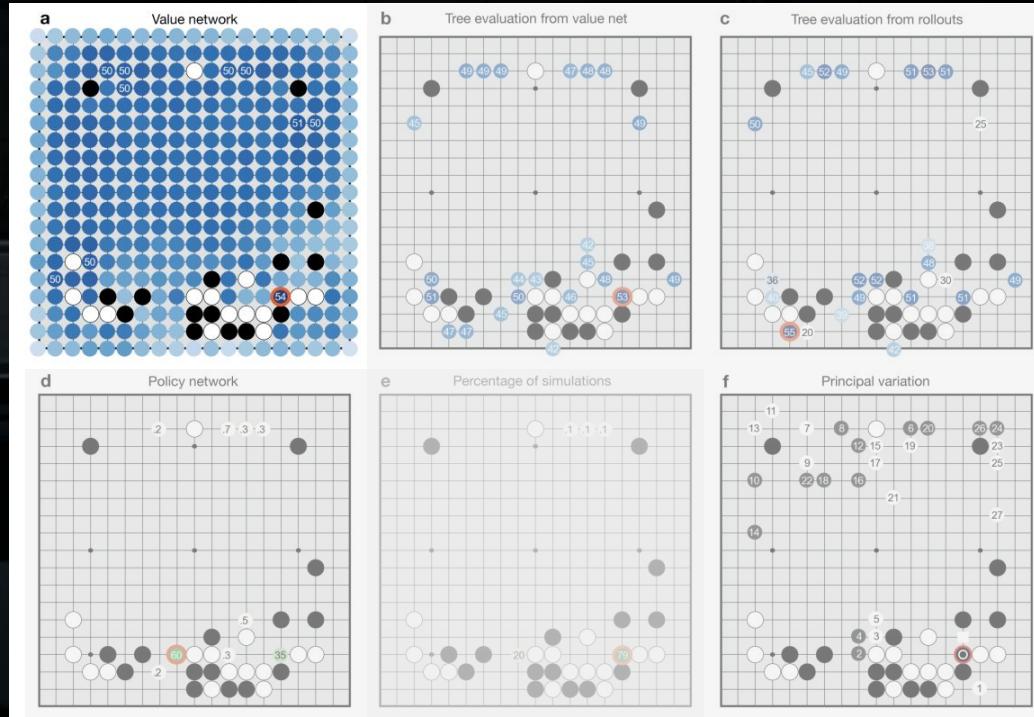
c 评估 Evaluation: 子节点开始, 随机模拟进行完整的模拟对局, 直到游戏结束

d 回溯 Backup: 根据模拟对局的结果, 更新搜索树路径上的统计数据, 例如胜利次数和访问次数。



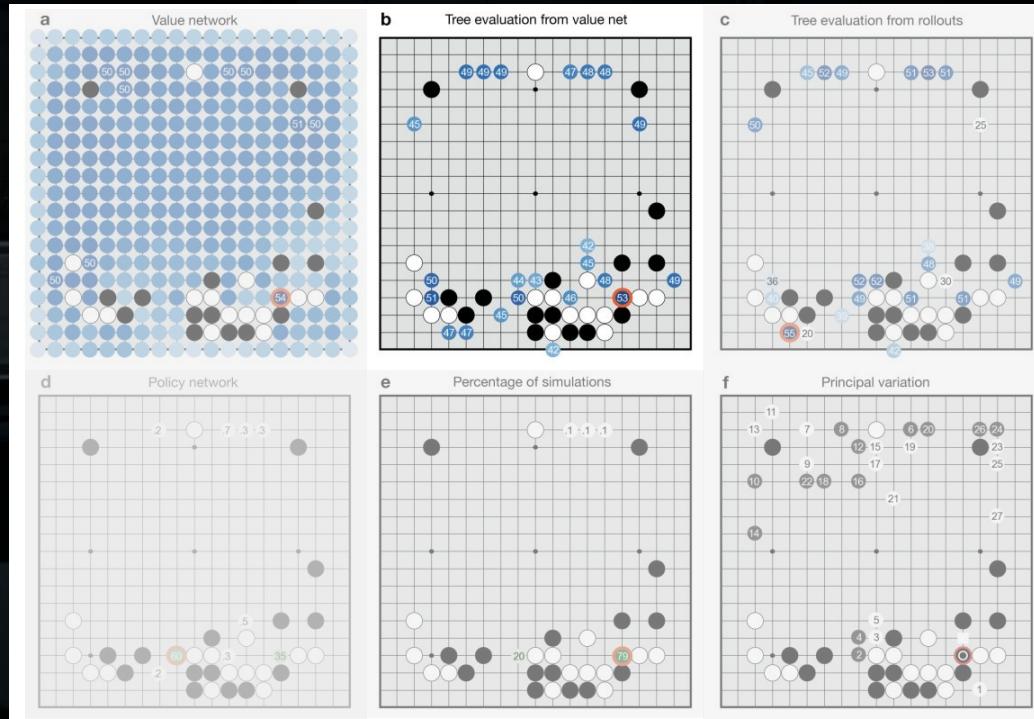
1/8: AlphaGo - MCTS 推理的具体示例

- a. 左上角，每一个蓝色位置作为下一个黑子落点，估算的胜率，靠近右下角54% 是局势判断下的最优
- b. 中上，通过低速模仿网络评估后选择有限的“好”位置，所以只有部分落子有胜率，同a最佳位置，但胜率降为 53%
- c. 右上，通过高速(低模仿力)的小网络评估后选择有限的“好”位置，可以看到，这里左下角有一个55%的位置
- d. 左下，只听从低速模仿网络，在中下的位置有一个60%胜率的位置
- e. 中下，MCTS网络对于各个“有潜力”起点位置的挖掘，可以看到79%的时候(非胜率)，网络选择从这一步开始
- f. 右下，综合上面信息，右下红点为最佳位置，其他数字位置为排序后的其他位置



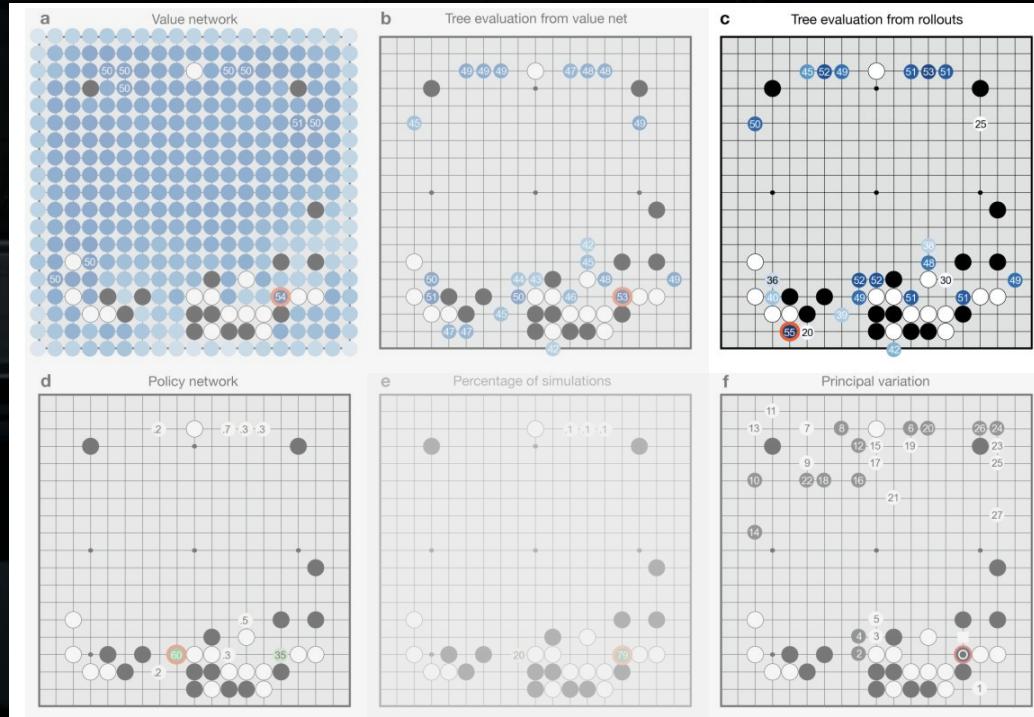
1/8: AlphaGo - MCTS 推理的具体示例

- a. 左上角, 每一个蓝色位置作为下一个黑子落点, 估算的胜率, 靠近右下角54% 是局势判断下的最优
- b. 中上, 通过低速模仿网络评估后选择有限的“好”位置, 所以只有部分落子有胜率, 同a最佳位置, 但胜率降为 53%
- c. 右上, 通过高速(低模仿力)的小网络评估后选择有限的“好”位置, 可以看到, 这里左下角有一个55%的位置
- d. 左下, 只听从低速模仿网络, 在中下的位置有一个60%胜率的位置
- e. 中下, MCTS网络对于各个“有潜力”起点位置的挖掘, 可以看到79%的时候(非胜率), 网络选择从这一步开始
- f. 右下, 综合上面信息, 右下红点为最佳位置, 其他数字位置为排序后的其他位置



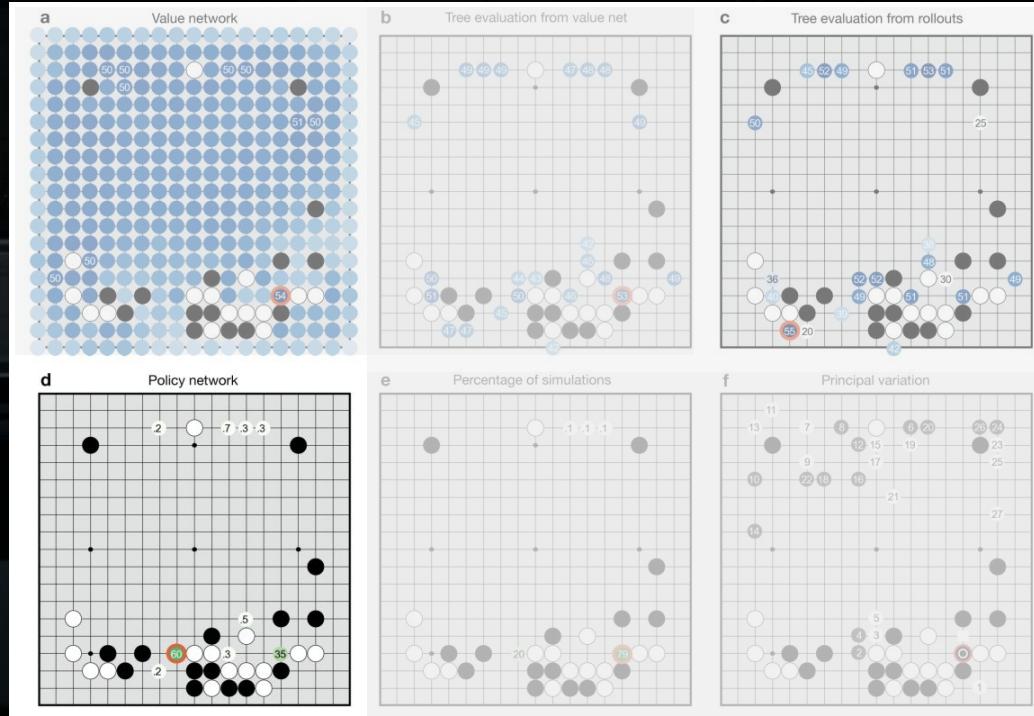
1/8: AlphaGo - MCTS 推理的具体示例

- a. 左上角, 每一个蓝色位置作为下一个黑子落点, 估算的胜率, 靠近右下角54% 是局势判断下的最优
- b. 中上, 通过低速模仿网络评估后选择有限的“好”位置, 所以只有部分落子有胜率, 同a最佳位置, 但胜率降为 53%
- c. 右上, 通过高速(低模仿力)的小网络评估后选择有限的“好”位置, 可以看到, 这里左下角有一个55%的位置
- d. 左下, 只听从低速模仿网络, 在中下的位置有一个60%胜率的位置
- e. 中下, MCTS网络对于各个“有潜力”起点位置的挖掘, 可以看到79%的时候(非胜率), 网络选择从这一步开始
- f. 右下, 综合上面信息, 右下红点为最佳位置, 其他数字位置为排序后的其他位置



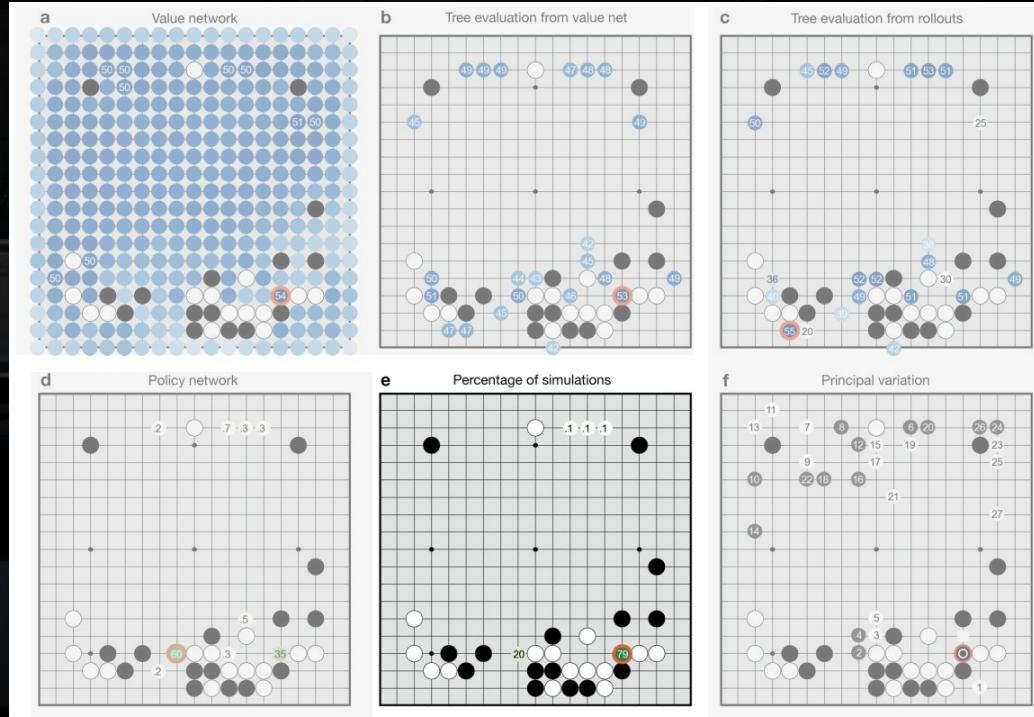
1/8: AlphaGo - MCTS 推理的具体示例

- a. 左上角, 每一个蓝色位置作为下一个黑子落点, 估算的胜率, 靠近右下角54% 是局势判断下的最优
- b. 中上, 通过低速模仿网络评估后选择有限的“好”位置, 所以只有部分落子有胜率, 同a最佳位置, 但胜率降为 53%
- c. 右上, 通过高速(低模仿力)的小网络评估后选择有限的“好”位置, 可以看到, 这里左下角有一个55%的位置
- d. **左下, 只听从低速模仿网络, 在中下的位置有一个60%胜率的位置**
- e. 中下, MCTS网络对于各个“有潜力”起点位置的挖掘, 可以看到79%的时候(非胜率), 网络选择从这一步开始
- f. 右下, 综合上面信息, 右下红点为最佳位置, 其他数字位置为排序后的其他位置



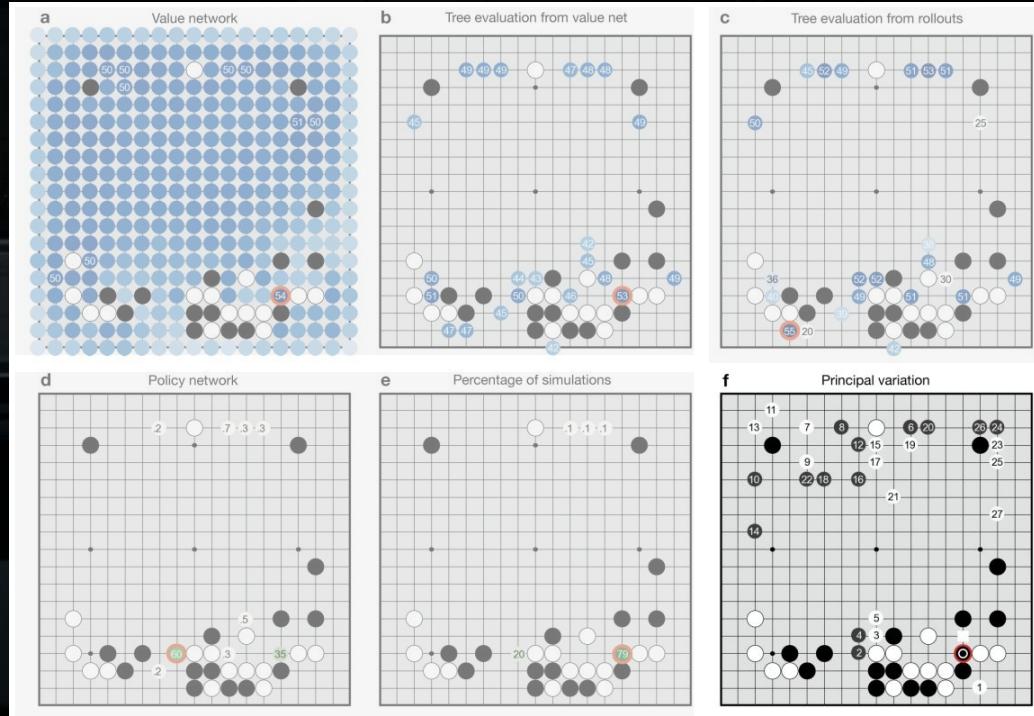
1/8: AlphaGo - MCTS 推理的具体示例

- a. 左上角, 每一个蓝色位置作为下一个黑子落点, 估算的胜率, 靠近右下角54% 是局势判断下的最优
 - b. 中上, 通过低速模仿网络评估后选择有限的“好”位置, 所以只有部分落子有胜率, 同a最佳位置, 但胜率降为 53%
 - c. 右上, 通过高速(低模仿力)的小网络评估后选择有限的“好”位置, 可以看到, 这里左下角有一个55%的位置
 - d. 左下, 只听从低速模仿网络, 在中下的位置有一个60%胜率的位置
 - e. 中下, MCTS网络对于各个“有潜力”起点位置的挖掘, 可以看到79%的时候(非胜率), 网络选择从这一步开始
 - f. 右下, 综合上面信息, 右下红点为最佳位置, 其他数字位置为排序后的其他位置



1/8: AlphaGo - MCTS 推理的具体示例

- a. 左上角, 每一个蓝色位置作为下一个黑子落点, 估算的胜率, 靠近右下角54% 是局势判断下的最优
- b. 中上, 通过低速模仿网络评估后选择有限的“好”位置, 所以只有部分落子有胜率, 同a最佳位置, 但胜率降为 53%
- c. 右上, 通过高速(低模仿力)的小网络评估后选择有限的“好”位置, 可以看到, 这里左下角有一个55%的位置
- d. 左下, 只听从低速模仿网络, 在中下的位置有一个60%胜率的位置
- e. 中下, MCTS网络对于各个“有潜力”起点位置的挖掘, 可以看到79%的时候(非胜率), 网络选择从这一步开始
- f. 右下, 综合上面信息, 右下红点为最佳位置, 其他数字位置为排序后的其他位置



2/8: AlphaGo Zero 核心问题

AlphaGo经历了 2 个阶段, 模仿和超越

- AI 对于高手的模仿是不是必须？
- 有没有可能抛弃模仿高手而直接自我博弈实现突破？

答案大家都知道了, yes ! AlphaGo Zero就是答案

- Zero这里就是不使用或模仿高手数据

nature

Search Login

Explore content ▾ About the journal ▾ Publish with us ▾

Published: 19 October 2017

Mastering the game of Go without human knowledge

David Silver , Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel & Demis Hassabis

Nature 550, 354–359 (2017) | [Cite this article](#)

319k Accesses | 3103 Citations | 2570 Altmetric | [Metrics](#)

Abstract

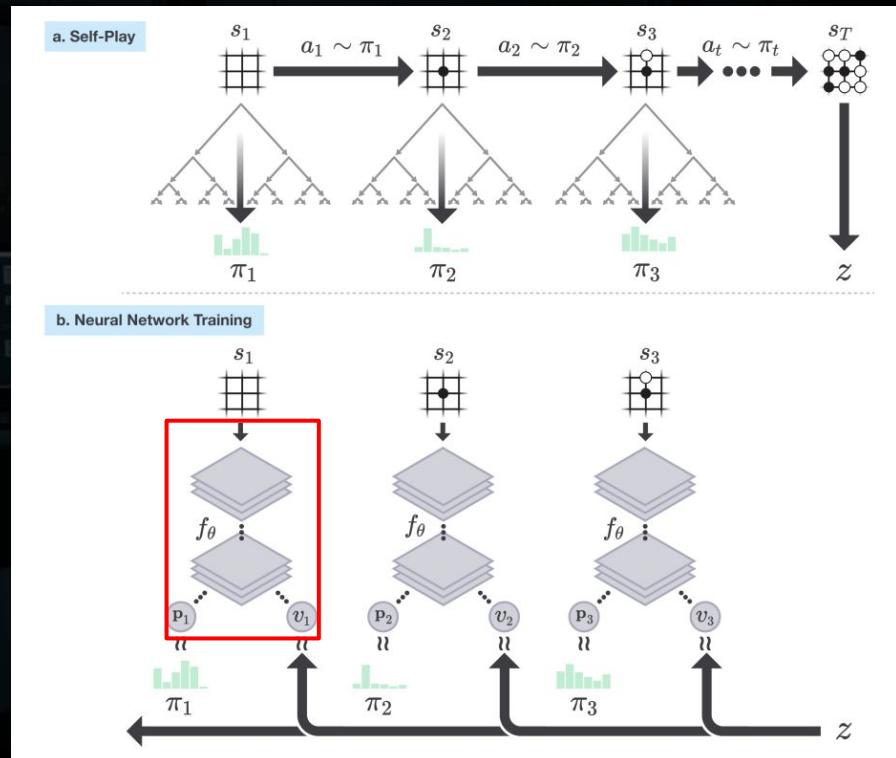
A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the

2/8 AlphaGo Zero: 整体思路

- 之前AlphaGo只在训练结束后各模型推理阶段使用MCTS
- 从模型优化的角度看这里有个很大的区别
 - 训练时候: 在最终优化 value network (胜率预测) 基础上训练自我博弈的 RL policy
 - 推理时候: 使用训练好的多个模型(包括模仿模型)用 MCTS做搜索树的优化
 - 两者不统一, 很不统一 !
- 所以
 - MCTS如果能放进训练过程, 那问题就解决了
 - 这就是AlphaGo Zero比起AlphaGo的核心变化 !

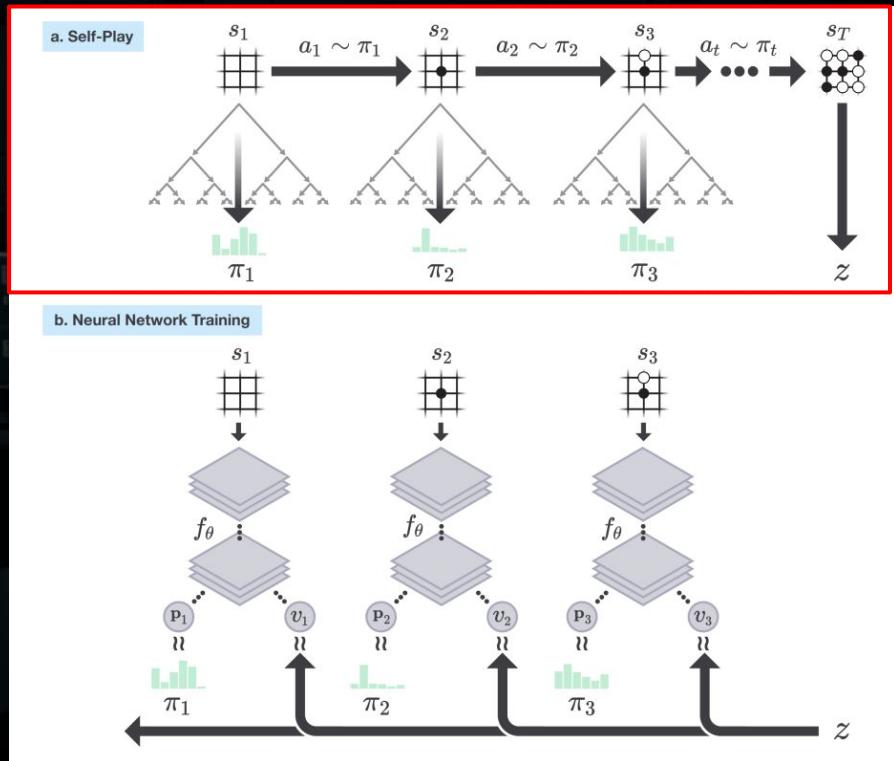
2/8 AlphaGo Zero: 具体实现

- a. AlphaGo Zero模型是一个模型，2个输出
 - i. Alphago有4个网络(2个模仿, 一个RL, 一个value)
 - ii. 一个是策略网络, 输出不同位置的概率(图上显示 ρ)
 - iii. 另一个是价值网络(理解为胜率估计, 图上显示 v)
- b. Self-Play 自我博弈
 - i. 在下棋的每一步, 使用 ρ 和 v 的结合调用MCTS, 大约1600次推演
 - ii. 综合得出下一步落子概率 π
 - iii. 一直到游戏结束
- c. Neural Network Training 模型训练
 - i. 根据游戏胜负, 回溯到游戏开始, 指导 v (胜率)的优化
 - ii. 根据 MCTS 得出的 π , 指导策略网络 ρ 的优化
- d. 重复亿次, 麻雀石头变凤凰



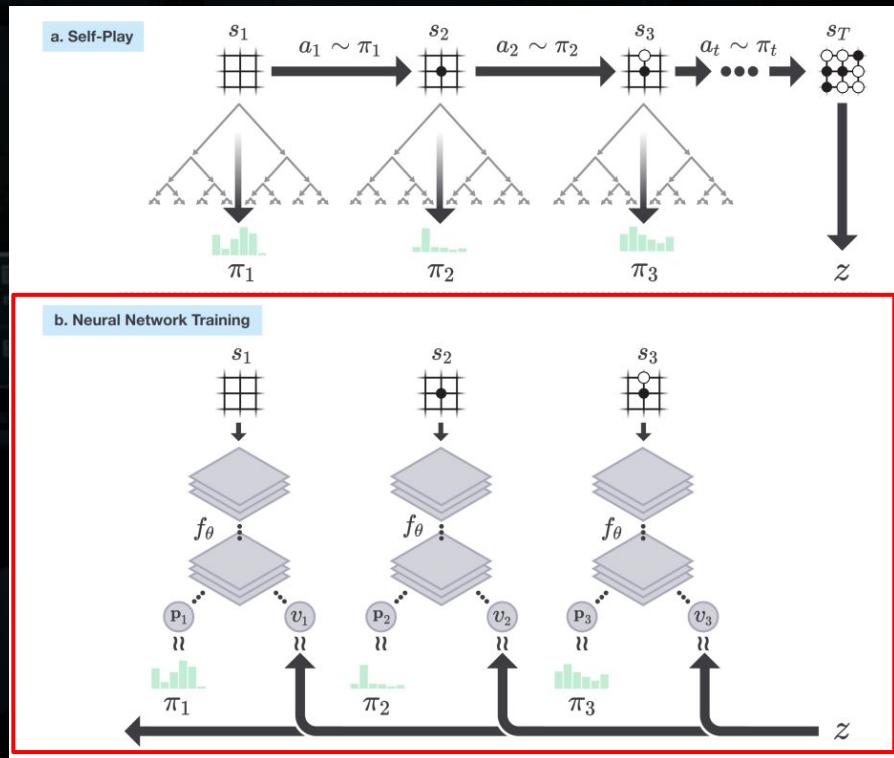
2/8 AlphaGo Zero: 具体实现

- a. AlphaGo Zero模型是一个模型，2个输出
 - i. Alphago有4个网络(2个模仿, 一个RL, 一个value)
 - ii. 一个是策略网络, 输出不同位置的概率(图上显示 ρ)
 - iii. 另一个是价值网络(理解为胜率估计, 图上显示 v)
- b. Self-Play 自我博弈
 - i. 在下棋的每一步, 使用 ρ 和 v 的结合调用MCTS, 大约1600次推演
 - ii. 综合得出下一步落子概率 π
 - iii. 一直到游戏结束
- c. Neural Network Training 模型训练
 - i. 根据游戏胜负, 回溯到游戏开始, 指导 v (胜率)的优化
 - ii. 根据MCTS得出的 π , 指导策略网络 ρ 的优化
- d. 重复亿次, 麻雀石头变凤凰



2/8 AlphaGo Zero: 具体实现

- a. AlphaGo Zero模型是一个模型，2个输出
 - i. Alphago有4个网络(2个模仿, 一个RL, 一个value)
 - ii. 一个是策略网络, 输出不同位置的概率(图上显示 ρ)
 - iii. 另一个是价值网络(理解为胜率估计, 图上显示 v)
- b. Self-Play 自我博弈
 - i. 在下棋的每一步, 使用 ρ 和 v 的结合调用MCTS, 大约1600次推演
 - ii. 综合得出下一步落子概率 π
 - iii. 一直到游戏结束
- c. Neural Network Training 模型训练
 - i. 根据游戏胜负, 回溯到游戏开始, 指导 v (胜率)的优化
 - ii. 根据MCTS得出的 π , 指导策略网络 ρ 的优化
- d. 重复亿次, 麻雀石头变凤凰



3/8 AlphaZero 核心问题

AlphaGo Zero在没有使用人类经验的基础上打败了李世石和柯洁，那类似的方法是不是可以用在更多的地方？

yes ! AlphaZero 能下围棋，香气，日本将棋！



3/8 AlphaZero 具体的实现

- 和 AlphaGo Zero 相同的一个网络架构
 - v 和 ρ 是同一个神经网络模型的 2 个输出
 - 价值网络 v 评估胜率
 - 策略网络 ρ 判断下一步概率
 - MCTS 用在自我博弈 self-play 阶段
 - 自我博弈的数据用来优化 v 和 ρ
- 哪些不同？
 - 棋盘的形状(比如 chess 是 8x8, 围棋是 19x19)
 - 棋盘的对称性(围棋可以旋转或镜像, 其他棋类不行)
 - 围棋 AI 没有和棋, 但 Chess 有
 - 具体的训练优化
 - AlphaGo Zero 有一个互相比赛的机制来阶段性得提升网络(两个网络对打, 新网络要赢过 55% 以上才会认为是更优得模型)
 - AlphaZero 就是一个连续的优化模型的过程

3/8 AlphaZero 相关的几个工作

nature Search Login

Explore content ▾ About the journal ▾ Publish with us ▾

Article | Published: 23 December 2020

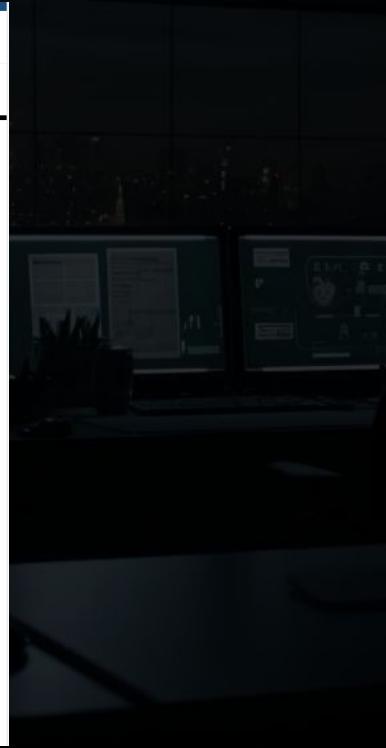
Mastering Atari, Go, chess and shogi by planning with a learned model

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap & David Silver

Nature 588, 604–609 (2020) | Cite this article
35k Accesses | 158 Citations | 1601 Altmetric | Metrics

Abstract

Constructing agents with planning capabilities has long been one of the main challenges in the pursuit of artificial intelligence. Tree-based planning methods have enjoyed huge success in challenging domains, such as chess¹ and Go², where a perfect simulator is available. However, in real-world problems, the dynamics governing the environment are often complex and unknown. Here we present the MuZero algorithm, which, by combining a tree-based search with a learned model, achieves superhuman performance in a range of challenging and visually complex domains, without any knowledge of their underlying



DeepMind 2022-5-13

A Generalist Agent

Scott Reed^{*†}, Konrad Żolna^{*}, Emilio Parisotto^{*}, Sergio Gómez Colmenarejo^{*}, Alexander Novikov, Gabriel Barth-Maron, Mai Giménez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar and Nando de Freitas[†]

^{*}Equal contributions, [†]Equal senior contributions, All authors are affiliated with DeepMind

Inspired by progress in large-scale language modeling, we apply a similar approach towards building a single generalist agent beyond the realm of text outputs. The agent, which we refer to as Gato, works as a multi-modal, multi-task, multi-embodiment generalist policy. The same network with the same weights can play Atari, caption images, chat, stack blocks with a real robot arm and much more deciding based on its context whether to output text, joint torques, button presses, or other tokens. In this report we describe the model and the data, and document the current capabilities of Gato.

4/8 AlphaStar 核心问题

棋类游戏已经攻克了

Atari类似的小游戏(比如pac-man)也被攻克了

是时候挑战更难的游戏了，哪一个游戏最难？

- RTS实时战略游戏, StarCraft星际争霸很难
 - DeepMind和一些其他公司选择StarCraft
- MOBA多人在线竞技游戏, DOTA2非常难
 - OpenAI选择DOTA2



4/8 AlphaStar 玩好星际为什么很难？

和围棋比一比，对于AI要玩好星际比玩好围棋可能要难 10^6 也就是百万倍！

- 围棋19x19棋盘，星际有1024x768或更多的像素 (100x)
- 围棋一个点控制黑或白一方，星际同时控制几十个单位(陆地，空军，建筑，盟友，敌人指向) (10x)
- 围棋都是摆在台面上的(虽然玲珑棋局让人吐血)，星际有地图迷雾，只能看到有限的区域 (10x)
- 和围棋一样，需要兼顾战略(比如资源分配，陆地还是空军)和战术(微操击杀或躲避能力) (1x)
- 围棋最多300多步，星际按照每秒20帧画面，20分钟就要24000步！ (100x)
- AlphaGo可以思考几分钟下一步棋，星际需要在~200毫秒做出反应！ (100x)

4/8 AlphaStar 整体思路

- 因为可能的下一步操作(什么单位那个位置做什么行为)太多, 如果用AlphaZero从头学起, 可能非常浪费资源
- 所以我们可以学习AlphaGo, 先学习人类高手的操作
 - 为了简化目标, 把建筑和资源的安排作为另外一个叫 z 的目标模型 去学习
 - 在 z 的基础上, 学习每一个时间点, 人类高手的操作(什么单位那个位置做什么行为)
- 在模型有了基础模仿能力之后, 自我博弈提高
 - 但是, 这里怎么证明“提高”需要更多思考
 - AlphaGo团队给出了一个league(联赛)的概念

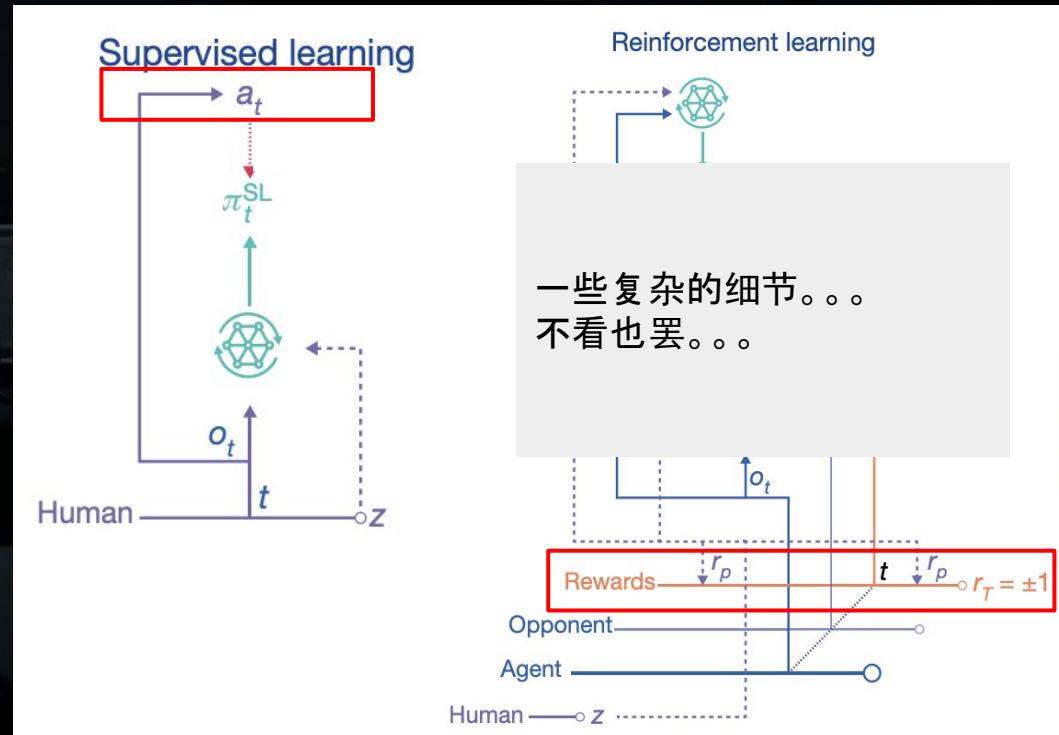
4/8 AlphaStar 设计

监督学习(模仿)

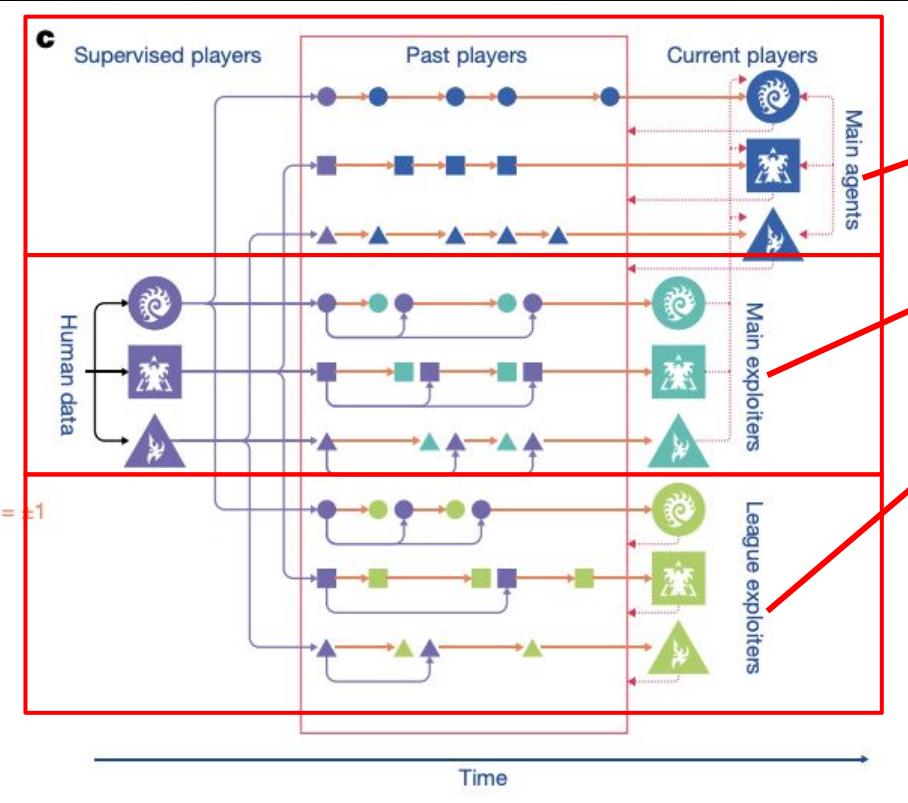
- Supervised learning
- 输入高手游戏的历史和当前游戏信息 o
- 模仿高手的行为 a
- z 作为战略上的指导进行约束

强化学习(自我博弈提高)

- 最重要的Rewards代表游戏的输赢，需要最大化！
- 自我博弈后，输赢的结果可以作为Rewards的参考去优化模型的评估



4/8 AlphaStar 实现



3 组 AI, 一共 12 个在训 AI

- Main Agent 组 (3)

- 作为核心智能AI, 和所有三组AI博弈提高

- Main exploiters 组 (3)

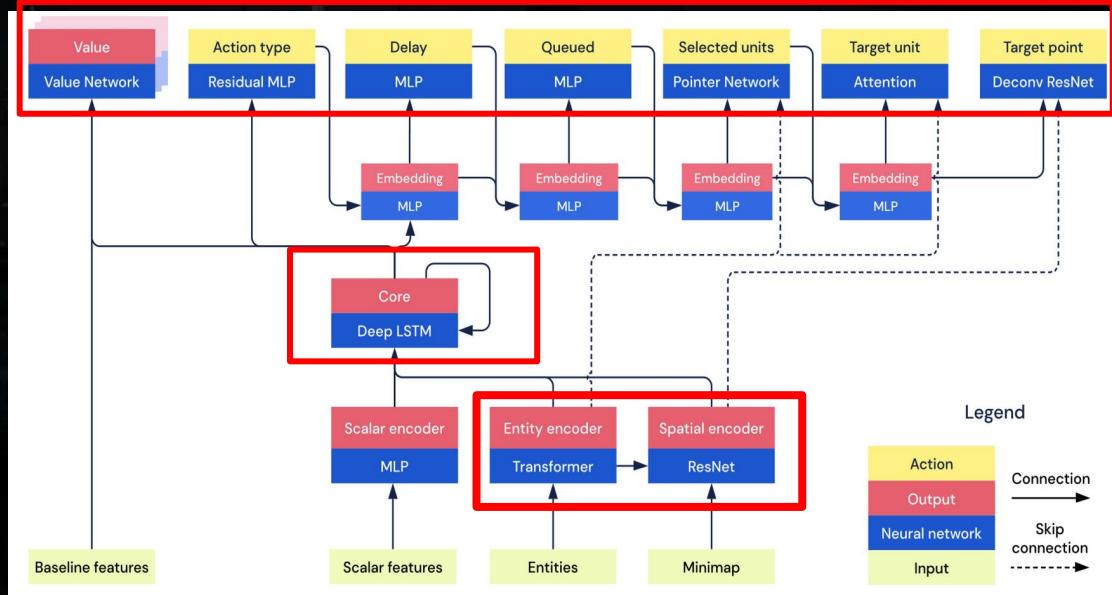
- 专门研究核心组AI的弱点

- League exploiters 组 (6)

- 尝试寻找所有AI的弱点

除了核心组, 其他 9 个 AI 都要定期更新来尝试不一样的攻坚技巧来尝试赢过核心组

4/8 AlphaStar 网络架构



- LSTM 是一个处理序列(游戏历史)的常用网络
 - ResNet是处理小地图图像的视觉网络
- 模型要输出
- Value, 当前局势判断
 - Action, 包括延迟多久对哪个单位在哪个位置做出什么动作

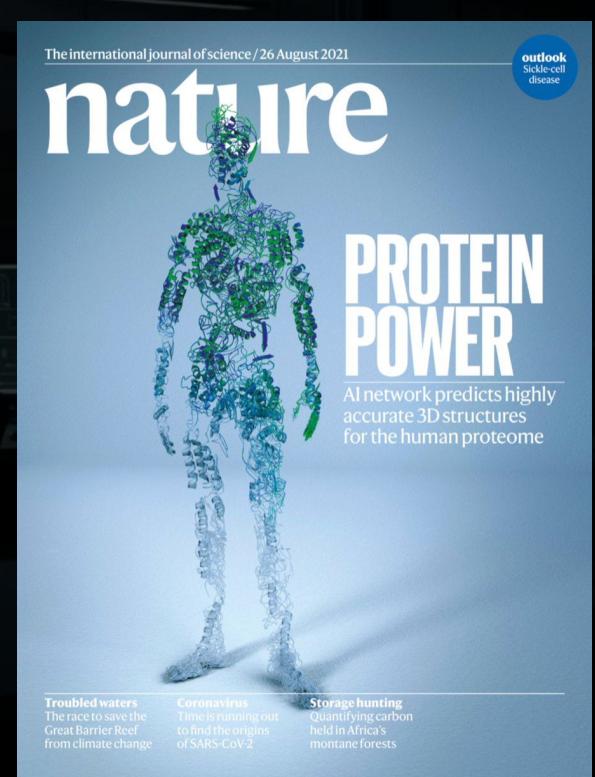
5/8 AlphaFold 动机猜测

游戏玩得够够的了，是时候搞点研究了！

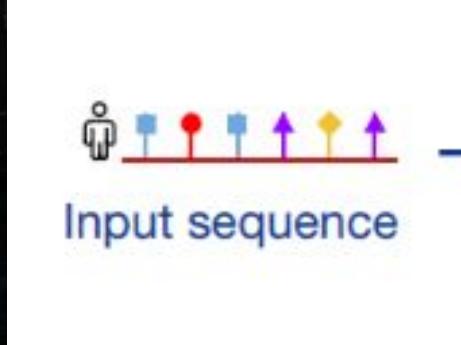
- 蛋白质的结构(而非氨基酸的组成)极大影响蛋白质的功能，氨基酸序列比较容易测量，但它们组成的稳定结构非常难预测
 - 就好比，给你很多乐高积木，但不告诉你它能搭成的物体，你能不能用全部的乐高积木来搭一个稳定的结构？

这是一个困扰人类50年的问题！

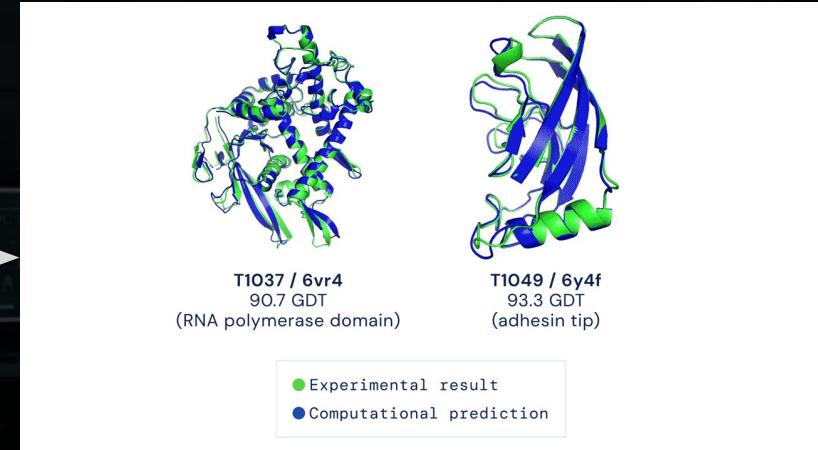
要是搞出来，可以考虑申请诺贝尔奖了！！



5/8 AlphaFold 具体的问题



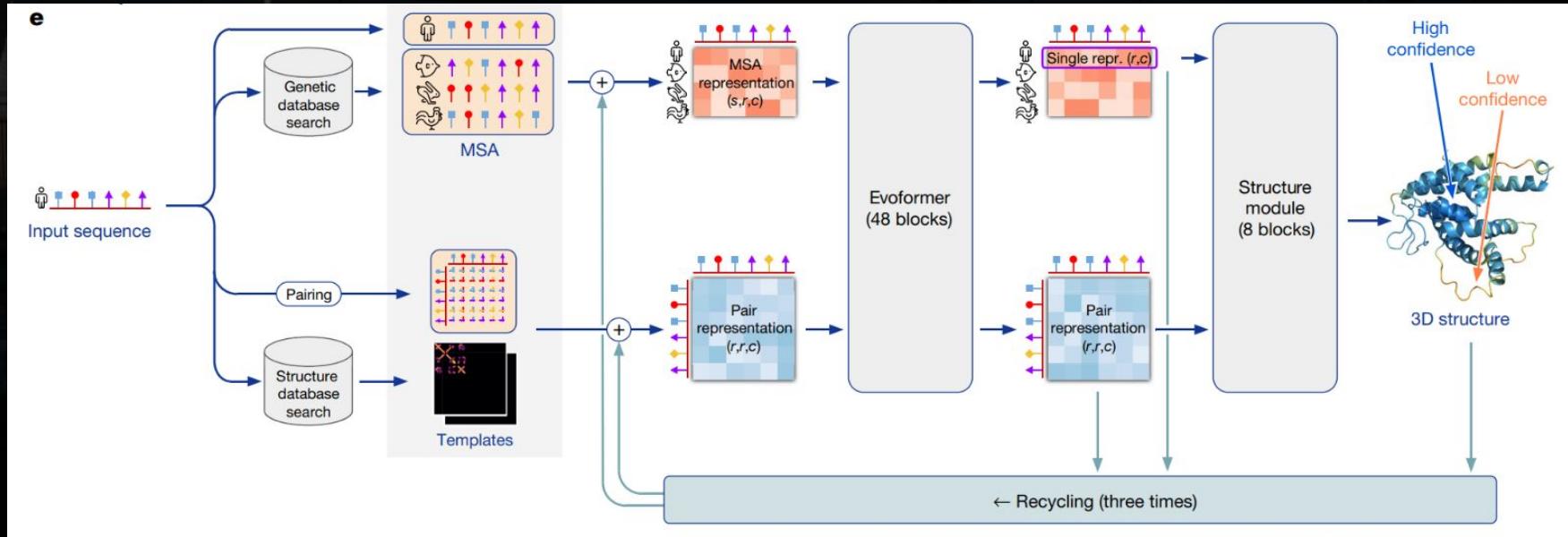
怎么预测？



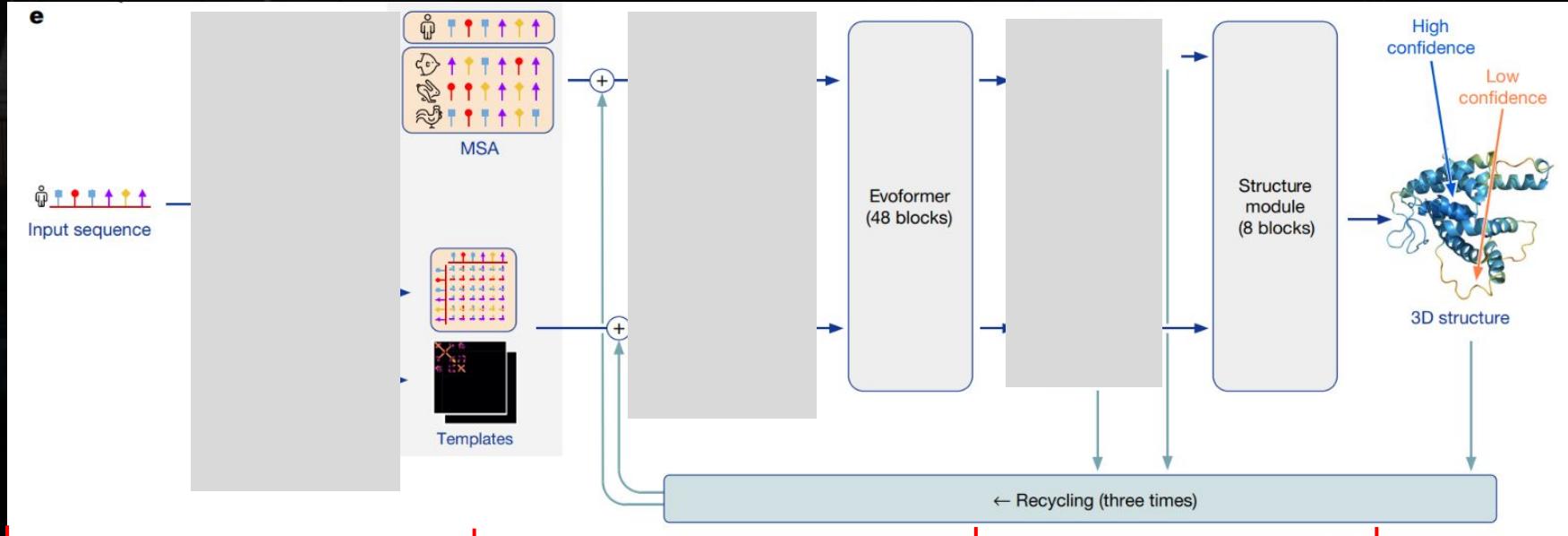
数据输入：
组成蛋白质的氨基酸序列

预测的“答案”：
稳定的蛋白质的3D结构组成

5/8 AlphaFold 的设计



5/8 AlphaFold 的【简化】结构和细节分析

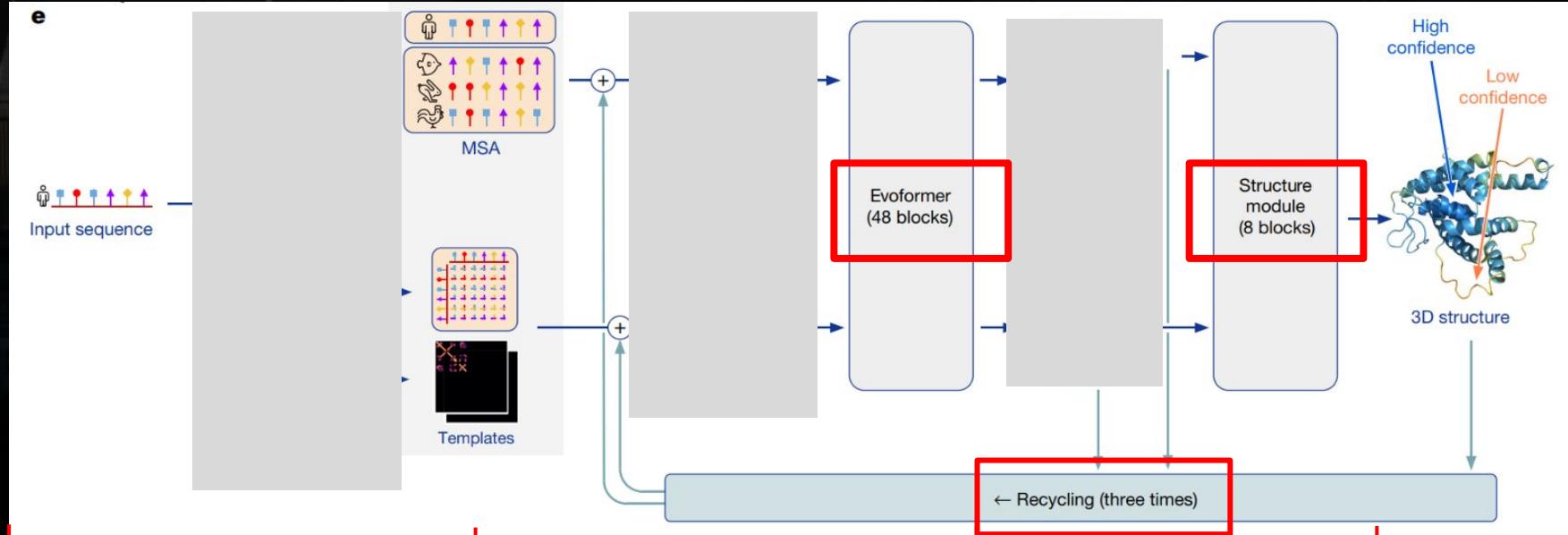


各种方法提取氨基酸碱基序列的特征

特征的融合和编码
(encoder)

预测结构的解码
(decoder)

5/8 AlphaFold 的结构和方案



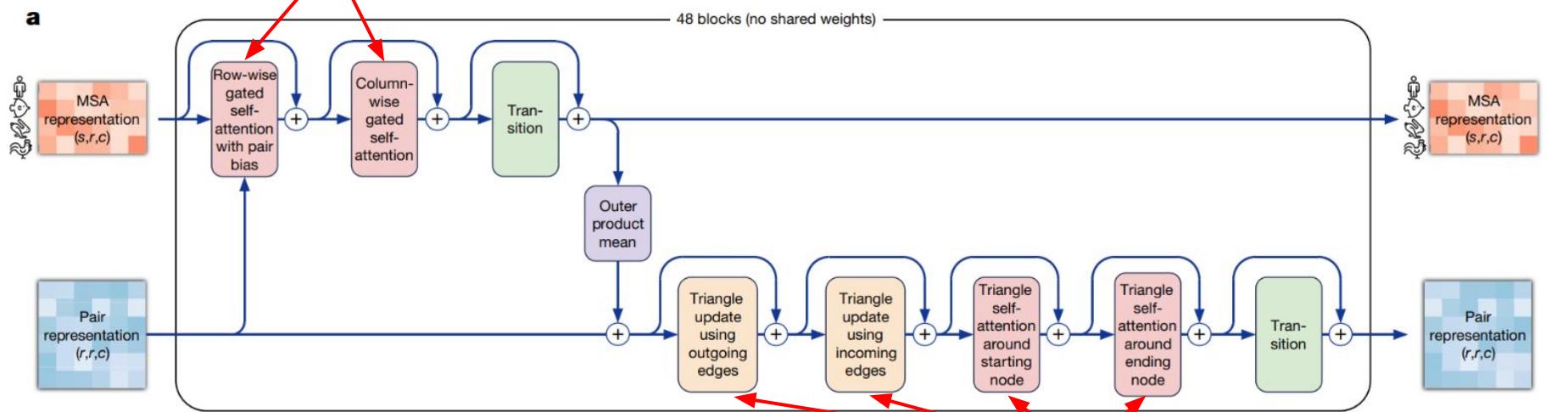
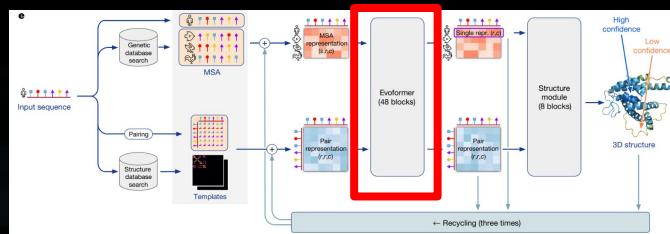
各种方法提取氨基酸碱基序列的特征

特征的融合和编码
(encoder)

预测结构的解码
(decoder)

5/8 AlphaFold, Evoformer

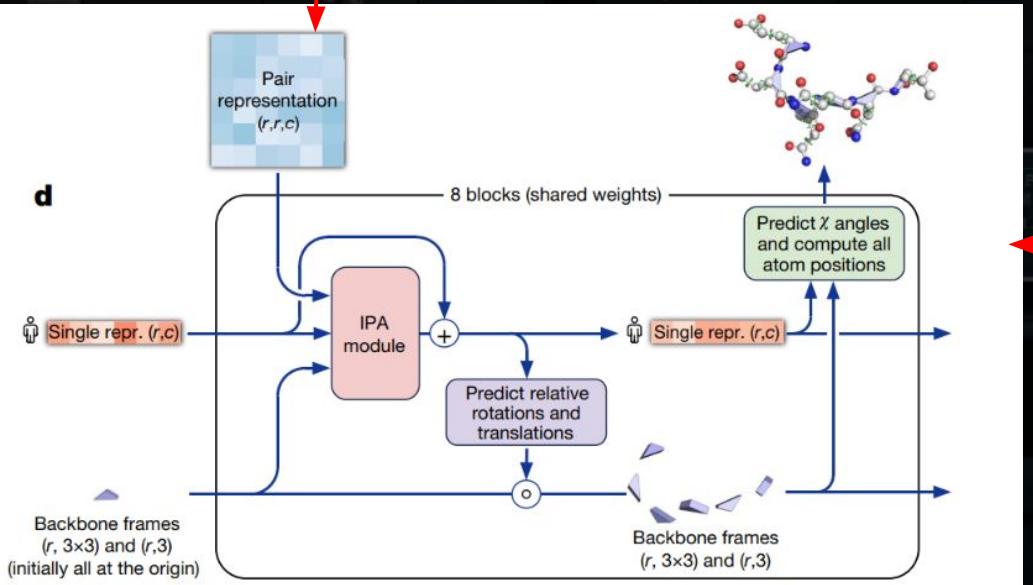
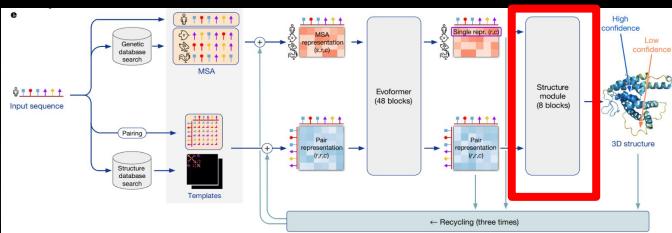
对于2维数据的基于行(row)和列(column)的注意力
(attention)计算



- Triangle Update 三角形的更新? 因为对于三维结构中紧靠的三点, 需要一个稳定的三角形(短的两边和比长边要长)

5/8 AlphaFold, Structure Module

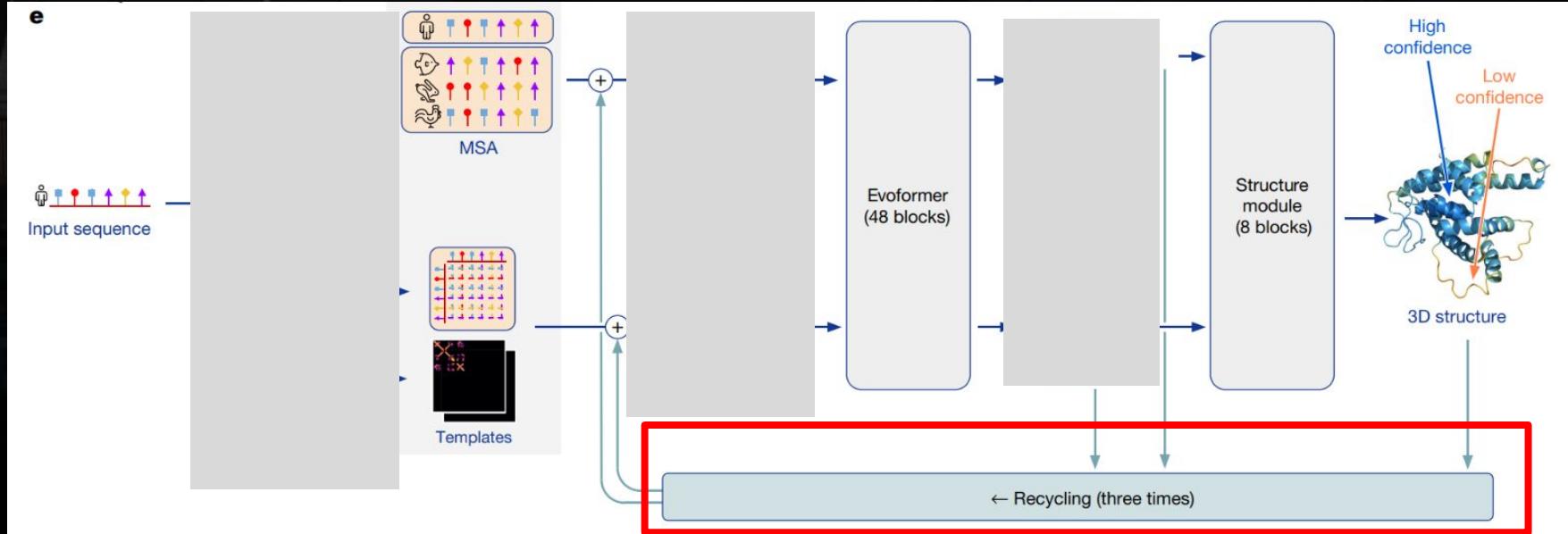
Evoformer 结构的输出，作为2D的
2x2比较(pair)的形式



有了新的3个数字表示的三维旋转
(rotation), 和另外三个数字表示的位置
变化(translation), 构建三位图像

空间3维结构的变化, 可以简化为3个数字表
示的三维旋转(rotation), 和另外三个数字
表示的位置变化(translation)

5/8 AlphaFold 2021, “recycling”



近似等价于将网络深度扩展3倍，并加入残差链接(residual connection)

6/8 AlphaCode 动机猜测

AlphaFold又放了一个大卫星，是时候考虑实际一点的工作内容了，写代码！

但是2020年GPT3还有比如CodeT5这样的工作又一些，怎么能显得与众不同？

答案：做难题！很难很难的编程竞赛题！



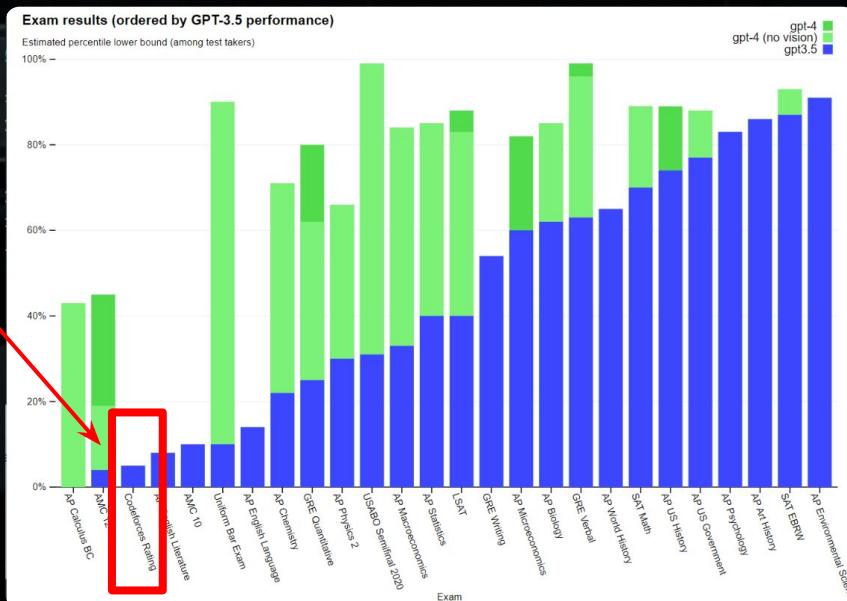
6/8 AlphaCode 目标:用AI写最难的竞赛题代码 !

我们不公平得比较一下

- 2022年3月份的AlphaCode
 - 各种技巧和更多时间下，15%-30%的解题律
 - 2023年3月份的GPT4
 - 没有AlphaCode这么多的技巧，能解~5%的题目

所以即使2023年了，GPT4 也
还没有解决AlphaCode之前在
研究的问题

Approach	Validation Set				Test Set		
	10@1k	10@10k	10@100k	10@1M	10@1k	10@10k	10@100k
9B	16.9%	22.6%	27.1%	30.1%	14.3%	21.5%	25.8%
41B	16.9%	23.9%	28.2%	31.8%	15.6%	23.2%	27.7%
41B + clustering	21.0%	26.2%	31.8%	34.2%	16.4%	25.4%	29.6%



6/8 AlphaCode 为什么算法竞赛题很难？

我们用一个例子([codeforce 58 + Google Translate](#))

- 问题的描述, 很长很抽象, 及其烧脑 !
- 时间限制, 所以对算法复杂度和技巧要求很高
- 代码的生成, 即使2023有了GPT4, codex, Bard等等, 还有很多挑战

B、算命

每次测试的时间限制: 2秒
每个测试的内存限制: 256兆字节
输入: 标准输入
输出: 标准输出

玛丽娜喜欢萨莎。但她一直想知道萨莎是否爱她。当然, 最好的了解方式就是算命。算命的方法有很多种, 但玛丽娜选择了最简单的一种。她手里拿着一朵或多朵洋甘菊, 然后一朵一朵地撕下花瓣。在每片花瓣之后, 她交替发音“爱”和“不爱”, 而玛丽娜总是以“爱”开头。田野里生长着 n 朵洋甘菊, 其花瓣数量等于 a_1, a_2, \dots, a_n 。玛丽娜想要挑选一束花瓣总数尽可能多的花束, 这样结果仍然是“爱”。帮助她做到这一点; 找出花束中可能出现的最大花瓣数。

输入

第一行包含一个整数 n ($1 \leq n \leq 100$), 它是田野中生长的花朵的数量。第二行包含 n 个整数 a_i ($1 \leq a_i \leq 100$), 表示给定第 i 个甘菊上的花瓣数量。

输出

打印一个数字, 即花束中花瓣的最大数量, 算命将导致“爱情”。如果没有这样的花束, 则打印0。花束可以由一朵花组成。

例子

输入

1

1

复制

输出

1

复制

输入

1

2

复制

输出

0

复制

输入

3

5

6

7

复制

输出

13

复制

6/8 AlphaCode 设计思路

- 基于 Encoder-decoder的架构
 - GPT是decoder-only
 - 为什么这里encoder需要？作者说因为问题本身的复杂度，需要深度分析(通过encoder的双向注意力机制)后，再和decoder结合
 - 另外时间回到2021年，encoder/decoder还是主流LLM架构
- 预训练(pretrain)和微调(finetune)
 - 比起chatgpt，少了RLHF
- 推理时候的额外技巧
 - 为什么有额外的技巧？因为解决算法题比回答ChatGPT类似的问题有更多的时间

6/8 AlphaCode 实现细节

预训练(pretrain)

- 在GitHub数据库上用类似填空的方式(具体参考[T5](#))训练

微调(finetune)

- 目标是解决codeforce上面的竞赛难度题目, 所以用老的codeforce问题微调, 把更新的codeforce文字作为测试集查看效果

推理技巧(也可以衍生理解为AI Agent的planning能力之一)

1. **更多采样(Sampling)**, 比如每个问题让语言模型回答100万次
2. **代码过滤(filtering)**, 我们可以用编译器或其他自动化方法筛选更优的答案
3. **自动聚类(Clustering)**, 基于“**正确的答案只有一个, 错误的答案五花八门**”的假设, 让所有代码对问题的示例或者相关的输入运行输出结果, 相同答案的就是同组, 从聚类后的每组按照群组大小进行尝试, 一般尝试10次

7/8 AlphaTensor 动机猜测

AlphaCode也有了进展，但是受众太小，要不来看看AI计算领域有哪些可以提升？

AI计算几乎都是大的矩阵乘法(matrix multiplication)，但现在的矩阵算法的硬件优化还是几十年前的东西(比如Strassen 算法)，有没有可能优化？

所以，尝试AI能不能帮助探索类似Strassen 算法这样的高效算法？答案是可以的，也就是AlphaTensor



7/8 AlphaTensor 背景

在2023年的计算机(非量子计算)硬件条件下

- 不管是CPU, GPU, TPU还是其他芯片, 数据的乘法(multiplication)要比加法(addition)昂贵, 而且贵非常多
- 所以, 对于矩阵乘法, 关注的这个矩阵乘法效率问题就可以
 - 从“寻找高效的硬件矩阵乘法算法”
 - 简化到“找到方法用加减法代替昂贵的乘法”
- 那到底什么是“**用加减法代替昂贵的乘法**”? 我们用一个简单的中学数学例子
 - 如果我们计算 $a^2 - b^2$
 - 我们需要 2 次乘法($a \times a$ 和 $b \times b$), 然后 1 次减法
 - 但是, $a^2 - b^2 = (a+b) \times (a-b)$ 如
 - 我们需要 2 个加减法($a+b$ 和 $a-b$), 然后 1 次乘法

所以, 我们需要把问题拓展到三维矩阵, 并借助 AI 找出类似 $(a+b) \times (a-b)$ 的算法

7/8 AlphaTensor 不得不用一点数学来解释

我们考虑最简单的两个 2×2 矩阵的乘法

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

$$c_1 = a_1 \times b_1 + a_2 \times b_3$$

$$c_2 = a_1 \times b_2 + a_2 \times b_4$$

$$c_3 = a_3 \times b_1 + a_4 \times b_3$$

$$c_4 = a_3 \times b_2 + a_4 \times b_4$$

- 8 个乘法, 4 个加法

7/8 AlphaTensor 我们投影这个问题到一个三维空间

我们考虑最简单的两个 2×2 矩阵的乘法

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

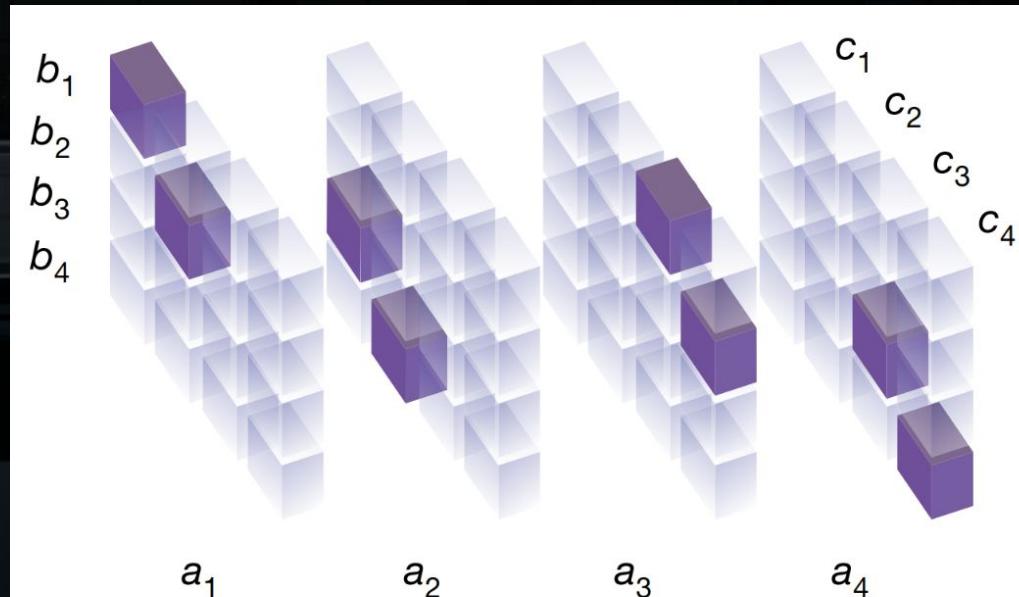
$$c_1 = a_1 \times b_1 + a_2 \times b_3$$

$$c_2 = a_1 \times b_2 + a_2 \times b_4$$

$$c_3 = a_3 \times b_1 + a_4 \times b_3$$

$$c_4 = a_3 \times b_2 + a_4 \times b_4$$

- 8 个乘法, 4 个加法



7/8 AlphaTensor 我们投影这个问题到一个三维空间

我们考虑最简单的两个 2×2 矩阵的乘法

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

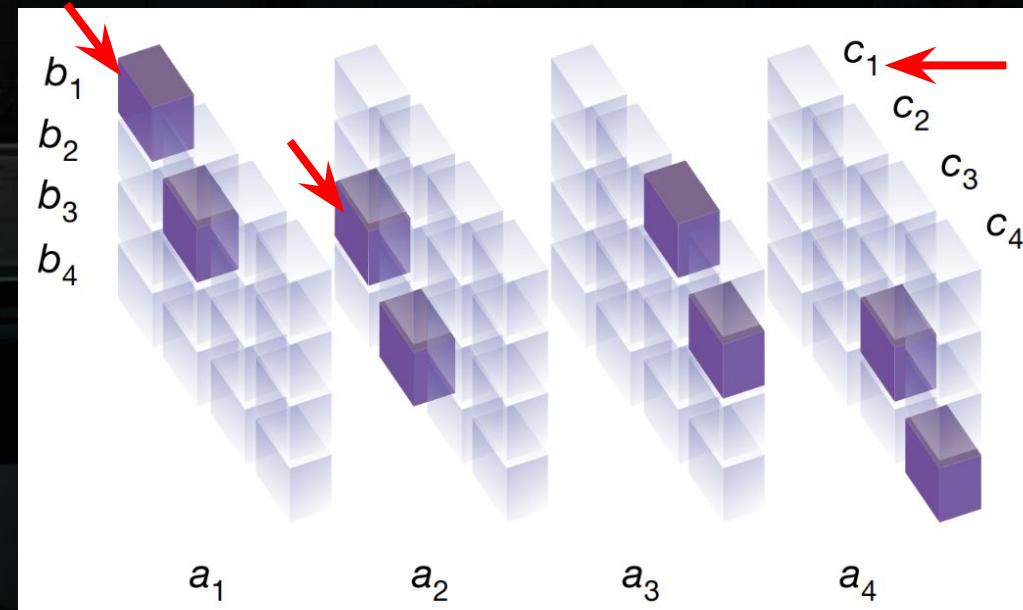
$$c_1 = a_1 \times b_1 + a_2 \times b_3$$

$$c_2 = a_1 \times b_2 + a_2 \times b_4$$

$$c_3 = a_3 \times b_1 + a_4 \times b_3$$

$$c_4 = a_3 \times b_2 + a_4 \times b_4$$

- 8 个乘法, 4 个加法



7/8 AlphaTensor 我们投影这个问题到一个三维空间

我们考虑最简单的两个 2×2 矩阵的乘法

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix} = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}$$

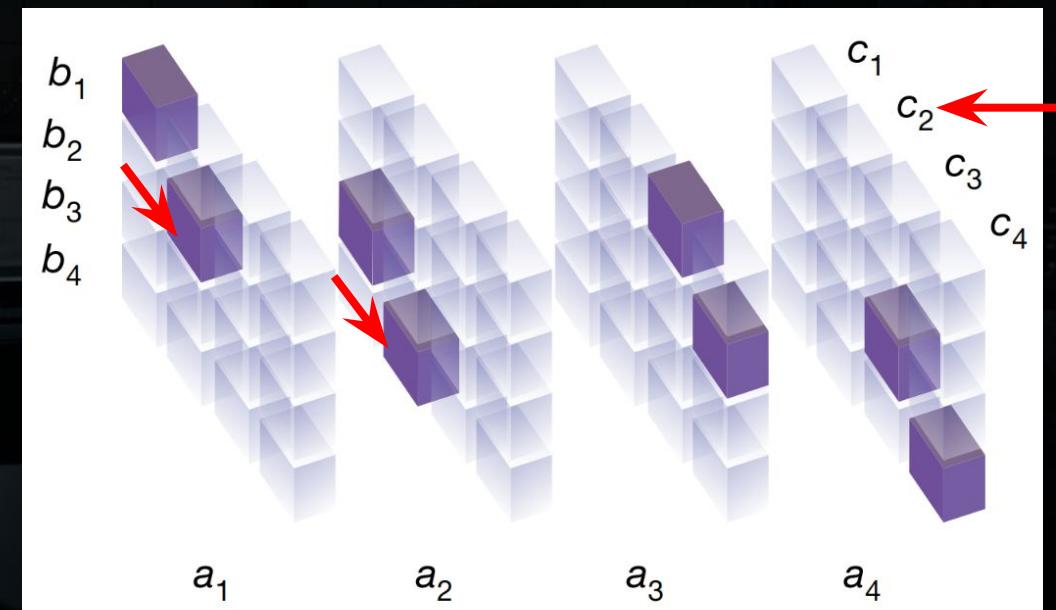
$$c_1 = a_1 \times b_1 + a_2 \times b_3$$

$$c_2 = a_1 \times b_2 + a_2 \times b_4$$

$$c_3 = a_3 \times b_1 + a_4 \times b_3$$

$$c_4 = a_3 \times b_2 + a_4 \times b_4$$

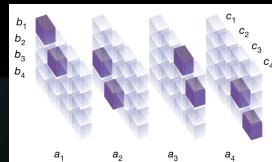
- 8 个乘法, 4 个加法



7/8 AlphaTensor 问题的具化

我们要寻找U V W三个相同维度的矩阵

- 按照每一列，三个矩阵的每一个位子用AI去推测更加可行的数值(限制-2到+2)
- 三个列的外积(outer product)就是一个 $4 \times 4 \times 4$ 的矩阵，和前面的3维投影尺寸一致！
- 然后就找出“足够少”的UVW列，他们的和与我们需要的三维投影重合！



$$\mathbf{U} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$
$$\mathbf{V} = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$
$$\mathbf{W} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$\mathcal{T}_n = \sum_{r=1}^R \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$$

7/8 AlphaTensor 具体实现

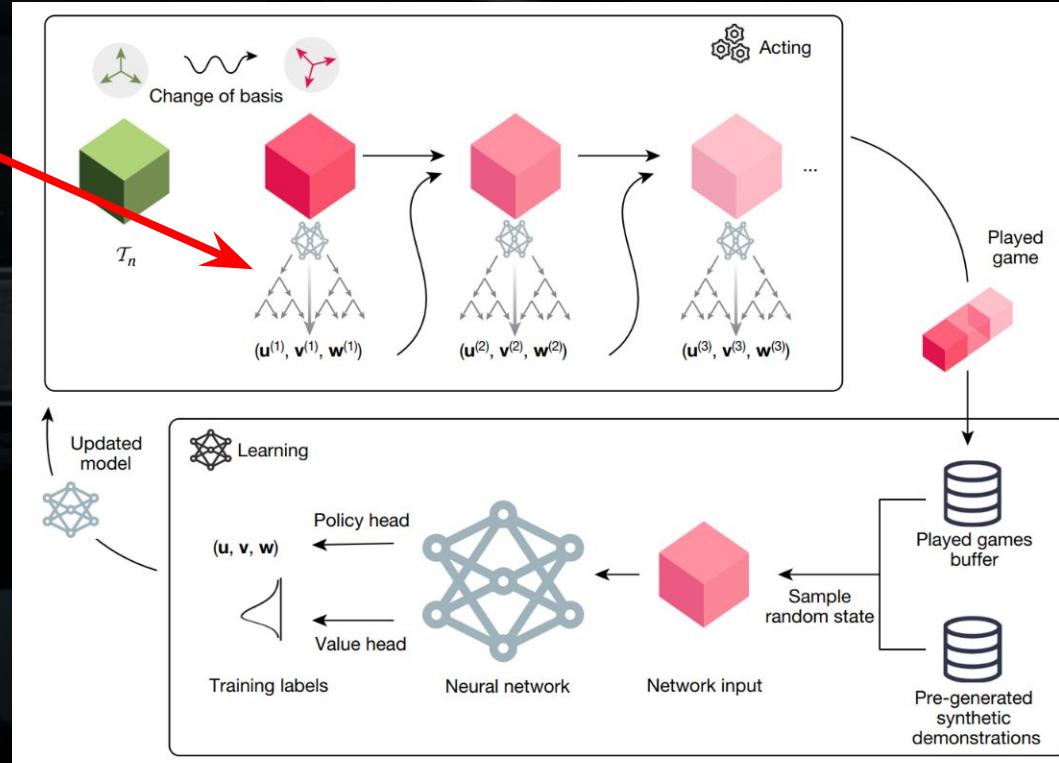
熟悉的AlphaZero又来啦！

同样包含

- Policy head, 看看每一个位置应该给出哪一个数值
- Value head, 评估整体游戏的状态

另外一些数据增强的技巧

强化学习用于自我练习和提高！



8/8 AlphaDev 动机猜想

用 AI 写代码可行，那能不能让 AI 深度优化代码？

用 AI 寻找算法，难道只有矩阵运算适用吗？

要不用AI去看看一些基本的C++ 类库，能不能提升？

- 答案：可以提升基本排序和哈希算法的效率！

Article

Faster sorting algorithms discovered using deep reinforcement learning

<https://doi.org/10.1038/s41586-023-06004-9>

Received: 25 July 2022

Accepted: 23 March 2023

Published online: 7 June 2023

Open access

 Check for updates

Daniel J. Mankowitz^{1,2,✉}, Andrea Micheli³, Anton Zhernov^{1,3}, Marco Gelmi^{1,3}, Marco Selvi^{1,3}, Cesarin Paduraru^{1,2}, Edouard Leurent^{1,2}, Sharis Isabell¹, Jean-Baptiste Lepsius¹, Alex Alemi¹, Thomas Kötter¹, Kevin Milligan¹, Stephen Giffney¹, Sophie Eister¹, Jackson Boshears¹, Chris Gamble¹, Kieran Milan¹, Robert Tung¹, Minjae Hwang¹, Taylan Cemgil¹, Mohammadamin Barekatian¹, Yujia Li¹, Anmol Mandhani¹, Thomas Hubert¹, Julian Schrittwieser¹, Demis Hassabis¹, Pushmeet Kohli¹, Martin Riedmiller¹, Oriol Vinyals¹ & David Silver¹

Fundamental algorithms such as sorting or hashing are used trillions of times on any given day¹. As demand for computation grows, it has become critical for these algorithms to be as performant as possible. Whereas remarkable progress has been achieved in the past², making further improvements on the efficiency of these routines has proved challenging for both human scientists and computational approaches. Here we show how artificial intelligence can go beyond the current state of the art by discovering hitherto unknown routines. To realize this, we formulated the task of finding a better sorting routine as a single-player game. We then trained a new deep reinforcement learning agent, AlphaDev, to play this game. AlphaDev discovered small sorting algorithms from scratch that outperformed previously known human benchmarks. These algorithms have been integrated into the LLVM standard C++ sort library³. This change to this part of the sort library represents the replacement of a component with an algorithm that has been automatically discovered using reinforcement learning. We also present results in extra domains, showcasing the generality of the approach.

Human intuition and know-how have been crucial in improving algorithms. However, many algorithms have reached a stage whereby human experts have not been able to optimize them further, leading to an ever-growing computational bottleneck. The work in classical program synthesis literature, spanning many decades, aims to generate correct programs and/or optimize programs using proxies for latency. These include enumerative search techniques^{4–6} and stochastic search^{7,8,9,10}, as well as the more recent trend of using deep learning in program synthesis for generating correct programs^{11–14}. Using deep reinforcement learning (DRL), we can take this a step further by generating correct and performant algorithms by optimizing for actual measured latency at the CPU instruction level, by more efficiently searching and considering the space of correct and fast programs compared to previous work.

One of the fundamental questions in computer science is how to sort a sequence^{10–12}. This is taught in elementary computer science classes around the world^{13,14} and is used ubiquitously by a vast range of applications^{15–17}. The field of computer science research has focused on developing and optimizing sorting algorithms^{18–20}. A key concern of practical solutions is a small sort over a short sequence of elements: this algorithm is called repeatedly when sorting large arrays that use divide-and-conquer approaches²¹. In this work, we focus on two types of small sort algorithm: (1) the fixed sort and (2) the variable sort. Fixed sort algorithms sort sequences of a fixed length (for example, sort 3

can only sort sequences of length 3), whereas variable sort algorithms can sort a sequence of varying size (for example, variable sort 5 can sort sequences ranging from one to five elements).

We formulate the problem of discovering new, efficient sorting algorithms as a single-player game that we refer to as AssemblyGame. In this game, the player selects a series of low-level CPU instructions, which we refer to as assembly instructions²², to combine to yield a new and efficient sorting algorithm. This is challenging as the player needs to consider the combinatorial space of assembly instructions to yield an algorithm that is both provably correct and fast. The hardness of the AssemblyGame arises not only from the size of the search space, which is similar to extremely challenging games such as chess (10^{120} games)²³ and Go (10^{170} games)²⁴, but also from the nature of the reward function. A single incorrect instruction in the AssemblyGame can potentially invalidate the entire algorithm, making exploration in this space of games incredibly challenging.

To play the game, we introduce AlphaDev, a learning agent that is trained to find correct and efficient algorithms. This agent is composed of two core components (simply) (1) a learning algorithm and (2) a representation function. The AlphaDev learning algorithm can incorporate both DRL as well as stochastic search optimization algorithms to play AssemblyGame. The primary learning algorithm in AlphaDev is an extension of AlphaZero²⁵, a well-known DRL algorithm, in which a neural network is trained to guide a search to solve

8/8 AlphaDev 具体的问题

“AssemblyGame”用来描述这个问题

- 什么是Assembly(汇编)？基于高级代码(比如C++)和机器代码的中间表示
- 每一行汇编码都是一个CPU指令，比高级语言更加严格，但比机器代码更加可读
- AssemblyGame 游戏的目的？
 - 用更少的CPU指令
 - 来实现同样的汇编代码功能

a

```
void variable_sort_2(int length, int *a)
{
    // Determine the number of elements
    switch (length)
    {
        case 0:
        case 1:
            // Exit if less than 2 elements
            return;
        case 2:
            // Below routine sorts 2 elements
            int tmp = a[0];
            a[0] = (a[1] < a[0]) ? a[1] : a[0];
            a[1] = (a[1] < tmp) ? tmp : a[1];
            return;
    }
}
```

b

```
variable_sort_2(int, int*):
; Determine the number of elements
    cmp    $2, %edi
; Exit if less than 2 elements
    jne    .Label
; Below routine sorts 2 elements
    mov    (%rsi), %eax
    mov    4(%rsi), %ecx
    cmp    %eax, %ecx
    mov    %eax, %edx
    cmovl %ecx, %edx
    mov    %edx, (%rsi)
    cmovg %ecx, %eax
    mov    %eax, 4(%rsi)

.Label:
    retq
```

8/8 AlphaDev 具体的问题

“AssemblyGame”用来描述这个问题

- 什么是Assembly(汇编)？基于高级代码(比如C++)和机器代码的中间表示
- 每一行汇编码都是一个CPU指令，比高级语言更加严格，但比机器代码更加可读
- AssemblyGame 游戏的目的？
 - 用更少的CPU指令
 - 来实现同样的汇编代码功能

a

```
void variable_sort_2(int length, int *a)
{
    // Determine the number of elements
    switch (length)
    {
        case 0:
        case 1:
            // Exit if less than 2 elements
            return;
        case 2:
            // Below routine sorts 2 elements
            int tmp = a[0];
            a[0] = (a[1] < a[0]) ? a[1] : a[0];
            a[1] = (a[1] < tmp) ? tmp : a[1];
            return;
    }
}
```

b

```
variable sort_2(int, int*):
; Determine the number of elements
    cmp    $2, %edi
; Exit if less than 2 elements
    jne    .Label
; Below routine sorts 2 elements
    mov    (%rsi), %eax
    mov    4(%rsi), %ecx
    cmp    %eax, %ecx
    mov    %eax, %edx
    cmovl %ecx, %edx
    mov    %edx, (%rsi)
    cmovg %ecx, %eax
    mov    %eax, 4(%rsi)

.Label:
    retq
```

可否在功能一样的前提下，减少这里类似cmp jne mov这样的CPU指令数目？

8/8 AlphaDev 实现

熟悉的配方, AlphaZero+MCTS

Policy, 每一个代码位置指导使用
哪一个CPU指令

Value, 评估当前“游戏”的状态

自我博弈(练习)后的代码能被验证
(功能是否一直)和比较(具体步数)

自我学习的经验交给AlphaDev去
学习提高

We refer to the agent that plays this single-player game as AlphaDev. The agent's primary learning algorithm is an extension of the AlphaZero agent³² and guides a Monte Carlo tree search (MCTS) planning procedure using a deep neural network^{33,34}. The input to the neural network is the state S_t and the output is a policy and value prediction. The policy prediction is a distribution over actions and the value function is a prediction of the cumulative returns R that the agent should expect to receive from the current state S_t . During a game, the agent receives as input the current state S_t . The agent then executes an MCTS procedure and uses this to select the next action to take. The generated games are then used to update the network's parameters, enabling the agent to learn.

参考

8篇论文

- [AlphaGo](#) (nature)
- [AlphaGo Zero](#) (nature)
- [AlphaZero](#) (science)
- [AlphaStar](#) (science)
- [AlphaFold2](#) (nature)
- [AlphaCode](#) (arxiv)
- [AlphaTensor](#) (nature)
- [AlphaDev](#) (nature)

相关讨论

- [Jonathan Hui: AlphaGo: How it works technically?](#)
- [Yannic Kilcher - YouTube](#)
- [Mu Li - YouTube](#)
- [Shusen Wang - YouTube](#)
- [Two Minute Papers - YouTube](#)
- [Hung-yi Lee - YouTube](#)

讨论时间！

浅谈 DeepMind 的 8 篇“Alpha”冠名论文



少剑

hululu.zhu@gmail.com

07/09/2023