引入威胁空间搜索的五子棋深度强化学习方法

牛学芬[1],王子游[1],陈　灵[1],吴育华[1],刘雨泽[1],徐长明[1,2]

(1.东北大学秦皇岛分校 计算机与通信工程学院,河北 秦皇岛 066004;

2.东北大学 研究生院,河北 秦皇岛 066004)

摘要:结合蒙特卡罗树搜索与深度神经网络的深度强化学习方法,已经成为解决复杂博弈问题的基准方法,但仍面临奖励稀疏以及训练成本高等问题。为此,提出引入威胁空间搜索的五子棋深度强化学习方法:首先,设计了嵌入到蒙特卡罗树搜索的统一威胁空间搜索算法,缓解了奖励稀疏的问题;其次,提出了基于领域知识的双层知识库,加快算法搜索速度;此外,将威胁动作空间作为神经网络的输入特征,增强了模型对关键局部形势的感知能力;最后,利用走法过滤机制有效地缩小了动作空间。实验结果表明,上述改进措施显著地提升了自博弈程序的学习速度和竞技水平。

关键词:蒙特卡罗树搜索;深度神经网络;威胁空间搜索;自博弈

中图分类号:TP18　　文献标识码:A

Deep Reinforcement Learning for Gomoku with Threat-space Search

NIU Xuefen[1],WANG Ziyou[1],CHEN Ling[1],WU Yuhua[1],LIU Yuze[1], XU Changming[1,2]

(1. School of Computer and Communication Engineering, Northeastern University, Qinhuangdao 066004, China;

2. Graduate, Northeastern University, Qinhuangdao 066004, China)

Abstract: Deep Reinforcement Learning methods integrating Monte Carlo Tree Search (MCTS) and Deep Neural Networks have become the benchmark approach for solving complex game problems. However, challenges such as sparse rewards and high training costs remain prominent. To address these issues, this paper proposes an improved deep reinforcement learning method for Gomoku by incorporating Threat Space Search (TSS). The main contributions include: designing a Unified Threat Space Search algorithm that seamlessly integrates with MCTS to mitigate the sparse reward problem; proposing a domain knowledge-based dual-layer knowledge base structure to enhance search efficiency; incorporating threat-based offensive and defensive sets as neural network input features to improve the model's perception of critical local game situations; and developing a move filtering mechanism based on threat space features to effectively reduce the action space. Experimental results demonstrate that these improvements significantly enhance both the learning efficiency and competitive performance of the self-play program.

Key words: Monte Carlo Tree Search; Deep Neural Networks; Threat Space Search; Self-play

0 引言

基本的蒙特卡罗树搜索[1](Monte Carlo Tree Search，MCTS)是一种平衡了探索和利用的决策算法,能够在估值函数水平不高的情况下,通过统计模拟来评估各候选动作的相对优劣。MCTS 通过与深度神经网络(Deep Neural Networks, DNN)结合,如 AlphaGo Zero[2]、AlphaZero[3]和 MuZero[4]等方法成为解决复杂博弈问题的首选方法,后又被陆续应用到更多的棋牌博弈问题当中[5-10]。然而,该方法始终面临奖励稀疏以及训练成本高等问题。Tian 等人[11]也指出, DNN 在训练过程中表现出高方差、复杂动作的学习缓慢,模型在某些局面下预测不可靠等。上述问题制约了 MCTS-DNN 范式的应用效果。

在五子棋、象棋等棋类中,借助特殊的技术使得回报稠密化将有助于提高学习效率,如证明“五子棋先手必胜”的威胁空间搜索(Threat Space Search, TSS)[12]。TSS 的核心思想是不直接搜索整个博弈树,而是通过尝试走出威胁走法,逼迫对手在狭小的动作空间内被动选择不利走法,从而以一棵稀疏子树为微小代价尝试快速证明当前走棋方必胜。但是,一旦证明失败, TSS 就无法为当前局面提供任何落子选择,需要借助其它搜索算法完成后续的决策,如

收稿日期:2024-10-30

基金项目:河北省自然科学基金项目面上项目(F2022501015)。

作者简介:牛学芬,女,讲师,主要从事 FPGA 开发,机器学习研究,E-mail:niuxuefen@neuq.edu.cn;通信作者 徐长明,男,工学博士,讲师,主要从事基于深度学习的机器博弈、时间序列异常检测等研究,E-mail:changmingxu@neuq.edu.cn

Alpha-Beta 搜索、MCTS 等。此外, TSS 的一次调用仅能回答“是否可以通过威胁走法获胜”的简单二元问题,不能给出是否输棋的结论。为此,以带禁手五子棋(Renju)为研究对象,提出一种与 MCTS 方法相结合的改进 TSS,并提供提高五子棋深度强化学习训练效率和竞技水平的整体方案。

无禁手五子棋的基本规则为:在 15×15 的棋盘上,黑方先手落一子,此后黑白双方轮流在空交叉点放置一枚棋子,率先在任意方向形成连续五个同色棋子者获胜[13]。Renju 在此基础上对黑方施加了特殊走棋限制,包括长连禁手、三三禁手和四四禁手,主要是为了平衡先手优势。Allis 指出[14]Renju 的状态空间复杂度约为 10^{102},博弈树复杂度约为 10^{140},是 PSPACE-完全问题[15]。

在本文中,主要工作如下: 1)为了将 TSS 融入 MCTS 中,设计了统一威胁空间搜索算法(Unified Threat Space Search, UTSS); 2)为了增强神经网络提取棋局中关键特征的能力,将威胁动作空间作为神经网络的输入特征; 3)为了进一步限制动作空间,设计了走法过滤机制; 4)为了加速棋形识别和更新速度,提出了预生成的基于“线段-棋形”双层结构的知识库。实验结果表明,所提出的方法显著地提升了模型的学习速度和程序的竞技水平,也降低了算力需求。

1 形式化定义和基础记号

棋形是表征五子棋领域知识的最小单位,依据棋子间配合和牵制作用,将全部棋形划分为 12 种类型,可近似认为类型相同的不同棋形彼此等价。

定义 1 玩家 p 的棋形 s :一条线上,长度不少于 k=5 且仅由 p 的棋子及空交叉点形成的最长序列。

定义 2 棋形类型映射 $T: S\to Z_1$={0, 1, ..., 11},其中, S 为全体棋形的集合, Z_1 为全体类型的集合。

下面以**形状**为 $s.s=(11010011)_2$ 的棋形 s 为例,给出一系列记号及涵义。从右向左 $s.s$ 各个**位置** i 的**状态**记为 $s.s[i]\in\{0,1\}$,值 1 代表棋子,0 代表空交叉点;**长度**记为 $s.len$=8;**数值表示** $s.v=\sum_{0\le i<s.len} 2^i\cdot s.s[i]$=211 描述了 s 中棋子分布情况;**编码表示** $s.id=2^{s.len}-2^k+s.v$, 可用作 s 在棋形知识库中的索引地址;**类型** $s.t$=7 表明 s 是活三(见表 2)类型;**所属玩家** $s.p\in${“B”, “W”},**对手** $s.o\in${“B”, “W”}; $s.E$ 表示 s 的空交叉点集;若 $s.p$=“B”,则 $s.o$=“W”,落黑子后新形成的$(1101\underline{1}011)_2$存在双冲四“1101**1**---”和“---1**1**011”,故位置 i=3 是四四**禁手**;白方可在任意空交叉点处落子,不存在白方禁手。$s'=(0011010011)_2$ 和 s 是不同棋形,但数值表示相同,即 $s.v=s'.v=(211)_{10}$;而编码表示则可以区分开 s' 和 s, $s.id=(435)_{10}\neq s'.id=(1203)_{10}$。表 1 列出了总计 131,008 个棋形的 12 种类型及其分布。

表 1 二进制棋形库中各棋形的数量和百分比

棋形	数量	百分比(%)	棋形	数量	百分比(%)
长连	10192	7.78	活二	8556	6.53
连五	11856	9.05	眠二	2357	1.80
活四	20094	15.34	活一	770	0.59
冲四	30635	23.38	眠一	147	0.11
活三	27227	20.78	禁手	710	0.54
眠三	18442	14.08	空	22	0.02

定义 3 威胁等级 $L_b: T\to Z_2$={-1, 0, 1, 2, 3, 4}和 **$L_w: T\to Z_3$**={0, 1, 2, 3, 4},定义了黑(或白)方棋形 s 中对方受到威胁的等级 $L_b(t)\in Z_2$ (或 $L_w(t)\in Z_3$), $t=T(s)\in Z_1$。表 2 列出了类型与威胁等级的对应关系;因不允许黑方走出长连和禁手,故 $L_b(11)$=-1 和 $L_b(1)$=-1。

表 2 在黑(白)方的棋形 s 中对手受到的威胁等级

	长连	连五	活四	冲四	活三	眠三	活二	眠二	活一	眠一	禁手	空
$t=T(s)$	11	10	9	8	7	6	5	4	3	2	1	0
$L_b(t)$	-1	4	3	2	1	0	0	0	0	0	-1	0
$L_w(t)$	4	4	3	2	1	0	0	0	0	0	0	0

定义 4 落子操作 $\oplus_p: S\times I\to S$,表示玩家 p 在玩家 $s.p$ 的棋形 s 中位置 $i\in I_s=\{i \mid s.s[i]=0\}$处落子,形成新棋形 s'。具体地,若 $p=s.p$,则 $s'.v\leftarrow s.v+2^i$;否则 $p=s.o$, 且 $s'.v\leftarrow \text{argmax}_w\ \{w.t \mid w\in\{s.v\&(2^i-1),\ s.v>>i\}\}$,其中 &表示按位与操作, $>>i$ 表示算数右移 i 位的位操作。得到 $s'.v$ 后,同步更新 s'中的其余属性,并返回 s'。

2 基于“线段-棋形”的双层领域知识库

为了加速棋形识别和更新,进而加速 TSS ,提出并实现了一种基于“线段-棋形”双层结构的领域知识库,包括:易于领域知识复用的二进制棋形知识库、方便检索的三进制线段知识库。

2.1 构建“线段-棋形”双层知识库

每条线段由三进制数表示(白棋为 0,空交叉点为 1,黑棋为 2),用于描述棋子分布、选择走法等。三进制线段可进一步分解为若干二进制棋形。

三进制线段知识库存储了 21,523,239 个有效线段。知识库预先为每个线段分配了一个结构,主要记录各个二进制棋形在三进制线段中的起始位置、对应的二进制棋形知识库的入口地址等信息。二进制棋形知识库存储了 131,008 个棋形的信息,包括棋形的类型、内部各空交叉点的类型、可能的禁手点等。

2.2 从“线段-棋形”双层知识库中检索

将 15×15 棋盘视为 72 条线段,包括 30 条水平和垂直线段以及 42 条对角线,每条线段长度为 5 至 15 不等。棋形识别采用两级查询机制:先计算对应的三进制编码从而定位其在线段知识库中的存储位置,读出所包含的各棋形在二进制棋形库中的地址;之后,跳转到二进制棋形库,提取棋形的类型等信息。通过预计算,将获取棋形类型等知识的时间复杂度降至 O(1)。此外,设计了局面状态的增量更新机制:每次落子最多更新四条相关线段,对受影响的线段重新计算编码并更新棋形信息,将更新的时间复杂度优化至 O(1)。

3 引入 TSS 的 MCTS 算法

3.1 TSS 搜索算法

每次调用基本 TSS 搜索都是在试图回答一个简单二元问题: “树根结点 c 的走棋一方 r_p,是否可以通过始终走威胁走法而获胜?”其优点在于: 1)需要考察的动作空间非常小; 2)有很大概率证明走棋方如何获胜; 3)可通过置换表等技术大幅提速。TSS 流程见算法 1,其中涉及到防守动作空间 $D(c)$和进攻动作空间 $A(c)$,由算法 2 和 3 描述。

算法 1: TSS(c, r_p)

输入: 棋局 $c.S$={棋形 s}, MAX 方记为 r_p。

输出: $v \in$ {True / False}。

```
01  if end_game(c, r_p):   return evaluate(c, r_p)
02  if r_p = c.p:   // 进攻方
03      if not A(c):   return False
04      for m in A(c):
05          t ← TSS(make_move(c, m), r_p)
06          if t =True:   return True
07      return False
08  else:           // 防守方
09      if not D(c):   return False
10      for m in D(c):
11          t ← TSS(make_move(c, m), r_p)
12          if t = False:   return False
13      return True
```

算法 2 的任务是筛选出所有可防守的动作。伪代码的第 02 行描述了从局面 c 中获取对手给我方 $c.p$ 造成威胁的棋形集合 *opp_patterns* ;第 03 行获取对手给 MAX 方造成威胁的最大等级 *opp_level* ;第 05 行~08 行,逐个考察对手给我方造成威胁的棋形,提取每一个可以消除威胁的走法,构造防守动作空间;第 10 行,如果我方是黑方,还要从动作空间中删去所有的禁手。

算法 3 对进攻动作空间进行了排序。优先考虑纯粹进攻的动作,如不能获胜;再考虑兼顾防守的进攻动作;如果是黑方走棋,仍要去掉禁手。

TSS 存在两个主要局限:一是当无法证明存在连续威胁获胜时,无法评估候选走法间的优劣;二是其评估结果过于简单,仅能判定是否存在必胜序列。这种粗粒度的评估特性导致 TSS 在 MCTS 中的应用效果受限。

算法 2: 获取棋局 c 下的防守动作空间 $D(c)$

输入: 棋局 $c.S$={棋形 s},轮到 $c.p$ 走棋,禁手集合 $c.H$。

输出: 防守动作空间 *def_moves*。

```
01  def_moves ← [ ]
02  opp_patterns ← [s for s in c.S if s.o=c.p and L_{s,p}(s.t)>0]
03  assert opp_patterns   // 排除对手没威胁到我方的情况
04  opp_level ← max([L_{s,p}(s.t) for s in opp_patterns])
05  assert opp_level < 4   //排除对手已获胜情况(见表 2)
06  for s in opp_patterns:
07      for m in s.E:
08          cond ← L(T(⊕_{c.p}(s, m))) <L(s.t)   // 可消除威胁
09          if cond:   def_moves.append(m)
10  if c.p = “B”:   def_moves ← def_moves - c.H
11  return def_moves
```

算法 3: 获取棋局 c 下的进攻动作空间 $A(c)$

输入: 棋局 $c.S$={棋形 s},轮 $c.p$ 走棋,黑方“B”,禁手集合 $c.H$。

输出: 进攻动作空间 *atk_moves*。

```
01  atk_moves ← [ ]
02  opp_level ← max([L_{s,p}(s.t) for s in c.S if s.o = c.p])
03  my_patterns ← [s for s in c.S
04      if s.p = c.p and s.t ≥ opp_level]     // 仅考虑进攻
05  for s in my_patterns:
06      for m in s.E:
07          cond ← L(T(⊕_{s,p}(s, m))) > opp_level
08          if cond:   atk_moves.append(m)
09  if opp_level > 0:
10      my_patterns ← [s for s in c.S
11          if s.p = c.p and 0 < s.t < opp_level] // 兼顾防守
12      for s in my_patterns:
13          for m in s.E ∩ D(c):
14              cond ← L(T(⊕_{s,p}(s, m))) > 0
15              if cond:   atk_moves.append(m)
16  if c.p = “B”:   atk_moves ← atk_moves - c.H
17  return atk_moves
```

3.2 统一威胁空间搜索算法

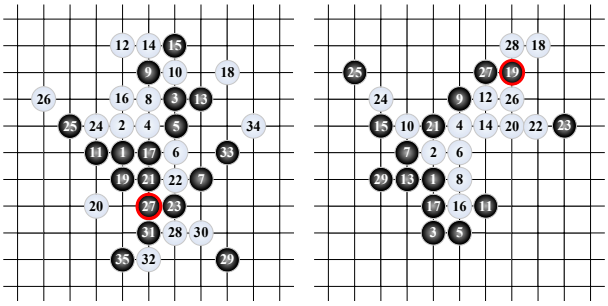
在 TSS 基础上,提出统一威胁空间搜索算法(Unified Threat Space Search, UTSS): 1)值域从{True, False}改为{Win/ Draw/ Lose}; 2)不再区分进攻方和防守方,为此采用由防守动作空间和进攻动作空间组合成的统一威胁动作空间 $M(c)$,令 $M(c) = A(c) \cup D(c)$。 UTSS 算法的基本框架如算法 4 所示。在实现 UTSS 时: 1)使用了迭代加深策略[16],从浅层逐步深入搜索最浅的解; 2)引入了置换表技术[17]缓存已搜索状态,避免状态被重复搜索。

算法 4: 统一威胁空间搜索 UTSS(*c*, *max_d*)

输入: 棋局 *c*,轮 *c.p* 走棋,最大迭代深度 *max_d*。

输出: (*done*, *val*) ∈ {True, false} × {UNCERTAIN = -2, LOSE = -1, DRAW = 0, WIN = +1}。

```
def UTSS(c, max_d):        // 迭代加深搜索
    (val, α, β) ← (UNCERTAIN, LOSE, WIN)
    for d in range(1, max_d+1):
        (done, v) ← DF_UTSS(c, d, α, β)//访问子结点
        if done: return v
    return UNCERTAIN       // 最大迭代深度不够
def DF_UTSS(c, d, α, β):
    M(c) ← A(c) ∪ D(c)     // 统一威胁动作空间
    if not M(c):           // 考虑禁手
        return (True, LOSE) if c.H else (True, DRAW)
    if end_game(c) or d < 1: return evaluate(c)
    (done, val) ← (True, UNCERTAIN)      // Fail-Soft
    for m in M(c):   // Nega-Max Alpa-Beta Search
        c'←make_move(c, m)
        (f, v) ← DF_UTSS(c', d-1, -β, -α)
        (done, val) ← ((done and f), max(val, -v))
        α ← max(val, α)
        if val ≥ β:   return (True, val)
    return (done, val)
def evaluate(c): //以 MAX 方视角评估叶结点
    if   c.p wins:   return (True, WIN)     //c.p 胜
    elif c.p loses:  return (True, LOSE) //c.p 负
    elif not M(c):   return (True, DRAW)   //不分胜负
    return (False, UNCERTAIN) //需加大最大搜索深度
```



(*a*)黑第 27 手已胜　　(*b*)黑第 19 手已胜

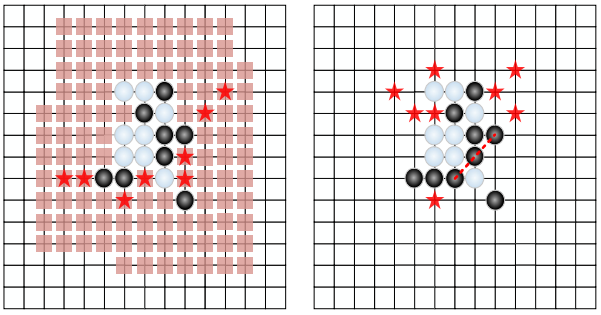
图 1　决赛对局实例

图 1 给出了在实际竞赛中出现的两个局面,红圈标记的是 UTSS 找到的必胜走法。

3.3 UTSS 结合 MCTS

UTSS 限制了动作空间大小,能以较大的概率发现局面的胜负。与 UTSS 不同,在深度强化学习中广泛采用的 MCTS ,依靠上限置信区间算法[18]更强调平衡探索与利用之间的关系。由于探索性的存在,MCTS 中存在相当数量的结点,其胜负能通过 UTSS 以较小代价证明。

为此,提出将 UTSS 与 MCTS 结合的方法: 1)在 MCTS 中对距离树根深度不超过 4 层的结点调用 UTSS; 2)将 UTSS 搜索中胜负结果确定的结点全部存入置换表,部分加入经验回放池用于训练神经网络,以缓解回报稀疏; 3)仅考虑棋局中棋子周围三格范围内的动作,约束 MCTS 候选动作空间,如图 2(a)中浅红色阴影区域; 4)当存在威胁时,MCTS 仅展开统一威胁动作空间,如图 2(b)中存在活三威胁(红色虚线)时的动作(五角星)。



(a) 黑方走棋　　(b) 白方走棋

图 2　走法过滤

3.4 神经网络及其训练

在神经网络结构设计中,采用类似 AlphaZero[3] 的深度学习框架,策略网络和价值网络分别用于评估棋局中的策略和当前状态的价值,如图 3。采用了残差卷积网作为骨干网,以保证梯度的有效传播。

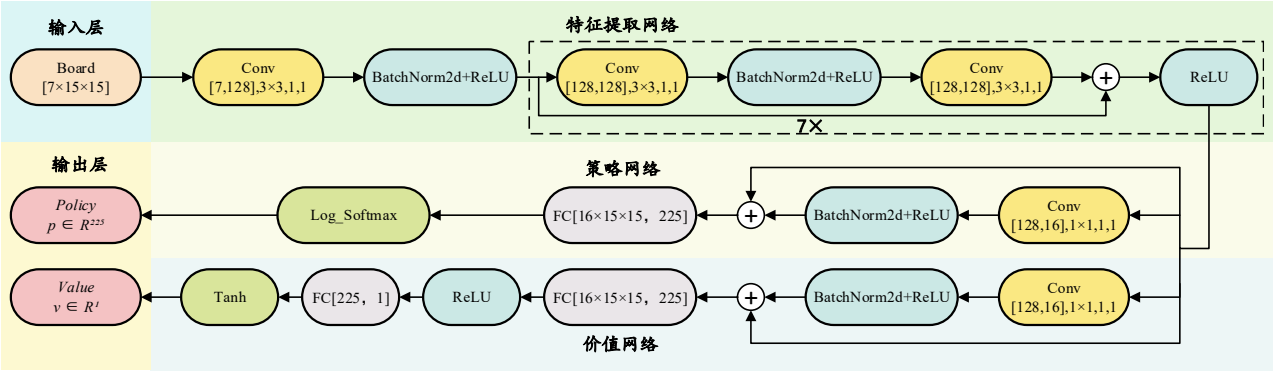


图 3　神经网络架构

为了有效地捕捉局面的关键局部威胁信息,神经网络的输入采用 7 种特征,如表 3。

表 3　神经网络的输入特征

输入通道	解释
1	我方棋子分布
2	对手棋子分布
3	走棋方的通道
4	已有棋子的周围三格范围
5	进攻动作空间
6	防守动作空间
7	可形成潜力棋形的空交叉点*

落子后可形成的活二及更好棋形,称为潜力棋形*。

训练模型的主要做法: 1)数据收集:自博弈产生数据,记录每局自博弈的胜负结果 z 以及 MCTS 在搜索过程中的策略输出 π ; 2)数据增强:利用对称性,对每个原始棋局样本进行四次 90°旋转,并对上述状态水平翻转,最多增强 8 倍数据; 3)经验回放:增强后的数据被存入双端队列缓冲区。当缓冲区被填满时,新数据会替换最旧的数据; 4)即时训练:每局结束后,当缓冲区中的数据量达到批次大小 512 ,立即训练和更新模型; 5)分层采样:将经验回放缓冲区中的数据按生成时间划分为近期数据和早期数据,每次训练从最新一轮生成数据随机抽 50% ,从早期数据抽取其余 50% ; 6)自适应学习率:在每个训练周期开始时记录当前策略的输出概率分布。在每轮训练后,计算新旧策略间的 KL 散度。基于 KL 散度值,动态调整学习率,设置 KL 散度的上限阈值,超过出将终止当前训练周期,防止策略剧烈变化。神经网络训练的超参数设置如表 4 所示。

表 4　神经网络训练超参数设置

超参数	含义	数值
replay buffer size	经验回放池	10000
num playout	MCTS 模拟次数	200
Lr	学习率	$2e^{-3}$
KL targ	*KL* 散度目标值	0.02
Epochs	每批次训练轮次	5
batch size	训练批次大小	512

4 实验结果与分析

实验环境为: Intel Core i7-12700H CPU、NVIDIA GeForce RTX3060 Laptop GPU (6GB VRAM)和 16GB 内存, Windows11 系统, PyTorch 深度学习框架。设计了包含四种方案的消融实验:目标方案 Target(MCTS+UTSS+MF+TC), 移除 UTSS (T-UTSS),移除威胁通道(T-TC)和移除走法过滤模块(T-MF)。随机初始化后,分别进行 500 局自博弈。

4.1 损失收敛分析

图 4 展示了四种方案的损失曲线。从总体趋势上: Target 方案损失函数下降最快, 400 盘以后损失降至 1.7 左右; T-TC 的损失在 450 盘以后和 Target 方案最接近; T-UTSS 和 T-MF 的损失函数在 400 盘以后数值接近,在 2.5~2.8 区间波动。以上实验结果表明 UTSS 和 MF 在自学习中起到了关键的促进作用。在整个训练过程中:移除 UTSS、走法过滤、威胁通道分别导致平均损失函数值相比目标方案增加了约 54.7%、43.4%和 13.8% ,再次说明 UTSS、走法过滤这两项改进的作用显著。

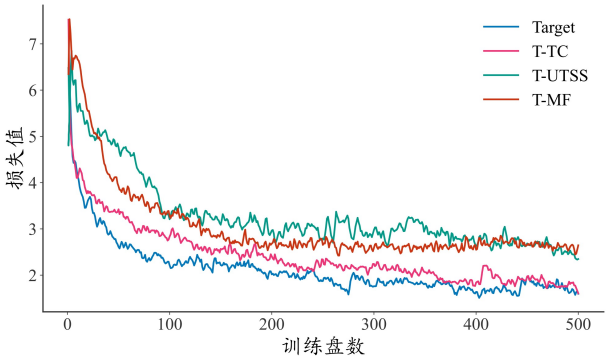


图 4　神经网络损失曲线

4.2 分支因子和对局步数分析

为了进一步理解三种改进的作用,下面将从平均分支因子、单局对弈长度两个角度进行对比实验。采用四种方案分别进行 500 盘自学习对弈,记录每盘棋的平均分支因子和对局长度,在此基础上,围绕各自方案的均值和标准差拟合出结果分布范围,结果见图 5。其中,纵轴“分支因子”是一局中每个决策结点可选动作个数的平均数,量化了动作空间大小;横轴“单局对局步数”体现了对抗水平。

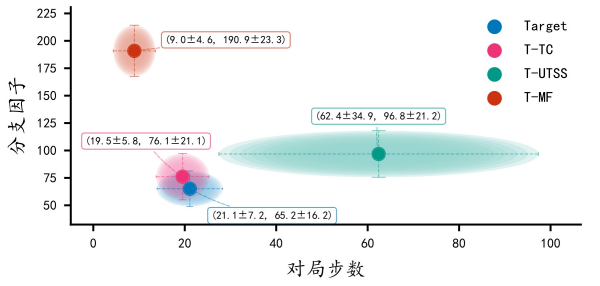


图 5　分支因子与对局步数分布对比

仅移除 UTSS(T-UTSS): 1)该方案具有最长的平均对局步长,是目标方案3倍,表明移除UTSS导致自博弈过程未能适时停止,也错失了高质量监督信号; 2)平均对局步长的标准差最大,高达 34.9 ,接近目标方案的 5 倍,说明采集到的数据包含较多不确定性,不利于神经网络学习; 3) MCTS 平均分支因子约为目标方案 1.5 倍,说明动作空间较大。

仅移除走法过滤(T-MF): 1)该方案具有最大的 MCTS 平均分支因子,约为目标方案的 3 倍,且其方差也最大; 2)单局平均对弈步数降至最低 9 步,说明明模型选取低水平动作而被 UTSS 早早终结的概率较大,应归咎于未排除动作空间中明显很差的动作。

仅移除威胁通道(T-TC): MCTS 平均分支因子比目标方案略有增加,平均对局长度略有下降,表明威胁通道提供的信息有助于动作选择。

综上,三项改进各自的作用: 1)作为输入的威胁通道向神经网络提供了重要的领域知识; 2)作为动作空间的先验剪枝方法,走法过滤移除了大量无用动作; 3)UTSS 实现了状态空间剪枝,还提供了可靠的监督信号,缓解了回报稀疏的窘境。

4.3 棋力水平分析

相较于采用传统的 Elo 评级系统,使用 TrueSkill 评级系统[19]进行棋力评估在评分收敛速度和稳定性方面具有显著优势。在训练初始、250 轮和 500 轮三个时间点,四种方案得到模型分别进行 50 局对弈(双方各执先 25 局)。将 2023 年中国大学生计算机博弈大赛参赛队伍的开源程序(采用 MCTS-30000 次模拟与启发式评分搜索)作为评测基准(Baseline)。

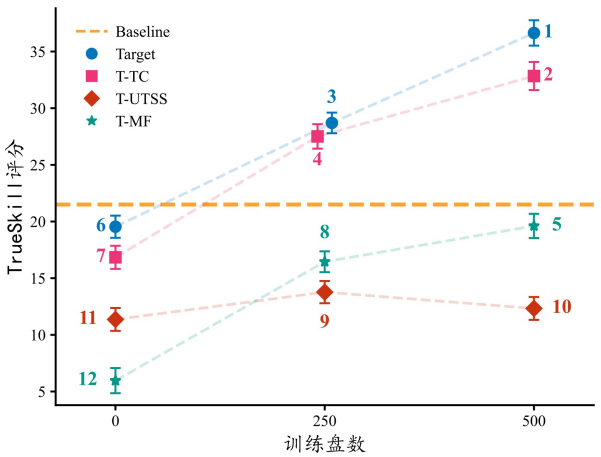


图 6 TrueSkill 评分对比

实验结果如图 6 所示,目标方案具有较高的初始棋力,经 500 轮训练后其 TrueSkill 评分从 19.54±0.98 提升至的 36.64±1.12 ,在 250 轮训练时达到 28.76±1.05 ,已经显著超越基准系统。 T-UTSS 表现最差, 500 轮训练仅带来 8.5%的提升,说明 UTSS 在 3 种改进中起到的作用最关键; T-MF 比 T-UTSS 好一些,但仍比基准程序差,这表明离开走法过滤,学习速度明显放缓; T-TC 的水平最接近目标方案,表明威胁通道有作用,但远不如 UTSS 和走法过滤效果显著。基于目标方案实现的五子棋程序在 2024 年中国大学生计算机博弈大赛五子棋项目中获得亚军,也表明所提出的方案合理可靠。

5 结论

在 AlphaZero 深度强化学习架构上,提出并验证了以下改进: 1)通过快速和轻量的 UTSS 检验棋局胜负,实现了状态空间上的后验剪枝,提升了数据分布质量的同时,还缓解了回报稀疏带来的学习困难; 2)通过简单的走法过滤规则,实现了动作空间上安全的先验剪枝,减轻策略网络学习负担的同时,也提升了 MCTS 在线推理质量;通过增加威胁通道,提高了神经网络利用领域知识的能力。此外,提出“线段-棋形”的双层棋形库提升 UTSS 执行速度,使最终方案更具实用性。未来将聚焦如何在把目标方案推广到六子棋、中国象棋等其它棋牌项目中。

参考文献:

[1]. BROWNE C B, POWLEY E, WHITEHOUSE D, et al. A survey of monte carlo tree search methods[J]. IEEE Transactions on Computational Intelligence and AI in games, 2012, 4(1): 1-43.

[2]. SILVER D, SCHRITTWIESER J, SIMONYAN K, et al. Mastering the game of go without human knowledge[J]. nature, 2017, 550(7676): 354-359.

[3]. SILVER D, HUBERT T, SCHRITTWIESER J, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play[J]. Science, 2018, 362(6419): 1140-1144.

[4]. SCHRITTWIESER J, ANTONOGLOU I, HUBERT T, et al. Mastering atari, go, chess and shogi by planning with a learned model[J]. Nature, 2020, 588(7839): 604-609.

[5]. 王亚杰, 祁冰枝, 张云博, 等. 结合神经网络的改进 UCT 在国际跳棋中的应用[J]. 重庆理工大学学报 (自然科学), 2021, 35(7): 259-265.

[6]. XU C, DING H, ZHANG X, et al. Data-efficient method of deep reinforcement learning for Chinese chess[C]//Proceedings of 2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion. Guangzhou: IEEE, 2022: 1-8.

[7]. 刘溜, 张小川, 彭丽蓉. 一种结合策略价值网络的五子棋自博弈方法研究[J]. 重庆理工大学学报 (自然科学), 2023, 36(12): 129-135.
[8]. 张小川, 严明珠, 涂飞, 等. 一种大众麻将计算机博弈的快速出牌方法[J]. 重庆理工大学学报 (自然科学), 2024, 38(5): 102-107.
[9]. LI X, ZHANG Y, WU L, et al. TibetanGoTinyNet: a lightweight U-Net style network for zero learning of Tibetan Go [J]. Frontiers of Information Technology & Electronic Engineering, 2024, 25(7): 924-937.
[10]. XIALI L, BO L, ZHI W, et al. Tjong: A Transformer‐based Mahjong AI Via Hierarchical Decision‐making and Fan Backward[J], CAAI Transactions on Intelligence Technology, 2024, 9(4): 982-995.
[11]. TIAN Y, MA J, GONG Q, et al. Elf opengo: An analysis and open reimplementation of AlphaZero[C]//Proceedings of the 36th International Conference on Machine Learning. Long Beach: PMLR, 2019: 6244-6253.
[12]. ALUS, L. V., H. J. HERIK,1 ,M. P. H. HUNTJENS. Go‐Moku solved by new search techniques[J]. Computational Intelligence, 1996, 12(1): 7-23.
[13]. WU I C, HUANG D Y. A new family of k-in-a-row games[C]//Advances in Computer Games: 11th International Conference, ACG 2005, Taipei, Taiwan, September 6-9, 2005. Revised Papers 11. Springer Berlin Heidelberg, 2006: 180-194.
[14]. ALLIS L V. Searching for solutions in games and artificial intelligence[D]. Maastricht: University of Limburg, 1994.
[15]. REISCH S. Gobang ist PSPACE-vollständig[J]. Acta Informatica, 1980, 13(1): 59-66.
[16]. CHANG-MING X, MA Z M, HAI-TAO M, et al. Generalized TSS and its optimization[C]//2013 25th Chinese Control and Decision Conference (CCDC). IEEE, 2013: 2904-2909.
[17]. 王栋年, 王军伟, 薛世超, 等. 基于深度强化学习的双置换表优化算法研究[J]. 重庆理工大学学报 (自然科学), 2024, 38(5): 145-153.
[18]. XIALI L, YANDONG C, YANYIN Z, et al. A Phased Game Algorithm Combining Deep Reinforcement Learning and UCT for Tibetan Jiu Chess[J], 2023 IEEE 47th Annual Computers, Software, and Applications Conference, Compsac, 2023: 390-395.
[19]. HERBRICH R, MINKA T, GRAEPEL T. TrueSkill™: a Bayesian skill rating system[J]. Advances in neural information processing systems, 2006, 19: 569-576..