面向机器博弈的即时差分学习研究

徐长明 马宗民 徐心和 李新星
(东北大学信息科学与工程学院 沈阳 110004)

摘 要 以六子棋机器博弈为应用背景,实现了基于即时差分学习的估值函数权值调整自动化。提出了一种新的估值函数设计方案,解决了先验知识与多层神经元网络结合的问题。结合具体应用对象的特性,提出了对即时差分序列进行选择性学习的方法,在一定程度上避免了无用状态的干扰。经过10020盘的自学习训练,与同一个程序对弈,其胜率提高了8%左右,具有良好的效果。

关键词 机器博弈,即时差分学习,六子棋
中图法分类号 TP18 **文献标识码** A

Study of Temporal Difference Learning in Computer Games

XU Chang-ming MA Zong-min XU Xin-he LI Xin-xing
(College of Information Science and Engineering, Northeastern University, Shenyang 110004, China)

Abstract Temporal Difference (Abbr. TD) learning algorithm was used to adjust weights of evaluation function by using Connect6 game as testbed in this paper, which makes the weights adjustment process can be done automatically. A new evaluation scheme was proposed, which can solve the difficult to combine the prior knowledge and multi-layer neural network organically. On account of the specific application, the method selecting part of the whole TD sequence to learn was proposed, by which the interference of useless states is prevented to a certain extent. After 10020 self-learning training, the winning rate is increased with 8% around against the same Connect6-playing program, which is a good result.

Keywords Computer games, Temporal difference learning, Connect6

机器博弈是人工智能领域中的一类复杂问题,它包括数据表示、搜索、估值、着法生成、知识库等几方面主题。其中,估值是一个难点。长期以来,要打造一个高水平的博弈程序,最耗时的工作莫过于估值函数的设计、实现以及人工调参[1]。因为手工调整估值函数参数的方法周期长、易出错、需要人工干预,所以一些自动调参的方法就被应用到机器博弈问题中,如遗传算法[2,3](Generic Algorithm)、人工神经元网络[3](Neural Network)、即时差分学习[1,2](Temporal Difference Learning, TD)等。本文以TD为主要方法。TD的主要目标是学习到一个优秀的拟合函数(即博弈程序中的估值函数)。

本文以六子棋博弈为背景。在TD学习程序TDLConn6中,选择了多层神经元网络作为估值函数的主体,并对自学习方法进行了两方面改进。第一,将先验知识与多层网络相结合。考虑到人类认知能力与机器学习能力各有短长,提出了一种同时提供人工调参接口和机器自动调参接口的估值方案,妥善地解决了整合先验知识与多层神经元网络的难题。第二,对于TD学习的序列而言,有些连续状态的变化并不适宜学习。结合领域知识,剔除了TD序列中不佳的状态。

1 估值函数

造成估值函数设计困难的因素是多方面的。对同一个客观的局面,即使专家也很难形成非常一致的评价标准;面对数目庞大的局面,估值函数应具有较高的概括性和准确性;在局面特征的识别和分析上难免机械教条,缺乏灵活性。总之,为博弈程序设计一个好的估值函数受到诸多条件的制约。

手工调参方法在早期的博弈程序中占据主流地位。在初步设计好程序的估值函数之后,主要通过观察大量对局中"犯严重策略性失误"的局面来调整估值函数的各个参数。但是,手工调参必然要求人类专家花费大量的时间和精力,容易出错。能击败人类世界冠军的西洋跳棋程序Chinook[4]的开发者竟耗费了近5年的时间手工调整估值函数的权值。

1.1 BP神经元网络

TD学习算法的主要作用是调整拟合函数的参数。在博弈问题中,估值函数就是作为TD学习目标的拟合函数。神经元网络具有极强的泛化能力,用它构造估值函数非常合适。此外,神经元网络的演化不需要人为干预,网络的训练没有假设任何先验的训练数据分布或输入-输出映射函数。对于非完整的、有缺失的和有噪声的数据,网络具有很强的适应性和容错性。必须承认,神经元网络也有明显的缺陷,如容易陷入局部最优解。

TDLConn6引入了一个3层BP神经元网络,将它作为六

到稿日期:2009-09-14 返修日期:2009-12-01 本文受国家自然科学基金项目(60873010),国家自然科学基金项目(60774097)资助。
徐长明(1978-),男,博士生,主要研究方向为机器博弈,E-mail:changmingxu@gmail.com;马宗民(1965-),男,博士生导师,主要研究方向为智能数据与知识工程;徐心和(1940-),男,博士生导师,主要研究方向为足球机器人、机器博弈等。

子棋的评估函数的有机组成部分：

·网络的输入层。参照吴的工作[5]以及我们的早期工作[6]，TDLConn6 的局面特征主要选择为整个局面的重要棋型的类型及数目、重要空交叉点的类型和数目等。最终，TDLConn6 设计了 32 个综合性的局面特征作为网络的输入。其中，黑方和白方各 16 个特征。每个特征对应一个输入单元，共 32 个输入单元。

·网络的隐藏层。隐藏层结点数目对网络的性能有一定的影响。结点过少时，学习的容量有限，不足以存储训练样本中蕴含的所有规律；结点过多时，不仅会增加网络训练时间，而且会将样本中非规律性的内容（如干扰和噪声）存储进去，反而降低泛化的能力。TDLConn6 设计了 14 个隐藏层单元。

·网络的输出层。TDLConn6 的网络输出层有 2 个单元。输出的值是实数类型，分别是对黑方和白方特征的综合评估值。局面最终的价值用“走棋方特征的综合评估值－另一方的综合评估值”这一差值来计算。

·传输函数和初始权值。网络的相邻两层之间的传输函数对网络的拟合性能影响很大。TDLConn6 选用 Sigmoid 函数，即 $g(x)=1/(1+\exp(-x))$作为传输函数。Sigmoid 函数的值域为(0.0,1.0)。对于机器的一方，用 1.0 表示取胜，0.5 表示和棋，0.0 表示输棋。权值的初始化决定了网络的训练从误差曲面的哪一点开始。在 Sigmoid 函数中，若每个结点的输入均在零点附近，则输出均在传输函数的中点。这个位置不仅远离转换函数的饱和区，而且是其变化最灵敏的区域，有助于加快网络的学习速度。因此，TDLConn6 将神经元网络的权值都初始化为零。为避免权值都为零时加权和也全为零，在隐藏层和输出层都加入了偏置值。

1.2 整合先验知识到估值函数

需要让多层神经元网络具备基本的先验知识，因为：

其一，人类知识是非常重要的，KnightCap 的相关经验[7,8]也启示我们，初始的网络权值的选取对学习速率有显著影响；增加先验知识能够加快 TD(λ)的学习速率。

其二，人的抽象认知能力是机器难以简单替代的，需要有一种机制，允许随时向估值函数中加入专家知识。自动调参可将估值函数的设计者从繁重的参数调整任务中解放出来。虽然手工调整参数有大量的缺点，但是我们没有让自动调整参数彻底地取代人的参与作用。尤其在训练初期，允许将人类知识整合到估值函数中，对训练的指导作用尤为明显。

不过，KnightCap 的估值函数是线性的，直接为某些特征赋予初始权值，就能向估值函数中顺利地引入先验知识。考虑到估值函数的表达能力，TDLConn6 采用了多层神经元网络，而网络权值本身几乎无法从数值上得到理解，当然也就没有办法以权值初值的形式将先验知识赋予神经元网络。

为此，本文提出了一种整合先验知识和多层神经元网络的估值函数，以很好地克服多层网络引入先验知识的难题。令 $f_1,f_2,\cdots,f_n$ 是局面 p 的所有特征，$s_1,s_2,s_3,\cdots,s_m\in\{f_1,f_2,\cdots,f_n\}$是与先验知识相关的一些特征。令 $S(p)=S_1(s_1)+S_2(s_2)+\cdots+S_m(s_m)$是一个完全由先验知识确定的函数，其中 S_i 是特征 s_1 的函数($1\leqslant i\leqslant m$)；再令 $NN(p)=Neural\text{-}Network(f_1,f_2,\cdots,f_n)$。整合先验知识与神经元网络的估值函数 $V(p)$形如

$$V(p)=\chi S(p)+NN(p)(S_{\max}(\cdot)-S_{\min}(\cdot))/(NN_{\max}(\cdot)-NN_{\min}(\cdot)) \quad (1)$$

式中，$S_{\max}(\cdot)$($S_{\min}(\cdot)$)是函数 $S(p)$中的最大(最小)值；$NN_{\max}(\cdot)$($NN_{\min}(\cdot)$)是函数 $NN(p)$中的最大(最小)值；χ($0\leqslant\chi$)是先验知识的影响因子，χ 越小，先验知识 $S(p)$对 $V(p)$的影响越小；反之，χ 越大，$S(p)$对 $V(p)$的影响越大。χ 可设置为一个恒定的常数，也可以在训练初期取一个较大的值，此后随着训练的增多而减小。

采用式(1)作估值函数，第一，兼顾了引入先验知识和自动调整权值的需求；第二，通过先验知识粗略勾勒出估值函数，通过神经元网络精调估值函数的权值，先验知识有助于训练加速收敛；第三，通过参数 χ 来表达对先验知识的信心。

2 TD(λ)学习

即时差分学习是最重要的增强学习方法[9]，已经成功地应用到许多实际问题中，包括机器人控制、天气预报等。TD 学习的重要目标是，在免于人工干预的情况下，通过自学习训练来调整估值函数的权值。它打破了监督学习的限制，无论在实践上，还是在理论上，以机器博弈为背景研究 TD 学习都具有重要意义。正如 Sutton 所言[1]：“在学习估值函数这方面，若 TD 方法比监督学习方法的效率还高，那么在一般的‘预测-学习’问题中，它的效率必将更高”。

TD 学习算法诞生于 Samuel 早在 1959 年提出的西洋跳棋程序[9]中。此后，在机器博弈领域内就有了更多的应用，如 Tesauro 的西洋双陆棋程序 TD-Gammon[11,12]、Schaeffer 的西洋跳棋程序 Chinook[4]、Baxter 的国际象棋程序 KnightCap[7,8]等，其水平都接近或超越人类大师。但是，相关经验还不能推广到所有的棋类。时至今日，将 TD 学习方法应用到机器博弈领域，仍然是一个开放性问题。

TD(λ)算法可解决多步的预测问题。文献[1,9]等均对它作了深入的介绍和分析。本文以机器博弈为背景，简要介绍基本 TD(λ)。令 $x_1,x_2,\cdots,x_m,z$ 为一个状态(局面)序列，其中 x_t 是 t 时刻的状态($t=1,2,\cdots,m$)，每个状态 x_t 都对应着一个向量，而向量的每个分量对应着一个局面特征；z 为走完最后一步棋的局面状态(即可从 z 判断出胜、负、和)。对于上述局面状态所构成的观测序列，有一个相应的对 z 值的估计序列 $P_1,P_2,\cdots,P_m$。理论上，每一个预测 P_t 应为 x_t 以及 x_t 之前所有状态的函数。但为了简化运算，只将 P_t 作为 x_t 的函数。令 $P_t=P_t(w,x_t)$，其中 w 是权值向量。于是，TD(λ)学习的过程便可归结为通过修正 w 来拟合估值函数的过程。本文采用“每个序列一更新”的权值更新方式，即每结束一盘棋之后，才一次性地将所有的权值增量都叠加到权值向量 w 上。

$$w\leftarrow w+\Delta w=w+\sum_{t=1}^{m}\Delta w_t \quad (2)$$

TD(λ)利用回报值进行学习。这样，利用现有的监督训练技术就能实现函数的拟合。一般的监督学习方法会构造一个监督学习序列$(x_1,z),(x_2,z),\cdots,(x_m,z)$，从而对权值向量 w 加以训练。而在即时差分学习中，用于训练的则是一个差分序列，即$(x_1,P_1),(x_2,P_2),\cdots,(x_m,P_m)$。TD 的关键在于误差 $z-P_t$ 可以用一系列预测值之和来表示，即

$$z-P_t=\sum_{k=t}^{m}(P_{k+1}-P_k) \tag{3}$$

式中，$P_{m+1}\equiv z$。

根据梯度法则，TD(λ)的 Δw_t 可由下式求出：

$$\Delta w_t=\alpha(P_{t+1}-P_t)\sum_{k=1}^{t}\left[\lambda^{t-k}\frac{\partial P_k}{\partial w}\right] \tag{4}$$

式中，梯度 $\frac{\partial P_k}{\partial w}=\frac{\partial P_k(w,x_k)}{\partial w}$ 是 P_k 关于向量 w 的偏导数。$\alpha\in[0.0,1.0]$是学习速率，决定着参数调整的幅度。α 越大，权值的调整幅度就越大；反之，幅度越小。$\lambda\in[0.0,1.0]$是衰减因子，λ 决定着学习过程是从短期的还是从长期的预测中学习。特别是，当 $\lambda=0$ 时，仅从下一个状态中学习；当 $\lambda=1$ 时，仅从最终结果中学习，TD 学习退化成监督学习。

最后，权值向量更新结果由下式得出。

$$w\leftarrow w+\sum_{t=1}^{m}\alpha(P_{t+1}-P_t)\sum_{k=1}^{t}\left[\lambda^{t-k}\frac{\partial P_k}{\partial w}\right] \tag{5}$$

式中，$P_{m+1}=z$。式(5)中还包含着可增量执行的计算[1]。若令

$$e_t=\sum_{k=1}^{t}\left[\lambda^{t-k}\frac{\partial P_k}{\partial w}\right] \tag{6}$$

则有

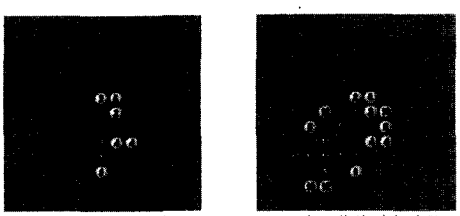
$$\begin{aligned}e_{t+1}&=\sum_{k=1}^{t+1}\left[\lambda^{t+1-k}\frac{\partial P_k}{\partial w}\right]=\frac{\partial P_{k+1}}{\partial w}+\sum_{k=1}^{t}\left[\lambda^{t+1-k}\frac{\partial P_k}{\partial w}\right]\\&=\frac{\partial P_{k+1}}{\partial w}+\lambda e_t\end{aligned} \tag{7}$$

3 应用对象——六子棋

connect(m,n,k,p,q)代表一族 k 子棋[5]，也称连珠棋。博弈双方分别执黑和执白，黑先。*connect*(m,n,k,p,q)的涵义是：第一，棋盘包含 $m\times n$ 个交叉点。第二，黑第一次下 q 枚棋子，此后双方轮流下 p 枚棋子。第三，在(水平的、垂直的、对角线方向的)任何一条线上，能率先形成本方连续不间断的 k 子序列者取胜；若双方均无法取胜，判和。其中，六子棋可形式化地描述为 *connect*$(m,n,6,2,1)$。

3.1 高级的领域知识——TSS 策略

威胁空间搜索(Threat Space Search，TSS)[14,15]是高水平连珠棋博弈程序不可或缺的技术，它通过不断地主动制造威胁，强制对手被迫防守，寻机获胜的策略，如图 1 所示。



(a)黑可 TSS 获胜的局面(轮黑走)　(b)一个可能的对弈过程

图 1　六子棋中的 TSS 搜索

由于 TSS 在六子棋中非常常见，因此本文将其作为一种高级的领域知识，而不仅仅是一种搜索。在图 1(a)中，黑可运用 TSS 搜索策略获胜。图 1(b)给出了黑胜的一个最优解。不难看出白完全被黑牵制，无论怎样努力，终究会形成类似黑 13 时所形成的局面，使得白告负。

设当前局面为 p，用 A(Attacker)和 D(Defenderr)表示两个博弈者。假设我方是 A，对方为 D，令

• TSS(p,A,D)=true 表示正轮到 D 走棋时，对于 D 的任意着法，A 都能通过 TSS 取胜；

• TSS(p,A,y)=false 表示正轮到 D 走棋时，存在着 D 的某个着法，使 A 不能通过 TSS 取胜；

• TSS(p,A,A)=true 表示正轮到 A 走棋时，存在着 A 的某个着法，使 A 能通过 TSS 取胜；

• TSS(p,A,A)=false 表示正轮到 A 走棋时，对于 A 的任意着法，A 都不能通过 TSS 取胜。

TSS 常常用于判断机器一方 A 在当前局面 p 下能否获胜。进一步地，提出了 Anti-TSS 策略，它用于判断对方 D 能否利用 TSS 获胜，有助于帮助 A 预防明显的失误。令

• Anti-TSS(p,D,A)=true 表示正轮到 A 走棋，对于 A 的任意着法 m_A，都有 $TSS(\text{successor}(p,m_A),D,D)$=true；

• Anti-TSS(p,D,x)=false 表示正轮到 x 走棋，存在着 x 的某个着法 m_A，使得 TSS$(\text{successor}(p,m_A),D,D)$=false；

• Anti-TSS(p,D,D)=true 表示正轮到 D 走棋，存在着 D 的某个着法 m_D，使得 TSS$(\text{successor}(p,m_D),D,x)$=true；

• Anti-TSS(p,D,A)=false 表示正轮到 D 走棋，对于 D 的任意着法 m_D，都有 TSS$(\text{successor}(p,m_D),y,A)$=false。

其中，successor(p,m)是着法 m 作用于局面 p 形成的新局面。

3.2 基于 TSS 的可学习状态序列截取

在 TD 学习中，连续的状态序列是学习的素材。就 TD 自学习程序而言，学习的素材就是一局局的棋谱。但是，TDLConn6 学习完整的棋谱效果并不好，最佳的方案是选择部分棋谱加以学习。如图 2 所示，站在 A 的立场上，当发现局面 x_m 能满足 TSS(x_m,A,A)=true∨Anti-TSS(x_m,D,A)=true 的时候，只要在 TSS 和 Anti-TSS 的指导下就能正确地选择出后继着法，根本不需要评估函数 $V(p)$。

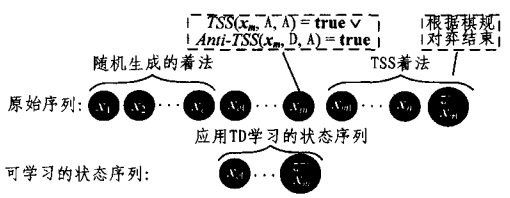


图 2　可应用 TD 学习的状态序列

这时，对于状态 $x_{m+2},\cdots,x_{n+1}$ 上的学习要分两种情况。对于 TDLConn6 就不会进行 TD 学习，转而执行简单的监督学习。当(TSS(x_s,A,A)=true)∧$(s\leqslant c)$成立时，在 s 时刻的状态直接进行监督学习$(P_s,1.0)$；当(Anti-TSS(x_s,A,A)=false)∧$(s\leqslant c)$时，在 s 时刻的状态直接进行监督学习$(P_s,0.0)$。

既然如此，后继局面 $x_{m+2},\cdots,x_{n+1}$ 就不应参与调整权值。因为这些局面已经完全被 TSS 和 Anti-TSS 搜索所覆盖，不必再由 $V(p)$来估值。所以，在 TDLConn6 中，一旦发现某个局面 p 使得 TSS(p,A,A)=true∨Anti-TSS(p,D,A)=true 成立，就应结束本局学习，开始下一局学习。

3.3 TDLConn6 的框架结构

TDLConn6 的框架结构如图 3 所示，由 3 部分构成：

1)六子棋机器博弈引擎——Conn6_w(黑)和 Conn6_b(黑)。这两个引擎均属于 NEUConn6 引擎的衍生版本，与 NEUConn6 最大的不同是估值模块增加了一个神经元网络。

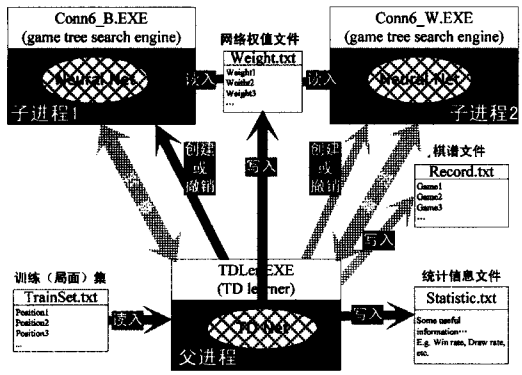


图 3　TDLConn6 的框架结构

2)TD 学习器——TDLer。TDLer 的任务是：a)保证两个引擎在免于人工干预的情况下连续对弈；b)以弈棋谱为输入，执行 TD(λ)算法修正权值向量；c)定期记录学到的黑白双方各自的网络权值，统计 TD 学习过程中的相关信息。

3)数据文件。数据文件中最重要的是存储神经元网络权值的文件。TDLer 则负责调整网络权值，并将调整后的最新权值写入该文件。引擎要从这个文件中读入神经元网络权值，从而获得最新学到的估值函数，用于指导当前的对弈。此外，还有棋谱文件、训练集文件、统计信息文件等。

TDLer 的神经元网络权值分为黑方权值和白方权值。轮黑走时，用黑方的权值评估；轮白走时，用白方的权值评估。同理，TDLConn6 的两个自学习引擎也应根据自己的角色读入正确的网络权值。图 3 的引擎 Conn6_b、引擎 Conn6_w 以及学习器 TDLer 都包含神经元网络，且网络都是同构的。所不同的是，两个引擎仅仅把网络用作估值函数；而 TDLer 中的网络则与 TD 算法紧密结合。TD 算法的执行细节如图 4 所示。在图 4 中，“第一时间顺序”用于刻画马尔科夫决策过程中各个状态的时序；“第二时间顺序”用于描述任意两个相邻状态之间 TD 学习要进行操作的时间顺序。

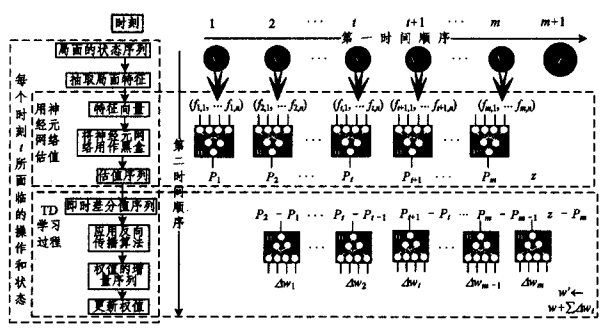


图 4　TD 学习算法的执行过程

4　实验

4.1　训练集的生成

六子棋的开局、中局、残局策略几乎没有明显的不同，不妨选择常见的开局局面作为训练对象。我们挑选了 26 个均衡的开局局面，它们的第一颗黑子均在天元，都只有 3 枚棋子。取 S_T＝{与这 26 个局面按旋转对称、轴对称等价的 170 个局面}。让 NEUConn6 对 S_T 中 170 个局面进行自对弈，结果为黑胜 76 盘(占 44.7%)，白胜 94 盘(占 56.3%)，和 0 盘。

S_T 仅有 170 个局面，因为太小而不宜直接用作训练集。在兼顾训练局面质量的基础上，提出一种随机性地自动扩展集合 S_T 的方法，使扩展后的局面集的训练样本具有代表性。

已知 $S=S_T$，$S'=${自动生成的训练局面}。$||p||$表示局面 p 中棋子的数目。在 TDLConn6 中，取常数 $c=8$，令局面 $pos\in S$，$pos'\in S'$，且$||pos'||-||pos||\leqslant c$，则随机生成满足上述条件的 pos'的算法如图 5 所示。在图 5 中，当前局面为 $p\in S\cup S'$，p'是 p 的一个直接后继局面，$p'\in S'$。此外，

$$P(m_s\mid p)=\frac{V(Succ(p,side,m_s))}{\sum_{i=1}^{k}V(Succ(p,side,m_i))} \tag{8}$$

$$P'(m_s\mid p)=\frac{(k-s+1)^2P(m_s\mid p)}{\sum_{i=1}^{k}[(k-i+1)^2P(m_i\mid p)]} \tag{9}$$

估值函数有双重作用，除了量化局面形势的作用外，还要充当着法选择的标准。式(8)中涉及的 $V(p)$是局面的估值函数，整合了先验知识的函数形式，见式(1)。由于着法选择带有随机因素，因此即便从同一个初始局面出发，后续着法也往往不同。况且，每盘棋结束之后，神经元网络权值往往发生变化，候选着法列表和候选着法的估值也随之变化。所以，图 5 扩展 S_T 的方法比较理想，可得到一个相对公平的测试集。

```
1.  move_list := GenAllMove(p);
2.  k := Min(TOP-K, num_of_moves);
3.  best_k_list := SelectBestK(k, move_list);
4.  for ( i=1; i ≤ k; ++i) {
5.      probability[i] := P'(best_k_list[i]|p);
6.  }
7.  x := SelectOneMoveAtRandom(probability, k);
8.  p' := MakeMove(p, best_k_list[x]);
```

图 5　训练集的随机性扩展算法

在自学习训练过程中，自动生成训练局面的阶段($||pos'||-||pos||\leqslant c$)采用带有随机性的着法选择策略；在剩余阶段($||pos'||-||pos||>c$)，根据极大极小原理选择最佳着法。

4.2　“零知识”自学习训练

在 TDLConn6 中，采用“零知识”的估值函数，即权值都为 0 的神经元网络 $NN(p)$(参见式(1)定义)作为估值函数，并且不采用 TSS 策略。经过 10000 盘自学习训练(约 4h)，其着法仍然散乱无章。这说明，它还不能很好地学到占据中间位置的重要性，更不用说 TSS 搜索这种高级知识了。与仅了解规则的选手的 3 盘比赛中，皆负。在“零知识”条件下，经 10000 盘训练的与未经训练的两个 TDLConn6 相比，除和棋外，其胜负比率几乎接近 1∶1。显然，“零知识”不可取。

4.3　先验知识引导的自学习训练

在 TDLConn6 中，采用式(1)作估值函数，以此来适度地加入先验知识。此外，TDLConn6 还将 TSS 策略视为六子棋的高级知识。特别地，两个对弈的引擎思考时，都检验可否通过 TSS 直接取胜。若一方发现自己能够取胜，则立刻通知对手终止本局对弈，并开始新一局训练。

对 S_T 中的每个局面都进行 60 次自学习训练，共 10020 次。考虑到训练次数不多(与 TD-Gammon 训练 150 万次相比)，令 $\alpha=0.8$，使参数调整的步幅较大；此外，取 $\lambda=0.7$。整个训练耗时约 157h(约 6.6 天)。为了评价学习的效果，每训练 1000 盘把神经元网络的权值保存下来，用于分析学习的效果，如图 6 所示。

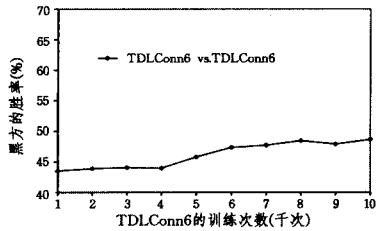


图 6　TDLConn6 自学习训练时黑方胜率

TD 训练过程中，为观察 TDLConn6 的水平变化，对于 S_T 中的每个局面，都让 TDLConn6 与 NEUConn6 对弈一局（分先后手）。这时，1)将图 3 中的一个引擎换成 NEUConn6；2)关闭 TDLConn6 中的随机策略，在整个对弈过程中，每个引擎都选择各自的最佳着法。TDLConn6 与 NEUConn6 的对弈结果如图 7 所示。在图 7 中的横坐标 i 处，TDLConn6 使用权值向量 w_i（训练 $1000\times i$ 盘棋时所保存下来的）与 NEUConn6 对弈。经 10020 盘（耗时约 157h）的自学习，TDLConn6 执黑时，在 S_T 中的胜率提高了 3%～5%；TDLConn6 执白时，胜率提高了 8%左右。程序水平变化的整体趋势是，随着训练的次数增多，水平逐渐提高。可见，应用 TD 学习有助于提高六子棋程序的博弈水平。

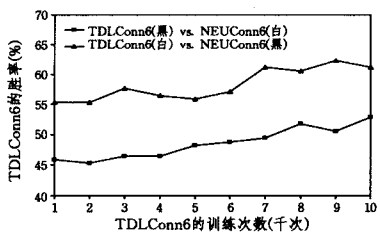


图 7　TDLConn6 与 NEUConn6 的对弈结果

结束语　本文将 TD(λ)算法应用到六子棋机器博弈中。实验结果已经表明，从零知识开始进行自学习训练的方案能够在西洋双陆棋中取得良好效果，但不适合六子棋博弈。为此，在估值函数的设计上，本文将先验知识平滑地嵌入到神经元网络中。经过 10020 盘自学习训练，程序的水平明显提高。可以断言，若像 TD-Gammon 那样进行上百万盘的自学习训练（大约将近 3 年时间），TDConn6 的博弈水平必然会更好。

参 考 文 献

[1] Sutton R S. Learning to Predict by the Method of Temporal Differences[J]. Machine Learning, 1988, 3(1): 9-44

[2] 王骄，王涛，等. 中国象棋计算机博弈系统评估函数的自适应遗传算法实现[J]. 东北大学学报：自然科学版，2005，26(10)：949-952

[3] Autonès M, Beck A, et al. Evaluation of Chess Position by Modular Neural Network Generated by Genetic Algorithm[J]. Genetic Programming, 2004, 3003: 1-10

[4] Schaeffer J, Burch N, Bjornsson Y, et al. Checkers Is Solved[J]. Science, 2007, 317(5844): 1518-1522

[5] Wu I-Chen, Huang Dei-Yen. A New Family of k-in-a-row Games [C]// Proceedings of The 11th Advances in Computer Games Conference. 2005: 88-100

[6] Xu Chang-ming, Ma Z M, Xu Xin-he. A Method to Construct Knowledge Table-base in k-in-a-row Games[C]// Proceedings of ACM Symposium on Applied Computing. 2009: 929-933

[7] Baxter J, Tridgell A, Weaver L. KnightCap: A Chess Program that Learns by Combining TD(λ) with Game-Tree Search[C]// Proceedings of the 15th International Conference on Machine Learning. Madison, 1998: 28-36

[8] Baxter J, Tridgell A, Weaver L. Experiments in Parameter Learning Using Temporal Differences[J]. ICGA Journal, 1998, 21(2): 84-99

[9] Sutton R S, Barto A G. Reinforcement Learning: An Introduction [M]. Cambridge: MIT Press, 1998

[10] Baxter J, Tridgell A, Weaver L. TDLeaf(λ): Combining Temporal Difference Learning with Game-Tree Search[C]// Proceedings of the 9th Australian Conference on Neural Networks. 1998: 168-172

[11] Tesauro G. Temporal difference learning and TD-Gammon[J]. Communications of the ACM, 1995, 38(3): 58-68

[12] Tesauro G. TD-Gammon: a Self-Teaching Backgammon Program, Achieves Master-Level Play[J]. Neural Computation, 1997, 6: 215-690

[13] Tesauro G. Practical Issues in Temporal Difference Learning [J]. Machine Learning, 1992, 8: 257-277

[14] Allis L V, van den Herik H J, Huntjens M P H. Go-Moku Solved by New Search Techniques[J]. Journal of Computational Intelligence, 1995, 12(1): 7-24

[15] Allis L V. Searching for Solutions in Games and Artificial Intelligence [D]. Maastricht, The Netherlands: University of Limburg, 1994

(上接第 174 页)

[2] 董威，王戟，齐治昌. UML Statecharts 的模型检验方法[J]. 软件学报，2003(14)：750-756

[3] Zhou Xiang, Shao Zhi-qing. ASM Semantic Modeling and Checking for Sequence Diagram[C]// ICNC'09. Vol. 5, 2009: 527-530

[4] 占学德，缪淮扣. 基于 UML 状态图测试的充分性准则[J]. 计算机科学，2005，32(5)：230-235

[5] 董涛，顾庆，陈道蕾. 基于状态图的测试技术研究[J]. 计算机科学，2007，34(10)：264-268

[6] Ober I. An Asm Semantics of UML Derived from the Metamodel and Incorporation Actions, Advances in Theory and Applications[C]// 10th International Workshop. ASM, 2003

[7] Bernardi S, Donatelli S, Merseguer J. From UML Sequence Diagrams and Statecharts to analysable Petri Net models[C]// WOSP '02. Italy: Rome, 2002: 35-45

[8] Tenzer J, Stevens P. Modelling recursive calls with UML state diagrams[J]. Lecture Notes in Computer Science, 2003, 2621: 135-149

[9] Borger E, Stark R. Abstract State Machines[M]. Springer-Verlag, 2003

[10] Compton K, Huggins J, Shen W. A Semantic Model for the State Machine in the Unified Modeling Language[C]// UML 2000 workshop. 2000: 25-31