学校代码：	10135
论文分类号：	TP399
学　　号：	20094019013
研究生类别：	全日制

内蒙古師範大學

硕士学位论文

对α－β剪枝算法的性能改进研究

Improvement of the performance of the alpha-beta pruning algorithm

学科门类：　工学

一级学科：　计算机科学与技术

学科、专业：　计算机应用技术

研究方向：　软件工程与集成技术

申请人姓名：　曹森

指导教师姓名：　苏贵斌 副教授

二〇一二年四月二十日

学校代码：	10135
论文分类号：	TP399
学　　号：	20094019013
研究生类别：	全日制

内蒙古師範大學

硕士学位论文

对α－β 剪枝算法的性能改进研究

Improvement of the performance of the alpha-beta pruning algorithm

学 科 门 类 ： 工学

一 级 学 科 ： 计算机科学与技术

学科 、专 业： 计算机应用技术

研 究 方 向： 软件工程与集成技术

申 请 人 姓 名 ： 曹森

指导教师姓名： 苏贵斌 副教授

二〇一二年四月二十日

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果，尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含本人为获得内蒙古师范大学或其它教育机构的学位或证书而使用过的材料。本人保证所呈交的论文不侵犯国家机密、商业秘密及其他合法权益。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示感谢。

签名：曹森　　日期：2012年6月14日

关于论文使用授权的说明

本学位论文作者完全了解内蒙古师范大学有关保留、使用学位论文的规定：内蒙古师范大学有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅，可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文，并且本人电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

签名：曹森　　导师签名：

日期：2012年6月14日

中 文 摘 要

α－β 剪枝算法是计算机博弈类游戏算法的基础，是一种优化的博弈树搜索算法，使用α－β 剪枝算法搜索博弈树时，可以剪去一些不必要的分枝，从而提高搜索效率；而α－β 剪枝算法的剪枝效率又取决于所构建的博弈树的分枝的排列顺序，理想排列顺序下的剪枝效率和最差情况下相比差别极大。本文提出的使用机器学习的算法来改进博弈树分枝的排列顺序，可以极大的提高α－β 剪枝算法的剪枝效率。

在文中，作者介绍了α－β 剪枝算法的剪枝原理及已经存在的各种优化算法，给出了构建用于辅助α－β 剪枝算法剪枝的学习系统的通用方法及注意事项。为了验证本文所提出的改进方法，作者首先设计并实现了基于α－β 剪枝算法的井字棋游戏，然后通过人机博弈来获取用于训练学习系统的训练样例；其次，作者设计并实现了一个 BP 神经网络，该网络由 14 个隐藏节点，9 个输出节点构成，能够对博弈树的分枝按其成为最佳路径的概率排序，经过前面提取的训练样例的训练后，BP 神经网络对测试样本中的 50%可以做出正确预测，即训练后的神经网络能够以当前棋局状态为输入，以 50%的正确率预测下一步的最佳走法；最后，作者把训练好的 BP 神经网络加到了井字棋游戏中，经过验证发现，改进后的井字棋与原有井字棋相比，在博弈结果为平局的情况下，所搜索的节点个数可以减少 35%左右。

经过一系列的尝试和验证发现，本文所提出方法确实能够很大程度的提高α－β 剪枝算法的剪枝效率。

关键词： α－β 剪枝算法，机器学习，BP 神经网络,井字棋，象棋

ABSTRACT

The alpha-beta pruning algorithm is the base of computer game algorithm, it is also an optimized game tree search algorithm, using the alpha-beta pruning algorithm searches the game tree, you can cut some unnecessary branches, thereby improving the search efficiency; while the alpha-beta pruning algorithm pruning efficiency in turn depends on the game tree branching order; the ideal order game tree pruning efficiency is much better than the worst case. This paper presents the use of machine learning algorithms to improve the order of the game tree branches, can greatly improve the efficiency of the pruning of the alpha-beta pruning algorithm.

In the text, the author describes the pruning principle of the alpha-beta pruning algorithm and various optimization algorithms that already exist, given the common methods and precautions of the build learning system for the auxiliary alpha-beta pruning algorithm for pruning. In order to verify that the proposed improvement method, the author first designed and implemented tic-tac-toe chess game which based on the alpha-beta pruning algorithm, and then by the man-machine game to get the training examples used to train the learning system; Second, the author designed and implemented a BP neural network, the network consists of 14 hidden nodes, 9 output nodes , which can sort the game tree branches according to their best path probability, after a training sample extracted earlier training, BP neural network can identify 50% of the test samples, that is, trained neural network to the current chess game state as input to predict the next best move, the correct rate of 50%; Finally, the author added the trained BP neural network to the game of tic-tac-toe chess, the improved tic-tac-toe chess compared with the original tic-tac-toe chess, to the case in the outcome of the game for a draw, the number of nodes search can reduce about 35% proven.

After a series of trials and validation, the proposed method really can

greatly improve the pruning efficiency of the alpha-beta pruning algorithm.

KEY WORDS: alpha-beta pruning algorithm, machine learning, BP neural network, tic-tac-toe chess, Chinese chess

目 录

第 1 章 引言

1.1 国内外研究现状

博弈问题无所不在，小到孩童的游戏与争论、各种场合下的讨价还价，大到商家的竞争、各种突发事件（恐怖、灾害）的应急处理、国家的外交、流血的和不流血的战争，只要局中的双方主体存在某种利益冲突，博弈便成为矛盾表现和求解的一种方式。

博弈一向被认为是富有智能行为的游戏，因而很早就受到人工智能界的重视，早在 60 年代就已经出现若干博弈程序，并达到一定水平。博弈问题的研究还不断提出一些新的研究课题，从而也推动了人工智能研究的发展。

计算机博弈是当前人工智能的研究热点之一。计算机博弈简单地说就是让计算机像人一样从事需要高度智能的博弈活动。由于博弈需要思维和推理，是人类的智能体现。因此智能博弈成为研究人工智能的常用平台之一。

自从计算机诞生以来,许多著名学者都曾经涉足这一领域的研究工作。计算机之父冯·诺依曼就提出了用于博弈的极大极小定理 ,信息论的创始人香浓教授,又给出了极大极小值算法 ,著名的计算机学家阿兰·图灵也曾做过计算机博弈的研究 。

如今,计算机博弈已经成为一个独立而重要,颇有发展前途的学术研究领域，但是它在中国的发展还很落后。可喜的是，计算机博弈在中国已经起步,并初具规模。象棋、围棋与六子棋等棋类开发队伍在不断壮大,国内外的比赛也引起了社会的关注。

通常，解决博弈问题的一般方法是构建博弈树，通过对博弈树的搜索来找到最佳的应对策略。α－β 剪枝算法是一种优化的博弈树搜索算法 ，可以减少对博弈树冗余数据的搜索。经人们研究发现，α－β 剪枝算法本身还可以再进一步优化，在前人的不懈努力下，目前已经有了多种对α－β 剪枝算法的优化方法，主要有 negscout 算法、fail soft 算法、mtdf 算法、渴望算法、迭代深化算法、历史启发算法、杀手启发算法、空着搜索等。

博弈树的搜索算法丰富多彩，改革、重组与创新的余地很大。除了前面提到的优化算法外，本文所提出的利用机器学习知识使博弈树的节点有序，也可以很大程度的提高α－β 剪枝算法的剪枝效率。

1.2 课题的主要工作及章节安排

本课题的主要工作是使用α－β 剪枝算法设计并实现一个井字棋游戏，并在井字棋游戏的基础上设计并实现一个可以对博弈树的节点进行排序的学习系统并将学习系统与α－β 剪枝算法相结合，从而提高α－β 剪枝算法的剪枝效率。具体内容有：

- 井字棋的设计与实现。
- 学习系统的构建。
- 学习系统的训练。
- 学习系统与α－β 剪枝算法的结合。

本文后续章节的安排如下：

第二章包括对α－β 剪枝算法的介绍，已有的α－β 剪枝算法的优化方法，本文采用的对α－β 剪枝算法的优化方法。

第三章为构建用于对博弈树节点进行排序的学习系统的通用方法。

第四章为井字棋的设计与实现，并以井字棋为例，详细说明了构建用于对博弈树节点进行排序的学习系统的具体方法，还说明了如何使学习系统与α－β 剪枝算法相结合。

最后总结了本文的研究工作和研究成果，并就本文的研究问题提出了本人的思考见解及展望。

第 2 章α－β 剪枝算法及其优化原理

2.1α－β 剪枝算法

2.1.1 博弈论

2.1.1.1 本文所研究的博弈

在讲α－β 剪枝算法之前，需要先讲一下博弈问题。博弈论是二人或多人在平等的对局中各自利用对方的策略变换自己的对抗策略，达到取胜目标的理论。博弈论是研究互动决策的理论。博弈可以分析自己与对手的利弊关系，从而确立自己在博弈中的优势，因此有不少博弈理论，可以帮助对弈者分析局势，从而采取相应策略，最终达到取胜的目的。博弈的类型分为：合作博弈、非合作博弈、完全信息博弈、非完全信息博弈、静态博弈、动态博弈，等等。本文所研究的内容属于完全信息动态博弈。

完全信息动态博弈，是指博弈中信息是完全的，即双方都对己方与对方的战略空间和战略组合下的下一步应对策略有完全的了解，但行动是有先后顺序的，后动者可以观察到前者的行动，了解前者行动的所有信息，而且一般都会持续一个较长时期（比如象棋）。用比较通俗一点的说法，本文中所研究的博弈问题具有以下特点：

- 双人对弈，轮流走步。
- 信息完备，双方所得到的信息是一样的。

零和，即对一方有利的棋，对另一方肯定是不利的，不存在对双方均有利或无利的棋。

对完全信息动态博弈问题，一般是通过构建博弈树，然后对博弈树搜索来解决。标准博弈树搜索由四部分组成：

1．状态表示。

2．候选走法产生。

3．确定目标状态。

4．一个确定相对优势状态的静态评估函数。

2.1.1.2 博弈树

在博弈过程中，如果把某一棋盘局面作为一个节点，那么对于该节点，所有可能着法所形成的新局面就成为该节点的儿子节点，从儿子节点再走一步棋形成的局面就是孙子节点，以此类推，直到可以分出胜负或平局的局面，这样就可以构造出一棵博弈树。下面以井字棋为例，说明一下博弈树的构建方法。

井字棋的规则与人们常玩的五子棋相似，不同点在于井字棋是在如图 2-1 所示的棋盘上，以叉（×）和圈（○）为棋子进行角逐，首先在横向、竖向或斜向上连成三子的一方获胜。

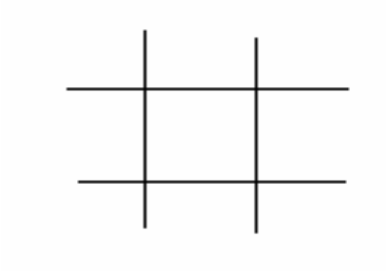

图 2-1 井字棋棋盘

为了说明方便，将用 1-9 来说明棋盘的九个位置，具体表示方法如图 2-2 所示。

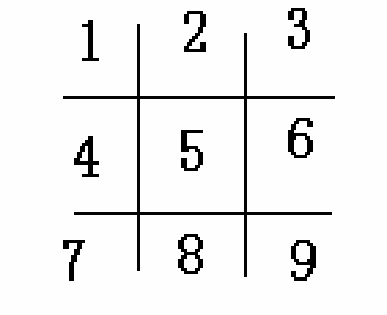

图 2-2 棋盘位置编号

假设当前棋局状态如图 2-3 所示，计算机使用圆圈，另一方使用叉，下一步该由计算机先走。

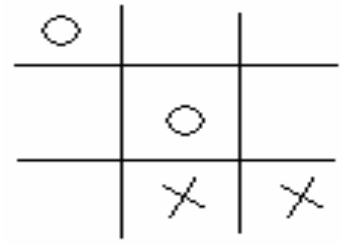

图 2-3 当前棋局状态

则计算机会构造如图 2-4 所示的博弈树，来决定下一步的走法。

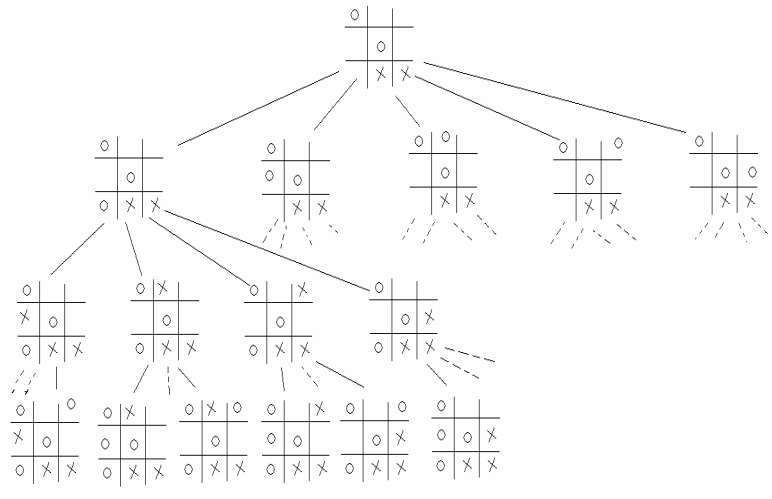

图 2-4 计算机构建的博弈树

把当前棋局状态作为根节点（根节点为第 1 层），圈方的每一个可着子点构成一个新的节点，形成第二层；然后轮到叉方走，以第二层的每一个节点为根节点，叉方的每一个可着子点构成一个新的节点，形成第三层，以此类推，形成第四层，第五层等，直到出现一方获胜的局面（即出现叉方或圈方有一方有三子连成一线）或者是平局（即九个位置已被占满，但还未分出胜负）。图 2-4 就可以看作是一个博弈树。

在博弈过程中，任何一方都希望自己取得胜利。因此，当某一方当前有多个行动方案可供选择时，他总是挑选对自己最为有利而对对方最为不利的那个行动方案。此时，如果我们站在 MAX 方（圈方）的立场上，则可供 MAX 方选择的若干行动方案之间是“或”关系，因为主动权操在 MAX 方手里，他或者选择这个行动方案，或者选择另一个行动方案，完全由 MAX 方自己决定。当 MAX 方选取任一方案走了一步后，MIN 方（叉方）也有若干个可供选择的行动方案，此时这些行动方案对 MAX 方来说，它们之间则是“与”的关系，因为这时主动权操在 MIN 方手里，这些可供选择的行动方案中的任何一个都可能被 MIN 方选中，MAX 方必须应付每一种情况的发生。（第二层的各个节点是轮到圈方走棋时可供选择的走法，它们之间是或的关系，只要有一个分枝能达到必赢（思考的时候可以把赢当作 1，把输当作 0），则圈方就必赢；第三层的节点是轮到叉方走棋时可供选择的走法，对于圈方来说，第三层的每一种走法都要考虑到，如果有一种走法导致必输，而圈方没有在第二层采用相应措施的话，则会导致圈方必输，所以叉方的所有选择对圈方来说是“与”的关系）。

这样，如果站在某一方(如 MAX 方，即 MAX 要取胜)，把上述博弈过程用图表示出来，则得到的是一棵“与或树”。描述博弈过程的与或树称为博弈树。

我们假定 MAX 先走，处于奇数深度级的节点都对应下一步由 MAX 走，这些节点称

为 MAX 节点，相应地偶数深度级为 MIN 节点。

构建博弈树时把初始格局当作根节点，或节点和与节点逐层交替出现，自己一方扩展的节点之间是或关系，对方扩展的节点之间是与关系，双方轮流扩展节点。

博弈树的搜索算法很多，如极大极小值算法，α－β 剪枝算法及它的改进算法等。一般采用类似于树的后根搜索的方法，利用评估函数来确定当前棋局状态下，下一步如何走赢的机会最大，下一步的最佳走法即是产生赢的机会最大的那种走法。在图 2-4 中，以根节点的第一分枝为例，第三层的前四个表示的是圈方在位置 7 画圈后叉方所有的可能走法，由第四层前 6 个可知，无论叉方如何走，都可以保证圈方必赢，对于必赢的情况，给根节点第一分枝赋一个较大的值（一般用正无穷来表示必赢）；与此类似，由下往上分析根节点的其他分枝的获胜机率，然后比较根节点各个分枝的获胜情况，产生获胜机率最大的那个节点的走法，即为当前棋局状态下的下一步最佳走法。由以上分析可知，下一步的最好走法是在位置 7 画圈。

上面只是简单的介绍了博弈树搜索的基本思想，下面介绍一下博弈树搜索的实用算法—极大极小值算法。

2.1.2 极大极小值算法

香侬教授在 1950 年首先提出了极大极小值算法，从而奠定了计算机博弈的理论基础。

博弈树搜索属于对抗性搜索。在博弈过程中，任何一方都希望自己取得胜利。在某一方进行博弈策略选择时，他总是挑选对自己最为有利而对对方最为不利的那个行动方案。因此，某一方在走棋时必须遵守如下原则:考虑到对自己最不利的情况，从最坏的可能中选择最好的;并假定对手不会犯错误，即对手总是选择对他最有利的方法走，所以不能采取任何冒险行动。这个原则就叫极大极小原则，用该原则指导搜索的算法就叫做极大极小值算法。

极大极小值算法是解决博弈问题的基本方法。极大极小值算法是通过构建极大极小博弈树然后对极大极小博弈树进行搜索来产生对当前局面最佳应对策略的一种算法。极大极小博弈树的构建方法与前面讲的博弈树的构建方法类似，不同点在于极大极小博弈树的每个节点是一个数值。

极大极小值算法的具体搜索过程如下例所示。

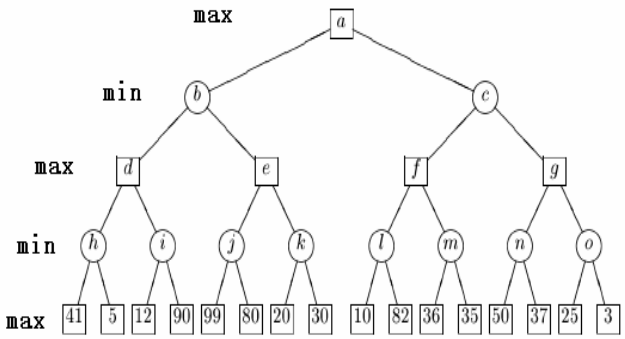

图 2-5 极大极小博弈树

在图 2-5 中，第一层做 max 搜索，第二层做 min 搜索，第三层做 max 搜索，第四层做 min 搜索，第五层做 max 搜索。节点 a 可看作是当前棋局状态，节点 b 和 c 可看作是当前棋局状态下走一步后所产生的棋局状态。节点 d 和 e 是当己方走一步产生 b 棋局状态后，对方走一步所产生的棋局状态。节点 f 和 g 的产生方式与 d 和 e 相同。节点 h 和 i 是在己方走一步产生 b 状态，对方走一步产生 d 状态后，己方再走一步产生的棋局状态。节点 j、k、l、m、n、o 的产生与节点 h 和 i 的产生相同。第五层的叶节点是己方和对方交替两次各走一步所产生的棋局状态，其值是利用评估函数对棋局状态进行评估所得的值。搜索这个博弈树的目的是确定当前棋局状态下应该如何走下一步，即是确定应该选择产生 b 状态的走法还是应该选择产生 c 状态的走法。具体决策时要由叶节点往根节点倒推。第四层的值由第五层确定（假设根节点为第一层）。第四层为 min 层，第四层的每个节点的值取它的子女节点中最小的值，h 节点的子女节点的值为 41 和 5，所以 h 节点的值为 5；同理，i 的值为 12，j 的值为 80，k 的值为 20，l 的值为 10，m 的值为 35，n 的值为 37，o 的值为 3；第三层为 max 层，第三层的每个节点的值取它的子女节点中最大的值，d 节点的子女节点的值为 5 和 12，所以 d 节点的值为 12，同理，e 节点的值为 80，f 节点的值为 35，g 节点的值为 37；第二层为 min 层，节点值的求法同第四层，得 b 节点的值为 12，c 节点的值为 35；第一层为 max 层，节点值的求法同第三层，a 节点的值为 35。得到所有节点的值的博弈树如图 2-6 所示。

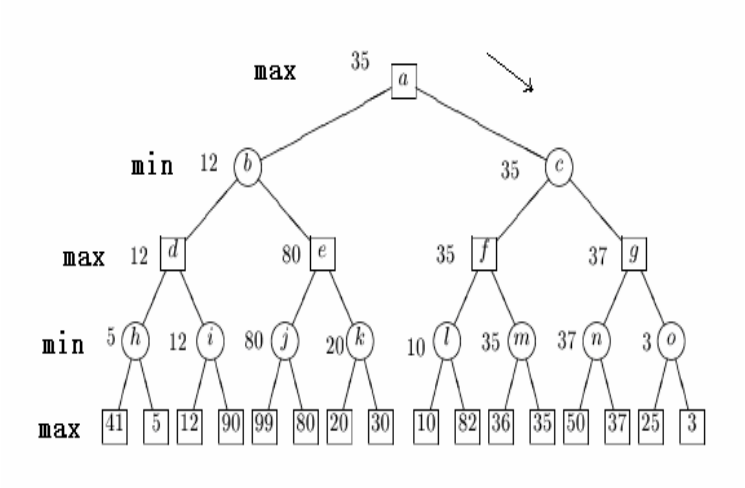

图 2-6 搜索完成后的极大极小博弈树

由以上分析可知，当前棋局状态下的最佳走法是产生 c 棋局状态的走法。

极大极小算法的基本思想是：

(1) 设博弈的双方中一方为 MAX，另一方为 MIN。然后为其中的一方(例如 MAX)寻找一个最优行动方案。

(2) 为了找到当前的最优行动方案，需要对各个可能的方案所产生的后果进行比较,具体地说，就是要考虑每一方案实施后对方可能采取的所有行动，并计算可能的得分。

(3) 为计算得分，需要根据问题的特征信息定义一个评估函数，用来估算当前博弈树叶子节点的得分。此时估算出来的得分称为静态估值。

(4) 当叶子节点的估值计算出来后，再推算出父节点的得分，推算的方法是：对“或”节点，选其子节点中一个最大的得分作为父节点的得分，这是为了使在可供选择的方案中选一个对自己最有利的方案；对“与”节点，选其子节点中一个最小的得分作为父节点的得分，这是为了立足于最坏的情况。这样计算出的父节点的得分称为倒推值。

(5) 如果一个行动方案能获得较大的倒推值，则它就是当前最好的行动方案。

如果我们能建立博弈树，则我们可以通过最终的结果倒推到根节点以选择一步好棋。然而，除了极少数非常简单的棋类游戏（如井字棋），大多数棋类游戏（如国际象棋、象棋、围棋、日本将棋），我们都没有建立完整博弈树的可能。我们知道若博弈树的分枝数为 b（即一个节点所拥有的子女节点的个数），深度为 d（即博弈树的层数），则构建的博弈树的节点个数为 1+b+b2+b3+ ...+bd = $\frac{b^{d+1}-b}{b-1}$ 。随着深度的增

加，总节点数将呈指数爆炸增长，使得计算机在时间及空间上都无法对其进行处理。据统计，国际象棋完整的博弈树约有 10123 个节点，中国象棋有 10150 个节点，日本将棋有个 10226 个节点为，中国的围棋有 10400 个节点，对这些节点的搜索所花费的时间都远远超过了人们的可接受范围，所以在实际应用的时候，人们所构建的都是有限深度的博弈树，通过对有限深度的博弈树搜索，来得到相对较好的下一步法。一般来讲，所构建的博弈树的深度越大，所得到的下一步的走法越好。

极大极小值的算法伪代码如下：

```
function minimax(node, depth)
    if node is a terminal node or depth = 0
        return the heuristic value of node
    if the adversary is to play at node
        let α := +∞
        foreach child of node
            α := min(α, minimax(child, depth-1))
    else { we are to play at node }
        let α := -∞
        foreach child of node
            α := max(α, minimax(child, depth-1))
    return α
```

在真正应用的时候，直接使用极大极小值算法的比较少，一般都是使用它的一种变体——负极大值算法。其基本原理是基于下面的公式：

max (a,b) = - min (- a, - b)

即在几个节点中选择得分最小的节点相当于将这些节点的得分乘以-1 然后取得分最大的节点。这样负极大值算法就将每一步递归过程都统一了起来，每一次递归都选取最大值。

（下面是负极大值算法的伪代码：

```
function negamax(node, depth)
if node is a terminal node or depth = 0
  return the heuristic value of node
let best := -∞
foreach child of node
```

```
        best := max(α, -negamax(child, depth-1))
    return best
```

极大极小值算法是一种很好的博弈树搜索算法，但它本身并不完美，搜索时构建的极大极小博弈树中有许多冗余的节点，对于冗余节点的搜索会花费不少的时间。为了解决冗余节点问题，人们又研究出了α－β 剪枝算法[1-24]

2.1.3α-β剪枝算法

α－β 剪枝算法是对极大极小值算法的一种优化算法。极大极小值算法是所有博弈树搜索算法的基础，α－β 剪枝算法则是所有剪枝搜索算法的基础。此算法是由匹兹堡大学的三位科学家于 1958 年提出的，α－β 剪枝算法大幅度的降低了原算法的时间复杂度。这是一个划时代的发现，让博弈 AI 的实现变得现实。

α－β 剪枝算法由两部分组成—α 剪枝和 β 剪枝，α 剪枝和 β 剪枝的区别在于所分析的三层中极大值层极小值的层先后顺序不同，α 剪枝用来处理所分析的三层中第一层和第三层为极大值层的情况，β 剪枝用来处理所分析的三层中第一层和第三层为极小值层的情况。下面以图 2-6 的子树来说明α 剪枝和 β 剪枝的具体过程。

图 2-7 是图 2-6 的一个分枝，用来说明什么是α 剪枝。

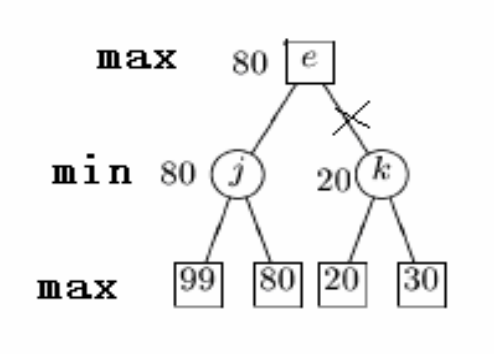

图 2-7 可进行α 剪枝的博弈树

用极大极小值算法确定 j 的值为 80 之后，为了确定 k 的值，需要从左往右搜索 k 的子节点，在搜索到 20 时，因为 k 属于 min 层，k 要取它的所有子节点中最小的一个，所以可以断定 k 的值一定小于等于 20；而节点 e 处于 max 层，e 要取它的所有子节点中最大的一个，j 为 80，所以 e 的值肯定大于等于 80，k 小于等于 20，所以 e 的值为 80。在确定 e 节点的值时，只搜索了 j 的子节点和 k 的第一个子节点，k 的其他子节点及其他子节点的子孙节点都不用再搜索了，这就是α 剪枝。

下面以图 2-6 的另一个子树来说明什么是 β 剪枝。

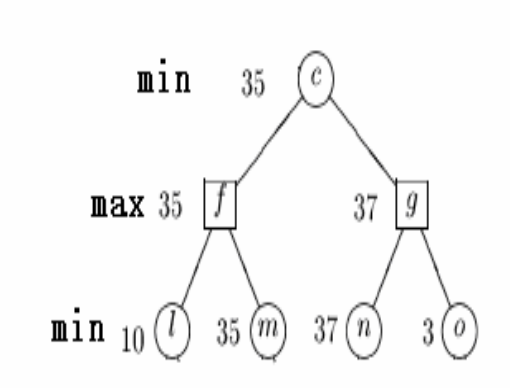

图 2-8 可进行 β 剪枝的博弈树

用极大极小值算法确定 f 的值为 35 之后，为了确定 g 的值，需要从左往右搜索 g 的子节点，在搜索到 37 时，因为 g 属于 max 层，g 要取它的所有子节点中最大的一个，所以可以断定 g 的值一定大于等于 37；而节点 c 处于 min 层，c 要取它的所有子节点中最小的一个，f 为 35，所以 c 的值肯定小于等于 35，而 g 大于等于 37，所以 c 的值为 35。在确定 c 节点的值时，只搜索了 f 的子节点和 g 的第一个子节点，g 的其他子节点及其他子节点的子孙节点都不用再搜索了，这就是 β 剪枝。

具体的剪枝方法如下：

1. 对于一个与节点 MIN，若能估计出其倒推值的上确界 β，并且这个 β 值不大于 MIN 的父节点(一定是或节点)的估计倒推值的下确界 α，即 α>=β，则就不必再扩展该 MIN 节点的其余子节点了(因为这些节点的估值对 MIN 父节点的倒推值已无任何影响了)，这一过程称为 α 剪枝。

2. 对于一个或节点 MAX，若能估计出其倒推值的下确界 α，并且这个 α 值不小于 MAX 的父节点(一定是与节点)的估计倒推值的上确界 β，即 α>=β，则就不必再扩展该 MAX 节点的其余子节点了(因为这些节点的估值对 MAX 父节点的倒推值已无任何影响了)。这一过程称为 β 剪枝。

从算法中看到：

1. MAX 节点(包括起始节点)的 α 值永不减少；

2. MIN 节点(包括起始节点)的 β 值永不增加。

在搜索期间，α 和 β 值的计算如下：

1. 一个 MAX 节点的 α 值等于其后继节点当前最大的最终倒推值。

2. 一个 MIN 节点的 β 值等于其后继节点当前最小的最终倒推值。

α－β 剪枝算法的伪代码如下：

```
function alphabeta(node, depth, α, β, Player)
    if  depth = 0 or node is a terminal node
        return the heuristic value of node
    if  Player = MaxPlayer
        for each child of node
            α   :=  max(α,  alphabeta(child,  depth-1,  α,  β,
not(Player) ))
            if β ≤ α
                break                          (* Beta cut-off *)
        return α
    else
        for each child of node
            β   :=  min(β,  alphabeta(child,  depth-1,  α,  β,
not(Player) ))
            if β ≤ α
                break                          (* Alpha cut-off *)
        return β
(* Initial call *)
alphabeta(origin, depth, -infinity, +infinity, MaxPlayer)
```

α－β剪枝算法能够让我们省略许多节点的搜索，不只是节点本身，而且还包括节点下面的子树，这样α－β剪枝算法需要搜索的节点数和极大极小值算法相比就少了很多，也就节省了大量的时间开销，并且仍不失为穷尽搜索的本性。但α－β剪枝算法的剪枝效果即搜索节点数与博弈树节点的搜索次序密切相关，从下面两个剪枝示例可以看出。

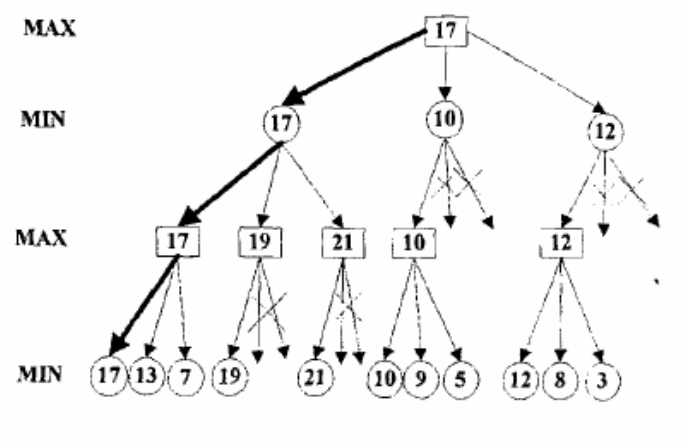

图 2-9 α－β 理想博弈树

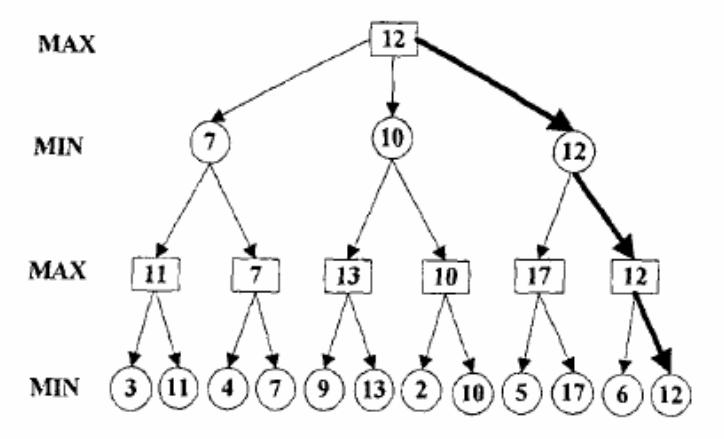

图 2-10 α－β 最差博弈树

图 2-9 和图 2-10 分别显示了搜索深度为 3 时，α－β 剪枝算法的最理想情况和最差情况。1975 年 Knuth 和 Moore 给出了在最理想的情况下，α－β 剪枝算法的节点数目计算公式:

ND=$2B^{D/2}-1$　　(D 为偶数)

ND=$B^{(D+1)/2}+B^{(D-1)/2}-1$　　(D 为奇数)

而在 α－β 剪枝算法最差的情况下，需要搜索的节点约为数为:

ND=B^{D}

其中 D 为搜索深度，B 为分枝因子。由上面的式子可以看出，最理想情况下 α－β 剪枝算法搜索深度为 D 需搜索的节点数相当于最差情况下需搜索的节点数的平方根的两倍左右，也可以说，在相同的时间内，使用 α－β 剪枝算法搜索的深度是不使用剪枝情况下的 2 倍。计算图 2-9 中需要搜索的节点数为 ND=$3^{(3+1)/2}+3^{(3-1)/2}-1=11$，而在图 2-10 这种情况下，α－β 剪枝算法搜索就相当于极大极小值搜索，显然 α－β 剪

枝算法就没有什么优势了。由于α－β剪枝算法与节点的排列顺序高度相关，寻找有效手段将候选着法排列调整为剪枝效率更高的顺序就显的非常重要了[1-24]。本文使用机器学习的知识改变博弈树分枝的排列顺序，从而提高α－β剪枝算法剪枝效率的根据也在于此。

2.2 已有的α－β剪枝算法优化方式

对α－β剪枝算法优化一般来说主要从四个方面入手：

- 编程方面的优化—虽然不能提高剪枝效率，但可以使编程简单。
- 利用节点有序进行优化—α－β剪枝算法的剪枝效率与节点的排列顺序有很大关系。
- 利用（α，β）构成的窗口进行优化—窗口越小，剪枝的效率越高。
- 利用启发信息进行优化—利用博弈问题本身的特点来提高剪枝的效率，通常与节点有序配合使用。

2.2.1 编程方面的优化

在编程时，真正使用标准的α－β剪枝算法的较少，用的较多是它的一种变体，一种把负极大值算法与α－β剪枝算法相结合产生的一种变体。这么做并不能提高算法的剪枝的效率，但在编程时会容易一些[31]。

其伪代码如下：

```
alphabeta(depth, alpha, beta){
  if (depth <= 0||棋局结束)
    return evaluation();//叶子节点或棋局结束，返回估值
  GenerateLegalMoves();//产生所有合法着法
  for(当前局面所有可能的着法 m){
    执行着法 m;
    val = - alphabeta(depth - 1, - beta, - alpha);
    撤消着法 m;
    if (val >= beta)
      return beta; //beta 剪枝
    if (val>alpha)
      alpha = val; //保存极大值
```

}

2.2.2 历史启发

历史启发是基于这样的思想：在搜索过程中，一些以前经过搜索认为最佳的走法，在相差不大的局面下，仍然有很大可能成为最优的走法。而且，某个走法被证明是最佳的次数越多，成为当前局面最佳的可能性也就越大。历史启发实际上就是记录搜索过的好的走法，并在后续着法中优先搜索。在搜索过程中，每找到一个好的着法，就将与该走法相应的历史得分作一个增量，一个多次被搜索并确认为好的走法的历史记录分值会较高，当搜索中间节点时，将走法根据其历史得分排序，以获得较佳的排列顺序[32]。

2.2.3 杀手启发

杀手走法是每一层中剪枝效率最高的走法。对于搜索树的特定层数上的一个中间节点来说，如果它的某个走法引发了剪枝，那么这个走法也有很大的可能性引发其兄弟节点剪枝。所以，如果随时记录下某一层数上引发剪枝的走法，在展开兄弟节点的子节点时把这个走法赋予较高的优先级，会对搜索效率的提高有很大帮助[32]。

2.2.4 空着搜索

空着搜索也是搜索算法中一种很有效的搜索策略，它的思想是当前棋局状态下放弃本方的走棋权利，让对方连续走棋，如果对方连续走棋后还导致剪枝，那么几乎可以肯定在当前棋局状态下，当本方走棋后会导致剪枝。空着搜索明显降低了搜索的数量，导致搜索深度显著提高，并且危险性比较小，实现较为简单。现已被几乎所有的博弈程序（除一些空着搜索没有意义的棋类）所采用。

值得注意的是，空着裁剪在某些棋局上是不能正常运行的，这是因为空着搜索是建立在“假定走一步棋会比不走棋有更高的分值”的基础上的，这个假定在很多局面上并不成立，这些局面称为无等着局面。无等着局面指的是不走棋反而局势更好些。空着载剪应用到这些局面上会产生副作用，因此提出了带验证的空着裁剪[32]。

2.2.5 置换表法

在博弈树中，不少节点之间虽然经过不同的路径到达，但其状态是完全一致的。通过建立置换表，保存已搜索节点的信息，那么再次遇到相同状态的节点时便可套用

之前的搜索结果，避免重重搜索。

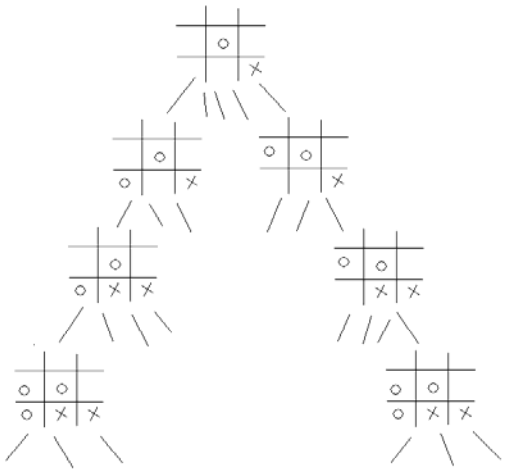

图 2-11 井字棋中出现相同节点的博弈树

如图 2-11 所示，经过不同的路径，在第四层出现两个相同的状态，如果第四层的前一个状态已经确定，那么后一个状态就不用再搜索下去了，直接把前一个状态的结果拷贝到后一个状态，然后把后一个状态的所有分枝剪去。

置换表的使用是一种空间换取时间的思想，如果在置换表中能直接得到结果的话，则可以避免该节点以及该节点为根的子树的搜索，从而减少搜索时间。同时如果在置换表中能查找到当前节点的信息，并且存储深度比当前节点将要搜索的深度大，那实际上是增加了当前子树的搜索深度，即增加了结果的准确性。正是因为置换表的诸多优点，所以置换表已成为博弈树搜索中广为采用的技术[27] 。

2.2.6 迭代加深

在搜索深度为 d-1 层时得到的最佳路径，最有可能成为搜索深度为 d 层的博弈树的最佳路径，Knuth 和 Moore 分析表明，对于分枝因子为 B 的博弈树，利用 α－β 剪枝算法搜索 d-1 层所需时间是搜索 d 层所需时间的$\sqrt{B}$倍。迭代深化搜索算法的思想是先利用$1/\sqrt{B}$的时间做 d-1 层的搜索，找到最佳路径，同时记下有价值的信息，然后再对 d 层进行剪枝搜索。

另外，逐层加深搜索算法比固定深度搜索算法更适合于对弈过程的时间控制。该方法通过逐渐加深搜索深度来重复调用 α－β 搜索，直至达到所要求的搜索深度或者搜索时间超过限度为止。这是一个去粗求精的过程。多数情况下，浅层最佳走棋，往往也是深层的最佳走棋，我们可以把浅层搜索的结果，当作深层搜索的第一子节点，

来提高搜索速度[32]。

2.2.7 negscout 算法

negascout 算法也称为 PVS(principal variation search)算法，也属α－β 算法的一种变体。它基于这样的假定:同层节点的排列高度有序，先搜索的节点优于后搜索的节点。一般来说，它在第一个子节点以一个完整窗口(α， β)进行搜索，得到值 value，后继的节点则以极小的窗口(value, value+l)进行搜索，以期高效率地剪枝。如果得到的值大于 value+l，则再以(value+l, β)进行搜索，如果小于 value 则可以直接剪枝。很多实验证明此算法剪枝效率优于α－β 算法[37]。

2.2.8 其他优化算法

除了以上介绍的优化算法外还有很多其他的优化算法，例如 fail-soft 算法[37]，渴望搜索算法[38],mtdf 算法[37]等。但总的来说，这些优化算法都没有超出上面提到的四大类。

2.3 本文所使用的α－β剪枝算法优化方式

由前面的介绍可知，通过改变博弈树节点的排列顺序来构建理想的博弈树，可以有效的提高α－β剪枝算法剪枝的效率，对理想博弈树采用α－β剪枝算法搜索深度为 D 需搜索的节点数相当于最差情况下需搜索的节点数的平方根的两倍左右，也可以说，在相同的时间内，使用α－β剪枝算法搜索的深度是不使用剪枝情况下的 2 倍。本文所介绍的算法的基本思想是，利用机器学习的一些知识来对博弈树的节点进行排序，通过构建理想博弈树来提高α－β剪枝算法的剪枝效率。由于机器学习算法都具有不精确性，所以构建理想博弈树比较困难，但即使不能构建理想博弈树，只要能让最佳路径尽量排在前几位，也能够提高α－β剪枝算法的剪枝效率。这就是本文能够用具有不精确特性的机器学习算法来辅助α－β剪枝算法进行剪枝的根据。

本文所提出的α－β剪枝算法优化方式的具体实现步骤如下：

1. 针对具体的博弈问题提取训练样例。

2. 选择一个适当的模型，利用训练样例训练一个可以对某一棋局状态的子节点进行排序的学习系统。

3. 把学习系统和α－β剪枝算法结合，在进行搜索之前，先利用训练好的学习系统对当前棋局状态的子节点进行排序，然后再进行搜索剪枝，从而提高算法的剪枝效

率。

2.4 本章小结

本章列出了α－β 剪枝算法的由来，介绍了什么是α－β剪枝算法，给出了已有的对α－β 剪枝算法的优化方式，并给出了本文所使用的α－β 剪枝算法的优化方法。

第 3 章 如何利用机器学习算法实现对节点排序

用于辅助α－β 剪枝算法的学习系统的设计思路与普通的用于分类的学习系统的设计思路一样，不同点只在于最后需要把学习系统的多个输出按照当前棋局状态下的下一步最佳走法的可能性大小进行排序，可能性大的排在前面，在利用α－β 剪枝算法进行搜索时，先搜索成为最佳路径可能性最大的分枝，从而提高α－β 剪枝算法的剪枝效率。

3.1 训练样例的特征项的选择

在收集训练样例时，刚开始一般会尽可能多的选取特征项，把能用上的特征全用上，确定模型之后，再用特征选择算法去掉一些影响不大的特征项。对特征优项化的算法也不只一种，基本思路如下：确定模型后，先用未对特征项优化的训练样例训练选定的模型，然后用去掉一个特征项后的训练样例在选定的模型上训练，若去掉某个特征项后，与没有去掉时的泛化能力相差不大，就可以把那个特征项去掉。用此方法依次测试原有训练样例中的所有项，最终保留下来的特征项即为我们需要的特征项[41]。

另外一种方式是逐渐添加特征项，先由一个特征项开始，若新添加的特征项不能提高模型的泛化能力，则去掉该特征项。

3.2 训练样例的获取

在获取训练样例之前，我们面临的第一个问题是如何表示训练样例。由前面的分析可知，要求我们的学习系统要能够根据当前棋局状态给出下一步的最佳走法。也就是说我们的学习系统要做的事是把棋局的当前棋局状态作为输入，把当前棋局状态下的下一步最佳走法作为输出。由此我们不难想到，训练样例的格式应该为{当前棋局状态的数字表示，下一步的最佳路径}这样的一个数据对。

如何获得这样的数据对呢？利用当前棋局状态所具有的特征的估计值组成的向量作为当前棋局状态的数字表示，把固定深度α－β 剪枝算法搜索到的最佳路径作为下一步的最佳路径。深度越深，所能预测的下一步走法的准确度就越高，但获得训练样例所花费的时间就越长。至于固定深度的大小要根据博弈问题本身的特点、项目的时间限定、资金投入等因素来综合考虑。经验表明，浅层搜索的结果往往也是深层搜索的结果，所以即使学习系统是由浅层搜索得到的训练样例训练得到的，也可以把它

用于辅助深层搜索的剪枝。

如何确定所需训练样例的数量？训练样例的数量与想要达到的泛化能力及所选取的目标函数等有关，具体的求法在机器学习中有系统的理论介绍。

目前，多数机器学习理论都是依赖于训练样例与测试样例分布一致这一假设。这就要求所选择的训练样例的分布与实际分布具有一致性，所以选择训练样例时，要根据具体的博弈问题慎重的选择。

3.3 模型选择

由前面对问题的分析可知我们所要得到的学习系统属于典型的监督学习。根据我们的训练样例的形式，可以很明显的看出我们的问题属于分类问题，即把当前棋局状态作为待分类项，以当前棋局状态下所有的下一步可能走法作为可以分为的类，而把当前棋局状态的最佳走法作为当前棋局状态所从属的类。根据分类问题与回归问题的关系，即分类问题是特殊的回归问题，也可以把我们的问题归为回归问题。为了达到分类的目的，需要把学习系统最终输出的实数值，用设定的阈值分成几个实数区间，一个实数区间代表一类，从而达到分类的目的。

无论是分类问题还是回归问题，都有很多成熟的模型可供选择，像神经网络、SVN、logistic 回归等。具体如何选择，也有不少成熟的算法，如简单交叉验证（hold-out cross validation）、k-折叠交叉验证（k-fold cross validation）等，但基本思路都差不多。基本思路如下：给定一个训练样例集，然后让各个模型在训练样例集上训练，得到几个假设，然后计算这些假设的经验风险（Empirical risk），经验风险较小的那个即为理想的模型。至于如何分割训练样例，降低方差，对于不同的算法会有略微的不同。

3.4 过度拟合与欠拟合

过度拟合是这样一种现象：一个假设在训练数据上能够获得比其他假设更好的拟合，但是在训练数据外的数据集上却不能很好的拟合数据。此时我们就叫这个假设出现了过度拟合的现象。比较标准的定义为：给定一个假设空间 H，一个假设 h 属于 H，如果存在其他的假设 h’属于 H,使得在训练样例上 h 的错误率比 h’小，但在整个实例分布上 h’比 h 的错误率小，那么就说假设 h 过度拟合训练数据。

过度拟合会导致泛化能力降低，如何解决过度拟合问题，对于不同的模型产生的原因和具体解决办法都有所不同，需要具体问题具体分析。

由于模型过于简单而导致的预测误差变大就叫做欠拟合。欠拟合也和过度拟合一样需要根据不同的模型采取不同的处理方法。

3.5 本章小结

这一章论述了构建能够用于辅助α－β剪枝算法的学习系统的通用过程及注意事项。给出了用于辅助α－β剪枝算法的学习系统所属的类型及建模方法，具体构建时的有关样本选取、模型选取及拟合问题的处理方法。

第 4 章 用于辅助 α－β 剪枝算法的学习系统的实现

井字棋的规则简单，取样、建模、训练、验证费时较少，但也能够说明构建用于辅助 α－β 剪枝算法的学习系统的整个过程。因此，本文以井字棋为例，详细的说明了构建用于辅助 α－β 剪枝算法的学习系统的具体做法。

4.1 井字棋的实现

井字棋的实现主要由四部分构成：

- 状态表示
- 确定目标状态
- 静态评估函数的选取
- 候选走法产生

下面从这四个方面来说明一下井字棋的具体实现。

4.1.1 状态表示

井字棋的棋盘由九个位置，实现时用一个 3*3 的二维数组（int cur[3][3];）来表示这九个位置。cur[0][0]表示位置 1，cur[0][1]表示位置 2，cur[0][2]表示位置 3，cur[1][0]表示位置 4，后面依此类推。用-1 表示人类棋手的棋子，以 0 表示没有棋子，以 1 表示计算机方的棋子。例如，若 cur[2][1]=-1,则表示在位置 8 有人类棋手的棋子。

利用上面的设计就可以把整个棋局状态在计算机中表示出来了。如图 4-1（圆圈表示计算机方），它在计算机中的表示为 cur[3][3]={1，0，0，0，1，0，0，-1，-1}。

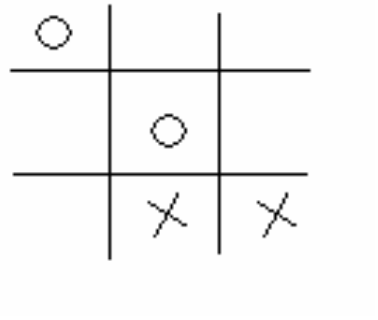

图 4-1 棋局状态

4.1.2 确定目标状态

确定目标状态即如何表示输、赢和平局。这个在编程时也不难实现，只要在水平、竖直、斜向有三个 1 出现，即为计算机赢，若有三个-1 出现，即为人类棋手赢，若棋盘已满仍未分出胜负，即为平局。下面的代码用于计算输、赢和平局三种情况。

```
int CheckWin(){//有人赢了吗？返回 0 表示没有人赢，返回-1 表示人赢了，返回//1 表示计算机赢
//判断在行方向是否有人赢
 for(int i=0;i<3;i++){
  if(cur[i][0]==1&&cur[i][1]==1&&cur[i][2]==1)return 1;
  if(cur[i][0]==-1&&cur[i][1]==-1&&cur[i][2]==-1)return -1;
 }
 //判断在列方向是否有人赢
 for(int i=0;i<3;i++){
  if(cur[0][i]==1&&cur[1][i]==1&&cur[2][i]==1)return 1;
  if(cur[0][i]==-1&&cur[1][i]==-1&&cur[2][i]==-1)return -1;
 }
 //判断斜向是否有人赢

if((cur[0][0]==1&&cur[1][1]==1&&cur[2][2]==1)||(cur[2][0]==1&&cur[1][1]==1&&cur[0][2]==1))
 return 1;

if((cur[0][0]==-1&&cur[1][1]==-1&&cur[2][2]==-1)||(cur[2][0]==-1&&cur[1][1]==-1&&cur[0][2]==-1))return -1;
  return 0;//没有赢的话则返回
 }
```

返回 0 表示没有人赢，返回-1 表示人赢了，返回 1 表示计算机赢了。

4.1.3 静态评估函数

静态评估函数用于计算某一棋局状态下的得分，不同的博弈类游戏所采用的评估函数不同。由于井字棋本身的简单性使得它的评估函数也不复杂。某一棋局状态下的得分由以下公式求得：

棋局状态得分 = 棋盘中所有未着子位置填上计算机的棋子后所得到的计算机的棋子三个连成一线的个数 - 棋盘中所有未着子位置填上人类棋手的棋子后所得到的人类棋手的棋子三个连成一线的个数。

下面以图 4-2 的棋局得分为例，说明一下棋局状态得分的计算方式（圆圈为计算机方）。

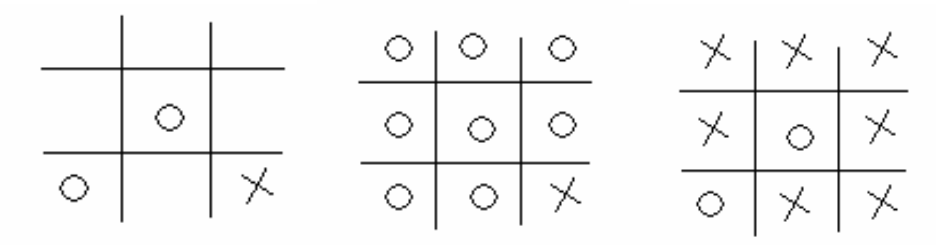

图 4-2 棋局状态评估

图 4-2 中的左图为待评估的棋局状态。把棋盘中的所有空闲位置都填上计算机方的棋子后得到图 4-2 中的中图，在中图中，计算机方的棋子连成三个的有 5 个；把棋盘中的所有空闲位置都填上人类棋手的棋子后得到图 4-2 中的右图，在右图中，人类棋手的棋子连成三个的有 2 个。由上面的公式可知，图 4-2 左图中的棋局状态得分为 5-2=3。

4.1.4 候选走法产生

候选走法产生即如何生成下一步的最佳走法，由于本文研究的是α－β 剪枝算法，所以候选走法产生使用的是α－β 剪枝算法。关于α－β 剪枝算法在前面已经做了具体介绍，所以在这里仅给出井字棋的α－β 剪枝算法的具体实现，不再做具体说明。

在我采用的算法中，可以通过增加生成树的层数，即增加搜索深度的值来提高计算机的智商。这相当于增加了计算机向前预测的步数。对井字棋来说，因为井字棋有 9 个格，所以搜索深度的最大值可以设为 9，但是实际上，经过试验，当搜索深度为 3 时，计算机对井字棋的落子的处理就能达到比较好的效果。

下面是井字棋剪枝部分的核心代码：

```
//主算法部分，实现α－β 剪枝的算法，val 为上一层的评价值，
//dep 为搜索深度，max 记录上一层是否为极大层
int cut(int &val,int dep,bool max)  {
    //如果搜索深度达到最大深度，或者深度加上当前棋子数已经达到，就直接
//调用评价函数
     if(dep==depth  || dep+num==9){
```

```
            return value();//value()为评估函数，用于评估当前棋局状态的得分
        }
        int i,j,flag,temp;
        bool out=false;                    //out 记录是否剪枝，初始为 false
        if(CheckWin()==1) {                //如果用户玩家输了，就置上一层的评
价//值为无穷（用很大的值代表无穷）
            val=10000;
            return 0;
        }
        if(max)                          //如果上一层是极大层，本层则需要是极小
    //层，记录 flag 为无穷大；反之，则为记录为负无穷大
            flag=10000;                  //flag 记录本层节点的极值
        else
            flag=-10000;
        for(i=0;i<3 && !out;i++)  {      //两重循环，搜索棋盘所有位置
               for(j=0;j<3 && !out;j++){
                if(cur[i][j]==0) {     //如果该位置上没有棋子
                    if(max) {    //并且为上一层为极大层，即本层为极小层,轮
到
    //用户玩家走了。
                        cur[i][j]=-1;       //该位置填上用户玩家棋子
                        if(CheckWin()==-1)     //如果用户玩家赢了
                            temp=-10000;     //置棋盘评价值为负无穷
                        else
                            temp=cut(flag,dep+1,!max);//否则继续调用α－β
剪枝//函数
                        if(temp<flag) //如果下一步棋盘的评价值小于本层节点
    //的极值，则置本层极值为更小者
                            flag=temp;
                        if(flag<=val)              //如果本层的极值已经小于
上一层的评价值，则不需要搜索下去，剪枝
```

```
                    out=true;
                } else{ //如果上一层为极小层，算法与上面刚好相反
                    cur[i][j]=1;
                    if(CheckWin()==1)
                        temp=10000;
                    else
                        temp=cut(flag,dep+1,!max);
                    if(temp>flag)
                        flag=temp;
                    if(flag>=val)
                        out=true;
                }
                cur[i][j]=0;   //把模拟下的一步棋还原，回溯
            }
        }
    }
    if(max)  { //根据上一层是否为极大层，用本层的极值修改上一层的评价值
      if(flag>val)
            val=flag;
    }else{
        if(flag<val)
            val=flag;
    }
    return flag;   //函数返回的是本层的极值
}
```

实现之后的井字棋运行效果如图 4-3 所示：

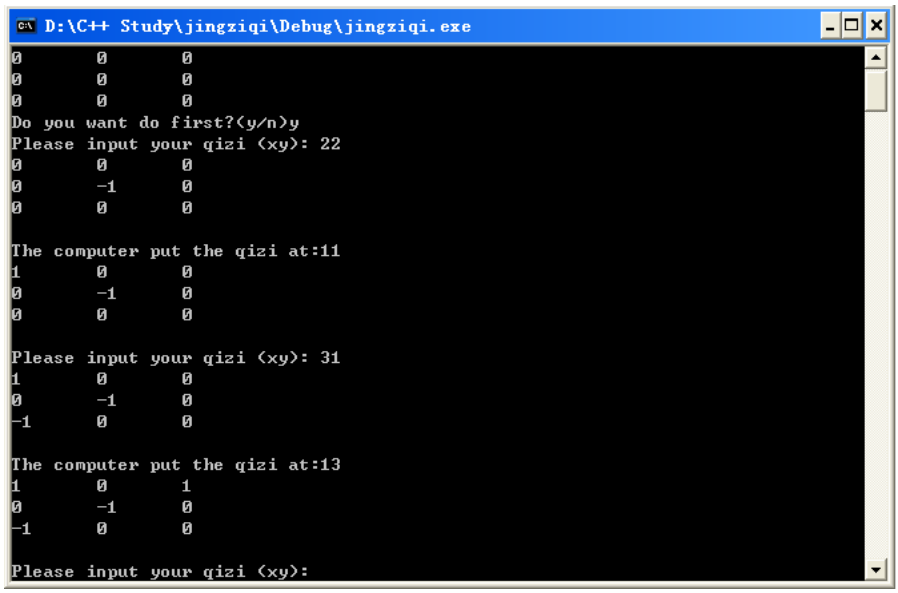

图 4-3 井字棋运行效果

输入数字表示要下的位置，11 表示位置 1，12 表示位置 2，13 表示位置 3，21 表示位置 4，后面的依此类推。由于本文的重点在于算法，所以并未对界面作太多美化。

本文所提到的有关象棋的部分，若对象棋的实现不了解的读者可以参看《中国象棋计算机博弈关键技术分析》—徐心和，王骄。那篇文章中比较详细的说明了有关象棋的状态表示，目标状态的确定，候选走法的产生及静态评估函数等方面的内容。

4.2 训练样例的特征项的选择

棋类游戏博弈问题训练样例的特征项的选择主要有两种方法。

第一种方法是从评估函数入手，凡是影响评估函数估值的内容都要作为特征项；以象棋为例，对于每一个棋子来说，可以从五个方面来影响棋局状态的估值——棋子本身的估值、棋子受威胁的估值、棋子受保护的估值、棋子的灵活度估值、棋子的位置附加值（兵过河之后的价值更大），可以把 32 个棋子的这些影响评估函数估值的 5 个方面作为训练样例的特征项。这种方法简单实用，但它存在一个缺点，即不具有通用性。对井字棋来说，这种方法就不适用了。井字棋的评估函数为：棋局状态得分 = 棋盘中所有未着子位置填上计算机的棋子后所得到的计算机的棋子三个连成一线的个数 - 棋盘中所有未着子位置填上人类棋手的棋子后所得到的人类棋手的棋子三个连成一线的个数；对于这种评估函数，想从它入手来表示棋局状态就比较困难了。

第二种是从棋子及其位置入手，把棋盘上的所有棋子及其位置作为训练样例的特征项。以井字棋为例，可以把数组结构本身所具有的位序表示棋盘上的各个位置，例如以一个 3*3 的二维数组（cur[3][3]）来表示井字棋的棋盘，cur[0][0]表示位置 1，cur[2][2]表示位置 9；以数组中存储的值来表示棋子的有无及种类，例如用 0 表示没有棋子，用 1 表示已方棋子，用-1 表示对方棋子。这种方法具有较强的通用性，本文即采用这种方法来选择特征项。

训练样例的特征项的选择方法仔细研究起来肯定不只这两种，只是本人以为这两种方法比较简单实用，更好的选择方法有待于后人继续努力。

4.3 训练样例的获取

由上文的讨论可知，训练样例可以用{位置 1 位置 2 … 位置 9，下一步的最佳走法}这种方式表示。用 1 表示棋盘上计算机方的棋子，用-1 表示另一方的棋子，0 表示没有棋子，用 1-9 表示下一步的最佳走法。 例如，当前棋局如图 4-4 所示，叉方表示计算机方，下一步该叉方走，下一步的最佳走法为位置 4，则表示成训练样例为{-1，0，1，0，-1，0，-1，1，0，4}。

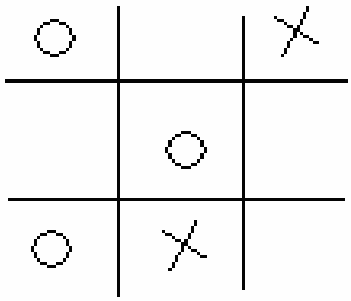

图 4-4 当前棋局状态

具体实现时用如下数据结构来存储一个训练样例：

```
//训练样例
typedef struct{
    int input[9];//用于存储棋盘上九个位置上的值
    int result;//用于存储下一步的最佳走法
} Exam;
```

为了操作及验证方便，本文自始至终都假设由人类棋手先走。为了训练学习系统，取的训练样例数为 100。为了使训练样本具有代表性，在这 100 个训练样例中，棋盘上有一子的情况取 5 个，三子的情况取 25 个，五子的情况取 40 个，七子的情况取

30。测试样本取 20 个，为了使训练样本与测试样本分布一致，其中 1 子的情况取 2 个，3 子的取 5 个，5 子的取 8 个，7 子的取 5 个。取的时候要保证测试样本和训练校本无重复。

由于这种训练样例需求量不是很大，而且获取比较简单（先记录下轮到计算机走时的当前棋局状态，然后再记录下使用了 α－β 剪枝算法搜索深度为 3 的井字棋游戏的下一步走法即可），所以可以编程实现训练样例的获取也可以手动获取。

若完全采用编程的方式来获取训练样例，编程的工作量有点大，所以最终采用的是编程和手工相结合的方式。

基本思路是把编写完成的井字棋代码中嵌入能够把当前棋局状态和下一步走法输出到文本文件中的代码，通过和计算机下棋来记录下一些棋局状态和下一步的最佳走法。在和计算机的对弈中记录下来的棋局状态和下一步最佳走法中会有很多重复的，最后用手工的方式从这些记录中挑选一些合适的样本。图 4-5 上图中是与计算机下棋时收集的样本，中图是挑选的用于训练的训练样本，下图是挑选的用于测试的测试样本。

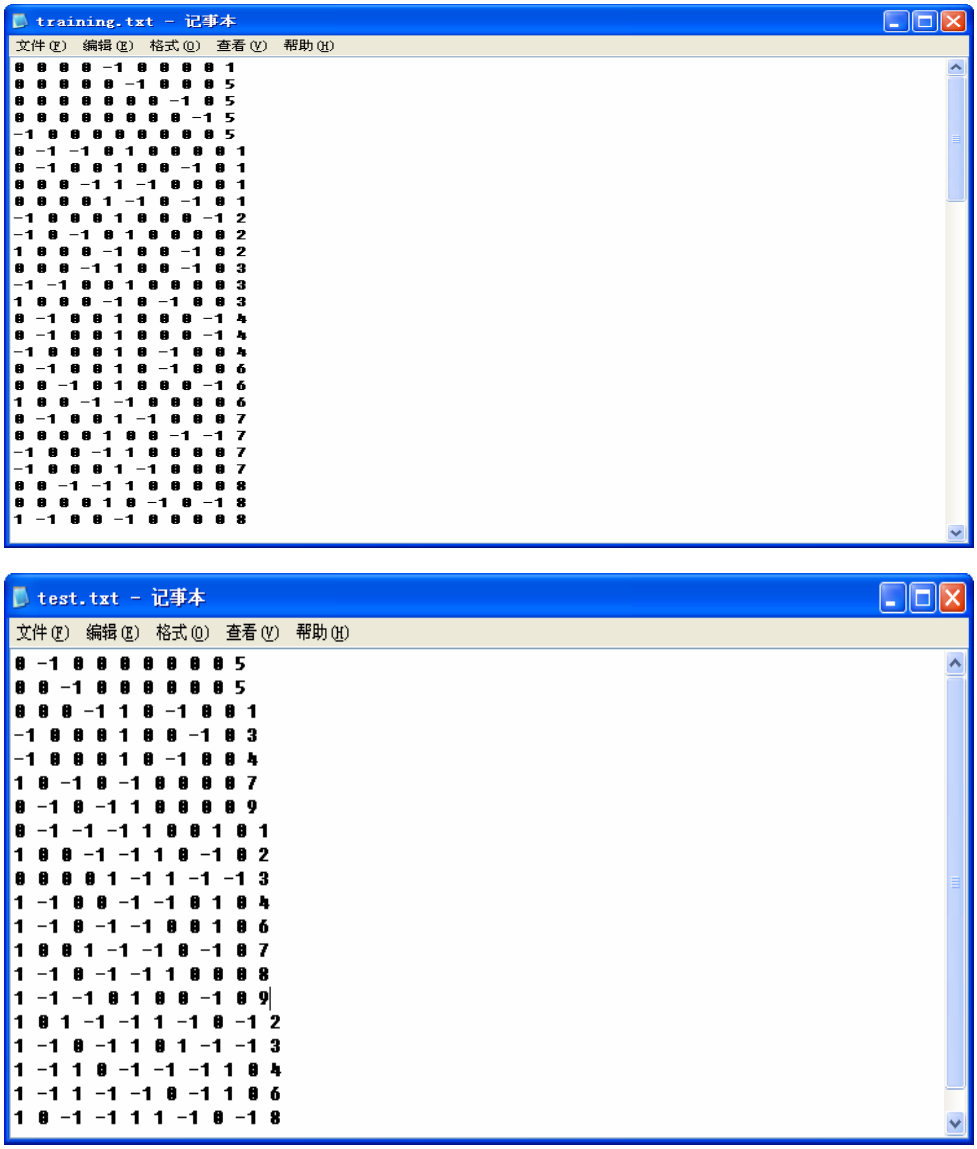

图 4-5 训练样例获取

上文在训练样例数量的确定上具有一定的不科学之处，训练样例数量的确定应由具体的公式推导所得，并且总的来说服从图 4-6 所示的规律。

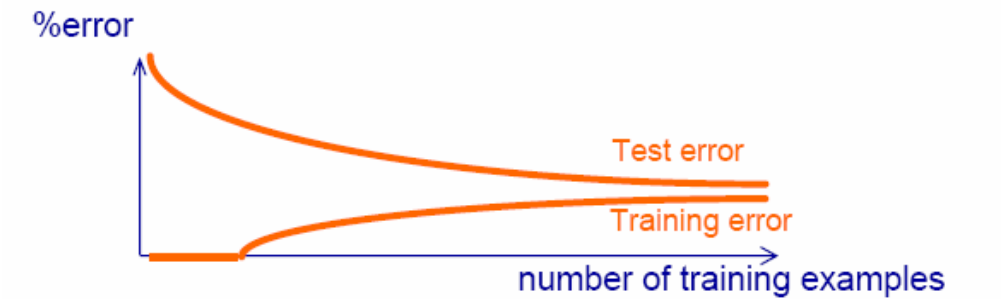

图 4-6 样本数量、测试误差、训练误差关系图

在训练样例数量的确定上本文更多的侧重与经验和训练样例的代表性上。虽然不科学，但并不影响我们目的的达成——验证学习系统具有辅助剪枝的功能，只要学习系统能够实现辅助剪枝，我们的目的就达到了，至于剪枝的效率最终能够提高多少并不是我们最关心的，所以训练样例数量的确定不太科学也没有太大关系。在模型选择、模型参数确定等方面没做太多优化的原因也在于此，后面遇到不精确之处不再做具体说明。

4.4 模型选择

对于井字棋来说，用于辅助 α－β 剪枝算法的学习系统的模型选择经过一系列的对比后，最终决定用 BP 神经网络。BP（Back Propagation）网络是一种按误差逆传播算法训练的多层前馈网络，是目前应用最广泛的神经网络之一。BP 网络能学习和存贮大量的输入-输出模式映射关系，而无需事前揭示描述这种映射关系的数学方程。它的学习规则是使用梯度下降法，通过反向传播来不断调整网络的权值和阈值，使网络的误差平方和最小。BP 网络具有较强的容错性，数学理论已经证明它具有实现任何复杂非线性映射的功能，使得它特别适合于求解内部机制复杂的问题，它还能通过学习带正确答案的实例集自动提取合理的求解规则，即具有自学习能力。BP 神经网络模型拓扑结构包括输入层（input）、隐藏层(hide layer)和输出层(output layer)，如图 4-7 所示。

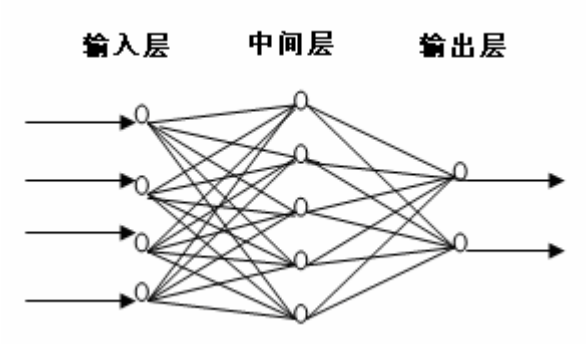

图 4-7BP 神经网络模型拓扑结构

4.4.1 BP 神经元

图 4-8 所示为一个基本神经元。其中 x_1、x_2…x_i…x_n 分别代表输入；w_{j1}、w_{j2}…w_{ji}…

w_{jn} 则分别表示神经元的连接强度，即权值；b_j 为阈值；$f(\cdot)$ 为传递函数；y_j 为神经元的输出。

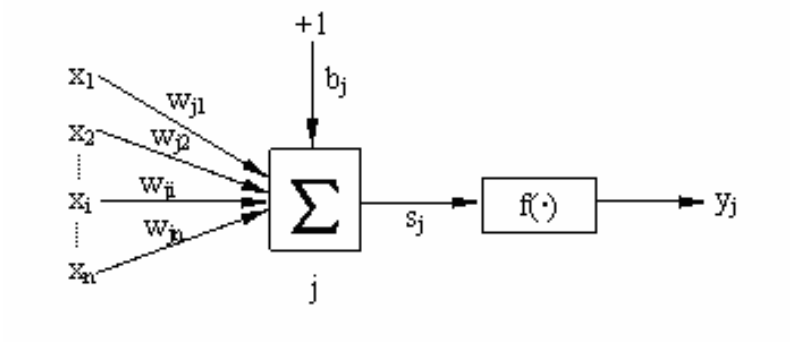

图 4-8 基本神经元

这个神经元的净输入值 S_j 为：

$$S_j = \sum_{i=1}^{n} w_{ji} x_i + b_j = W_j X + b_j \qquad (4.1)$$

其中：$X=[x_1\ x_2\ \cdots\ x_i\ \cdots\ x_n]^T$　$Wj=[w_{j1}\ w_{j2}\ \cdots\ w_{ji}\ \cdots\ w_{jn}]$

若视 $x_0=1, w_{j0}=b_j$, 即令 X 及 Wj 包括 x_0 及 w_{j0}，则

$X=[\ x_0\ x_1\ x_2\ \cdots\ x_i\ \cdots\ x_n]T$　　$Wj=[\ w_{j0}\ w_{j1}\ w_{j2}\ \cdots\ w_{ji}\ \cdots\ w_{jn}]$

于是节点 j 的净输入 S_j 可表示为：

$$S_j = \sum_{i=0}^{n} w_{ji} x_i = W_j X \qquad (4.2)$$

净输入 Sj 通过传递函数（Transfer Function）f（•)后，便得到第 j 个神经元的输出 y_j:

$$y_j = f(s_j) = f(\sum_{i=0}^{n} w_{ji} x_i) = F(W_j X) \qquad (4.3)$$

式中 f(•)是单调上升函数，而且必须是有界函数，因为细胞传递的信号不可能无限增加，必有一最大值。在本文中，我们选用的是 sigmoid 函数。

$$f(net) = \frac{1}{1+e^{-net}} \qquad (4.4)$$

对应的代码如下：

```
double fnet(double net) { //Sigmoid 函数,神经网络激活函数
    return 1/(1+exp(-net));
}
```

4.4.2 BP 网络

BP 算法由数据流的前向计算（正向传播）和误差信号的反向传播两个过程构成。正向传播时，传播方向为输入层→隐藏层→输出层，每层神经元的状态只影响下一层神经元。若在输出层得不到期望的输出，则转向误差信号的反向传播流程。通过这两个过程的交替进行，在权向量空间执行误差函数梯度下降策略，动态迭代搜索一组权向量，使网络误差函数达到最小值，从而完成信息提取和记忆过程[39-40]。

4.4.2.1 正向传播

图 4-9 是本文用于说明神经网络工作原理的三层 BP 神经网络拓扑结构。

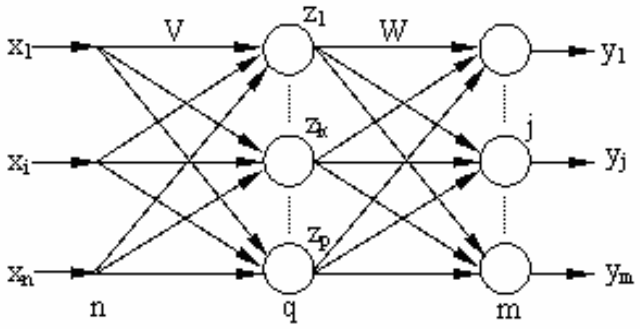

图 4-9 BP 神经网络拓扑结构

设 BP 网络的输入层有 n 个节点，隐藏层有 q 个节点，输出层有 m 个节点，输入层与隐藏层之间的权值为 v_{ki}，隐藏层与输出层之间的权值为 w_{jk}。隐藏层和输出层的传递函数为都为 sigmoid 函数，为了便于区分，隐藏层的传递函数用 $f_1(\cdot)$表示，输出层的传递函数用 $f_2(\cdot)$表示，则隐藏层节点的输出为（将阈值写入求和项中）：

$$z_k = f_1(\sum_{i=0}^{n} v_{ki} x_i) \qquad k=1,2,\cdots\cdots q \qquad (4.5)$$

输出层节点的输出为：

$$y_j = f_2(\sum_{i=0}^{q} w_{jk} z_k) \qquad j=1,2,\cdots\cdots m \qquad (4.6)$$

至此 B-P 网络就完成了 n 维空间向量对 m 维空间的近似映射。

4.4.2.2 反向传播

1)　　定义误差函数

输入 P 个学习样本，用 $x_1, x_2, \cdots, x_p$ 来表示。第 p 个样本输入到网络后得到输出 y_{jp}（j=1,2,…m）。采用平方型误差函数，于是得到第 p 个样本的误差 Ep：

$$E_p = \frac{1}{2}\sum_{j=1}^{m}(t_j^p - y_j^p)^2 \qquad (4.7)$$

式中：t_j^p 为期望输出。

对于 P 个样本，全局误差为：

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{j=1}^{m}(t_j^p - y_j^p)^2 = \sum_{p=1}^{P}E_p \qquad (4.8)$$

2）输出层权值的变化

采用累计误差 BP 算法调整 wjk，使全局误差 E 变小，即

$$\Delta w_{jk} = -\eta\frac{\partial E}{\partial w_{jk}} = -\eta\frac{\partial}{\partial w_{jk}}(\sum_{p=1}^{P}E_p) = \sum_{p=1}^{P}(-\eta\frac{\partial E_p}{\partial w_{jk}}) \qquad (4.9)$$

式中：η—学习率

由链定理得：

$$\frac{\partial E_p}{\partial w_{jk}} = \frac{\partial E_p}{\partial S_j}\frac{\partial S_j}{\partial w_{jk}} \qquad (4.10)$$

（4.10）式中的第一项再次使用链定理得：

$$\frac{\partial E_p}{\partial S_j} = \frac{\partial E_p}{\partial y_j}\frac{\partial y_j}{\partial S_j} \qquad (4.11)$$

（4.11）式中的第一项：

$$\frac{\partial E_p}{\partial y_j} = \frac{\partial}{\partial y_j}[\frac{1}{2}\sum_{j=1}^{m}(t_j^p - y_j^p)^2] = -(t_j^p - y_j^p) \qquad (4.12)$$

（4.11）式中的第二项：

$$\frac{\partial y_j}{\partial S_j} = f_2'(S_j) \qquad (4.13)$$

又因为 $f_2(\cdot)$ 为 sigmoid 函数所以有

$$\frac{\partial y_j}{\partial S_j} = f_2'(S_j) = \frac{\partial}{\partial S_j}(\frac{1}{1+e^{-S_j}}) = y_j(1-y_j) \qquad (4.14)$$

（4.10）式中的第二项：

$$\frac{\partial S_j}{\partial w_{jk}} = z_k \qquad (4.15)$$

于是：

$$\frac{\partial E_p}{\partial w_{jk}} = \frac{\partial E_p}{\partial S_j}\frac{\partial S_j}{\partial w_{jk}} = -(t_j^p - y_j^p)y_j(1-y_j)z_k \qquad (4.16)$$

于是输出层各神经元的权值调整公式为：

$$\Delta w_{jk} = -\eta\frac{\partial E}{\partial w_{jk}}$$

$$= -\eta\frac{\partial}{\partial w_{jk}}(\sum_{p=1}^{P} E_p)$$

$$= \sum_{p=1}^{P}\eta(t_j^p - y_j^p)y_j(1-y_j)z_k \qquad (4.17)$$

3）隐藏层权值的变化

$$\Delta v_{ki} = -\eta\frac{\partial E}{\partial v_{ki}} = -\eta\frac{\partial E}{\partial v_{ki}}(\sum_{p=1}^{P} E_p) = \sum_{p=1}^{P}(-\eta\frac{\partial E_p}{\partial v_{ki}}) \qquad (4.18)$$

由链定理得：

$$\frac{\partial E_p}{\partial v_{ki}} = \frac{\partial E_p}{\partial S_k}\frac{\partial S_k}{\partial v_{ki}} \qquad (4.19)$$

（4.19）式中的第一项再根据链式定理得：

$$\frac{\partial E_p}{\partial S_k} = \frac{\partial E_p}{\partial z_k}\frac{\partial z_k}{\partial S_k} \qquad (4.20)$$

（4.20）式中的第一项：

$$\frac{\partial E_p}{\partial z_k} = \frac{\partial}{\partial z_k}[\frac{1}{2}\sum_{j=1}^{m}(t_j^p - y_j^p)^2] = -\sum_{j=1}^{m}(t_j^p - y_j^p)\frac{\partial y_j}{\partial z_k} \qquad (4.21)$$

依链定理及（4.14）式得：

$$\frac{\partial y_j}{\partial z_k} = \frac{\partial y_j}{\partial S_j}\frac{\partial S_j}{\partial z_k} = f_2^{'}(S_j)w_{jk} = y_j(1-y_j)w_{jk} \qquad (4.22)$$

（4.20）式的第二项：

$$\frac{\partial z_k}{\partial S_k} = f_1^{'}(S_k) \qquad (4.23)$$

又因为 f1(•)为 sigmoid 函数所以有：

$$\frac{\partial z_k}{\partial S_k} = f_1^{'}(S_k) = \frac{\partial}{\partial S_j}(\frac{1}{1+e^{-S_k}}) = o_k(1-o_k) \quad (4.24)$$

其中 o_k 为隐藏层的输出。

（4.19）式的第二项为：

$$\frac{\partial S_k}{\partial v_{ki}} = x_i \quad (4.25)$$

于是：

$$\frac{\partial E_p}{\partial v_{ki}} = -\sum_{j=1}^{m}(t_j^p - y_j^p)y_j(1-y_j)w_{jk}o_k(1-o_k)x_i \quad (4.26)$$

从而得到隐藏层各神经元的权值调整公式为：

$$\Delta v_{ki} = \sum_{p=1}^{P}\sum_{j=1}^{m}\eta(t_j^p - y_j^p)y_j(1-y_j)w_{jk}o_k(1-o_k)x_i \quad (4.27)。$$

4.4.3 BP 神经网络的设计与实现

本文采用 C++语言来实现多层前馈 BP 网络，训练时使用前面所述的实测样本数据。

另外，目前尚未找到较好的网络构造方法。确定神经网络的结构和权系数来描述给定的映射或逼近一个未知的映射，只能通过学习方式得到满足要求的网络模型。神经网络的学习可以理解为：对确定的网络结构，寻找一组满足要求的权系数，使给定的误差函数最小。设计多层前馈网络时，主要侧重试验、探讨多种模型方案，在实验中改进，直到选取一个满意方案为止，可按下列步骤进行：对任何实际问题先都只选用一个隐藏层；使用很少的隐藏层节点数；不断增加隐藏层节点数，直到获得满意性能为止；否则再采用两个隐藏层重复上述过程。

训练过程实际上是根据目标值与网络输出值之间误差的大小反复调整权值和阈值，直到此误差达到预定值为止。

4.4.3.1 确定 BP 网络的结构

确定了网络层数、每层节点数、传递函数、初始权系数、学习算法等也就确定了 BP 网络。确定这些选项时有一定的指导原则，但更多的是靠经验和试凑。

1）隐藏层数的确定：

1998 年 Robert Hecht-Nielson 证明了对任何在闭区间内的连续函数，都可以用

一个隐藏层的 BP 网络来逼近，因而一个三层的 BP 网络可以完成任意的 n 维到 m 维的映照。因此本文用的是含有一个隐藏层的网络。

2） BP 网络常用传递函数:

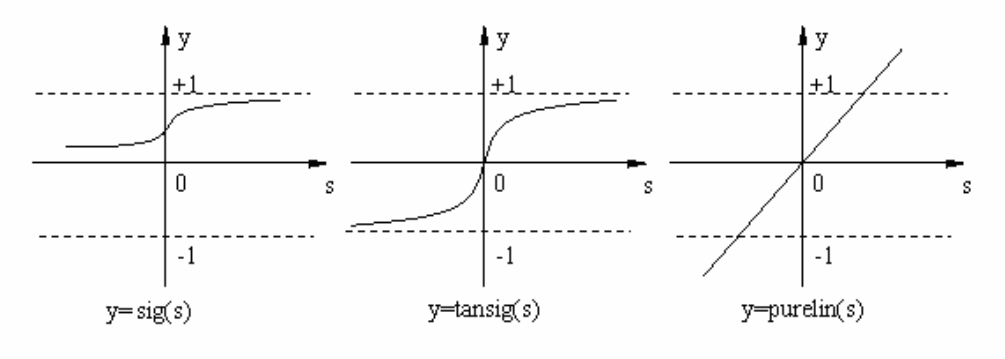

图 4-10 常用的传递函数

BP 网络的传递函数有多种。sigmoid 型函数的输入值可取任意值，输出值在 0 和 1 之间；tan-sigmoid 型传递函数的输入值可取任意值，输出值在-1 到+1 之间；线性传递函数 pure-line 的输入与输出值可取任意值。各种传递函数如图 4-10 所示。本文采用的是 sigmoid 型传递函数。

3） 每层节点数的确定：

使用神经网络的目的是实现根据井字棋的当前棋局状态来决定下一步的最佳走法，BP 网络的输入层是当前棋局状态下棋盘上九个位置上的棋子值，所以输入层的节点数为 9，输出层给出的是下一步的最佳走法，共有九个位置可能是最佳走法，所以输出层的节点个数也是 9。下面主要介绍隐藏层节点数量的确定。

对于多层前馈网络来说，隐藏层节点数的确定是成败的关键。若数量太少，则网络所能获取的用以解决问题的信息太少；若数量太多，不仅增加训练时间，更重要的是隐藏层节点过多还可能出现所谓过度拟合问题，因此合理选择隐藏层节点个数非常重要。关于隐藏层节点个数的选择比较复杂，一般原则是：在能正确反映输入输出关系的基础上，应选用较少的隐藏层节点数，以使网络结构尽量简单。本论文中采用网络结构增长型方法，即先设置较少的节点数，对网络进行训练，并测试学习误差，然后逐渐增加节点数，直到学习误差不再有明显减少为止。

具体操作时，一般是根据一些经验公式来大致确定隐藏节点的个数范围，然后再逐一测试这一范围内的隐藏节点数，直到找到一个最佳的为止。确定隐藏层节点个数的经验公式不只一种，本文采用的是如下的经验公式：

$$n_1 = \sqrt{n+m} + a \qquad (4.28)$$

式中：n 为输入节点个数，m 为输出节点个数，a 为 1 到 10 之间的常数。针对本论文隐藏层节点个数的取值范围为 5～15。

4）误差的选取

在神经网络训练过程中选择全局误差较为合理，原因如下：

标准 BP 算法中，误差定义为：

$$E_p = \frac{1}{2}\sum_{j=1}^{m}(t_j^p - y_j^p)^2 \qquad (4.29)$$

每个样本训练网络时，都对权矩阵进行了一次修改。由于每次权矩阵的修改都没有考虑权值修改后对其他样本的输出误差的影响，因此将导致迭代次数增加。

累计误差 BP 算法的全局误差定义为：

$$E = \frac{1}{2}\sum_{p=1}^{P}\sum_{j=1}^{m}(t_j^p - y_j^p)^2 = \sum_{p=1}^{P}E_p \qquad (4.30)$$

这种算法是为了减小整个训练集的全局误差，而不针对某一特定样例，克服了上一种误差表示方法的缺点，本文采用的是这种误差表示方式。

5）网络的初始化

网络的初始化包括对权矩阵、学习速率、学习精度、最大训练次数的初始化。

对于权矩阵的初始化比较简单，只要给一个小的随机数即可。具体实现如下：

```
srand((unsigned)time(NULL));
for (i = 0; i < IN_COUT; i++)
    for (j = 0; j < (*bp).h; j++)
        (*bp).v[i][j] = rand() / (double)(RAND_MAX);    //隐藏层权值
for (i = 0; i < (*bp).h; i++)
    for (j = 0; j < OUT_COUT; j++)
        (*bp).w[i][j] = rand() / (double)(RAND_MAX);  //输出层权值
```

学习速率、学习精度、最大训练次数的初始化相对来说更复杂一些。在具体实现时，它们的确定和节点个数的确定一起实现。具体代码如下：

```
for(int i=5;i<15;i++){//节点数
    for(int k=1;k<20;k+=2) {//精度控制
        for(int l=10000;l<100000;l=l+1000){
            AutoInitBp(&bp,i,0.01,0.001*k,l);     //初始化 bp 网络
```

结构

```
                    AutoTrainBp(&bp, x, y);                    //训练 bp 神经网络
                    nihe(&bp);
                    fanhua(&bp);
                }
            }
        }
```

函数 AutoInitBp(&bp,i,0.01,0.001*k,l);第三个参数为学习速率，经过实验发现，学习速率取 0.01 比较合适，再小则收敛速度就太慢了。

执行以上代码虽然能找到较好的结果，但所花费的时间太长，可以使用一些技巧性的方法。开始的时候使每层循环的增加量比较大，先得到一个较大区间，然后再对这个区间进行较细致的训练。例如刚开始时 i 每加 2，k 每次加 4，l 每次加 1000。通过上面的训练之后，发现当 i=14，k=1,l=100000 时泛化能力最好，重新设置 i,k,l 的值，使得 i 的范围为 13-15，每次加 1，k 的范围为 1-5，每次加 1，l 的范围为 10000-200000，每次加 100。然后再在这个小范围内找泛化能力最好的。

通过以上方式可以解决隐藏层节点个数的确定、学习速率、学习精度、最大训练次数的初始化问题和过度拟合问题。

6）训练样例的读取

训练样本存储在名为 training.txt 的文本文件中，一个训练样本为一行，由 9 个输入和一个输出构成。如下图 4-9 所示。

图 4-11 训练样本

读取的样本存储在用于存储训练样本的结构体数组 sample 中。结构体的定义如下：

```
//训练样本
typedef struct{
    int input[9];
    int result;
} Exam;
Exam sample[100],test[20];
```

具体实现代码如下：

```
//获取训练样本数据
void get_tr_ex(){
    //把 100 个训练集从文本中提取到训练样本数组中
    ifstream fin("d:\\training.txt");
    int tem=0;
    int i=0;
    int j=-1;
    while(fin>>tem)   {
        int temp=i%10;
        if(temp==0)
            j++;
        if(temp!=9)
            sample[j].input[temp]=tem;
        else
            sample[j].result=tem;
        i++;
    }
    fin.close();
}
```

测试样本的读取与此相似，训练样本存储在结构体数组 test 中。

7）输出结果的表示

网络的输出层由 9 个 sigmoid 单元构成，9 个输出单元的排列位置值即为这个输出单元所表示的棋盘位置值。9 个输出节点输出值最大的那个，即为下一步的最佳走法。

训练样例中的输出结果为 1-9 的数字，为了便于训练，需要把训练样例中的输出结果转化为适用于神经网络的输出结果。具体转化方式如代码所示：

```
int y[COUT][OUT_COUT];            //适用于神经网络的输出
    int temp;
    for(int i=0;i<COUT;i++)
    {
        for(int j=0;j<OUT_COUT;j++){
            temp=j+1;
            if(temp==sample[i].result)
                y[i][j]=1;
            else
                y[i][j]=0;
        }
    }
```

二维数组 y 的第一维用于存储是第几个训练样例，第二维用于存储便于训练的适用于神经网络的输出。例如第 4 个训练样例的理想输出为 5，则转化之后的理想输出的样式为 y[3][]={0,0,0,0,1,0,0,0,0}。

8）泛化及拟合函数

它们的基本流程是以训练样本的九个位置上的值为输入，利用训练的时候确定的权矩阵得到九个取值范围为 0-1 的输出，九个输出中值最大的那个即为下一步的最佳走法。若输出的最佳走法与样本中的最佳走法一致，则把正确预测的个数加一，对 20 个测试样本循环一遍后，用所得到的正确预测的个数除以 20 即为泛化值。

有时候会出现这种情况，所预测的下一步的最佳走法位置已经有棋子了，这时候预测的下一步最佳走法肯定不对。对于这种情况可以加上冲突检测。具体做法是首先对网络的输出结果的按从大到小顺序进行排序，若预测的最佳走法位置上已经有棋子了，则可以把预测的排名第二的最佳走法作为下一步的最佳走法，若排名第二的也已经有棋子了，则看排在第三位的，后面依此类推。

经过改进的泛化函数代码参看附录 1。

拟合函数的实现与泛化函数类似，不同点在于拟合函数没有优化这一步。

9）网络的训练函数

网络的训练函数的具体实现是根据“4. 2.2 BP 网络”推得的结果，其实现代码

如参看附录 2。

函数 int AutoTrainBp(bp_nn *bp, float x[COUT][IN_COUT], int y[COUT][OUT_COUT])的第一个参数为定义的网络的结构体指针，网络的结构体定义如下：

```
typedef struct {        //bp 人工神经网络结构
    int h;              //实际使用隐层数量
    double v[IN_COUT][50];   //隐藏层权矩阵,隐藏层节点最大数量为 50
    double w[50][OUT_COUT];   //输出层权矩阵
    double a;           //学习率
    double b;           //精度控制参数
    int LoopCout;       //最大循环次数
} bp_nn;
```

第二个参数为训练样本的输入，其由来见下面的代码：

```
float x[COUT][IN_COUT]; //训练样本
   for(int i=0;i<COUT;i++)
       for(int j=0;j<IN_COUT;j++)
           x[i][j]=(float)sample[i].input[j];
```

第三个参数为训练样本的输出，其由来已在前面输出结果的表示中说明过了。

4.4.4 结果分析

按照上面的方法，构造的网络经过训练后得到的最佳泛化结果为 0.5。得到最佳泛化结果时，节点个数为 14，学习速率为 0.01，精度为 0.001，循环次数为 32500 次，拟合度为 0.76。此时的权矩阵的值参看附录 5。

其中 v 表示隐藏层权重，w 表示输出层权重；对于隐藏层权重 v 二维数组中的第一维表示第几个输入，第二维表示第几个隐藏层节点，里面存储的值就表示第几个输入到第几个隐藏层节点的权重;对于输出层权重 w 二维数组中的第一维表示第几个隐藏层节点，第二维表示第几个输出层节点，里面存储的值就表示从第几个隐藏节点到第几个输出节点的权重。

对于井字棋来说，若棋盘上有 1 个棋子，则所构建的博弈树的分枝因子为 8，棋盘上有 2 个棋子时，分枝因子为 7，依此类推，当棋盘上有 8 个棋子时分枝因子为 1。平均分枝因子为（8+7+6+5+4+3+2+1）/9=4。剪枝时，所搜索的第一分枝为最佳路径

的分枝的概率为 1/4=0.25，若利用 BP 神经网络辅助剪枝，可以使所搜索的第一分枝为最佳路径的分枝的概率为 0.5，是不使用 BP 神经网络辅助剪枝的 2 倍。若井字棋所搜索的三层博弈树都用神经网络辅助剪枝，则三层搜索的第一分枝都为最佳路径（即出现如图 2-9 所示的情形）的概率为 0.53=0.125，而不用神经网络辅助剪枝，三层搜索的第一分枝都为最佳路径的概率为 0.253=0.015625。使用神经网络和不使用神网络相比，出现理想情况剪枝的概率提高了 8 倍。

4.5 结合神经网络的井字棋游戏设计

4.5.1 网络的初始化

网络的初始化即利用训练时产生的权矩阵及确定的节点数来构建一个 BP 神经网络。此时对于学习速率、精度控制参数、最大循环次数都直接赋值为 0 即可。具体的实现代码参看附录 3。

4.5.2 用于排序的神经网络

4.5.2.1 参数转换

用于排序的神经网络函数的声明为 void net(int inp[3][3])，其参数为当前棋局状态，是一个二维数组，而训练的神经网的输入是一个一维数组，为了保持与所训练的神经网络的一致性，需要把二维数转化为一维数级，转化方式如下代码所示：

```
for(int i=0,j=0;j<3;j++){
    for(int k=0;k<3;k++){
        Input[i]=inp[j][k];
        i++;
    }
}
```

4.5.2.2 下一步最佳走法的生成

其基本原理与 4.4.3.1 节中“输出结果的表示”部分一样，9 个输出节点中输出结果中最大的那个即为下一步的最佳走法。

4.5.2.3 下一步最佳走法排序

根据成为最佳走法的可能性对当前棋局状态的子节点进行排序，输出节点输出结果的值越大成为最佳路径的可能性也越大。网络的输出结果为 0 至 1 之间的实数，根据是第几个输出节点来确定这个实数值对应的具体位置，对节点排序时是对实数值排

序，为了确定实数值所对应的具体位置，需要把实数值和它对应的位置关系保存起来，为此定义了如下的结构体类型：

```
typedef struct{
    double ab;
    int pos;
} Clash;
Clash cl[9];
```

通过对 cl[9]中的实数排序来达到对当前棋局状态的子节点进行排序，其具体实现代码如下：

```
for (i = 0; i < OUT_COUT; i++){
    cl[i].ab=O2[i];//O2[i]为输出层节点的输出值
    cl[i].pos=i+1;
}

//对绝对误差排序
for(int i=0;i<8;i++){
    for(int j=i+1;j<9;j++){
        if(cl[i].ab<cl[j].ab){
            int tem=cl[i].pos;
            cl[i].pos=cl[j].pos;
            cl[j].pos=tem;
            double temp=cl[i].ab;
            cl[i].ab=cl[j].ab;
            cl[j].ab=temp;
        }
    }
}
```

4.5.2.4 把可着子位置排在前面

经过上面的排序之后，有可能把不可着子点排在前几位，为了避免把不可着子点排在前几位，利用稳定排序算法把不可着子位置排在可着子位置后面。这样排序之后，所有的不可着子点都在后面，搜索时，只要遇到一个不可着子点，后面的节点就不用再搜索了，从而进一步减少所需搜索节点的个数。具体代码如下：

```
//把可着子点分枝排在前面
for(int i=0;i<8;i++)
    if(Input[cl[i].pos-1]!=0){
        for(int j=i+1;j<9;j++){
            if(Input[cl[j].pos-1]==0){
                int tem=cl[i].pos;
                cl[i].pos=cl[j].pos;
                cl[j].pos=tem;

                double temp=cl[i].ab;
                cl[i].ab=cl[j].ab;
                cl[j].ab=temp;
                break;
            }
        }
    }
```

4.5.2.5 神经网络与井字棋结合

在对博弈树进行搜索之前，先把当前棋局状态作为神经网络的输入，通过神经网络得到排序之后的当前棋局状态的子节点，然后再对排序之后的子节点调用α－β 剪枝算法进行搜索。在α－β 剪枝算法函数的内部，首先把子节点作为当前棋局状态，再次调用神经网络对子节点的子节点进行排序，最后再递归调用α－β 剪枝算法。在整个实现过程中，有一点需要注意，每次调用神经网络后，会把排序的结果放在全局变量 cl[9]中，由于神经网络需要被多次调用，为了避免出现结果覆盖现象，需要每次调用后把结果转存到局部变量中。神经网络与井字棋的结合主要体现在主函数和α－β 剪枝算法函数中。与主函数的结合代码可参看附录 4。与α－β 剪枝算法函数结合的代码参看附录 6。

结合之后与结合之前的效果图对比如下，人类棋手的走棋序列都为{22，13，21，32}：

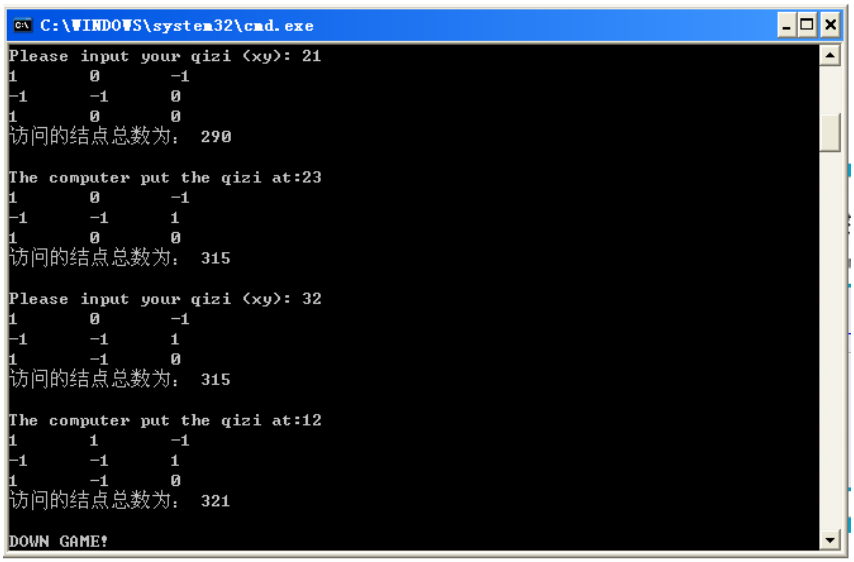

图 4-12 结合神经网络的井字棋效果图

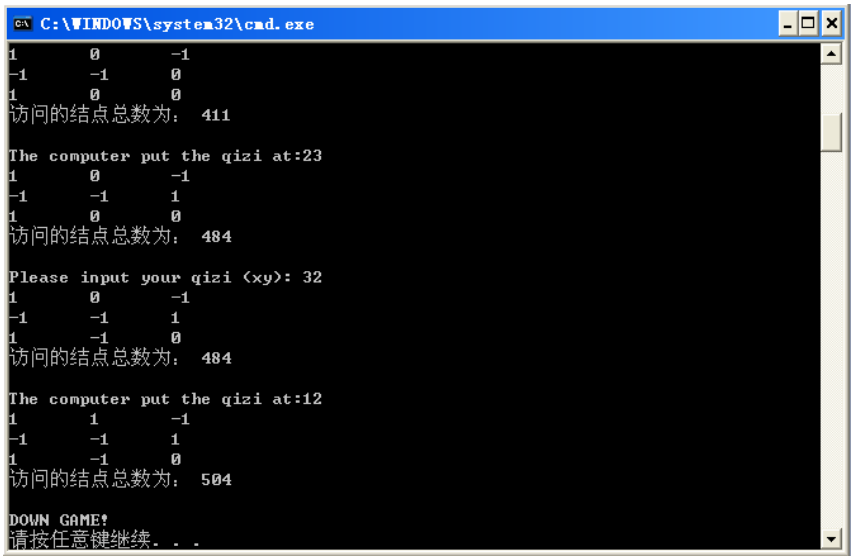

图 4-13 未结合神经网络的井字棋效果图

4.5.3 结果分析

结果分析时，通过对比未使用神经网络和使用神经网络的井字棋在对弈结果为平局时一局所搜索的节点个数，来计算使用神经网络后所搜索的节点个数的减少量。为了对比总共提取了 10 组数据，具体数据如下表所示（假设每次都是人先走）：

表 5-1 实验结果列表

人走子序列	无神经网络	有神经网络
22，13，21，32	504	321
22，13，21，12	510	321
22，21，12，31	511	325
22，32，13，21	406	268
12，13，33，21	471	309
11，12，31，23	498	317
12，13，33，21	471	309
13，33，21，32	570	364
21，31，33，12	499	334
32，33，13，21	581	389

第一列为人类棋手着子点序列，第二列为未使用神经网络井字棋在博弈结果为平局时一局需要搜索的节点数，第三列为使用神经网络时需要搜索的节点数。

未使用神经网络时 10 局需要搜索的总节点数为 5021，使用神经网络时搜索的总节点数为 3257，使用神经网络后减少搜索的百分比为（5021-3257）/5021=35.12%。

4.6 本章小结

这一章论述了井字棋的设计与实现方法，训练样例的特征项的选取，训练样例的获取，BP 神经网络的工作原理，BP 神经网络的设计与实现，BP 神经网络的训练，BP 神经网络训练结果的分析，BP 神经网络与井字棋结合的设计与实现，结合后的结果分析。通过以上工作，验证了本文件所采用的方法确实可以很大程序的提高 α－β 剪枝算法的剪枝效率，减少所需搜索节点的个数。

第 5 章 总结与展望

5.1 总结

α－β 剪枝算法是博弈树高效搜索的基本算法，对博弈树搜索的各种改进算法都是以α－β 剪枝算法为基础的。本文利用机器学习算法极大的改进了α－β 剪枝算法剪枝效率。

文中给出了α－β 剪枝算法的剪枝原理，论述了α－β 剪枝算法剪枝的基本思路。利用博弈树分枝的排列顺序对剪枝效率的影响，提出了利用机器学习的算法来改进α－β 剪枝算法的剪枝效率，给出了构建用于辅助α－β 剪枝算法的学习系统的基本思路及注意事项，并以井字棋为例，选用 BP 神经网络作为模型，详细论述了构建用于辅助α－β 剪枝算法的学习系统的过程和方法。

5.2 展望

由于本人机器学习方面的知识和编程能力的不足，使得所编写的学习系统的泛化能力并不是很高。主要原因有以下几处：

在选取特征项时，没做选取不同特征项的对比实验，可能存在更好的特征项选取方法。

在确定训练样例的数量时，没做科学分析，只是把具有代表性的训练样例每种选择了几个。

在选择模型时，没做多个模型的对比实验，选择的模型不一定就是最合适的模型。

在确定使用 BP 神经网络后，网络参数的选择不够科学，选择时是根据多次尝试和自己想到的一些办法选择的。

BP 神经网络中很多很好的优化算法由于本人能力及时间的限制而没有采用，本文所用到的 BP 神经网络只是在标准 BP 神经网络的基础上作了一些有非常有限的优化，若能采用一些好的优化方法，泛化能力肯定还能再提高。

后来者若是对使用机器学习的算法来改进α－β 剪枝算法有兴趣，可以再从这些个方面入手进行改进。

参考文献

[1] 张聪品，刘春红,徐久成. 博弈树启发式搜索的α－β 剪枝技术研究 [J]. 计算机工程与应用，2008, 44（16）：54-55.

[2] 王镌. 博弈树搜索的算法改进 [J]. 福建电脑，2004，（2）：26-27.

[3] 李红,吴粉侠,刘小豫. 博弈树搜索算法研究 [J]. 长春工程学院学报(自然科学版)，2007，8（2）：.59-62.

[4] 岳金朋，冯速.博弈树搜索算法在中国象棋中的应用 [J].计算机系统应用，2009，（9）：140-143.

[5] 肖齐英,王正志. 博弈树搜索与静态估值函数 [J]. 计算机应用研究，1997，（4）：75-76.

[6] 焦尚彬，刘丁. 博弈树转换表启发式算法研究 [J]. 计算机工程与应用，2010，46（6）：42-45.

[7] 张振，顾治华. 机器博弈及其搜索算法的研究 [J]. 电脑知识与技术，2008, 3(6):1269-1271

[8] 徐心和,邓志立,王　骄,徐长明,刘纪红,马宗民 [J]. 机器博弈研究面临的各种挑战,2008,3(4):287-293.

[9] 邹竞. 基于 MTD(f) 的中国象棋人机博弈算法的设计与优化 [J]. 计算机与数字工程,2008,36(9):38-43.

[10] 王京辉,乔卫民. 基于 PVM 的博弈树的网络并行搜索[J]. 计算机工程,2005,31(9):29-30.

[11] 戴锦锟. 计算机象棋 [J]. 微计算机应用，1994，15（3）：5-11.

[12] 张全中. 连珠棋算法研究 [J]. 中国新技术新产品，2009，（2）：186-187.

[14] 黄继平，张栋，苗华. 六子棋智能博弈系统的研究与实现[J]. 电脑知识与技术，2009，5（25）：7198-7200.

[15] 瞿锡泉，白振兴，包建平. 棋类博弈算法的改进 [J]. 现代电子技术,2005，（1）：96-99.

[16] 张培刚,陈克训. 使用不同的博弈树搜索算法解决计算机围棋的吃子问题[J]. 智能系统学报，2007，2（3）：84-90.

[17] 朱全民，陈松乔. 五子棋算法的研究与思考 [J]. 计算技术与自动化,2006,25(2):71-74.

[18] 张海峰，白振兴. 张登福. 五子棋中的博弈智能设计[J]. 现代电子技术，2004，（174）：25-27.

[19] 董红安 ,蒋秀英. 智能五子棋博弈程序的核心算法[J]. 枣庄学院学报,2005,22(2):61-65.

[20] 蔡增玉, 方娜, 甘勇, 贺蕾. 智能五子棋博弈关键技术研究[J]. 郑州轻工业学院学报,2010,25(6):76-80.

[21] 杨庆文. 智能五子棋中的博弈问题[J]. 智能控制，2006.

[22] 徐心和, 王　骄. 中国象棋计算机博弈关键技术分析[J]. 小型微型计算机系统,2006,27(6):961-969.

[23] 舒康元,胡福乔. 中国象棋计算机博弈引擎改进[J]. 微计算机信息,2009,25(102):39-41.

[24] 王晓鹏, 王骄, 徐心和, 郑新颖. 中国象棋与国际象棋比较分析[J]. 重庆工学院学报,2007,21(1):71-76.

[25] 安涌. 六子棋机器博弈研究与开发[D]. 沈阳：沈阳航天工业大学，2008.

[26] 李果. 六子棋计算机博弈及其系统的研究与实现[D]. 重庆：重庆大学，2007.

[27] 万翼. 计算机国际象棋博弈系统的研究与实现[D]. 成都：西南交通大学，2003.

[28] 莫建文. 机器自学习博弈策略研究与实现[D]. 广西：广西师范大学，2002.

[29] 闵文杰. 六子棋计算机博弈关键技术研究[D]. 重庆：重庆交通大学，2010.

[30] 李翠珠. 六子棋计算机博弈系统的研究与实现[D]. 重庆：重庆理式大学，2010.

[31] 王一非. 具有自学习功能的计算机象棋博弈系统的研究与实现[D]. 哈尔滨：哈尔滨工程大学，2007.

[32] 高强. 一种混合博弈树算法在中国象棋人机博弈中的应用研究[D]. 大连：大连交通大学，2006.

[33] 张颖. 六子棋计算机博弈及其系统的研究与优化[D]. 重庆：重庆理工大学，2008.

[34] 王志水. 基于搜索算法的人工智能在五子棋博弈中的应用研究[D]. 青岛：中国石油大学，2006.

[35] 王骐. 博弈树搜索算法的研究及改进[D]. 浙江：浙江大学，2006.

[36] 陈光年. 基于智能算法的六子棋博弈行为选择的应用研究[D]. 重庆：重庆理工大学，2010.

[37] 岳鹏. 计算机围棋中的算法研究[D]. 重庆：西南大学，2007.

[38] 李小舟. 基于改进博弈树的黑白棋设计与实现[D]. 广州：华南理工大学，2010.

[39] Tom M. Mitchell. 机器学习[M]. 北京：机械工业出版社，2003：60.

[40] Ethem Alpaydin. 机器学习导论[M]. 北京：机械工业出版社，2009：150.

[41] 张丽新，王家钦，赵雁南，杨泽红. 机器学习中的特征选择[J]. 计算机科学，2004，31(11)：180-184.

附录 1 经过改进的泛化函数代码

```
double fanhua(bp_nn *bp){
    float Input[IN_COUT];
    double O1[50];
    double O2[OUT_COUT]; //O1 为隐层输出,O2 为输出层输出
    int n=0;
    for(int k=0;k<20;k++){
    for(int j=0;j<IN_COUT;j++){
        Input[j]=test[k].input[j];
    }
    double temp;
    int i,j;
        for (i = 0; i < (*bp).h; i++) {
            temp = 0;
            for (j = 0; j < IN_COUT; j++)
                temp += Input[j] * (*bp).v[j][i];
            O1[i] = fnet(temp);
        }
        for (i = 0; i < OUT_COUT; i++) {
            temp = 0;
            for (j = 0; j < (*bp).h; j++)
                temp += O1[j] * (*bp).w[j][i];
            O2[i] = fnet(temp);
        }
        for (i = 0; i < OUT_COUT; i++){
        c1[i].ab=O2[i];
        c1[i].pos=i+1;
    }
    int best=1;
```

```
double tem=cl[0].ab;
for(int k=1;k<9;k++){
    if(tem<cl[k].ab){
        best=cl[k].pos;
        tem=cl[k].ab;
    }
}
if(test[k].input[best-1]!=0){//进一步优化，找排名靠前的为 0 的
    //对绝对误差排序
    for(int i=0;i<8;i++){
        for(int j=i+1;j<9;j++){
            if(cl[i].ab<cl[j].ab){
                int tem=cl[i].pos;
                cl[i].pos=cl[j].pos;
                cl[j].pos=tem;
                double temp=cl[i].ab;
                cl[i].ab=cl[j].ab;
                cl[j].ab=temp;
            }
        }
    }
    for(int i=1;i<9;i++){
        int x=cl[i].pos-1;
        if(test[k].input[x]==0){
            best=i+1;
            break;
        }
    }
}
if(best==test[k].result )
    n++;
```

```
    }
  double r=n/20.0;
  cout<<"泛化能力为："<<r<<endl;
  of<<r<<endl;
  return r;
}
```

附录 2 用于训练的 BP 神经网络代码

```
int AutoTrainBp(bp_nn *bp, float x[COUT][IN_COUT], int y[COUT][OUT_COUT]) {
//训练 bp 网络，样本为 x，理想输出为 y
    double f = (*bp).b;                          //精度控制参数
    double a = (*bp).a;                          //学习率
    int h = (*bp).h;                             //隐层节点数
    double v[IN_COUT][50], w[50][OUT_COUT]; //权矩阵
    double Ch_v[IN_COUT][50], Ch_w[50][OUT_COUT]; //权矩阵修改量
    double ChgH[50], ChgO[OUT_COUT];             //修改量矩阵
    double O1[50], O2[OUT_COUT];                 //隐层和输出层输出量

    int LoopCout = (*bp).LoopCout;               //最大循环次数
    int i, j, k, n;
    double temp;
    for (i = 0; i < IN_COUT; i++)                // 复制结构体中的权矩阵
        for (j = 0; j < h; j++)
            v[i][j] = (*bp).v[i][j];
    for (i = 0; i < h; i++)
        for (j = 0; j < OUT_COUT; j++)
            w[i][j] = (*bp).w[i][j];
    double e = f + 1;
    for (n = 0; e > f && n < LoopCout; n++) { //对每个样本训练网络
        e = 0;
        for (j = 0; j < OUT_COUT; j++)
            ChgO[j] = 0;
        for (j = 0; j < h; j++)
            ChgH[j] = 0;
        for (j = 0; j < h; j++)
            for (k = 0; k < OUT_COUT; k++)
```

```
            Ch_w[j][k] = 0;
    for (j = 0; j < IN_COUT; j++)
        for (k = 0; k < h; k++)
            Ch_v[j][k] = 0;
    for (i= 0; i < COUT; i++) {
        for (k= 0; k < h; k++) {          //计算隐层输出向量
            temp = 0;
            for (j = 0; j < IN_COUT; j++)
                temp = temp + x[i][j] * v[j][k];
            O1[k] = fnet(temp);
        }
        for (k = 0; k < OUT_COUT; k++) { //计算输出层输出向量
            temp = 0;
            for (j = 0; j < h; j++)
                temp = temp + O1[j] * w[j][k];
            O2[k] = fnet(temp);
        }
        for (j = 0; j < OUT_COUT ; j++)   //计算输出误差
            e = e + 0.5*(y[i][j] - O2[j]) * (y[i][j] - O2[j]);
        for (j = 0; j < OUT_COUT; j++)
            ChgO[j] = O2[j] * (1 - O2[j]) * (y[i][j] - O2[j]);
        for (j = 0; j < h; j++)     //累加输出层权重改变量
            for (k = 0; k < OUT_COUT; k++)
                Ch_w[j][k] += a * O1[j] * ChgO[k];
        for (j = 0; j < h; j++) {
            temp = 0;
            for (k = 0; k < OUT_COUT; k++)
                temp = temp + w[j][k] * ChgO[k];
            ChgH[j] = temp * O1[j] * (1 - O1[j]);
        }
        for (j = 0; j < IN_COUT; j++)   //累加隐藏层权重改变量
```

```
                for (k = 0; k < h; k++)
                    Ch_v[j][k] += a * x[i][j] * ChgH[k];
            }
            for (j = 0; j < h; j++)            //修改输出层权矩阵
                for (k = 0; k < OUT_COUT; k++)
                    w[j][k] = w[j][k] + Ch_w[j][k];
            for (j = 0; j < IN_COUT; j++)      //修改隐藏层权矩阵
                for (k = 0; k < h; k++)
                    v[j][k] = v[j][k] + Ch_v[j][k];
        }
        for (i = 0; i < IN_COUT; i++)              //把结果复制回结构体
            for (j = 0; j < h; j++)
                (*bp).v[i][j] = v[i][j];
        for (i = 0; i < h; i++)
            for (j = 0; j < OUT_COUT; j++)
                (*bp).w[i][j] = w[i][j];
        return 1;
    }
```

附录 3 用于辅助剪枝的 BP 神经网络的初始化代码

```
int AutoInitBp() { //初始化 bp 网络
    bp.h=14;//bp 是定义的网络结构体的全局变量
    bp.a=0;
    bp.b=0;
    bp.LoopCout=0;

    bp.v[0][0]=-1.54932;    bp.v[0][1]=-2.64304;    bp.v[0][2]=-1.64264;
    bp.v[0][3]=-3.25002;
    bp.v[0][4]=2.38381; bp.v[0][5]=-0.18277;    bp.v[0][6]=0.738268;
    bp.v[0][7]=0.120511;
    bp.v[0][8]=2.95988; bp.v[0][9]=-3.02651;    bp.v[0][10]=1.97908;
    bp.v[0][11]=3.96975;
    bp.v[0][12]=-6.79855;   bp.v[0][13]=-5.25065;bp.v[1][0]=1.87907;
    bp.v[1][1]=-1.04873;
    bp.v[1][2]=5.46207; bp.v[1][3]=-4.70036;    bp.v[1][4]=1.88722;
    bp.v[1][5]=1.37385;
    bp.v[1][6]=6.97036; bp.v[1][7]=1.70322; bp.v[1][8]=-3.69343;
    bp.v[1][9]=-3.42518;
    bp.v[1][10]=4.75579;    bp.v[1][11]=1.14141;    bp.v[1][12]=-4.03896;
    bp.v[1][13]=1.16385;
        bp.v[2][0]=-3.67114;    bp.v[2][1]=-4.84876;    bp.v[2][2]=2.98004;
    bp.v[2][3]=1.96668;
    bp.v[2][4]=5.16119; bp.v[2][5]=2.11564; bp.v[2][6]=-3.38408;
    bp.v[2][7]=-3.32787;
    bp.v[2][8]=2.93433; bp.v[2][9]=-3.67531;    bp.v[2][10]=5.77898;
    bp.v[2][11]=6.64437;
    bp.v[2][12]=3.14022;    bp.v[2][13]=-2.50283;   bp.v[3][0]=1.31837;
    bp.v[3][1]=1.50928;
```

bp.v[3][2]=0.794951;　　bp.v[3][3]=1.98766; bp.v[3][4]=2.3927;

bp.v[3][5]=-2.32548;

bp.v[3][6]=7.11997; bp.v[3][7]=-0.507721;　　bp.v[3][8]=-0.516147;

bp.v[3][9]=4.07224;

bp.v[3][10]=-2.06203;　　bp.v[3][11]=4.02263;　　bp.v[3][12]=-0.250574;

bp.v[3][13]=-7.82151;

bp.v[4][0]=-0.301851;　　bp.v[4][1]=0.0456787;　　bp.v[4][2]=-2.48966;

bp.v[4][3]=3.17639;

bp.v[4][4]=-0.184938;　　bp.v[4][5]=4.37905; bp.v[4][6]=-2.67048;

bp.v[4][7]=4.76606;

bp.v[4][8]=-4.53445;　　bp.v[4][9]=-0.148119;　　bp.v[4][10]=-3.45874;

bp.v[4][11]=-1.82532;

bp.v[4][12]=-5.0037;　　bp.v[4][13]=1.621;　bp.v[5][0]=-2.15928;

bp.v[5][1]=1.55795;

bp.v[5][2]=0.764308;　　bp.v[5][3]=-0.0440164;　bp.v[5][4]=3.48545;

bp.v[5][5]=-6.55315;

bp.v[5][6]=3.43032; bp.v[5][7]=3.52491; bp.v[5][8]=0.632833;

bp.v[5][9]=4.95727;

bp.v[5][10]=-0.565783;　bp.v[5][11]=6.82051;　　bp.v[5][12]=-2.52825;

bp.v[5][13]=-0.727183;

bp.v[6][0]=1.8868;　bp.v[6][1]=5.16044; bp.v[6][2]=2.70485; bp.v[6][3]=5.83487;

bp.v[6][4]=-7.3593; bp.v[6][5]=2.55148; bp.v[6][6]=-1.70949;

bp.v[6][7]=2.30232;

bp.v[6][8]=6.00181; bp.v[6][9]=3.58597; bp.v[6][10]=-3.90435;

bp.v[6][11]=0.360749;

bp.v[6][12]=2.07918;　　bp.v[6][13]=1.70093;　　bp.v[7][0]=10.1169;

bp.v[7][1]=3.17304;

bp.v[7][2]=-6.37947;　　bp.v[7][3]=2.52387; bp.v[7][4]=1.55252;

bp.v[7][5]=0.943822;

bp.v[7][6]=5.93514; bp.v[7][7]=1.47473; bp.v[7][8]=3.66855;

bp.v[7][9]=0.00891552;

bp.v[7][10]=5.29308; bp.v[7][11]=-1.69415; bp.v[7][12]=1.59962;
bp.v[7][13]=-2.01975;
bp.v[8][0]=-0.0253135; bp.v[8][1]=0.109; bp.v[8][2]=1.08753;
bp.v[8][3]=-1.0906;
bp.v[8][4]=3.63433; bp.v[8][5]=2.52604; bp.v[8][6]=1.70379; bp.v[8][7]=1.09961;
bp.v[8][8]=-1.83156; bp.v[8][9]=1.35916; bp.v[8][10]=-1.21808;
bp.v[8][11]=-5.01025;
bp.v[8][12]=2.89275; bp.v[8][13]=7.33972; bp.w[0][0]=1.84411;
bp.w[0][1]=1.21872;
bp.w[0][2]=2.0864; bp.w[0][3]=-7.35755; bp.w[0][4]=3.20192;
bp.w[0][5]=-0.0767378;
bp.w[0][6]=-1.83168; bp.w[0][7]=-7.15758; bp.w[0][8]=3.11927;
bp.w[1][0]=-4.08172;
bp.w[1][1]=-3.12192; bp.w[1][2]=-6.05208; bp.w[1][3]=5.46743;
bp.w[1][4]=-2.13438;
bp.w[1][5]=3.46535; bp.w[1][6]=-3.31072; bp.w[1][7]=-1.10259;
bp.w[1][8]=-2.39925;
bp.w[2][0]=-0.162863; bp.w[2][1]=8.34151; bp.w[2][2]=-4.26683;
bp.w[2][3]=-2.07903;
bp.w[2][4]=3.45238; bp.w[2][5]=6.15629; bp.w[2][6]=-1.98426;
bp.w[2][7]=-7.56696;
bp.w[2][8]=-1.14722; bp.w[3][0]=7.49487; bp.w[3][1]=-4.58016;
bp.w[3][2]=-1.93129;
bp.w[3][3]=-6.58068; bp.w[3][4]=-2.38002; bp.w[3][5]=1.21929;
bp.w[3][6]=3.23355;
bp.w[3][7]=-2.38843; bp.w[3][8]=-6.70159; bp.w[4][0]=-3.19563;
bp.w[4][1]=2.53749;
bp.w[4][2]=-1.57519; bp.w[4][3]=-3.50391; bp.w[4][4]=-6.77981;
bp.w[4][5]=4.95564;
bp.w[4][6]=-10.7141; bp.w[4][7]=3.09325; bp.w[4][8]=4.43507;
bp.w[5][0]=-7.93149;

```
bp.w[5][1]=-1.16657;    bp.w[5][2]=-0.146154;    bp.w[5][3]=2.80679;
bp.w[5][4]=-3.47621;
bp.w[5][5]=-5.46728;    bp.w[5][6]=-3.53774;    bp.w[5][7]=-0.567767;
bp.w[5][8]=0.276433;
bp.w[6][0]=-5.18736;    bp.w[6][1]=-7.2391; bp.w[6][2]=5.19392;
bp.w[6][3]=2.46613;
bp.w[6][4]=-4.29701;    bp.w[6][5]=-4.85649;    bp.w[6][6]=5.20966;
bp.w[6][7]=-6.09054;
bp.w[6][8]=-0.02287;    bp.w[7][0]=-2.5123; bp.w[7][1]=-7.77927;
bp.w[7][2]=-1.64884;
bp.w[7][3]=-0.297743;    bp.w[7][4]=-5.24631;    bp.w[7][5]=1.38073;
bp.w[7][6]=1.89569;
bp.w[7][7]=-2.05734;    bp.w[7][8]=-0.591917;    bp.w[8][0]=-4.99702;
bp.w[8][1]=-3.75094;
bp.w[8][2]=3.37311; bp.w[8][3]=2.57412; bp.w[8][4]=-3.67417;
bp.w[8][5]=2.53157;
bp.w[8][6]=-4.61084;    bp.w[8][7]=-3.14392;    bp.w[8][8]=-3.36722;
bp.w[9][0]=-0.280678;
bp.w[9][1]=-0.850102;    bp.w[9][2]=-4.33943;    bp.w[9][3]=3.6332;
bp.w[9][4]=2.60567;
bp.w[9][5]=1.23123; bp.w[9][6]=-8.37692;    bp.w[9][7]=-0.654244;
bp.w[9][8]=-4.80188;
bp.w[10][0]=4.8945; bp.w[10][1]=4.00873;    bp.w[10][2]=3.34533;
bp.w[10][3]=-5.74758;
bp.w[10][4]=0.836194;    bp.w[10][5]=-4.57472;    bp.w[10][6]=-5.63311;
bp.w[10][7]=1.62927;
bp.w[10][8]=-6.25364;    bp.w[11][0]=-4.27091;    bp.w[11][1]=-5.84482;
bp.w[11][2]=-3.82447;
bp.w[11][3]=-0.66141;    bp.w[11][4]=-1.17675;    bp.w[11][5]=-12.5953;
bp.w[11][6]=6.7627;
```

```
    bp.w[11][7]=1.66002;    bp.w[11][8]=-0.941827;  bp.w[12][0]=-7.12809;
    bp.w[12][1]=0.532058;
    bp.w[12][2]=-7.46452;   bp.w[12][3]=-3.27382;   bp.w[12][4]=3.65773;
    bp.w[12][5]=-6.19783;
    bp.w[12][6]=8.34403;    bp.w[12][7]=2.36572;    bp.w[12][8]=-5.15799;
    bp.w[13][0]=6.52283;
    bp.w[13][1]=-4.9169;    bp.w[13][2]=-3.35692;   bp.w[13][3]=-7.97591;
    bp.w[13][4]=1.32507;
    bp.w[13][5]=-5.62273;   bp.w[13][6]=-0.629386;  bp.w[13][7]=-7.31327;
    bp.w[13][8]=0.241638;
    return 1;
}
```

附录 4 BP 神经网络与主函数结合的代码

```
//主程序
int main(){
     int m=-10000,val=-10000,dep=1; //m 用来存放最大的 val
     int x_pos,y_pos; //记录最佳走步的坐标
     Init();
     cout<<"Qipan: "<<endl;
     PrintQP();
     char IsFirst;
     cout<<"Do you want do first?(y/n)";
     cin>>IsFirst;
     while(IsFirst!='y'&&IsFirst!='n'){
         cout<<"ERROR!"<<"Do you want do first?(y/n)";
         cin>>IsFirst;
     }
     if(IsFirst=='y'){
        L4: // 人先走
           UserInput();
           PrintQP();//输出当前棋局
           cout<<endl;
           num++;//棋盘上的棋子数加 1
           value();//评估当前棋局状态
           if(q==0){//是否是平局
               cout<<"DOWN GAME!"<<endl;
               return 0;
           }
           if (CheckWin()==-1){//是否赢了
              cout<<"You Win! GAME OVER."<<endl;
              return 0;
```

```
        }
        net(cur);//用神经网络对当前棋局的子节点进行排序
        Clash cl_m[9];
        for(int i=0;i<9;i++){
            cl_m[i].pos=cl[i].pos;
            cl_m[i].ab=cl[i].ab;
        }
      //在人下完之后即不是平局，也没有赢则轮到计算机下棋
    //对棋盘中的所有空位置遍历，利用 alpha 剪枝算法计算这些位置的评估值，
     //找出其中评估值最大的一个作为下一步的着子点。
        for(int i=0;i<9;i++){
           int x=(cl_m[i].pos-1)/3;
           int y=(cl_m[i].pos-1)%3;
           count++;
           if(cur[x][y]==0){//如果 x,y 位置为空，则把此位置置 1，然后利用
//剪枝算法计算置 1 后的棋局的评估值。
               cur[x][y]=1;
               if(CheckWin()==1){//判断计算机走一步之后否获胜
                   cout<<"The computer put the qizi at:"<<x+1<<y+1<<endl;
                   PrintQP();
                   cout<<"The computer WIN! GAME OVER."<<endl;
                   return 0;
                }
                cut(val,dep,1);//利用 alpha 剪枝算法计算当前棋局的评估值，//并把它
存放在 val 中
                if(val>m) {//如果当前棋局的评估值大于用于存放候选走法
//最大值的变量 m，则把 val 值赋给 m,并记下导致出现 val 值着法的位置。
                    m=val;
                    x_pos=x;y_pos=y;
               }
//为了计算下一个候选走法的评估值，把 val 的值重新置为负无穷大（即-1000），
```

```
                    //并把上一步的着法恢复
                    val=-10000;
                    cur[x][y]=0;
                }else{
                  break;
                }
            }
        //把候选着法中评估值最大的位置作为下一步的着法，置 1
        cur[x_pos][y_pos]=1;
        //重新置 val 值和 m 值为负无穷，起始遍历深度为 1
        val=-10000;m=-10000;dep=1;
        //打印输出计算机走一步之后的棋局
        cout<<"The computer put the qizi at:"<<x_pos+1<<y_pos+1<<endl;
        PrintQP();
        cout<<endl;
        num++;//让棋盘上已有的棋子数加 1
        value();//评估当前棋局的评估值
    //在人先走的情况下，如果计算机走后，把所有的空位都填上计算机的子，
            //计算机仍不能赢，则双方平局
        if(q==0){
            cout<<"DOWN GAME!"<<endl;
            return 0;
        }
        goto L4;
      }
     return 0;
}
```

附录 5 训练完成后的 BP 神经网络权矩阵

v[0][0]=-1.54932 v[0][1]=-2.64304 v[0][2]=-1.64264 v[0][3]=-3.25002

v[0][4]=2.38381 v[0][5]=-0.18277 v[0][6]=0.738268 v[0][7]=0.120511

v[0][8]=2.95988 v[0][9]=-3.02651 v[0][10]=1.97908 v[0][11]=3.96975

v[0][12]=-6.79855 v[0][13]=-5.25065

v[1][0]=1.87907 v[1][1]=-1.04873 v[1][2]=5.46207 v[1][3]=-4.70036

v[1][4]=1.88722 v[1][5]=1.37385 v[1][6]=6.97036 v[1][7]=1.70322

v[1][8]=-3.69343 v[1][9]=-3.42518 v[1][10]=4.75579 v[1][11]=1.14141

v[1][12]=-4.03896 v[1][13]=1.16385

v[2][0]=-3.67114 v[2][1]=-4.84876 v[2][2]=2.98004 v[2][3]=1.96668

v[2][4]=5.16119 v[2][5]=2.11564 v[2][6]=-3.38408 v[2][7]=-3.32787

v[2][8]=2.93433 v[2][9]=-3.67531 v[2][10]=5.77898 v[2][11]=6.64437

v[2][12]=3.14022 v[2][13]=-2.50283

v[3][0]=1.31837 v[3][1]=1.50928 v[3][2]=0.794951 v[3][3]=1.98766

v[3][4]=2.3927 v[3][5]=-2.32548 v[3][6]=7.11997 v[3][7]=-0.507721

v[3][8]=-0.516147 v[3][9]=4.07224 v[3][10]=-2.06203 v[3][11]=4.02263

v[3][12]=-0.250574 v[3][13]=-7.82151

v[4][0]=-0.301851 v[4][1]=0.0456787 v[4][2]=-2.48966 v[4][3]=3.17639

v[4][4]=-0.184938 v[4][5]=4.37905 v[4][6]=-2.67048 v[4][7]=4.76606

v[4][8]=-4.53445 v[4][9]=-0.148119 v[4][10]=-3.45874 v[4][11]=-1.82532

v[4][12]=-5.0037 v[4][13]=1.621

v[5][0]=-2.15928 v[5][1]=1.55795 v[5][2]=0.764308 v[5][3]=-0.0440164

v[5][4]=3.48545 v[5][5]=-6.55315 v[5][6]=3.43032 v[5][7]=3.52491

v[5][8]=0.632833 v[5][9]=4.95727 v[5][10]=-0.565783 v[5][11]=6.82051

v[5][12]=-2.52825 v[5][13]=-0.727183

v[6][0]=1.8868 v[6][1]=5.16044 v[6][2]=2.70485 v[6][3]=5.83487

v[6][4]=-7.3593 v[6][5]=2.55148 v[6][6]=-1.70949 v[6][7]=2.30232

v[6][8]=6.00181 v[6][9]=3.58597 v[6][10]=-3.90435 v[6][11]=0.360749

v[6][12]=2.07918 v[6][13]=1.70093

v[7][0]=10.1169 v[7][1]=3.17304 v[7][2]=-6.37947 v[7][3]=2.52387

v[7][4]=1.55252 v[7][5]=0.943822 v[7][6]=5.93514 v[7][7]=1.47473

v[7][8]=3.66855 v[7][9]=0.00891552 v[7][10]=5.29308 v[7][11]=-1.69415

v[7][12]=1.59962 v[7][13]=-2.01975

v[8][0]=-0.0253135 v[8][1]=0.109 v[8][2]=1.08753 v[8][3]=-1.0906

v[8][4]=3.63433 v[8][5]=2.52604 v[8][6]=1.70379 v[8][7]=1.09961

v[8][8]=-1.83156 v[8][9]=1.35916 v[8][10]=-1.21808 v[8][11]=-5.01025

v[8][12]=2.89275 v[8][13]=7.33972

w[0][0]=1.84411 w[0][1]=1.21872 w[0][2]=2.0864 w[0][3]=-7.35755

w[0][4]=3.20192 w[0][5]=-0.0767378 w[0][6]=-1.83168 w[0][7]=-7.15758

w[0][8]=3.11927

w[1][0]=-4.08172 w[1][1]=-3.12192 w[1][2]=-6.05208 w[1][3]=5.46743

w[1][4]=-2.13438 w[1][5]=3.46535 w[1][6]=-3.31072 w[1][7]=-1.10259

w[1][8]=-2.39925

w[2][0]=-0.162863 w[2][1]=8.34151 w[2][2]=-4.26683 w[2][3]=-2.07903

w[2][4]=3.45238 w[2][5]=6.15629 w[2][6]=-1.98426 w[2][7]=-7.56696

w[2][8]=-1.14722

w[3][0]=7.49487 w[3][1]=-4.58016 w[3][2]=-1.93129 w[3][3]=-6.58068

w[3][4]=-2.38002 w[3][5]=1.21929 w[3][6]=3.23355 w[3][7]=-2.38843

w[3][8]=-6.70159

w[4][0]=-3.19563 w[4][1]=2.53749 w[4][2]=-1.57519 w[4][3]=-3.50391

w[4][4]=-6.77981 w[4][5]=4.95564 w[4][6]=-10.7141 w[4][7]=3.09325

w[4][8]=4.43507

w[5][0]=-7.93149 w[5][1]=-1.16657 w[5][2]=-0.146154 w[5][3]=2.80679

w[5][4]=-3.47621 w[5][5]=-5.46728 w[5][6]=-3.53774 w[5][7]=-0.567767

w[5][8]=0.276433

w[6][0]=-5.18736 w[6][1]=-7.2391 w[6][2]=5.19392 w[6][3]=2.46613

w[6][4]=-4.29701 w[6][5]=-4.85649 w[6][6]=5.20966 w[6][7]=-6.09054

w[6][8]=-0.02287

w[7][0]=-2.5123 w[7][1]=-7.77927 w[7][2]=-1.64884 w[7][3]=-0.297743
w[7][4]=-5.24631 w[7][5]=1.38073 w[7][6]=1.89569 w[7][7]=-2.05734
w[7][8]=-0.591917
w[8][0]=-4.99702 w[8][1]=-3.75094 w[8][2]=3.37311 w[8][3]=2.57412
w[8][4]=-3.67417 w[8][5]=2.53157 w[8][6]=-4.61084 w[8][7]=-3.14392
w[8][8]=-3.36722
w[9][0]=-0.280678 w[9][1]=-0.850102 w[9][2]=-4.33943 w[9][3]=3.6332
w[9][4]=2.60567 w[9][5]=1.23123 w[9][6]=-8.37692 w[9][7]=-0.654244
w[9][8]=-4.80188
w[10][0]=4.8945 w[10][1]=4.00873 w[10][2]=3.34533 w[10][3]=-5.74758
w[10][4]=0.836194 w[10][5]=-4.57472 w[10][6]=-5.63311 w[10][7]=1.62927
w[10][8]=-6.25364
w[11][0]=-4.27091 w[11][1]=-5.84482 w[11][2]=-3.82447 w[11][3]=-0.66141
w[11][4]=-1.17675 w[11][5]=-12.5953 w[11][6]=6.7627 w[11][7]=1.66002
w[11][8]=-0.941827
w[12][0]=-7.12809 w[12][1]=0.532058 w[12][2]=-7.46452 w[12][3]=-3.27382
w[12][4]=3.65773 w[12][5]=-6.19783 w[12][6]=8.34403 w[12][7]=2.36572
w[12][8]=-5.15799
w[13][0]=6.52283 w[13][1]=-4.9169 w[13][2]=-3.35692 w[13][3]=-7.97591
w[13][4]=1.32507 w[13][5]=-5.62273 w[13][6]=-0.629386 w[13][7]=-7.31327
w[13][8]=0.241638

附录 6 井字棋与α－β剪枝算法函数结合的代码

```
int cut(int &val,int dep,bool max)
{
//如果搜索深度达到最大深度，或者深度加上当前棋子数已经达到 9，就直接调用评
//价函数
    if(dep==depth  || dep+num==9){
        return value();
    }
    int i,j,flag,temp;
    bool out=false;//out 记录是否剪枝，初始为 false
    if(CheckWin()==1){//如果用户玩家输了，就置上一层的评价值为无穷（用很大的//值代表
无穷）
        val=10000;
        return 0;
    }
    if(max)//如果上一层是极大层，本层则需要是极小层，记录 flag 为无穷大；
    //反之，则为记录为负无穷大
        flag=10000;//flag 记录本层节点的极值
    else
        flag=-10000;
    net(cur);//对当前棋局的子结点排序
    Clash cl_c[9];
    for(int i=0;i<9;i++){
        cl_c[i].pos=cl[i].pos;
        cl_c[i].ab=cl[i].ab;
    }
    for(int k=0;k<9 && !out;k++){
        int i=(cl_c[k].pos-1)/3;
        int j=(cl_c[k].pos-1)%3;
        count++;
        if(cur[i][j]==0){      //如果该位置上没有棋子
```

```
            if(max){//并且为上一层为极大层，即本层为极小层,轮到用户玩家走了。
                cur[i][j]=-1;      //该位置填上用户玩家棋子
                if(CheckWin()==-1)//如果用户玩家赢了
            temp=-10000;//置棋盘评价值为负无穷
                else
                    temp=cut(flag,dep+1,!max);//否则继续调用 ab 剪枝函数
            if(temp<flag)//如果下一步棋盘的评价值小于本层节点的极值，则置本层
//极值为更小者
                flag=temp;
            if(flag<=val)//如果本层的极值已经小于上一层的评价值，则不需要搜索下//去，
剪枝
                out=true;
            }else{//如果上一层为极小层，算法与上面刚好相反
                cur[i][j]=1;
                if(CheckWin()==1)
            temp=10000;
        else
            temp=cut(flag,dep+1,!max);
                    if(temp>flag)
                        flag=temp;
                    if(flag>=val)
                        out=true;
                }
                cur[i][j]=0;//把模拟下的一步棋还原，回溯
        }else{//遇到第一个不可着子点后就退出循环
            break;
        }
    }
    if(max){//根据上一层是否为极大层，用本层的极值修改上一层的评价值
        if(flag>val)
            val=flag;
```

```
    }else{
        if(flag<val)
            val=flag;
    }
    return flag;//函数返回的是本层的极值
}
```

致谢

在毕业论文完成之际，我要忠心的感谢我的导师苏贵斌副教授，感谢他在我整个三年研究生生涯中对我的关怀和指导以及在写论文过程中的督促和建议。还要感谢在这三年中曾经教过我的老师，没有他们的教导和帮助，对计算机方面的知识的掌握不可达到现在的程度。除此之外还要感谢内蒙古师范大学，是师大给我提供了学习的机会和条件。

最后，我要感谢我的父母，由于他们的关怀和教育使我走到了今天，他们的殷切期望和鼓励是我最大的动力，没有他们就没有我的一切。