1 はじめに

1.1 背景と目的

コンピュータゲームプレイヤの研究は人工知能研究の一分野として研究されてきており

① ルールが明快であり勝敗がはっきり決まることにより研究の成果が分かりやすい

② 人間の名人に勝つという分かりやすい目標がある

などの理由から盛んである。

コンピュータゲームプレイヤの研究の題材とされるゲームにはオセロやチェス、将棋、囲碁などがあり、中でもオセロやチェスでは既に人間の名人に勝つまでに達したプログラムが作られている。しかし、将棋や囲碁では着手可能な指し手の多さによる探索空間の広さなどのためにまだ人間の名人クラスには達していない。今回研究の対象とする将棋は現在アマチュア 4 段程度の強さと言われている。

コンピュータゲームプレイヤが強くなるためには

① コンピュータハードウェアの進化により探索をより深くまでできるようになる。

② 探索アルゴリズムの改良により効率のいい探索ができるようになる。

③ 静的評価関数の改善により正しい判断を行えるようになる。

などが考えられるが、本研究では静的評価関数の改善を行うことを考える。

静的評価関数は盤面から自分、相手が持っている評価要素に重みをかけて足していくことにより盤面の評価を行うものであるが、どのような評価要素を作り、その重みはどれくらいに設定するのが良いのかを決めるのが非常に難しく、またこの調整がコンピュータゲームプレイヤの強さに非常に大きな影響を与えるため、この調整に非常な労力を費やしている。そこでこの調整をコンピュータに自動的に学習させようという研究が行われてきていて、チェスでは TD 法による学習で、大きな成果を挙げている。また、将棋でも駒の価値に関しては学習がうまくいっているが、そのほかの評価要素の学習に成功し強くなったというような例はない。そこで本研究では近山・田浦研究室のコンピュータ将棋プログラム『激指』を用いて駒の価値以外の評価要素も含めて将棋の静的評価関数を TD 法により自動学習させて改善することを目的とする。

1.2 本研究の貢献

本研究ではチェスで大きな成果を挙げている TD 法による静的評価関数の学習を強くなったというほどの成果を挙げていない将棋でも通用するかどうかを検証する。

1.3 本論文の構成

本論文では

2 章に関連研究としてゲーム木探索、TD 法について紹介し、

3 章では本研究で行った『激指』の静的評価関数の改善方法を述べ、

4 章で行った学習実験について述べる。

2 関連研究

本章では関連研究としてゲーム木探索と TD 法について説明する。

2.1 ゲーム木探索

ここでは２プレイヤ完全情報ゲームでコンピュータゲームプレイヤが指し手を選択するために用いているゲーム木、評価関数、探索アルゴリズムについて説明する。

2.1.1 ゲーム木

ゲーム木とは指し手を枝とし、局面をノードとして現在の局面から展開した木のことである。この木を完全に展開すると葉ノードは勝ち、負け、引き分けのいずれかを表すことになり、それを元に探索し指し手を選択すればいいのであるが、探索空間が膨大であるため現実的に不可能である。そこで通常、ある深さで探索を打ち切り、葉ノードに対してその局面でどちらがどれくらい優勢かというのを判断する評価関数を用いて評価し、その評価値に基づいて指し手を選択する。

2.1.2 静的評価関数

評価関数は上記で説明したようにある局面を与えるとその局面で自分がどれくらい有利かという数値を返す関数である。通常、その局面で自分が勝ちなら＋∞、引き分けなら±0、負けなら－∞とし、途中の局面では基本的には

$$\sum_{\text{全ての評価要素}}(\text{特徴量}\times\text{重み}) \tag{1}$$

によって計算される。ここで特徴量とはその評価要素を自分が持っている数から相手が持っている数を引いたものである。

たとえば『盤上の歩』の価値が 100 であったとして、盤上の自分の歩は 8 枚、相手の歩が 6 枚だったとすると、『盤上の歩』の特徴量は 2 であり、それにより自分の評価値が 200 上昇する。

2.1.3 Minimax 法

Minimax 法は相手が自分にとって最も不利となる手を指してくると仮定して自分が最も高い評価値の局面に進めるように指し手を選ぶ方法である。

具体的には葉ノードに評価関数を適用して評価値を求め、ノードが自分の手番(自局面)であるときには子ノードの評価値の中から最も高い評価値を自分の評価値として、ノードが相手の手番(相手局面)であるときには子ノードの評価値の中から最も低い評価値を自分の評価値として葉ノードからルートノードまで評価値を伝えていき、最終的にルートノードではその評価値に到達できるノードを次の手として選択する。このとき選ばれたルートノードから葉ノードまでの一連のノードを PV(Principal Variation)ノードという。

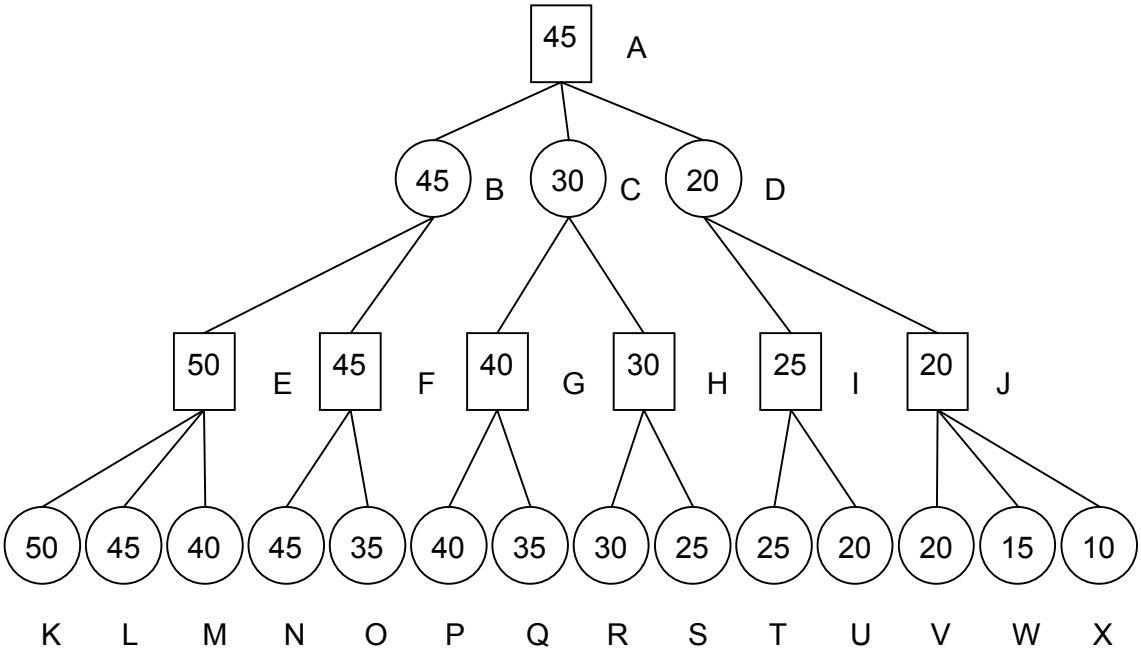

図 1　Minimax 法によるゲーム木

例えば図 1 では、まず K～Ｘまでの葉の評価値を求める。次に E は自局面なのでその子ノードのＫ，Ｌ，Ｍのうち評価値が最高のＫを選ぶ。同様にＥＦＧＨＩＪも選ぶ。次にＢは相手局面なのでその子ノードのＥ，Ｆのうち評価値が最低の F を選ぶ。ＣＤも同様に選ぶ。そしてＡは自局面なので評価値が最高のＢを選び、結局手としてＢに移る指し手を選ぶことになる。このとき A,B,E,N が PV ノードである。

自局面では子ノードの局面から最高の評価値を選び、相手局面では子ノードの局面から最低の評価値を選ぶという操作は子ノードの局面の評価値にマイナスを掛けて最高の評価値を選ぶというようにすれば一つの関数で書けるのでアルゴリズムが簡潔になる。その方式(Negamax Form)の擬似コードを図 2 に示す。

```
int Minimax(node_t n, int d){
   int i, score = -∞;
   if(d == 0 || n == terminal) return Evaluate(n);
   for(i = 0; i < n.num_of_children; i++){
      g = Minimax(n.child_node[i], d-1)
      score =max(score, g);
   }
}
```

図 2　Minimax 法(Negamax Form)の擬似コード

2.1.4　Alpha-Beta 法

Alpha-Beta 法は Minimax 法と全く同じ指し手を探索不要な手を枝狩りすることにより高速に選ぶ方法である。

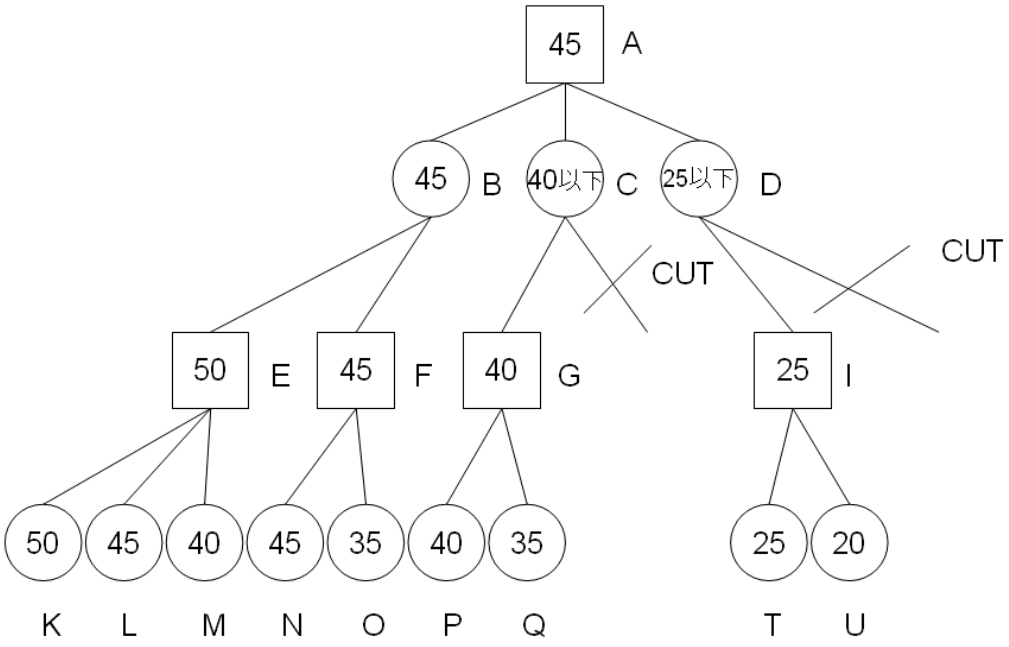

図 3　Alpha-Beta 法によるゲーム木

例えば図 3 だと左から探索していくものとして、G の評価値が 40 と決まった時点でCの評価値は 40 以下であることが確定し、Bの評価値が 45 であることから A は 45 未満の値は選ばないので C を選ぶ可能性はなくなる。そこで C の残りの子を探索する必要はないので枝狩りできる。D の子も I が決まった時点で同様に枝狩りできる。

このように探索不要な手を枝狩りすることにより左から探索していくものとして理想的に左から順に評価値が高い順番に並んでいるとすれば同じ数の葉ノードの評価値を計算するとして Minimax 法の 2 倍の深さまで探索することが可能になる。

図 4 に Alpha-Beta 法の擬似コード(Negamax Form)を示す。

```
int AlphaBeta(node_t n, int d, int _, int _){
   int score = -∞;
   if(d == 0 || n == terminal) return Evaluate(n);
   for(i = 0; i < n.num_of_children; i++){
      score = max(score, -AlphaBeta(n.child_node[i], d-1, -_, -_));
      _ = max(_, score);
      if(_≧_) return _;
   }
   return score;
}
```

図 4　Alpha-Beta 法の擬似コード

2.2 TD 法

ここでは TD(Temporal Difference)法の説明と TD 法を使ったチェス、将棋の静的評価関数の学習例を紹介する。

2.2.1　TD 法の概要

TD(Temporal Difference)法とは経験から学習する学習方法であるが、従来のアルゴリズムとの大きな違いは従来のアルゴリズムが最終的な結果を待って(結果が出るまでの一連の流れをエピソードと呼び、将棋だと 1 局である)その結果を使って推定値を修正していくのに対して、TD 法では状態の変化一つ一つに対してそのつど推定値を修正していく点にある。

将棋のように 1 つのエピソードが長いものでは従来のアルゴリズムのように最終的な結果(勝ち、負け、引き分け)を待って推定値を更新するという方法では学習に時間がかかりすぎるがその点 TD 法では 1 手 1 手更新していくので速く学習できることが期待できる。

まずはテーブル形式で状態と推定値が 1 対 1 で対応している場合についての TD 法について説明する。最も単純な TD 法である TD(0)のアルゴリズムは以下のとおりである。
取りうる全ての状態を S、時刻 t での非終端状態を $s_t \in S$、そのときの推定値を $V(s_t)$として、その次の状態 s_{t+1} の推定値 $V(s_{t+1})$との差から $V(s_t)$の値を修正する。

$$V(s_t) \leftarrow V(s_t) + \alpha[V(s_{t+1}) - V(s_t)] \qquad (2)$$

ここで α はステップサイズパラメータと呼ばれどれくらい一気に修正するかという基準であり $0 < \alpha \leqq 1$ である。

TD(0)は現在の状態と次の状態から現在の状態の推定値を更新するアルゴリズムであるが、これを発展させて次の状態から今までの状態全ての推定値を更新するアルゴリズムがTD(λ)である。その更新式を示す。

$$V(s_k) \leftarrow V(s_k) + \alpha[V(s_{t+1}) - V(s_t)]\lambda^{t-k} \tag{3}$$

1≦k≦t なる全ての k に対して(3)式で推定値を更新する。
ここでλは減衰パラメータと呼ばれ、どれだけ過去の状態に対する影響を残すかを決めるパラメータであり 0≦λ≦1 である。このαとλは学習の速さや収束性に影響する。また、λ=0 のときが TD(0)である。

2.2.2 関数近似

2.2.1 では状態と推定値がテーブル形式で 1 対 1 で対応し、その 1 つ 1 つの推定値を更新する場合の TD 法について述べたが、将棋のようなゲームでは状態の数が多すぎて 1 つ 1 つの状態に推定値を定めることは不可能である。したがって推定値 $V(s_t)$をパラメータベクトル$\vec{\theta}$の関数として表現する。パラメータ数($\vec{\theta}$の要素数)は状態数と比べてはるかに小さく一つの状態の価値関数を更新することは他の多くの状態の価値がその変化量の影響を受けることになる。したがって全ての状態に対して真の価値との誤差をゼロにまで小さくすることは不可能になるので通常入力分布 P 上の平均 2 乗誤差 MSE を最小化することを目標とする。

$$\mathrm{MSE}(\vec{\theta}_t) = \sum_{s \in S} P(s)[V^{\pi}(s) - V_t(s)]^2 \tag{4}$$

ここで$V^{\pi}(s)$とは状態 s での真の価値である。

最急降下法はパラメータベクトルが限られた個数の実数値の要素を持つ列ベクトル$\vec{\theta}_t = (\theta_t(1), \theta_t(2), ..., \theta_t(n))^T$で表現され、$V_t(s)$がすべての状態 s∈S に対して連続で微分可能な$\vec{\theta}$の関数であらわされているとき、実例を得るたびにそれに対する平均 2 乗誤差$\sum_{s_t}[V_{t+1}(s_t) - V_t(s_t)]^2$を最も減少させる方向にパラメータベクトルを修正する方法である。

つまり TD(0)であれば

$$\begin{aligned}\vec{\theta}_{t+1} &= \vec{\theta}_t - \tfrac{1}{2}\alpha \nabla_{\vec{\theta}_t}[V_{t+1}(s_t) - V_t(s_t)]^2 \\ &= \vec{\theta}_t + \alpha[V_{t+1}(s_t) - V_t(s_t)]\nabla_{\vec{\theta}_t} V_t(s_t)\end{aligned} \tag{5}$$

により$\vec{\theta}$を調整していく。TD(λ)であれば

$$\vec{\theta}_{t+1} + \vec{\theta}_t - \alpha[V_t(s_{t+1}) - V_t(s_t)]\sum_{k=1}^{t} \lambda^{t-k} \nabla_{\vec{\theta}_t} V_t(s_k) \tag{6}$$

により$\vec{\theta}$を調整していく。

平均 2 乗誤差を最も減少させる方向に修正するために収束速度は速いが、局所最適解への収束となってしまう。

これをアルゴリズムにしたものを図 5 に示す。

$\vec{\theta}$ を任意に初期化し、$\vec{e} = \vec{0}$ とする
各エピソードに対して繰り返し：
　s ← エピソードの初期状態
　エピソードの各ステップについて繰り返し：
　　行動を取り、次状態 s′を観測する
　　$\delta \leftarrow V(s) - V(s)$
　　$\vec{e} \leftarrow \lambda \vec{e} + \nabla_{\vec{\theta}} V(s)$
　　$\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \vec{e}$
　　$s \leftarrow s'$
　s が終端状態ならば繰り返しを終了

図 5　TD(λ)のアルゴリズム

近似関数が線形つまり

$$V_t(s) = \vec{\theta}_t^T \vec{\phi}_s = \sum_{i=1}^{n} \theta_t(i)\phi_s(i) \tag{7}$$

の形の式で表される場合

$$\nabla_{\vec{\theta}_t} V_t(s) = \vec{\phi}_s \tag{8}$$

となり非常に簡単な式になる。

またこのとき、最適解は 1 つしか存在しないので局所最適解が大域最適解になる。

2.2.3　TDLEAF(λ)

2.2.1、2.2.2 で説明した TD 法であるが、それをどうやってゲーム木での学習に適用するのかというのをここで説明する。

まずゲーム木では状態数は非常に多いので 2.1.2 で説明したような特徴量ベクトルをパラメータとする静的評価関数で関数近似を行っている。この特徴量を Principal Variation の葉ノードから抽出して TD(λ)を行うのが TDLEAF(λ)である。

2.2.4　TD 法によるチェスの静的評価関数の学習

KnightCap の例を紹介する。

チェスプログラム KnightCap の静的評価関数の学習では駒の価値の重みを標準的な値にし、それ以外の要素については重みを 0 としてインターネット上のチェス対戦サーバでさまざまな人間やコンピュータチェスプレイヤを相手に学習した。α は 1.0、λ は 0.7 で一定。学習させた評価要素は 1368 個を序盤、初盤、中盤、終盤で異なる値とした。

その結果人間の手で設定した重みよりも強くなるという結果を得た。

2.2.5　TD 法による将棋の静的評価関数の学習

北村は Alpha-Beta 法による全幅探索、探索の深さは基本的に 5 として静的評価関数は駒の価値のみを考慮したもので駒の価値について学習させた。[3]

学習側の駒の価値は全て同じ値からスタートさせた。

その結果は学習の対戦相手に用いた駒の価値に対しては 5 割を超える勝率を残した。

また、駒の価値を固定して王の周りの利きに関する価値について学習もさせたがこちらは勝率が 1 割を切る結果となりうまく学習できなかった。

3 『激指』の静的評価関数の改善

3 章では本研究で行った『激指』の評価関数の改善方法について述べる。

3.1 『激指』の静的評価関数

ここでは本研究で改善を試みる将棋プログラム『激指』の評価関数について述べる。

『激指』には

① 駒の価値

『激指』が用いている駒の価値について盤上の駒の価値について表 1 に、持ち駒の価値について表 2 に示す。

表 1 『激指』が用いている盤上の駒の価値

王	飛車	竜	角	馬	金	銀	成銀	桂馬	成桂	香	成香	歩	と
100000	950	1300	800	1150	600	550	600	400	600	370	600	100	600

表 2 『激指』が用いている持ち駒の価値

飛車	角	金	銀	桂馬	香	歩
1150	1000	660	605	440	407	110

また、持ち駒が 2 枚目以降は 1 枚目より価値を落とすなどの工夫も行われている。

② 駒の位置

駒がどれくらい働いているのかを判断するために味方の王との位置関係、相手の王との位置関係から判断し評価値に加えている。

また、この重みは終盤になるほど効果を大きくしている。

③ 特定の形に関する要素

浮いている駒に対してマイナスにしたり、駒の組み合わせで悪形になってないかのチェック、王の囲いなどに関する評価要素があり、また盤面の進行度によりその重みを変えたりする工夫もされている。

3.2 学習方法の概要

本研究では『激指』同士の自動対戦により静的評価関数の学習をした。

『激指』では序盤は定跡としてデータベースを用いて探索せずに次の手を生成しているが、この部分では学習は行わず、定跡が終わった局面(10 手～50 手目)から学習を開始した。

定跡が終わった局面というのは 1000 局すると 800 局以上が異なる局面となるのでこれにより、さまざまな局面を経験することができる。

また、250 手を超えて勝敗がつかない場合は引き分けとしてそこで対局を終えた。

先手を学習側、後手をデフォルトの『激指』とし、学習は先手に手番が回ってくるごとに行った。学習は TDLEAF(λ)で行い、λは 0.9 で固定した。

また、『激指』の評価値[-99999,99999]は勝率に対して線形ではないのでシグモイド関数を使い予想勝率に変換した。用いたシグモイド関数を(9)式に、図を図 6 に示す。

$$P(E) = \frac{1}{1+e^{-E/1000}} \tag{9}$$

ここで E は静的評価関数により計算された評価値である。
例えば、盤上の金の価値が 600、持ち駒の金の価値が 660 だとして、相手の金を取り、それ以外の評価要素に優劣はついていないとすると評価値は＋1260 になるが、この時

$$P(1260) = \frac{1}{1+e^{-1260/1000}} = 0.78 \tag{10}$$

である。

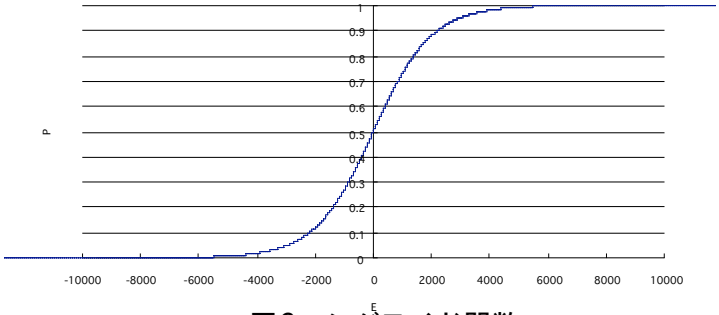

図6　シグモイド関数

シグモイド関数を使うと最急降下法のときに必要な導関数が簡単になり

$$\frac{dP}{dE} = \frac{1}{1000}P(1-P) \tag{11}$$

したがって線形のとき

$$\frac{\partial}{\partial \theta_i}P = \frac{\partial P}{\partial E}\frac{\partial E}{\partial \theta_i} = \frac{1}{1000}\phi_i P(1-P) \tag{12}$$

となる。

(6)式の意味から考えるにαは(0,1]の範囲にあるべき数値であるが、このシグモイド関数を使うことにより 20 万の幅のある評価値を幅 1 にしているので、()式のαには 20 万倍しておかなくてはならない。しかし、『激指』で勝ちを＋99999、負けを−99999 としているのは便宜上であり、勝敗が決まっていない場面でそれより絶対値の大きい数値が出なければ±50000 にしようと±1000000 にしようと関係ないのであり、200000 という数値に特に意味があるわけではない。

そこでαの決定は試行錯誤により 5 万から開始し、学習が行われるごとに徐々に減らし 3000 局で 1000 程度まで減らすこととした。

3.3 TD 法による学習機能の実装

『激指』に学習機能を実装した。実装した学習の流れは以下のとおりである。

```
while(1){
   _ を更新
   初期局面に戻す。
   10 手～50 手定跡によって進める
   while(1){
      if(手番が先手)パラメータを学習している値にセット
      else パラメータをデフォルトの値にセット
      『激指』の探索を行う
      if(手番が先手){
         探索して得られた principal variation の葉ノードから評価値、特徴量ベクトルを抽出
         評価値にシグモイド関数をかける
         これらの値を使って図の方法によりパラメータを更新
         変更したパラメータで principal variation の葉ノードの評価値を取ってくる
      }
   }
}
```

図 7　実装した学習の流れ

4 学習実験

この章では本研究で行った学習実験とその結果について述べる。

なお、学習前後での勝率は、定跡が終わった時点から学習側を先手、デフォルト側を後手として対局させ、またその同一局面から学習側を後手、デフォルト側を先手として対局させ、どちらも先手が勝つ、またはどちらも後手が勝つという場合には無効として数にいれずに 1000 局程度対戦させたときの勝率である。

このようにしたのは普通に対局させたのでは定跡部分で既に優劣がついていると正確な勝率が得られないと考えられるからである。

4.1 駒の価値の学習

まずは 3.1 の①の駒の価値に関して学習実験を行った。

ここで駒の価値とは 3.1 の①に述べたように盤上の駒、持ち駒のそれぞれの価値である。持ち駒の 2 枚目以降の価値を少し下げるというような要素および、そのほか②③の要素などは全てデフォルトの値のまま使用した。

学習側の駒の価値を盤上の歩のみ 100 に固定し、そのほかの駒の価値を全て 0 にし、駒の価値以外の評価要素の重みはデフォルトの値のまま固定して学習を開始した。なお、この駒の価値が評価関数では最も影響の大きい要素であり、この状態で学習前の勝率は 0 である。

ここで盤上の歩の価値を固定したのは評価関数では駒得、駒損の要素が非常に大きいため、その全てを動かすと基準となる値がなくなり、ある駒の価値が大きくなると他の駒の価値も大きくなるというような現象が繰り返されるのを防ぐためである。

学習結果について盤上の基本駒の価値の学習の様子を図 8 に、終了時の値を表 3 に、盤上の成駒の価値を図 9 に、終了時の値を表 4 に、持ち駒の価値を図 10 に終了時点値を表 5 に示す。

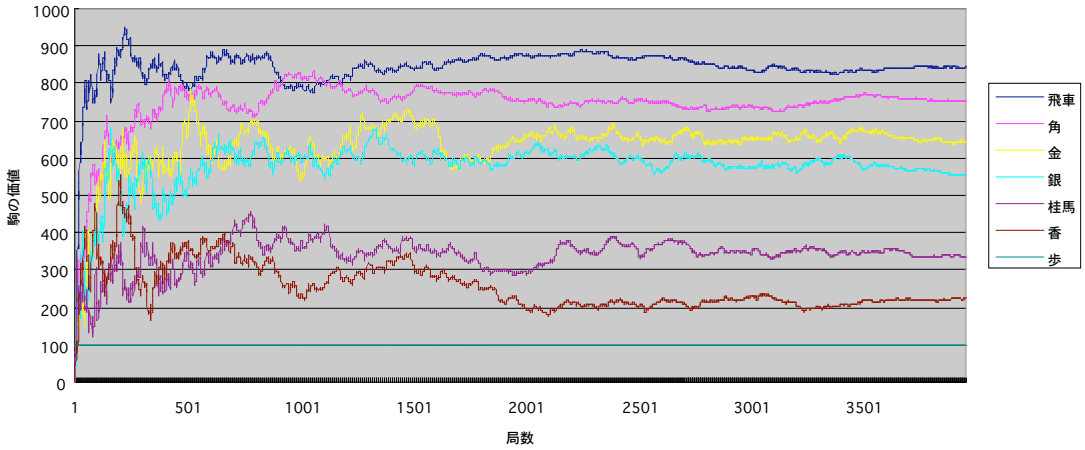

図8 盤上の駒の価値の学習の様子

表 3　学習後の盤上の駒の価値

	飛車	角	金	銀	桂馬	香車	歩(100 で固定)
学習後の価値	842	754	641	554	334	224	100
デフォルトの値	950	800	600	550	400	370	100
デフォルトとの比	0.88	0.94	1.07	1.01	0.84	0.61	1.00

盤上の駒の価値に関しては香車が低くなった以外はほぼデフォルトの『激指』、つまり人間の手で設定した値に近い値になった。

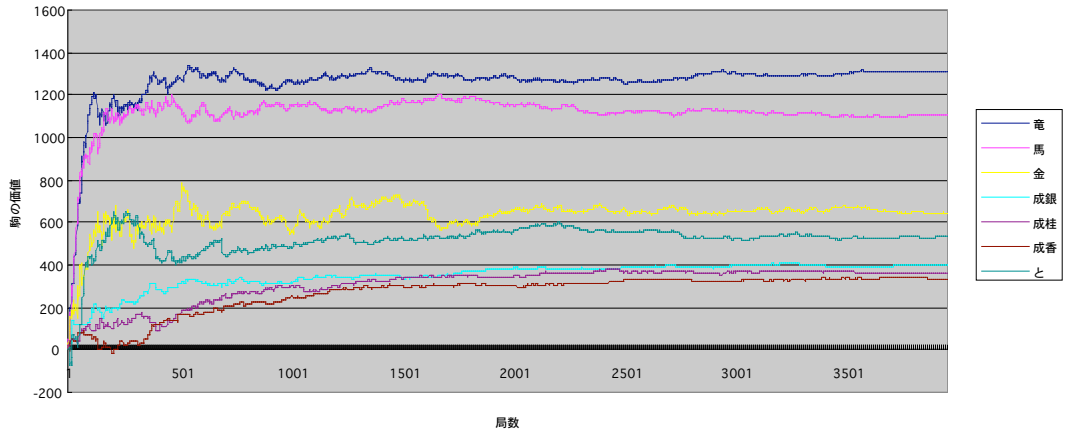

図9　盤上の成駒と金の価値の学習の様子

表 4　学習後の盤上の成駒および金の駒の価値

	竜	馬	金	成銀	成桂	成香	と
学習後の価値	1309	1104	641	399	361	336	533
デフォルトの値	1300	1150	600	600	600	600	600
デフォルトと比	1.01	0.96	1.07	0.67	0.60	0.56	0.89

成駒の価値については竜と馬に関してはデフォルトの『激指』の値とほぼ同じような値になったが、それ以外の成って金と同じ働きをする駒についてはかなり異なる結果となった。

ここでなぜこれらの価値がこのように異なる値になったのかについて考える。

これらの駒は盤上にあるときは同じ働きをするが、相手に取られて持ち駒になると価値は　金＞銀＞桂馬≒香車＞歩　なので、盤上にあるときの価値は　と＞成香≒成桂＞成銀

＞金　となると考えられる。しかし表に示すように今回の結果では金＞と＞成銀＞成桂＞成香となっている。まず、金の価値が高くなったのには金は守り駒としての価値が高いため、成って金になる駒は相手陣地にいることが多いが、金は自陣にいることも多く、自陣にいるときの価値の方が高いためではないかと考えられる。成銀、成桂、成香について逆になっているがこの間はあまり差がないことから、その差まで厳格には学習できなかったのではないかと考えられる。また、成銀、成桂、成香は他と比べて出現頻度の低い要素であるため学習が遅れていたということも考えられる。

また、金、成銀、成桂、成香、と、を同じ種類の駒としてデフォルトの『激指』と同じようにこれらが同じ価値になるようにして他の条件は同じで学習させたところこの金の価値は 490 となった。これはちょうど中間的な値であるので納得できるところである。

このときもデフォルトの『激指』と対戦させたが 1000 局程度対戦して勝率は 4 割 7 分程度であった。

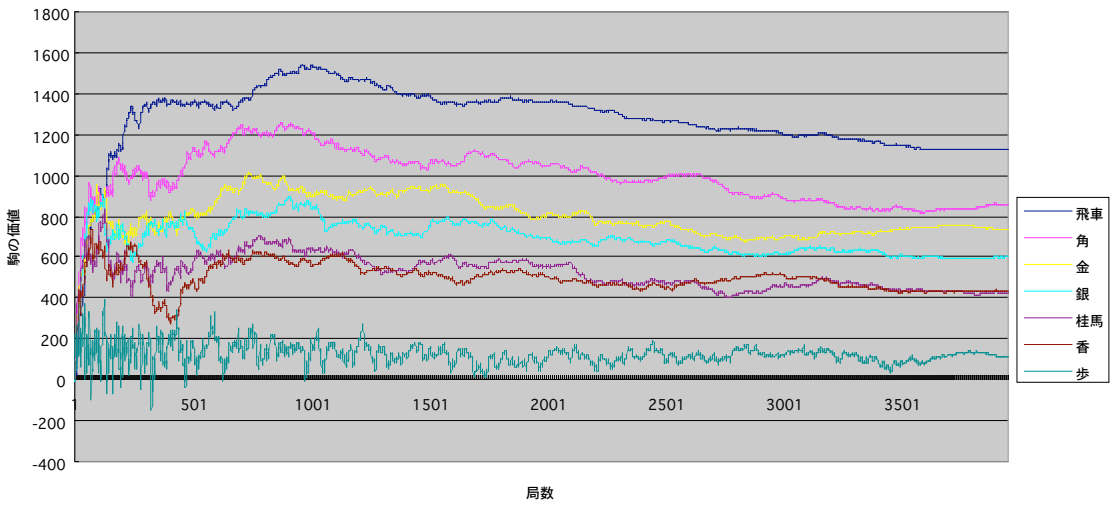

図10　持ち駒の価値の学習の様子

表 5　学習後の持駒の価値

	飛車	角	金	銀	桂馬	香車	歩
学習後の価値	1128	857	740	602	428	436	114
デフォルトの値	1150	1000	660	605	440	407	110
デフォルトと比	0.98	0.86	1.12	0.99	0.97	1.07	1.03

持ち駒の価値については角と金の差がやや小さいがほぼデフォルトの『激指』と同じような結果となった。

学習後、デフォルトの『激指』と対戦させて見ると 1000 局程度対戦して勝率は 4 割 8 分と互角からやや弱い程度となった。

また盤上の歩を固定しないでも学習を行ったが大差ない結果で勝率もやはり 4 割 8 分であった。

4.2 他の『激指』の既存評価要素の学習

次に『激指』が既に持っている評価要素(3.1 で示した③)のうち、盤面の進行度によって値を変えないもので、複雑な計算をせず、その評価要素の特徴量の定数倍だけ加えるという評価要素約 30 個について学習を行った。

学習させた評価要素以外の駒の価値やその他の評価要素の重みなどについてはデフォルトの値のままとして学習を行った。なお、学習前の勝率は 4 割 6 分であった。

学習した評価要素のうち、比較的絶対値が大きくなったものについて学習の様子を図 11 に学習後の値を表６に示す。

表 6　学習後の各評価要素の重み

評価要素	A	B	C	D	E	F	G	H
学習後の重み	-110	-99	-84	-102	-83	-95	-125	71
激指の設定	-20	-6	+15	-30	-10	-15	-50	-10

激指が設定している値と比べると絶対値の非常に大きな値となった。また正負が異なっている要素もあった。

学習後デフォルトの激指との勝率は 4 割 4 分と学習前より弱くなってしまった。

4.3 新たに加えた評価要素の学習

4.2 で駒の価値以外の『激指』に既存の評価要素の重みについての学習を行ったがうまくいかなかったことから、既存のパラメータは既にかなりいいところまで設定がなされているため、それに勝つというところまで学習できなかったため、うまくいかなかったのであろうと考え、新たな評価要素を追加し、その重みについて学習を行った。

加えた評価要素は王の周りにどんな駒があるかという要素である。王の周り 8 マスについて、8 マスを中央側横上、上、端側横上、中央側横、端側横、中央側横下、下、端側下というように王が右にいるか左にいるかで左右を反対にして(5 筋にいるときは便宜上左右両方中央側とした)8 マスを区別し、そこに味方のどの駒がいるかというのを評価要素とした。王以外の駒の種類は飛車、竜、角、馬、金、銀、成銀、桂馬、成桂、香車、成香、歩、と、の１３種類であるので８×13＝104 種類の評価要素を付け加えて学習した。ただし、穴熊の場合は他の場合とかなり周りの駒が異なるので除いて行った。

歩	歩	銀
	王	金
香	桂	

図 12　王の周りの駒

例えば図 12 の状態で王がいるのが先手で 1~4 筋(後手なら 6~9 筋)であれば中央側は右で

あり、端側は左であり、端側横上に歩、上に歩、中央側横上に銀、中央側横に金・・・という具合であり、王がいるのが先手で 6~9 筋(後手なら 1~4 筋)であれば中央側は左であり、端側は右であり、端側横上に銀、上に歩、中央側横上に歩、・・・となる。5 筋だとすると左右どちらも中央側とするので中央側横上に歩、上に歩、中央側横に銀、・・・となる。
なお、学習前はどちらもデフォルトの激指になるので勝率は 5 割である。

学習結果から比較的絶対値が大きくなったものについて結果を図 13 に示す。

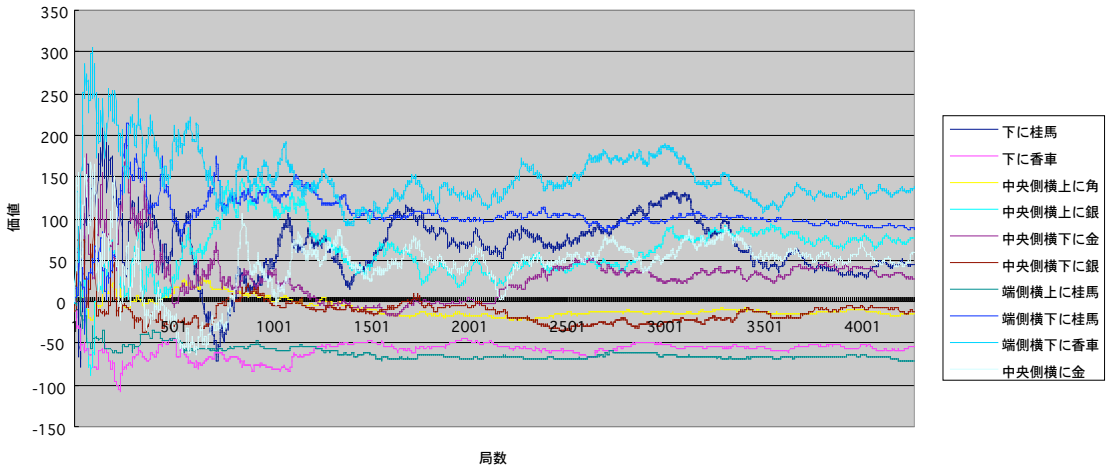

図13　王周りの駒の価値の学習の様子

学習後勝率は 4 割 6 分に落ちてしまっていた。

次にこの学習を駒の価値、駒の位置以外の 3.1 の③のパラメータを(学習側、デフォルト側両方)全て除いて、学習を行った。なお、学習前はどちらもデフォルトの激指から 3.1 の③を除いたものなので勝率は 5 割である。

その結果を図 14 に示す。

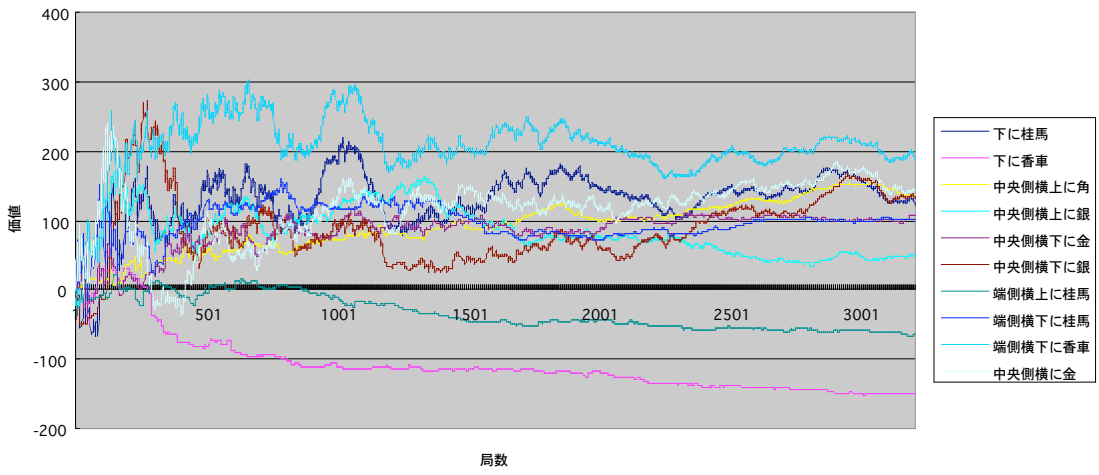

図14　王周りの駒の価値(駒価値以外の既存要素を0にした場合)

この場合勝率が 5 割 8 分と強くなった(相手も既存のパラメータなしの相手)。

駒の価値以外の評価要素を取り除いて学習を行うと強くなったということは学習自体は進んでいると考えられるが、しかし取り除かないで学習すると弱くなったというのは取り除く評価要素にも王の周りの駒に関係するような評価要素も入っているので、人間が設定した値ほどまで調整できないために人間が設定したバランスを壊すような形になってしまったと考えられる。

表 7　学習後の比較的重みの大きくなった評価要素

	下に桂馬	下に香車	中央側横上に角	中央側横上に銀	中央側横下に金	中央側横下に銀	端側横上に桂馬	端側横下に桂馬	端側横下に香車	中央側横に金
『激指』既存評価要素あり	44	-53	-15	78	27	-11	-71	87	138	60
『激指』既存評価要素なし	124	-151	137	51	106	132	-65	103	190	142
関係する囲い	矢倉	米長玉	左美濃	矢倉	舟囲い	天守閣美濃		金無双	矢倉	矢倉

『激指』既存パラメータなしとありとなしとで大きく異なるパラメータについて、『激指』既存の評価要素に関係するような評価要素がありそこで既に影響を受けているためと

考えられる。例えば、中央側横上に角、あるいは中央側横下に銀であれば、『激指』既存の評価要素に左美濃の時にプラスという評価値があり、その分小さかったりしたのではないかと考えられる。

駒の価値以外の既存評価要素なしで学習した評価要素の重みを見ると正で大きくなったものは、一般的に良く使われる囲いにある形であり、負で絶対値が大きくなったものはその形が出てくる囲いもあることにはあるが、一般的にあまり良くない形(例えば下に香車であれば、米長玉などあることにはあるが、一般的には玉が端に追われた状態であまり良くない)になっているのが分かり、ここからも学習できていることが分かる。

4.4 考察・検討

今回の実験では表のような結果を得た。

学習した評価要素	対戦相手	学習前の勝率	学習後の勝率	備考
駒の価値	デフォルトの『激指』	0	4 割 8 分	
『激指』の既存の評価要素	デフォルトの『激指』	4 割 6 分	4 割 4 分	
王周りの駒	デフォルトの『激指』	5 割	4 割 6 分	
王周りの駒	評価関数が駒の価値のみの『激指』	5 割	5 割 8 分	学習側も評価関数は駒の価値と王周りの駒のみ

駒の価値に関しては『激指』が設定している値とほぼ同じような結果が得られ、デフォルトの『激指』と対戦して勝率を 4 割 8 分まで持ってくることができたことに対しては学習は成功したといえる。しかしその他の要素に関してはデフォルトの『激指』に対して学習前よりも勝率が下がる結果となってしまった。まず、学習前より下がってしまった原因について考える。

王周りの駒の学習において、駒の価値以外の『激指』に既存の評価要素を取り除いて学習させると学習前より勝率が上がったことから学習自体は進んでいるということが分かる。駒の価値以外の『激指』に既存の評価要素がある状態で学習させると勝率が下がったというのは既存の評価要素で王の周りの駒の部分もうまく設定されていて、そのためにむしろ動かすことでそのうまく設定されたバランスを壊す結果となってしまったのではないかと考えられる。既存評価要素の学習で勝率が下がってしまったのも同じように学習に使った

既存評価要素以外の既存評価要素でもある程度バランスの取れていたのを壊す結果となってしまったのではないかと考えられる。

今回、このように学習した評価要素が人の設定したパラメータより劣る結果になった原因についてチェスで TDLEAF(λ)による学習で大きな成果を挙げた KnightCap との比較で考察することにする。

まずはチェスと将棋の違いであるが、その最も大きな違いは将棋には持ち駒があることであり、これにより特に中盤以降での着手可能な手が大幅に増える、駒の種類も多く評価関数も複雑になるがこれは TD 法を行う上では収束が遅くなるなどの影響があるかもしれないが根本的なところでは問題ないように思われる。

次に KnightCap の学習では自己対戦ではなく、インターネット上でさまざまな人と対戦して学習を行った。これにより自己対戦よりさまざまなパターンが得られよかったのではないかということが考えられる。

KnightCap では学習する要素を 1368 個と大量に学習させている。これが 1 つ 1 つでは人間が設定した値にはかなわないもののこれだけの量になると人間では手に負えなくなり、学習結果のほうが強くなったということも考えられる。

また、学習全体を通して、α の決め方に非常に苦労した。α を小さくする速度を速くしてしまうとパラメータが収束したように見えてもそれはただ α が小さいために動きが少なくなっただけで、その状態から α を大きく値に戻して学習してみると全然違うところに収束するというような現象が多く見られた。この現象を回避するために α を小さくする速度を遅くし、さらに学習終盤でもある程度の大きさを残した。しかし、もっといい設定の仕方があったのではないかということも考えられる。

5 おわりに

5.1 まとめ

本研究では、コンピュータチェスプレイヤでは大きな成果を挙げているが、コンピュータ将棋プレイヤではそれによって強くなったというほどの結果を挙げていない TD 法による静的評価関数の学習を現在最高レベルのコンピュータ将棋プレイヤの一つである『激指』に適用してみることで効果があるのかどうかを検証した。

これまで行われていたコンピュータ将棋プレイヤでの TD 法による学習は浅い全幅探索で行っているものや評価関数も単純化して行っているものがほとんどであり、駒の価値に関する学習についてはうまくいったという例もあったがそれ以外の評価要素の学習に成功し強くなったというような例は見られなかった。そこで本研究では『激指』の探索アルゴリズムを用いることによってより深くまで探索することにより、駒の価値、他の『激指』し既存評価要素、新たに加えた評価要素の学習を行った。結果として、駒の価値の学習においては人間が設定した値に近い値が得られ、学習後の勝率は 4 割 8 分程度であった。また『激指』の既存の評価要素の学習では学習前より勝率が下がる結果となった。新たな評価要素として王の周りの駒という評価要素の学習でデフォルトの『激指』に付け足すという形では勝率は下がってしまったが、駒の価値以外の評価要素を除いた評価関数に付け加えて学習させると同じく駒の価値以外の評価要素を除いた評価関数を用いた『激指』より強くすることに成功した。

これらの結果から多くの評価要素について人間の手でうまく設定した評価関数に対してはそれに勝ち越すほどの学習はできないものの、駒の価値以外の評価要素でも学習は進んでいるということがわかった。

5.2 課題

本研究で駒の価値以外の評価要素でもある程度学習はできるということはわかったが、人間が設定した値にはかなわなかった。人間が設定したものに対抗するためには、自己対戦ではなくさまざまな人を対戦相手としたほうが良いのではないか、α や λ をもっとうまく設定することによりよい学習結果が得られるのではないかということが考えられる。また、学習する評価要素を増やすことにより 1 つ 1 つの設定はそれほど良くなくても全体として人間の設定した値よりよい値が得られるのではないかということが考えられる。

謝辞

本研究にあたり、近山隆教授には論文の書き方や発表の仕方など多岐にわたりご指導していただきました。

田浦健次朗助教授には週 1 回卒業論文に関するミーティングを開いてくださり、研究を進める上での多くの助言をいただきました。

横山大作助手には『激指』のソースコードを提供していただきました。また、研究中の質問にも丁寧に答えてくださり、進め方の提案などもしていただきました。

また、研究室のみなさんにはいろいろお世話になりました。ありがとうございました。

参考文献

[1]Jonathan Baxter : Learning To Play Chess Using Temporal Differences.

[2]Richard S. Sutton, Andrew G.Barto 共著 三上貞芳,皆川雅章 共訳：強化学習(2000 年森北出版株式会社).

[3]北村圭：TD 法によるコンピュータ将棋の静的評価関数の学習(1999 年度東京大学工学部卒業論文).

[4]薄井克俊,鈴木豪,小谷善行:TD 法を用いた将棋の評価関数の学習(1999 年 GPW'99 資料).

[5]Akihiro Kishimoto : Transposition Table Driven Scheduling for Two-Player Games, M.Sc. Thesis, University of Alberta (Final version), January 2002.

[6]佐々木宣介,竹下信夫,橋本剛,飯田弘之：着手決定の複雑さの指標とゲームの進化論的変遷

[7]松原仁,竹内郁雄：ゲームプログラミング(1998 年 共立出版)