2009－GI－21 (5)
2009／3／9

囲碁における勾配法を用いた確率関数の学習

松井利樹[†1] 野口陽来[†1]
土井佑紀[†1] 橋本剛[†1]

囲碁の局面評価は非常に複雑であり特徴の数は膨大となる．そのため膨大なパラメータを人間が調整していく事は困難を極める．そこで本稿は機械学習のアプローチからこの問題を解決する．学習手法にはコンピュータ将棋において評価関数の学習に用いられている勾配法をコンピュータ囲碁に応用する．将棋と囲碁の学習で決定的に異なる点は，将棋では手の順序関係を学習できれば十分であったが，囲碁では確率分布を生成しなければいけないため比率まで考慮しなければいけない点にある．本稿では異なる二つの誤差関数を設計する事でこの問題を解決している．ベンチマークとして，現在世界最強の囲碁プログラムである Crazy Stone で用いられている学習手法である「小数化-最大化」と比較実験を行った結果，大きな性能向上を確認できた．

Machine Learning of Probability Function in Game of Go by Gradient Method

Toshiki Matsui,[†1] Haruki Noguchi,[†1] Yuki Doi[†1]
and Tsuyoshi Hashimoto[†1]

Programmer cannot make accurate an evaluation function because of enormous number of features. we solve that problem with a machine learning approach. we present gradient method for supervised learning of such an evaluation function from game records, based on a learning method for computer Shogi's an evaluation function. It is enough to learn order relation of move in Shogi while in Go move evaluation should also be learned to generate a probability distribution. we solved these problem through the design of two error functions. we compare that method with Minorization-maximization that was used by Crazy Stone (the best program in the world). As a result of experiment, our method is more efficient than previous approaches,

1. はじめに

ゲームにおける機械学習の多くのターゲットは評価関数の自動調整にある．古くはバックギャモン[4] やオセロ[5] などに学習が応用され一定の成果を出している．しかし将棋や囲碁などの難しいゲームでは複雑な局面をうまく学習する事は困難であった．そのため大部分のゲームでは評価関数はプログラマの手で設計するのが常であった．そのような状況下で 2005 年に世界コンピュータ将棋選手権において，局面評価関数を強化学習によって調整された Bonanza が優勝し，将棋の強化学習の走りとなった[1]．またコンピュータ囲碁でも *Rémi Coulom* の『Computing Elo Ratings of Move Patterns in the Game of Go』[2] によって Elo Rating の概念と少数化-最大化アルゴリズムを用いた学習手法が考案され (以後少数化-最大化と呼ぶ)，囲碁における評価関数の自動調整を実現している．彼によって作られた Crazy Stone は現在世界最強の囲碁プログラムとなっている．彼らの活躍は将棋や囲碁のような難しいゲームにおいても強化学習を用いて強いプログラムを作る事が可能であるという事を体現した．モンテカルロ囲碁の誕生以来，囲碁は現在大きな盛り上がりを見せており，今後のさらなる躍進の原動力として大きな期待をされているのがこの機械学習である．囲碁ではゲームの特性上，評価関数の特徴の数が膨大になりがちで手作りによる調整は困難である．そこで機械学習によるアプローチが考えられるわけだが，優れた学習を実現することは難しい[6][7]．優れた学習に成功したプログラムはその強さにおいて大きなアドバンテージを握ることになる．囲碁の機械学習で最も大きな成果を挙げたのは前述した少数化-最大化であり，本稿では将棋で成功した勾配法によるアプローチを使って少数化-最大化を上回る学習システムの構築を目指す．

†1 北陸先端科学技術大学院大学
japan advanced institute of science and technology

2. 確率関数の設計

モンテカルロ囲碁では局面評価にモンテカルロ法を利用するため，局面評価関数は必要ない．囲碁における評価関数は局面評価関数ではなく手の評価関数である．手の評価値から木探索中での枝刈りやプレイアウト中での確率分布の生成[3]を行う．我々が設計するモンテカルロ囲碁の評価関数は従来のゲームと役割が異なり，ある局面から次の局面への遷移確率を計算するのに用いられる．そのためほとんどのゲームで評価関数は線形和で与えられるが，囲碁では確率を表現しやすい積の形を採用する．確率を表現する上で積の方が優れている要因として確率を生成しやすい事があげられる．積は評価値が否負であるため確率を生成しやすく，値域も容易に拡大できるため評価の強弱をつけやすい．ある手 p が与えられた時の評価関数 $\gamma(p)$ は以下のように定義される[2]．

$$\gamma_{\mathbf{x}}(p) = \prod_{k=1}^{K} l_k(p)$$

ある手 p が与えられた時，特徴があるかないか判定する特徴関数 $l_k(p)$ は以下のように定義される．

$$l_k(p) := \begin{cases} x_k & (\text{特徴がある}) \\ 1 & (\text{特徴がない}) \end{cases}$$

パラメータベクトルである $\mathbf{x}$ は以下のように表記される．

$$\mathbf{x} := \left[\begin{array}{cccc} x_1, & \cdots & , x_K \end{array} \right]^T > \mathbf{0}$$

ある局面からある手 p_i が打たれる確率を求める確率関数 $f_{\mathbf{x}}(p_i)$ にはルーレット選択によるアプローチを採用し以下のように定義する．

$$f_{\mathbf{x}}(p_i) := \frac{\gamma_{\mathbf{x}}(p_i)}{\sum_{m=0}^{M} \gamma_{\mathbf{x}}(p_m)}$$

3. 特徴の設計

囲碁の特徴は大まかにいうと，「直前手からの距離」のようなヒストリー，「ノビ」「アタリ」のような囲碁の重要な分野知識，そして「着手点周辺の石の形」でできている．直前手からの距離は囲碁に特に強く現れる特徴で，囲碁は直前手の近辺が最善手である確率が非常に高く，重要な特徴となっている．二つの座標間の距離 d を以下の式で数値化する．

$$d(\delta x, \delta y) = |\delta x| + |\delta y| + max(|\delta x|, |\delta y|)$$

これから設計する特徴の具体的な数値は表.1 に掲載する．

I:着手点の座標

天元 (盤の中心) と着手点との距離 d を計算して $length$ で場合分けする．

II :直前手からの距離

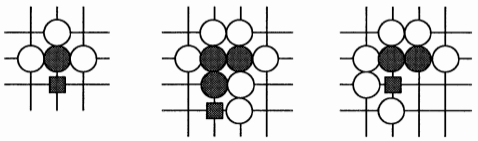
直前手と着手点の距離 d を求め，$length$ で場合分けして計算する．

III :二手前からの距離

二手前の手と着手点の距離 d を求め，$length$ で場合分けして計算する．

IV:ノビに関する特徴

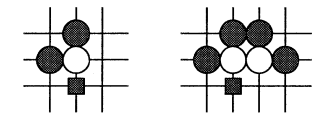
ここで言うノビとはアタリになっている連だけを指すのではない．隣接する連の $liberty$ の数が (1,2) の時に計算される．対象連の石の数 $stone$ と着手すると対象連の $liberty$ がいくつ増えるか up で場合分けする．



$stone = 1$	$stone = 3$	$stone = 2$
$liberty = 1$	$liberty = 2$	$liberty = 2$
$up = 2$	$up = 1$	$up = -1$

V:アタリに関する特徴

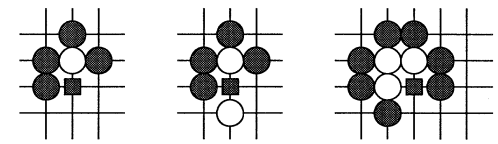
隣接する相手の連の $liberty$ の数が 2 の時に計算される．対象連の石の数 $stone$ によって場合分けする．



$stone = 1$　　$stone = 2$

VI:トリに関する特徴

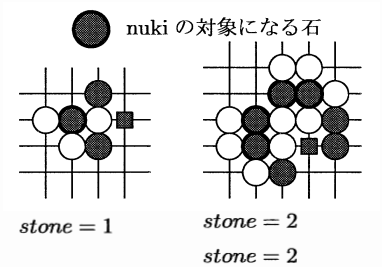
隣接する相手の連の $liberty$ の数が 1 の時に計算される．対象連の石の数 $stone$ と相手が着手点に着手したとすると対象連の $liberty$ がいくつ増えるか up によって場合分けする．



$stone = 1$	$stone = 1$	$stone = 3$
$up = 2$	$up = 3$	$up = 0$

VII :ヌキに関する特徴

隣接する相手の連の $liberty$ の数が 1 の時に計算される．着手によって取る事が出来る相手の連が自分の石を当ててる場合にのみ計算される．複数の連を当てている時はそのつど計算される．当てられている自分の連の石の数 $stone$ によって場合分けする．



VIII :着手点周囲の形の特徴

着手点からの距離 d が $size$ 以下である範囲の石の配置を特徴に組み込む．(自分の石，相手の石，空白 or 盤端) という三種類を考慮し，位置情報を石の形に付加する．これより石の形に位置情報を与える事ができるのだが，特徴の組み合わせが大きくなってしまうのが欠点である．そこで回転と反転を考慮する事で，右図上 A,D のような石の形は異なるが本質的に同じ特徴を同一視する．

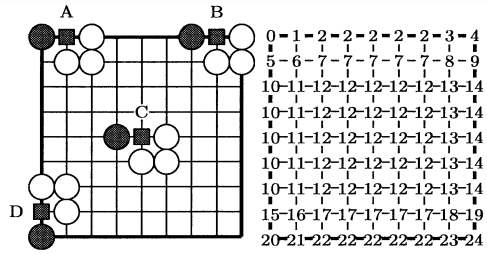


図 1 パターンと位置情報

囲碁のパターンで考慮すべき回転と反転は，基準となる形 (**0**)，基準となる形から 90° 回転させた形 (**1**)，180° 回転させた形 (**2**)，270° 回転させた形 (**3**) とそれら四つの図形をそれぞれ反転させた形 (**4,5,6,7**) の計 8 パターンを考慮すれば十分である．これら 8 パターンを盤面の各座標に分類したのが図 2 である．それぞれのパターンのユニークナンバー毎に始点と左回りか右回りか異なり，始点から終点までなぞりながら図 4 のように φ を計算する事で回転と反転の処理を行う事が出来る．計算した φ に図 3 の位置情報を付加して特徴が生成される．特徴の組み合わせは同じ情報を扱いながら九路では 76% も減らすことに成功している．また，このアルゴリズムは配列参照が一度生じるだけなので非常に高速である．

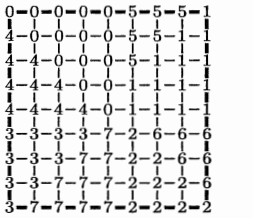


図 2 回転と反転のテーブル

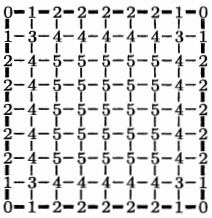
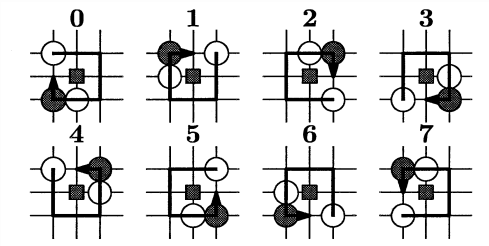
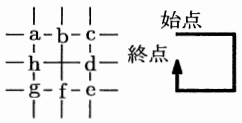


図 3 位置情報のテーブル



上の形は全て $\varphi = 1702$ で同一の形と認識される



$$\varphi = p(a)\cdot 3^0 + p(b)\cdot 3^1 + p(c)\cdot 3^2 + p(d)\cdot 3^3 + p(e)\cdot 3^4 + p(f)\cdot 3^5 + p(g)\cdot 3^6 + p(i)\cdot 3^7$$

$$p(x) := \begin{cases} 0 & (x = \text{空白 or 盤端}) \\ 1 & (x = \text{相手の石}) \\ 2 & (x = \text{自分の石}) \end{cases}$$

図 4 回転と反転を考慮するアルゴリズム

上記の 8 項目から特徴を設計する，木探索中で用いる評価関数とプレイアウト中で用いる評価関数は求められる速度が違うため，それぞれ表 1 のように設計した．

表 1　特徴ベクトルの設計

	UCT(九路)	MC(九路)
I	$length = (1, 2, \ldots, 15$ 以上)	$length = (1, 2, \ldots, 15$ 以上)
II	$length = (1, 2, \ldots, 9$ 以上)	$length = (1, 2, \ldots, 3$ 以上)
III	$length = (1, 2, \ldots, 9$ 以上)	-
IV	$stone = (1, 2, 3$ 以上) $up = (-1, 0, \ldots, 2$ 以上)	$stone = (1, 2, 3$ 以上) $up = (-1, 0, \ldots, 2$ 以上)
V	$stone = (1, 2, 3$ 以上)	$stone = (1, 2, 3$ 以上)
VI	$stone = (1, 2, 3$ 以上) $up = (-1, 0, \ldots, 2$ 以上)	$stone = (1, 2, 3$ 以上) $up = (-1, 0, \ldots, 2$ 以上)
VII	$stone = (1, 2, 3$ 以上)	$stone = (1, 2, 3$ 以上)
VIII	$size = 4$	$size = 3$

4. 学習の設計

棋譜を教師信号とした強化学習では，棋譜からは「どの手が一番優れている」という情報 (**順序**) しか得られないため，棋譜で選ばれた手 (**教師手**) を最良とみなし，その他の手を教師手よりも小さくするという学習が行われる．これにより，教師手と確率関数が選択する上位 n 位の手が一致する確率 (**一致率**) を最大化する評価関数の発見が可能になる．しかし，優れた評価関数の設計のためには順序だけを理解するのは十分でなく，各手がどれくらいの強さを持つか (**比率**) という情報が必要になってくる．だが，比率の情報は棋譜から与えられないので何らかの指標を使って解決しなければいけない．将棋では駒の価値という重要な指標があり，経験的にどの程度のスコアであるか知られていたため，駒の価値を基準に各パラメータの比率を与える事が可能だった．しかし，囲碁では駒の価値のような絶対的価値が知られておらず，どの手がどれくらいのスコアを持つべきかという事を理解するのは簡単ではない．我々のアプローチでは手の評価値から確率を生成するので，各手に割り振られる確率をどういった分布にするべきかという議論が必要になる．例えば，棋譜との一致率が非常に高く，エキスパートプレイヤーに匹敵する精度を誇る場合，各手に与える確率がほとんど一様なのは不適切である．逆に棋譜との一致率が非常に悪くランダムとほとんど変わらない場合，各手の確率に大きな差を与える事もまた適切でない．本稿では「**各手に与えるべき確率は棋譜との一致率と大きな相関がある**」と考え，一致率を基準に各手の比率を調整するという方針を立てる．一致率の学習は比率が与えられなくても行う事ができるので，学習は「強いプレイヤーの手との一致率を高める順序の学習を行い，一致率に合わせて適切な確率分布を選択するために比率の学習を行う」という指針で行う．そこで「**順序の最適化**」「**比率の最適化**」の二つの最適化から $f_{\mathbf{x}}(p_i)$ を最適化する．順序の最適化は 5 項で，比率の最適化は 6 項でそれぞれ解説する．

5. 順序の最適化

5.1 誤差関数

棋譜中で選択された手はその局面で選ばれなかったどの手よりも価値が高いとして，棋譜中で選択された手の確率を最大化する事を考える．棋譜サンプル中のある局面 p_s でプレイヤが選択した手 $p_{s,0}$ を教師信号とし，教師信号以外の全ての手 $p_{s,m}$ を訓練集合とすると，誤差関数 $J_{\mathbf{x}}(S)$ は以下のように定義するすm

$$J_{\mathbf{x}}(S) = \sum_{s=1}^{S} \sum_{m=1}^{M-1} T[f_{\mathbf{x}}(p_{s,m}) - f_{\mathbf{x}}(p_{s,0})] + \Phi_{\mathbf{x}}$$

S はサンプル数で，M はサンプル局面 p_s から選択できる手の数である．誤差を定量化する誤差汎化関数 $T[\cdot]$ については 5.4.3 項で説明し，拘束条件 $\Phi_{\mathbf{x}}$ については 5.4.4 項で説明する．

5.2 誤差汎化関数

誤差汎化関数にはシグモイド関数を使うことにする．

$$T[x] = sig(x) = \frac{1}{1 - e^{\eta x}}$$

$$sig'(x) = \eta \cdot sig(x)(1 - sig(x))$$

教師信号の中には，

- 悪手と呼ばれる不正解の手 (誤差)
- 難解な手 (与えられた特徴ベクトルでは表現が不可能)
- 既に理解している絶対の一手

が存在する．それらを考慮せずに学習を行うと過学習を起こしたり，不安定な学習になってしまう．そこでそれらの悪影響を軽減するために，教師値に近い手を優先して学習し教師値から大きく離れた手はあまり学習したくないという性質が好ましいと考えた時，ぴったりなのがシグモイド関数である．一般的に，S 字単調増加関数はある集合をある領域とある領域に分離したい時によく使われる．今回の学習では教師値より悪い手と良い手を二つに分離する事で，安定した学習を実現している．

5.3 勾配ベクトル

パラメータベクトル $\mathbf{x}$ による誤差関数 $J_{\mathbf{x}}(S)$ の偏

微分である勾配ベクトルは以下のようになる．

$$\frac{\partial}{\partial \mathbf{x}} J_{\mathbf{x}}(S) = \sum_{s=1}^{S} \sum_{m=1}^{M-1} \left[\frac{\partial}{\partial \mathbf{x}} J_{\mathbf{x}}(S) \right] + \frac{\partial}{\partial \mathbf{x}} \Phi$$

$$\frac{\partial}{\partial \mathbf{x}} \Phi = \left[\begin{array}{ccc} \phi_1, & \cdots & , \phi_K \end{array} \right]^T$$

5.4 拘束条件

値をある領域に抑えるために拘束条件 $\phi_{\mathbf{x}}$ を付加する必要がある．出現頻度の高いものは勾配の絶対値が大きいため拘束の影響を受けにくくなるため，サンプル毎の出現回数を考慮する必要がある．

$$\phi_k = w \cdot \left(x_k - \frac{1}{x_k} \right) \cdot \rho(x_k)$$

w はペナルティの強さを決める定数であり，$\rho(x_k)$ は全サンプル中 x_k が出現した回数を返す関数である．この関数はパラメータを 1 の方向に引き寄せる挙動をする．拘束条件を付加すると棋譜との一致率は減少してしまうが，その代わり学習が安定する．

5.5 パラメータの更新

上記の勾配ベクトルを計算し，勾配方向に従って反復的にパラメータベクトルを更新していく事で値が収束する．

$$x_k^{N+1} = \left\{ \begin{array}{ll} \Upsilon[x_k^N] & (\rho(x_k) > \lambda) \\ 1 & (\rho(x_k) \leq \lambda) \end{array} \right.$$

特徴サンプルが非常に少ないものをそのまま学習に加えると，学習が不安定になってしまう。そこで特徴サンプルの数が λ を超えない限り，学習は行わず 1 にするようにした．値の更新を行う関数 $\Upsilon[\cdot]$ については，6 項で設計する．

6. 比率の最適化

6.1 誤差関数の設計

各局面 p_s で，棋譜で選択された手の順位が $r_{s,m}$ (手 $p_{s,m}$ の確率がその局面で上位何番目であるか) である確率を全サンプルで算出したものを，$\chi(r)$ とする．ここで $\chi(r)$ **が最適な確率分布であるとして**，累積確率分布 (各局面 p_s でそれぞれの手の順位 $r_{s,m}$ を求め，順位毎に手の確率の平均を全サンプルで算出したもの) が $\chi(r)$ と一致する事を目指す．この $\chi(r)$ と確率関数との誤差関数 $L_{\mathbf{z}}(S)$ を以下のように定義した．

$$L_{\mathbf{z}}(S) := \sum_{s=1}^{S} \sum_{m=1}^{M-1} [f_{\mathbf{z}}(p_{s,m}) - \chi(r_{s,m})]^2$$

これは，例えば $r_{s,m} = 1$ の手が棋譜で選択される確率が 35% の時，$r_{s,m} = 1$ の手の確率が平均的に 35% である事を要求するという意味である．局所的に確率が上下する事を許容することで，絶対の一手や，決定的な有効手でない手を表現できる．誤差汎化関数には単純に自乗誤差を使用する．

フィルターの設計

ここで $\mathbf{z}$ とはパラメータベクトル $\mathbf{x}$ をフィルタリングしたものである．このフィルタを調整する事で累積確率分布の最適化を実現する．この時，フィルタの特性として以下の性質を満たさなければならない．

- $\chi(r)$ が $\mathbf{z}$, $\mathbf{x}$ とも変わらない (順位が保証される)
- 値が正のままである (確率の生成に支障を与えない)

これを実現するには単純にパラメータの各要素を累乗すればよい．評価関数 $\gamma(p)$ は積和であるから，上記の条件を満たす事が出来る．よって $\mathbf{z}$ は以下のように定義される．

$$\mathbf{z} := \left[\begin{array}{ccc} z_1 & \cdots & z_K \end{array} \right]^T > \mathbf{0}$$

$$z_k^N = (x_k^N)^{\xi}$$

結局，フィルター係数である ξ を調整する事により，学習を実現できる．

6.2 勾配ベクトル

ξ を調整する事によって誤差関数を最小化するので，誤差関数 $L_{\mathbf{z}}(\mathbf{s})$ を ξ で微分した $\Delta_{\xi} J_{\mathbf{z}}(\mathbf{s})$ は以下のようになる．

$$\Delta_{\xi} L_{\mathbf{z}}(\mathbf{s}) = \sum_{s=1}^{S} \sum_{m=0}^{M} c \cdot (f_{\mathbf{z}}(p_{s,m}) - \chi(r_{s,m}))$$

$$c = 2 \Delta_{\xi} f_{\mathbf{z}}(p_{s,m})$$

この c を計算するのは非常に骨が折れるわりにはあまり変化がないので適当な定数にした ($c > 0$)．この勾配ベクトルを計算し，次項で提案する手法を使って反復学習する．

7. 学習の収束法について

勾配法を使用した囲碁や将棋の評価関数の学習は大規模な非線形計画問題になりがちで収束速度はしばしば重要な問題となる．これらの最適化問題は学習サンプルが膨大である，複雑な制約式が組み込まれている，誤差関数が十分に滑らかでない，各パラメータが従属しているなどの理由で優れた収束は非常に難しい．そこでこのような難解な非線形計画問題でも有効に作用する新たなアルゴリズムを提案し，既存の手法である SGD 法や最急降下法と性能比較を行いその有効性を検証する．

7.1 事前研究

筆者は過去に将棋に対して勾配法を用いた学習を行っており，そこでこれから提案される手法を実行し成功を収めた経緯がある．値を更新するに当たって，将棋と囲碁で決定的に異なるのは評価関数が線形和と積という事である．具体的には，$x_k^N = 1$ の時，$x_k^{N+1} = x_k^N + 1$ と値を更新するのと $x_k^N = 100$ の時，$x_k^{N+1} = x_k^N + 1$ と更新する事は，線形和では同じ増

加量として扱うことが可能だが，積では前者は 100% の増加，後者は 1% の増加となるため全く意味が異なる．そのため同じように更新していては値は全く収束しない．重要なのは「線形和では値の更新は加減」「積では値の更新は乗算」で更新を行うという事である．繊細な調整が必要なのは値域が広くなる積の方であり，今回の議論は囲碁のドメインの方がより重要である．

7.2 各手法のアルゴリズム

確率的勾配降下法, 最急降下法, 提案手法のアルゴリズムをそれぞれ解説する．

(1) 確率的勾配降下法(stochastic gradient descent method SGD 法)

求めた勾配ベクトルから線形に刻み幅を与える手法．各特徴が勾配ベクトルをもとに変動するので，勾配ベクトルが滑らかな時には非常に有効に働く．

$$x_k^{N+1} = \begin{cases} x_k^N \cdot (1 + a \cdot |\partial_{x_k} J_{\mathbf{x}}^N|) & (\partial_{x_k} J_{\mathbf{x}}^N > 0) \\ x_k^N / (1 + a \cdot |\partial_{x_k} J_{\mathbf{x}}^N|) & (\partial_{x_k} J_{\mathbf{x}}^N < 0) \end{cases}$$

刻み幅 a の値は最適化中変化しない．優れた収束を得るためには適切な a の設定が必要である．

(2) 最急降下法(steepest gradient)

勾配方向によって値を定数で更新する．全ての特徴に対して同一の刻み幅で更新する．

$$x_k^{N+1} = \begin{cases} x_k^N \cdot (1 + a) & (\partial_{x_k} J_{\mathbf{x}}^N > 0) \\ x_k^N / (1 + a) & (\partial_{x_k} J_{\mathbf{x}}^N < 0) \end{cases}$$

精度を上げるためには a の初期値を大きめに取りじょじょに小さくする必要がある．

(3) 提案手法

最急降下法において特徴ごとに刻み幅を持たせた手法である．特徴は前回得られた勾配ベクトルと今回得られた勾配ベクトルの符号の変化によって刻み幅の調整を行うことにある．符合が反転した時に刻み幅を小さくし，符号が等しい時に刻み幅を大きくする事で安定して収束する．

$$x_k^{N+1} = \begin{cases} x_k^N \cdot (1 + a_i^N) & (\partial_{x_k} J_{\mathbf{x}}^N > 0) \\ x_k^N / (1 + a_i^N) & (\partial_{x_k} J_{\mathbf{x}}^N < 0) \end{cases}$$

a の値は以下のようにステップ毎に変化させていく．

$$t_k = \partial_{x_k} J_{\mathbf{x}}^N \cdot \partial_{x_k} J_{\mathbf{x}}^{N+1}$$

$$a_k^{N+1} = \begin{cases} a_k^N \cdot \beta (t_k > 0) \\ a_k^N / \alpha (t_k < 0) \end{cases}$$

$$\alpha > 1, \beta > 1, \alpha > \beta$$

優れた収束のためには α,β を適当に設定する必要がある．β は刻み幅を大きくし，α は刻み幅を小さくする．$\alpha > \beta$ とは増加量が減少量を超える事は許さないという事である．

7.3 実験結果

収束の指標には成功率を使う．成功率とは全サンプル中で, 教師信号よりも悪いと判断できた訓練信号の割合である．実験結果をまとめると図 5 になる．それぞれの実験環境とその結果を示す．

- SGD 法
  設計した誤差関数は特徴によって出現頻度が大きく異なるため一般的な式のままではまったく収束しなかった．そこで勾配ベクトルの要素ごとに要素の出現回数に対する平均値を求めて使用することにした．結果として SGD 法は成功率が非常に悪かった．初期収束はもっとも優れており最初の数ステップは一番効果が高い．
- 最急降下法
  成功率は優れているが, 収束速度がもっとも悪い．また刻み幅の設定はかなり煩雑であり, 優れた収束を実現するためには実験を重ねる必要がある．
- 提案手法
  収束速度, 成功率ともに優れた結果を出した．いくつか刻み幅ベクトルから適当に要素を選んで刻み幅が反復毎にどのように変化していくかを図 6 に示した．最初は大きく学習され，じょじょに学習率が小さくなる事がみてとれる．

7.4 考　　察

SGD 法の収束が悪かったのは，今回の誤差関数が十分に滑らかではない事が原因だと考えられる．パラメータが停留点に十分に近い時でも勾配ベクトルが十分に小さくならないためパラメータの微妙な調整ができずに収束速度, 成功率ともに劣化してしまったと思われる．対して提案手法は, 特徴ごとに刻み幅を持たせ勾配方向をもとに調整する事で, 誤差関数が滑らかでない場合でも有効な刻み幅を与える事ができるため優れた収束速度を実現したといえる．

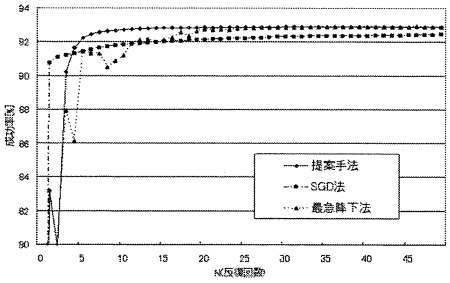


図 5 各手法の収束比較

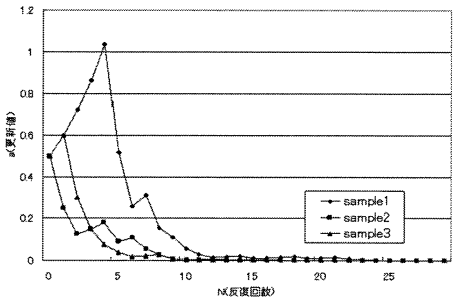


図 6 更新幅の変化の様子

8. 実験と考察

8.1 実験環境

プログラマが学習システムに与えなければいけない環境は以下の通りである.

- 特徴関数
  3 項で述べた.
- 棋譜サンプル
  九路盤
  学習に使う棋譜サンプルは九路盤は強いプレイヤーの棋譜があまり存在しないので, 十分に強いプログラム (CGOS で Rating2200 以上) 同士の対局の棋譜を使用した. この Rating であれば人間プレイヤーと比較しても十分強い. しかしコンピュータ同士の対局は勝負が決まっても終局まで打ち続けてしまうため, 終盤の手は非常に精度が悪い. そこで棋譜サンプルは初手から 45 手までとした.
  十九路盤
  プロの棋譜およそ四万棋譜を初手から終局まで使用した.
- η(sigmoid scale)
- ω(penalty scale)
- λ(sample scale)
- α, β, a(larning scale)

9. 結果と考察

ここでの結果は全て九路盤での結果である. 十九路盤でもほとんど同様の傾向となる.

成功率と一致率

図 7, 図 8 より成功率 (教師手以外の手が教師手よりも評価値が低いと判断できる確率), 一致率 (教師手と確率関数が選択する上位 n 位の手が一致する確率) は両方とも勾配法の方が若干優れているようだ. これは勾配法の方が過学習が起こりにくく, 値の調整を堅調に行えているからだと思われる. 特に勾配法は周囲のパラメータの変化によって停留点の位置が移動した時にも, 滑らかに調整を行う事が出来る利点がある.

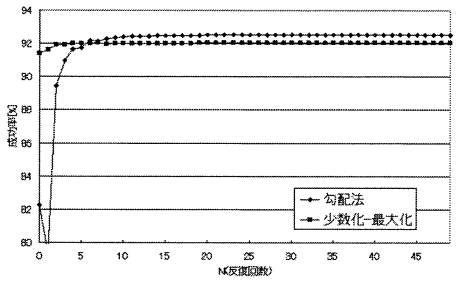


図 7 成功率の比較

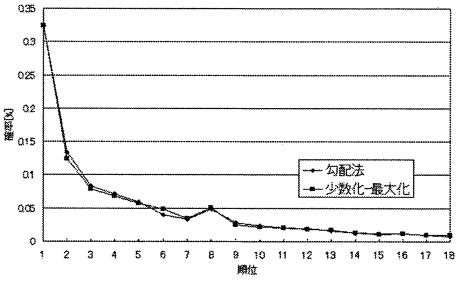


図 8 一致率の比較

一致率と累積確率

累積確率と棋譜との一致度の比較だが, 図 9 より, 勾配法では順位だけの最適化と棋譜との一致率を比べると大きな乖離がみられ, 過小評価してしまっている事がわかる. しかし, 比率を調整する事によってその弱点を克服できることが見て取れる. 図 10 より少数化-最大化ではほぼ完全な一致を見せている. それは少数化-最大化はそういった挙動を内包しているアルゴリズムであり, その意味で勾配法を凌駕している.

表 2　設定した変数

	サンプル局面数	η	ω	λ	α	β	a
九路盤	808679	1000	$1 \cdot 10^{-10}$	100	1.8	1.2	0.5

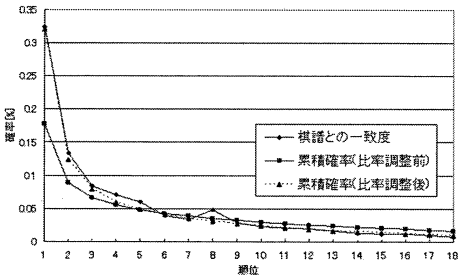


図 9　勾配法の累積確率と一致率の関係

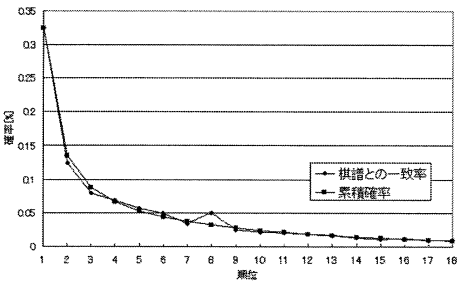


図 10　少数化最大化の累積確率と一致率の関係

学習にかかる時間

一回の反復にかかる時間は，勾配法が 30% 程度遅い．また収束に必要な反復回数も勾配法の方が 2 倍程度必要になる．総じて勾配法は少数化-最大化と比べて三倍程度の時間がかかるようである．学習にかかる時間で重要なのは，棋譜サンプル数と特徴数との依存関係であり，これら二つの変数に対して少数化-最大化，勾配法は収束にかかる時間は指数的に増えることは無い．規模に対して線形な時間の増加で対応できるため，大規模な学習に対しても柔軟性がある．

勾配法 *vs* 少数化-最大化

我々が開発した Nomitan に，勾配法と少数化-最大化アルゴリズムによって学習された評価関数を組み込み対戦を行った．勾配法は図 2 にあるパラメータを使って学習した．勾配法，少数化-最大化とも使用する特徴は同一である．200 回対戦を行ったところ，**142** 勝 **58** 負と大きく勝ち越すことが出来た．

10. ま と め

結果より，提案した学習法はコンピュータ囲碁で非常に有効に働く事がわかる．本稿で設計した学習法は，少数化-最大化の利点を全て併せ持ちながら，より高精度な学習を行う事ができる．しかし，勾配法は自由度が高く様々な改良を行える反面，プログラマに対して繊細な調整を要求する変数や設定が多く，安定した収束を実現するには修練が必要である．そういった意味では少数化-最大化と比べて敷居が高く煩雑である．結論として本稿で提案した学習法は今まで数々の研究が行われてきたコンピュータ囲碁における確率分布の学習の中でもっとも有望な手法であると言えるだろう．

参 考 文 献

1) 保木邦仁, 局面評価の学習を目指した探索結果の最適制御，第 11 回ゲームプログラミングワークショップ，pp.78-83 (2006).
2) Coulom, R.,Computing Elo Ratings of Move Patterns in the Game of Go, In Computer Game Workshop, Amsterdam, The Netherlands(2007).
3) Sylvain Gelly, Yizao Wang, Remi Munos, Olivier Teytaud, Modifications of UCT with Patterns in Monte-Carlo Go, Technical Report RR-6062,INRIA (2006).
4) G. Tesauro. Temporal difference learning and TD-Gammon. Communications of the ACM, Vol. 38, No. 3, pp. 58-68, (1995).
5) M. Buro, Experiments with Multi-ProbCut and a New High-Quality Evaluation Function for Othello, Games in AI Research, H.J. van den Herik, ISBN, 90-621-6416-1, (2000)
6) Bruno Bouzy and Guillaume Chaslot. Bayesian generation and integration of K-nearest-neighbor patterns for 19x19 Go. In G.Kendall and Simon Luas, editors, IEEE Symposium on Computational Intelligene in Games, pages 176-181, Colh-ester, UK, (2005).
7) David Stern, Ralf Herbrih, and Thore Graepel. Bayesian pattern ranking for move prediction in the game of Go. In Proceedings of the 23rd international onferencon Mahine learning, pages 873-880, Pittsburgh, Pennsylvania, USA, (2006).