将棋の棋譜を利用した，大規模な評価関数の調整

金 子 知 適 † 山 口 和 紀 †

棋譜を教師とした評価関数の調整は古くから研究されてきたが，将棋のような複雑なゲームで実用となったのは最近のことで，明らかになっていない点も未だ多い．現在有望とされる手法は，棋譜の指手が選ばれるように評価関数を調整するもので，探索により最善応手手順を求めることと，最善応手手順後の局面を比較して評価値を調整することを繰り返す．本稿では一般的なモデルを提案し，用いる損失関数の違いや訓練例の与え方が得られる評価関数にどのように影響するかを議論する．さらに，実際に 2 万を越える特徴で学習させ，強いプログラムの評価関数の作成に成功したことを報告する．

Learning of Evaluation Functions by Game Records in Shogi

TOMOYUKI KANEKO † and KAZUNORI YAMAGUCHI †

This paper presents a general framework for learning of evaluation functions, based on comparison of sibling nodes appeared in a large database of game records. In the framework, the weights of evaluation functions are adjusted so that the value of the move selected by human players become higher than that of the other legal moves, where the value of a move is the evaluation value of the position at the leaf of the principal variation after the move computed by search. In our experiments in Shogi, we compared the effectiveness of four kinds of loss functions as well as other parameters in the framework. Then, we report that the weights of more than 20 thousands of features were successfully adjusted in our method.

1. はじめに

ゲームプログラミングにおいて評価関数を自動的に求めることは重要な研究課題であり広く研究されてきた．強いゲームプログラムを作るためには良い評価関数が必要であり，そのためには評価項目の設計と各項目の重みづけが必要である．評価項目とは局面の中で何に注目するかであり，特徴と呼ばれる．将棋の場合，駒の損得や玉の危険度などが知られている[12]．評価関数は複数の評価項目を考慮して総合的に有利・不利を判定する．一般的に用いられる線形結合の場合は，各評価項目の重みの総和が評価値となる．各評価項目の重みは事前に決めておく必要がある．複雑な評価関数において，プログラマが各評価項目に矛盾なく重みを設定することは難しい．一方で，将棋の様々な評価項目について自動的な調整を行うことは長らく困難と言われていた．

2006 年に発表された保木[19] の手法では，棋譜に指された手を実際に探索が指す方向に重みを調整することで 1 万を越える重みの調整に成功した．それ以来，評価関数の実用的な調整方法についての注目が集まっている．保木の手法はプログラムが実際にコンピュータ将棋選手権で優勝していることから有用性は疑う余地がない．しかし，計算時間が長くかかる点は今後の研究が期待されている．☆また，他の機械学習の手法との比較についても，興味深いテーマである．本研究では，ロジスティック回帰も含めて一般的な枠組で実験を行う．約 2 万の特徴を用いて実際に強いプログラムの評価関数の作成に成功したことを報告するとともに，損失関数の種類や訓練例の作成方法や学習における様々な側面を議論する．

2. 関連研究

オセロでは多数のパターンの重みを最小二乗法で調整した評価関数が早くからトップレベルの強さを実現している[3]．これは，局面に対し評価関数がとるべき評価値の教師を与えて調整するものである．一方，将棋やチェスのトップレベルのプログラムでは手で調整した評価関数が使われていた．その理由の一つは学習のための評価値の教師を用意することが難しいことと考

† 東京大学大学院総合文化研究科
Graduate School of Arts and Sciences, The University of Tokyo
kaneko@graco.c.u-tokyo.ac.jp

☆ 2006 年のゲームプログラミングワークショップの質疑では，3ヶ月程度かけたとのこと．

表 1　アルゴリズムの比較

	比較	PV
TD	親子節点	不使用
TDLEAF[2]	親子節点	使用
駒の関係[16]	兄弟節点	不使用
保木[19]	兄弟節点	使用
金子[15]	兄弟節点	使用

えられる．学習を円滑に進めるためには同じ勝ちの局面でも，プログラムが勝ちやすい局面と難しい局面を区別することが望ましい．オセロでは終盤の評価関数に対しては，探索で読み切った値を，中盤の局面に対しては終盤の評価関数を用いて探索した評価値が，学習のための適切なラベルとして機能した[3]．しかし，将棋やチェスでは評価が定まった方法は知られていない．

別の考え方として，局面の比較に基づく学習手法が研究されてきた．**表 1** に，主だった手法を掲載する．TD 法は，手を指す前と指した後の局面 (本稿では親子と呼ぶ) を比較して重みを調整する手法である．改良版の TDLEAF[2] では，親子局面を直接比較する代わりに，それぞれで探索を行い最善応手手順後の局面を求め，それらを比較する．

将棋では，棋譜で指された手の後の局面と他の合法手の後の局面 (兄弟) を比較し，棋譜で指された手の方が評価値が高くなるように重みを調整する手法が提案されている[15),16),19)]．文献[16)] では，親子節点の比較より兄弟節点の比較の方が優れると報告されている．文献[19)] は，多数の特徴を用意し実際に強いプログラムが作成されたという点で，また文献[15)] は，特徴の種類が 4,000 程度に留まるもののロジスティック回帰で有望な結果を示した点で特色がある．本研究でも実際的な強さを実現できるような多数の特徴を用意し，またロジスティック回帰を含めて手法の比較を行う．

3. 兄弟節点の比較に基づく重みの調整

初めに，棋譜で指された手が探索により選択されるように評価関数を調整するための一般的な枠組について説明する．これは，直感的には，棋譜で指された手の評価値が他の合法手の評価値のどれよりも高くなるように調整するものである．一般にゲーム木探索では min-max 法に基づいて選択された局面 (そこに至る手順を最善応手手順と呼ぶ) の評価値に基づいてルートの局面の評価値が定まる．そこで，指された手の評価値を他の手の評価値より高くするためには，それぞれの手の最善応手手順後の局面を取り出し，前者の局面の評価値が (手番にとって) 高くなるように調整すれば良い．実際には，最善応手手順は評価関数に依存し，評価関数を調整すると最善応手手順が変化しうる．そのため，この調整がうまく動作するためには，評価関数の調整の範囲では最善応手手順が変化しないか，変化の影響が限定的であることを仮定する必要がある．これは一見大胆な仮定であるが，実用としてはうまく動作することが保木[19)] により主張され，その後多くのプログラムによって肯定的な結果が得られている．

(1) 棋譜データベース中の各局面について，以下の計算を行う
  (a) 合法手を全て作成する．
  (b) 棋譜で人間により指された手の後の局面 s_i と，他の手の後の局面 t_i のペア $\langle s_i, t_i \rangle$ を作る
  (c) 各局面に対して探索を行い，最善応手手順を求め，ディスクに格納する．
(2) 格納した最善応手手順を利用して，棋譜で人間により指された手の評価値が他より高くなるように，以下の手順で重みを更新する
  (a) 各ペア $\langle s_i, t_i \rangle$ について，それぞれの局面の最善応手手順後の局面 $\langle \overline{s_i}, \overline{t_i} \rangle$ を作成し，各局面での特徴ベクトル $\langle x_{s,i}, x_{t,i} \rangle$ を求める．
  (b) ペアの差分を x_i とする ($x_i = x_{\overline{s},i} - x_{\overline{t},i}$)．また，評価値の差が正であるべきか負であるべきかを表す変数を y_i とする (先手番なら 1, 後手番なら-1)
  (c) x, y を用いて，$yw^t x > 0$ となるように w を調整する．
(3) (2) に戻り，新たな重みで探索し，その結果を元に w を調整し直すことを繰り返す．

図 1　評価関数調整の枠組

具体的な手順を，**図 1** に掲載する．大きく分けて，手順 (1) で最善応手手順を求める部分と手順 (2) で重みを調整する部分の二つからなり，収束に近付くまで全体を繰り返す．

3.1 最善応手手順の探索

手順 (1) では，棋譜の各局面で全ての合法手に対して最善応手手順を求める．ここでは，使う棋譜や，最善応手手順を求めるための探索手法などに選択肢がある．先駆者である保木の手法[19)] ではプロ棋士の棋譜を用い，また探索では全幅で一手読みと静止探索を行っている．また探索窓を，指した手の評価値の前後の歩何枚分かに絞っている．これは採用された損失関数の仕様から自然な判断である．本稿では，保木に倣って全幅で一手を読みその後静止探索を行った．但し，静止探索を長手数行うことは計算時間がかかるため，4

手で打ち切った．多くの将棋プログラムでも静止探索の深さに制限を設けているため，妥当であると考えている．棋譜の種類と探索窓の幅については様々な組合せを試した．

3.2 重みの調整

手順 (2) では手順 (1) の結果を利用して重みを調整する．手順 (1) の段階で，局面のペア $\langle \overline{s_i}, \overline{t_i} \rangle$ と，どちらの局面を高く評価するか y_i が求まっている．そこで，それをなるべく満たすように評価関数の重み (ベクトル)w を調整する．簡単のため評価関数は線形であるとして，局面 p の特徴ベクトルを x_p とすると，局面 p の評価値は w と x_p の内積 $w^t x_p$ である．従って，局面 p は局面 q より評価値が高いという条件は，$w^t x_p > w^t x_q$ と表せるが，それは $x_i = x_p - x_q$ を用いて $w^t x_i > 0$ と変形できる．ルート局面が後手の場合は棋譜で選択された手の評価値を低くすることが目的となるので，一般には手番で決まる変数 y_i を用いて $y_i w^t x_i > 0$ という形式になる．重みの調整は，このような不等式が多数与えらた時にそれらをなるべく満たすような w を求める問題となる．具体的な調整方法にはいくつか選択肢がある．本稿では複数の調整手法を比較するが，初めに今まで使われてきた手法を説明した後に，本稿で扱う調整方法を説明する．

なお，棋譜には，非常に高度な指手も，ポカのような指手も存在する．それらの指手は計算機の探索結果とは一致しにくいため，それらの影響を最小限に留めることが望ましいことが指摘されている[19]．また，既に最善手より低い点数がついている手の評価をより下げることや，その逆も無駄である☆．この点で，単純な最小二乗法は有効でないと考えられる．

3.2.1 ロジスティック回帰

ロジスティック回帰はこのような目的に使われる手法の一つであり，シグモイド関数

$$P(x,y) = \frac{1}{1+e^{-yw^t x}}$$

の出力を尤度と考えて，訓練例の全体の尤度

$$\prod_i P(x_i,y_i) = \prod_i \frac{1}{1+e^{-y_i w^t x_i}} \qquad (1)$$

を最大にする w を求める．$P(x,y)$ は $(0,1)$ の範囲を取るが，$y=1$ の時は $w^t x$ が大きいほど 1 に近付き，$y=-1$ の時は $w^t x$ が小さいほど 1 に近付く．従って $\Pi_i P(x_i,y_i)$ を最大化することは，前述の不等式を満たすことと関連が深い．過学習を回避するためには，重み全体の大きさに制限を加える l_1, l_2 正則化が用いられる．ロジスティック回帰には様々な計算手法が存在する[1),5)~8),10)] が，このような学習に用いる場合には，訓練例の数が多いため係数行列がメモリにもハードディスクにも収まらないことに注意が必要である．

☆ 同様の検討は古くからなされていたようである[9)]

表 2 損失関数

(1)	$\sum_i \log(1+e^{-y_i w x_i})$	(2)	$\sum_i \max(1-y_i w x_i, 0)$
(3)	$\sum_i e^{-y_i w x_i}$	(4)	$\sum_i \frac{1}{1+e^{y_i w x_i}}$

3.2.2 保木の最適化モデル

一方，保木[19)] は異なるモデル化を行っており，適当なスケール a のシグモイド関数を $T(x)$ として，

$$\sum_i T(-y_i w^t x_i) = \sum_i \frac{1}{1+e^{a y_i w^t x_i}} \qquad (2)$$

を最小化するように w を定める．また，保木は各反復における重みの更新において，各重みを動かす幅の制御に勾配を直接利用せずに，勾配の符号に基づいた定数幅の変更を行っている．これは，目的関数が滑らかでない (w を変更すると最善応手手順が変化しうる) ことを重視した設計と述べている．さらに過学習を避けるための正則化についても，異なる表現を用いている．

3.2.3 本稿での枠組

重みの調整を最適化問題として扱う場合には，不等式の「満たされなさ」を表す損失関数を定義し，その関数を最小化する重みを求める問題となる．本稿では，ロジスティック回帰や保木のモデルを含めた複数の手法を比較するために，**表 2** に掲載する 4 種類の損失関数を用いた．表の関数 (1) はロジスティック回帰で使われるものである．ロジスティック回帰は，式 (1) の最大化を行うが，対数をとって整理すると，表の関数 (1) を最小化する問題となる．また，関数 (1) から (3) は，正の部分より負の部分の方が傾きが大きいため，評価値について正しい順序となった局面のペアについて，評価値の差をより広げる改善よりは，誤った順序となったペアを正しくする方向を重視する性質がある．関数 (4) は保木[19)] が用いたシグモイド関数である．この関数では，上記の性質に加えて，他の指手を棋譜の指手より極端に高く評価してしまっているペアについては軽視し，差がある程度の範囲にあるペアについての改善を重視するという特徴がある．

また，本稿の実験では多数の特徴を用いることから正則化が必要であると考えられる．そこで，l_1 正則化を実現しながら目的関数を最適化する枠組として文献[4)] の手法を用いた．最適化したい目的関数を $L: R^n \to R$ とし $L(w)$ を最小化する w を

$$\|w\|_1 \le z \qquad (3)$$

という条件で求めたいとする．文献[4)] の手法では，t を

反復回数として，以下の式に基づいて w_i を更新する:

$$w^{t+1} = \Pi_X(w^t - \eta_t \nabla^t).$$

ここで，∇^t は w^t の付近での L の勾配を表し，η_t は学習の速度を制御するスカラーである．また，Π_X は $\|w\|_1 \leq z$ を満たすように原点方向に近付ける写像である．$\|w\|_1 \leq z$ が満たされている場合は，Π_X は恒等変換であり，上式は通常の勾配降下に基づく w の最適化となる．詳細は文献[4] を参照されたい．

学習の効率を高めるためには η_t が重要な役割を果たす．文献[4] の実験では，$\eta_t = \frac{\eta_0}{\sqrt{t}}$ という，反復が増えると小さくなるような単純な値を用いてもにしても十分性能が発揮されるという結果が掲載されている．しかし，予備的な実験では，本稿の題材である将棋の評価関数においては効率が良くなかった．そこで，stochastic meta descent[11] という手法を組み合わせ，η_t をベクトルに拡張して重みの更新を行った:

$$w^{t+1} = \Pi_X(w^t - diag(\eta_t)\nabla^t).$$

ベクトル η_t は学習の速度を，w の各要素別に制御するもので，以下のように更新される:

$$\eta_t = diag(\eta_{t-1}) Max_{0.5}(1 - \mu diag(\nabla_t) v_t), \quad (4)$$
$$v_{t+1} = \lambda v_t - diag(\eta_t)(\nabla_t + \lambda H_t v_t).$$

ただし μ はメタ学習定数，$Max_{0.5}$ は 0.5 より小さい要素を 0.5 で置き換えたベクトルを返す演算，また λ は $[0,1]$ の範囲の値で，慣性のような効果を持つパラメータである．H は L の w に関するヘシアンで，v は η を動かした場合の w への影響を近似するベクトルである．詳しくは文献[11] を参照されたい．実装上は H を陽に持つ必要はなく，各訓練例を処理しながら順次足し込むことで Hv を求めることができる．これは ∇ の計算と同時に行うことができるため，計算コストはほとんど増えない．なお，この手法は本来オンライン学習のための手法である．すなわち，全ての訓練例を考慮して重みを更新するのではなく，一つ，あるいは，一部の訓練例を用いて重みを更新することで，素早く解に近付くために使われる．しかし，予備的な実験では，訓練例を一部しか使わない場合は全て使う場合に比べて収束が遅かった．その理由の詳細な検討はできていないが，全ての訓練例を用いても特に問題はないため，本稿では全ての訓練例を元に重みを一括で更新する方法をとった．

さらに，初期値 η_0 の値も極めて重要である．この適切な値は，用いる特徴集合によって異なるため，全ての実験で共通の値を用いることは難しい．そこで，今回の実装では，重みの更新により損失関数 L の値を小さくすることに連続して成功している間は η_{t+1} を式 (4) で求めた値からさらに 1.2 倍に，失敗した場合は半分にするというヒューリスティックな更新を採り入れた．これにより初期値の差をかなり吸収できることが分かったため，本稿の実験では全て 10^{-6} を初期値として用いた．

表 3 駒の初期値 (1)

歩	香	桂	銀	角	飛
128	512	512	704	1024	1216
馬	龍	金・他の成駒			
1472	1664	768			

4. 実験結果

4.1 駒割の学習

はじめに，駒の価値のみの評価関数を作成し，その重みを学習させた．まず，**表 3** の初期値で探索して訓練例を作成した後に，重みを初期値 0 から調整してどのような値となるかを調査した．実験条件を単純にするため，図 1 における手順 (1) と (2) を一度づつ行った段階での結果を示す．手順 (2) においては重みの更新を 100 回行ったところで打ち切った．表 3 に近い値が復元できれば，学習に成功したと考えられる．学習は浮動小数で行ったが，結果は歩の値が 128 点となるように全体をスケールして表示する．最善応手手順を求める際の探索窓は，歩の枚数に換算して 3 枚分，8 枚分，16 枚分の 3 通りを試した．また棋譜は，将棋クラブ 24[13] の棋譜，同将棋クラブ 24 で先手後手双方のプレイヤのレートが 1500 以上のもの (以降では簡単に有段の棋譜と呼ぶ)，プロの棋譜の三種類をそれぞれ約 200 試合を用いた．

結果をまとめて**表 4** に掲載する．全体の傾向として似たような値になっているが，関数 (2) は値が小さい (歩の価値が相対的に高い) 傾向にある．また，棋譜による差はあまりない．探索窓は狭い方が飛車や龍などの価値の高い駒の値が高くなり，駒の値の差がはっきり出るようである．代表として，飛車の値をグラフに描いたものを**図 2** に掲載する．飛車の価値が 1000 点を越えたのは，関数 (1) を用いた場合のみ (将棋クラブ 24 有段者の棋譜とプロの棋譜で探索窓が歩 3 枚分の時) である．以上より，値を復元するという観点では関数 (1) が適していると考えられる．

4.2 進行度等を加味した駒割の学習

続いて評価関数に，特徴を足した場合に先ほどの傾向が変化するかどうかを調査した．具体的には，終盤の駒割と玉の安全度の差[14] を加えて同様の実験を行った．進行度 p が 0 から 1 の値をとるとして，

評価値 = 駒割 + p·(終盤用駒割 + 玉の安全度の差)

という式として学習を行った．また，文献[19] を参考に，

表 4　学習後の駒の価値 (歩の値を 128 とした時の相対的な値)

棋譜	探索窓	関数	香	桂	銀	金	角	飛	と	成香	成桂	成銀	馬	龍
24	3	(1)	401	409	589	678	816	977	650	618	633	661	1195	1347
		(2)	260	260	388	408	520	648	392	242	314	364	776	904
		(3)	381	392	569	659	775	904	654	660	630	627	1141	1255
		(4)	380	398	556	636	785	948	598	417	558	580	1145	1301
24	8	(1)	342	350	514	606	684	825	575	574	568	608	1027	1153
		(2)	240	243	348	409	449	546	384	213	302	299	677	768
		(3)	339	347	507	596	680	799	578	610	576	589	1017	1110
		(4)	324	346	491	583	666	812	543	433	499	562	999	1140
24	16	(1)	323	330	489	579	642	773	549	483	527	566	964	1082
		(2)	215	220	314	370	385	467	356	162	236	249	565	630
		(3)	322	317	479	559	625	733	543	562	553	555	940	1022
		(4)	301	322	460	546	616	746	509	356	447	489	918	1043
24 有段	3	(1)	409	422	604	696	838	1010	658	598	652	664	1220	1385
		(2)	257	257	385	418	513	641	385	223	312	319	769	897
		(3)	386	399	573	661	775	928	638	599	639	632	1122	1280
		(4)	377	397	559	642	787	946	586	319	518	461	1126	1284
24 有段	8	(1)	340	358	521	614	697	845	577	538	579	590	1039	1186
		(2)	246	252	360	426	462	562	386	246	312	298	703	790
		(3)	341	356	515	603	687	821	567	573	589	594	1014	1149
		(4)	317	346	491	582	666	804	541	427	495	489	994	1126
24 有段	16	(1)	324	342	500	592	661	801	561	510	558	562	990	1126
		(2)	218	226	324	384	400	490	384	171	243	230	596	658
		(3)	313	329	479	563	625	742	541	547	567	557	925	1041
		(4)	310	338	481	570	647	779	531	417	482	478	965	1089
プロ	3	(1)	413	428	605	698	879	1030	677	563	657	699	1247	1404
		(2)	257	257	385	387	514	642	385	157	276	336	770	897
		(3)	400	414	588	674	843	962	663	587	638	679	1192	1267
		(4)	376	388	546	631	799	951	606	300	508	550	1121	1285
プロ	8	(1)	344	362	516	601	743	853	590	450	564	597	1051	1168
		(2)	238	248	338	386	468	546	387	145	266	273	642	712
		(3)	359	373	531	610	764	855	602	554	603	614	1071	1137
		(4)	296	318	449	538	653	767	531	280	419	459	921	1053
プロ	16	(1)	329	346	495	577	708	809	575	421	536	573	999	1109
		(2)	203	212	287	340	392	446	337	103	179	206	494	536
		(3)	331	344	499	568	701	777	569	503	565	580	978	1031
		(4)	299	321	453	543	658	771	537	322	438	495	930	1061

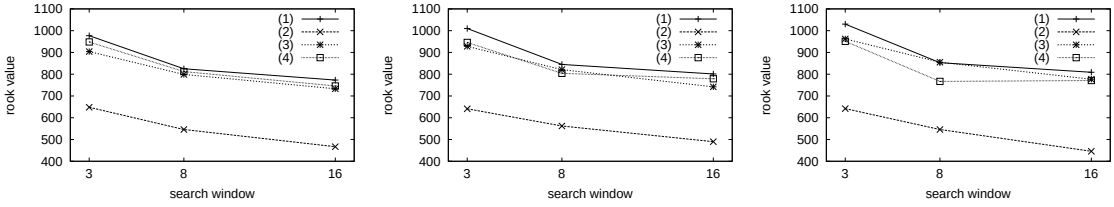

図 2　パラメータを変えた場合の飛車の値の変化 (左:将棋クラブ 24，中央:将棋クラブ 24 有段，右:プロ)

学習後の棋譜との不一致度を以下のように測定した:

$$\text{不一致度} = \frac{1}{\text{局面数}} \sum_{\text{合法手 i}} T\left(\xi(\mathrm{i}) - \xi(\text{棋譜の手})\right)$$

ξ は最善応手手順後の局面の評価値，$T(x)$ はシグモイド関数 $T(x) = 1/(1+\exp(-3x/128))$ で 128 は歩の点数である．この指標は，評価値の差が十分ある場合は 0 または 1 となり，各局面で最善手より評価値が高い指手の数を数えることに相当する．一方，評価値が近い指手については 0.5 近辺となるため，最善手より評価が低くても値が近いものがあれば評価は悪くなる．この数値は高いほど悪い評価である．なお，学習において探索窓を絞った場合も，棋譜との不一致度を測る際には共通の広い窓を用いた．

表 5 に結果をまとめる．紙面の都合で将棋クラブ 24 の棋譜を使ったもののみ掲載した．全体にわたり，成っていない駒は，ほぼ駒の強さに応じた点数の順番

表 5　学習結果 (重みは歩の値を 128 とした時の相対的な値)

探索窓	関数	特徴	歩	香	桂	銀	金	角	飛	と	成香	成桂	成銀	馬	龍
3	(1)	全体	128	337	336	475	517	649	772	479	325	361	380	864	994
		終盤	-29	38	21	66	113	96	137	128	244	223	242	296	310
				安全度差		56						(棋譜との不一致度)			12.36
3	(2)	全体	128	251	249	356	370	472	585	358	182	251	264	685	792
		終盤	-40	41	31	60	113	111	130	94	136	161	170	209	226
				安全度差		48						(棋譜との不一致度)			12.52
3	(3)	全体	128	320	332	509	544	704	827	494	299	349	362	925	1064
		終盤	-29	31	6	-13	46	-31	-32	101	223	206	205	141	102
				安全度差		50						(棋譜との不一致度)			12.37
3	(4)	全体	128	382	358	453	526	649	789	552	262	394	431	977	1102
		終盤	-27	41	59	180	192	251	334	89	205	231	249	368	478
				安全度差		85						(棋譜との不一致度)			12.21
8	(1)	全体	128	293	293	413	460	555	642	400	284	311	336	727	819
		終盤	-33	9	-8	32	74	29	91	123	198	185	207	238	263
				安全度差		64						(棋譜との不一致度)			12.20
8	(2)	全体	128	207	209	276	306	349	422	290	143	196	206	505	567
		終盤	-44	30	5	68	108	102	145	109	102	122	136	220	263
				安全度差		69						(棋譜との不一致度)			12.25
8	(3)	全体	128	296	304	461	512	631	733	419	293	318	340	805	913
		終盤	-32	-1	-27	-46	-13	-81	-69	93	208	185	182	102	74
				安全度差		49						(棋譜との不一致度)			12.34
8	(4)	全体	128	323	305	382	471	539	683	475	228	337	384	868	970
		終盤	-39	34	42	175	188	215	308	110	160	180	215	323	420
				安全度差		101						(棋譜との不一致度)			11.97
16	(1)	全体	128	276	276	387	425	511	584	377	249	280	308	660	745
		終盤	-36	0	-21	23	71	13	82	112	172	175	191	227	248
				安全度差		67						(棋譜との不一致度)			12.14
16	(2)	全体	128	196	194	258	278	316	382	273	114	163	178	464	529
		終盤	-45	28	4	64	117	91	142	107	79	105	117	205	240
				安全度差		79						(棋譜との不一致度)			12.15
16	(3)	全体	128	284	295	438	480	587	675	396	264	290	308	722	820
		終盤	-37	-19	-62	-66	-35	-110	-95	68	184	180	166	94	60
				安全度差		49						(棋譜との不一致度)			12.29
16	(4)	全体	128	322	316	388	469	533	674	490	264	328	401	873	967
		終盤	-32	40	29	169	188	218	308	91	180	189	216	283	385
				安全度差		102						(棋譜との不一致度)			11.96

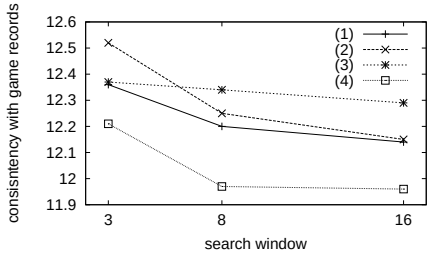

図 3　棋譜との不一致度

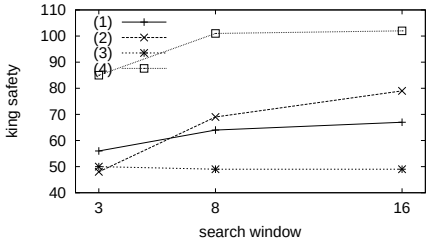

図 4　玉の安全度の値

になっている．また，終盤になると，歩の価値は下がり金の価値は上がる．しかし，個々の値はかなり異なったものとなっている．棋譜との不一致度に注目すると，**図 3** に示したように，全体として探索窓を広げる方が微差ではあるが良い成績(値が低い) となっている．また，損失関数の違いに着目すると関数 (4) がもっとも成績が良い．これは不一致度と同じ形式の関数を最適化の対象としていることを考えれば自然である．(4) に続いては (1) の成績が良い．玉の安全度の差についても，**図 4** に示したように探索窓や損失関数によって傾向が異なる．文献[14] では角 2 枚分程度に調整されたプログラムが強かったことから大きい方が良い調整であると考えると，探索窓は広く取る方が良く，また損失関数は関数 (4) や (1) が向いていると考えられる．

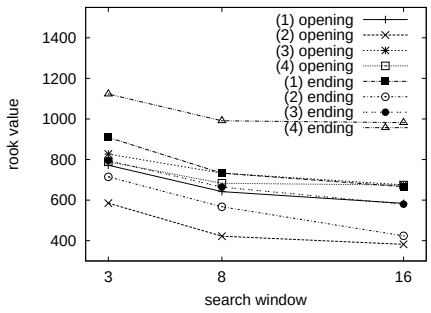

図 5 飛車の値

最後に代表として飛車の値についてグラフにしたものを，**図 5** に示す．opening は序盤 (進行度 $p = 0$) の値で，ending は終盤 (進行度 $p = 1$) の値である．前節同様に探索窓が広がるほど値は下がる傾向にある．また，前節とは異なり関数 (1) よりも関数 (4) の方が高い値を保っている．図 2 と比較すると序盤の値ははっきり低いが終盤の値は似たようなものになっている．

これだけの実験で明確な結論を出すことは難しいが，まず，駒割のような基本的な特徴の値ですら，実験条件により大きく左右されることが分かる．詳細には，探索窓を狭くするとデメリットがある可能性があり，損失関数は関数 (1) や (4) が向いていると示唆される．

4.3 実用的な評価関数の調整とその評価

続いて，実用的な正確さの評価関数を得るために特徴を増やした実験について報告する．実際的な性能を評価するために，Bonanza や棚瀬将棋，奈良将棋で用いられている特徴を参考に駒の関係や玉との相対位置，絶対位置，玉の回りの利きの状態などを用いて約 7 万種類の特徴を準備した．一部の特徴は，序盤と終盤に別々に用いて重みを割り当てた．特徴を随時増やしながら学習を行ったところ OSL revision 3320 の時点で約 2 万 2 千の特徴に重みがついた．この時は l_1 正則化を行っていない．従って，重みが 0 となる場合は，値が非常に小さく整数に変換した際に 0 となった場合か，そもそも棋譜に表れなかった場合である．いくつかのバージョンのうち最も有望と思われたものを GPS 将棋 (http://gps.tanaka.ecc.u-tokyo.ac.jp/gpsshogi/) に組み込み，15 分切れ負けで GPS 将棋の本来の評価関数のプログラムと対戦させたところ 76 勝 32 敗と統計的に有意に勝ち越した．このことから，学習により十分強い (世界コンピュータ選手権で決勝レベルの) プログラムを作成できたと言える．

続いて，l_1 正則化の効果を測るために，同じ特徴集合を用いて正則化パラメータを途中で変更しながら重みを 0 から調整しなおした．なお損失関数は (1) を用い (ロジスティック回帰相当)，探索窓は歩 8 枚分とし，プロ棋士の棋譜 8,000 局を用いて学習を行った．図 1 における手順 (2) では重みを 50 回更新したところで打ち切って，次の反復に移った．

表 6 反復の進行と成績の変化

z/N	不一致度	1	2	有効特徴	駒割率	勝/敗
0.004	5.25	163	180	22818	0.04	
	4.47	158	183	23120	0.20	
	4.27	163	179	23286	0.22	
0.0006	4.767	162	178	6102	0.18	
	4.584	157	181	5027	0.22	
	4.528	160	176	4179	0.25	
	4.465	160	178	3696	0.30	
	4.452	159	178	3687	0.30	
0.002	4.522	163	176	22970	0.14	
	4.308	163	185	16481	0.21	
	4.244	160	181	15627	0.22	
	4.186	161	184	13540	0.26	
0.003	4.159	155	183	23036	0.16	
	4.131	159	184	23094	0.16	
	4.116	161	183	23136	0.17	
	4.075	168	186	23155	0.21	23-17
	4.059	162	184	23211	0.21	
	4.044	166	188	17719	0.29	
	4.043	166	184	17744	0.29	
	4.030	168	182	16162	0.33	
	4.022	165	185	15604	0.34	23-17
	4.023	165	181	15373	0.35	
0.004	4.022	167	182	23017	0.23	
	4.016	168	182	23048	0.24	24-16
	4.017	167	186	23074	0.24	
	3.993	166	183	20713	0.34	
	3.989	166	183	19718	0.35	
	3.986	164	187	18815	0.37	
	3.978	170	181	17463	0.41	22-18
0.005	3.978	165	187	23017	0.31	
	3.976	166	189	23055	0.31	
	3.974	165	187	23081	0.31	
	3.963	166	183	21148	0.42	
	3.960	164	184	20750	0.42	
	3.956	166	184	18797	0.48	
	3.952	168	189	18978	0.47	
	3.950	160	185	18063	0.50	
	3.946	165	181	18260	0.50	20-20

強さの測定は，棋譜との不一致度，ならびに問題集の正答数と対局により行った．棋譜との不一致度は訓練に用いたものとは別の 200 棋譜中の 4 手毎の局面を利用した．問題集は，文献[17),18)] のものを用い，それぞれ 216 題である．1 題最大 30 秒で探索した際の正答数を測定した．対局は，l_1 正則化を行わずに学習したものとの 40 試合で評価した．

結果を**表 6** に掲載する．表の項目の z/N は，式 (3) の l_1 正則化のパラメータである重みの絶対値の合計 z を特徴の数 N で割った値を示したものである．続いて左から順に，棋譜との不一致度，問題集 1, 2 の正答

率，0 でない重みのついた特徴の数，0 でない重みについてその絶対値の平均を歩の重みで割ったもの，対局の勝敗である．また，表の各行は一回の反復に対応し，上から下に向かうほど反復が進む．z/N は初めに 0.004 に設定して始めたが，$\|w\|_1$ が急速に増大したため 4 回目の反復に移る前に小さく設定し直し，その後も重みの変化が落ち着いたら z/N を増やす変更を適宜手で行った．

その結果，不一致度は，反復が進むほど順調に下がり，またその値は前節の実験での値よりも大幅に低い．これは用いた特徴が実際に有効なものであったためと考えられる．一方，問題集の正答数は反復によって大きくばらつき傾向を見出すことは難しい．有効特徴の数は，正則化パラメータの大きさとほぼ対応し，同じ正則化パラメータを使う限りにおいては反復が進むほど数が減る．これは重要な特徴に大きな重みをつけるためと考えられる．そのことは駒割率が，反復が進むほど大きくなることからも読み取れる．対戦においては負け越すことはないものの，統計的に有意に勝ち越すことはなかった．このことは，l_1 正則化を行わずに学習した評価関数は過学習の弊害があると予想していたため，やや意外な結果である．以上より最適な z/N の値についてはさらに詳細な研究が必要である．

5. おわりに

本稿では，兄弟節点の比較に基づく評価関数の調整方法について汎用的な枠組を提案し，過去に提案された手法の共通点と相違点をそれを用いて議論した．将棋の駒割や小規模な特徴集合において，4 種類の損失関数を用いて調整を行ったところ，複数の損失関数で現実的な値が得られることを確認した．また 2 万を越える大規模な特徴集合について，ロジスティック回帰モデルで重みを調整を行った．調整後の評価関数を用いたプログラムは，手で調整した評価関数で十分な強さを持つ GPS 将棋に有意に勝ち越した．このことから，勾配を利用する標準的な方法が，この枠組における高次元の学習においても有効に働いたと言える．現時点では，先駆者である保木[19] の手法とどちらが優れるかについて厳密な比較はできないが，本稿には，異なる損失関数や重みの更新方法を用いても強いプログラムができることを明らかにしたという意義がある．

参考文献

1) G. Andrew and J. Gao. Scalable training of L1-regularized log-linear models. In Z. Ghahramani ed., *ICML*, pp. 33–40. Omnipress, 2007.
2) J. Baxter, A. Tridgell, and L. Weaver. Learning to play chess using temporal-differences. *Machine Learning*, 40(3):242–263, Sept. 2000.
3) M. Buro. Improving heuristic mini-max search by supervised learning. *Artificial Intelligence*, 134(1–2):85–99, Jan. 2002.
4) J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In A. McCallum and S. Roweis eds., *ICML*, pp. 272–279. Omnipress, 2008.
5) A. Globerson, T. Koo, X. Carreras, and M. Collins. Exponentiated gradient algorithms for log-linear structured prediction. In Z. Ghahramani ed., *ICML*, pp. 305–312. Omnipress, 2007.
6) K. Koh, S.-J. Kim, and S. Boyd. A method for large-scale l_1-regularized logistic regression. In *22nd National Conference on Artificial Intelligence*, pp. 565–571, 2007.
7) S.-I. Lee, H. Lee, P. Abbeel, and A. Y. Ng. Efficient l1 regularized logistic regression. In *AAAI*, pp. 401–408. AAAI Press, 2006.
8) C.-J. Lin, R.C. Weng, and S.S. Keerthi. Trust region newton methods for large-scale logistic regression. In *Proceedings of the 24th international conference on Machine learning*, Vol. 227, pp. 561–568, 2007.
9) T. Marsland. Evaluation function factors. *ICCA Journal*, 8(2):47–57, 1985.
10) A. M. Paul Komarek. Fast robust logistic regression for large sparse datasets with binary outputs. In *Artificial Intelligence and Statistics*, 2003.
11) N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.
12) 鶴岡. 将棋. 情報処理, 44(9):900–904, 2003. 特集ゲーム情報学.
13) 久米. 将棋倶楽部 24 万局集. ナイタイ出版, 2002.
14) 竹内, 金子, 林, 山口, 川合. 勝率に基づく評価関数の評価と最適化. 情報処理学会論文誌, 48(11):3446–3454, 2007.
15) 金子. 兄弟節点の比較に基づく評価関数の調整. 第 12 回ゲームプログラミングワークショップ, pp. 9–16, Nov. 2007.
16) 金子, 田中, 山口, 川合. 駒の関係を利用した将棋の評価関数. 第 8 回ゲームプログラミング ワークショップ, pp. 14–21, Nov. 2003.
17) 日本将棋連盟書籍（編）. ラクラク次の一手 基本手筋集. 日本将棋連盟, 2002.
18) 日本将棋連盟書籍（編）. ラクラク次の一手 2 基本手筋集. 日本将棋連盟, 2003.
19) 保木. 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, Nov. 2006.