# 囲碁に対する2つの情報工学的アプローチ

荒木伸夫

電気通信大学
情報理工学研究科　情報・通信工学専攻
工学博士の学位申請論文
2017年3月

# 囲碁に対する２つの情報工学的アプローチ

博士論文審査委員会

村松正和
山本野人
伊藤毅志
岡本吉央
小林聡
武永康彦

# Abstract

Our themes of research are “Approaching the game of Go by Monte-Carlo method” (Part I) and “Approaching the game of Go by CNN(Convolutional Neural Network)” (Part II).

The effectiveness of the Monte-Carlo method on Go has become popular since around 2006, and the strength of Go AI(Artificial Intelligence) had rapidly improved. But since around 2012 till 2015, it had hardly improved.

Our method “Simulation Adjusting” was made for breaking this hiatus. This method enables Monte-Carlo AI to select the same moves as human experts' moves by adjusting the simulation part.

This method is based on “Simulation Balancing.” However, our method does not need “trainer Monte-Carlo AI” like “Simulation Balancing” but only game records of human experts or training problems.

After various improvements, the value of the objective function steadily decreased and correct answer ratio steadily increased in our experiments on 5x5 board problems. We describe about this in Part I of this thesis.

On the other hand, DCNN(Deep Convolutional Neural Network) made a break-through at the competition of image recognition. After that, since around 2014, the effectiveness of DCNN on Go has been revealed. In 2016, DeepMind team of Google revealed AlphaGo, which uses DCNN move prediction, the world's first static evaluation function of Go based on DCNN, and Monte-Carlo method. AlphaGo has beaten the world's top level human for the first time as Go AI.

Around the revealing of AlphaGo, we have tried “Estimating Player's Strength by CNN from One Game Record of Go.” We used Caffe as a CNN library and estimated the players' strength from only one game record of GoQuest. We succeeded to make the average mean squared error of ratings smaller. We describe about this in Part II of this thesis.

# 概要

本研究は「囲碁における情報工学的アプローチ」をテーマとしている．現在も囲碁で用いられているモンテカルロ法に関連した機械学習法，Simulation Adjusting の提案 (第 I 部) と，畳み込みニューラルネットワークを用いた棋力推定の提案 (第 II 部) が二大テーマである．

囲碁 AI（Artificial Intelligence，人工知能）の研究は 1960 年代に始まったが，その棋力は，2005 年頃まで平均的なアマチュアプレイヤーにすら及ばなかった．そのような状況で，囲碁におけるモンテカルロ法の有効性が 2006 年頃から知られ始め，囲碁 AI の棋力は急速に伸び，2012 年頃にはアマチュア上級レベルに達した．しかし，2012 年頃から 2015 年頃まで，囲碁 AI の棋力は再び停滞した．

その状況を打破しようと，モンテカルロ法のシミュレーション部の新たな学習手法として提案した手法が，第 I 部で述べる Simulation Adjusting である．これは，モンテカルロ法を用いた AI が返す手が，人間の上級者の手と一致するように，シミュレーションの方策を調整していく手法である．既存研究に Silver らによる Simulation Balancing があるが，Simulation Balancing が「教師となるモンテカルロ AI」から学習するのに対し，Simulation Adjusting は「上級者の棋譜や問題集」から学習する点が異なる．

Simulation Adjusting の実験では，最終的には 5 路盤の問題集での安定した目的関数の減少と正答率の向上を確認できた．

一方，2012 年に DCNN（Deep Convolutional Neural Network，階層の深い畳み込みニューラルネットワーク）を用いたシステムが画像認識のコンペティションでブレイクスルーを起こした．その後，囲碁の盤面認識にも DCNN が有効なのではないかと言われ始め， 2014 年頃から徐々に，DCNN が囲碁の着手予測にも有効であるという研究が発表され始めた．2016 年に Google 社の DeepMind チームが発表した AlphaGo は人間のトッププレイヤーに初めて勝利を収めたが，これには，DCNN が決定的な役割を果たしている．

第 II 部では，CNN（Convolutional Neural Network，畳み込みニューラルネットワーク）の局面認識能力を活用し，一局の棋譜のみからの棋力推定に関する研究を述べる．

プレイヤーの棋力を少数の棋譜からコンピュータが自動的に推定できるようになれば，インターネット碁会所の運営などにたいへん役に立つと考えられる．また，実際に人間の上級者は一局の棋譜のみからプレイヤーの棋力をかなりの精度で推測できると言われており，この研究は人工知能の観点からも興味深い．

コンピュータによる棋力推定に関する既存研究は囲碁を含めていくつか存在したが， CNN を使用したものは当時は見当たらなかった．我々の研究では DCNN のライブラリである Caffe を用いた棋力推定のシステムを構築し，インターネット対局場「囲碁クエスト」の 13 路盤の棋譜を用いて，どれくらいの精度で棋力を推定可能かを調べる実験を行った．その結果，適合誤差を既存研究より小さくすることができた．この結果より，畳み込みニューラルネットワークは，AI の棋力向上だけでなく棋譜からの棋力推定にも有用であると分かった．

# 目 次

# 序論

囲碁は数千年の歴史を持つと言われているボードゲームである．中国，韓国，日本，台湾に大きなプロ組織があり，近年は欧州にもプロが誕生している [1]．誕生から数千年経っても，プロ同士のゲームをアマチュアがお金を出して観戦したり，アマチュア同士でも盛んにプレイされているゲームである．当然ながら社会的関心も高い．

囲碁に対しては，科学的，工学的な意味における研究も様々に試みられて来た．それらの中で,「認知科学的研究」「数理的研究」「情報工学的研究」の 3 つが大きな部分を占めている．これらは厳密に分けられるものではなく，お互いに重なり合う部分もある（図 1）．以下では，まずそれぞれの分野における代表的な研究を紹介する．

認知科学的研究は，人間の囲碁プレイヤーが囲碁について考えている様子を外部から観測することで，人間が囲碁をどのように理解しているかを追究し，人間の認知能力を解明することを目的とする．代表的なものとして高橋らの研究 [2] がある．この研究では，囲碁の局面を人間の囲碁プレイヤーに見せ，視線の動きをカメラで追い，考えていることを発話してもらうという実験を行なうことで，初級者，中級者，上級者の思考方法に差があることが分かった．

数理的研究は，数学で囲碁を解析するものである．代表的なものとして Berlekamp らによるヨセの研究 [3] や中村による攻め合いの研究 [4] がある．これらの研究では，囲碁のヨセや攻め合いなどの部分問題を「組み合わせゲーム理論」という数学の理論を用いて解いている．人間プレイヤーにはまず解けないような難解な問題も解けるという結果が出ている．しかし，実際のゲームに適用できる場面は少ない．

そして情報工学的研究は，コンピュータに囲碁を打たせる，いわゆる囲碁 AI（Artificial Intelligence, 人工知能）の研究等，囲碁に対して計算機を用いてアプローチするものである．

囲碁 AI の棋力の推移を図 2 に載せる．このように人間超えを果たすまで非常に長い時間がかかっている．

人間超えを果たすのに中心的役割を担った研究として，Coulom によるモンテカルロ木探索の研究 [5] と，Silver et al. による DCNN（Deep Convolutional Neural Network，階層の深い畳み込みニューラルネットワーク）の研究 [6] がある．

Coulom の研究は，モンテカルロ木探索を用いることで，それまでの囲碁 AI を上回る強さの AI を作ることに成功しており，この研究が原動力となって 2006 年から 2012 年まで囲碁 AI は急激に進歩した．

Silver et al. による研究は 2012 年から 2015 年にかけての囲碁 AI の棋力の上昇の停滞を打ち破った研究である．彼らの作った「AlphaGo」は，DCNN を駆使して史上初めて人間のプロプレイヤーにハンデ無しで勝った AI である．論文 [6] を発表後に世界トップレベルのプロプレイヤーであるイ・セドルとの 5 番勝負も 4 勝 1 敗で制した．

また，AI 以外にも，Moudřik et al. による棋力推定の研究 [7]，Kishimoto et al. による詰碁ソルバーの研究 [8] などがある．棋力推定については第 II 部で詳しく述べる．詰碁ソルバーに関しては，証明数探索の有効性が示されている．

本研究は「モンテカルロ法」に対する新たな提案と,「CNN（Convolutional Neural Network，畳み込みニューラルネットワーク）」に対する新たな提案から成っている．双方とも，囲碁における情報工学的アプローチである．

第 I 部で述べる Simulation Adjusting は，モンテカルロ法による囲碁 AI に適したシミュレーション方策を見つける為の手法である．本研究は囲碁 AI の棋力の上昇が停滞していた 2013 年に開始した．モンテカルロ法においては，シミュレーション方策により囲碁の局面を評価する精度が変わってくる．囲碁というゲームは，局面を評価することが，AI にとっても人間にとっても最も難しく，AlphaGo が登場するまでは高精度の静的盤面評価関数は存在しなかった．モンテカルロ法のシミュレーション方策を

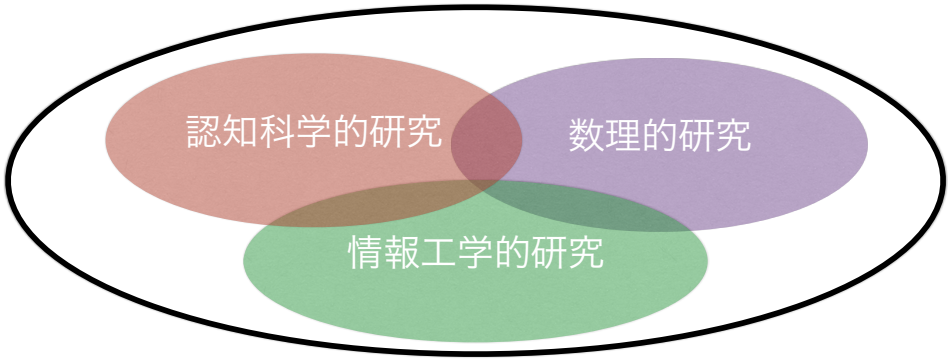



図 1: 研究対象としての囲碁.

改良していくことが，局面評価の最良の方法だった．シミュレーション方策を改良して人間に近づけることが，囲碁 AI の棋力上昇につながると考えられた．

本研究では「Simulation Balancing」という既存研究 [9, 10] を基に目的関数を設計した．Simulation Balancing では「教師 AI が局面に対して評価値として出す勝率」に,「今から強くしたい AI が同じ局面に対して評価値として出す勝率」を近づけるという最適化が行われていた．それに対し，我々の Simulation Adjusting では,「ある局面に対して正解と見なされる着手」に,「今から強くしたい AI が同じ局面で返す着手」を近づけるという最適化を行なった．我々の手法は，教師 AI を必要とせず，棋譜だけあれば適用できるという利点がある．

実験に当たっては，正解着手が各局面に対し明確に一つに定まっているような問題集を用いた．最初は 4 路盤問題集で実験を行い，その後様々な改良を行って，最終的には 5 路盤問題集での目的関数の安定した減少と正答率の安定した向上を確認できた．

第 II 部では，CNN を棋力推定に使うという新たな提案について述べる．AlphaGo もモンテカルロ法を用いているが，DCNN の研究以降，囲碁 AI の手法としてモンテカルロ法の重要性は相対的に下がり，DCNN の重要性が上がった．そのため本研究も，DCNN を主体とする方針に切り替えた．本研究では，(D)CNN を用いた囲碁の局面認識を，棋力推定に用いることを提案した．

棋力推定のニーズは実は多い．現在のインターネット碁会所では，多くの場合，新規参入者は数局から数十局打たないと適正な段位やレーティングが付かない．本来の実力より過小あるいは過大な段位やレーティングでの対局が発生することは，インターネット碁会所の利用者にとって精神的な負担となりえる．そのため，一局の棋譜から棋力を推定することを目指した．

CNN を棋力推定に用いた既存研究は見当たらなかったため，全結合ニューラルネットワークを用いて数十局の棋譜から棋力推定を行った研究 [7] と我々の手法を比較した．論文 [7] ではあらかじめ様々な特徴を取り出し，それを全結合ニューラルネットワークに入力として与え，棋力を出力させている．それに対し，我々の手法は，ほぼ生の盤面を bit 列として CNN に入力し，棋力を出力させている．今回は与える棋譜として一局分と，論文 [7] との比較のために十局分を試し，棋力としてレーティング値を用いた．

その結果，一局分の場合は CNN を用いた我々の手法の方が，本来のレーティング値と推定されたレーティング値の間の適合誤差が小さく，相関係数が大きいという結果が得られた．また，十局分の場合も

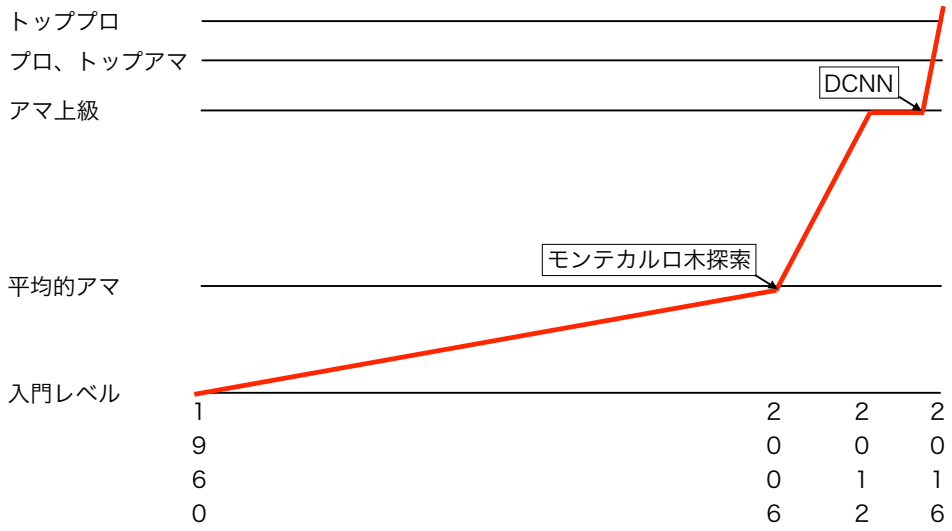



図 2: 囲碁 AI の棋力の推移.

論文 [7] の手法より良い適合誤差を示した．加えて，大まかな棋力クラス分類の実験も行い，レーティング値を出すネットワークが必ずしも万能ではないことを示した．

本研究が，今後の囲碁研究，モンテカルロ法の研究，及び CNN の研究の一助になれば幸いである．

# 第I部

# Simulation Adjusting
# ～モンテカルロ法によるアプローチ～

# 第1章　はじめに

第 I 部では，まず第 2 章で，囲碁 AI でモンテカルロ法が用いられるようになるまでの歴史を述べる．その後，第 3 章で，囲碁 AI の世界に第一の革新を起こした「モンテカルロ木探索」とその限界について述べる．その中で，本研究のベースとなった既存研究である「Simulation Balancing」についても述べる．それから，第 4 章から第 7 章で本研究の「Simulation Adjusting」について述べる．まず第 4 章で Simulation Adjusting の動機と理論について述べる．そして，第 5 章では自乗誤差の期待値を最小化する目的関数を用いた場合を述べ, 第 6 章では正解以外の勝率を小さくする目的関数を用いた場合を述べる．最後に第 7 章で結論を述べる ．

# 第2章　ゲームAI研究の歴史 ～囲碁AIがアマチュア上級レベルに達するまで～

## 2.1　チェスAIが強くなるまで

AI （Artificial Intelligence, 人工知能）の研究は，コンピュータに人間のような知性を持たせようとする試みであり，コンピュータの黎明期から始まっていた [11].

例えば，McCulloch and Pitts [12] では人工ニューロンが初めて提案され，その後のニューラルネットワーク [1] の研究の基礎となった.

また Shannon [13] ではチェスのプログラムが初めて提案されており，それ以降，チェスや囲碁のような二人完全情報確定零和ゲーム[2] を，より高いレベルでプレイできる AI を作ろうとする試みは盛んに行われてきた [14]．代表的なものだけでも，チェス，将棋，囲碁，連珠，チェッカーなどの AI が研究されてきた．その理由は大きく二つある.

一つは，二人完全情報確定零和ゲームは現実世界の問題より情報量が制限されており，当時の貧弱な計算機とアルゴリズムでも容易に扱えたからである.

もう一つは，二人完全情報確定零和ゲームの場合は，勝率という明確な指標があり，AI の進歩を客観的に測定可能だったからである．二人完全情報確定零和ゲームの AI の場合，改良後のアルゴリズムが以前のアルゴリズムに対して統計的有意に勝ち越すことが出来れば，そのアルゴリズムは進歩したと言える場合が多い．それに対し，現実世界の問題の場合，AI の評価を多次元的にする必要があることが多く，評価自体が非常に難しい.

チェスにおいて，強いプレイヤーには

- 戦略：局面を正しく評価すること，及び長期的な視野に立って計画を立てて戦うこと
- 戦術：より短期的な数手程度の作戦

の２つが必要であると言われている [15].

この二つは一般に二人完全情報確定零和ゲームをプレイする AI にとっても重要な能力であると考えられ，この二つをどのようにコンピュータプログラムで再現するかが，二人完全情報確定零和ゲーム AI 研究の主な課題であった.

チェスは欧米で「知性の象徴」として扱われていたこともあり，チェス AI の研究の開始は 1950 年頃とかなり早かった [13]．そして，チェス AI は局面を正しく評価するための静的盤面評価関数 と，短期的な先読みの力に相当する Min-Max 型の探索 を使うことにより，順調に強くなった．なお,「長期的な視野に立って計画を立てる」ことは，チェス AI が上記 2 つの構成要素によって非常に強くなった現在，自然に達成されている.

[1] ニューラルネットワークについては付録を参照.

[2] 以下，この章の下線が引かれている用語については，2.5 節を参照されたい.

## 2.2 囲碁 AI の低迷期

一方，囲碁 AI は 1960 年頃から Lefkovitz [16] によって研究され始めたが，中々強くならなかった．チェス AI が 1997 年に世界チャンピオンを破ったのに対し，囲碁 AI は 2005 年時点で最強のものでも，平均的なアマチュア未満のレベルであり，アマチュア上級者やさらに強いプロプレイヤーには全く及ばなかった [17].

強い囲碁 AI の作成が難しかった理由は大きく二つあった．

第一に，囲碁では「良い」静的盤面評価関数が作れなかったことが挙げられる．プレイヤーはチェスでは駒を，囲碁では石を使用するが，チェスは駒に決まった役割があり，どの駒を取ったか，取られたかをベースとした静的盤面評価関数がかなり正確な値を返すことができる．一方，囲碁の場合，黒白以外は，石自体に決まった役割が無く，石の組み合わせにより石の役割が生まれ，局面が特徴づけられる．この石の組み合わさり方と局面の形勢の対応関係は非常に複雑である．静的盤面評価関数を作成する試みは様々に行われたが，それらが効率的な探索に資することはなかった．

第二に，囲碁はチェスよりもはるかに合法手が多いことが挙げられる．チェスの合法手の数は，平均的には 35 であると言われている．それに対し，囲碁では，空いている交点のほぼ全てに着手できることから，公式戦で使われる 19 路盤の場合，合法手の数は平均的には 250 ほどであると言われている [18]．終局直前でも合法手が 100 を超えることも珍しくない．チェス AI で採用されていた Min-Max 型の探索は，基本的にあり得る手を全て，一定手数先まで調べる探索手法である．囲碁 AI にそれを適用しても非常に浅い探索しか出来なかった．

## 2.3 囲碁 AI におけるブレイクスルー

そのような状況の中，2006 年から「モンテカルロ木探索」という手法が囲碁で使われるようになり，第一の革新が起きた．モンテカルロ木探索の詳細については第 3 章で述べるが，これは，擬似乱数を用いて局面を評価する「原始モンテカルロ法」と「モンテカルロ法向きの特殊な探索」を組み合わせた手法である．

囲碁の AI を作るためにモンテカルロ法を使おうという試みが初めてなされたのは 1990 年代の Brügmann [19] の研究であると言われているが，提案された当時はあまり注目を浴びず，囲碁 AI の進歩に貢献するとは思われていなかった．

ところが 2006 年になって,「Crazy Stone」[5] と「MoGo」[20] という囲碁 AI が，原始モンテカルロ法を発展させたモンテカルロ木探索を採用し，他のプログラムを圧倒した．Crazy Stone と MoGo が登場してから 2012 年ごろまで，モンテカルロ木探索を用いた囲碁 AI は様々なテクニックを用いることで順調に強くなり，プロプレイヤー上位が相手でも 4 個の置石のハンデキャップ（4 段級差のハンデキャップ）をもらえば五分以上の勝負ができるようになった [21].

しかし，2012 年以降，AlphaGo[6] が登場するまでは，囲碁 AI の目に見える進歩は殆ど無かった．2012 年頃から 2015 年頃まで，当時トップだった囲碁 AI「Zen」のインターネット対局場での段位は「5d」から殆ど上がらず [22]，定期的に行われていた人間上位プレイヤーとの公式戦でもハンデキャップは殆ど減らなかった [21]．囲碁 AI がプロレベルのプレイヤーと対局するためには 3 個から 4 個の置石のハンデキャップが必要な状況を脱することが出来ず，2015 年時点では，人間トップを超える AI を作るのにはまだ時間がかかるという見方も多かった．

## 2.4 歴史のまとめ

囲碁 AI の研究が始まってから 2015 年までの状況を表 2.1 にまとめる．

本研究の最初のテーマである「Simulation Adjusting」は，2013 年から，モンテカルロシミュレーションの学習手法を工夫することでこの状況を打破しようと始めた研究である．この手法は，モンテカルロ法を用いた囲碁 AI の打つ手を，モンテカルロシミュレーション部分を調整することで上級プレイヤー

表 2.1: 囲碁 AI の進歩（2015 年まで）.

| | |
|---|---|
| 1960 年代 | 囲碁 AI の研究が始まる |
| 2000 年代前半 | 囲碁 AI は平均的なアマチュアプレイヤー未満の強さ |
| 2006 年 | 囲碁におけるモンテカルロ木探索の有効性が分かり始める |
| 2012 年から 2015 年 | 囲碁 AI がアマチュア上位レベルになったものの，そこで停滞 |

の打つ手に近づけるというものである．当時はモンテカルロシミュレーションの精度がまだ悪かったため，この手法によりモンテカルロシミュレーションを改善していくことが次のブレイクスルーにつながると期待された．

## 2.5 用語補足

この節では，これまでに用いた用語の補足を述べる．

**定義 2.5.1** (二人零和有限確定完全情報ゲーム)．二人零和有限確定完全情報ゲームとは，以下の要素を持ったゲームのことである [23]．ここでは，スコアや勝敗などで優劣を競う行為をゲームと呼ぶ．

- 二人：ゲームを行うプレイヤーが二人のゲーム．ゲーム理論でいうプレイヤーとはゲームを行う際にゲームの着手を決定する，意思決定する主体を指す．
- 零和：ゲーム上，プレーしている全プレイヤーの利得の合計が常にゼロ，または個々のプレイヤーの指す手の組み合わせに対する利得の合計が全て一定の数値（零和）となるゲーム．利得とはプレイヤーがゲーム終了時（あるいはターンの終了時）に獲得する状況に対する評価である．
- 有限：そのゲームにおける各プレイヤーの可能な手の組み合わせの総数が有限であるゲーム．一般に各種ボードゲームやカードゲームはゲームの途中の状態が理論上有限であるため，ある状態から別の状態に変わり，そこからまた元の状態に戻るといった反復が無限に繰り返されない限り有限のゲームとなる．
- 確定：プレイヤーの着手以外にゲームに影響を与える偶然の要素が入り込まないという意味．
- 完全情報：各プレイヤーが自分の手番において，これまでの各プレイヤーの行った選択（あるいは意思決定）について全ての情報を知ることができるゲーム．

チェスは，二人零和有限確定完全情報ゲームの代表的なものである．奇数番目に着手するプレイヤーを先手と呼ぶ．1 手目，つまり「先」に着手するプレイヤーである．それに対し，偶数番目に着手するプレイヤーを後手と呼ぶ．2 手目，つまり「後」に着手するプレイヤーである．また，そのようなゲームにおいて，勝負の成り行きを局面と呼ぶ．例えば，チェスにおける局面とは，盤上の駒の配置と次にどちらのプレイヤーが着手するかということである．ルール上打てる手のことを合法手と呼ぶ．また，形勢とは，その局面がどちらのプレイヤーにとってどれくらい勝ちやすいかということである．

**定義 2.5.2** (静的盤面評価関数)．「ある局面を入力として受け取り，その形勢が，先手，後手，どちらにとってどれくらい勝ちやすいかを，局面の状況のみから数値化して返す関数」のことを静的盤面評価関数と呼ぶ．

例えば，形勢が少し有利なら $+100$，少し不利なら $-100$，ほぼ勝ちなら $+9000$，ほぼ負けなら $-9000$ のように形勢を数値化する．

**定義 2.5.3** (Min-Max 型探索)．Min-Max 型の探索というのは，ある決まった手数まで先の，とりうる手順全てを展開し，末端でのそのゲームの結果，あるいは静的盤面評価関数の値を基にして,「相手手番のときは自分にとって最小の評価値（Min）の手を，自分手番のときは自分にとって最大の評価値（Max）の手を選んだ場合」に得られる評価値が最大になる手を調べる探索手法である．

図 2.1 を使って説明する．まず，ルートノードは自分手番なので Max を選ぶことになり，その下のノードは相手手番なので Min を選ぶことになる．そしてさらにその下のノードで静的盤面評価関数が呼ばれるものとする．左側の Min ノードでは，子ノードの評価値 $1, 3$ の中で最小の値である 1 が，右側の Min ノードでは，子ノードの評価値 $-1, -2$ の中で最小の値である $-2$ が選ばれる．その後 Max ノードでは，子ノードの評価値 $1, -2$ のうち最大の値である 1 が選ばれる．現在の自分の手番では評価値が 1 であり，最善手は左の子を選ぶことであることが分かる．

なお，二人完全情報確定零和ゲームの AI においては，Min-Max 型の探索の中でも，通常は効率の良い $\alpha$-$\beta$ 探索等が用いられる．

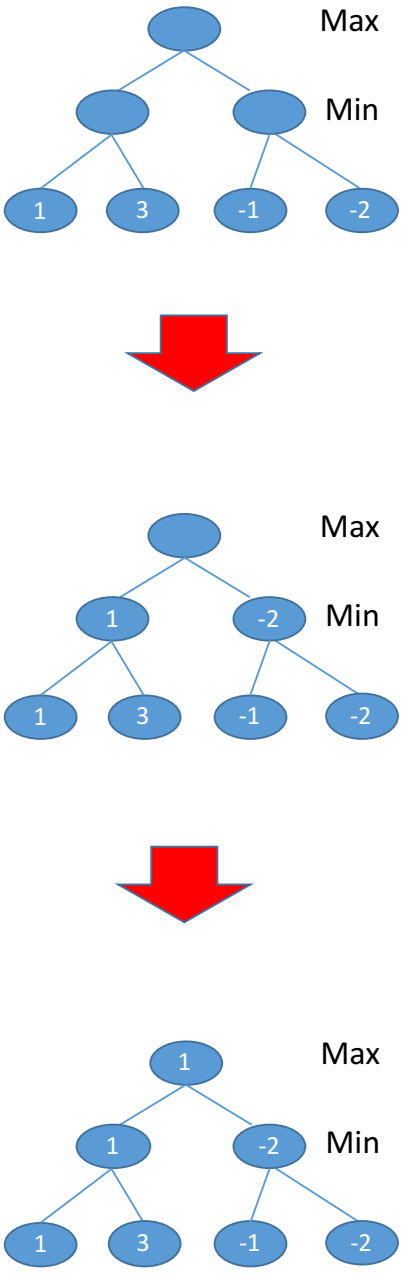



図 2.1: Min-Max 探索のイメージ.

# 第3章　モンテカルロ木探索

## 3.1　モンテカルロ木探索の基礎

モンテカルロ木探索の基礎は原始モンテカルロ法である．

**定義 3.1.1** (囲碁における原始モンテカルロ法)．評価したい局面から，擬似乱数を用いて全ての合法手から何らかの確率分布に従って着手を選んで打つ操作を繰り返して終局まで打つことを何度も行い，結果の平均を評価値とすることで，局面を評価する手法．

擬似乱数を用いて着手を選び，終局まで打つことを囲碁における「モンテカルロシミュレーション」と呼ぶ．局面を入力として，モンテカルロシミュレーションにおける着手の確率分布を出力する関数またはそのパラメータを方策と呼ぶ．

なお，ここで注意しておくべきことが一つある．囲碁 AI において原始モンテカルロ法や後述するモンテカルロ木探索を用いる場合は，囲碁 AI 内部では局面を中国ルール [1] で処理する．「終局まで打つ」ときの「終局まで」というのは，人間同士の対局の場合とは異なり，「全ての死に石を盤上から取り上げ，さらに自分の眼を潰す以外合法手が無くなるまで」ということを意味する．

日本ルールで局面を扱うと，自分の陣地と考えられる場所の中や相手の陣地と考えられる場所の中に余計な着手をすると損になってしまい，結果が変わることがある．そのため，「最後まで」打つと結果が変わってしまう．中国ルールならその心配が殆ど無いため，囲碁 AI 内部では中国ルールを用いるのである．

原始モンテカルロ法の概略は図 3.1 にあるようなものである．ただし，オリジナルのアイディアではすべての合法手に対して原始モンテカルロ法を適用するが，この図では，合法手二つについて原始モンテカルロ法を行っている．まず現在の局面から左の子の候補手を打った後，擬似乱数で最後まで打つことを何度も繰り返し，勝率が $X\%$ だったとする．同様に現在の局面から右の子の候補手を打った後，擬似乱数で最後まで打つことを何度も繰り返し，勝率が $Y\%$ だったとする．「AI は，もし $X > Y$ なら現局面の左の子の候補手を，$X < Y$ なら現局面の右の子の候補手を選ぶ」という仕組みで，静的盤面評価関数が無くてもコンピュータに次の一手を選ばせることができる．

原始モンテカルロ法は，「ある局面が『擬似乱数で最後まで打つ』操作を繰り返して勝率が高いということは，おそらくその局面は通常のプレイヤーにとっても勝ちやすいのであろう」という仮定に基づいた手法であると言える．

そしてモンテカルロ木探索の定義は以下の通りである [17]．

**定義 3.1.2** (モンテカルロ木探索)．以下の 1 から 4 の操作を繰り返す探索法のことである．

1. 可能な手から一手選んで木を下る操作を，木の末端に到達するまで繰り返す．
2. その末端ノードの訪問回数が閾値に達していたら，そのノードの子ノードをさらに展開（木に追加）し，その中から一手選んで木を下る．
3. モンテカルロシミュレーションを 2 で到達したノードから一回行う．
4. モンテカルロシミュレーションの結果を，1 と 2 で辿ってきたノードに反映する．

[1]「付録:囲碁のルールについて」を参照のこと

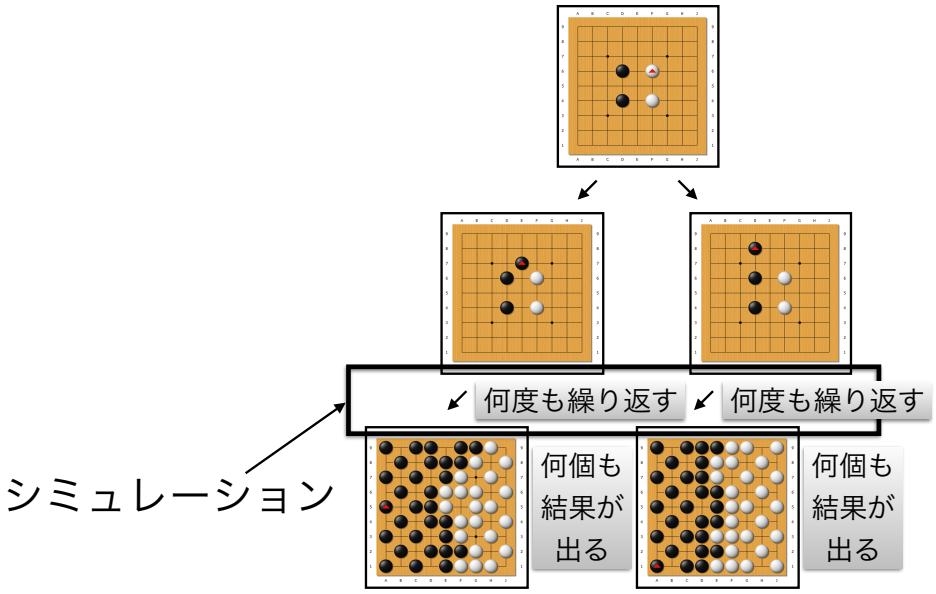



図 3.1: 囲碁における原始モンテカルロ法.

モンテカルロ木探索とは，大まかに言えば，原始モンテカルロ法を木探索と組み合わせた手法である．ここで言う「木」とは,「ゲーム木」の事であり，すなわち，ゲームにおける何手か先までの局面への分岐からなる，現局面をルートとする木構造の事である．モンテカルロ木探索の概略を図 3.2 に，アルゴリズムを Algorithm 1 に示す．図 3.2 の 1 から 4 は定義の 1 から 4 と対応している．

なお，子ノードの選択の基準としてよく用いられる UCB (Upper Confidence Bound) 値の計算式は以下の通りである [24]：

$$UCB_i = \bar{w}_i + C\sqrt{\frac{\log n}{n_i}}. \tag{3.1}$$

ただし，$i$ は子ノードの番号を表し $\bar{w}_i$ はこの子ノードの勝率，$C$ は実験で調整する値，$n_i$ はこの子ノードの訪問回数，$n = \sum_i n_i$ である．UCB 値は，勝率が高くて有望そうな場合に高くなり，また試行回数が少ない場合にも高くなる．

この値は元々，当たる確率の異なる多数のスロットマシンと多数のコインがあり，当たる確率の具体的な値が不明なときに，どのように試行していけば最終的な利得が最も高くなるかを理論的に求めることによって得られたものである.「UCB 値が最も高いスロットマシンに一個コインを入れて UCB 値を更新する」という操作を繰り返すと，得られる利得が理論上限に収束するのである．

モンテカルロ木探索を用いると，単純に原始モンテカルロ法を行うより棋力が高くなることが知られている．また，モンテカルロシミュレーション方策に関しても様々な研究がされており，そのうちの一つが次節で述べる Coulom の手法である．

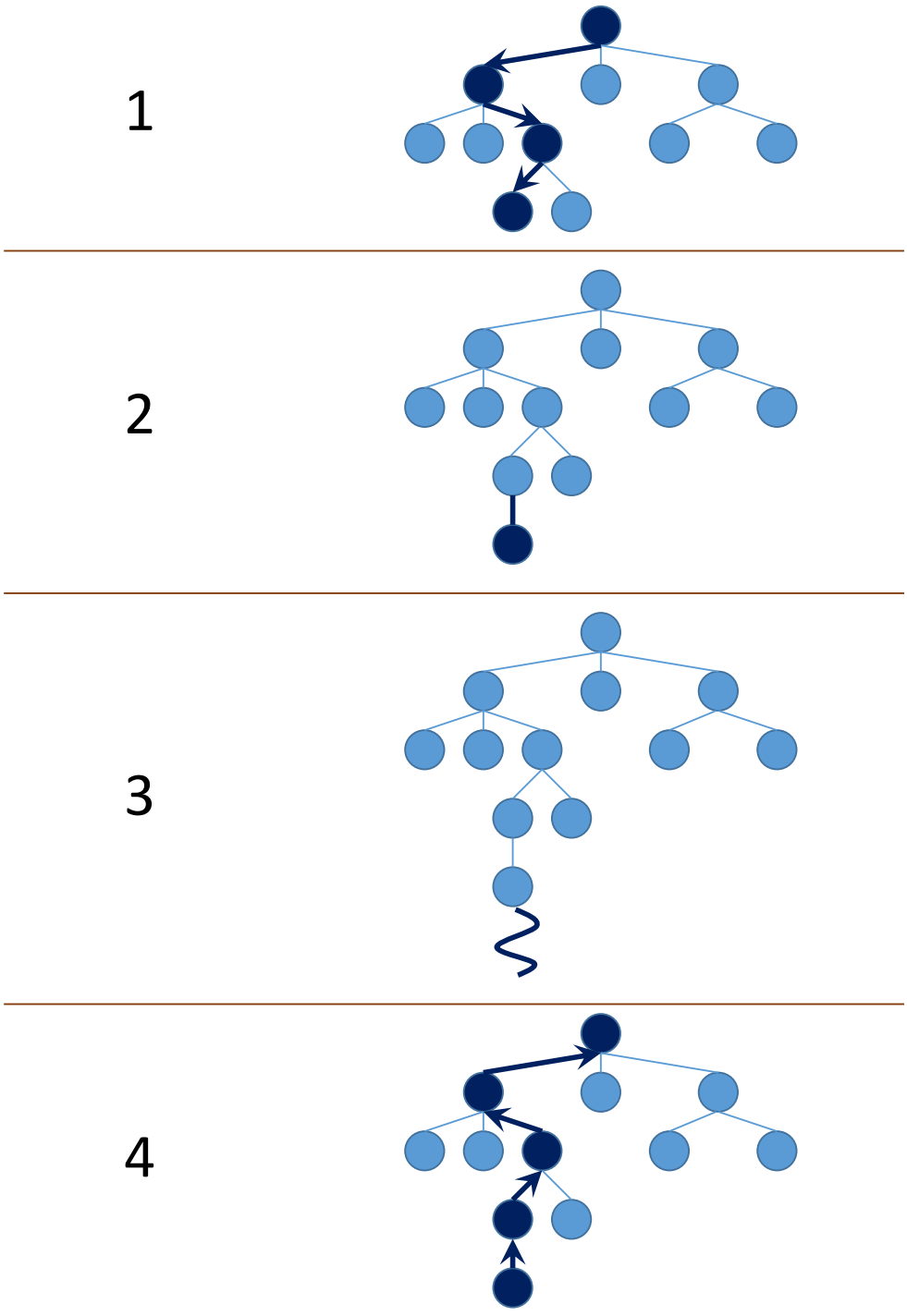



図 3.2: モンテカルロ木探索の概略.

**Algorithm 1** モンテカルロ木探索

```
ROOTNODE ← 初期値
for L = 0 からシミュレーション回数上限まで do
  TEMPNODE ← ROOTNODE
  for TEMPNODE が子ノードを持っている間 do
    CHILDNODE ← TEMPNODE の子ノードから一つ選んだもの /* 例えば UCB 等で */
    TEMPNODE ← CHILDNODE
  end for
  if TEMPNODE の訪問回数が閾値に達していたら then
    TEMPNODE の子ノードを展開
    CHILDNODE ← TEMPNODE の子ノードから一つ選んだもの /* 例えば UCB 等で */
    TEMPNODE ← CHILDNODE
  end if
  w ← TEMPNODE から行ったシミュレーションの結果 /* もし勝ちなら w = 1，負けなら w = 0
  */
  for TEMPNODE が親ノードを持っている間 do
    TEMPNODE の訪問回数をインクリメント
    TEMPNODE の勝ち数に w を加える
    w = −w
    TEMPNODE ← TEMPNODE の親ノード
  end for
end for
ROOTNODE の子ノードの勝率を見て，一番高い勝率の手を選ぶ
```

## 3.2 Coulom の手法

Crazy Stone は，モンテカルロシミュレーション方策を人間の棋譜から機械学習している [5]．Crazy Stone のモンテカルロシミュレーション方策は，モンテカルロシュミレーション中での着手の確率分布を，BT モデル（Bradley-Terry model）に基づいた Move Prediction によって決めるものである．

BT モデルでは，以下のように各候補手の確率が決められる．まず，候補手が 2 つ，特徴 $i$ だけを持つ着手と特徴 $j$ だけを持つ着手があるとし，特徴 $i$ の強さを正の数 $\gamma_i$，特徴 $j$ の強さを正の数 $\gamma_j$ とすると，特徴 $i$ を持つ着手が選ばれる確率が

$$\frac{\gamma_i}{\gamma_i + \gamma_j}$$

で表される．強い特徴 $i$ ほど大きな $\gamma_i$ を持つ．このモデルを候補手 2 個ではなく $n$ 個の候補手からの選択に適用し，$x$ 番目の候補手が特徴 $x$ のみを持つとすると，$i$ 番目の候補手が選ばれる確率は

$$\frac{\gamma_i}{\gamma_1 + \gamma_2 + \cdots + \gamma_n}$$

で表される．さらに各候補手が特徴を複数持つ場合にも適用すると，候補手 $a$（特徴 1, 2, 3），候補手 $b$（特徴 2, 4），候補手 $c$（特徴 1, 5, 6, 7）の 3 つの候補手から $a$ が選ばれる確率は，

$$\frac{\gamma_1\gamma_2\gamma_3}{\gamma_1\gamma_2\gamma_3 + \gamma_2\gamma_4 + \gamma_1\gamma_5\gamma_6\gamma_7}$$

で表される．

そのパラメータは Hunter [25] の MM 法（Minorization-Maximization method）によって学習されている． MM 法について簡単に述べる．学習したい値が $\gamma_1, \gamma_2, \ldots, \gamma_n$ と $n$ 個存在して，手の選択が $N$ 回

あるとする．$j$ 回目の選択の尤度は

$$\frac{A_{ij}\gamma_i + B_{ij}}{C_{ij}\gamma_i + D_{ij}}$$

と，$\gamma_i$ に関係ない値 $A_{ij}, B_{ij}, C_{ij}, D_{ij}$ を使って書ける．例えば $j$ 回目の選択が「$a$ が $b, c$ に勝ち」の場合，$i = 1$ として上記のようにその確率を表すと，

$$\frac{\gamma_2\gamma_3 \cdot \gamma_1}{(\gamma_2\gamma_3 + \gamma_5\gamma_6\gamma_7)\gamma_1 + \gamma_2\gamma_4} \quad (A_{ij} = \gamma_2\gamma_3, B_{ij} = 0, C_{ij} = \gamma_2\gamma_3 + \gamma_5\gamma_6\gamma_7, D_{ij} = \gamma_2\gamma_4)$$

と表せる．

$E_j = C_{ij}\gamma_i + D_{ij}$，$W_i$ を $\gamma_i$ が勝った回数として，MM 法では以下の式：

$$\gamma_i \leftarrow \frac{W_i}{\sum_{j=1}^{N} \frac{C_{ij}}{E_j}}$$

に従った更新を $\gamma_i$ が収束するまで繰り返す．このようにすることで，$N$ 回の手の選択全体の尤度が最も高くなる $\gamma_i$ を決めることが出来る．

Bradley-Terry モデルに基づいた Move Prediction は Coulom [5] の発表当時としては最高精度（テストデータの棋譜との一致率が最高）であった．この手法を以下では「Coulom の手法 [5]」と呼ぶ．

この Move Prediction をモンテカルロシミュレーションの方策に用いた AI は，一様分布によるモンテカルロシミュレーションを用いた AI より遥かに強いことが実験的に示されていた．

なお MoGo は，候補手を中心とする上下左右斜め 1 点ずつを見る $3 \times 3$ パターンを手作りして，殆どの場合に相手の直前の手の周辺のみを候補手とするモンテカルロシミュレーションを用いていた．

次節で述べる Simulation Balancing はモンテカルロシミュレーション部の改良に該当する既存研究である．そして，第 4 章から第 7 章で述べるように，本研究で取り組んだ Simulation Adjusting も，モンテカルロシミュレーション部の改良に該当する．

## 3.3 Simulation Balancing

「Simulation Balancing」は Silver and Tesauro [9] で提案された手法である．ある局面において，教師となる強い AI がモンテカルロ法によって出した勝率を理想値とする．そして，強くしたい AI がその局面でモンテカルロ法によって出す勝率がその理想値となるように，モンテカルロシミュレーションを改善していくことを目指すものである．

ここではその理論を要約して述べる．

$\mathcal{F}$ を，着手が任意の局面で持ちうる特徴の集合とし，各 $i \in \mathcal{F}$ について，局面 $s$ と着手 $a$ についての指標関数を以下のように定義する：

$$\phi_i(s, a) = \begin{cases} 1 & \text{着手 } a \text{ が局面 } s \text{ において特徴 } i \text{ を持つ場合,} \\ 0 & \text{それ以外の場合.} \end{cases}$$

次に，$\theta \in \mathbb{R}^{\mathcal{F}}$ でパラメータ化された関数を作る．ここで，$\theta_i$ は特徴 $i$ の重みを表す．着手 $a$ が局面 $s$ で選ばれる確率 $\pi_\theta(s, a)$ が以下のように表現されるモデルを考える：

$$\pi_\theta(s, a) = \frac{\exp(\sum_{i \in \mathcal{F}} \phi_i(s, a)\theta_i)}{\sum_{b \in M(s)} \exp(\sum_{i \in \mathcal{F}} \phi_i(s, b)\theta_i)}.$$

ここで，$M(s)$ は $s$ における合法手の集合である．

これは soft-max 関数 [26] と呼ばれるものの一種である．soft-max 関数とは一般的に以下の関数である：

$$f(a) = \frac{\exp w_a}{\sum_b \exp w_b}.$$

$w_a = \sum_{i \in \mathcal{F}} \phi_i(s,a)\theta_i$，$w_b = \sum_{i \in \mathcal{F}} \phi_i(s,b)\theta_i$ と置き換えると今回の形になる．

なお，$\gamma_a = \exp(\sum_{i \in \mathcal{F}} \phi_i(s,a)\theta_i)$ と考えると，このモデルは BT モデルの特殊な場合になっている．

そして，$P_\theta(\xi)$ を，モンテカルロシミュレーション方策のパラメータ $\theta$ の下での，モンテカルロシミュレーション $\xi$ の確率とする．モンテカルロシミュレーション $\xi$ の報酬は以下のように与えられるものとする：

$$z(\xi) = \begin{cases} 1 & \xi\text{が勝っている局面で終了した場合,} \\ 0 & \text{それ以外の場合.} \end{cases}$$

Simulation Balancing では，局面 $s$ からパラメータ $\theta$ に基づいたシミュレーション方策によってシミュレーションを実行し，その結果を基に $\theta$ を調整していくことを繰り返す．$\chi_\theta(s)$ で，実行されたシミュレーションの集合を表すものとする．

まず理想的な場合を考えるので可算無限回のシミュレーションを実行したものとする．このときパラメータ $\theta$ による $s$ の勝率 $V(s,\theta)$ を以下のように表すことができる:

$$V(s,\theta) = \sum_{\xi \in \chi(s)} P_\theta(\xi) z(\xi).$$

Simulation Balancing では，何らかの教師プログラムを用意し，局面 $s$ で出すべき勝率の理想値 $V^*(s)$ を計算しておく，そして，各 $s$ について，$V(s,\theta)$ が $V^*(s)$ に近づくように，$\theta$ を確率的勾配降下法で計算するのである．Simulation Balancing における最適化問題は

$$\min_\theta \mathbb{E}_\rho \left[ \left( V(s,\theta) - V^*(s) \right)^2 \right]$$

と定義されている．ただし，$\rho$ は，教師プログラムと強くしたいプログラムの比較に用いる棋譜データにおける局面 $s$ の分布である．

Silver and Tesauro [9] では，$V(s,\theta)$ の $\theta$ の各要素に対する偏微分が求められること，及びそれを用いて目的関数の偏微分が求められることが示されており，さらに 5 路盤と 6 路盤において目的関数の減少が確認されている．

その後，Huang ら [10] で 9 路盤での Simulation Balancing の実験が試みられている．Huang ら [10] では，教師となるプログラムとして，強くしたいプログラムのモンテカルロシミュレーション回数を増やしたものを用いた実験を行なっている．しかし，GNU Go [27] 相手のテストでは，Simulation Balancing の結果得られたモンテカルロシミュレーション方策を用いても，勝率は殆ど上がっていない．

# 第4章　Simulation Adjusting

## 4.1　動機

Coulom [5] の手法は，Crazy Stone [5], Ray [28], AlphaGo [6] を含め，現在多くの囲碁 AI で用いられている．

ところが，囲碁に対するモンテカルロ木探索で良く用いられている UCB は，Auer [24] による「多腕バンディット問題」の理論的な基盤があるのに対し，Coulom [5] の手法をモンテカルロシミュレーションに用いる効果について理論的な証明は無い．

Coulom [5] の手法は「盤面から次の一手を予測する精度を上げる」手法である．Coulom [5] の手法を用いて学習した方策をモンテカルロ木探索で用いたときに，結果として返される手が良い手である保証はどこにもない．

そこで，直接「良い」モンテカルロシミュレーション方策を目指して学習することで，Coulom [5] の手法のものより良いモンテカルロシミュレーション方策を作れるのではないかという発想が出てくる．言い換えると，モンテカルロ木探索 AI の中で使われたときに，その AI が返す手が良くなるようなモンテカルロシミュレーション方策は作れるだろうかということである．

「モンテカルロシミュレーション方策自体の棋力」，すなわち「モンテカルロシミュレーション方策を対局プログラムとして用いた時の強さ」を強くすれば，モンテカルロ木探索 AI 全体も強くなると考える人もいるかもしれない．しかし，そうとは限らないことが知られている．

Gelly and Silver [20] で以下のような実験が行われた．まず 2 つのシミュレーションの方策 $\pi_{RLGO}$ と $\pi_{MOGO}$ を用意する．方策 $\pi_{RLGO}$ と $\pi_{MOGO}$ は局面を受け取ったら（確率的に）合法手を返すものであるので，$\pi_{RLGO}$ と $\pi_{MOGO}$ を戦わせることが可能である．$\pi_{RLGO}$ と $\pi_{MOGO}$ を戦わせたところ，$\pi_{RLGO}$ の方が有意に強かった．しかし，$\pi_{RLGO}$ と $\pi_{MOGO}$ をモンテカルロ木探索の方策として用いたところ，$\pi_{MOGO}$ を用いたときの方が勝率が高かった．

3.3 節で述べた Simulation Balancing は,「良い」モンテカルロシミュレーション方策を，強い教師プログラムの助けを得て作る手法である．Simulation Balancing から着想を得て，本研究では Simulation Adjusting という，モンテカルロシミュレーション方策を学習する新手法を提案した．

Simulation Adjusting は，モンテカルロ木探索を用いた AI が打つ手が，上級者の対局中に打った手と一致するように，Simulation Balancing と同様の勾配法を用いて学習していく手法である．

Coulom [5] の手法と異なり，これは高精度の Move Prediction ではなく，モンテカルロ木探索を用いた AI がそれぞれの局面において良い手を返すモンテカルロシミュレーション方策を目指す手法である．その点に関しては Simulation Balancing と似ている．ただし，Simulation Balancing と違って，学習には強い教師 AI を必要とせず，代わりに棋譜から学習する．AI より強い人間の棋譜が手に入る場合，棋譜から学習することで既存の AI を超えられる可能性があるのがメリットである．逆に，AI より強い人間の棋譜があまり手に入らない場合，本手法を適用するのは困難である．

## 4.2　理論

モンテカルロ木探索，またはモンテカルロ法を適用できるようなゲームを考える．

実際にはモンテカルロ法自体は二人以上の零和ゲームにも適用できるが，Simulation Adjusting の現在の研究対象は二人零和完全情報ゲーム（有限である必要や確定である必要は無い）であるので，二人

零和完全情報ゲームを考える．そのようなゲームにおける局面の集合を $G$ とし，ある局面 $s \in G$ における合法手の集合を $M(s)$ で表す．合法手 $a \in M(s)$ はいくつかの特徴を持っているものとする．例えばチェスならばその手によって取れる駒やその座標に対する他の駒の効き，囲碁なら石を取れるかどうかや直前手からの距離などが挙げられる．

$\mathcal{F}$ を，着手が任意の局面で持ちうる特徴の集合とし，各 $i \in \mathcal{F}$ について，局面 $s$ と着手 $a$ についての指標関数を以下のように定義する：

$$\phi_i(s,a) = \begin{cases} 1 & \text{着手 } a \text{ が局面 } s \text{ において特徴 } i \text{ を持つ場合,} \\ 0 & \text{それ以外の場合.} \end{cases}$$

次に，$\theta \in \mathbb{R}^{\mathcal{F}}$ でパラメータ化された関数を作る．ここで，$\theta_i$ は特徴 $i$ の重みを表す．着手 $a$ が局面 $s$ で選ばれる確率 $\pi_\theta(s,a)$ は以下のように表現される：

$$\pi_\theta(s,a) = \frac{\exp(\sum_{i\in\mathcal{F}} \phi_i(s,a)\theta_i)}{\sum_{b\in M(s)} \exp(\sum_{i\in\mathcal{F}} \phi_i(s,b)\theta_i)}.$$

このゲームについて，エキスパートのプレイした棋譜が手に入るものと仮定し，その棋譜の集合を $\mathcal{G}$ とする．通常，棋譜は「一ゲームを最初から最後までプレイした記録」であるが，ここでは棋譜をまとめて扱い，棋譜の集合 $\mathcal{G}$ をタプル $(s,a^*)$ の集合として考える． ここで，$s$ は対局中に現れる局面，$a^*$ は人間の上級者が $s$ でプレイした着手とする． Coulom [5] の手法では，これを基に以下のような最尤推定を考えている：

$$\max_\theta \prod_{(s,a^*)\in\mathcal{G}} \pi_\theta(s,a^*). \tag{4.1}$$

そして，(4.1) を MM 法を用いることで最適化することを提案している．理論的には，この手法は「モンテカルロシミュレーション中での」局面に対する着手が，同じ局面で上級者が選んだ着手と一致する確率を上げる．

対照的に，本研究ではパラメータ $\theta$ を,「モンテカルロシミュレーション後に」選ばれる着手が人間の上級者の着手と一致する確率を上げることを考える．

このために，(4.2)，(4.3) のように最適化問題を設計する．はじめに，$s \circ a$ で，局面 $s$ において着手 $a$ をプレイした後の局面を表すものとする．そして，$n_\theta(\xi)$ を，パラメータ $\theta$ の下で実行された，モンテカルロシミュレーション $\xi$ の回数とする．また，$P_\theta(\xi)$ を，パラメータ $\theta$ の下での，モンテカルロシミュレーション $\xi$ の確率とする．モンテカルロシミュレーション $\xi$ の報酬は以下のように与えられるものとする：

$$z(\xi) = \begin{cases} 1 & \xi\text{が勝っている局面で終了した場合,} \\ 0 & \text{それ以外の場合.} \end{cases}$$

また，$\chi_\theta(s,a)$ で，局面 $s \circ a$ からパラメータ $\theta$ によって実行されたシミュレーションの集合を表すものとする．Simulation Balancing と同様に，まずは可算無限回のシミュレーションが実行されたものと考える．このときパラメータ $\theta$ による $s \circ a$ の勝率 $V(s,a,\theta)$ を以下のように表すことができる:

$$\begin{aligned} V(s,a,\theta) &= \left|\frac{1}{\chi_\theta(s,a)}\right| \sum_{\xi\in\chi_\theta(s,a)} n_\theta(\xi) z(\xi) \\ &\sim \sum_{\xi\in\chi_\theta(s,a)} P_\theta(\xi) z(\xi). \end{aligned}$$

Simulation Adjusting はパラメータ $\theta$ を，$s$ から $\theta$ に基づいてモンテカルロシミュレーションを行った後に $a^*$ が選ばれる確率が，候補手の中で最大になるように調整する．我々は以下の 2 つの最適化問題を考えた．

- 自乗誤差の期待値を最小化する最適化問題

$$\min_{\theta} \mathbb{E}_{\rho}\left[\left(\max_{a} V(s,a,\theta) - V(s,a^*,\theta)\right)^2\right]. \tag{4.2}$$

- 正解以外の勝率を小さくする最適化問題

$$\min_{\theta} \mathbb{E}_{\rho}\left[\sigma\left(\max_{a\neq a^*} V(s,a,\theta) - V(s,a^*,\theta)\right)\right]. \tag{4.3}$$

ここで，$\sigma(x) = (1+e^{-x})^{-1} - 1/2$ はシグモイド関数に定数を加えたものである．

以下では，これらの目的関数の性質を述べる．以下，log は自然対数を表すものとする．まず，各 $i \in \mathcal{F}$ に対して，

$$\psi_i(s,a,\theta) = \frac{\partial}{\partial \theta_i} \log \pi_\theta(s,a)$$

とする．

**補題 1.** 各 $i \in \mathcal{F}$ について，

$$\psi_i(s,a,\theta) = \phi_i(s,a) - \sum_{b\in M(s)} \pi_\theta(s,b)\phi_i(s,b)$$

が成り立つ．

証明：定義より，以下が成り立つ：

$$\begin{aligned}
\psi_i(s,a,\theta) &= \frac{\partial}{\partial \theta_i} \log \pi_\theta(s,a). \\
&= \frac{\partial}{\partial \theta_i} \log \frac{\exp(\sum_{j\in\mathcal{F}} \phi_j(s,a)\theta_j)}{\sum_{b\in M(s)} \exp(\sum_{j\in\mathcal{F}} \phi_j(s,b)\theta_j)} \\
&= \frac{\partial}{\partial \theta_i} \sum_{j\in\mathcal{F}} \phi_j(s,a)\theta_j - \frac{\partial}{\partial \theta_i} \log\left(\sum_{b\in M(s)} \exp\left(\sum_{j\in\mathcal{F}} \phi_j(s,b)\theta_j\right)\right) \\
&= \phi_i(s,a) - \frac{\frac{\partial}{\partial \theta_i}\sum_{b\in M(s)} \exp(\sum_{j\in\mathcal{F}} \phi_j(s,b)\theta_j)}{\sum_{b\in M(s)} \exp(\sum_{j\in\mathcal{F}} \phi_j(s,b)\theta_j)} \\
&= \phi_i(s,a) - \frac{\sum_{b\in M(s)} \phi_i(s,b)\exp(\sum_{j\in\mathcal{F}} \phi_j(s,b)\theta_j)}{\sum_{b\in M(s)} \exp(\sum_{j\in\mathcal{F}} \phi_j(s,b)\theta_j)}.
\end{aligned}$$

$\pi_\theta(s,a)$ の定義より，以下が成り立つ：

$$\psi_i(s,a,\theta) = \phi_i(s,a) - \sum_{b\in M(s)} \pi_\theta(s,b)\phi_i(s,b).$$

□

各 $a \in M(s)$ と各 $i \in \mathcal{F}$ に対して以下のように定義する：

$$\eta_i(s,a,\theta) = \frac{\partial}{\partial \theta_i} V(s,a,\theta).$$

局面 $s_1 = s\circ a$ から始まるモンテカルロシミュレーション $\xi$ は，局面 $s_1, s_2, \ldots, s_T$（ただし $s_{t+1} = s_t \circ a_t$ $(t = 1, \ldots, T-1)$），と結果 $z$ から成ると考えられる．そのとき，以下の補題が成立する：

**補題 2.** 各 $i \in \mathcal{F}$ について,

$$\eta_i(s,a,\theta) = \mathbb{E}_{P_\theta}\left[z\sum_{t=1}^{T}\psi_i(s_t,a_t,\theta)\right]$$

が成立する.

証明：$\eta_i(s,a,\theta)$ の定義より,

$$\eta_i(s,a,\theta) = \frac{\partial}{\partial\theta_i}V(s,a,\theta) \tag{4.4}$$

が成り立つ.

$V(s,a,\theta)$ の定義より，以下が成立する：

$$((4.4)\text{ の右辺}) = \frac{\partial}{\partial\theta_i}\sum_{\xi\in\chi_\theta(s,a)}P_\theta(\xi)z(\xi). \tag{4.5}$$

和を取る操作と偏微分を入れ替え，そしてモンテカルロシミュレーション方策が $\pi_\theta$ に基づいていることから，以下が成立する：

$$((4.5)\text{ の右辺}) = \sum_{\xi\in\chi_\theta(s=s_0,a=a_0)}\frac{\partial}{\partial\theta_i}(\pi_\theta(s_1,a_1)\cdots\pi_\theta(s_T,a_T))z(\xi). \tag{4.6}$$

積の微分の公式より，以下が成立する.

$$\begin{aligned}
&((4.6)\text{ の右辺})\\
=&\sum_{\xi\in\chi_\theta(s=s_0,a=a_0)}\left(\pi_\theta(s_1,a_1)\cdots\pi_\theta(s_T,a_T)\left(\frac{\frac{\partial}{\partial\theta_i}\pi_\theta(s_1,a_1)}{\pi_\theta(s_1,a_1)}+\cdots+\frac{\frac{\partial}{\partial\theta_i}\pi_\theta(s_T,a_T)}{\pi_\theta(s_T,a_T)}\right)\right)z(\xi)\\
=&\sum_{\xi\in\chi_\theta(s=s_0,a=a_0)}\left(P_\theta(\xi)\left(\frac{\frac{\partial}{\partial\theta_i}\pi_\theta(s_1,a_1)}{\pi_\theta(s_1,a_1)}+\cdots+\frac{\frac{\partial}{\partial\theta_i}\pi_\theta(s_T,a_T)}{\pi_\theta(s_T,a_T)}\right)\right)z(\xi).
\end{aligned} \tag{4.7}$$

対数関数の微分の公式より，以下が成立する.

$$((4.7)\text{ の右辺}) = \mathbb{E}_{P_\theta}[z\sum_{t=1}^{T}\frac{\partial}{\partial\theta_i}\log\pi_\theta(s_t,a_t)]. \tag{4.8}$$

$\psi_i(s_t,a_t,\theta)$ の定義より，以下が成立する.

$$((4.8)\text{ の右辺}) = \mathbb{E}_{P_\theta}[z\sum_{t=1}^{T}\psi_i(s_t,a_t,\theta)].$$

□

また，目的関数の偏微分を以下のように計算できる.

**定理 3.** 以下の命題が成立する：

- 自乗誤差の期待値を最小化する最適化問題：

$$\begin{aligned}
&\frac{\partial}{\partial\theta_i}\mathbb{E}_\rho\left[\left(\max_a V(s,a,\theta)-V(s,a^*,\theta)\right)^2\right]\\
&\quad=\mathbb{E}_\rho\left[2\left(V(s,a^{\max},\theta)-V(s,a^*,\theta)\right)\left(\eta_i(s,a^{\max},\theta)-\eta_i(s,a^*,\theta)\right)\right],
\end{aligned}$$

ただし， $a^{\max}=\mathrm{argmax}_a V(s,a,\theta)$,

- 正解以外の勝率を小さくする最適化問題：

$$\frac{\partial}{\partial\theta_i}\mathbb{E}_\rho\left[\sigma\left(\max_{a\neq a^*}V(s,a,\theta)-V(s,a^*,\theta)\right)\right]$$
$$=\mathbb{E}_\rho\left[\sigma'\left(V(s,a^{\max},\theta)-V(s,a^*,\theta)\right)\left(\eta_i(s,a^{\max},\theta)-\eta_i(s,a^*,\theta)\right)\right],$$

ただし，$\sigma'$ は $\sigma$ の微分であり，$a^{\max}=\mathrm{argmax}_{a\neq a^*}V(s,a,\theta)$.

証明：正解以外の勝率を小さくする最適化問題についてのみ示す．自乗誤差の期待値を最小化するものも同様に証明できる．和を取る操作と偏微分を入れ替えることで，以下が成立する：

$$\frac{\partial}{\partial\theta_i}\mathbb{E}_\rho[\sigma(\max_{a\neq a^*}V(s,a,\theta)-V(s,a^*,\theta))]=\mathbb{E}_\rho[\frac{\partial}{\partial\theta_i}\sigma(\max_{a\neq a^*}V(s,a,\theta)-V(s,a^*,\theta))]. \tag{4.9}$$

ここで，$a^{\max}=\mathrm{argmax}_{a\neq a^*}V(s,a,\theta)$ とする．そのとき，以下が成立する：

$$((4.9)\text{ の右辺})=\mathbb{E}_\rho[\frac{\partial}{\partial\theta_i}\sigma(V(s,a^{\max},\theta)-V(s,a^*,\theta))]. \tag{4.10}$$

合成関数の微分の公式より，以下が成立する：

$$((4.10)\text{ の右辺})=\mathbb{E}_\rho[\sigma'(V(s,a^{\max},\theta)-V(s,a^*,\theta))\frac{\partial}{\partial\theta_i}(V(s,a^{\max},\theta)-V(s,a^*,\theta))]. \tag{4.11}$$

$\eta_i(s,a,\theta)$ の定義より，以下が成立する：

$$((4.11)\text{ の右辺})=\mathbb{E}_\rho[\sigma'(V(s,a^{\max},\theta)-V(s,a^*,\theta))(\eta_i(s,a^{\max},\theta)-\eta_i(s,a^*,\theta))].$$

□

# 第5章　自乗誤差の期待値を最小化する目的関数による実験

## 5.1　アルゴリズム

定理 3 より，偏微分の近似を計算する方法が分かり，Simulation Adjusting のアルゴリズムを組み立てることができる．

局面 $s$ から有限回のモンテカルロシミュレーションを実行するとし，$\bar{\chi}_\theta(s,a)$ で，その有限回のモンテカルロシミュレーションのうち，局面 $s \circ a$ からパラメータ $\theta$ に基づいて実行されたシミュレーションの集合を表すものとする．

その際，同じシミュレーションが二回現れないとすると，以下の近似が成り立つ：

$$\begin{aligned} V(s,a,\theta) \sim \bar{V}(s,a,\theta) &= \frac{1}{|\bar{\chi}_\theta(s,a)|} \sum_{\xi \in \bar{\chi}_\theta(s,a)} n_\theta(\xi) z(\xi) \\ &= \frac{1}{|\bar{\chi}_\theta(s,a)|} \sum_{\xi \in \bar{\chi}_\theta(s,a)} z(\xi). \end{aligned}$$

このとき，以下のように近似できる：

$$\begin{aligned} \eta_i(s,a,\theta) &= \mathbb{E}_{P_\theta}\left[ z \sum_{t=1}^{T} \psi_i(s_t,a_t,\theta) \right] \\ &\sim \frac{1}{|\bar{\chi}_\theta(s,a)|} \sum_{\xi \in \bar{\chi}_\theta(s,a)} z(\xi) \left( \sum_{t=1}^{T} \psi_i(s_t,a_t,\theta) \right). \end{aligned}$$

上記の議論に基づいて，Algorithm 2 のように Simulation Adjusting のアルゴリズムを組み立てることができる．

以下はこのアルゴリズムの注意点である．

添字 $\mathcal{F}$ は対応する変数が，各要素が特徴 $i \in \mathcal{F}$ によってインデックス付けされているベクトルであることを意味する．

$N$ は $a \in M(s)$ から始めるモンテカルロシミュレーションの回数である．

今回のバージョンでは，全ての $a$ についてモンテカルロシミュレーション回数は同じであると想定している．勿論，例えば UCB 値に基づいて学習器がモンテカルロシミュレーションを非一様に分散させることも考えられる．

$V$ を $\bar{V}$ で近似することにより，

$$a^{\max} \sim \bar{a}^{\max} = \operatorname*{argmax}_a \bar{V}(s,a,\theta),$$

として，最適化問題は以下のように近似計算される：

$$\min_\theta \sum_{(s,a^*) \in \mathcal{G}} \frac{1}{|\mathcal{G}|} \left( \bar{V}(s,\bar{a}^{\max},\theta) - \bar{V}(s,a^*,\theta) \right)^2.$$

**Algorithm 2** 自乗誤差の期待値を最小化するアルゴリズム

$\theta \leftarrow 0$
**for** $L = 0$ から反復回数上限 LOOPLIMIT まで **do**
  **for all** $s_0 \in$ 訓練データ **do**
    $V_{\max} \leftarrow 0, V^* \leftarrow 0, a_{\max} \leftarrow$NULL
    $a^* \leftarrow$ 訓練データで，$s_0$ において選ばれた着手
    **for all** $a_0 \in s_0$ の全ての合法手 **do**
      $V \leftarrow 0$
      **for** $k = 1$ から $N$ **do**
        $\pi_\theta$ を使って $s_0 \circ a_0$ からモンテカルロシミュレーション．一連の局面，着手と結果を $(s_0, a_0, \ldots, s_T, a_T; z)$ とする．
        $V \leftarrow V + \frac{z}{N}$
      **end for**
      **if** $V > V_{\max}$ ならば **then**
        $V_{\max} \leftarrow V, a_{\max} \leftarrow a_0$
      **end if**
      **if** $a_0 = a^*$ ならば **then**
        $V^* \leftarrow V$
      **end if**
    **end for**
    $h_{\mathcal{F}\max} \leftarrow 0, h^*_{\mathcal{F}} \leftarrow 0$
    **for** $j = 1$ から $N$ まで **do**
      $\pi_\theta$ を使って $s_0 \circ a_{\max}$ からモンテカルロシミュレーション．一連の局面，着手と結果を $(s_0, a_{\max}, \ldots, s_T, a_T; z)$ とする．
      $h_{\mathcal{F}\max} \leftarrow h_{\mathcal{F}\max} + \frac{z}{N} \sum_{t=1}^{T} \psi(s_t, a_t)$
    **end for**
    **for** $j = 1$ から $N$ まで **do**
      $\pi_\theta$ を使って $s_0 \circ a^*$ からモンテカルロシミュレーション．一連の着手と結果を $(s_0, a^*, \ldots, s_T, a_T; z)$ とする．
      $h^*_{\mathcal{F}} \leftarrow h^*_{\mathcal{F}} + \frac{z}{N} \sum_{t=1}^{T} \psi(s_t, a_t)$
    **end for**
    $\theta_i \leftarrow \theta_i - \alpha(V_{\max} - V^*)(h_{i\max} - h^*_i) \quad (i \in \mathcal{F})$
  **end for**
**end for**

## 5.2 実験と考察

今回の実験では，我々が作成した囲碁プログラム「MC_ark」を用いた．MC_ark はモンテカルロ木探索に基づいたプログラムであり，モンテカルロシミュレーション方策を学習するのに Berger [29] による最大エントロピー法を用いている．使用する特徴の種類は Coulom [5] の手法に基づいている．MC_ark は第 8 回 UEC 杯 [30] で 7 位に入賞した．

訓練データとテストデータには 4 路盤問題集である張 [31] の「黒猫のヨンロ」を使用した．この問題集のうち，中国ルールのコミ 0.0 という条件下で黒が勝ち，かつ初手の合法手が 2 つ以上あるものを選んだ．それらを 60 問の訓練データと 10 問のテストデータに分けた．

各問題の各合法手に対し 50 回ずつシミュレーションを行って勝率を計算し，別途 50 回ずつシミュレーションを行って目的関数の偏微分を計算した．学習の反復回数は 150 回までとした．すなわち，Algorithm 2 において $N = 50$ とし，LOOPLIMIT= 149（1 刻み）とした．これらの値は，実験の時間的制約を基に決めた．

論文 [32] での結果は図 5.1 の通りである．

各反復において，訓練データにおける 1 回の Closed テスト（CT）と，テストデータにおける 100 回の Open テスト（OT）を行った [1]．Open テストの結果は，擬似乱数のシードを変えて 100 回行い，平均を取ったものである．また，モンテカルロシミュレーション方策を，訓練データに Coulom [5] の手法を適用して調整した場合の結果も載せている（MM）．横軸が Simulation Adjusting の反復回数 (ログスケール)，左縦軸が正答数の軸，右縦軸が目的関数の軸である．灰色のラインが Closed テストの目的関数，黒色のラインが Open テストの正答数，破線が Coulom [5] の手法を用いた場合の Open テストの正答数である．目的関数の値は徐々に減少し，正答数は徐々に増加している．ただし，どちらも非常に不安定であり，もっと安定した結果が必要である．

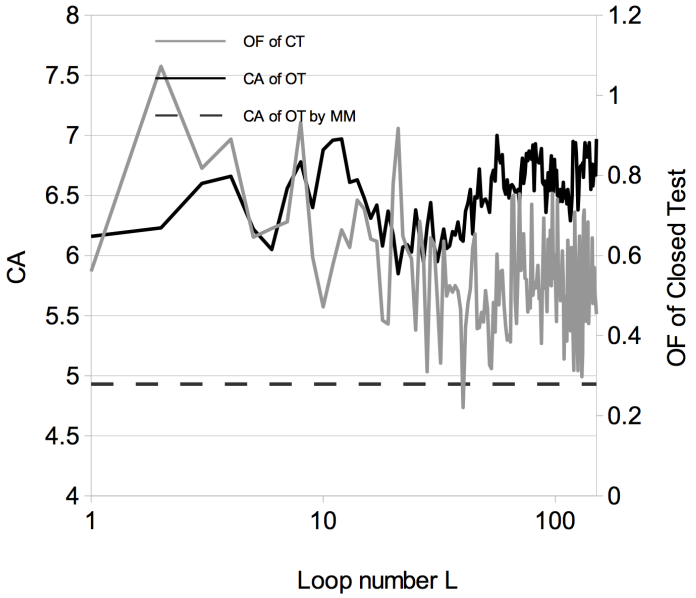



図 5.1: 目的関数と正答数の推移．横軸が Simulation Adjusting の反復回数 (ログスケール)，左縦軸が正答数の軸，右縦軸が目的関数の軸．灰色のラインが Closed テストの目的関数，黒色のラインが Open テストの正答数，破線が MM 法を用いた場合の Open テストの正答数．

## 5.3 問題点

論文 [32] を発表後，以下のことが分かった．

1. 目的関数には欠陥があった．目的関数を最小化するような $\theta$ によって，$V(s,a,\theta)$ は局面 $s$ における全ての着手 $a$ に対して一定の値を持つようになってしまう．これは望ましい結果ではないが，目的関数から得られる自然な結果である．

2. 論文 [32] では，訓練データの数は 60 であり，それに対して，特徴の数は数千であった．それ故，学習される重みはすぐにデータに過適合してしまった [2]．

[1] Closed テストとは「訓練データで学習し，同じ訓練データでテストする」ことである．Open テストとは「訓練データで学習し，別途用意したテストデータでテストする」ことである．

[2] 過適合というのは，訓練データに特化しすぎて，十分な汎化が出来ていない状態のことである．

3. 論文 [32] の実験ではシミュレーション回数が少なかった為，目的関数が安定して減少せず，分散も無視できなかった．結果として，正答数の分散も比較的大きくなってしまった．

# 第6章　正解以外の勝率を小さくする目的関数による実験

## 6.1　改良点

5.3 節の問題点を解決するために，以下のように改良を行なった．

1. 問題点 1 を避けるために，$V(s, a^{\max}, \theta)$ と $V(s, a^*, \theta)$ の差をシグモイド関数にかけて定数を加えたものを新たに目的関数として用いた．
2. 問題点 2 を解決するために，$3 \times 3$ パターンの特徴を取り除き，戦術的特徴のみ用いた．結果として，今回の実験では 28 個だけの重みを 286 問の囲碁の問題から学習することとなった．
3. 問題点 3 を解決し，より信頼できる結果を得るために，実験におけるモンテカルロシミュレーション回数を 50 から 1000 に増やした．その代わり，$\psi$ と $z$ の計算に別々のシミュレーションを用いていたのをまとめ，さらにプログラムを改良することで実行時間を短縮した．また，問題集を変えて，問題数を増やした．

## 6.2　アルゴリズム

$V$ を $\bar{V}$ で近似することにより，

$$a^{\max} \sim \bar{a}^{\max} = \operatorname*{argmax}_{a \neq a^*} \bar{V}(s, a, \theta),$$

として，目的関数は以下のように近似できる：

$$\min_{\theta} \sum_{(s,a^*) \in \mathcal{G}} \frac{1}{|\mathcal{G}|} \sigma\left(\bar{V}(s, \bar{a}^{\max}, \theta) - \bar{V}(s, a^*, \theta)\right).$$

アルゴリズムは Algorithm 3 のようになる．

## 6.3　実験と考察

今回の実験では，5 路盤囲碁の終盤問題で Simulation Adjusting のアルゴリズムをテストした．問題は福井 [33] のものを用いた．そのうち一問を図 6.1 に，その解答を図 6.2 に示す．

この書籍の問題は日本ルールに基づいて作られているが，中国ルールでも問題として成立する（黒番が勝ちになる手順がある）ようにコミを調整した．

元の問題と異なる点としては，今回の実験では解答に示されている一連の黒番の着手を全て問題として使用したことが挙げられる；黒番は，常に勝てる着手が存在するからである．このようにして，$\mathcal{G}$ を生成した．これは 286 組の問題と解答から成る．

**Algorithm 3** 正解以外の勝率を小さくするアルゴリズム

$\theta \leftarrow$ 初期化/*我々の実験では 0 または MM 法によって決定された値で初期化*/
**for** $L = 0$ から反復回数上限 LOOPLIMIT まで **do**
  $\mathcal{G} \leftarrow$ データを選んでシャッフルする /*我々の実験では，交差検定を行う*/
  **for all** $(s, a^*) \in \mathcal{G}$ **do**
    **for all** $a \in M(s)$ **do**
      $W_a \leftarrow 0$, $H_{\mathcal{F}a} \leftarrow 0$
      **for** $k = 1$ to $N$ **do**
        $\pi_\theta$ を使って $s \circ a$ からモンテカルロシミュレーション．一連の局面，着手と結果を $(s, a, \ldots, s_T, a_T; z)$ とする．
        $W_a \leftarrow W_a + z$
        $H_{\mathcal{F}a} \leftarrow H_{\mathcal{F}a} + z \sum_{t=1}^{T} \psi_{\mathcal{F}}(s_t, a_t)$
      **end for**
    **end for**
    **for all** $a \in M(s)$ **do**
      $V_a \leftarrow W_a/N$, $h_{\mathcal{F}a} \leftarrow H_{\mathcal{F}a}/N$
    **end for**
    $a^{\max} \leftarrow \mathrm{argmax}\{V_a : a \in M(s), a \neq a^*\}$
    $\theta_i \leftarrow \theta_i - \alpha\sigma'\left(V_{a^{\max}} - V_{a^*}\right)\left(h_{ia^{\max}} - h_{ia^*}\right) \quad (i \in \mathcal{F})$
  **end for**
**end for**

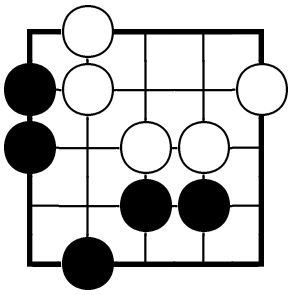


図 6.1: 5 路盤問題の例.

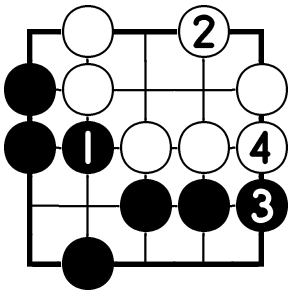



図 6.2: 6.1 の解答.

訓練データが偏って並んでいることによる過適合を防ぐため，Simulation Adjusting の各反復の始めに訓練データをシャッフルする．

本章における実験では，計算資源の制約の都合により，候補手を制限した．局面 $s$ における全合法手 $M(s)$ ではなく，その部分集合 $\bar{M}(s)$ を使用した．

$\bar{M}(s)$ は以下のように構築した． 始めに，MC_ark の，表 a（付録）にある特徴を用いた soft-max 関数に基づく元の着手評価システムを用いて，4 個の候補手を局面 $s$ について選び $\bar{M}(s)$ に加えていく．そして，その 4 個の候補の中に解答が含まれていなければ，それも $\bar{M}(s)$ に加える．

モンテカルロシミュレーションでは表 a に示されている特徴を用いた．改めて明記しておくが，モンテカルロシミュレーションでは $3 \times 3$ パターンを用いていない．何故なら，$3 \times 3$ パターンは多くの情報を含み過ぎ，5 路盤の問題では簡単に過適合を起こすからである．

今回の実験では，Simulation Adjusting での学習と，MM 法を用いた Coulom [5] の手法の学習において，上限 $\log_e 10 = 2.30$ と下限 $\log_e 0.01 = -4.61$ を $\theta_i$ $(i \in \mathcal{F})$ の値に設けた． このようにすることで，オーバーフローとアンダーフローを避けられるようにした．

ステップサイズ $\alpha$ は，まず $\bar{\alpha} = 1/(5 \times 10^5 \times (N + 1))$ を用いた．

$\theta$ の初期値として以下の二つを試した：

1. ZR: $\theta_i = 0$ $(\forall i \in \mathcal{F})$.
2. MM: $\theta$ を Coulom [5] の手法で同じ訓練データを用いて学習したもの.

ZR においては，各特徴が確率計算において同じ重みを持っており，これは自然な初期値である.

さらに，MM に関しては，Coulom [5] の手法が囲碁において標準的な機械学習手法なので採用した. MM の実験に関しては，我々の手法が正答率という点に関して，MM 法より良い解を見つけることができるのかどうかに関心がある.

LOOPLIMIT= 99 と設定したので，グラフ中の各線に関して，初期点を含め 100 個の点がプロットされている. 1 実験の結果のグラフにある線をプロットするために，16 コア（Xeon E5-2687W 3.10GHz×2）マシン 1 台を約半日動かす必要があった.

ここで,「Closed」は Closed テストを,「Open」は Open テストを,「cross」は交差検定を意味する.

データの偏りを避けるため，交差検定を以下のように実施した. 始めに，$\mathcal{G}$ を $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_{11}$ に分けた. ここで各 $\mathcal{G}_i$ $(i = 1, \ldots, 11)$ は 26 個のペアを含み，互いに共通要素を持たない. 各 $i \in \{1, \ldots, 11\}$ について順番に，$\mathcal{G}_i$ をテストデータ，残りを訓練データとする実験を行った. 言い換えると，260 個のペアを訓練データ，26 個のペアをテストデータとする実験を 11 回行った.

ZR の結果を図 6.3 と図 6.4 に示す.

図 6.3 は，交差検定による 11 回の実験の目的関数値の平均を示している.

図 6.4 は，交差検定による 11 回の実験の正答率の平均を示している.

最初の 30 反復では，Open テスト，Closed テスト共，目的関数値が徐々に減少している. さらに言えば，最初の 20 反復では，Open テスト，Closed テスト共，正答率が徐々に上昇している.

より小さなステップサイズ $0.5\bar{\alpha}$ と $0.1\bar{\alpha}$ でも実験を行った. 結果を図 6.5，図 6.6，図 6.7，そして図 6.8 に示す.

これらの結果において，目的関数値は $\alpha = \bar{\alpha}$ の場合に比べて大きく，正答率は $\alpha = \bar{\alpha}$ の場合に比べて悪くなっている.

MM の実験を $\alpha = \bar{\alpha}$ の条件下で行った. 結果を図 6.9 と図 6.10 に示す. 各グラフの横軸に平行に引かれている線は，MM 法で得られた値（0 反復目の値）を示している.

最初の 30 反復では目的関数は安定して減少したが，その後減少は止まり，増加し始めている. 正答率を見ると，Simulation Adjusting は，最初の数回の反復では MM 法を上回った. しかし，その後正答率は徐々に減少し，MM 法で得られた値より低いところまで下がってしまった.

5 章の実験よりも，安定した目的関数の減少と正答率の向上が見られた. しかし，MM の実験から分かるように，学習の反復を止めるタイミングをうまく決める必要もあり，まだ改良が必要である.

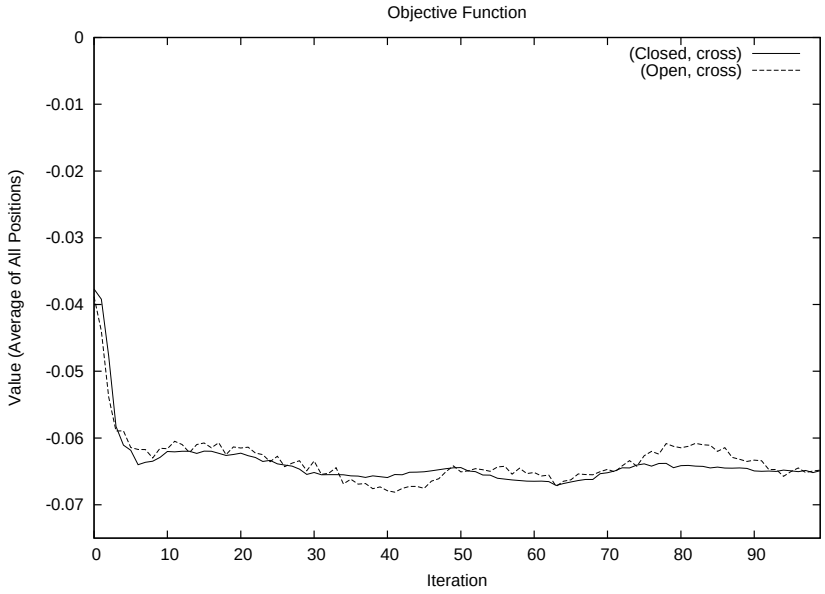



図 6.3: 目的関数の平均値の推移.（ZR, $\alpha = \bar{\alpha}$）

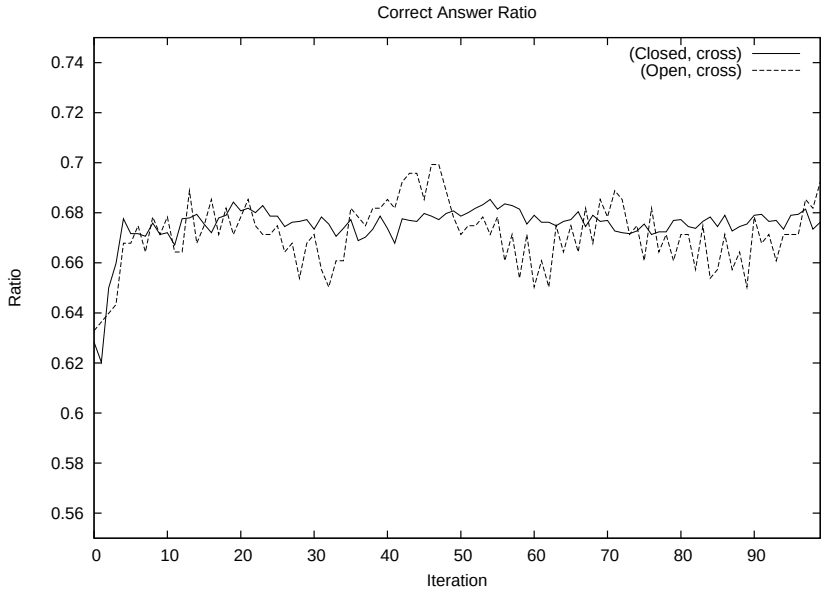



図 6.4: 正答率の平均値の推移.（ZR, $\alpha = \bar{\alpha}$）

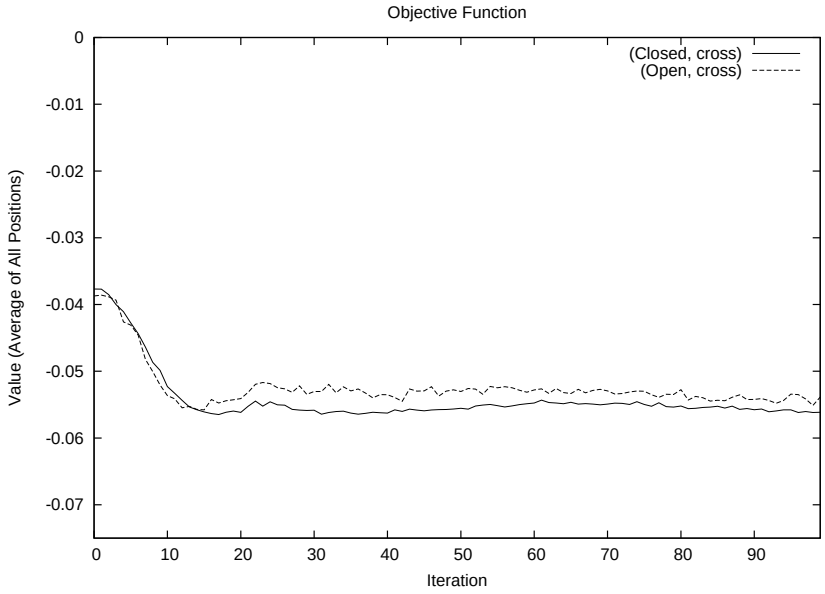



図 6.5: 目的関数の平均値の推移. （ZR, $\alpha = 0.5\bar{\alpha}$）

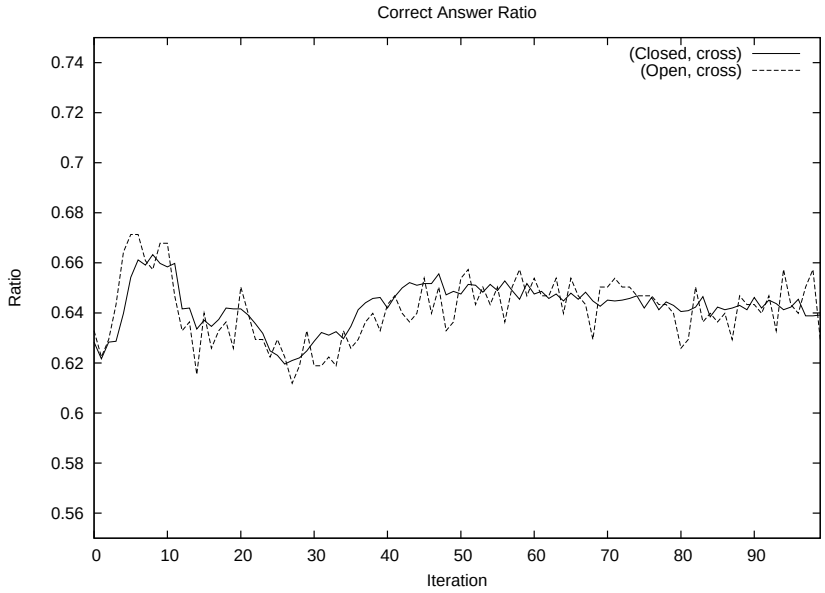



図 6.6: 正答率の平均値の推移. （ZR, $\alpha = 0.5\bar{\alpha}$）

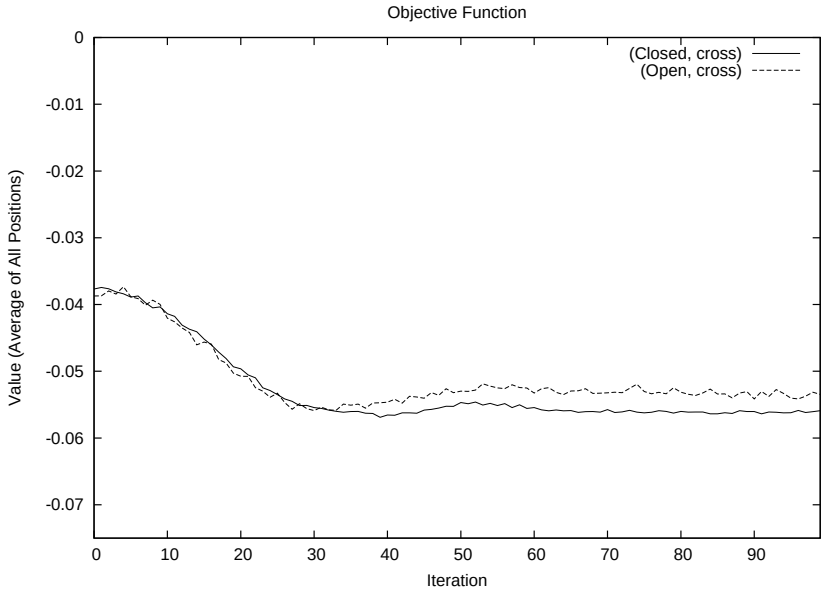



図 6.7: 目的関数の平均値の推移. （ZR, $\alpha = 0.1\bar{\alpha}$）

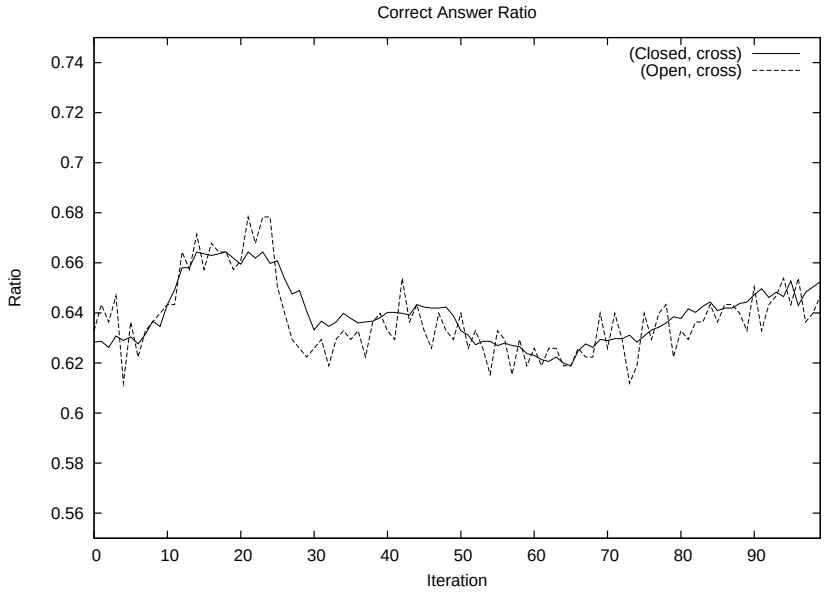



図 6.8: 正答率の平均値の推移. （ZR, $\alpha = 0.1\bar{\alpha}$）

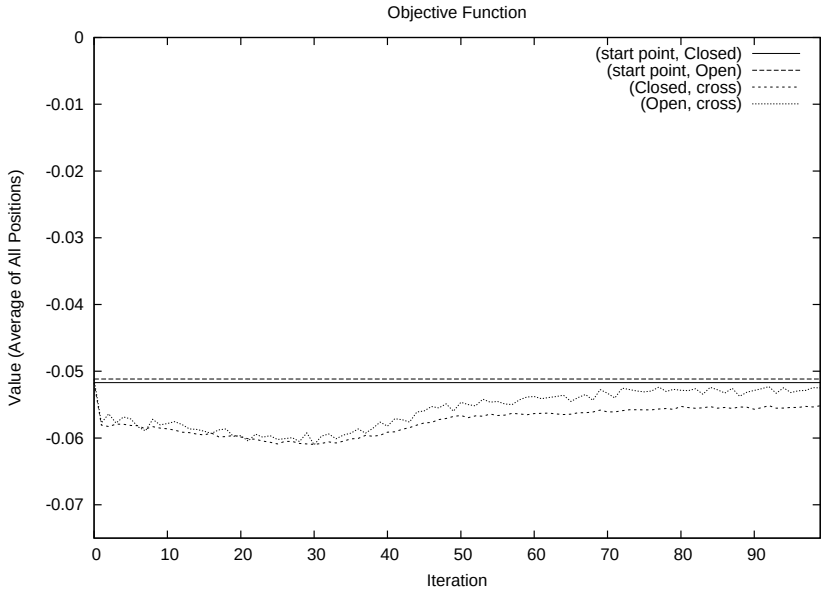



図 6.9: 目的関数の平均値の推移. （MM, $\alpha = 0.1\bar{\alpha}$）

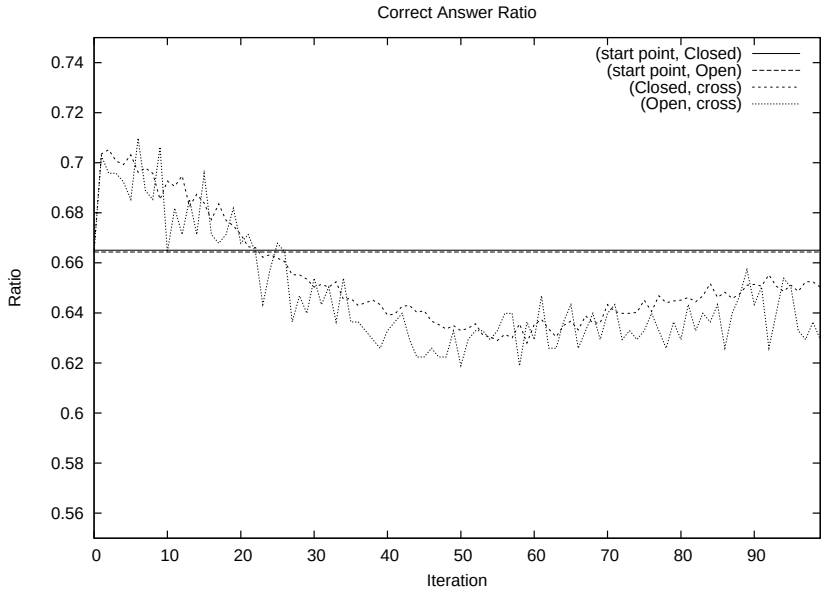



図 6.10: 正答率の平均値の推移. （MM, $\alpha = 0.1\bar{\alpha}$）

全ての実験において，目的関数，正答率とも，Closed テストと Open テストで近い値が出た．これにより，学習が正常に進んでいたと言える．

図 6.11 に，Simulation Adjusting により学習された特徴の値が，MM 法により学習されたものから離れていく様子を示す．この結果より，Simulation Adjusting は新たなモンテカルロシミュレーション方策を見つける可能性を持つことが言える．

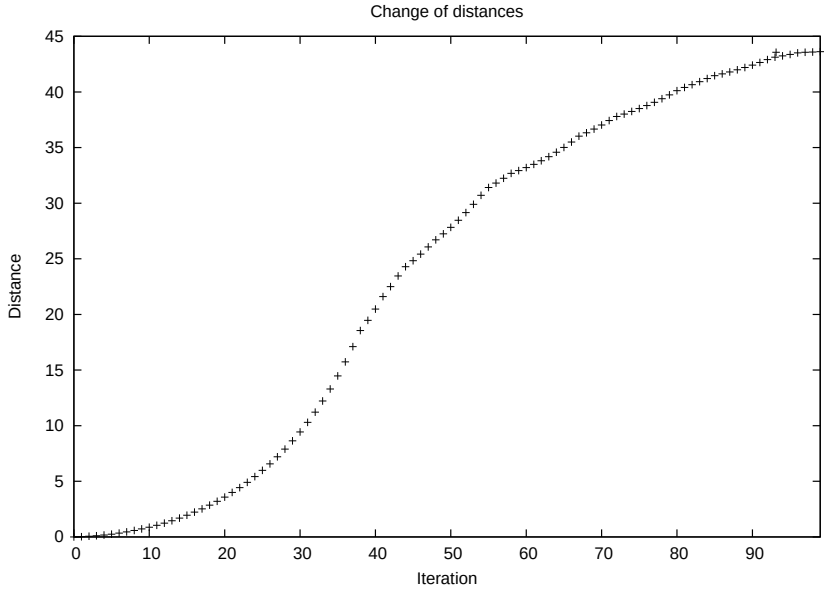



図 6.11: MM 法で得られた特徴ベクトルと Simulation Adjusting で得られた特徴ベクトルとの距離.

# 第7章　結論

第 I 部では Simulation Adjusting の自乗誤差の期待値を最小化する最適化問題 [32] と正解以外の勝率を小さくする最適化問題について述べた．

正解以外の勝率を小さくする最適化問題に基づく Simulation Adjusting の，論文 [32] との違いは，(i) シミュレーション結果の勝率が一定の値に収束してしまわないような，新たな目的関数を提案したこと，(ii) 過適合を避けるために，訓練データを増やし，特徴の数を減らした実験を設計したこと，(iii) 実験の精度を上げるための工夫をしたことである．特に，今回はモンテカルロシミュレーション回数を 20 倍に増やした．

今回の実験で，Simulation Adjusting の改良したバージョンは重みを安定して調整できることを確認した．すなわち，Closed テストと Open テストの両方で目的関数値が安定して減少した．

また，正答率という観点から，Simulation Adjusting の改良したバージョンはモンテカルロシミュレーションの性能を上げることも確認できた．

今後の課題として，以下が挙げられる．

まず，Simulation Adjusting を UCB 等と組み合わせ，候補手にかけるモンテカルロシミュレーション回数のバランスを上手く取れるようにする必要がある．今回の実験では，モンテカルロシミュレーションの結果ではなく，パターン等を用いて候補手を絞り込んでいる．

次に，Simulation Adjusting を使って学習したモンテカルロシミュレーション方策をモンテカルロ木探索に組み込んだときの性能を計る必要がある．今回の実験では，モンテカルロシミュレーション方策の性能はモンテカルロ木探索ではなく原始モンテカルロ法で計られている．

最後に，盤のサイズを大きくし訓練データを増やすために，計算時間を短縮する必要がある．そして，対局プログラムに組み込んだときに強さが向上しているか検証する必要がある．5 路盤で実験を行なっている現状では，強さの検証は難しい．何故なら，5 路盤で対局する場合，黒が初手を中央に打てば，相当弱いプログラムでも黒が勝ててしまうからである．強さの検証のためにも，盤のサイズを大きくした実験を行う必要がある．

# 付録：MC_ark で用いられている特徴

表 a: 特徴とその重み（「絞り込み」は， 候補手を絞り込むために使われた重みを意味し,「シミュレーション」は, Simulation Adjusting で得られた重みの一例を意味している.「-」は値が使われていないことを意味し,「$*$」はその特徴に分類される値が多すぎて記述できないことを意味している．また,「$*'$」も「$*$」と同様であるが，第 5 章の実験では重みが得られたことを意味している.）

| 特徴 | 重み | |
|---|---|---|
| | 絞り込み | シミュレーション |
| パターン （$3 \times 3$ 及びさらに大きなパターン） | $*$ | $*'$ |
| 盤端からの距離 | $1 : -0.341$<br>$2 : 0.334$<br>$3 : 0.054$ | $*'$<br>$*'$<br>$*'$ |
| 今の着手の盤端からの距離と<br>直前の着手の盤端からの距離のペア | $*$ | $*'$ |
| 直前の着手からのマンハッタン距離 | $*$ | $*'$ |
| 2 手前の着手からのマンハッタン距離 | $*$ | $*'$ |
| 直前の着手からの Common Fate Graph 距離 | $*$ | $*'$ |
| 2 手前の着手からの Common Fate Graph 距離 | $*$ | $*'$ |
| 直前の着手が（1,2）だったときの（2,1）<br>（対称形も考慮する） | 0.759 | 0.931 |
| 相手の連をとることでアタリから逃げる<br>（逃げる連の大きさで 2 種類） | 2.61<br>3.35 | 2.30<br>2.30 |
| 直前の着手で他の連に接続しようとした連を取る | 0.528 | 1.48 |
| 連を取る | $-0.0888$ | 2.30 |
| ノビでアタリから逃げる（大きな連）<br>（小さな連） | 3.40<br>2.19 | $-0.824$<br>2.30 |
| 眼を埋めることでアタリから逃げる | $-0.268$ | 2.30 |
| 自己アタリ（自分の連の大きさに基づいて 3 種類） | $-3.95, -2.97, -2.67$ | $-4.61, -1.94, -2.51$ |
| アタリ | 0.289 | 2.30 |
| 切りながらのアタリ | $-0.368$ | 2.30 |
| 空きダメ 2 個の自分の連の空きダメに打つ | 0.817 | 2.30 |
| 空きダメ 3 個の相手の連の空きダメに打つ | 0.0964 | 2.30 |
| 相手の連の空きダメを 1 から 2 に増やす場所に打つ | 0.0410 | $-1.92$ |
| 盤端に逃げようとした連を追いかける | 0.111 | 2.30 |
| 空きダメ 2 個の相手の連を追いかける | 0.111 | 0.682 |
| 空きダメ 2 個の連の根元をつなぐ | 0.0574 | 2.30 |
| 直前に大きさ 2 の連が取られたときの抜き跡 | 1.38 | 0.669 |
| 大きさ 3 から 6 の連が直前に取られたときの抜き跡の真ん中 | $-0.0259$ | 0.144 |
| 直前の着手に隣接する場所（斜めか上下左右かの 2 種類） | - | 2.30, 2.30 |
| 2 手前の着手に隣接する場所（斜めか上下左右かの 2 種類） | - | 2.30, 1.24 |
| 空きダメ 3 以下の攻め合い<br>（自分の連のダメを増やすか相手の連のダメを減らすかの 2 種類） | 0.838<br>0.239 | 2.30<br>2.30 |
| 直前の着手が（3,2）だったときの（2,2）（対称形も考慮） | - | $-4.61$ |

# 第II部

# 棋譜からの棋力推定
# ～畳み込みニューラルネットワークによるアプローチ～

# 第8章　はじめに

インターネットの普及した現代は，顔をつき合わせなければ囲碁を打てなかった時代と異なり，人はインターネットを介していつでも，どこでも，誰とでも囲碁を打てるようになった．このようなインターネットを介して囲碁を打てるシステムは「インターネット対局場」，あるいは「インターネット碁会所」と言われる．

インターネット碁会所のおかげで，特に強い人同士の対局が増加し，アマチュア高段の棋力は数十年前に比べて上がっていると言われている．

一方，囲碁は新たに始めるにはハードルの高いゲームとして知られている．街ではいくつも初心者教室を開催したり，級位者大会を開催したりしているが，なかなか競技者人口は増えていない [34]．初心者に長くゲームを楽しんでもらい，上達を促すのは日本棋院含め多くのプロ組織の設立目的の 1 つであるし，現在も喫緊の課題である．

それではインターネット碁会所が初心者にとって学習の場となるかというと，現実にはそうはなっていない．

現在のインターネット碁会所では，多くの場合，新規参入者は数局から数十局打たないと適正な段位やレーティングが付かない．本来の実力より過小あるいは過大な段位やレーティングでの対局が発生することは，インターネット碁会所の利用者にとって精神的な負担となりえる．実際，初期に負け続けてインターネット碁会所を使わなくなってしまう人は特に初心者，子供の中に少なからずいる．初心者にとって囲碁を一局打ち切るのはかなりのエネルギーを必要とすることもこの傾向に拍車をかけている．例えば 5 局打てば適正な棋力が推定されるとわかっていてもそこまで打てない人が多いのが実情である．このことは初心者がインターネット碁会所を使う上での障壁の一つになっている．

一方，人間は一局の棋譜から棋力を精度よく推定することができると言われている．特にプロ棋士を含む囲碁の上級者は，棋譜を一局見ればプレイヤーの棋力をかなりの精度で推定できると信じられている．このことに関する文献は特に見当たらないが，日本棋院や関西棋院などの多くのプロ組織において，プロ棋士がプロ棋士志望者と，一局打っただけでその志望者のプロ養成機関への合否を判断することが行われていることからも，これは事実に近いと推測される．

人間が一局の情報から棋力を推定できるならば，計算機にも同様に推定させたいと考えることは自然である．もし計算機が一局だけから棋力を推定できるようになれば，インターネット碁会所でも，あるいは街の碁会所でも，新規参入者に一局打たせて適切な棋力を判定し，以後の対局を適切なハンデで行なうことができる．

本研究 [35] では，この「一局の棋譜データから棋力を推定する」ということを目標とする．棋力という表現は曖昧な意味を持つがここでは囲碁クエストというインターネット碁会所におけるレーティング値を推定することを目指す．この目標に対し，CNN（Convolutional Neural Network，畳み込みニューラルネットワーク）を用いる手法を提案し，実験を行ない，評価する．

本論文では Moudřik ら [7] の手法を従来手法として実装し，提案手法と比較検討する．棋譜は，棚瀬 [36] のインターネット碁会所「囲碁クエスト」の 13 路盤の棋譜 [37] を用いる．囲碁クエストの特徴として以下が挙げられる．

- パソコンから参加することもできるが，スマートフォンやタブレットのアプリからの参加者が多い．
- 9 路盤と 13 路盤で対局することができる．
- 自動対局マッチングで対局相手が決まる．

表 1: 囲碁クエストの棋譜の統計情報.

| レーティング値 | 黒番のレーティングがこの範囲の棋譜 訓練/テスト | 白番のレーティングがこの範囲の棋譜 訓練/テスト |
|---|---|---|
| 0 − 1500 | 9330/1246 | 9555/1051 |
| 1501 − 2000 | 16939/2188 | 16926/2271 |
| 2001 − 2800 | 10713/671 | 10506/778 |

- チャットシステムが無い.
- 中国ルールで対局が行われる.
- AI で自動的に結果が判定される.
- 短い持ち時間で対局が行われる.
- コンピュータ AI が参加している.

わかりやすいシステムで，スピーディーな対局が可能なことから，初心者にも魅力あるインターネット碁会所となり得る．また，初心者だけでなくトップアマやプロプレイヤーも参加しており，棋力の層は非常に幅広い.

最初に参加したときはレーティング 1000 であり，その後 1 局ごとにレーティングが上下していく．レーティングの他に「レーティング X 以上で Y 連勝したら Z 段」のように勲章的な役割を持つ段位もつく．対局時点でのプレイヤーの棋力に応じたレーティング値が各棋譜に残っているため，今回の実験で使用するには適している．棋譜に記録されている着手以外の情報は，手順，黒番のプレイヤー名とレーティング値，白番のプレイヤー名とレーティング値， 結果である.

今回の学習では，黒番プレイヤーの棋力推定を行なうシステムと白番プレイヤーの棋力推定を行なうシステムを別々に用意した.

囲碁の棋譜から盤面を再現して入力データにする際には，小林 [28] の囲碁ソフト Ray と自作の補助スクリプトを用いた.

黒番プレイヤー推定用と白番プレイヤー推定用に別々に訓練データとテストデータを用意した.

例えば黒番用のデータは以下のように作成した．まず，棋譜 [37] の中から，置き石のある対局と，どちらかのプレイヤーが AI の対局を除く．残った棋譜全体を棋譜数が 9:1 に近くなるように訓練データとテストデータに分けるのであるが，訓練データとテストデータに同じプレイヤーが黒番として現れないように注意した.

白番でも，同じ棋譜全体から，訓練データとテストデータに同じプレイヤーが白番として現れないように注意して分割を行なった.

その結果，総棋譜データ量は 41087 局，黒番用訓練データ量は 36982 局，テストデータ量は 4105 局，白番用訓練データ量は 36987 局，テストデータ量は 4100 局となった．それらの棋譜のレーティング値の分布は表 1 の通りである.

以下では，まず関連研究について説明し，その後，提案手法と，比較対象となる Moudřik らの手法 [7] について述べる．次にレーティング値の推定に関する実験とその結果を報告し，従来手法との比較を述べる．それから，クラス分類問題に関して，レーティング値を推定してからクラス分類する手法とクラス分類自体を CNN によって学習する手法を比較する．最後に結論を述べる.

なお付録では，提案手法に関連の深いニューラルネットワークについて述べ，その中でも，提案手法で実際に用いている CNN 及び DCNN（Deep Convolutional Neural Network，階層の深い畳み込みニューラルネットワーク）について詳しく述べる．加えて，それらのこれまでの囲碁に対する応用についても述べる.

# 第9章　棋力推定に関する過去の研究

囲碁に限らず，棋譜から自動的にプレイヤーの段位やレーティング値を推定できれば，インターネット対局場などの運営がスムーズになる．よって，そのような推定をするシステムに関する研究は今まで様々に行われてきた．

チェスや将棋では，着手の優劣をグランドマスターレベルの思考プログラムを用いて解析し，プレイヤーの棋力を推定する手法が知られている [38, 39]．どちらも，基本的にはプログラムの指す手が正しいものとして，棋譜の局面でプログラムが指す手とプレイヤーの指す手の差をベースに棋力を推定している．チェスにおいて最近では棋力を推定する精度を競うコンペティションも開催され，トップグループは一局の棋譜からプレイヤーの Elo レーティングを 160 ほどの平均自乗誤差で見積もる [40]．

上記の研究は，コンピュータが人間のトップレベルと同等以上の棋力があることを利用している．囲碁でも，もし人間とトップレベルのコンピュータが手軽に使えるなら，同様の手法が使えるかもしれない．Silver ら [6] に基づく AlphaGo がトッププロに勝利するという成功を収めた現在，同様な手法の囲碁への適用は興味深い．しかし現時点では，AlphaGo 自身大変大きな計算資源を必要とするため，すぐにその方向の研究を行なうことは困難と思われる．

囲碁に関しても，いくつか棋力推定に関する論文が発表されている．Ghoneim ら [41] は囲碁において，GNU Go を用いてプレイヤーのレベルを「casual」，「intermediate」，「advanced」 に分類した．しかし，分類の結果の精度が明記されておらず，どの程度成功したのか論文からは判断できない．

Moudřik ら [7] では，あらかじめ手作業で設定した特徴に基づき囲碁の棋譜から特徴抽出を行ない，得られた特徴ベクトルを用いてニューラルネットワークにプレイヤーの棋力を学習させ，なるべく正確に棋力を推定しようとした．取り出された特徴は，戦術的特徴，パターン，局所的な応酬を始めた手番がその後続けて他の場所の応酬を始めたかどうか，ゲームの進行状況とそのとき打たれた着手の盤端からの距離，ゲームの進行状況とそのとき取り上げた石の数，及びゲームの結果である．

論文 [7] では，インターネット碁会所 KGS [22] の 20k から 6d のプレイヤーの棋力を，各プレイヤーあたり数十局の棋譜を用いて標準偏差 2.6 – 2.7k(d) 程度で見積もることに成功している．

なお，KGS ではプレイヤーのランクを下から 30k,29k,. . . ,1k,1d,2d,. . . ,9d と定めている．[42]．彼らの手法を用いて棋譜から棋力を推定するシステム [43] も Web 上で公開されている．

また，論文 [7] では，棋力推定の延長としてプレイヤーのプレイングスタイルの見積もりにも挑戦している．

本論文では，論文 [7] を基準として，提案手法を比較検討する．

# 第10章　提案手法と従来手法

## 10.1　提案手法

CNN の訓練で用いるデータは次の通りである．

- 局面情報： 黒（白）番時の局面を表すビット列
- レーティング値： 黒（白）番のプレイヤーのレーティング値

このうち局面情報については，$N$ 手目までの局面を用いることとする．$N$ は実験により異なる．$N$ 手に満たない棋譜は最終手以降 $N$ 手目までパスが続いたものとして扱う．

黒番のプレイヤーのレーティング値を予測するシステムには黒番の局面のみ，白番のプレイヤーのレーティング値を予測するシステムには白番の局面のみ入力する．したがって黒番の局面も白番の局面も一棋譜あたり $N/2$ の局面を入力することになる．

各局面については，図 10.1 のように

1. 既に置かれている黒石の位置が 1 でそれ以外が 0 の長さ $13 \times 13$ のビット列,
2. 既に置かれている白石の位置が 1 でそれ以外が 0 の長さ $13 \times 13$ のビット列,
3. 次の一手が 1 でそれ以外が 0 の長さ $13 \times 13$ のビット列

を準備する．これらが $N/2$ 盤面分あるので，入力層のビット数は $13 \times 13 \times (3N/2)$ となる．

今回用いた CNN の構造を図 10.2 に示す．全体は 5 層のニューラルネットワークになっており，$3 \times 3$ の畳み込みを 3 回行なったのち，全結合させた内積値を出力する．中間層 1, 2, 3 のチャネル数はそれぞれ 12, 8, 8 に固定した．これらは，訓練データのラベル数が約 37000 であることから，パラメータ数がそれを大きく超えないように決めた．後述の通り，$N = 50$ のときパラメータ数が 9961 個になる．

畳み込み層の活性化関数は正規化線形関数 [44] を用いる．

出力はレーティング値そのものを意味するものとし，平均自乗誤差を最小化するよう, 確率的勾配降下法により学習させた．

$N$ 手までの情報を用いる場合，調整するパラメータの総数はバイアス 28 個を含めて $162N + 1861$ 個であり，例えば $N = 50$ のときには 9961 個となる．

なお，囲碁を CNN で扱う場合に padding をしたり盤端を特別に扱ったりすることが有効であることもあるが，今回予備実験を行なった範囲では特に何もしない場合に比べて結果が良くはならなかったので，採用しなかった．

今回の実験ではライブラリとして Caffe を用いるが，Caffe の詳細については 13.3 節を参照のこと．一度の反復で処理する訓練データ量（Caffe [45] におけるメタパラメータ batch_size; 以下同様にカッコ内は Caffe のメタパラメータ名を表す.）は 250 とした．学習係数（base_lr）は $10^{-6}$ の固定値（ただしバイアスに関しては 2 倍），平滑化係数（momentum）はデフォルト通り 0 とした．重みの初期化（weight_filler）は Glorot and Bengio の手法 (xavier) [46]，バイアスの初期化（bias_filler）は全て 0 とした．それ以外のメタパラメータは，Caffe のデフォルト値を用いた．

以下，混同の恐れのない場合には，提案手法を *CNN* と略記することにする．

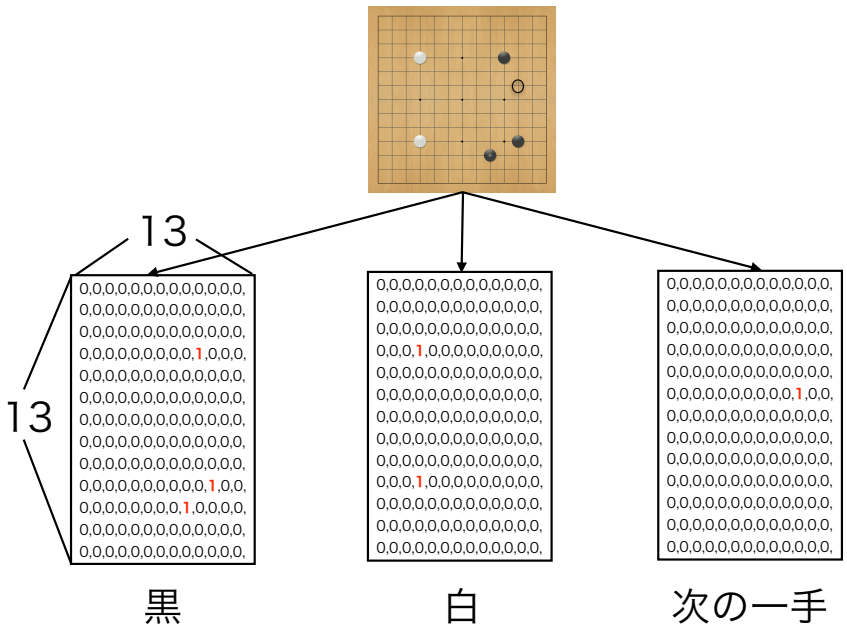



図 10.1: 局面のビット列化.

## 10.2 *MBN*(Moudřik et al. Based Network) – 従来手法の実装

Moudřik ら [7] は，あらかじめ手作業で設定した特徴に基づき，棋譜から特徴抽出を行ない，得られた特徴ベクトルとレーティング値も用いて 3 層ニューラルネットワークに学習させた．

比較のため，我々はこの論文 [7] に記述されている手法を実装した．ただし，13 路盤を対象とすること，論文 [7] のみでは不明な部分があること，使用するソフトが異なることなどから，いくつか異なる部分がある．この節ではそれを説明する．

以下簡単のため，論文 [7] を基に我々が実装した手法を *MBN*(Moudřik et al. Based Network) と呼ぶことにする．

*MBN* と原論文 [7] との主な相違点を表 1 ，表 2 に掲げる．○は論文 [7] と同じように再現できているが，△は完全には再現できていないことを表す．開発環境が異なるため，ベースとなる囲碁ソフト，学習アルゴリズムなどで違いがある．また [7] では 3 層ニューラルネットワークを用いているのに対し，*MBN* では 4 層としている．詳しくは実験の節で述べるが，4 層で *MBN* の学習が問題なく行われていることは確認している．

文献 [7] においては，20 個の Bagging を行なっているが，我々は行なっていない．実は我々も 20 個の Bagging を実装した版も作成しいくつか実験を行なったたが，ほとんど効果がなかったため，*MBN* では取り除いた．

論文 [7] で用いられている特徴と，*MBN* で用いられる特徴を表 2 に記す．*MBN* は 13 路盤を目的としているため，パラメータをいくつか変更している．例えば手数と取った石数のペアを特徴としている

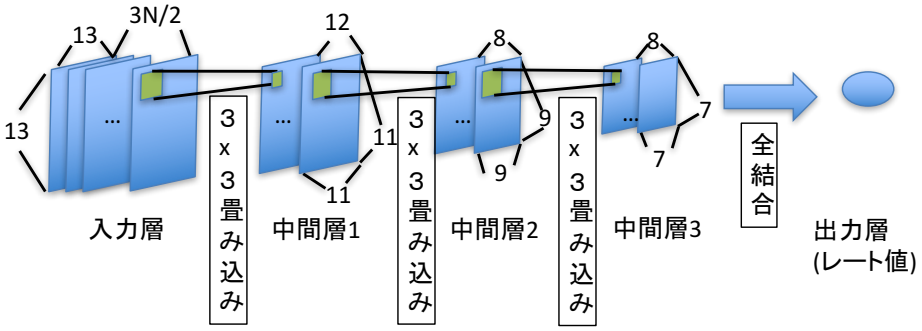



図 10.2: CNN の構造.

表 1: 文献 [7] と *MBN* の比較.

| | 文献 [7] | *MBN* |
|---|---|---|
| 盤の大きさ | 19 路 | 13 路 |
| 使用ソフト | pachi10.00 | Ray (2016) |
| NN の層数 | 3 層 | 4 層 |
| 隠れ層のノード数 | 20 | 20 (2 つとも) |
| 学習アルゴリズム | Resilient Backpropagation | Stochastic Gradient Descent |
| 重み初期化 | 不明 | Xavier |
| バイアス初期化 | 不明 | 定数 0 |
| 学習レーティング | 不明 | $10^{-6}$ |
| Bagging | 20 | 0 |

ところは，論文 [7] では手数を 1 から 60, 61 から 240, 241 以降の 3 段階にしているが，我々は 13 路盤という条件を考慮して 1 から 50, 51 から 100, 101 以降の 3 段階としている．

また，手数と盤端からの距離のペアでは，論文 [7] では手数を 1 から 10, 11 から 64, 65 から 240, 241 以降の 4 段階にしているが，我々はやはり 13 路盤であることを考慮して，手数は 10 手刻みとした．

$\omega$-local 先手というのは論文 [7] で提案されている特徴であるが，この値はやはり盤の大きさに依存して決めるべきものであるので，原論文では $\omega = 10$ のところ半分の $\omega = 5$ としている．

また，パターンについては，どのようなパターンを使用したのかは明示的には書かれていない．我々は Ray の MD4 と呼ばれるマンハッタン距離 4 以内のパターンを用いたので，ここでも少し相違している可能性がある．

各層で全結合しているため，*MBN* のパラメータ数は 188701 と，*CNN* に比べかなり多くなっている．

表 2: 文献 [7] で用いられている特徴と *MBN* で用いられる特徴.

| 文献 [7] で用いられている特徴 | *MBN* |
|---|---|
| アタリ | ○ |
| アタリからの逃げ | ○ |
| （手数，取った石数） | △ |
| 直前の手との隣接関係 | ○ |
| （手数，盤端からの距離） | △ |
| パターン | △ |
| $\omega$-local 先手 | △ |
| 勝敗 | △ |

# 第11章　レーティング値推定実験

まず，提案手法 *CNN* の学習結果を述べ，そのあと従来手法である *MBN* との比較を行なう．

## 11.1　*CNN* によるレーティング値推定

最初に，実験結果の乱数依存性を調べるために，乱数の種を変えた 1000 反復の学習を 10 セット行なった．

図 11.1 は *CNN* の 1000 反復の学習を 10 セット行なった際の，100 反復ごとにテストデータに対する平均自乗誤差の平方根（以下これを適合誤差と呼ぶ）をプロットしたものである．乱数の種を変えた 10 セットの学習について，その最大，最小，平均をプロットしている．この図では黒番で 50 手までの棋譜データを用いている．

なお，囲碁クエストで使われているレーティングシステムは，レーティングに 100 の差があるプレイヤー同士がハンデ無しで打つと，レーティングが高い方の勝率が 64% になるという基準で計算されている．よって，適合誤差が 100 以内に収まれば，かなり精度の高い推定が出来ていると言える．

10 セットの試行いずれも，初期に急激に適合誤差が減少し，その後 300 回を過ぎたあたりからは安定していることが見て取れる．図 11.1 の右上には，同じ図の 600 反復以上の部分を拡大したものを示す．試行による適合誤差の差はどの反復においても 10 以下であり，しかも学習が進むに従いだんだんと差が減少していることがわかる．白番でも同様の実験を行なったが，傾向は変わらなかった．

そこで，以降では特に断らない限り，一種類の乱数の種による 1 回の試行の結果を提示することとする．

CNN, DCNN では，長い時間学習させると良い結果が得られる場合がある．その可能性を探るため，黒番で $N = 50$ 手まで用いた場合に，30000 回の反復を行なった様子を図 11.2 に示す．横軸は対数目盛であることに注意する．また，図 11.2 の右上には，1000 反復以降の部分の拡大図を示す．

図 11.2 より，1000 反復を過ぎても適合誤差が下がっていることがわかる．これは $N = 50$ の白番でもほとんど同様の結果が得られている．計算資源の都合から，本研究では 30000 回反復し，その中で最も良いモデルを選択することを基本とすることにしたが，30000 回よりもっと増やすとより良い結果が得られる可能性がある．

次に，使用する局面の数を変えて，どのように学習結果が変化するかを見る．表 1 に，$N = 2, 26, 50$ と変えて 30000 反復させた場合の最良の適合誤差とそのときの反復を記す．なお，(B) は黒番の予測を，(W) は白番の予測を表している．

これをみると，$N = 50$ のときが最も適合誤差が小さいことがわかる．また，$N = 76$ および $N = 100$ の場合も学習させてみたが，30000 回反復させても適合誤差はほぼ初期値のままであった．$N = 76$ 以上のパラメータ数だと学習が困難になり，より多くの反復を必要としたり，あるいは最適化のパラメータを調整したりしなければならないようである．この点に関しては今後の課題とする．

また，*AVE* は「いつも訓練データの平均値を返す」レーティング値予測器の結果である．これと比べると，$N = 2$ においても，すなわち，1 手のみの情報からも，*AVE* よりはかなり良い適合誤差を得ていることがわかる．

1 手のみからこのような良い結果を得られるのは予想外であったので，今回の棋譜の初手に関して，棋力と位置の関係を調べた．

するとまず，レーティングが 2000 以上の上級者は，2000 未満の人たちより黒番の一手目を右上に打つ傾向が強いことがわかった．これは，伝統的に囲碁においては，黒番の初手を右上に打つのがマナーとされていることに起因すると思われる．また，左下や 2 線に初手を打つ人はほとんど初心者であった．

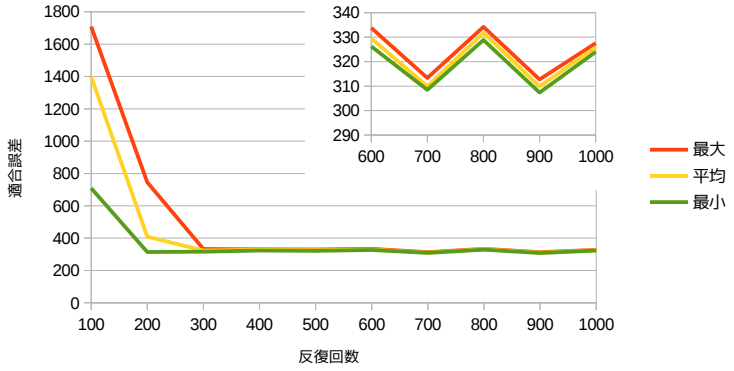



図 11.1: *CNN* の学習における適合誤差の減少の様子（10 試行の最大・平均・最小，縦軸：適合誤差，横軸：反復回数）.

このようなことから，1 手のみでも棋力に関してある程度の情報を持っていることが類推され，*CNN* はそれを抽出していると考えられる.

*MBN* の行，相関係数の列に関しては次の小節で説明する.

表 1: 適合誤差と相関係数の比較.

| 手法 | 反復 | 適合誤差 | 相関係数 |
|---|---|---|---|
| *MBN* (B) | 10200 | 310.492 | 0.2984 |
| *MBN* (W) | 8600 | 285.741 | 0.2891 |
| *CNN*, $N = 50$ (B) | 27600 | 249.318 | 0.6091 |
| *CNN*, $N = 50$ (W) | 29700 | 243.272 | 0.5818 |
| *CNN*, $N = 26$ (B) | 29600 | 262.326 | 0.5533 |
| *CNN*, $N = 26$ (W) | 25200 | 259.227 | 0.4936 |
| *CNN*, $N = 2$ (B) | 24700 | 283.022 | 0.4345 |
| *CNN*, $N = 2$ (W) | 18400 | 270.319 | 0.3964 |
| *AVE* (B) | – | 329 | – |
| *AVE* (W) | – | 302 | – |

## 11.2 *MBN* によるレーティング値推定との比較

図 11.3 は一度に入力するデータを一局分とした *MBN* による黒番の学習を 30000 反復まで行なった場合の適合誤差の減少の様子である.

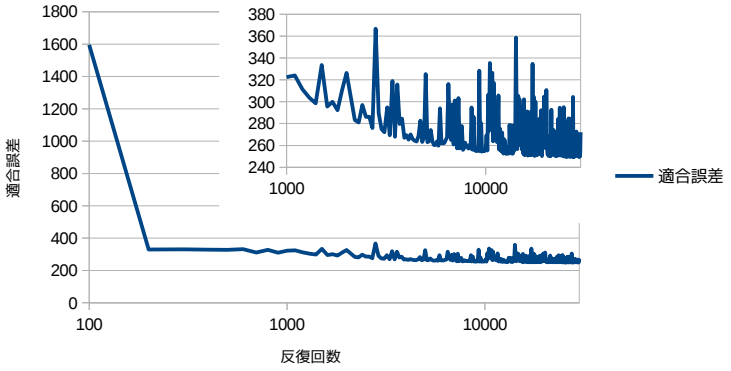



図 11.2: *CNN* の学習における適合誤差の減少の様子（30000 反復，縦軸：適合誤差，横軸：反復回数）.

まず注意して欲しいのは，100 反復時の値がすでにかなり良いことである．例えば 図 11.2 では 100 反復時の値が 1600 を超えているのに対し，図 11.3 では 340 以下から始まっている．それでも，1000 反復程度まで安定して適合誤差が減少しており，学習が成功していることがわかる．しかし，10000 反復を過ぎるとむしろ適合誤差は上昇しており，過適合の疑いがある．白番でも実験したが，30000 反復では過適合の傾向は同じであった．

なお，*MBN* のパラメータ数は 188701 である．

*MBN* の黒番 10200 反復目の適合誤差を表 1 に示す．白番 8600 反復も示されている．これは，30000 反復のうち最も適合誤差が小さかったものである．

$N = 2$ の場合の *CNN*, すなわち一手のみから予測する場合でさえ，黒番においても白番においても適合誤差は *MBN* より優れていることがわかる．

黒番の *MBN*（10200 反復目）と *CNN*（$N = 50$, 27600 反復目） に関して，出力と正答をプロットした散布図を 図 11.4 に示す．

この散布図で示されているデータの相関係数は 表 1 の一番右のカラムに載せる．数値からも，図 11.4 からも，*MBN* では出力と正答の間の相関が強くないことがわかる．

加えて，ウィルコクソンの符号順位検定を行なった．この検定は，値のペアが多数与えられた時に，ペア間の代表値にどの程度差があるかを調べる検定である．帰無仮説はペア間の代表値に差が無いことである．言い換えると，ペアの間に関係が無い時、帰無仮説は棄却される．結果を表 2 に示す．*MBN* で

表 2: ウィルコクソンの符号順位検定.

| | 正解ラベル | *MBN* | *CNN* |
|---|---|---|---|
| 正解ラベル | - | (p1) 差がある<br>(帰無仮説は棄却される) | (p2) 差が無い<br>(帰無仮説は棄却されない) |
| *MBN* | (p1) | - | (p3) 差がある<br>(帰無仮説は棄却される) |
| *CNN* | (p2) | (p3) | |

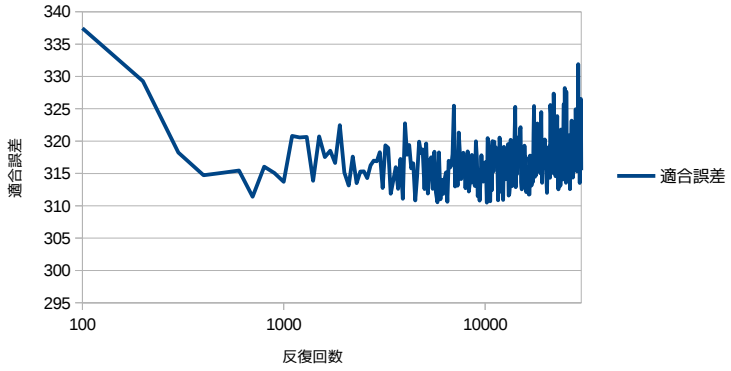



図 11.3: Moudřik ら [7] における適合誤差の減少の様子（縦軸：適合誤差，横軸：反復回数）.

表 3: 囲碁クエストの棋譜の統計情報.

| レーティング値 | 黒番のレーティングが<br>この範囲の棋譜<br>訓練/テスト |
|---|---|
| $0-1500$ | 3346/443 |
| $1501-2000$ | 5247/640 |
| $2001-2800$ | 1667/147 |

は帰無仮説は棄却されたが，*CNN* では帰無仮説は棄却されなかった.

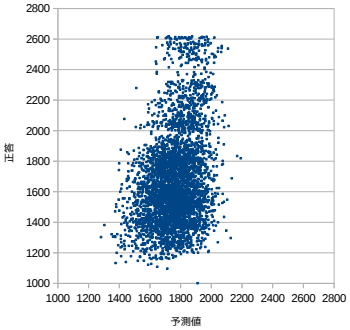
これらの実験から，一局の棋譜からの棋力推定の場合は，提案手法は従来手法よりもレーティング値の推定精度がすぐれていることが確かめられた.

## 11.3 1 プレイヤーあたり十局の棋譜を用いた場合の，*MBN* によるレーティング値推定との比較

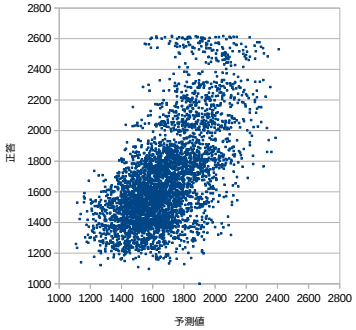
今回の実験の主たる目的は「一局の棋譜からの棋力推定」であるが，比較対象としている論文 [7] の手法は十局以上の棋譜からの棋力推定を想定している．そのため，提案手法について十局の棋譜からの棋力推定も試みた.

まず，訓練データ，テストデータとも，十局以上打っているプレイヤーの棋譜を，各プレイヤーについて十局ずつ取り出した．その内訳を表 3 に示す.

$N = 50$ の *CNN* に，学習時もテスト時も，データを先の実験と同様に与える．ただしテスト時のみ，同じプレイヤーの十局の棋譜を学習後の *CNN* に与えて出てきた 10 通りのレーティング値の平均と，棋




*MBN*




*CNN* ($N = 50$)

図 11.4: レーティング値の出力と正答の散布図（縦軸：正答，横軸：出力）.

譜に記録されている 10 通りのレーティング値の平均との適合誤差を計る．囲碁クエストでは一局打つたびにレーティング値が変動するため，同じプレイヤーの棋譜でもレーティング値にはばらつきがある．そのため，平均を取ることにする．

なお，入力としてまとめて十局分の棋譜を受け付けるように *CNN* の入力を増やすことも考えられる．しかし，$N$ の値を増やしていった時に目的関数が一向に減少せず学習が困難になったことから，*CNN* の入力を増やすことは得策ではないと考えた．

黒番のみ実験を行い，結果は図 11.5 のようになった．訓練データが大幅に減っているにも関わらず，テスト時に十局分の平均を取ることで，適合誤差はさらに減少している．

*MBN* の方は，入力はそのままに，十局から取り出した特徴をまとめて与えることができる．そのため，学習時もテスト時も，十局のデータから取り出した特徴をまとめて与えている．

図 11.6 は一度に入力するデータを十局分とした *MBN* による黒番の学習を 30000 反復まで行なった場合の適合誤差の様子である．1000 反復の時点で 250 以下になっているが，その後殆ど下がらない状態がしばらく続いた後，上昇している．訓練データとして与えられるデータの総量は，一度に入力するデータを一局分とした場合より減っていることから，過適合の疑いが強い．

一度に入力するデータを十局分とした *CNN* の黒番 28900 反復目の適合誤差と，*MBN* の黒番 1400 反復目の適合誤差を表 4 に示す．これも，30000 反復のうち最も適合誤差が小さかったものである．

$N = 50$，一局分の場合の *CNN* より小さい値が出ているが，$N = 50$，十局分の *CNN* よりは大きな値が出ている．適合誤差という観点からは，十局分を使う場合でも *CNN* の方が *MBN* より優れていることがわかる．

一度に入力するデータを十局分とした黒番の *MBN*（1400 反復目）と *CNN*（$N = 50$, 28900 反復目）に関して，出力と正答をプロットした散布図を 図 11.7 に示す．この散布図で示されているデータの相関係数も 表 1 の一番右のカラムに載せる．

*MBN* では一局分の場合に比べて劇的に相関係数が高くなっているが，*CNN* ではむしろ下がっている．*MBN* の方は複数局の特徴をまとめて扱うことに適しているためと考えられる．*MBN* の方で十局の特徴をバラバラに与えて平均を取り比較する等の実験を行えば，よりはっきりすると思われる．

なお，十局の棋譜を用いる場合についても，ウィルコクソンの符号順位検定を行なった．結果を表 5 に示す．*MBN*，*CNN* とも帰無仮説は棄却されなかった．

これらの実験から，一局の棋譜からの棋力推定の場合は，提案手法は従来手法よりもレーティング値の推定精度がすぐれていることが確かめられた．ただ，ウィルコクソンの符号順位検定の結果から，提

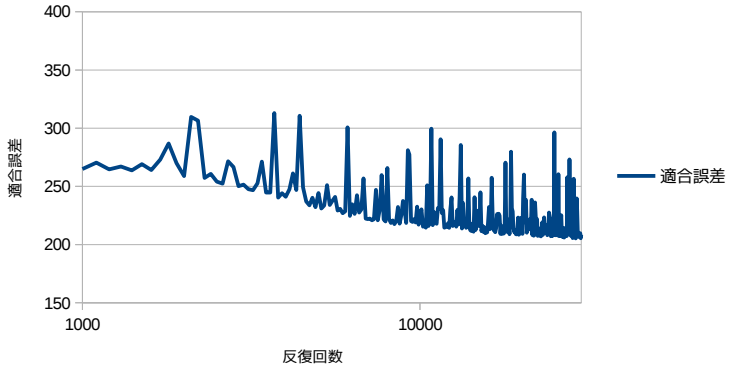



図 11.5: *CNN* の学習における適合誤差の減少の様子（30000 反復，十局分，縦軸：適合誤差，横軸：反復回数）.

表 4: 適合誤差と相関係数の比較 (1 プレイヤーあたり十局の棋譜を用いた場合).

| 手法 | 反復 | 適合誤差 | 相関係数 |
|---|---|---|---|
| *MBN* (B), 十局分 | 1400 | 242.195 | 0.7005 |
| *CNN*, $N = 50$ (B), 十局分 | 28900 | 205.251 | 0.4439 |

案手法に十局ずつ棋譜を与えた場合もある程度の推定は出来ていることが確かめられた.

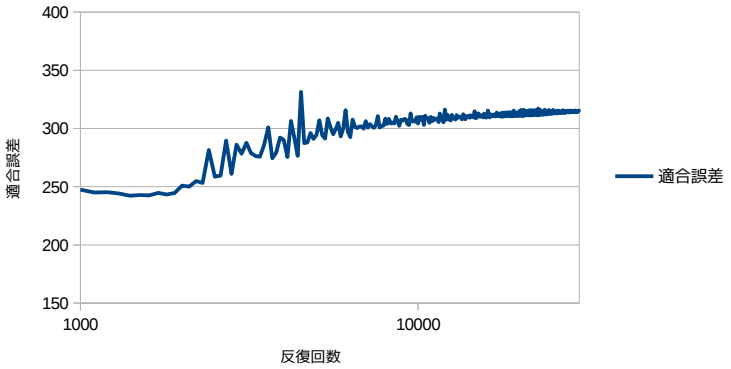
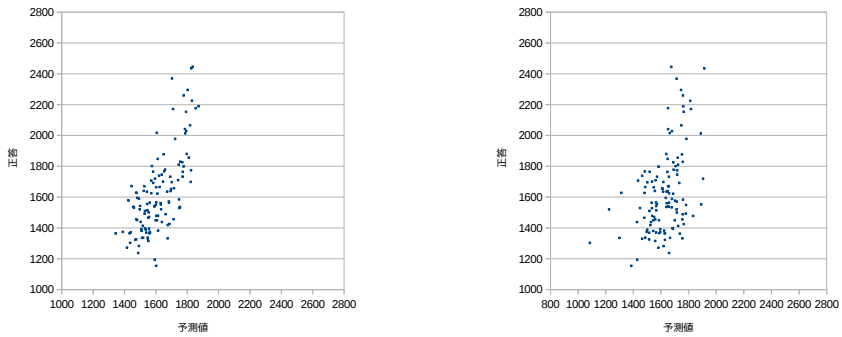



図 11.6: Moudřik ら [7] における適合誤差の減少の様子（30000 反復，十局分，縦軸：適合誤差，横軸：反復回数）.

表 5: ウィルコクソンの符号順位検定 (十局ずつ棋譜を用いる場合).

| | 正解ラベル | *MBN* | *CNN* |
|---|---|---|---|
| 正解ラベル | - | (p4) 差が無い<br>(帰無仮説は棄却されない) | (p5) 差が無い<br>(帰無仮説は棄却されない) |
| *MBN* | (p4) | - | (p6) 差が無い<br>(帰無仮説は棄却されない) |
| *CNN* | (p5) | (p6) | |




$MBN$ $CNN\,(N = 50)$

図 11.7: レーティング値の出力と正答の散布図（十局分，縦軸：正答，横軸：出力）.

# 第12章　クラス分類実験

新規参入者の棋力を判定して，適切な組み合わせを見出すという目的から鑑みると，正確なレーティング値を推定できなくてもある程度のクラス分けができれば実用上は問題無い場合も多い．

そこでこの章では，棋力のクラス分けに関する実験を行なう．クラスはレーティング 1500 未満を初級者（ラベル 0），レーティング 1500 以上 2000 未満を中級者（ラベル 1），2000 以上を上級者（ラベル 2）とする．棋譜におけるこれらのクラスにおけるプレイヤーの分布は表 1 の通りである．

ここでは 2 つの分類器を作成し，比較検討する．

1. 前章におけるレーティング値推定ネットワークにおいて，最良の適合誤差を示したものを用いて，出力されたレーティング値からクラス分けするもの．これを以下 *CNN-R* と呼ぶ．
2. 図 10.2 のネットワークの最終層を 3 つのクラスを表す 3 ノードに変更し学習させたもの．これを以下 *CNN-C* と呼ぶ．

*CNN-C* においては，最終層への活性化関数として soft-max 関数を用い，交差エントロピーを最小化した．

その他のほとんどのメタパラメータは *CNN-R* と同じであるが，予備実験を行った結果，学習のステップサイズ（base_lr）のみは 0.01 の固定値（ただしバイアスに関しては 2 倍）とした．

30000 回の反復ののち得られたネットワークを用いて実験した結果を表 1 に掲げる．なお，分類の実験では 15000 反復以降は正解率（後述）の変化が，57.3% を中央としておおよそ 1% 未満であり，図 11.2 の適合誤差のような値のバラつきは見られなかった．

ここで 0 は初級クラスのラベル，1 は中級クラスのラベル，2 は上級クラスのラベルを表す．

正解率，すなわち

$$\frac{\text{正解と出力が一致した棋譜数}}{\text{全棋譜数}}$$

を計算すると *CNN-R* は 59.1%, *CNN-C* は 57.1% であり，わずかながら *CNN-R* の方が高い．

また，上級を初級と誤ったり，初級を上級と誤ったりすることは実用上問題があると考えられるので，

$$\frac{\text{正解が 2 で出力が 0 の個数} + \text{正解が 0 で出力が 2 の個数}}{\text{出力が 0 の個数} + \text{出力が 2 の個数}}$$

を最悪分類割合と呼ぶことにして計算すると，*CNN-R* が 0.9%, *CNN-C* は 7.0% であり，*CNN-R* の方がかなり良い．

図 11.4 で見たように，レーティング値を学習する CNN は出力レーティング値と真の値に強い相関がある．よって，真の値と非常に離れたレーティング値を出力し，最悪分類になってしまう確率は低いと考えられる．

一方，*CNN-R* の出力は明らかに 1 が多く，偏っている．実際，正解が 0 の人が正しく 0 と判定される割合は *CNN-R* が 38.5%, *CNN-C* が 42.3%であり，正解が 2 の人が正しく 2 と判定される割合は *CNN-R* が 34.3%, *CNN-C* が 61.5% とかなりの開きがある．もともとの訓練データにおいて中級者の棋譜は初級者および上級者の 2 倍程度あり（表 1 参照），*CNN-R* では特にそれに引きずられる傾向が顕著である．数の少ない初級者，上級者をきちんと見分けたいならば，最初からクラス分類を目的とした CNN を構成して学習させた方が性能が良い可能性がある．

なお，ここでは黒番のみによる実験を報告したが，白番であっても個々の数値は異なるものの，傾向には差異が見られなかった．

表 1: クラス分類実験の結果（黒番）.

| *CNN-R* | | | | | | |
|---|---|---|---|---|---|---|
| | 出力が 0 | | 出力が 1 | | 出力が 2 | |
| 正解が 0 | 480 | (11.7%) | 759 | (18.5%) | 7 | (0.2%) |
| 正解が 1 | 365 | (8.9%) | 1718 | (41.9%) | 105 | (2.6%) |
| 正解が 2 | 5 | (0.1%) | 436 | (10.6%) | 230 | (5.6%) |
| 合計 | 850 | | 2913 | | 342 | |

| *CNN-C* | | | | | | |
|---|---|---|---|---|---|---|
| | 出力が 0 | | 出力が 1 | | 出力が 2 | |
| 正解が 0 | 527 | (12.8%) | 612 | (14.9%) | 107 | (2.6%) |
| 正解が 1 | 396 | (9.6%) | 1402 | (34.2%) | 390 | (9.5%) |
| 正解が 2 | 21 | (0.5%) | 237 | (5.8%) | 413 | (10.1%) |
| 合計 | 944 | | 2251 | | 910 | |

# 第13章　結論

CNN を用いて一局の棋譜からプレイヤーの棋力を推定する手法を提案し，Moudřik らの手法 [7] と比較実験を行なった．

その結果，一局の情報からレーティング値を推定することに限れば，提案手法は Moudřik らの手法 [7] よりも良い精度を持つことが確かめられた．

また，十局分を与える場合でも，提案手法は [7] よりも小さい適合誤差を示した．相関係数については，提案手法においては十局分を与えると悪化してしまい，[7] の方が良い値を示した．当初の目標である一局分の情報からの棋力推定には，CNN が適していると言える．

また，レーティング値推定 CNN を用いて，出力の推定値を 3 分類する手法と直接 3 分類する CNN を構成する手法を比較した．最悪分類割合に関してはレーティング値推定 CNN を用いた方が良い結果を得た．

一方，この手法は「訓練データ量の小さいクラス（上級および初級)」の判定に関して問題があり，直接 3 分類することを学習した CNN の方がこれらのデータに関しては正答率が高かった．これらの分類器は，目的によって使い分ける必要がある．

今後の研究課題として，以下のことが考えられる．

大きな $N$ に対する学習手法の確立
: 現状では $N = 76$ の場合ですでに学習が困難になっている．$N$ が 50 手以下であるということは，小ヨセが無視されているということである．正解がわかる小ヨセで明らかなミスがあれば棋力推定の有力な手がかりとなるが，現状では小ヨセを対象とできていない．これを克服するには，メタパラメータの慎重な調整，汎化性能のより高い特徴を利用するなどのモデルの変更，訓練データ量を増加させるさまざまヒューリスティックの適用などを用いて学習が困難となっている理由をつきとめなければならない．大きな $N$ での学習を可能にし，その上で $N$ を 100 手以上まで上げた場合に有意に適合誤差が下がるのかどうかを調べることは重要な今後の課題である．

相手プレイヤーの棋力分布の偏りの改善
: 囲碁クエストでは，近い棋力のプレイヤー同士を組合わせる傾向があるので，訓練データの 2 プレイヤー間には，レーティングの相関があると思われる．現状のモデルでも，相手が過去に着手した石の位置が入力データとして与えられており，相手プレイヤーの情報は間接的に入力されていると考えられるので，このレーティングの相関が学習に影響を与えている可能性は考えられる．この影響は「インターネット碁会所の新規参入者に対する一局からのレーティング値の推定」という当初目的に照らすと適切ではない．この影響を減らすためには，2 プレイヤーのレーティングがちらばるような訓練データがあれば良い．しかし現状では，それでは十分な数の訓練データが得られない．これに対し，サンプル法の改善などを検討していく必要がある．

棋力推定に対する他のアプローチの開発
: Maddison ら [47] においては，棋力を入力として与え，着手を予測することに成功している．これを応用し，棋力別に手を予測し，1 手ごとの誤差を計算して累積の誤差などを用いて棋力を推定するというアプローチも考えられる．このようなアプローチでどの程度本研究の結果を上回ることができるのか，あるいはできないのか，調査すべき研究課題である．

CNN が囲碁に有効であることは，AlphaGo を見ても明らかであるし，本研究でもその一端を示すことが出来た．

しかし，CNN を囲碁に応用する研究にはまだ未開拓な部分も多い．確かに AlphaGo は CNN を利用することで「総合的な」棋力という点ではトッププロを上回ってしまった．だが，囲碁 AI「Aya」の開発者が AlphaGo の真似をして Policy-Network と Value-Network を 13 路盤で実装してみたところ，「3 手と 4 手の攻め合いが分かっていない，中手の攻め合いも分かっていない，両コウも分かっていない，コウ争いは下手，など今までの Aya と同じ弱点を持つにもかかわらず，ディープラーニングの追加だけでプロと戦えるレベルに到達した.」という報告もあった [48]．果たして CNN を使ってこれらの弱点を乗り越えることもできるのかどうかは今後の研究課題である．CNN だけでは上記のような弱点を克服することは不可能で，本研究の「Simulation Adjusting」などの手法でモンテカルロシミュレーションを改良していくことが必要かもしれない．

本研究が今後の囲碁 AI 及びその周辺分野の研究の一助になれば幸いである．

# 付録：囲碁のルールについて

この付録では，囲碁のルールについて説明する．

- 囲碁は黒番と白番の二人でプレイするゲームであり，図 a のように黒から始めて黒白交互に交点に石を置いていく．この石を置くことを「打つ」と言う．囲碁は二人で交互に石を打っていくゲームである．
- 盤の大きさは，公式戦では $19 \times 19$（19 路盤）が最も多く用いられる．他に $13 \times 13$（13 路盤）や $9 \times 9$（9 路盤）もよく用いられる．

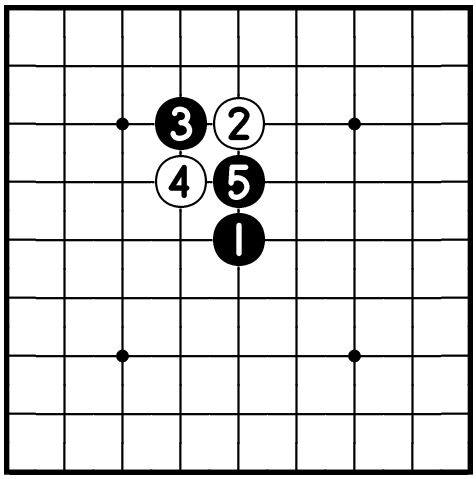



図 a: $9 \times 9$ の盤での対局の途中経過．

- 相手の石を上下左右を自分の石で囲むと取れる．図 b の左の局面で A の位置に打つと図 b の右の局面のように，図 c の左の局面で A の位置に打つと図 c の右の局面のように白石を打ち上げることができる．

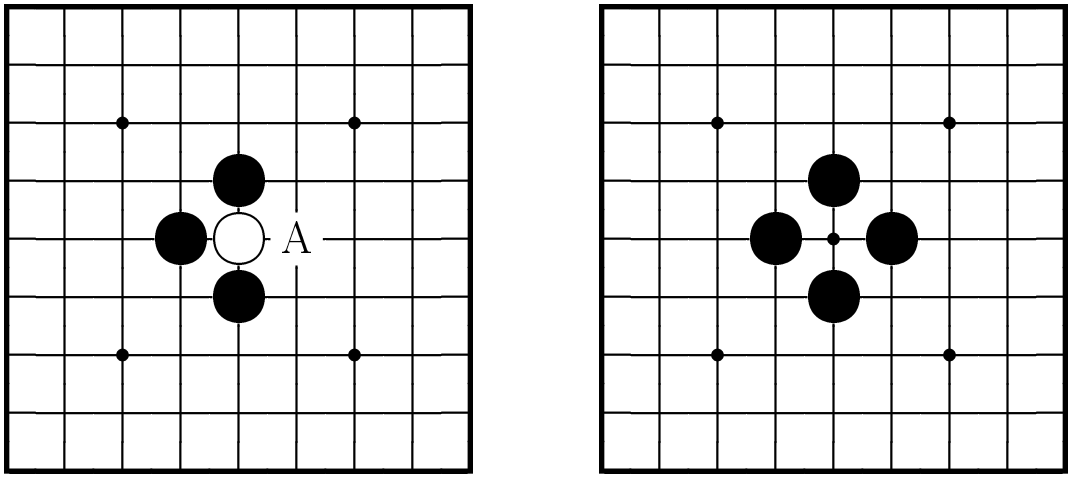



図 b: 石を打ち上げる例その 1．右図は打ち上げた後の局面．

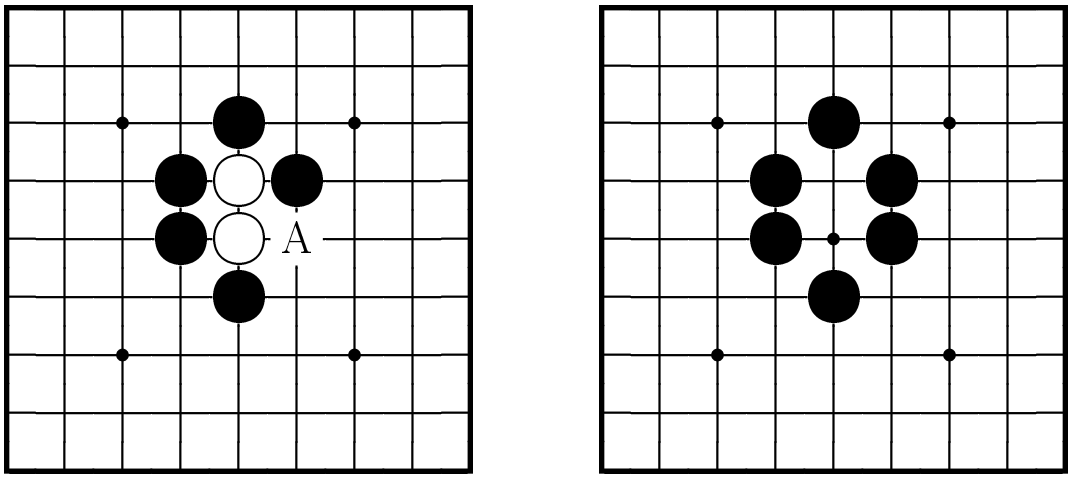



図 c: 石を打ち上げる例その 2．右図は打ち上げた後の局面．

- 囲碁では，そこに打つと打った瞬間にその打った石が打ち上げられる状態になってしまう交点には打てない．この交点を着手禁止点という．例えば図 d の A は着手禁止点である．ただし，一見囲まれているように見えても周りの相手の石を打ち上げることができる場合は打てる．例えば図 e である．

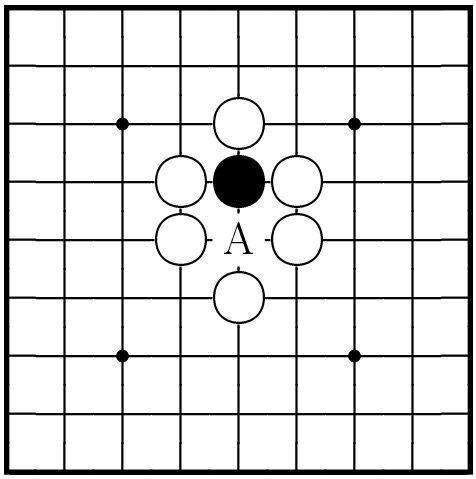



図 d: 着手禁止点の例．A に黒は打てない．

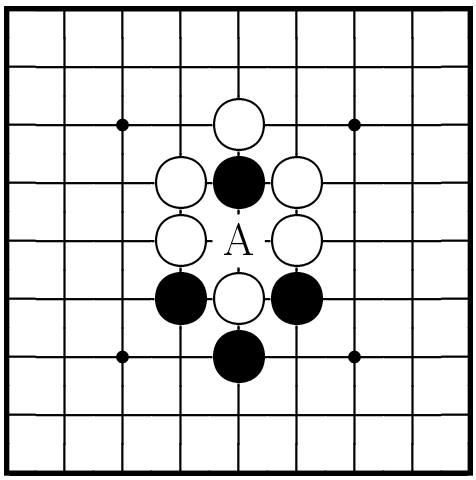



図 e: 着手禁止点ではない例．A に黒は打てる．

- 自殺手以外の着手禁止点として図 f のような「コウ」に関するものがある．「黒が 1 に打つ」→「白が 2 に打つ」→「黒が 1 に打つ」→... と互いに打ち合うと同型反復でゲームが終わらないため，黒が 1 に打った後は，一度他の場所に打ってからでないと白は 2 の箇所に打つことが出来ない．

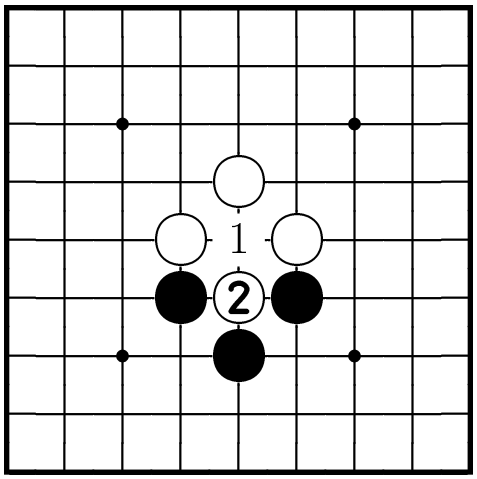



図 f: コウの例.

- 囲碁は黒番が有利であることが知られているため，力量が互角の相手と打つときは「コミ」というものを導入する．黒は，コミより大きな差をつけて勝たないと勝ったことにならない．コミとしては 5.5 から 7.5 が使われることが多い．引き分けがあっても良い場合は整数のコミを，そうでない場合は小数のコミを用いる．
- 最終的な勝敗は，囲った陣地と置いた石の数の合計が多い方が勝ちとなる．例えば図 g では，黒のスコアが $4 \times 9 + 7 = 43$，白のスコアが $4 \times 9 + 2 = 38$ で，コミが 6.5 なら白の勝ちである．なお，これは中国ルールであり，日本ルールの場合は囲った陣地から取られた石の数を引いたスコアで競う．

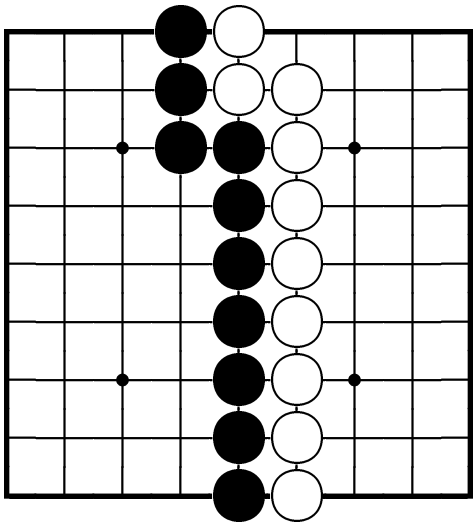


図 g: この状態だと，黒のスコアが $4 \times 9 + 7 = 43$，白のスコアが $4 \times 9 + 2 = 38$ で，コミ 6.5 目だと白が勝っている．

# 付録：ニューラルネットワークについて

## 13.1　ニューラルネットワーク

ニューラルネットワークは，人間の脳の神経網を模して [44]，ニューロンから構成されたネットワークである．2000 年から 2010 年頃までは，層と層を全結合した 3 層のネットワークが主流であった．

ニューラルネットワークの定義を述べる前に，まず構成要素であるニューロンの定義を述べる．

**定義 13.1.1** (ニューロン)．ここではニューロンを，以下のような式で表される多入力値，単一出力値の機能素子と定義する：

$$z = f(\sum_{i=1}^{n} w_i x_i - h).$$

ただし，$x_i$ $(i = 1, 2, \ldots, n)$ が入力値，$z$ が出力値である．入力値 $x_i$ と結合荷重 $w_i$ の積の総和から閾値 $h$ を引いた値を，関数 $f$ で処理することで出力値が決定される．

なお，常に入力値が 1 である $n+1$ 番目の入力があると考えて $x_{n+1} = 1, w_{n+1} = -h$ とすることで，

$$z = f(\sum_{i=1}^{n+1} w_i x_i)$$

と表すことができる．$f$ としては微分可能な関数が用いられることが多く，代表的なものはシグモイド関数である．

**定義 13.1.2** (ニューラルネットワーク)．ニューラルネットワークとは，多数のニューロンによって構成されたネットワークのことである．

図 h は，その中でも階層型のニューラルネットワークのイメージである．ノードはニューロンを，線は出力と入力の接続を表している．層とは，入力からの距離（ニューロンを通った回数）が等しい，互いに結合の無いニューロンの集合を表す．このように，単純な関数を何層にもつなげることで，複雑な処理ができるようになる．

重みは通常，教師あり学習によって決める．階層型ニューラルネットワークの場合，まず訓練データの入力をニューラルネットワークに与えてみて，階層型ニューラルネットワークの出力と本来あるべき出力（訓練データのラベル）との誤差を見る．そこから「BP（Back Propagation, 誤差逆伝搬法）」という手法で，出力層に近い層から順に関数を調整していく．これを繰り返すことで，ニューラルネットワーク全体が，入力に対して本来あるべき出力を返せるように調整されていく．そして，最終的な精度確認をテストデータで行う．

各層が $M$ 個のニューロンから成り，層の数が $N$ の階層型ニューラルネットワークの学習を例に取り，BP について解説する．まず，第 $n-1$ 層の第 $j$ ノードの出力を $x_j^{n-1}$, 第 $n-1$ 層の第 $j$ ノードの出力と第 $n$ 層の第 $i$ ノードの入力の間に重み $w_{ji}^{n-1}$ がかかるとする．このとき，第 $n$ 層の第 $i$ ノードの出力 $x_i^n$ は以下のようになる：

$$x_i^n = f(\sum_{j=1}^{M} w_{ji}^{n-1} x_j^{n-1}).$$

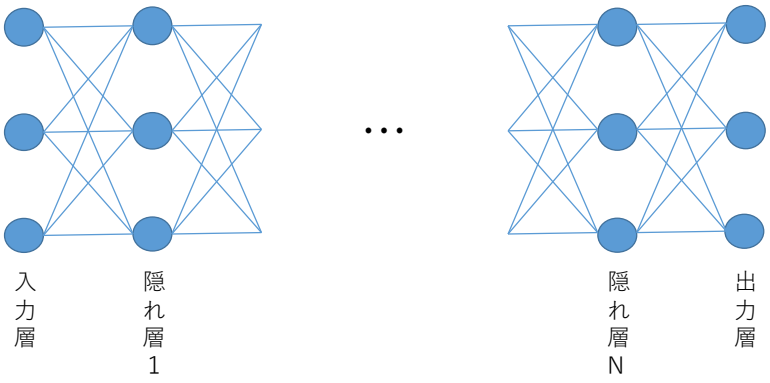



図 h: 階層型のニューラルネットワークのイメージ．ノードはニューロン，線は出力と入力の接続．層とは，入力からの距離（ニューロンを通った回数）が等しい，互いに結合の無いニューロンの集合．

このとき，出力層 (第 $N$ 層とする) の第 $i$ ノードの本来あるべき出力を $d_i$ とすると，出力層の第 $i$ ノードから戻される学習信号は，

$$\delta_i^N = (d_i - x_i^N) f'(\sum_{j=1}^{M} w_{ji}^{N-1} x_j^{N-1})$$

となる．$n$ 層のノード $i$ から $n-1$ 層のユニット $j$ へ向かって戻される学習信号 $\delta_i^n$ は，

$$\delta_i^n = \sum_{k=1}^{M} \delta_k^{n+1} w_{ik}^n f'(\sum_{j=1}^{M} w_{ji}^{n-1} x_j^{n-1})$$

となる．このとき，重みの修正量 $\Delta w_{ji}^n$ は，

$$\Delta w_{ji}^n = \tau \delta_i^n x_j^{n-1}$$

$\tau$ は実験で決める定数である．

## 13.2 CNN

CNN（Convolutional Neural Network, 畳み込みニューラルネットワーク）は，階層型ニューラルネットワークの一種である．Krizhevsky ら [49] が DCNN（Deep CNN, 階層の深い畳み込みニューラルネットワーク）を用いたシステムで 2012 年に画像認識のコンペティション ILSVRC で 2 位に精度差 10% 以上の大差をつけて優勝する（9 層の DCNN を使用）という出来事もあり，近年注目を浴びている．

CNN の場合は，各層を 2 次元平面の集合と見なし，入力値 $x$ と出力値 $z$ の関係は，

$$\begin{aligned} z_{ijk} &= f(\sum_{l=1}^{A} \sum_{p=1}^{C} \sum_{q=1}^{D} w_{pq} x_{i+p,j+q,l}) \\ & (1 <= i <= E-C,\ 1 <= j <= F-D,\ 1 <= k <= B,\ 1 <= l <= A) \end{aligned}$$

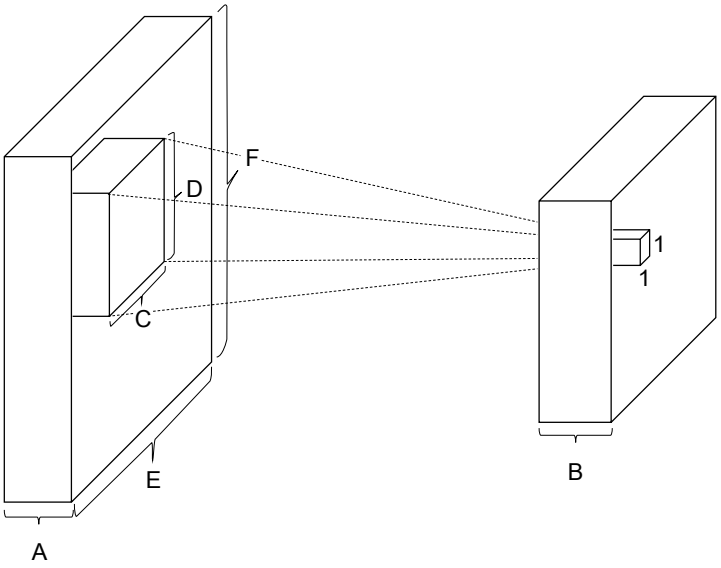



図 i: CNN のイメージ．前の層の $C \times D$ が $1 \times 1$ に圧縮．それが $A \times B$ 通り存在．

のようになる．図 i のように，前の層の $C \times D$ のノードと次の層の一つのノードを結合する．ただし，CNN の場合は $f$ として $f(x) = \max(x, 0)$ を用いることが多い．

CNN は，文字認識や顔認識など画像認識で成功を収めてきた [50, 51]．中間層のノードは，直前の層の出力の中に存在する特徴的なパターンを検出する．入力に近い層ではエッジなどの局所的な特徴が抽出され，後段にゆくほど，前段の特徴が複数組み合わされた，より大域的な特徴が抽出されるようになると言われている

## 13.3 Caffe

CNN を扱えるライブラリとして近年 Jia ら [45] の Caffe が広く使用されている．Caffe は，テキストによる設定ファイルを使って比較的容易にネットワークの構造や学習のパラメータを指定することができる．Caffe は基本的に Python から使うことを想定しているが，学習後のモデルを C++から使うこともできる．

Caffe を使用した囲碁 AI として Niekerk and Schmicker [52] の oakfoam などがある．

今回の我々の実験でも Caffe（git ハッシュ値 7953918）を用いる．Caffe に必要な入力は，ネットワークの構造を記述したファイル，メタパラメータを記述したファイル，及びデータファイルである．

また GPU 使用もフラグを一つ切り替えるだけでできる．CNN は階層が深くなってくると計算量が莫大になるため，GPU を使用することは計算時間の節約に大いに役立つ．今回は GPU として NVIDIA の Tesla K40 [53] を使用した．

## 13.4 DCNN の囲碁における研究

Clark and Storkey [54] では，対称性を考慮した DCNN を使うことで，囲碁に関する知識に基づいた特徴抽出を殆ど行わなくても 41 から 44% の精度の Move Prediction（着手予測）ができることが示さ

れた．

ここで言う Move Prediction とは，テストデータの棋譜中のある盤面に対して，全合法手から次の着手を当てることである．平均 250 ある合法手から次の一手を当てる精度が 41 から 44% ということである．これはそれまでの DCNN を使わない Move Prediction の最高精度を上回るものであった．しかも，探索を一切行わず，この Move Prediction 単体をプレイヤーとして使用するだけで，GNU Go を上回る強さであることも示された．

同様の基準で，Maddison ら [47] では，囲碁に関する知識に基づいた特徴抽出も組み合わせた DCNN を用いることで，約 55% の精度の Move Prediction ができることが示された．加えて，DCNN をモンテカルロ木探索と組み合わせることで強さが向上することも示された．

さらに，Tian and Zhu [55] では，DCNN の中で先読みを行なうことで，約 57% の精度の Move Prediction ができることが示された．しかも，この DCNN による Move Prediction 単体をプレイヤーとして使用するだけで，アマチュア有段者レベルの強さであることも示された．

そして，2016 年 1 月に Silver ら [6] の AlphaGo の論文で，DCNN を使って囲碁の高精度静的盤面評価関数の作成に成功したこと，及びそれにより囲碁 AI として初めてプロレベルのプレイヤーに勝利したことが発表された．その後 AlphaGo は，プロプレイヤーの中でも世界トップレベルのプレイヤーとの 5 番勝負も 4 勝 1 敗で制し，DCNN の囲碁における有効性を示した．

これらの研究は，DCNN が囲碁の局面を認識するツールとして有効であることを示唆している．その事実を基に我々が始めたのが，畳み込みニューラルネットワークを用いた棋力推定に関する研究である．

# 謝辞

指導教員として本研究を指導して下さった村松先生，論文共著者として様々なアドバイスを下さった保木先生，高橋先生，囲碁 AI 製作についてアドバイスを下さった清さん，山下さん，Coulom さん，囲碁クエストの棋譜を提供して下さった棚瀬さん，そして精神面でサポートして下さった研究室の皆さんに深く感謝致します.

また，特別研究員として採用して下さった日本学術振興会に深く感謝致します.（JSPS 科研費番号 15J11695）.

# 関連図書


[1] 棋士 (囲碁). https://ja.wikipedia.org/wiki/%E6%A3%8B%E5%A3%AB_(%E5%9B%B2%E7%A2%81)#.E3.83.97.E3.83.AD.E6.A3.8B.E5.A3.AB.E5.88.B6.E5.BA.A6. [Online: accessed 30-November2016].

[2] 高橋克吉, 伊藤毅志, 村松正和, 松原仁. 次の一手問題を用いた囲碁プレイヤの局面認識についての分析. 情報処理学会論文誌, Vol. Vol.52, No. No.12, 2011.

[3] Elwyn Berlekamp and David Wolfe. Mathematical Go –Chilling Gets the Last Point–. A.K.Peters, 1994.

[4] 中村貞吾. 囲碁の攻合いの数理的解析–組合せゲーム理論に基づく手数の評価法. 情報処理学会論文誌, Vol. Vol.48, No. No.11, 2007.

[5] Rémi Coulom. Computing Elo Ratings of Move Patterns in the Game of Go. In H. Jaap van den Herik, Mark Winands, Jos Uiterwijk, and Maarten Schadd, editors, Computer Games Workshop, Amsterdam, Netherlands, 2007.

[6] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. Nature, Vol. 529, No. 7587, pp. 484–489, 01 2016.

[7] Josef Moudřik, Petr Baudiš, and Roman Neruda. Evaluating Go Game Records for Prediction of Player Attributes. In Computational Intelligence and Games (CIG), 2015 IEEE Conference on, pp. 162–168, 2015.

[8] Akihiro Kishimoto. Search versus knowledge for solving life and death problems in go. In In Twentieth National Conference on Artificial Intelligence (AAAI-05, pp. 1374–1379. AAAI Press, 2005.

[9] David Silver and Gerald Tesauro. Monte-Carlo Simulation Balancing. In Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, pp. 945–952, New York, NY, USA, 2009. ACM.

[10] Shih-Chieh Huang, Rémi Coulom, and Shun-Shii Lin. Monte-Carlo Simulation Balancing in Practice. In Computers and Games, pp. 81–92, 2010.

[11] 人工知能学会. 人工知能の歴史. http://www.ai-gakkai.or.jp/whatsai/AIhistory.html, 2016.

[12] Warren S. McCulloch and Walter Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. The Bulletin of Mathematical Biophysics, Vol. 5, pp. 115–133, December 1943.

[13] Claude E. Shannon. Programming a Computer for Playing Chess. Philosophical Magazine Ser.7, Vol. 41, pp. 209–312, March 1950.

[14] 松原仁, 竹内郁雄（編）. ゲームプログラミング. 共立出版, 1998.

[15] チェス. https://ja.wikipedia.org/wiki/%E3%83%81%E3%82%A7%E3%82%B9. [Online: accessed 11-October-2016].

[16] David Lefkovitz. A Strategic Pattern Recognition Program of the Game of Go, 1960.

[17] 美添一樹, 山下宏. コンピュータ囲碁　モンテカルロ法の理論と実践. 共立出版, 2012.

[18] Alan Levinovitz. The Mystery of Go, the Ancient Game That Computers Still Can't Win. Wired, 2014.

[19] Bernd Brügmann. Monte Carlo Go. Unpublished technical report., 1993.

[20] Sylvain Gelly and David Silver. Combining Online and Offline Knowledge in UCT. In International Conference on Machine Learning, ICML 2007, 2007.

[21] 電気通信大学エンターテインメントと認知科学研究ステーション. 電聖戦. http://entcog.c.ooco.jp/entcog/densei/, 2012-. [Online: accessed 25-August-2016].

[22] The KGS Go Server. http://www.gokgs.com/?locale=ja_JP. [Online: accessed 25-August-2016].

[23] 二人零和有限確定完全情報ゲーム. https://ja.wikipedia.org/wiki/%E4%BA%8C%E4%BA%BA%E9%9B%B6%E5%92%8C%E6%9C%89%E9%99%90%E7%A2%BA%E5%AE%9A%E5%AE%8C%E5%85%A8%E6%83%85%E5%A0%B1%E3%82%B2%E3%83%BC%E3%83%A0. [Online: accessed 15-October2016].

[24] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. Machine Learning, Vol. 47, No. 2-3, pp. 235–256, May 2002.

[25] David R. Hunter. MM algorithms for generalized Bradley-Terry models. The Annals of Statistics, Vol. 32, p. 2004, 2004.

[26] Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

[27] Man Lung Li, Wayne Iba, Daniel Bump, David Denholm, Gunnar Farnebäck, Nils Lohner, Jerome Dumonteil, Tommy Thorn, Nicklas Ekstrand, Inge Wallin, Thomas Traber, Douglas Ridgway, Teun Burgers, Tanguy Urvoy, Thien-Thi Nguyen, Heikki Levanto, Mark Vytlacil, Adriaan van Kessel, Wolfgang Manner, Jens Yllman, Don Dailey, Mans Ullerstam, Arend Bayer, Trevor Morris, Evan Berggren Daniel, Fernando Portela, Paul Pogonyshev, S.P. Lee, Stephane Nicolet, Martin Holters, and Grzegorz Leszczynski. GNU Go. https://www.gnu.org/software/gnugo/. [Online: accessed 12-October-2016].

[28] 小林祐樹. モンテカルロ木探索を用いた強い囲碁プログラムの設計と開発. Master's thesis, UEC, 2016.

[29] Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. A Maximum Entropy Approach to Natural Language Processing. Comput. Linguist., Vol. 22, No. 1, pp. 39–71, March 1996.

[30] 電気通信大学. 第 8 回 UEC 杯. http://www.computer-go.jp/uec/public_html/past/2014/index.shtml, 2015. [Online: accessed 11-October-2016].

[31] 張栩. 黒猫のヨンロ. 日本棋院, 2012.

[32] Nobuo Araki, Masakazu Muramatsu, Kunihito Hoki, and Satoshi Takahashi. Monte-Carlo Simulation Adjusting. In Proceedings of the 28th AAAI Conference on Artificial Intelligence and the 26th Innovative Applications of Artificial Intelligence Conference, AAAI '14, pp. 3094–3095, 2014.

[33] 福井正明. 画期的囲碁上達法. 日本棋院, 2002.

[34] 公益財団法人日本生産性本部（編）. レジャー白書 2015 国内旅行のゆくえと余暇. 生産性出版, 2015.

[35] 荒木伸夫, 保木邦仁, 村松正和. 畳み込みニューラルネットワークを用いた囲碁における 1 局の棋譜からの棋力推定. 情報処理学会論文誌, Vol. Vol.57, No. No.11, 2016.

[36] 棚瀬寧. 囲碁クエスト. http://wars.fm/go9?lang=ja.

[37] 棚瀬寧. 囲碁クエスト棋譜. by private communication.

[38] Matej Guid and Ivan Bratko. Using Heuristic-Search Based Engines for Estimating Human Skill at Chess. ICGA Journal, Vol. 2, No. 34, pp. 71–81, 2001.

[39] 山下宏. 将棋名人のレーティングと棋譜分析. ゲームプログラミングワークショップ 2014 論文集 , 2014.

[40] Kaggle. FindingElo. https://www.kaggle.com/c/finding-elo, 2014-2015.

[41] Amr S. Ghoneim, Daryl L. Essam, and Hussein A. Abbass. Competency Awareness in Strategic Decision Making. In Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), 2011 IEEE First International Multi-Disciplinary Conference on, pp. 106–109, 2011.

[42] ランクシステム. https://www.gokgs.com/help/rank.html. [Online: accessed 10-January-2017].

[43] Josef Moudřik and Petr Baudiš. GoStyle - Determine playing style in the game of Go. http://www.gostyle.j2m.cz/, 2013.

[44] 麻生英樹, 安田宗樹, 前田新一, 岡野原大輔, 岡谷貴之, 久保陽太郎, ボレガラダヌシカ. 深層学習. 近代科学社, 2015.

[45] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv preprint arXiv:1408.5093, 2014.

[46] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In not specified, editor, International Conference onartificial intelligence and statistics, pp. 249–256, Amsterdam, Netherlands, 2010.

[47] Chris J. Maddison, Aja Huang, Ilya Sutskever, and David Silver. Move Evaluation in Go Using Deep Convolutional Neural Networks. In International Conference on Learning Representations. 2015.

[48] Computer Go Forum mailing list.

[49] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pp. 1097–1105. Curran Associates, Inc., 2012.

[50] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. In Proceedings of the IEEE, Vol. 86, pp. 2278–2324, 1998.

[51] Steve Lawrence, C. Lee Giles, Ah Chung Tsoi, and Andrew D. Back. Face Recognition: A Convolutional Neural-Network Approach. IEEE Transactions on Neural Networks, Vol. 8, No. 1, pp. 98–113, 1997.

[52] Francois van Niekerk and Detlef Schmicker. oakfoam. https://bitbucket.org/dsmic/oakfoam.

[53] NVIDIA. Tesla GPU でデータセンターを高速化. http://www.nvidia.co.jp/object/tesla-servers-jp.html.

[54] Christopher Clark and Amos Storkey. Teaching Deep Convolutional Neural Networks to Play Go. In Proceedings of ICML 2015, 7 2015.

[55] Yuandong Tian and Yan Zhu. Better Computer Go Player with Neural Network and Long-Term Prediction. In International Conference on Learning Representations. 2016.