

博士論文

ゲーム木探索における知識の獲得とその評価

万代 悠作

目次

第 1 章	序論	1
第 2 章	背景	5
2.1	ゲーム木とゲームのモデル	5
2.1.1	P-game tree	6
2.1.2	Finnsson と Björnsson によるゲーム	7
2.2	ゲーム木探索	8
2.2.1	Minimax 探索	9
2.2.2	モンテカルロ木探索	9
2.3	ゲームにおける評価関数の学習	11
2.3.1	AlphaGo	11
2.3.2	DeepChess	13
2.3.3	Discriminator を利用した学習	14
2.4	バンディット問題とそのアルゴリズム	15
2.4.1	多腕バンディット問題	15
2.4.2	線形バンディット問題	16
2.5	その他の機械学習手法	20
2.5.1	マルチタスク学習	20
2.5.2	RankNet	20
2.6	深層ニューラルネットワークの判断根拠の可視化	22
2.6.1	Saliency Map	22
2.6.2	SmoothGrad	23
第 3 章	局面の特徴を考慮した Incremental Random Game Tree	24
3.1	Point Mutation Feature	24
3.1.1	Point Mutation Feature の性質	26
3.2	Incremental Random Feature	29
3.2.1	Incremental Random Feature の性質	30
3.3	まとめ	31
第 4 章	LinUCT による探索中の知識の獲得	32
4.1	アルゴリズム	32
4.1.1	LinUCT-PLAIN	32

4.1.2	LinUCT-RAVE	33
4.1.3	LinUCT-FP	34
4.1.4	LinUCT-WP	34
4.1.5	Leaf-LinUCT	35
4.2	評価実験	36
4.2.1	Point Mutation Tree での評価	36
4.2.2	Shock Step Game での評価	38
4.2.3	Incremental Random Feature Tree での評価	40
4.2.4	Progression game での評価	42
4.3	まとめ	43
第 5 章	LinUCT の時間計算量向上	46
5.1	fLinUCT-GD	46
5.2	実験環境	46
5.3	Point Mutation Feature Tree における性能	47
5.4	囲碁における性能	48
5.4.1	実装	48
5.4.2	特徴	49
5.4.3	UCT との比較	50
5.4.4	同一思考時間における比較	51
5.5	まとめ	52
第 6 章	RankNet による効率的な評価関数の学習	53
6.1	RankNet による評価関数の学習	53
6.2	評価実験	54
6.2.1	学習棋譜と実装	54
6.2.2	学習結果	56
6.2.3	対局実験	56
6.2.4	テストセットでの性能評価	59
6.3	まとめ	60
第 7 章	Discriminator を用いたマルチタスク学習による効率的な評価関数の学習	62
7.1	囲碁における Discriminator によるマルチタスク学習	62
7.1.1	ネットワークの構成	62
7.1.2	損失関数	63
7.2	実験による評価	63
7.2.1	実験設定	64
7.2.2	学習結果	65
7.2.3	対局実験	65
7.2.4	次の一手の正答率による評価	66
7.3	まとめ	66

第 8 章	深層ニューラルネットワークと探索を組合せた判断根拠の可視化	69
8.1	囲碁の深層ニューラルネットワークに対する Saliency Map	69
8.2	深層ニューラルネットワークとモンテカルロ木探索による価値予測結果に対する Saliency Map	70
8.3	重要な石の検出実験	73
8.3.1	Saliency Map の結果の集約方法	73
8.3.2	方策予測に対する Saliency Map	73
8.3.3	局面価値予測に対する Saliency Map	74
8.3.4	MCTS に対する Saliency Map	75
8.4	まとめ	75
第 9 章	結論	77

目次

1.1	本論文の構成	4
2.1	P-game tree の例	7
2.2	Finnsson と Björnsson の Shockstep Game	8
2.3	Finnsson と Björnsson の Progression Game	8
2.4	DeepChess のネットワーク構成 [43]	13
2.5	マルチタスク学習の例	21
3.1	Point Mutation Feature Tree の例.	25
3.2	Point Mutation Feature Tree の報酬の分布	27
3.3	Point Mutation Feature Tree の報酬の分散	27
3.4	Point Mutation Feature の要素が異なる確率	29
3.5	Parent-Grandparent Mutation Feature の要素が異なる確率	29
3.6	Incremental Random Feature Tree の例	30
4.1	パラメータによる LinUCT の性能変化	37
4.2	UCT と LinUCT の性能比較 (point mutation feature tree)	38
4.3	UCT と LinUCT の性能比較 (decayed point mutation feature tree)	39
4.4	UCT と LinUCT の性能比較 (parent-grandparent feature tree)	40
4.5	UCT と LinUCT の比較 (shock step game)	41
4.6	UCT と LinUCT の比較 (incremental random feature tree における誤答率)	42
4.7	UCT と LinUCT の比較 (incremental random feature tree における average regret)	42
4.8	LinUCT の予測値の正確性	44
4.9	LinUCT の予測ベクトルの時間変化	45
4.10	LinUCT の予測値の時間変化	45
5.1	fLinUCT-GD と UCT の実行時間比較 (分枝数 4, 深さ 4)	49
5.2	fLinUCT-GD と UCT の実行時間比較 (分枝数 8, 深さ 4)	49
5.3	第 5 章 で用いた特徴ベクトルの例	50
6.1	学習曲線 (BASELINE-CONF1)	57
6.2	学習曲線 (BASELINE-CONF2)	57
6.3	学習曲線 (POLICY-RANKNET)	57

6.4	学習曲線 (RANKNET)	58
6.5	学習局面と学習曲線の関係 (BASELINE と POLICY-RANKNET)	58
6.6	テストセットにおける Sign accuracy	61
6.7	テストセットにおける Rank accuracy	61
6.8	テストセットにおける Policy accuracy	61
7.1	学習曲線 (value loss)	68
7.2	対局時の勝率 (対 Pachi)	68
7.3	次の一手予測の Top-1 Accuracy	68
7.4	次の一手予測の Top-5 Accuracy	68
7.5	次の一手予測の Top-10 Accuracy	68
8.1	得られた Saliency Map の例	70
8.2	方策評価の手数ごとの平均変動値 ($H(p(s_0), p(s'))$)	74
8.3	パラメータごとの方策評価の平均変動値 ($H(p(s_0), p(s'))$)	74
8.4	局面価値評価の手数ごとの平均変動値 ($ v(s_0) - v(s') $)	75
8.5	パラメータごとの局面価値評価の平均変動値 ($ v(s_0) - v(s') $)	75
8.6	ノイズレベルごとの MCTS 評価値の平均変動値 ($ \hat{v}(s_0) - \hat{v}(s') $)	76

表目次

4.1	UCT と LinUCT の比較 (progression game)	43
5.1	第 5 章 の実験環境	47
5.2	fLinUCT-GD と UCT の誤答率の比較 (分枝数 4, 深さ 4)	48
5.3	fLinUCT-GD と UCT の誤答率の比較 (分枝数 8, 深さ 4)	48
5.4	第 5 章 の実験パラメータ	49
5.5	fLinUCT-PLAIN と UCT の勝率 (同一プレイアウト回数)	51
5.6	fLinUCT-RAVE と UCT の勝率 (同一プレイアウト回数)	51
5.7	fLinUCT-PLAIN と UCT の実行時間比較	51
5.8	fLinUCT-RAVE と UCT の実行時間比較	51
5.9	fLinUCT-GD と UCT の時間当たりの平均プレイアウト回数比較	52
5.10	fLinUCT-GD と UCT の勝率 (同一思考時間)	52
5.11	fLinUCT-GD と UCT-RAVE の勝率 (同一思考時間)	52
6.1	第 6 章 の学習データセットの内訳	55
6.2	第 6 章 のニューラルネットワークの要約	55
6.3	第 6 章 の学習設定	56
6.4	各モデルの最良 epoch	58
6.5	対局時の勝率 (対 Pachi)	59
6.6	対局時の勝率 (POLICY-RANKNET 対 BASELINE)	59
7.1	第 7 章 のニューラルネットワークの要約	64
8.1	対局時の勝率 (対 Pachi)	73

アルゴリズム目次

1	LinUCB	19
2	fLinUCB-GD	20
3	Supplementary Procedures in LinUCT	33
4	Back-propagation process of LinUCT-FP	34
5	Back-propagation process of LinUCT-WP	35
6	fLinUCT-GD	47

概要

本研究ではゲームを題材に、対象となるゲームの知識をゲーム木探索に利用するためにどのように効率よく獲得すればよいか、また獲得した知識をどのように評価するかについての研究を行った。

ゲーム木探索とは与えられたゲームの利得を明らかにするために行われる探索手法であり、ゲームをうまくプレイするためには木探索によって先読みをする必要がある。ゲーム木探索に関してはこれまでにいくつかの先行研究が存在し、優れた手法によっていくつかのゲームでは人間のトッププレイヤーよりも強いコンピュータプレイヤーを作成することに成功している。一般的に広く遊ばれているようなゲームでは現実的な時間で全ての展開を探索することは不可能であるため、評価関数と呼ばれるゲームの局面の良さを測る関数を用いてヒューリスティックに探索を行う。評価関数とは対象となるゲームの知識を関数の形で表現したものであり、評価関数が正確なほどそのゲームをうまくプレイできる。コンピュータプレイヤーの強さは、実行されるコンピュータの性能と探索アルゴリズムの巧拙、そして評価関数の正確性によって決定される。

評価関数は人間が関数の形を設計し、機械学習によって関数のパラメータの重みを調整するという手法が主流である。学習は人間の知識を一切用いない状態からの強化学習によって行われることもあるが、強いプレイヤーの棋譜を利用した教師あり学習による学習が多くの先行研究で行われており、精度の高い評価関数を作成することが可能であると示されている。

しかし棋譜が存在しないゲームにおいて評価関数を学習する際には上記のような教師あり学習を行うことが不可能であるし、棋譜が存在するが数が少ないようなゲームでは少ない棋譜を用いて効率的に学習をする必要がある。また別の状況として、事前に学習ができないようなゲームも考える。例えば General Game Playing に代表される、初見のゲームをうまくプレイするようなコンピュータプレイヤーを作成することを目的とした環境では強化学習、教師あり学習ともに行うことができないため、うまくゲームをプレイするためには探索と同時にそのゲームの知識をなんらかの方法で学習することが重要である。

本研究ではそのような問題を念頭に、ゲーム木探索とともに利用する知識の獲得を如何に効率的に獲得できるかという点について研究を行った。具体的には対象とするゲームに関して事前に強いプレイヤーの棋譜が存在する場合と、事前に学習を行えない場合の二つの場合について研究を行った。

第一に、対象となるゲームについて事前に学習ができないような状況におけるゲーム木探索アルゴリズムの性能改善に取り組んだ。このような状況は新たに考案されたゲームや、General Game Playingなどで課題となりうる。この問題設定において、本研究ではモンテカルロ木探索と呼ばれるゲーム木探索アルゴリズムと、線形バンディット問題において有用であるとされた LinUCB アルゴリズムとを組み合わせた新たなモンテカルロ木探索である LinUCT を提案し、その性能を評価した。モンテカルロ木探索は事前知識を必要としないシミュレーションを用いたアルゴリズムで、先述の General Game Playing やそのビデオゲーム版である General Video Game Playing などで有効であると確かめられている。LinUCB は行動選択のドメインでオンライン学習を用いてある行動に対する報酬を予測し、予測した報酬を次の行動選択に役立てることができるアルゴリズムである。これら二つを組み合わせたゲーム木探索を提案し、局面の特徴と報酬とが関係するドメインで有効性を確認した。また囲碁などの人間がプレイするような現実的なゲームにおいては局面の特徴空間が膨大になり、単純な LinUCB アルゴリズムでは時間計算量的に不利になる。このようなゲームにおいて、報酬の予測を近似することで時間計算量を改善し、現実的な実行時間を達成するアルゴリズムについても提案した。

第二に、強いプレイヤーの棋譜が存在し事前に利用できるような状況において、その限られた棋譜を有効に利用できるような手法を考案し、その性能を実験的に確認した。強いプレイヤーの棋譜を利用した学習は、棋譜に存在する局面の価値（最終的な勝敗）の予測や、その局面で選択すべき着手の予測という定式化が可能であり、標準的な学習法では単一の局面についての予測を行っている。本研究では新たに二つの局面の優劣の予測に基づくアルゴリズムを考案し、そのアルゴリズムの効率性を議論した。この二つの局面の比較という新たな枠組みでは、既存のアルゴリズムと比べて学習に利用できる実質的な訓練例の数を増やすことが可能であり、少ない棋譜しか利用できないような場合でも効率的に学習することが可能であることを示した。また学習において、複数の学習目標を同時に学習するマルチタスク学習と呼ばれる手法が汎化性能を上げるために有効であると既存研究が示している。既存手法ではある局面について着手予測と局面価値予測とのマルチタスク学習を行っており、性能が向上したと報告されている。この複数の学習目標について、着手予測ではなくより単純な学習目標を用いることでも汎化性能が向上することを示した。より単純な学習目標として discriminator と呼ばれる予測器を採用し、discriminator と局面価値の予測器とのマルチタスク学習の性能を調査した。この discriminator は generative adversarial networks で用いられたものを参考にしており、与えられた局面が現実の対局でどれだけ現れやすいかを示すスカラー値を出力する。Discriminator はベクトル値を出力する着手予測と比較して構成要素が単純になるという利点がある。

最後に、得られた知識を検証するという目的で、上記二つそれぞれに対応する手法を検証した。最初の事前知識が得られず、探索アルゴリズムによって知識を逐次得ていくようなアルゴリズムの評価を行えるようなテストベッドを考案し、そのいくつかの性質について議論した。提案したテストベッドは局面の特徴を用いない既存の標準的なテストベッドを自然な形で拡張することにより、既存のテストベッドの良い性質を保ったまま利用できるものである。二つ目の、ゲームに関する事前知識が得られ効率的に学習が行えた場合においては、得られた知識の妥当性を人間が検証できるような方法について議論を行った。具体的には知識を機械学習アルゴリズムを用いて獲得し、予測器が構成できたときにその予測器が入力のどの部分について注目して判断を下したのかについて、人間が容易に理解できるような形で入力の影響を可視化する手法について、ゲーム特有のアルゴリズムを念頭においた改良を行い、妥当性を実験的に示した。

第 1 章

序論

ゲームに関する研究は人工知能という一大研究分野と同時期に始まったといえる。ゲームはそのルールの明確性からゲームをプレイする人工知能（コンピュータプレイヤー）の性能の評価が容易であり、人間のような知性を機械に与えるという人工知能の目的を達成する上で重要なテストベッドの役割を果たしている。ゲームをうまくプレイするためには先読みの能力、現状および未来の状況を適切に判断する能力の両方が求められ、その二つの観点から研究が進められた。前者は探索アルゴリズムの深化に繋がり、後者はゲームの局面を評価する評価関数の研究につながる。

ゲームにおける評価関数 $f_\theta : \mathcal{S} \rightarrow \mathbb{R}$ は局面 $s \in \mathcal{S}$ を入力、その局面の良さを実数値として出力とする関数である。評価関数の作成には主に (1) 評価関数 f_θ の設計 と (2) 設計した評価関数の重み θ の調整 の二つの段階に分けることができる。設計の段階では f_θ の形を決定し、その後重み θ を調整する。簡単な実装では人間が f を決定木の形で設計し、人間の知識を利用して決定木の重みを調整する方法が取られた（ルールベース）。評価関数の設計は人間によってなされているが、評価関数の重みの調整は機械学習の手法を用いて行う研究が盛んに行われている。古くは Samuel がチェッカーの評価関数を機械学習により調整 [1] しており、Tesauro は TD 学習によってバックギャモンの評価関数を作成した [2]。Tesauro の TD-Gammon は人間の實力を超え、コンピュータが人間を超えることができることを最初期に示した例の一つである。その後計算機の性能と探索アルゴリズムの進歩も伴い、チェスにおいて IBM の DeepBlue が当時の人間の世界チャンピオンに勝利した [3]。またオセロにおいても Buro の Logistello が世界チャンピオンを 6-0 で勝利している [4]。日本においては囲碁とともに人気の高い将棋においての研究が精力的になされている。特に保木の Bonanza は将棋において当時としては大規模な棋譜からの教師あり学習を行なった評価関数と探索アルゴリズムを用い、2006 年、2013 年の世界コンピュータ将棋選手権で優勝している [5]。一方で囲碁においては 2010 年代になるまで正確な評価関数の作成は困難であった。囲碁で使われる石はすべて価値が等しいためチェスや将棋のように駒得を手がかりできず、またオセロのように盤面に特徴のある箇所もないため評価関数の設計が難しかったためである。そのため、シミュレーションによって局面を評価するモンテカルロ木探索と呼ばれる手法が取られ、多くのプログラムがモンテカルロ木探索を採用していた [6, 7, 8]。しかし 2016 年に DeepMind による AlphaGo が登場し、囲碁の正確な評価関数が深層ニューラルネットワークを用いた教師あり学習と強化学習によって作成できることを示した [9]。その後継である AlphaGo Zero は人間の棋譜を用いず、強化学習を行うことで人間の棋譜を用いたものよりも強いコンピュータプレイヤーを作成した [10]。AlphaGo Zero の手法は汎用的であり、同様の手法がチェスと将棋においても有効であることも示された [11]。AlphaGo Zero が用いた、深層ニューラルネットワークを用いた強化学習による評価関数の学習はその他のゲームでも有効であると考えられる。しかし、AlphaGo Zero の強化学習に要する計算は非常に大規模で、2017 年における一

般的な家庭用計算機を用いる場合その再現に 1000 年単位の時間がかかるとの試算がある^{*1}。そのため、そのような方法が取れない場合には別のアプローチによって評価関数を学習する必要がある。

あるゲームを対象に評価関数を学習するとき上述のような大規模な強化学習ができない場合、既存の情報源を利用して学習するという方法が考えられる。既存の情報源とは強いプレイヤーが行動した記録である棋譜が代表的であり、棋譜を利用して学習することによりそのプレイヤーの行動を模倣することが可能になる。さらに、これをブートストラップとし、ある程度強いプレイヤーを作成したのちに強化学習を行うことでさらに強くするという方法も考えられる。また別の状況として、既存の棋譜もなく、事前に強化学習も行うことができないようなゲームも考えられる。例えば General Game Playing (GGP) や General Video Game Playing (GVGP) は、複数の初見のゲームを上手にプレイするようなコンピュータプレイヤーの作成を目的として作られた研究環境である。GGP や GVGP では事前に学習をすることができないため、評価関数を用いない探索手法であるモンテカルロ木探索が標準的なアルゴリズムとされている [12, 13]。このような状況下においては、探索を行いつつ、そのゲームの知識を獲得していくという手法が考えられる。

本研究では上述した二つの側面についてそれぞれ別のアプローチを取ることで、ゲーム木探索で用いる知識を効率的に獲得できるような方法について研究を行った。

評価関数の学習に強いプレイヤーの棋譜が利用できる場合について、既存のアルゴリズムよりも効率的に学習が行えるようなアルゴリズムを考案し、その性能を評価した。具体的には単一の局面の良さの予測を最適化するのではなく、二つの局面のうちどちらがその局面から勝利しやすいのかという予測を最適化する学習法を構築した。この場合組み合わせを用いるため学習に利用できる実質的なデータ数を増やす効果があると予想される。また学習を安定化させ、汎化性能を向上させる手法として古くからマルチタスク学習という手法が知られている [14]。AlphaGo Zero は局面価値関数と方策予測のマルチタスク学習によってより効率的な学習を行っている。そこで方策予測の代わりとなる、discriminator と呼ばれる局面のもっともらしさを予測する関数と局面価値関数とのマルチタスク学習の性能を方策予測とのマルチタスク学習の性能と比較し、有効性を示した。

また棋譜が利用できず、事前に学習ができない場合について、ゲーム木探索を行うと同時にゲームの知識を獲得し、獲得した知識を次の行動選択に役立てるアルゴリズムについて考案した。木探索アルゴリズムとしてモンテカルロ木探索を用い、そのシミュレーションによる評価を元にゲームの知識を評価関数の形で利用できるアルゴリズムである。その際にできるだけ無駄な行動を行わないように、線形バンディット問題と呼ばれるに利用される LinUCB を用いることで探索と知識利用のジレンマ (exploration-exploitation dilemma) を解決し、現在興味のある局面の近傍についてうまく学習できるような設計を行なった。

最後に、提案したゲームの特徴を利用する探索アルゴリズムを効果的に検証できる評価手法の提案や、得られた知識の説明を行うようなアルゴリズムを現時点で可能な範囲で評価を行った。探索アルゴリズムの検証に用いる評価手法は研究者が容易に探索空間の設計ができ、探索アルゴリズムの性質に応じて評価実験が行いやすいものである。またその評価手法の性質について分析を行い、提案した評価手法は既存の特徴を利用しない探索アルゴリズムの評価に用いられた手法の自然な拡張になっていることを示した。得られた知識の説明を行うアルゴリズムは今後人間社会が人工知能への依存度を高めていくような状況では、人間が人工知能の意思決定の妥当性を検証する上で重要な応用となる。得られた知識を評価するという点で、提案したアルゴリズムはゲーム木探索アルゴリズムとともに用いられる知識の評価が妥当に行えることを示した。

^{*1} <https://github.com/gcp/leela-zero#gimme-the-weights>. Accessed on 2018/12/10.

■本論文の構成 第2章では本研究に関連する研究を記した。

第3章では局面の特徴を用いるゲーム木探索アルゴリズムを評価するための新たな人工木についての提案について記した。

第4章では事前に学習ができないまたは困難であるようなゲームについて、効率的に知識を獲得しつつ探索の効率を向上させる新たなアルゴリズムについて記した。このアルゴリズムはモンテカルロ木探索の変種として提案しており、モンテカルロ木探索の選択の際に特徴を効率的に利用する LinUCB アルゴリズムを用いて探索を行う方法について論じ、その後第3章で提案した人工木での評価を行った。

第5章では第4章で提案したアルゴリズムの時間計算量的を改善したアルゴリズムについて記した。また実際のゲームにおけるアルゴリズムの計算量や性能についても論じた。

第6章では事前に対象とするゲームの知識が棋譜の形で利用できるような状況において、既存のアルゴリズムよりも効率的に知識を獲得する方法について記した。ここでは Learning-to-rank 学習における RankNet アルゴリズムをゲームにおける評価関数の学習に応用し、既存のアルゴリズムとの実験的な比較を行った。

第7章では第6章と同様に強いプレイヤーの棋譜が利用できる場合における効率的な学習方法について述べた。この章ではマルチタスク学習に着目し、AlphaGo Zero とは異なるマルチタスク学習の性能を比較した。具体的には Wan らが提案した、棋譜中に含まれる局面とランダムに生成された局面とを見分ける discriminator [15] と局面価値予測とのマルチタスク学習について囲碁における性能比較を行なった。

第8章ではゲームに関する知識が得られた状況において、その得られた知識による意思決定の根拠の可視化について議論を行った。具体的には囲碁において、学習によって得られた予測器の出力およびそれを探索アルゴリズムとともに用いた際の着手判断・形勢判断についての根拠についてその可視化手法を論じている。

第9章には本論文の結論を記した。

図 1.1 に本論文の構成を示す。第3章と第4章、および第5章は探索中に知識を獲得するという側面について述べており、互に関連した議論を行っている。第6章と第7章は限られた棋譜による効率的な学習について、それぞれ独立した手法を扱っている。第8章では前二章のアルゴリズムによって獲得した知識を対象とした判断根拠の可視化を行っている。

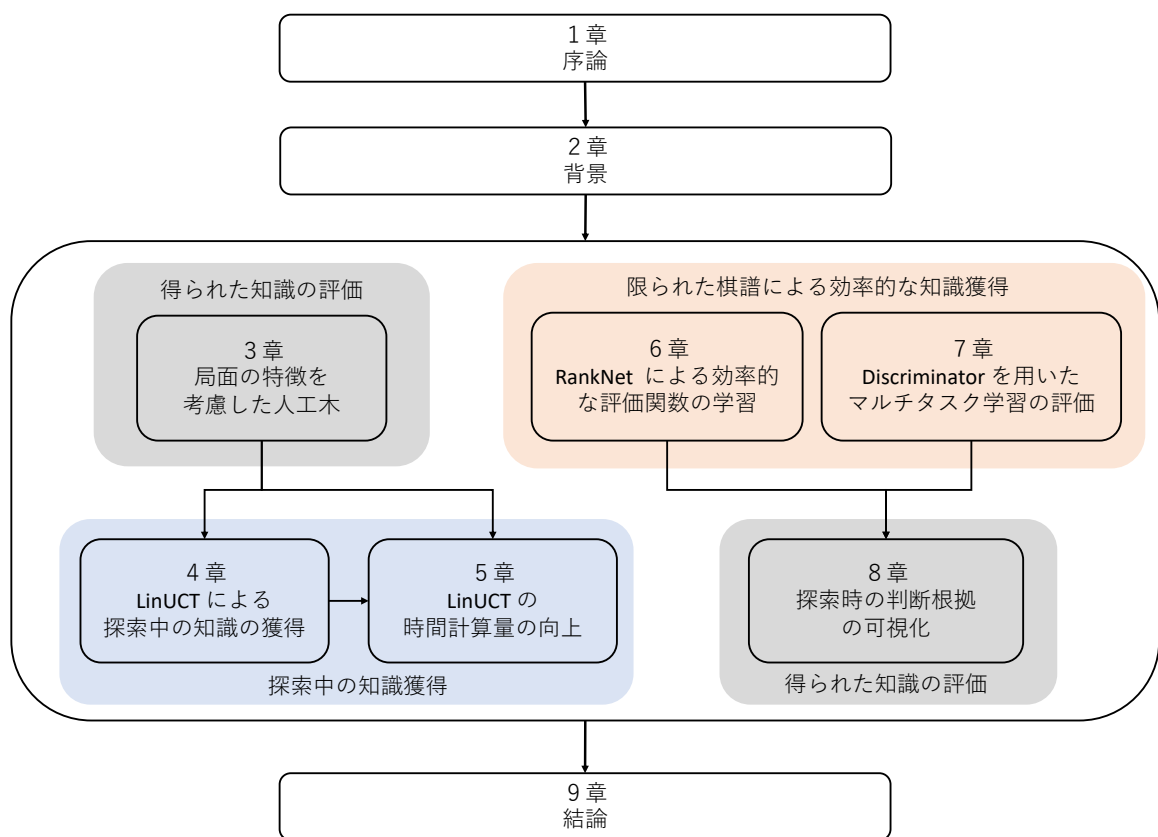


図 1.1: 本論文の構成

第 2 章

背景

本研究ではゲームを一人以上のプレイヤーが明確なルールのもとでプレイヤー同士及び環境と相互作用しながら定量化可能な結果について競うものと定義する。ゲームは明確なルールがあるため計算機が扱いやすく、定量化可能な結果によって性能の議論がしやすいという点で人工知能の研究の初期段階から多くの研究者が対象としてきた。特にチェスや囲碁に代表される二人零和有限確定完全情報ゲームについては様々な研究が存在する。

二人零和有限確定完全情報ゲームは、プレイヤー数が二人であり（二人）、一方のプレイヤーの利得はもう一方のプレイヤーの損になり（零和）、プレイヤーが取りうる行動の組み合わせの数が有限で（有限）、任意の行動のあとのゲームの状態が確定的に定まり（確定）、ゲームの完全な状態が双方のプレイヤーに提供される（完全情報）ゲームである。本稿では特に断らない限り二人零和有限確定完全情報ゲームのことをゲームと呼ぶ。

本章では本研究に関連する研究を紹介する。特に探索の対象としたときのゲームの表現であるゲーム木、その代表的な探索手法であるモンテカルロ木探索や、ゲームの評価関数の作成法について主に述べる。また評価関数がニューラルネットワークとしてモデル化されている際、その評価関数の出力の妥当性の検証のために入力がどのように出力に影響したかを可視化する手法についての関連研究も述べる。

2.1 ゲーム木とゲームのモデル

ゲーム木はゲームをグラフ構造で表したもので、ゲームの状態を節点、プレイヤーの行動を辺とした有向グラフである。多くのゲームはある局面から同一の局面に遷移することができるような複数の異なる着手列（合流）が存在する場合があります。ある局面から同じ局面に遷移することができる複数の着手（閉路）が存在しうる。よって一般的にゲームをグラフ構造で表したときには木でないことが多い。

原理的にはゲーム木を全探索すれば各プレイヤーが最善を尽くしたときの利得を計算することが可能である。あるゲームについて理論的な利得を明らかにすることをゲームを解く (solve) といい、効率的にゲーム木を探索することでゲームを解くことが可能になる。ゲームを解くとはいくつかの段階があり、ultra weakly solved (開始局面の理論的利得が得られる), weakly solved (開始局面における最善戦略が得られる), および strongly solved (すべての合法局面における最善戦略が得られる) がある [16, 17]。Strongly solved されたゲームとしては Nim や三目並べなどのゲームやどうぶつしょうぎ [18], 6×6 までの Hex [17] や 4×4 および 6×6 のオセロ ^{*1} などが存在する。Checkers (draughts) [19] と heads-up limit hold'em [20] は weakly

^{*1} 論文として出版されていない。 <https://web.archive.org/web/20091101013931/http://www.feinst.demon.co.uk/Othello/6x6sol.html>

solved された比較的大きなゲームとして有名である。しかしながらより複雑なゲームを解くためには時間的・空間的により多くの計算量が必要となる。

ゲームのコンピュータプレイヤー（対象のゲームをプレイすることができるコンピュータプログラム）の多くは、ゲームの状態（局面）や行動の評価と展開されうる局面の先読みを組み合わせることで次の着手を決定する。局面の先読みは探索アルゴリズムによって実行され、その巧拙がコンピュータプログラムの性能に影響する。局面や行動を評価するために評価関数と呼ばれる関数が主に利用され、評価関数が正確であればあるほど強いプレイヤーが構成できる。ゲームを strongly solve するということはすべての合法局面で勝ち・負け・引き分けを正確に判定できる評価関数を作成するということと同義である。次節以降で探索アルゴリズムと評価関数について概説する。

探索アルゴリズムの評価としてゲームが有用であると述べたが、探索アルゴリズムの理論的な評価を行う際には不向きである場合も多い。複雑すぎるゲームではアルゴリズムが正しく次の着手を評価しているかどうかを研究者が判断することが困難であるからである。そこで探索空間を研究者が制御でき、事前に最善手を研究者が知ることができるようなゲームを模したモデルがいくつか提案されてきた。本節の以降ではいくつかのそのようなモデルを紹介する。

2.1.1 P-game tree

P-game tree [21, 22] は二人零和有限確定完全情報ゲームを模したゲームのモデルであり、探索アルゴリズムの性能評価のテストベッドとして用いられてきた [6, 23, 24, 25]。P-game tree ではアルゴリズムの評価者がある程度自由に探索空間を制御でき、またその構成方法が単純なため広く用いられている。図 2.1 に P-game tree の構成例を示す。P-game tree ではすべての辺にスコアを割り与える。このとき先手番プレイヤー（後手番プレイヤー）の手番を表す節点からの辺には $[0, 127]$ ($[-127, 0]$) のスコアを一様ランダムに割り当てる。ある局面の評価値は、根節点からその局面までへの経路上のスコアの総和とする。図 2.1 の例では、スコア s_1, s_2 の値は $[0, 127]$ の値となり、 s_{11}, s_{12}, s_{21} , および s_{22} の値は $[-127, 0]$ の値となる。葉節点 A, B, C, および D のスコアはそれぞれ $s_1 + s_{11}, s_1 + s_{12}, s_2 + s_{21}$, および $s_2 + s_{22}$ となる。

スコアの割り当てに単一のシード値を用いて幅 W 、深さ D の一様な P-game tree をメモリ上に木を保持する場合、単純な実装では $O(W^D)$ の空間計算量が必要となる。しかし乱数生成器に線形合同法を用いれば木全体をメモリ上に保持しておかなくとも巨大な木の探索が可能である。例えばレベル順に節点を作成することにより木を構築する場合を考える。このとき根節点から深さ d のある節点への経路を復元するためには $n_1 + n_2 + \dots + n_d$ 個の節点を作成する必要がある。ここで n_i は深さ $i - 1$ の経路上の節点から深さ i の経路上の節点の間に存在する節点をレベル順に数えたときの数である。図 2.1 中の節点 D では $n_1 = 2, n_2 = 3$ となる。すべての節点でスコアを求めるとするとこの経路復元は $n_1 + \dots + n_d = O(W^d)$ の時間計算量がかかるが、すべての節点についてスコアを求める必要はなく、線形合同法は現在の状態から n 個先の乱数を $O(\log n)$ で行うことができる^{*2}。よって復元にかかる時間計算量は $O(\log n_1 + \dots + \log n_d)$ である。 $n_i \approx W^i$ であることに注意すると、 $O(d^2 \log W)$ となる。

線形合同法以外の乱数生成アルゴリズムについて、例えばメルセンヌ・ツイスタでは文献 [26] のアルゴリズムによって乱数の先読みにかかる時間計算量を改善できる。しかしながらここで議論するような大きさの人工木については文献 [26] のアルゴリズムが有効には働かない（内部状態のビット数 k について $O(k^{\lg 3})$ の時間

^{*2} Boost などの実装

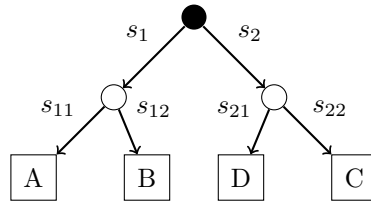


図 2.1: P-game tree の例. ある節点の評価値は根節点からその節点までの経路上の辺に割り当てられたスコアの総和で定義される. 例えば ノード A の評価値は $\text{eval}(A) = s_1 + s_{11}$ となる.

計算量であり, 典型的な設定のメルセンヌ・ツイスタは $k = 19937$) ため, 線形合同法の使用が適している.

2.1.2 Finnsson と Björnsson によるゲーム

Finnsson と Björnsson は後述する UCT の性能評価のために仮想的な二人ゲームを提案している. 彼らはいくつかのゲームを提案しているが, ここでは *Shock Step Game* と *Progression Game* と呼ばれるゲームについて紹介する. どちらのゲームも後述する UCT の性質を調査するために用いられた人工的なゲームであり, UCT の性能が分枝数や深さなどによる探索空間の大きさではなく, 次善手がどのように割り当てられているかが重要であることを実験的に示した.

Shock Step Game

図 2.2 は Shock Step Game の局面を模式化したものである. このゲームは横 c , 縦 r のチェス盤を用いて行う. このゲームにおいて, 先手番プレイヤーは左半分の, 後手番プレイヤーは右半分の盤面を使用する. それぞれのプレイヤーの目的はゲーム開始時に初期位置 (a1 および i8) にいる駒を上向き (下向き) に動かしゴール位置 (旗) へ動かすことである. プレイヤーは手番において駒を一つ上 (下) の行へ進める. このときに選択した移動先のマスによって罰金が与えられる. 先手番プレイヤーは一番左の列を選択すれば罰金が 0 であるが, 左から二番目の列に移動した場合は 1, 左から三番目に移動した場合は 2, というふうに罰金が決定される. 後手番プレイヤーは逆に一番右の列であれば 0, 右から一番目であれば 1, となる. 盤中 X は進入不可のマスを表す. 両方のプレイヤーがゴールにたどり着いたときにゲームは終了となり, 罰金が少ないプレイヤーが勝利となる. 探索アルゴリズムには各マスの罰金の値は知らされず, ゲームの報酬は終了時の報酬のみが明かされる.

Progression Game

図 2.3 は Progression Game の初期配置を示している. このゲームにおいて, プレイヤーの合法手数はゲームのボードサイズの半分存在する. ゲームボードの左半分の駒 (図中白) が先手プレイヤーの駒, 右半分の駒 (図中黒) が後手プレイヤーの駒である. 各ターンにおいてプレイヤーは自分の駒の中から一つ前に進める駒を選択する. 選択した駒が active (図中ポン) であれば駒は一マス前に進むが, 選択した駒が inactive (図中円形矢印) であれば駒は前に進まない. 先に一つでも駒をゴールマスに進めたプレイヤーの勝ちとなる.

このゲームにおいて各プレイヤーに対しどの駒が active か inactive かは明らかにされない. 各駒が active か inactive かどうかを覚えていれば単純な探索で最善手を見つけ出すことが可能である. また単純なヒューリスティックを使えばさらに効率的に探索可能であるし, 人間にとってみれば至極容易なゲームである.

8	旗	X	X	X	X				人
7		1	2	3	X	3	2	1	
6		1	2	3	X	3	2	1	
5		1	2	3	X	3	2	1	
4		1	2	3	X	3	2	1	
3		1	2	3	X	3	2	1	
2		1	2	3	X	3	2	1	
1	人				X	X	X	X	旗
	a	b	c	d	e	f	g	h	i

図 2.2: Finnsson と Björnsson の Shockstep Game の開始局面. 盤面左側 (a 列から d 列) は先手番プレイヤーの, 盤面右側 (f 列から i 列) は後手番プレイヤーのプレイ領域である. 旗はゴールを, 数字は罰金を, X は壁を表す

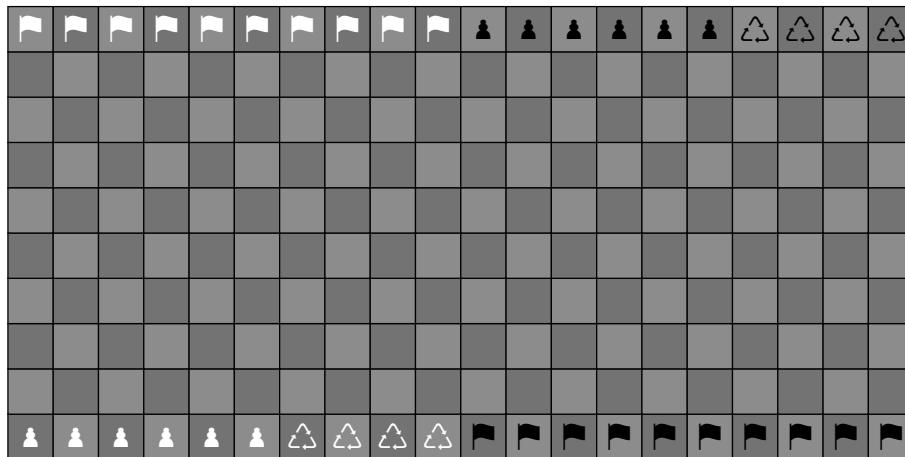


図 2.3: Finnsson と Björnsson の Progression Game の開始局面. ポーンはアクティブ駒, 円状矢印は非アクティブ駒, 旗はゴールを表す

しかしモンテカルロ木探索は後述する通りシミュレーションを元に局面の評価を行い, 最も基本的なアルゴリズムの場合ヒューリスティックを用いないため, このゲームにおける最善手を探索によって見つけ出すことは困難になりうる. なぜなら, シミュレーションを元にしてこのゲームの局面の評価を行う場合, 単一の駒を選択し続けることは確率的に少なく, よって active の駒を選び続けるという単純な最善手を見つけ出すことが困難であるためである.

2.2 ゲーム木探索

先程述べたとおりコンピュータがゲームをプレイする際に重要な要素の一つとして探索アルゴリズムが挙げられる. ここではゲームにおいて探索するとは現在の局面から自分にとって最良の局面へ遷移する行動を見つ

けることとする．この節では代表的なゲーム木探索アルゴリズムについて述べる．

2.2.1 Minimax 探索

二人ゲームにおいて最も基本的な探索アルゴリズムは minimax 探索である [27]．Minimax 探索では minimax 値と呼ばれる探索木の各節点に与えられる値を再帰的に計算して探索を行う．Minimax 値は葉節点・自分の手番の節点 (MAX ノード)・相手番の節点 (MIN ノード) のそれぞれで別々に定義される．葉節点ではその局面での利得 (ゲームの勝敗) もしくは評価関数によって得られたその局面での利得の見積もりが minimax 値となり，MAX ノードではその子節点の minimax 値の中で最大の値，MIN ノードでは子局面の minimax 値の中で最小の値とする．根節点の子節点のうち minimax 値が最大の節点へ遷移するような枝 (行動) を探索結果として出力する．

Minimax 探索では探索するゲーム木の深さを d ，各節点での分枝数を b としたとき，時間計算量が $O(b^d)$ となる．しかし探索する必要のない節点を枝刈りすることで時間計算量が改善する．代表的なアルゴリズムとして $\alpha\beta$ 枝刈り [28, 27] があり， $\alpha\beta$ 枝刈りをする minimax 探索を $\alpha\beta$ 探索と呼ぶこともある． $\alpha\beta$ 枝刈りを行うと最善の場合時間計算量が $O(b^{d/2})$ に改善する．

2.2.2 モンテカルロ木探索

モンテカルロ木探索 (Monte Carlo tree search, MCTS) [6, 22, 29] は局面評価をモンテカルロ・シミュレーションで行う探索手法であり，ある局面の評価をその局面からの多数のシミュレーション (乱数を用いてゲームを終局までプレイすること) の結果の平均によって近似する．これにより評価関数が必要なく，モデルが利用できる，すなわちシミュレーションが可能なゲームであれば適用が可能であるという利点がある．この利点により，評価関数の作成が困難だった囲碁においては minimax 探索に代わって標準的な手法となり，コンピュータプレイヤーの棋力は飛躍的に向上した．

モンテカルロシミュレーションによる局面 s における行動 a の評価 $Q(s, a)$ は

$$Q(s, a) = \frac{1}{N(s, a)} \sum_{i=1}^{N(s)} \mathbf{1}_i(s, a) z_i$$

で計算される．ここで $N(s, a)$ はこれまでに s から a を選択した回数， $N(s)$ は s からシミュレーションを行った回数， $\mathbf{1}_i(s, a)$ は i 回目のシミュレーションにおいて s から a を選択していれば 1，そうでなければ 0 となる指示関数である ($N(s, a) = \sum_{i=1}^{N(s)} \mathbf{1}_i(s, a)$)． i 回目のシミュレーションの報酬 z_i は，シミュレーションしたゲームの結果が勝利ならば 1，敗北であれば 0 とする．モンテカルロ木探索では，探索開始局面を根局面とした単一節点の探索木を構築し，その探索木に対して以下の手順を繰り返す行う：

1. 選択: 根局面から葉に到達するまで，局面をある選択基準 (tree policy) によって選択し探索木を下る．
2. 展開: 葉局面が展開する基準を満たしている場合，次の局面が探索木に加える．
3. シミュレーション: 葉局面から終端局面まで進行し，報酬を観測する．局面進行を完全にランダムに行うのではなく，特定の確率分布に従うことでより性能が上がることを示されている [30, 31, 32]．
4. 伝播: シミュレーションの結果をたどってきた局面に反映する．

探索のために割り当てられた計算資源が終了するまで以上の手順を繰り返す行い，終了時には根局面の子局面のうち最も良いものを次の着手として出力する．終了時の良さの基準としては訪れた回数が最大のもの

(robust child) を選ぶという基準が典型的であり [29], 本稿ではこの基準を採用する.

Tree policy として様々なものが提案されている [29]. 以下に代表的なものを記す.

UCT

UCT は tree policy として UCB1 [33] という基準を採用したモンテカルロ木探索である. UCB1 は後述するバンディット問題のアルゴリズムとして提案されている. UCT はある局面 s において子局面を選択する際に, それぞれの子局面へ遷移する行動 a について以下の式で与えられる UCB1 値を計算し, 最大の UCB1 値をもつ行動を次に選択する.

$$Q(s, a) + \sqrt{\frac{2 \ln N(s)}{N(s, a)}}$$

UCT はモンテカルロ木探索の反復回数を無限回行くと最善手へ収束するという性質をもつ [22].

Rapid Action Value Estimation

モンテカルロ木探索は探索木内のそれぞれの局面の評価をその局面からのそれぞれのシミュレーションによって行っている. そのため局面を正確に評価するためには多くのシミュレーションを行う必要があり, 結果としてあまり有望ではない局面にも多くのシミュレーションを割く必要がある. これは限られた資源のなかで探索を行う上では大きな障害となる. そこでシミュレーション回数が少ない段階でもある程度粗い見積もりを用いて局面の価値を推定することを目指したアルゴリズムが *Rapid Action Value Estimation* [34, 35] である. RAVE はもともと囲碁において提案され, 囲碁特有の「着手の順番が多少前後しても勝率に対するその着手の影響は小さい」という特徴を用いている. この特徴は *all-moves-as-first* ヒューリスティック (AMAF) と呼ばれる.

まず AMAF 価値関数 $\tilde{Q}(s, a)$ を以下のように定義する.

$$\tilde{Q}(s, a) = \mathbf{E}_{\pi} [z \mid s_t = s, \exists u \geq t \text{ s.t. } a_u = a]$$

つまり開始局面 s_t から終了までに行動 a を取ったときの報酬 z の期待値を表している. この AMAF 価値関数をモンテカルロシミュレーションによって推定すると,

$$\tilde{Q}(s, a) = \frac{1}{\tilde{N}(s, a)} \sum_{i=1}^{N(s)} \tilde{\mathbf{1}}_i(s, a) z_i$$

ここで $\tilde{\mathbf{1}}_i(s, a)$ は i 回目のシミュレーションにおいてシミュレーション中に行動 a を取ったとき 1, そうでないとき 0 となる指示関数である. $\tilde{N}(s, a)$ はシミュレーション中に行動 a を取った回数で, $\tilde{N}(s, a) = \sum_{i=1}^{N(s)} \tilde{\mathbf{1}}_i(s, a)$ を満たす. モンテカルロシミュレーションによって得られた行動価値関数を AMAF 価値関数によって置き換えたアルゴリズムが RAVE と呼ばれる [35].

文献 [35] では単純な RAVE 以外にも改良したアルゴリズムを提案している. MC-RAVE はモンテカルロシミュレーションによって推定された行動価値 $Q(s, a)$ と AMAF 価値関数とを組み合わせる行動価値を推定する.

$$Q_*(s, a) = (1 - \beta(s, a))Q(s, a) + \beta(s, a)\tilde{Q}(s, a)$$

$\beta(s, a)$ は重みパラメータで, AMAF による予測とモンテカルロシミュレーションによる予測のどちらを重視するかを決定する. またモンテカルロ木探索にて用いる場合には, ある局面 s を根局面とする部分木について AMAF 価値関数を計算する.

UCT-RAVE では UCT における選択の基準値に MC-RAVE で得られた評価値を用いる。

$$Q_*^\oplus(s, a) = Q_*(s, a) + c \sqrt{\frac{\ln N(s)}{N(s, a)}}$$

この値が最高の行動を各局面にて選択する。

重みパラメータ $\beta(s, a)$ は以下の式で定義される。

$$\beta(s, a) = \sqrt{\frac{k}{3N(s) + k}}$$

パラメータ k によってどの程度 RAVE の評価値を重視するかを決定し、訪問回数 $N(s)$ が k と等しいとき二つの項が等しく扱われる。またこの式の他にも AMAF の予測値の平均二乗誤差が最小になるように β を調整する方式も提案されている [35]。

2.3 ゲームにおける評価関数の学習

探索アルゴリズムと組み合わせ用いられる評価関数は、コンピュータプレイヤーの棋力に重要な影響を与える。評価関数は人間の知識をルールベースで実装したものや、線形結合による設計、ニューラルネットワークを用いたものなどが存在する。設計した評価関数の重みの調整は人手による調整から棋譜を用いた教師あり学習、近年では大規模な自己対戦による強化学習などによって行われている。以下に本研究に関連する評価関数の学習についての研究を記す。

2.3.1 AlphaGo

コンピュータ囲碁において評価関数を作成することは困難であった。チェスや将棋と比較して、用いる石自体には特徴がなく、駒得などを手がかりに評価することが困難で、人間が行っているような形やスジのパターン認識がコンピュータにとって難しいことが評価関数作成の障壁となっていた [36, 37]。そのため、長らくコンピュータ囲碁プレイヤーはモンテカルロ木探索を用いた評価手法により着手を決定していた [38]。

しかしながら囲碁の評価関数を作成する試みは続けられていた。古くは文献 [39] などが存在し、ニューラルネットワークを用いて評価関数を作成することを目指した。近年では計算機的能力向上と理論的進歩の成果として、深層ニューラルネットワークを用いることで棋力が向上している [40, 41, 40]。特筆すべき例として、AlphaGo [9] とその後継の AlphaGo Zero [10] が挙げられる。

AlphaGo は DeepMind が開発した囲碁プログラムであり、ニューラルネットワークによる局面価値関数とモンテカルロシミュレーションを組み合わせたモンテカルロ木探索によって次の着手を決定している。AlphaGo の最初のバージョンである AlphaGo Fan は 2015 年 10 月に行われた、当時の欧州王者である Fan Hui 二段との対局において五勝〇敗の成績を残している。また改良された AlphaGo Lee は 2016 年に行われた、当時最も強い棋士の一人である Lee Sedol 九段との対局で四勝一敗の成績を収めた。その後も改良が続けられ、AlphaGo Master はオンライン対局場にて人間のプロ棋士相手に 60 連勝を果たし^{*3}、2017 年に行われた Future of Go Summit では Ke Jie 九段に勝利した^{*4}。この節では文献 [9] で述べられた AlphaGo

^{*3} <https://deepmind.com/research/alphago/match-archive/master/> Accessed on 2018/12/10.

^{*4} <https://deepmind.com/research/alphago/alphago-matches-from-the-future-of-go-summit/> Accessed on 2018/12/10.

Fan を AlphaGo と呼ぶ.

その後も DeepMind は研究を続け、自己対戦による強化学習により学習を行う AlphaGo Zero は人間の棋譜を用いることなく AlphaGo Master の棋力を超えた. また AlphaGo Zero と同様の学習アルゴリズムがチェスと将棋においても有効であることを示した [11].

評価関数の学習

AlphaGo の評価関数は方策関数 $\mathbf{p}_\sigma(s)$, $\mathbf{p}_\rho(s)$ と局面価値関数 $v_\theta(s)$ の三種類存在する. ここで σ, ρ および θ はパラメータを表す. それぞれの関数は独立したニューラルネットワークとして実装されており, 三段階に分けて独立に学習される. まずはじめに棋譜から方策関数 $\mathbf{p}_\sigma(s)$ の学習を行う. 学習は棋譜中に選択された行動の対数尤度を最大化することで行われる. 次に方策関数 $\mathbf{p}_\rho(s)$ を強化学習によって学習する. ρ は σ によって初期化され, その後 \mathbf{p}_ρ 同士で自己対戦を行い, 基準値を用いた REINFORCE アルゴリズム [42] によって方策を更新する. その後 $\mathbf{p}_\sigma, \mathbf{p}_\rho$ の両方を用いた自己対戦により局面価値関数 $v_\theta(s)$ を平均二乗誤差を最小化することにより学習する. この学習は非常に大規模であり, 既存の棋譜は 3000 万局面, 自己対戦も同程度の数を実行している. 対局時には学習した \mathbf{p}_σ と v_θ を用いたモンテカルロ木探索により次の着手を決定している.

AlphaGo が二つのニューラルネットワークを独立に保持し, 学習が三段階となっていたのに対し, AlphaGo Zero は単一のニューラルネットワークのみを持っている. AlphaGo Zero のニューラルネットワークは単一の局面 s を入力として受け取ると $\langle \mathbf{p}(s), v(s) \rangle$ の二つの値を出力する. ここで $\mathbf{p}(s)$ は局面 s に対する着手予測を意味するベクトルで, $v(s)$ がその局面の価値を表すスカラー値である. AlphaGo Zero のニューラルネットワークは二つの出力を持ち, 一つが \mathbf{p} を, もう一つが v を出力する. この構造により正則化効果をもたらし, 学習が安定すると報告されている. またこのニューラルネットワークの学習にはこのニューラルネットワークを用いた自己対戦により棋譜を生成し, その棋譜を用いてニューラルネットワークの学習を行う. 自己対戦はランダムシミュレーションを使わない \mathbf{p} と v のみを用いたモンテカルロ木探索によって着手を選択しており, 自己対戦を数千万回行い学習を行うことで AlphaGo の棋力を超えた. 学習は方策予測と局面価値関数それぞれについて損失関数 L_p, L_v を計算し, 二つの損失関数の和 $L = L_p + L_v$ を最小化することで行われる.

$$L_v(s, z) = (v(s) - z)^2, \quad (2.1)$$

$$L_p(s, \pi) = -\pi^\top \log \mathbf{p}(s), \quad (2.2)$$

$$L(s, \pi, z) = L_v + L_p \quad (2.3)$$

ここで π はモンテカルロ木探索によって得られた訪問回数に比例する行動確率ベクトルであり, ニューラルネットワークが予測する着手予測はモンテカルロ木探索の結果に近づくよう学習が行われる. 実際の対局時にも自己対戦のときとほとんど同様のモンテカルロ木探索によって着手を決定する.

評価関数への入力

評価関数を計算するためには評価する局面を何らかの形で計算機が扱えるように符号化し, 評価関数へ入力する. このとき様々な手法が存在し, 局面を単純に符号化する場合もあれば人間が有用であると判断した特徴を同時に符号化する場合も多い. AlphaGo では局面の符号化の際に石の配置だけでなく, ある座標に着手できるかどうかやある石が何手前から存在していたかなどの人間の知識を利用して符号化を行っている. 一方 AlphaGo Zero ではそのような人間の事前知識をニューラルネットワークの入力として符号化しておらず, 黒

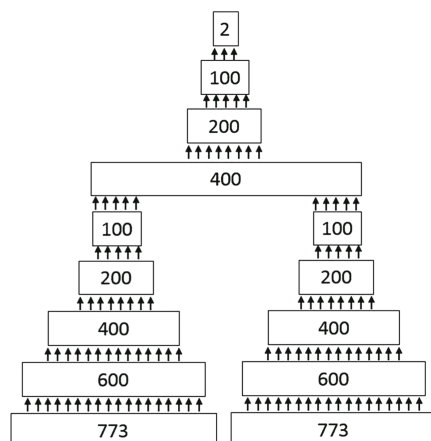


図 2.4: DeepChess のネットワーク構成 [43]

石・白石の配置とその履歴，現在の手番のみを符号化してニューラルネットワークの入力としている。

2.3.2 DeepChess

ゲームの評価関数を局面の組合せを用いて行った先行研究として DeepChess [43] がある。DeepChess はコンピュータチェスプログラムであり，本稿ではそれが用いている深層ニューラルネットワークを指す。DeepChess は入力として局面对を受け取り，そのどちらがより好ましい局面かを二項分類問題として出力する。入力を二つ受け取るため，DeepChess は図 2.4 のような二叉のネットワーク (Siamese network) である。図中の長方形は全結合層を表しており，書かれている数字の数だけ入力を受け付ける。

DeepChess では重みの初期値を事前に自己符号化器 (autoencoder) で調整してから学習を行っている。図 2.4 中の $773 \rightarrow 600 \rightarrow 400 \rightarrow 200 \rightarrow 100$ の部分ネットワークを積層自己符号化器 (stacked autoencoder) として学習している。

自己符号化器によって重みの初期値を得た後，学習を次のとおりに実行する。まず，用いる棋譜に含まれる局面を先手番から見た勝ち局面集合 W と 負け局面集合 L に分割する。その後 W, L から一局面ずつ w, l を乱択抽出し，順番をランダムに入れ替えてネットワークに入力する。つまり確率 0.5 で (w, l) ，0.5 で (l, w) の順番でそれぞれの下部ネットワークに入力する。ネットワークは勝ち局面がどちらであるかを正しく判定するように重みを更新する。

DeepChess の学習では $|W|, |L|$ とともに 百万程度であり，手番の対称性なども考慮した場合，組合せを考えただけにはおよそ 2×10^{12} 通り程度作成することができる。実験ではテスト集合における予測精度は 98% 程度であると報告されている。対戦においては学習した二叉のネットワークを使った alphabeta 探索により局面を評価している。 α, β を値ではなく局面として保持し，探索して訪れた新たな局面と， α, β それぞれとを学習したニューラルネットワークで比較する。比較した結果が α より良ければ alpha-cut， β より悪ければ beta-cut が行われる。このアルゴリズムを実装した DeepChess はグランドマスターレベルのチェスプログラムである FALCON ^{*5} と同程度の棋力であると報告されている。

^{*5} <https://www.chessprogramming.org/Falcon>

2.3.3 Discriminator を利用した学習

深層ニューラルネットワークを利用した興味深い応用例として GAN (Generative Adversarial Networks) [44] がある。GAN は generator と discriminator と呼ばれる二つのニューラルネットワークを同時に学習する。Generator の目的は「本物」のデータと見分けがつかないようデータを出力することであり、discriminator の目的は与えられた入力「本物」かどうかを判定することである。例えば対象とするデータが猫の画像の集合だとすると、generator は入力のノイズから discriminator を騙すような猫の画像を出力することが目的であり、反対に discriminator の目的は入力された画像がもとの猫の画像の集合に属しているのか generator が生成したものかどうかを見分けることが目的となる。Generator と discriminator に十分表現力があれば、generator が生成するデータの分布はもとのデータの分布に収束することが示されている [44]

ゲームにおける GAN の応用例として、STGAN (Style Transfer Generative Adversarial Networks) というものが存在する [45]。これはゲームのプレイスタイルを生成することを目指した研究で、対象となるプレイヤーの棋譜からそのプレイスタイルを模倣する生成機を学習することを目的としている。

Discriminator を用いた評価関数

ゲームの評価関数の学習に GAN の枠組みを応用した研究も存在する [46, 15]。ここでは 文献 [15] の学習を説明する。この学習では棋譜に含まれる局面とランダムに生成された局面とを見分ける discriminator を導入し、それを訓練する。学習には次の三つ組 $\langle s, s', s_r \rangle$ を用いる。 s は棋譜に含まれるある局面、 s' は棋譜中に実際に s から取られた行動 a を経た後局面 s' 、 s_r は s における合法手集合のうちランダムに選択された行動によって遷移した局面 s_r である。この三つの局面から二つの組 $\langle s, s' \rangle$ と $\langle s, s_r \rangle$ を見分けられるように discriminator は訓練される。損失関数は以下のように定義されている。

$$L_D(s, s', s_r) = (d(s, s') - d(s, s_r) - 2)^2 \quad (2.4)$$

つまり $d(s, s')$ を $+1$ に、 $d(s, s_r)$ を -1 に近づけるように学習を行う。一般的な最小二乗誤差ではなくこのような定式化になっているのは、この式のほうが学習が高速に進んだためである *6。

また discriminator の学習の際に、学習時のミニバッチについて制約を加える uniformity loss という損失も提案されている [15]。これは学習における n 組のデータを含むミニバッチ $\mathcal{D} = \{\langle s^{(i)}, s'^{(i)}, s_r^{(i)} \rangle\}_{i=1}^n$ について、 discriminator の出力結果をできるだけ均一に保つように設計された損失である。ミニバッチ \mathcal{D} に対する uniformity loss は次の式

$$L_{UR}(\mathcal{D}) = \frac{1}{n/2} \sum_{i=1}^{n/2} [(d(s^{(i)}, s'^{(i)}) - d(s^{(i+n/2)}, s'^{(i+n/2)}))^2 + \quad (2.5)$$

$$(d(s^{(i)}, s_r^{(i)}) - d(s^{(i+n/2)}, s_r^{(i+n/2)}))^2] \quad (2.6)$$

で定義される。ミニバッチが棋譜中の局面をランダムに抽出したものとすると、この式は discriminator の出力結果の分散を最小化しようとしているとみなせる。Uniformity loss を加えることで学習の精度が向上したとされている [15]。

*6 文献 [15] の著者に確認した。

文献 [15] ではこの discriminator と同時に局面価値関数を学習し、より精度の高い学習結果が得られている。

2.4 バンディット問題とそのアルゴリズム

この節では古典的な強化学習の題材として研究されてきたバンディット問題 (bandit problem) について述べる。バンディット問題には様々な応用先が存在する重要な問題であるが、ゲーム木探索においては前述のモンテカルロ木探索において理論的に重要な問題となっている。

バンディット問題とは行動集合のなかから一つの行動を選び、その他の行動に関する情報は得られないが選択した行動に対する報酬を得るという試行を繰り返すという問題設定において報酬の累積和の最大化を目的とする逐次決定問題である。バンディットとは盗賊の意味で、一つのアームをもつ古典的なスロットマシンをプレイヤーに金銭を使わせる（金銭を奪う）ということから one-armed bandit と呼ぶことに由来している。

バンディット問題は確率的バンディット問題と敵対的バンディット問題の二つに大きく分類される。確率的バンディット問題は行動を選択した際に得られる報酬が確率的に決定される問題で、敵対的バンディット問題は事前にプレイヤーの方策を知っている敵対者が報酬を選択するというより性質の悪い問題を扱っている。本稿では敵対的バンディット問題は取り上げず、確率的バンディット問題についてのみ述べる。

確率的バンディット問題には様々な派生問題が存在する。代表的な問題は、 K 台のスロットマシンから 1 台のスロットマシンを選択することを繰り返し、その最終的な報酬を最大化することを目指すもので、多腕バンディット問題 (multi-armed bandit problem) や K -腕バンディット問題 (K -armed bandit problem) と呼ばれる。またより現実に近い問題をモデル化したものとして線形バンディット問題 (linear bandit problem) や文脈付きバンディット問題 (contextual bandit problem) と呼ばれるものがある。本節ではこれらについて述べる。

2.4.1 多腕バンディット問題

この節では代表的な確率的バンディット問題の一種である多腕バンディット問題について述べる。多腕バンディット問題ではそれぞれの行動の報酬が確率的に決定される。プレイヤーは各時刻 t において行動集合 \mathcal{A} の中から一つ行動を選択する。それぞれの行動はプレイヤーには明らかにされないそれぞれ独立の報酬の確率分布をもっており、選択した行動に対応する確率分布に基づきプレイヤーは報酬を受け取る。プレイヤーの目的は受け取る報酬の総和を最大化することである。ここでプレイヤーが受け取る報酬は有界であるとする。報酬の総和を最大化するという点で、1) 有限時間区間 (finite horizon) における累積報酬和、2) 無限時間区間における幾何割引累積報酬和の二つが主に研究をされているが、ここでは有限時間区間における累積報酬和について考える。つまり、あらかじめ与えられる試行回数 T についての次の累積報酬和

$$\sum_{t=1}^T X_{i(t)}(t)$$

を最大化することを目的とする。ここで $i(t)$ はプレイヤーが時刻 t に選択した行動、 $X_{i(t)}(t)$ は時刻 t においてプレイヤーが受け取った報酬である。

確率的バンディット問題において、もしプレイヤーに対して報酬の確率分布が既知であれば、報酬の総和を最大化する戦略は確率分布の期待値が最大の行動 i^* を常に選択するというものになる。このとき得られる報酬の総和の期待値は、行動 i^* の報酬の期待値を μ_{i^*} としたとき $T\mu_{i^*}$ となる。この値を基準として、あるプ

レイヤーの方策の性能を議論するために regret $R(T)$ と呼ばれる値を定義する.

$$R(T) = \sum_{t=1}^T (\mu_{i^*} - \mu_{i(t)}) = T\mu_{i^*} - \sum_{t=1}^T \mu_{i(t)}$$

この値が小さければ小さいほどよい方策であると評価される. またこの regret を T で除した average regret と呼ばれる指標も用いられる.

$$R(T)/T = \mu_{i^*} - \frac{1}{T} \sum_{t=1}^T \mu_{i(t)}$$

UCB1

UCB1 [33] は確率的バンディット問題では広く用いられている方策の一つで, 各時刻 t において以下の値が最大の行動を選択する:

$$\bar{X}_{i, T_i(t)} + \sqrt{\frac{2 \ln t}{T_i(t)}},$$

ここで $T_i(t)$ は時刻 t までに行動 i を選択した回数, $\bar{X}_{i, T_i(t)}$ はこれまでの $T_i(t)$ 回の行動 i の試行で得られた報酬の平均である. UCB1 を用いることで期待 regret を $O(\log t)$ に抑えることが可能である [33].

2.2.2 項 で述べたとおり, UCB1 をモンテカルロ木探索に応用した手法が UCT であり, コンピュータ囲碁プログラムで広く用いられていた [38].

PUCB

PUCB [47] は UCB1 と同じく確率的バンディット問題のアルゴリズムであるが, episode context $\mathbf{M} \in \mathbb{R}^{|\mathcal{A}|}$ と呼ばれる付随的な情報が与えられているときに用いられるアルゴリズムである. \mathbf{M} の要素 M_i は行動 a_i に割り当てられる重みであり, $\sum_i M_i = 1$ を満たす. 典型的には \mathbf{M} は事前知識を表しており, 良さそうな行動を優先的に試すという戦略を定式化したものである. 具体的には以下の式で計算される値が最大の行動を次に選択する

$$\bar{X}_{i, T_i(t)} + \sqrt{\frac{3 \log t}{2 T_i(t)}} - \frac{2}{M_i} \sqrt{\frac{\log t}{t}}$$

最善行動への重みを M_* としたとき, PUCB が達成する最悪 regret の上界は $O(\frac{1}{M_*} \sqrt{t \log t})$ となる [47]. PUCB をモンテカルロ木探索に応用した PUCT の変種が AlphaGo, AlphaGo Zero, および AlphaZero において用いられている [9, 10, 11].

2.4.2 線形バンディット問題

確率的バンディット問題の興味深い応用例として線形バンディット問題 (linear bandit problem) が挙げられる [48]. 線形バンディット問題はそれぞれの行動に特徴ベクトル $\mathbf{x}_i \in \mathbb{R}^d$ が存在し, 行動の報酬がその特徴ベクトルの一次結合に従って決定される, バンディット問題の一種である. つまり時刻 t における各行動 $i \in \mathcal{A}$ の報酬 $r_{t,i}$ が,

$$r_{t,i} = \mathbf{x}_i^\top \boldsymbol{\theta}^* + \epsilon(t)$$

で表される. ここで $\boldsymbol{\theta}^*$ は未知の係数ベクトルであり, これによって報酬が決定される. $\epsilon(t)$ は期待値 0 の誤差項を表す. 先程の確率的バンディット問題は $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{|\mathcal{A}|})$ が単位行列であるときに対応する.

■文脈付きバンディット問題 線形バンディット問題では各行動の特徴ベクトルが時間によらず一定であるという仮定をしていたが、これが時間によって変化するという点を許す設定が文脈付きバンディット問題 (contextual bandit problem) である [48]。文脈付きバンディット問題では各行動に時間に依存する特徴ベクトル $\mathbf{x}_{t,i}$ が存在し、その特徴ベクトルと $\boldsymbol{\theta}^*$ とを用いて報酬の期待値を

$$\mathbf{E}[r_{t,i} | \mathbf{x}_{t,i}] = \mathbf{x}_{t,i}^\top \boldsymbol{\theta}^* \quad (2.7)$$

として仮定した問題である。

特徴ベクトル $\mathbf{x}_{t,i}$ を時間に依存しないベクトル \mathbf{y}_i と時間に依存するベクトル $\mathbf{z}_{t,i}$ との結合で表すことができる場合を考える。つまり、 $\mathbf{x}_{t,i} = (\mathbf{y}_i^\top, \mathbf{z}_{t,i}^\top)^\top$ と表すことができる場合である。この場合において、時間に依存しないベクトルに対応する係数ベクトルを $\boldsymbol{\theta}^{(y*)}$ 、依存するベクトルに対応する係数ベクトルを $\boldsymbol{\theta}^{(z*)}$ としたとき、式 (2.7) は

$$\mathbf{E}[r_{t,i} | \mathbf{y}_i, \mathbf{z}_{t,i}] = \mathbf{y}_i^\top \boldsymbol{\theta}^{(y*)} + \mathbf{z}_{t,i}^\top \boldsymbol{\theta}^{(z*)}$$

となる。このとき $\boldsymbol{\theta}^* = (\boldsymbol{\theta}^{(y*)\top}, \boldsymbol{\theta}^{(z*)\top})^\top$ である。この仮定では、行動 i の時間に依存する特徴ベクトル $\mathbf{z}_{t,i}$ が文脈 (context) と呼ばれる。また上の式では行動と文脈が相互に作用しないという仮定も表している。

例えばニュース記事配信サイトにおいて次に表示する記事の選択問題は文脈付きバンディット問題の応用例の一つであり、文献 [49] で詳しく述べられている。これはある時刻におけるサイト訪問者に対してどのニュース記事を提示すればクリック率を最大化できるかという問題である。この問題では文脈がサイト訪問者の属性に相当し、文脈によって行動の価値が変化することをモデル化している。つまりニュース記事の特徴とともにサイト訪問者の属性によってもクリック率は変化するということをモデル化している。上述の行動と文脈が相互作用しないという仮定では、ニュース記事自体の固有のクリック率とサイト訪問者の属性によって決定されるクリック率との和がニュースを提示したときのクリック率となるとモデル化している。このほか、自然な仮定としてサイト訪問者の属性によってニュース記事のクリックされやすさが異なることをモデル化するような、行動と文脈が相互に作用するといった仮定も文献 [49] で述べられている。

LinUCB

LinUCB [50, 49] は上で述べた文脈付きバンディット問題のアルゴリズムである。ゲームへの応用として、LinUCB とモンテカルロシミュレーションを組み合わせたアルゴリズムの評価を麻雀において行なった研究が存在する [51]。文献 [50, 48] と文献 [49] で異なる定式化をしているが、ここでは前者の定式化でアルゴリズムを説明する。以降、行動 i の時刻 t における報酬が $r_{t,i} \in [0, 1]$ であり、特徴ベクトルが $\|\mathbf{x}_{t,i}\|_2 \leq 1$ であるとする。このとき $\|\boldsymbol{\theta}^*\|_2 \leq 1$ である。また選択可能な行動集合は時間によらず一定で、 $|\mathcal{A}| = K$ とする。また行動と文脈の相互作用は考えない。

LinUCB は $\boldsymbol{\theta}^*$ を推定し、各行動の報酬の予測値を計算する。このとき、UCB 方策と同様に報酬を楽観的に予測し、期待値が最大に見える行動を選択する。 $\boldsymbol{\theta}^*$ は標準的な線形回帰により、

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \arg \min_{\boldsymbol{\theta}'} \sum_{s=1}^t (r_{s,i_s} - \mathbf{x}_{s,i_s}^\top \boldsymbol{\theta}') \\ &= \left(\sum_{s=1}^t \mathbf{x}_{s,i_s} \mathbf{x}_{s,i_s}^\top \right)^{-1} \sum_{s=1}^t r_{s,i_s} \mathbf{x}_{s,i_s} \end{aligned}$$

と推定できる． i_s は時刻 s において選択した行動を表す．ここで，

$$\mathbf{A}_t = \sum_{s=1}^t \mathbf{x}_{s,i_s} \mathbf{x}_{s,i_s}^\top$$

とおき，また 式 (2.7) より $r_{t,i} = \mathbf{x}_{t,i}^\top \boldsymbol{\theta}^* + \epsilon(t)$ なので

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \mathbf{A}_t^{-1} \sum_{s=1}^t \mathbf{x}_{s,i_s} (\mathbf{x}_{s,i_s}^\top \boldsymbol{\theta}^* + \epsilon(s)) \\ &= \mathbf{A}_t^{-1} \sum_{s=1}^t \mathbf{x}_{s,i_s} \mathbf{x}_{s,i_s}^\top \boldsymbol{\theta}^* + \mathbf{A}_t^{-1} \sum_{s=1}^t \mathbf{x}_{s,i_s} \epsilon(s) \\ &= \boldsymbol{\theta}^* + \mathbf{A}_t^{-1} \sum_{s=1}^t \mathbf{x}_{s,i_s} \epsilon(s) \end{aligned}$$

となって，推定値 $\hat{\boldsymbol{\theta}}$ は期待値 $\boldsymbol{\theta}^*$ をもつ．また $\epsilon(t)$ がそれぞれ独立で分散 σ^2 をもつと仮定すると，推定値 $\hat{\boldsymbol{\theta}}$ の分散は $\mathbf{Var}[\hat{\boldsymbol{\theta}}] = \sigma^2 \mathbf{A}_t^{-1}$ になる．よって報酬 $r_{t,i}$ の点推定量 $\hat{r}_{t,i} = \mathbf{x}_{t,i}^\top \hat{\boldsymbol{\theta}}$ の期待値と分散は，

$$\begin{aligned} \mathbf{E}[\hat{r}_{t,i}] &= \mathbf{x}_{t,i}^\top \boldsymbol{\theta}^* = r_{t,i} \\ \mathbf{Var}[\hat{r}_{t,i}] &= \mathbf{E} \left[\left(\mathbf{x}_{t,i}^\top (\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*) \right)^\top \left(\mathbf{x}_{t,i}^\top (\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}^*) \right) \right] \\ &= \mathbf{x}_{t,i}^\top \mathbf{Var}[\hat{\boldsymbol{\theta}}] \mathbf{x}_{t,i} \\ &= \sigma^2 \mathbf{x}_{t,i}^\top \mathbf{A}_t^{-1} \mathbf{x}_{t,i} \end{aligned}$$

となる．よって報酬の楽観的な推定値を定数 α を用いて，

$$\mathbf{x}_{t,i}^\top \hat{\boldsymbol{\theta}} + \alpha \sqrt{\mathbf{x}_{t,i}^\top \mathbf{A}_t^{-1} \mathbf{x}_{t,i}} \quad (2.8)$$

と計算し，これを評価値として最大の行動を各時刻に選択するアルゴリズムである．LinUCB のアルゴリズムを アルゴリズム 1 に示す．ここで， $\mathbf{A} \leftarrow \lambda \mathbf{I}_d$ と初期化しているのは \mathbf{A} が最大階数をもつように，つまり逆行列が存在するようにするためであり，ridge 回帰と同等になる．特徴ベクトルの次元を d としたとき，LinUCB によって regret を $\tilde{O}(\sqrt{dT})$ に抑えることが可能であると示されている [50]． \mathbf{A} の逆行列の計算は Sherman-Morrison の公式を用いると，

$$(\mathbf{A} + \mathbf{x} \mathbf{x}^\top)^{-1} = \mathbf{A}^{-1} - \frac{(\mathbf{A}^{-1} \mathbf{x})(\mathbf{x}^\top \mathbf{A}^{-1})}{1 + \mathbf{x}^\top \mathbf{A}^{-1} \mathbf{x}}$$

となり，逆行列のみを保持しておけば良く，一試行あたりの計算量は $O(d^2)$ となる．

ここではすべての行動に共通の係数ベクトルが存在すると仮定していたが，文献 [49] では各行動それぞれに独立の係数ベクトル $\boldsymbol{\theta}_i^*$ が存在すると仮定している．つまり，アルゴリズム 1 中の変数 \mathbf{A} , \mathbf{b} は行動ごとに存在する．この場合，各行動の報酬の推定値は他の行動の観測報酬に依存しないためより詳細に誤差を評価することができ， $\alpha = 1 + \sqrt{\ln(2/\delta)/2}$ としたとき確率 $1 - \delta$ で

$$|\mathbf{x}_{t,i}^\top \hat{\boldsymbol{\theta}}_i - \mathbf{x}_{t,i}^\top \boldsymbol{\theta}_i^*| \leq \alpha \sqrt{\mathbf{x}_{t,i}^\top \mathbf{A}_i \mathbf{x}_{t,i}} \quad (2.9)$$

の不等式を満たす [50, 49]．

アルゴリズム 1 LinUCB

Inputs: $\alpha \in \mathbb{R}_+$, $\lambda \in \mathbb{R}_+$

$\mathbf{A} \leftarrow \lambda \mathbf{I}_d$

$\triangleright d$ dimensional identity matrix

$\mathbf{b} \leftarrow \mathbf{0}_d$

$\triangleright d$ dimensional zero vector

for $t = 1, 2, 3, \dots$ **do**

$\hat{\boldsymbol{\theta}} \leftarrow \mathbf{A}^{-1} \mathbf{b}$

for all $a \in \mathcal{A}$ **do**

$\triangleright \mathcal{A}$ is a set of available actions

 Observe feature $\mathbf{x}_{t,a}$

$p_a \leftarrow \mathbf{x}_{t,a}^\top \hat{\boldsymbol{\theta}} + \alpha \sqrt{\mathbf{x}_{t,a}^\top \mathbf{A}^{-1} \mathbf{x}_{t,a}}$

end for

$a_t \leftarrow \arg \max_{a \in \mathcal{A}} p_a$ with ties broken arbitrarily

 Observe payoff $r_t \in \{0, 1\}$

$\mathbf{A} \leftarrow \mathbf{A} + \mathbf{x}_{t,a_t} \mathbf{x}_{t,a_t}^\top$

$\mathbf{b} \leftarrow \mathbf{b} + r_t \mathbf{x}_{t,a_t}$

end for

fLinUCB-GD

LinUCB では一試行あたり $O(d^2)$ の計算量がかかり，そのため次元数が大きい場合には多くの計算時間が必要となる．そこで，この計算を確率的勾配降下法で行う fLinUCB-GD と呼ばれる手法が提案されている [52]．fLinUCB-GD では $\hat{\boldsymbol{\theta}}$ を L_2 正則化を行う確率的勾配降下法により近似する．文献 [52] ではこの正則化付きの確率的勾配降下法を fRLS-GD (fast Regularised online Least Squares - Gradient Descent) と呼んでいる．この節でもすべての行動に共通の係数ベクトル $\boldsymbol{\theta}^*$ が存在すると仮定する．

fLinUCB-GD では，時刻 t においてこれまでに観測したデータを用いて予測を更新する．観測した特徴ベクトルと報酬の対の集合 $\{\langle \mathbf{x}^{(1)}, y^{(1)} \rangle, \dots, \langle \mathbf{x}^{(t-1)}, y^{(t-1)} \rangle\}$ の中からランダムに一つ選択し，以下の更新式によって現在の $\hat{\boldsymbol{\theta}}$ の予測を更新する．

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \gamma_t \left((y^{(i_t)} - \mathbf{x}^{(i_t)\top} \boldsymbol{\theta}_{t-1}) \mathbf{x}^{(i_t)} - \lambda_t \boldsymbol{\theta}_{t-1} \right). \quad (2.10)$$

ここで i_t はランダムな添字で， $i_t \sim \mathcal{U}\{1, t-1\}$ とする ($\mathcal{U}\{a, b\}$ は a から b までの要素を取りうる離散一様分布)． γ_t は学習率で， $\sum_t \gamma_t = \infty, \sum_t \gamma_t^2 \leq \infty$ となるように設定する． λ_t は ridge 回帰における拘束条件の強さを表し， λ_t が大きいほど正則化効果が強い．

式 (2.8) の第二項にも逆行列の計算が必要となるが， $\mathbf{A}^{-1} \mathbf{x}_{t,i}$ を以下のような $\boldsymbol{\phi}_{t,i}$ で近似することで，逆行列の計算をせずに LinUCB 値の計算を行うことができる．

$$\boldsymbol{\phi}_{t,i} = \boldsymbol{\phi}_{t-1,i} + \gamma_t ((\mathbf{x}_{t,i}/t - (\mathbf{x}^{(i_t)\top} \boldsymbol{\phi}_{t-1,i}) \mathbf{x}^{(i_t)}) \quad (2.11)$$

fLinUCB-GD でも LinUCB と同様，以下の式

$$\mathbf{x}_{t,i}^\top \boldsymbol{\theta}_t + \frac{\kappa}{t} \sqrt{\mathbf{x}_{t,i}^\top \boldsymbol{\phi}_{t,i}}$$

を用いて各行動の LinUCB 値を計算し，最大の行動を選択して報酬を観測する．その後特徴ベクトルと得られた報酬の対を保存しておき，次回以降の試行での計算で用いる．これらの式の計算量は $O(d)$ であるため，

アルゴリズム 2 fLinUCB-GD

```
Inputs:  $\kappa$ ,  $\gamma_t$ , and  $\hat{\boldsymbol{\theta}}_0$ 
for  $t = 1, 2, 3, \dots$  do
  Approximate  $\hat{\boldsymbol{\theta}}_t$  using (2.10)
  for  $a \in \mathcal{A}$  do
    Observe feature  $\mathbf{x}_{t,a}$ 
    Estimate confidence parameter  $\boldsymbol{\phi}_{t,a}$  using (2.11)
     $p_a \leftarrow \mathbf{x}_{t,a}^\top \hat{\boldsymbol{\theta}}_t + \frac{\kappa}{t} \sqrt{\mathbf{x}_{t,a}^\top \boldsymbol{\phi}_{t,a}}$ 
  end for
   $a_t \leftarrow \arg \max_{a \in \mathcal{A}} p_a$  with ties broken arbitrarily
  Observe payoff  $r_t$ 
end for
```

逆行列の計算を必要とする通常の LinUCB よりも高速に行うことが可能である。なお、文献 [49] では \mathbf{A} を $\sum_i \mathbf{x}_i^\top \mathbf{x}_i$ と定義しているのに対し、文献 [52] では $\mathbf{A} = t^{-1} \sum_i \mathbf{x}_i^\top \mathbf{x}_i$ と定義している。そのため、文献 [49] と同じスケールとするために本稿では $\boldsymbol{\phi}_{t,a}$ に t^{-1} を乗じている。

これまで説明した fLinUCB-GD のアルゴリズムを アルゴリズム 2 に示す。

2.5 その他の機械学習手法

この節では本研究に関連するその他の機械学習のアルゴリズムについて述べる。

2.5.1 マルチタスク学習

マルチタスク学習 (multitask learning) は様々な意味で使われる用語であるが、ここでは単一の入力から複数の目的変数を予測する学習の方法とする。ニューラルネットワークの研究では古くから提案されている手法 [14] であり、汎化性能を向上させることができるとされている。

図 2.5 はマルチタスク学習で典型的に用いられるニューラルネットワークの構造を表している。学習では単一の入力 \mathbf{x} から二つの目的変数 $\mathbf{y}^{(1)}$, $\mathbf{y}^{(2)}$ を予測する。この学習方法がうまくいくためにはこの二つの学習目的が統計的に関連している必要がある。そのような場合では、共有層 $\mathbf{h}^{(s)}$ が二つのタスク間に共通する因子をうまく抽出することができると期待される [14, 53]。またタスク間で学習の難易度に差があるときには、簡単なタスクから情報を得て困難なタスクに役立てることができる (eavesdropping)。さらに、各タスク間に複数の局所解が存在したとしても、タスク間で共通の局所解が選ばれやすくなる性質がある。

AlphaGo Zero で採用された、局面価値と着手予測を同時に学習するニューラルネットワークはマルチタスク学習の典型である [10]。

2.5.2 RankNet

RankNet [54] は learning-to-rank 学習の手法の一つである。Learning-to-rank 学習は入力されたクエリに対してそれぞれのクエリの好ましさを出力するような学習器を構築することが目的である。例えば検索エンジ

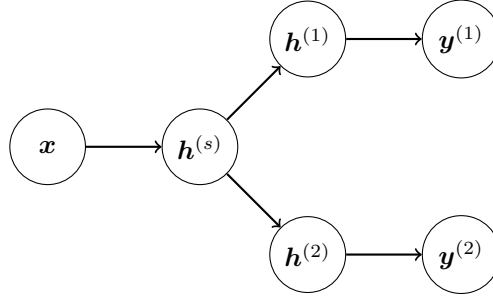


図 2.5: マルチタスク学習の例 ([53] を参考に作図). 入力 x は共有されている中間層 $h^{(s)}$ を通り, 別々の中間層 $h^{(1)}, h^{(2)}$ を経由してからそれぞれの予測値 $y^{(1)}, y^{(2)}$ を出力する.

ンにおいて, 入力されたクエリに関連する文書を表示することを考える. このとき, 文書を表示する際にできるだけ入力されたクエリに関連する順に文書を表示させたい. クエリを q , 文書を d とし, クエリと文書の特徴を $\phi(q, d)$ としたとき, 何らかのスコア関数 $f(\phi(q, d))$ を用いてスコア順に並べるというふうに定式化できる. 学習の手法によってランキング学習を分類することができる. 学習の際に単一の入力を用いるものを pointwise, 二つの入力の場合 pairwise, そしてリストが入力される場合 listwise と呼ばれる.

RankNet はランキング学習にニューラルネットワークを用いた手法で, pairwise なランキング学習に分類される. つまり, 入力 U_i, U_j が与えられたとき, その特徴ベクトル x_i, x_j を用いて学習を行う. 上の例でいうと $x_i = \phi(q, d_i)$, $x_j = \phi(q, d_j)$ である. 学習は特徴ベクトルを実数値に写像する微分可能な関数 f_θ を最適化することで行われる. RankNet の名が表すとおり, 関数としてニューラルネットワークが用いられているが, 他のも (線形結合など) でも可能であると思われる. 簡単のためそれぞれの入力結果を $s_i = f_\theta(x_i)$, $s_j = f_\theta(x_j)$ と表記する.

入力 U_i, U_j について, U_i が U_j よりも好ましい (検索エンジンの例でいうと U_i が U_j よりも先に表示されるべき) という事象を $U_i \triangleright U_j$ と表記する. このとき, この事象の確率をシグモイド関数を用いて

$$\Pr(U_i \triangleright U_j) = P_{ij} \quad (2.12)$$

$$= \frac{1}{1 + e^{-\sigma(s_i - s_j)}} \quad (2.13)$$

$$= \frac{e^{\sigma s_{ij}}}{1 + e^{\sigma s_{ij}}} \quad (2.14)$$

と仮定する. ここで $s_{ij} = s_i - s_j$ とし, σ はパラメータである. また U_i と U_j , U_j と U_k のスコア差 s_{ij} , s_{jk} が既知の場合,

$$s_{ij} + s_{jk} = (s_i - s_j) + (s_j - s_k) = s_{ik} \quad (2.15)$$

と s_{ik} が求まる. この定式化により以下の定理 [54] が成り立つ.

定理 1. 入力 U_1, U_2, \dots, U_n と任意の $\{1, 2, \dots, n\}$ の置換 Q , および隣接要素の確率 $P_{rs} = P_{i, i+1}$ ($r = Q(i), s = Q(i+1), i = 1, \dots, n-1$) が与えられているとする. このとき, 任意の二つの組み合わせの確率 P_{jk} が一意に定まる.

Proof. 任意の $j \leq k$ について $s_{j,k} = \sum_{m=j}^{k-1} s_{m,m+1}$ であり, これにより P_{jk} が求まる. 一意性は以下のように示すことができる: P_{jk} は式 (2.12) の定義と式 (2.15) の性質より任意の「経路」 $P_{jm_1}, P_{m_1 m_2}, \dots, P_{m_n k}$ を使って計算できるが, 途中の $s_{m_i m_\kappa} = s_{m_i} - s_{m_\kappa}$ はそれぞれ打ち消し合い, s_j と s_k のみ残る. よって,

得られる P_{jk} は一意である。 □

つまり入力 U_1, \dots, U_m について m 個の確率 $\Pr(U_i \triangleright U_{i+1})$ ($i = 1, \dots, m-1$) が与えられたとき、すべての組み合わせの確率が矛盾なく定まる。

学習は交差エントロピーを損失として、

$$L = -\bar{P}_{ij} \log P_{ij} - (1 - \bar{P}_{ij}) \log(1 - P_{ij}).$$

ここで \bar{P}_{ij} は U_i が実際に U_j よりも優れている確率で、教師データとして与えられる。また、ラベル変数として $S_{ij} \in \{-1, 0, 1\}$ を導入し、 $U_i \triangleright U_j$ であるならば $S_{ij} = 1$ 、 $U_j \triangleright U_i$ であるならば $S_{ij} = -1$ 、どちらでもないとき（二つが同じ優先度である場合） $S_{ij} = 0$ とする。このとき既知の確率 \bar{P}_{ij} は S_{ij} を用いて $\bar{P}_{ij} = \frac{1}{2}(1 + S_{ij})$ となる。式変形すると損失関数は

$$L = \frac{1}{2}(1 - S_{ij})\sigma(s_i - s_j) + \log(1 + e^{-\sigma(s_i - s_j)}) \quad (2.16)$$

とできる。この関数は漸近的に線形となり、雑音が多いような問題では二次の損失関数よりも頑強であるとされている [54]。また $S_{ij} = 0$ のときは 0 を中心に対称となり、 $s_i = s_j$ のとき 0 である。

2.6 深層ニューラルネットワークの判断根拠の可視化

人間社会が計算機による意思決定に依存する割合は多くなってきており、特に機械学習によって得られた予測器を用いて、ある個人や集団に対して影響の大きい判断が下されることが今後ますます多くなると予想される。そういったときには予測器が下した判断が妥当かどうかを人間が検証できる必要がある。予測器が決定木であれば決定木の計算過程をたどることで判断の根拠を理解することができる。また予測器が線形モデルであれば入力に対応する重みを調べることでその入力の重要度を判定することができる。しかし予測器がニューラルネットワーク、特に層が多い深層ニューラルネットワークになると先程のように直接計算過程や重みを観察して判断の根拠を理解することが困難となる [55]。

この節ではニューラルネットワークを利用して判断を下した際の、判断の根拠となった理由を人間にとってわかりやすく可視化するための手法について述べる。

2.6.1 Saliency Map

Saliency Map は画像分類を行うニューラルネットワークの salience（顕著性）を抽出する手法である [56]。Salience とはもともと認知科学の用語であり、複数の感覚刺激のなかである一つの刺激が周囲の刺激と顕著に異なる場合に発生する、夜空を見たときにすぐに月に注意が向くような感覚特性を意味する^{*7}。本稿では Saliency Map を文献 [56] で述べられている、ニューラルネットワークの出力結果を入力で微分したときに得られる二次元画像として定義する。

ニューラルネットワーク $f(\mathbf{x}) \in \mathbb{R}$ を考える。このニューラルネットワークに入力として $\mathbf{x}_0 = (x_0^{(1)}, x_0^{(2)}, \dots, x_0^{(d)})$ が与えられ、出力結果として $f(\mathbf{x}_0)$ が得られたとする。このときの出力結果について、入力の要素 $x_0^{(i)}$ が出力に対してどの程度寄与したかは偏微分係数を用いて、

$$\left. \frac{\partial f}{\partial x^{(i)}} \right|_{\mathbf{x}^{(i)} = x_0^{(i)}}$$

^{*7} 吉田 正俊 サリエンシー 脳科学辞典 <https://bsd.neuroinf.jp/wiki/サリエンシー> (2017)

として考えることができる．なぜなら f は \mathbf{x}_0 まわりで線形近似をするとヤコビ行列 J_f を用いて

$$\begin{aligned} f(\mathbf{x}) &\approx J_f(\mathbf{x}_0)\mathbf{x} + \mathbf{b} \\ &= \sum_{i=1}^d \left. \frac{\partial f}{\partial x^{(i)}} \right|_{\mathbf{x}^{(i)}=\mathbf{x}_0^{(i)}} \cdot x^{(i)} + \mathbf{b} \end{aligned}$$

となり， \mathbf{x}_0 の近傍においてはこの偏微分係数が対応する入力要素の出力に対する寄与として解釈可能であるからである．

画像分類を行うニューラルネットワークは入力として $w \times h \times 3$ の 3 次元配列を採用することが多い (w, h は画像の縦横ピクセル数)．この場合，入力の画像のピクセル値 (i, j) に対応する偏微分係数は 3 つ存在する．偏微分係数から Saliency Map を得るためにはそれぞれのピクセルに対する 3 つの値を集約する必要がある．文献 [56] では 3 つの値のうち絶対値が最大のものの絶対値をある画素に対する salience として定義している．

2.6.2 SmoothGrad

単純な偏微分係数を用いる Saliency Map では得られた結果に雑音が入ることがある．この雑音を除去するための手法として SmoothGrad と呼ばれる手法が提案されている [57]．SmoothGrad では上述のヤコビ行列を用いて Saliency Map を得る際に，入力にガウス雑音 $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_d)$ を加えて偏微分係数を求め，それを平均することで雑音を除去できるとしている．

$$\hat{J}_f(\mathbf{x}_0) = \frac{1}{n} \sum_{j=1}^n J_f(\mathbf{x}_0 + \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_d))$$

ここで n はサンプルサイズであり，文献 [57] では 50 程度で十分であるとしている．入力にガウス雑音を加えた結果の平均を計算することで入力画像の近傍で平滑化された値が得られるため，人間にとって雑音が少ない結果が得られると期待されている．

第 3 章

局面の特徴を考慮した Incremental Random Game Tree

2.1 節 で述べたとおり，ゲームを模した人工的な探索空間は探索アルゴリズムの性能評価を行う際に有用となる．この章では 2.1.1 項 で述べた P-game tree を拡張した「特徴ベクトル付き Incremental Random Game Tree」(Incremental Random Game Tree with Feature Vectors) を提案し，その構成方法および性質について議論する．

提案する特徴ベクトル付き Incremental Random Game Tree では，P-game tree の構成法に倣い，根局面からの経路を元にそれぞれの局面における d 次元の特徴ベクトル \mathbf{x}_s を生成する．そしてある局面のスコアはその局面の特徴ベクトルと，木全体で共有する単一の d 次元係数ベクトル $\boldsymbol{\theta}^* \in \mathbb{R}^d$ との内積

$$v_s = \mathbf{x}_s^\top \boldsymbol{\theta}^* \quad (3.1)$$

で決定する．ここでこの係数ベクトル $\boldsymbol{\theta}^*$ は探索アルゴリズムには明かされず，実験者が適当に決定するものとする．このような局面のスコアがある重みベクトルと局面の特徴ベクトルとの線形結合で表されるという性質は，ゲーム木探索アルゴリズムで標準的に用いられる評価関数の計算方法を模している [58, 5]．

このような木の構成法により，1) 木全体の幅と高さのパラメータを変更することにより探索空間を容易に変更することができ，2) 実際のゲームで期待されるような親子局面間の関係性が生成される，という利点がある．

各局面に対応する特徴ベクトルの構成方法はいくつか考えられる．ここでは主に 2 つの方法について提案し，それぞれの性質について議論する．

3.1 Point Mutation Feature

はじめに提案する特徴ベクトルの構成方法は，特徴ベクトルの要素を一定確率で反転させることにより，実際のゲームで存在するような「特徴が少しずつ変化していく」という性質を模倣している．以下この手法を *point mutation feature*，この手法により作成した人工木を *point mutation feature tree* と呼ぶ．この構成法では，根局面に二つの d 次元の二値ベクトル $\mathbf{x}^m, \mathbf{x}^o \in \{0, 1\}^d$ を割り当てる． $\mathbf{x}^m, \mathbf{x}^o$ は手番プレイヤーにとって好ましい特徴と相手番プレイヤーにとって好ましい特徴をそれぞれ表している．

内部節点の特徴ベクトルは，親局面の特徴ベクトルを引き継いだ $\mathbf{x}^{m'}$ と $\mathbf{x}^{o'}$ をもつ．その際，二人ゲームの特徴を表すため， $\mathbf{x}^{m'}$ は \mathbf{x}^o を， $\mathbf{x}^{o'}$ は \mathbf{x}^m を引き継ぐ．またこのとき着手による局面の進行を表すため

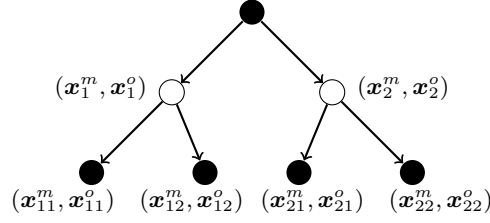


図 3.1: Point Mutation Feature Tree の例. 各局面は二値の局面特徴ベクトルを二つ持つ

に, 引き継いだ特徴ベクトルの要素 $(\mathbf{x}^{m'})_i, (\mathbf{x}^{o'})_i$ ($i \in \{1, \dots, d\}$) を確率 p で反転 (0 を 1 に, 1 を 0 に) させる. つまりある要素 x_n がその親節点の対応する要素 x_{n-1} と異なる確率が以下の式で表される.

$$\Pr(x_n \neq x_{n-1}) = p$$

この反転は各要素ごとに独立に行う. この $\mathbf{x}^m, \mathbf{x}^o$ を用いて局面 s の特徴ベクトル \mathbf{x}_s を

$$\mathbf{x}_s = \mathbf{x}^m - \mathbf{x}^o$$

と定める. 局面の評価値は, 係数ベクトル $\boldsymbol{\theta}^*$ との内積

$$\begin{aligned} v_s &= \mathbf{x}_s^\top \boldsymbol{\theta}^* \\ &= (\mathbf{x}^m - \mathbf{x}^o)^\top \boldsymbol{\theta}^* \end{aligned}$$

となる. 図 3.1 にこの手法によって構築される人工木とそれぞれの局面を示した模式図を示す.

また特徴ベクトルの要素を反転させる方法を変えることにより, 再現したいゲームの性質を変化させることができる. 以下にいくつかのゲームの性質とその方法について議論する.

■減衰係数付き Point Mutation Feature 上述した, 反転確率 p が固定されている point mutation feature は木の深さによらず確率が一定であるため, 木が深くなるほど根局面の特徴ベクトルとの相関が薄れていく. 序盤と中盤では局面が大きく異なる, つまり特徴ベクトルが大きく異なるようなゲームが普通であると考えられるが, ゲーム終盤などにおいては局面があまり変化しないなどの状況も十分考える.

減衰係数付き point mutation feature は反転確率 p に減衰係数 $\gamma \in [0, 1]$ を乗じて木の深さが深くなるにつれて変化を緩やかにすることをモデル化した手法である. この方式では, 特徴ベクトルの各要素の反転確率を以下のように定義する.

$$\Pr(x_n \neq x_{n-1}) = p\gamma^n$$

この方式により, 局面進行によって特徴ベクトルがあまり変化しないゲーム展開を模式化することができる.

■Parent-Grandparent Mutation Feature 次に, 変異確率を以下の式で定義する parent-grandparent mutation を導入する.

$$\Pr(x_n \neq x_{n-1}) = \begin{cases} p & (x_{n-1} = x_{n-2} \text{ or } n = 1) \\ q & (x_{n-1} \neq x_{n-2}) \end{cases}$$

実際のゲームにおいては一方のプレイヤーは自分のプレイヤーにとって有利であるような局面特徴をできるだけ保持しようとすると考えられる. そのような状況を分枝数が限られた人工木で模倣するために, 親の局面特徴が祖父局面の特徴と同一の場合と, そうでない場合に場合分けしてそれぞれ別の変異確率をもつようなモ

デルが parent-grandparent mutation feature である．ここで p と q は変異を制御するパラメータであり， $p < q$ のとき深さ n における局面の特徴ベクトルの要素を深さ $n-2$ の局面（祖父局面）と同じになる確率が高くなる．

3.1.1 Point Mutation Feature の性質

評価値の分散

Point mutation feature では特徴ベクトルが二値ベクトルであり，式 (3.1) によって局面における評価値を生成する． $\mathbf{x}_s = \mathbf{x}^m - \mathbf{x}^o$ の要素は $\{-1, 0, 1\}$ の値を取りうるので，この評価値の値域は $\boldsymbol{\theta}^*$ の値域によって決定される．例えば v_s の値域を $[-1/2, 1/2]$ にするためには $\boldsymbol{\theta}^*$ の要素は $[-1/2d, 1/2d]$ である必要がある．しかしながら $\boldsymbol{\theta}^*$ の要素の値域をこのように定めてしまうと，特徴ベクトルの次元が増えるにつれて要素が取りうる範囲が狭くなり，その結果 v の取りうる範囲も統計的に狭まってしまう．図 3.2 に， $\boldsymbol{\theta}^*$ の要素を $\mathcal{U}(-1/2d, 1/2d)$ によって生成した際の $v_s + 0.5$ の値のヒストグラムをベクトルの次元別に示す．図から明らかなように，ベクトルの次元が大きくなるにつれて観測された値は徐々に 0.5 に集中していく．これは $\boldsymbol{\theta}^*$ の要素を $\mathcal{U}(-1/2d, 1/2d)$ によって生成した際に分散が以下のように計算されるからである．

$$\begin{aligned}\text{Var}[v_s] &= \text{Var}[(\mathbf{x}^m - \mathbf{x}^o)^\top \boldsymbol{\theta}^*] \\ &= \sum_{i=1}^d \text{Var}[(x_i^m - x_i^o)\theta_i^*], \\ &= \sum_{i=1}^d (\text{Var}[x_i^m] + \text{Var}[x_i^o]) \text{Var}[\theta_i^*] \\ &= \sum_{i=1}^d \left(\frac{1}{4} + \frac{1}{4}\right) \frac{(1/2d - (-1/2d))^2}{12} = \frac{1}{24d}.\end{aligned}$$

図 3.3 に観測した分散と，理論値を示す．このように特徴ベクトルの次元によって取りうる値が統計的に狭まってしまうことは好ましくない．特徴ベクトルの次元はアルゴリズムの計算量的性能に大きく影響を与えるため，評価の段階ではさまざまな次元でテストができる必要がある．

そこで要素の範囲を次元の平方根に反比例する 0 対称な一様分布 $\mathcal{U}(-w/\sqrt{d}, w/\sqrt{d})$ で生成する．このとき評価値の分散は，

$$\begin{aligned}\text{Var}[v_s] &= \sum_{i=1}^d \left(\frac{1}{4} + \frac{1}{4}\right) \frac{(w/\sqrt{d} - (-w/\sqrt{d}))^2}{12} \\ &= d \cdot \frac{1}{2} \frac{2w^2}{12d} \\ &= \frac{w^2}{6}\end{aligned}$$

となり，次元によらず分散が一定となる．このとき $w = \sqrt{6}\sigma'$ とすると評価値の分散が σ'^2 となり， $\sigma' = 1/6$ ならば正規分布として近似したときに評価値の 99.7% が $[-1/2, 1/2]$ の範囲に収まる．まとめると， $\boldsymbol{\theta}^*$ の各要素を $\mathcal{U}(-1/\sqrt{6d}, 1/\sqrt{6d})$ で生成すると適した値となる．

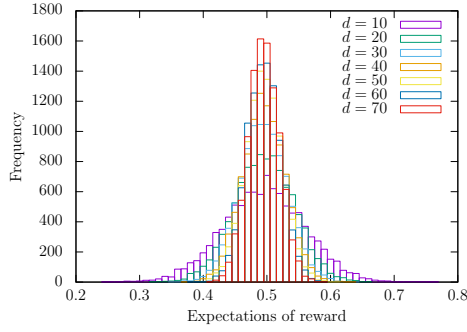


図 3.2: 実際に観測した報酬の分布

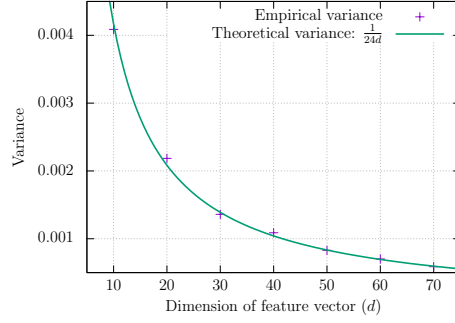


図 3.3: 報酬の分布の分散

変異確率と評価値の期待値

次に変異確率 p と局面の評価値との関係について述べる．この項では x_n を木の深さ n における特徴ベクトルの要素と表記する．

まず，変異確率が一定の木のモデルについて次の定理を示すことができる．

定理 2. 変異確率 p が与えられたとき，深さ n における局面特徴ベクトルのある要素 x_n が根局面の対応する要素 x_0 と異なる確率 $\Pr(x_n \neq x_0)$ は $\frac{1}{2}(1 - (1 - 2p)^n)$ ．

Proof. ある要素 x_n が根局面における要素 x_0 と異なる場合は，値の反転が奇数回起こったときである． n 回の試行のうち，反転が k 回起こる組み合わせの数は $\binom{n}{k}$ である．すなわち $q = 1 - p$ としたとき， $(q + p)^n$ の展開した係数を用いて，

$$\Pr(x_n \neq x_0) = \sum_{i=0, i \in 2\mathbb{Z}+1}^n \binom{n}{i} q^{n-i} p^i.$$

二項定理から

$$(q + p)^n = \sum_{i=0}^n \binom{n}{i} q^{n-i} p^i, \quad (q - p)^n = \sum_{i=0}^n (-1)^i \binom{n}{i} q^{n-i} p^i$$

であるため，

$$\begin{aligned} (q + p)^n - (q - p)^n &= \sum_{i=0}^n (1 - (-1)^i) \binom{n}{i} q^{n-i} p^i \\ &= \sum_{i=0, i \in 2\mathbb{Z}+1}^n 2 \binom{n}{i} q^{n-i} p^i \\ \sum_{i=0, i \in 2\mathbb{Z}+1}^n \binom{n}{i} q^{n-i} p^i &= \frac{1}{2}((q + p)^n - (q - p)^n). \end{aligned}$$

よって $\Pr(x_n \neq x_0) = \frac{1}{2}(1 - (1 - 2p)^n)$ が導かれた． \square

また漸化式 $\Pr(x_n \neq x_0) = (1 - p)\Pr(x_{n-1} \neq x_0) + p(1 - \Pr(x_{n-1} \neq x_0))$ ， $\Pr(x_1 \neq x_0) = p$ を解くことによっても同様の結果が得られる．この定理から直ちに次のことが言える．

系 1. $\forall n \in \mathbb{N}_+, p = 0.5 \Rightarrow \Pr(x_n \neq x_0) = 0.5$

系 2. $n \rightarrow \infty \Rightarrow \Pr(x_n \neq x_0) \rightarrow \frac{1}{2}$

図 3.4 は局面の深さ n と反転確率 $\Pr(x_n \neq x_0)$ の関係を表したグラフである。これらの結果から分かる通り、深さが深くなるにつれて根局面の特徴ベクトルと異なる確率が $1/2$ に近づくため、木が深くなるに連れて根局面の特徴ベクトルと独立な値が生成される。

また、深さ n の局面における特徴ベクトルが根局面の特徴ベクトルと比較してどれだけの要素が反転しているかについて、それぞれの要素の反転は独立試行であるから二項分布の期待値を取って、

$$\begin{aligned} \mathbf{E}[B(d, \Pr(x_n \neq x_0))] &= d \cdot \Pr(x_n \neq x_0) \\ &= \frac{d}{2}(1 - (1 - 2p)^n) \end{aligned} \quad (3.2)$$

となる。つまり大きな n については要素の半分が根局面に割り当てた特徴ベクトルと異なると期待される。

根局面に割り当てた特徴ベクトル \mathbf{x}_0 と深さ n における特徴ベクトル \mathbf{x}_n の関係について、以下の式が成り立つ。

$$\begin{aligned} \mathbf{E}[\mathbf{x}_n] &= (1 - \Pr(x_n \neq x_0))\mathbf{x}_0 + \Pr(x_n \neq x_0)(\mathbf{1} - \mathbf{x}_0) \\ &= \frac{1}{2}(1 - (1 - 2p)^n)\mathbf{1} + (1 - 2p)^n\mathbf{x}_0 \end{aligned}$$

ここで $\mathbf{1}$ はすべての要素が 1 であるベクトルである。また、

$$\begin{aligned} \mathbf{E}[\mathbf{x}_n^m - \mathbf{x}_n^o] &= (1 - 2p)^n\mathbf{x}_0^m + \frac{1}{2}(1 - (1 - 2p)^n)\mathbf{1} - (1 - 2p)^n\mathbf{x}_0^o - \frac{1}{2}(1 - (1 - 2p)^n)\mathbf{1} \\ &= (1 - 2p)^n(\mathbf{x}_0^m - \mathbf{x}_0^o) \end{aligned}$$

なので係数ベクトル $\boldsymbol{\theta}^*$ が与えられたとき、深さ n の局面の評価値 v_n は、根の評価値を v_0 とすると、

$$\begin{aligned} \mathbf{E}[v_n] &= \mathbf{E}[(\mathbf{x}_n^m - \mathbf{x}_n^o)^\top \boldsymbol{\theta}^*] \\ &= (1 - 2p)^n(\mathbf{x}_0^m - \mathbf{x}_0^o)^\top \boldsymbol{\theta}^* \\ &= (1 - 2p)^n v_0 \end{aligned}$$

となる。すなわち、深さ n の局面における評価値は、 n が大きいときに 0 に近づく。

減衰係数のある point mutation feature についても同様に次の定理が成り立つ。

定理 3. 反転確率 p と減衰係数 γ が与えられたとき、深さ n における特徴ベクトルの要素 x_n が根局面の対応する特徴ベクトルの要素 x_0 と異なる確率 $\Pr(x_n \neq x_0)$ は $\frac{1}{2}(1 - \prod_{i=1}^n (1 - 2p\gamma^i))$

Proof. 漸化式

$$\Pr(x_n \neq x_0) = (1 - p)\Pr(x_{n-1} \neq x_0) + p(1 - \Pr(x_{n-1} \neq x_0))$$

を解くと

$$\Pr(x_n \neq x_0) = \frac{1}{2} \left(1 - \prod_{i=1}^n (1 - 2p\gamma^i) \right)$$

が得られる。 □

また parent-grandparent モデルについて以下に述べる。このモデルは祖父局面と親局面に応じて特徴ベクトルの要素の反転確率が異なるモデルであるため、深さ n の局面における要素の反転確率 $\Pr(x_n \neq x_0)$ は条件付き確率で以下の式で表される。

$$\Pr(x_n \neq x_0) = \Pr(x_n \neq x_0 \mid x_{n-1} = x_0) + \Pr(x_n \neq x_0 \mid x_{n-1} \neq x_0)$$

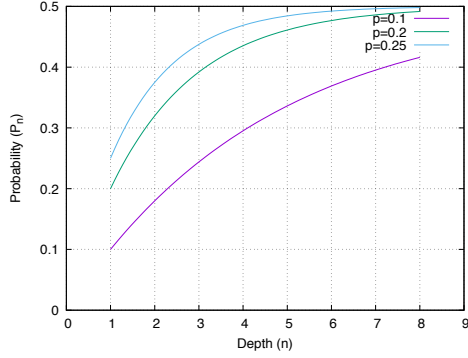


図 3.4: Point Mutation Feature における要素が異なる確率 (理論値)

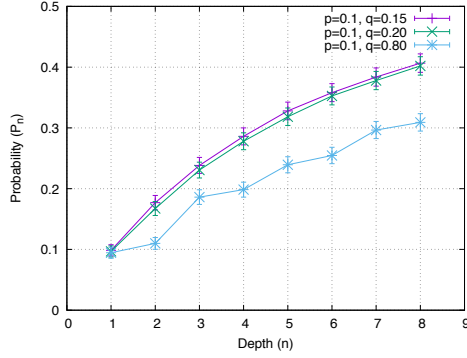


図 3.5: Parent-Grandparent Mutation Feature における要素が異なる確率 (観測値, エラーバーは 99.5% 信頼区間)

この関係式から、以下の行列による表現が可能である。

$$\begin{pmatrix} \Pr(x_n = x_0 | x_{n-1} = x_0) \\ \Pr(x_n = x_0 | x_{n-1} \neq x_0) \\ \Pr(x_n \neq x_0 | x_{n-1} = x_0) \\ \Pr(x_n \neq x_0 | x_{n-1} \neq x_0) \end{pmatrix} = \begin{pmatrix} 1-p & 1-q & 0 & 0 \\ 0 & 0 & p & q \\ 0 & 0 & 1-p & 1-q \\ p & q & 0 & 0 \end{pmatrix} \begin{pmatrix} \Pr(x_{n-1} = x_0 | x_{n-2} = x_0) \\ \Pr(x_{n-1} = x_0 | x_{n-2} \neq x_0) \\ \Pr(x_{n-1} \neq x_0 | x_{n-2} = x_0) \\ \Pr(x_{n-1} \neq x_0 | x_{n-2} \neq x_0) \end{pmatrix},$$

$$= \begin{pmatrix} 1-p & 1-q & 0 & 0 \\ 0 & 0 & p & q \\ 0 & 0 & 1-p & 1-q \\ p & q & 0 & 0 \end{pmatrix}^n \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

一般の場合についてこの関係式は簡単ではないが、 p, q および n が与えられたとき、この行列は $O(\log n)$ の計算量で計算できる。ここではいくつかの p, q についてシミュレーションを行い反転確率を観測した。図 3.5 に観測結果を示す。図 3.4 の $p = 0.1$ の場合と比較して、 $p = 0.1, q = 0.15$ の場合はすべての n についてわずかに小さい値となっている。 $p = 0.1, q = 0.8$ の場合はそれと比較して大きく様子が異なる。すべての n について確率は小さくなり、また n の偶奇によって値が上下している。

3.2 Incremental Random Feature

3.1 節では局面特徴ベクトルを変化させる際に、ベクトルの要素をそれぞれ独立に変化させる方法を提案した。この節では、別の手法である *incremental random feature* を提案する。Incremental random feature では、局面に割り当てる特徴ベクトル $\mathbf{x}_s \in \mathbb{R}^d$ をすべての枝に独立に割り当てた行動特徴ベクトル \mathbf{x}^a の総和で定義する。つまり、ある局面 s_0 から以下の状態行動列

$$\{a_1, s_1, a_2, s_2, \dots, a_{n-1}, s_{n-1}, a_n\}$$

を経由して局面 s_n へたどり着いたとき、局面 s_n の特徴ベクトルは行った行動の特徴ベクトルを用いて、

$$\mathbf{x}_{s_n} = \mathbf{x}_{s_0} + \sum_{i=1}^n \mathbf{x}^{a_i} \quad (3.3)$$

として定める。根局面の特徴ベクトルと行動特徴ベクトルの要素は 0 を中心とする連続一様分布でそれぞれ独立に生成する。これは P-game tree のスコアが枝の総和となっていることを念頭に、同様の枠組みを特徴

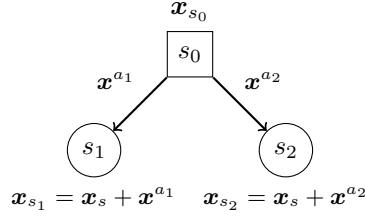


図 3.6: Incremental Random Feature Tree の例. 局面 s_1 (s_2) への行動に対する行動特徴ベクトルは x_{s_1} (x_{s_2}). 局面 s_1 (s_2) の局面特徴ベクトルは $x_s + x_{s_1}$ ($x_s + x_{s_2}$)

ベクトルに拡張したものと考えることができる. 図 3.6 に incremental random feature tree の例を示す.

3.2.1 Incremental Random Feature の性質

ここでは incremental random feature のいくつかの自明な性質について述べる. 評価値計算に用いられる係数ベクトル θ^* の各要素は $\mathcal{U}(-w_\theta, w_\theta)$ によって生成され, また根局面の特徴ベクトル x_{s_0} と行動特徴ベクトル x^{a_i} の要素も $\mathcal{U}(-w_a, w_a)$ によって生成されたとする.

■期待値 式 (3.3) で定義される局面 s_n の特徴ベクトル x_{s_n} の期待値は,

$$\begin{aligned} \mathbf{E}[x_{s_n}] &= \mathbf{E}\left[x_{s_0} + \sum_{i=1}^n x^{a_i}\right] \\ &= x_{s_0} + \sum_{i=1}^n \mathbf{E}[x^{a_i}] \\ &= x_{s_0} \end{aligned}$$

ここで行動特徴ベクトル x^{a_i} の要素は 0 中心の一様分布により生成されているため $\mathbf{E}[x^{a_i}] = \mathbf{0}$ である. よって, 局面 s_n の評価値 (式 (3.1)) の期待値は, 係数ベクトル θ^* が与えられたとき,

$$\begin{aligned} \mathbf{E}[v_{s_n}] &= \mathbf{E}[x_{s_n}^\top \theta^*] \\ &= \mathbf{E}\left[(x_{s_0} + \sum_{i=1}^n x^{a_i})^\top \theta^*\right] \\ &= \mathbf{E}[x_{s_0}^\top \theta^*] \\ &= \mathbf{E}[v_{s_0}] \end{aligned}$$

と計算できる. つまりある局面 s_0 のスコアと, その子孫である局面 s_n のスコアは同じ期待値をもつ. この性質はもともとの P-game tree と同様である [22].

■分散 局面 s_n における評価値の分散は以下のように求められる．

$$\begin{aligned}
\mathbf{Var}[v_{s_n}] &= \mathbf{Var}[\mathbf{x}_{s_n}^\top \boldsymbol{\theta}^*] \\
&= \mathbf{Var}\left[\left(\mathbf{x}_{s_0} + \sum_{i=1}^n \mathbf{x}^{a_i}\right)^\top \boldsymbol{\theta}^*\right] \\
&= \mathbf{Var}\left[\sum_{j=1}^d \sum_{i=0}^n (\mathbf{x}^{a_i})_j (\boldsymbol{\theta}^*)_j\right] \\
&= \frac{(2w_a)^2 (2w_\theta)^2}{12^2} (n+1)d \\
&= \frac{16w_a^2 w_\theta^2}{144} (n+1)d \\
&= \frac{w_a^2 w_\theta^2}{9} (n+1)d
\end{aligned} \tag{3.4}$$

ここで $\mathbf{x}_{s_0} = \mathbf{x}^{a_0}$ とした．一様分布に従う n 個の独立な確率変数の総和は Irwin-Hall 分布と呼ばれるが， n が十分大きいときには正規分布として近似できる．すなわち特徴ベクトルの次元や局面の深さ n が大きいとき，平均が 0，分散が式 (3.4) の正規分布として近似できる．

3.3 まとめ

本章では新たに局面の特徴を用いて探索を行うアルゴリズムの性能評価を行うことができる人工的なテストベッドを二種類提案し，その性質について論じた．提案したどちらのモデルも P-game tree を拡張したものとみなすことができ，P-game tree のように実験者が容易に探索空間の大きさを設計できると期待できる．特に Incremental Random Feature Tree は P-game の自然な拡張であると思われる．またどちらのモデルでも評価値の分布が正規分布と仮定することができ，分布の裾もパラメータで設定可能になっている．

より実際のゲームへ近づけていくための今後の課題として，ベクトルの要素間の依存関係の導入が挙げられる．今回導入したモデルはどちらも特徴ベクトルの要素も係数ベクトルの要素もそれぞれ独立に生成していた．これは評価値の分布などの解析が行いやすいという利点もあるが，実際のゲームではゲームの要素が互に関連しているという状況はよく存在する．こういったことをモデル化することでより実際のゲームに近づけることが可能であると思われる．

第 4 章

LinUCT による探索中の知識の獲得

この章では事前に学習が行えないようなゲームにおいて効率よく知識を獲得し、その獲得した知識を探索に役立てることができるようなアルゴリズム LinUCT を提案する。ベースとなる探索アルゴリズムはモンテカルロ木探索を用い、知識の獲得に LinUCB アルゴリズムを用いる。事前に学習できないゲームとして、例えば General Game Playing やそのビデオゲーム版である General Video Game Playing などが挙げられる。これらの設定ではゲームのプレイヤーはゲームのルールのみが与えられ、事前にそのゲームについて知ることができない。このような状況であっても、ゲームのプレイ中にゲームの知識を得て、得られた知識をプレイに活かすことができれば有用であると考えられる。

LinUCT ではゲームのシミュレーションの報酬を線形バンディット問題として捉え、その報酬を効率よく予測することで探索を行うことを目指している。つまり、ある局面 s の特徴ベクトルを \mathbf{x}_s としたとき、その局面からのシミュレーションの報酬 r_s が、

$$\mathbf{E}[r_s] = \mathbf{x}_s^\top \boldsymbol{\theta}^*$$

として表すことができると仮定している。ここで $\boldsymbol{\theta}^*$ は探索木全体で共有する未知の係数ベクトルであり、LinUCB と同様 LinUCT もこれを推定していく。局面の特徴ベクトルから評価値が得られるという仮定は標準的であり [58, 5]、この仮定は妥当であると思われる。

まずはじめに単純に LinUCB をモンテカルロ木探索に応用したアルゴリズムについて述べ、ゲーム木探索の性質に沿うように変更したアルゴリズムについて説明する。その後提案したアルゴリズムを 第 3 章 で提案した人工木や 2.1.2 項 で紹介したゲームでの評価を行う。

4.1 アルゴリズム

4.1.1 LinUCT-PLAIN

まずはじめに単純に LinUCB をモンテカルロ木探索に適用したアルゴリズムである LinUCT-PLAIN について述べる。LinUCT-PLAIN ではモンテカルロ木探索における選択の基準値に LinUCB アルゴリズムで計算した報酬の予測値を用いて次の局面を選択する。具体的には以下の式で基準値を計算する。

$$\text{LinUCB}' = \mathbf{x}_a^\top \hat{\boldsymbol{\theta}} + \alpha \sqrt{\mathbf{x}_a^\top \cdot \frac{N(s_0)}{N(s_0, a)} \mathbf{A}^{-1} \cdot \mathbf{x}_a}, \quad (4.1)$$

アルゴリズム 3 にモンテカルロ木探索における LinUCT の付随的なアルゴリズムを記す．Procedure LINUCT-INITIALIZE では予測に用いる変数 \mathbf{A} , \mathbf{b} の初期化を行い, procedure BACK-PROPAGATION-MATRIX でモンテカルロ木探索の伝播のプロセスにおいて各局面の勝率と訪問数を更新すると同時にその局面における特徴ベクトルを用いて回帰の更新を行う．更新はそれぞれの局面の特徴ベクトルと, その局面における手番プレイヤーから見た報酬の結果（ゲームの勝ち負け）を用いて行う．この方式では, 一回の報酬の観測（プレイアウト）において辿った経路上の局面の数だけ回帰の更新を行う．しかしもともとの LinUCB アルゴリズムでは一回の観測に対し一回の回帰予測の更新を行っている．このため, モンテカルロ木探索に LinUCB を単純に適用した LinUCT-PLAIN では回帰の更新回数を考慮して補正するため, 式 (4.1) の第二項に N_{s_0}/N_a を加えている．

アルゴリズム 3 Supplementary Procedures in LinUCT

```

procedure LINUCT-INITIALIZE
     $\mathbf{A} \leftarrow \mathbf{I}_{d \times d}$ 
     $\mathbf{b} \leftarrow \mathbf{0}_{1 \times d}$ 
end procedure

procedure BACK-PROPAGATION-MATRIX(path,  $\Delta$ )
    for  $s \in \text{path}$  do
         $\mathbf{A} \leftarrow \mathbf{A} + \mathbf{x}_s \mathbf{x}_s^\top$ 
         $\mathbf{b} \leftarrow \mathbf{b} + \Delta_s \mathbf{x}_s$ 
    end for
end procedure

```

4.1.2 LinUCT-RAVE

LinUCT-PLAIN ではモンテカルロ木探索における選択の際に用いられる基準値がその局面における静的な特徴ベクトルでのみ定まってしまう．これは反復的な木探索アルゴリズムでは問題になる．なぜなら, ある局面の評価値が静的な特徴ベクトルでのみ定まってしまうとその評価値が他の兄弟局面よりも悪ければ, その局面の子孫局面に最善局面が存在したとしても決して探索されないからである．

これを解決するために, モンテカルロ木探索の選択における基準値に LinUCB と UCB1 値を組み合わせた LinUCT-RAVE を考案する．RAVE とは囲碁で用いられた評価値をより早く推定するためのアルゴリズムであるが, ここでは RAVE で主に用いられている AMAF ヒューリスティックではなく, 式 (4.2) で定義する．

$$\text{LinUCB}_{\text{RAVE}}(s, a) = \beta(s) \text{LinUCB}'(s, a) + (1 - \beta(s)) \text{UCB1}(s, a). \quad (4.2)$$

β は次の式で定める

$$\beta(s) = \sqrt{\frac{k}{3N_s + k}}$$

k はパラメータである．この式の値は局面への訪問回数が増えるにつれて UCB1 値の影響が大きくなり, 実際の報酬を重視するようになる．

4.1.3 LinUCT-FP

ここで述べる LinUCT-FP は異なるアプローチによって上述の問題を解決しようとしている．LinUCT-FP ではある局面を根とする部分木中の局面の特徴ベクトルを考慮に入れるために，部分木の根局面の特徴ベクトルを子孫局面の特徴ベクトルへ少しずつ近づけていく．具体的にはプレイアウトを行ったあとのモンテカルロ木探索の更新の処理中に，子局面の特徴ベクトルを使って親局面の特徴ベクトルを以下の式に従って変化させる．

$$\mathbf{x}_p \leftarrow (1 - \gamma) \mathbf{x}_p + \gamma \mathbf{x}_s, \quad (4.3)$$

ここで $\gamma \in (0, 1)$ はパラメータである．この更新式はモンテカルロ木探索の伝播の際，探索木の葉節点から順に行う．この処理によって，ある局面の特徴ベクトルは以前にプレイアウトを行った部分木中の局面の特徴ベクトルの情報を含むことになり，式 (4.1) の値はその部分木中の子孫局面の価値を反映することが可能になる．アルゴリズム 4 に LinUCT-FP において追加される手続きを示す．また LinUCT-FP と LinUCT-RAVE のアルゴリズムはそれぞれ独立した処理であるため，二つの処理を行う LinUCT-RAVE-FP も考えられる．

アルゴリズム 4 Back-propagation process of LinUCT-FP

```

procedure BACK-PROPAGATION-MATRIX(path,  $\Delta$ )
  for  $s \in \text{path}$  do
     $\mathbf{A} \leftarrow \mathbf{A} + \mathbf{x}_s \mathbf{x}_s^\top$ 
     $\mathbf{b} \leftarrow \mathbf{b} + \Delta_s \mathbf{x}_s$ 
     $p \leftarrow \text{parent of } s$ 
    if  $p$  is not null then
       $\mathbf{x}_p \leftarrow (1 - \gamma) \mathbf{x}_p + \gamma \mathbf{x}_s$ 
    end if
  end for
end procedure

```

4.1.4 LinUCT-WP

LinUCT-FP と同様に，ここで述べる LinUCT-WP もある局面の特徴ベクトルをその部分木中の特徴ベクトルを用いて表現することで，静的な特徴ベクトルによる評価という LinUCT-PLAIN の問題の解決を目的としたアルゴリズムである．LinUCT-WP ではモンテカルロ木探索の反復ごとにプレイアウトの経路上のある局面の特徴ベクトルを，その局面の子局面の特徴ベクトルの重み付き平均で置き換える．平均の重みはそれぞれの子局面の訪問回数に比例する値としている．具体的には，ある局面 s の特徴ベクトル \mathbf{x}_s は

$$\mathbf{x}_s = \frac{\sum_{s_i \in \text{children}} (T_{s_i}(t) + 1) \cdot \mathbf{x}_{s_i}}{\sum_{s_i \in \text{children}} (T_{s_i}(t) + 1)}$$

となるように特徴ベクトルを更新する．これにより，訪問回数が多い（つまりより有望な）子局面を重視することになる．また，このように特徴ベクトルが更新されることで，

$$\begin{aligned}\mathbf{x}_s^\top \boldsymbol{\theta}^* &= \left(\frac{\sum_{s_i \in \text{children}} (T_{s_i}(t) + 1) \cdot \mathbf{x}_{s_i}}{\sum_{s_i \in \text{children}} (T_{s_i}(t) + 1)} \right)^\top \boldsymbol{\theta}^* \\ &\approx \sum_{s_i \in \text{children}} \frac{T_{s_i}(t)}{T_s(t)} \cdot (\mathbf{x}_{s_i}^\top \boldsymbol{\theta}^*) \\ &= \sum_{l_i \in \text{leaves in sub-tree under } s} \frac{T_{l_i}(t)}{T_s(t)} \cdot (\mathbf{x}_{l_i}^\top \boldsymbol{\theta}^*)\end{aligned}$$

となり，LinUCT-WP で変更されたある局面 s の特徴ベクトル \mathbf{x}_s と $\boldsymbol{\theta}^*$ との内積は局面 s を根とする部分木の葉局面の特徴ベクトルと $\boldsymbol{\theta}^*$ との内積を訪問回数で重み付けしたものと解釈できる．アルゴリズム 5 にアルゴリズムを示す．また特徴ベクトルの更新の際にはある局面の全ての子局面集合を列挙する必要はなく，訪問した子局面の特徴ベクトルを用いて増分を計算することが可能である．

アルゴリズム 5 Back-propagation process of LinUCT-WP

procedure UPDATEFEATURE-LINUCTWP(s)

if s is leaf node **then return**

end if

$S \leftarrow \text{children of } s$

$\mathbf{x}_s \leftarrow \sum_{s_i \in S} (T_{s_i}(t) + 1) \cdot \mathbf{x}_{s_i} / \sum_{s_i \in S} (T_{s_i}(t) + 1)$

end procedure

procedure BACKPROPAGATION(path, Δ)

$s_l \leftarrow \text{leaf node}$

 UPDATEREGRESSION(s_l, Δ_s)

 # supplementary procedure of LinUCT-WP

for $s \in \text{path}$ **do**

▷ From bottom to root

 UPDATEFEATURE-LINUCBWP(s)

end for

end procedure

4.1.5 Leaf-LinUCT

これまでに述べてきたアルゴリズムはプレイアウトで辿ってきた経路上のすべての局面の特徴ベクトルを用いて $\boldsymbol{\theta}^*$ の予測を更新してきた．しかしながら，シミュレーションで観測した報酬と局面の特徴ベクトルとの関係性は探索木の浅い局面よりも深い局面のほうがより強いと推測される．したがってプレイアウトの経路上のすべての局面の特徴ベクトルを回帰予測に用いるよりも，葉局面の特徴ベクトルのみを利用したほうが単一の回帰係数によって報酬が予測できるという仮定のもとでは有効に働くと考えられる．そこで，ここでは探索木の葉局面の特徴ベクトルのみを利用する Leaf-LinUCT を考える．

4.2 評価実験

4.2.1 Point Mutation Tree での評価

まずはじめに 3.1 節 で定義した人工木である point mutation feature tree でのそれぞれのアルゴリズムの評価を行った。ここで比較したアルゴリズムは LinUCT-PLAIN, LinUCT-RAVE, LinUCT-FP, LinUCT-RAVE-FP, および UCT である。比較する指標として、探索アルゴリズムの誤答率 (failure rate) [22] と average regret を用いた。

時刻 t における誤答率は、探索アルゴリズムが時刻 t で終了したとしたときに返した最善手と実際の最善手とが異なる場合の割合である。UCT を始めとするモンテカルロ木探索では、次に選択する局面は終了時に最も多く訪問された根局面の子局面とすることが多い (robust child)。この実験では時刻 t における探索木において、根局面の子局面集合のうち最も多く訪問された子局面が最善でなかった場合に誤答とし、その割合を計測した。また average regret は 2.4.1 項 で述べたとおり以下の式で定義される。

$$R/n = \mu^* - 1/n \sum_{t=1}^n \mu_{i_t}.$$

この値が小さければ小さいほどよい。

実験で用いる人工木は 3.1 節 で述べた手順によってランダムに作成したが、根局面の子局面全てに対して独立に特徴ベクトルを割り与えた。これにより部分木ごとに異なった報酬の分布が存在すると期待できる。人工木のパラメータである p は 0.1 とし、特徴ベクトルの次元は人工木の葉節点の多さに従って決定した。係数ベクトル θ^* の要素は 3.1.1 項 で述べたとおり $\mathcal{U}(-1/\sqrt{6d}, 1/\sqrt{6d})$ の連続一様分布で生成する。また最善手が複数存在する木は除外している。実験で用いたモンテカルロ木探索アルゴリズムはすべて探索木の節点を二度目の訪問で展開する (展開できる場合)。シミュレーションは一様ランダムな選択により葉局面にたどり着くまで行われ、式 (3.1) で定義した、たどり着いた葉局面の評価値に従って報酬を生成する。報酬は局面 s の評価値 v_s を用いるベルヌーイ分布によって報酬を生成する。

$$\begin{aligned} \Pr(r = 1) &= \text{clamp}(v_s + \frac{1}{2}, 0, 1) \\ \Pr(r = 0) &= 1 - \Pr(r = 1) \end{aligned}$$

ここで $\text{clamp}(x, a, b) = \max(a, \min(x, b))$ である。

パラメータによる性能変化

これまでに述べた LinUCT アルゴリズムの中にはパラメータがいくつか存在する。ここではそれぞれのパラメータを変更した際のアルゴリズムの性能の変化について実験を行った。

LinUCT の選択に影響を与える α は 1.0, 1.59, 1.83, および 2.22 を実験で用いた。これらの値は基本値として 1.0 を、そして式 (2.9) における $\delta = 1.0, 0.5$, および 0.1 に対応する値としてその他の値を選択した。また k として 100, 1,000, および 10,000 を、 γ として 0.1, 0.01, および 0.001 を比較した。

実験では探索空間とアルゴリズムの性能との関係を観測するために、(深さ, 分枝数) として (4, 4), (2, 16), (4, 4), および (8, 2) の人工木を用いた。人工木中の特徴ベクトルの次元はどれも 8 とした。

図 4.1 にそれぞれのアルゴリズム・パラメータにおける誤答率と average regret をプレイアウト回数ごとに計測した結果を示す。結果は (4, 4) の人工木 100 本の平均である。図から、 $\alpha = 1.0$, $k = 100$, および

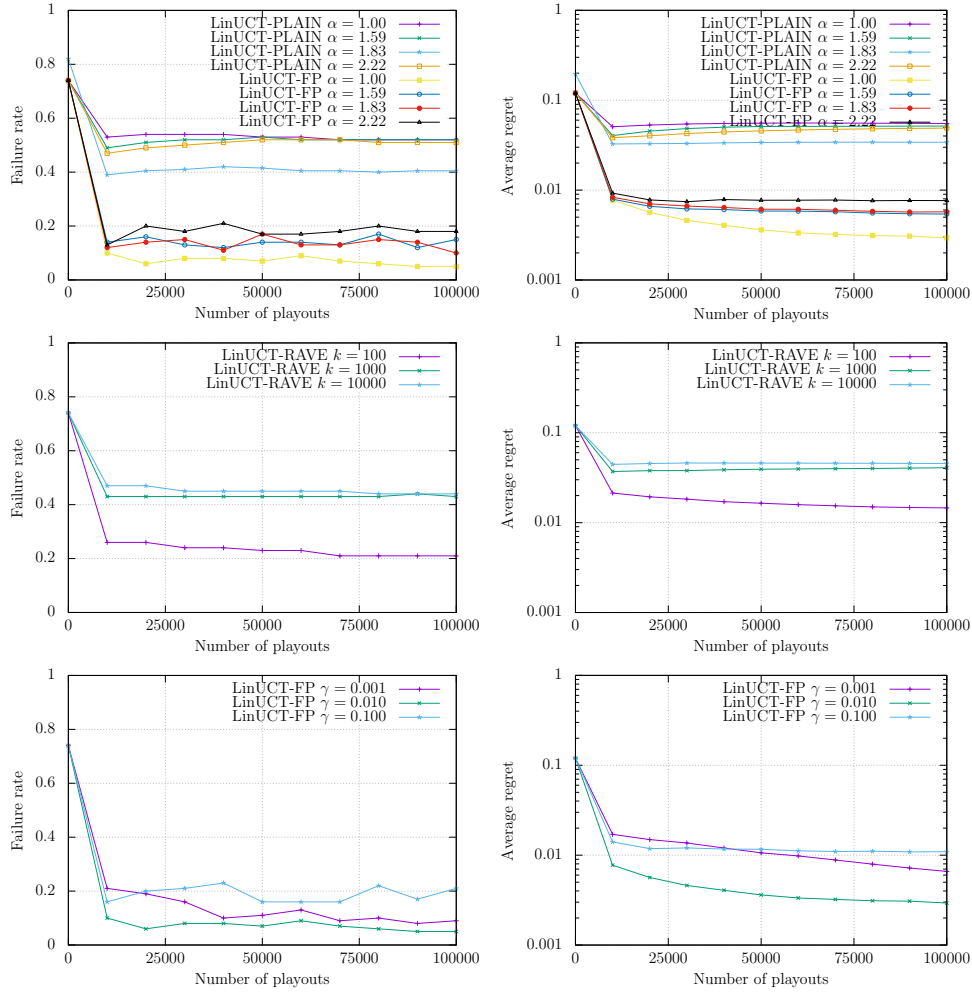


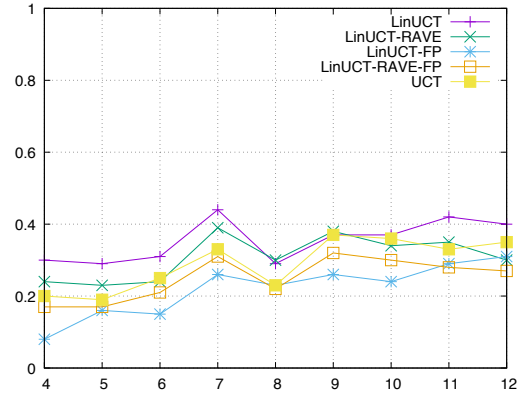
図 4.1: パラメータ α , k , and γ による性能の変化 (depth=4, branching factor=4, $d=8$)

$\gamma = 0.01$ がわずかに他のパラメータよりも性能が良い．よって次の実験ではこのパラメータを用いた．しかしながらパラメータの差異よりもアルゴリズム自体の差異のほうが性能により大きな影響を与えている．

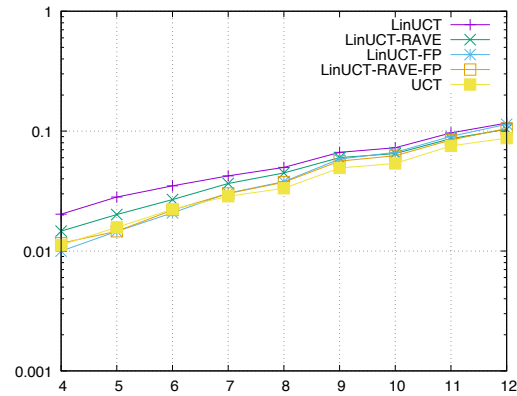
UCT との比較

この実験では UCT と LinUCT との性能の比較を先程の実験と同様誤答率と average regret の観点から観測した．今回は木の深さを 4 に固定し、分枝数を 4 から 12 に変化させつつ性能を評価した．またこの実験では単純な point mutation feature tree だけでなく decayed point mutation feature tree, parent-grandparent mutation feature tree も用いて実験を行った．木のパラメータは $p = 0.1$, $q = 0.2$ (parent-grandparent), $\gamma = 0.85$ (decayed point mutation tree) を用いた．図 4.2-4.4 にそれぞれの木におけるそれぞれのアルゴリズムのプレイアウト回数 10,000 における結果を示す．図において横軸は分枝数を、縦軸は誤答率もしくは average regret を示している．

予想された通り、LinUCT-PLAIN が最も高い誤答率と average regret となっている．この実験で用いた他の LinUCT アルゴリズムについては、誤答率については概ね UCT よりも良い結果を残している．しかしな



(a) Failure rate



(b) Average regret

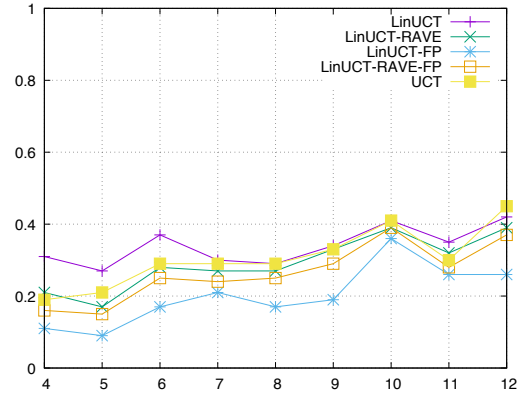
図 4.2: Point mutation feature tree ($p = 0.1$) における UCT と LinUCT の比較 (depth=4, $d=16$, $t = 10,000$)

がら average regret についてはどれも同じような結果となった。これは実験環境である point mutation tree においては最善手と次善手の報酬の差異が比較的小さく、regret に関しては差がつきにくいためであると考えられる。また実験に用いた point mutation tree, decayed point mutation tree, および parent-grandparent tree 間の差異も大きくない。実験では深さ 4 という比較的小さい木を用いたため、それぞれの木の種類で用いる特徴ベクトルの作成において差ができにくく結果的に似た木を生成してしまったためと考えられる。

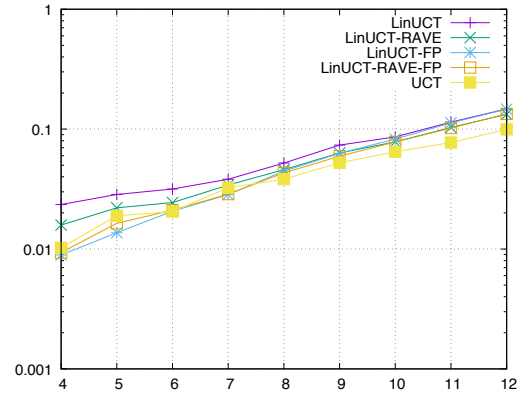
4.2.2 Shock Step Game での評価

先程の実験で比較したアルゴリズムの性能を別の人工的なテストベッドで比較した。ここでは 2.1.2 項で述べた Shock Step Game を用いて評価を行った。

この実験ではそれぞれの列をどの程度の割合でプレイしたかを LinUCT で用いる局面の特徴ベクトルとして用いた。例えば 図 2.2 の盤面で、黒プレイヤーの行動が b2, c3, d4, b5 であれば特徴ベクトルは $(0, 0.5, 0.25, 0.25)$ となる。同様に白プレイヤーの行動に対しても特徴ベクトルを作成し、二つを結合したものを局面の特徴ベクトルとした。初期局面においては零ベクトルを局面特徴ベクトルとして用いた。また先程



(a) Failure rate



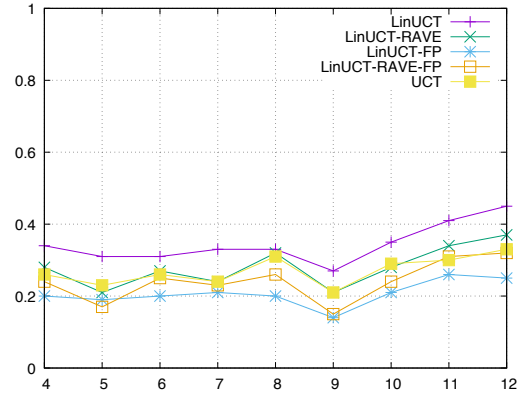
(b) Average regret

図 4.3: Decayed point mutation feature tree ($p = 0.1$, $\gamma = 0.85$) における UCT と LinUCT の比較 (depth=4, $d=16$, $t = 10,000$)

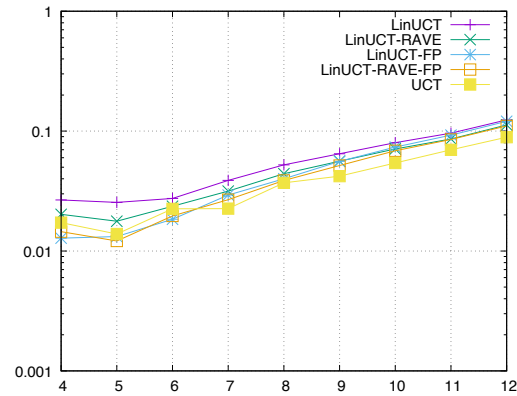
のアルゴリズムに加えて AMAF ヒューリスティックを用いる RAVE プレイヤーも比較している。同じ列の着手を AMAF について同じとみなしている [35]。実験ではプレイアウト回数を 1,000 とし、評価基準として最善着手からどれだけ離れているかを採用した。このゲームにおける最善手は罰金が 0 の列（最左もしくは最右）を選び続けることである。

それぞれのアルゴリズムは最善プレイヤーに対して盤面サイズが 16, 24 のゲーム（スコアはそれぞれ 0 から 8, 0 から 12 になる）をそれぞれ 250 回ずつ行い、ゲーム終了時のスコア（罰金）を計測した。図 4.5 に結果を示す。最善プレイヤーは常にスコア 0 を達成するため、図中の線が左にあるほどよいアルゴリズムである。

最も性能が高かったのは AMAF ヒューリスティックを用いる RAVE であり、LinUCT-PLAIN, UCT と続く。人工木では LinUCT-PLAIN が LinUCT-RAVE, LinUCT-FP よりも性能が悪かったが、このゲームにおいては異なる結果となった。Shock Step Game においては実験で用いた局面特徴ベクトルを用いて完全に回帰予測をすることが可能であるため単純な LinUCT-PLAIN の性能が良かったと思われる。LinUCT-RAVE や LinUCT-FP ではシミュレーションの結果や探索木中の最善手以外の局面の特徴ベクトルを考慮に



(a) Failure rate



(b) Average regret

図 4.4: Parent-grandparent feature tree ($p = 0.1, q = 0.15$) における UCT と LinUCT の比較 (depth=4, $d=16, t = 10,000$)

入れてしまうため性能が下がったと思われる。

4.2.3 Incremental Random Feature Tree での評価

次に, 3.2 節 で述べた incremental random feature tree での評価を行った. 比較したアルゴリズムは UCT と Leaf-LinUCT, および LinUCT アルゴリズムである. 特徴ベクトルの次元は 16 とし, 係数ベクトル θ^* の要素は $\mathcal{U}(-1/\sqrt{6d}, 1/\sqrt{6d})$ に, 行動特徴ベクトルの要素は $\mathcal{U}(-1, 1)$ に従って生成した. またここではベルヌーイ試行による報酬ではなく, 式 (3.1) で計算する局面の評価値をそのまま報酬として与えた.

UCT は報酬の値域が $[0, 1]$ を仮定しているため, UCT の報酬には値域を $\text{clamp}(v/2 + 1/2, 0, 1)$ による線形変換するものとシグモイド関数 $1/(e^{-av} + 1)$ を用いて変換するものを用意し, 別々のアルゴリズムとして比較した. シグモイド関数の定数は $a = 10^{-2}, 10^{-1}, 10^0, 10^1, 10^2$ と, $a = \infty$ (ステップ関数) を比較した. LinUCT のパラメータは 4.2.1 項 で得られたものを用いた.

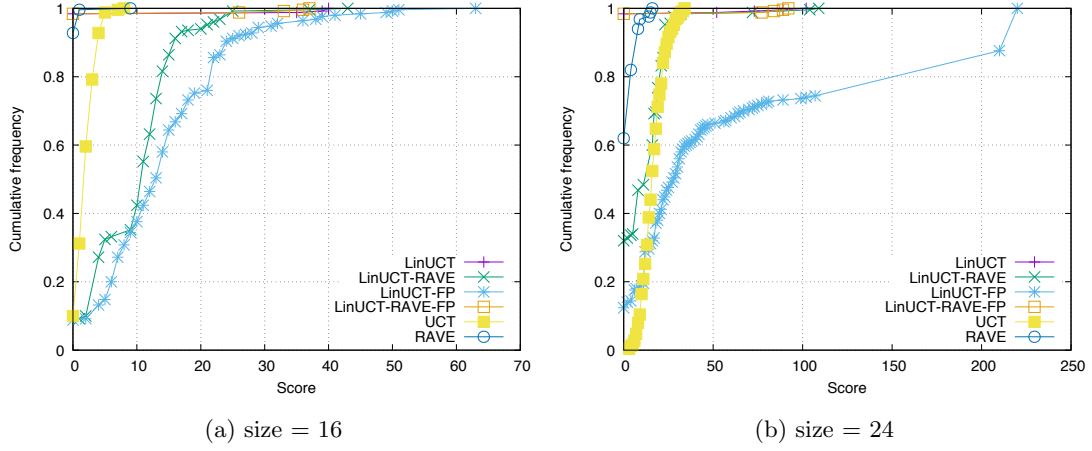


図 4.5: Shock Step Game におけるそれぞれのアルゴリズムのスコアの割合の累積グラフ (250 対局)

アルゴリズムの性能比較

4.2.1 項と同様、誤答率と average regret の二種類の指標をアルゴリズムの性能評価に用いた．図 4.6 と図 4.7 に分枝数を 4 から 16 へ変化させた場合のそれぞれのアルゴリズムの結果を示す．Leaf-LinUCT-WP と Leaf-LinUCT-FP がいくつかの例外はあるが最も良い成績を収めた．シグモイド関数を用いた UCT が分枝数が小さい場合には高性能であるが、分枝数が大きい場合にはうまく働かなかった．探索木の葉局面のみを回帰予測に用いる Leaf-LinUCT のうち、Leaf-LinUCT-PLAIN は結果が良くなく、Leaf-LinUCT-WP、Leaf-LinUCT-FP と比較すると性能が悪かった．このため、報酬予測が向上したとしても前述の静的な局面特徴ベクトルによる探索の問題が大きく影響したものと考えられる．

予測値の正確性

アルゴリズムの性能の差を分析するために、LinUCT アルゴリズムが予測する値 $\mathbf{x}_s^\top \hat{\boldsymbol{\theta}}$ と minimax 値との比較を行った．図 4.8 に 10,000 プレイアウト後における予測値と minimax 値の散布図を示す．100 回以上訪問された局面の値を記載している．予測値が完全に minimax 値と一致していれば点は $y = x$ 上に描画される．

LinUCT-PLAIN (図左上) は深さ 4 (この木の葉) であっても minimax スコアを予測できていない．これは回帰予測に葉局面だけでなく内部局面の特徴ベクトルも用いているため予測がずれてしまっていると考えられる．対照的に Leaf-LinUCT-PLAIN (図右上) は深さ 4 の局面の予測を正しく行うことができている．

LinUCT-FP や LinUCT-WP は LinUCT-PLAIN と比較してより正確な予測ができており、内部局面においてそれが顕著である．内部局面において予測に用いる特徴ベクトルを変化させることで葉局面の予測がより正確にできるということがわかる．LinUCT-FP と LinUCT-WP においては Leaf-LinUCT-FP や Leaf-LinUCT-WP と比較して、この指標では大きな認められなかった．

次に、LinUCT アルゴリズムにおける係数ベクトルの予測値 $\hat{\boldsymbol{\theta}}$ が真の係数ベクトル $\boldsymbol{\theta}^*$ とどれだけ近いかを計測した．図 4.9 にプレイアウト回数と予測値と真値の L_2 ノルムとのグラフを示す．グラフから、Leaf-LinUCT はこの木の設定においては概ね 200 回のプレイアウトを行うと L_2 ノルムが 0 になっている．内部局面の特徴ベクトルも回帰に用いる LinUCT では傾向が異なり、LinUCT-FP, LinUCT-WP は L_2 ノ

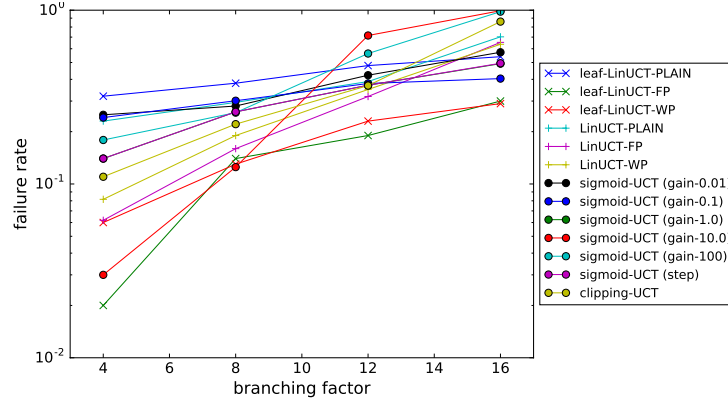


図 4.6: 各アルゴリズムの誤答率 (縦軸) と 分枝数 (横軸) ($t = 10\,000$)

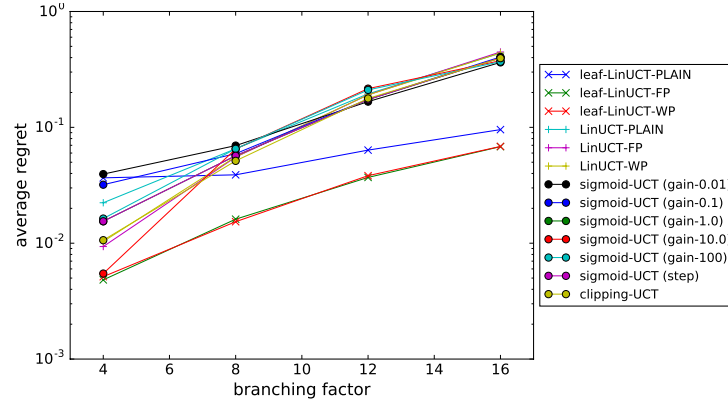


図 4.7: 各アルゴリズムの average regret (縦軸) と 分枝数 (横軸) ($t = 10\,000$)

ルムが 0 へ漸近しているが, LinUCT-PLAIN では減少の後また増加している. LinUCT-PLAIN では局面の特徴ベクトルが定常で, 回帰予測の更新には変化しない特徴ベクトルを用いているため, それがノイズとなってしまうと思われる. また 図 4.10 は PV の葉局面の評価値 $\mathbf{x}_{PV}^T \boldsymbol{\theta}^*$ とその予測値 $\mathbf{x}_{PV}^T \hat{\boldsymbol{\theta}}$ との平均誤差をプレイアウト回数ごとに観測したものである. 図 4.9, 4.10 から, 葉局面のみを使う LinUCT のほうがよい性能であることがわかる.

4.2.4 Progression game での評価

最後に, Leaf-LinUCT と UCT の性能を Finnsson と Björnsson の progression game において比較した. 局面の特徴ベクトルとして, (1) ratio feature と (2) raw feature の二つを用いた.

Ratio feature では盤面上の駒の進んだ割合を局面の特徴ベクトルとした. Raw feature では盤面のマスの数だけ次元を持ち, あるマスに駒があるならば非ゼロの値をもつ. どちらの場合も特徴ベクトルの大きさが 1 になるように正規化している.

各アルゴリズムは最善プレイヤーと 1,000 回対戦を先手番で行い, プレイアウト回数は 10,000 回とした. 先手番プレイヤーであるため, 一度以上最善手でないものを選択すると敗北する. 表 4.1 にそれぞれのアルゴリズムの成績を示す. Ratio feature のアルゴリズムについてはどのアルゴリズムも UCT 以上の成績になった. しかしながら raw feature のアルゴリズムでは Leaf-LinUCT-WP 以外のアルゴリズムは UCT よりも

表 4.1: Progression Game での各アルゴリズムの結果

Algorithm	Feature Vector	#Wins	Win%
leaf-LinUCT-PLAIN	ratio-feature	2 000	100
leaf-LinUCT-FP		2 000	100
leaf-LinUCT-WP		1 342	67
leaf-LinUCT-PLAIN	raw-feature	55	3
leaf-LinUCT-FP		30	2
leaf-LinUCT-WP		1 677	84
UCT		801	40

悪い。

Leaf-LinUCT-WP はある局面の評価値が子局面の評価値の平均となるため、このようにどの局面も一見有望に見えてしまうようなゲームでは正しく最善手を選択することがより難しくなっていると考えられる。そのため他のアルゴリズムと比較してより性能が悪くなってしまったと思われる。

4.3 まとめ

本章では線形バンディット問題のアルゴリズムの一種である LinUCB をモンテカルロ木探索に応用した LinUCT を提案し、その性能を 第 3 章 で提案したモデルおよび人工的なゲームにおいて評価した。まず単純に LinUCB をモンテカルロ木探索へ適用した LinUCT-PLAIN について観察し、そのままではゲーム木探索に適していないということを実験的に観察した。ゲーム木探索に適用するために、変更を加えたアルゴリズムをいくつか提案し、その特性についても実験的に観察した。

得られた結果から、LinUCT は事前に学習が行えないような状況において効率的にゲームの知識を獲得し、探索に役立てることができると推察される。今回はいずれも人間が遊ぶようなゲームではなく、評価が行いやすいモデルを用いて実験を行った。今後の課題では実在のゲームや General Game Playing などにおいて評価をすることが挙げられる。

今回の LinUCT が予測した値は局面から行ったシミュレーションの期待値である。これは局面価値関数とほぼ同義であるが、AlphaGo では局面価値関数だけでなく方策関数も用いている。確率的バンディット問題にも確率一致法という、方策に基づいたアルゴリズムがいくつか提案されており、例えば Thompson sampling [59] は各行動についてその行動が最適である確率を推定し、それに比例して行動を選択する。そのようなアルゴリズムを利用して方策関数を予測して探索に役立てるようなアルゴリズムも今後の課題である。

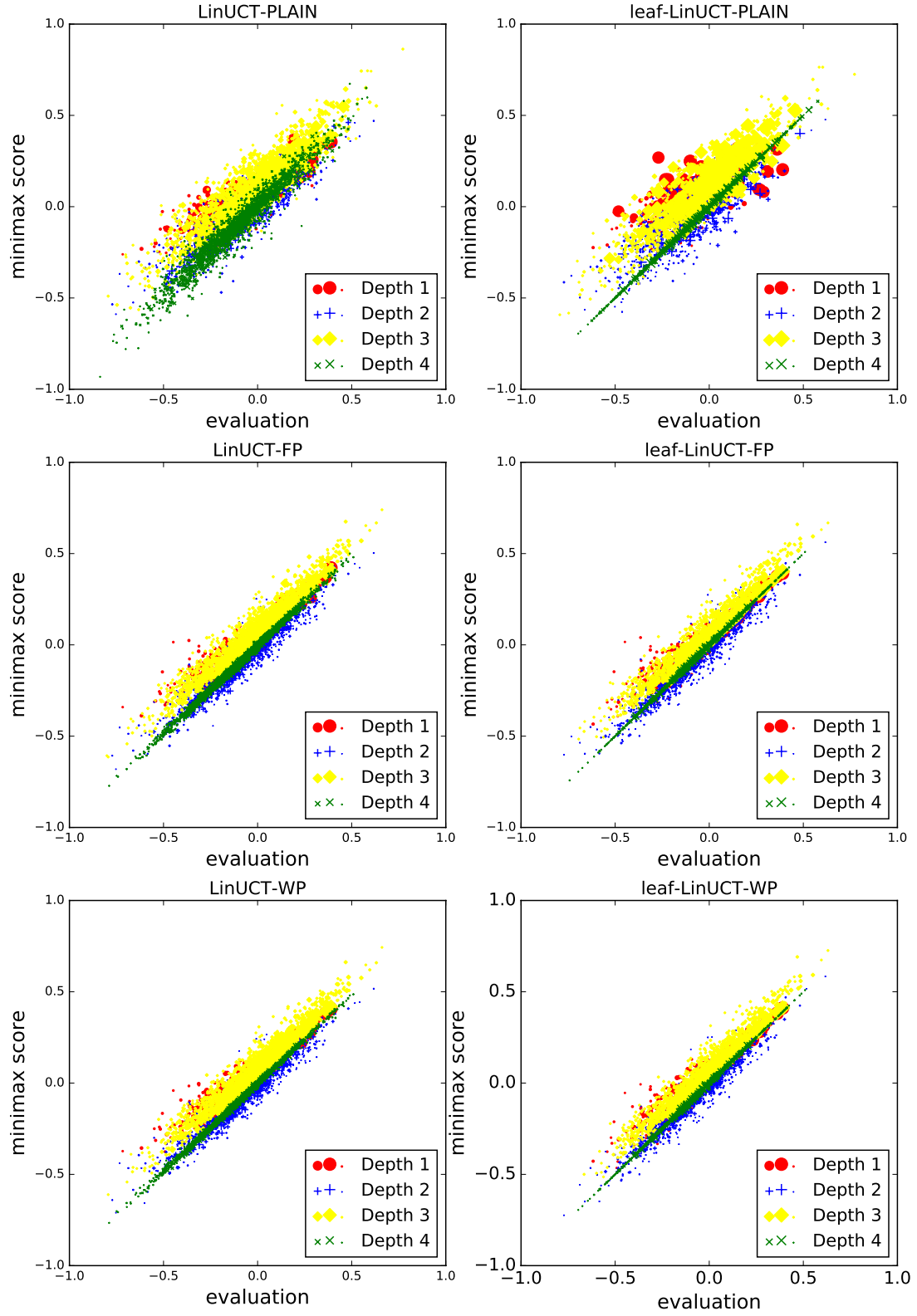


図 4.8: Minimax 値 (縦軸) とアルゴリズムが予測した評価値 (横軸) の散布図 (10 000)

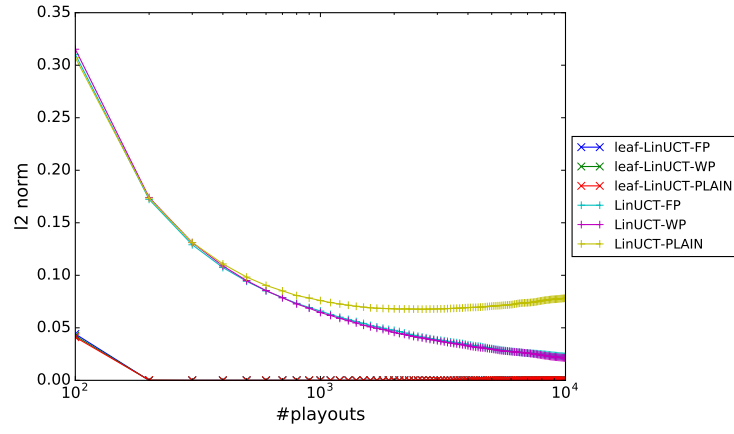


図 4.9: $\|\theta^* - \hat{\theta}\|_2$ の時間変化 (log-scale, depth = 4, branching factor = 4)

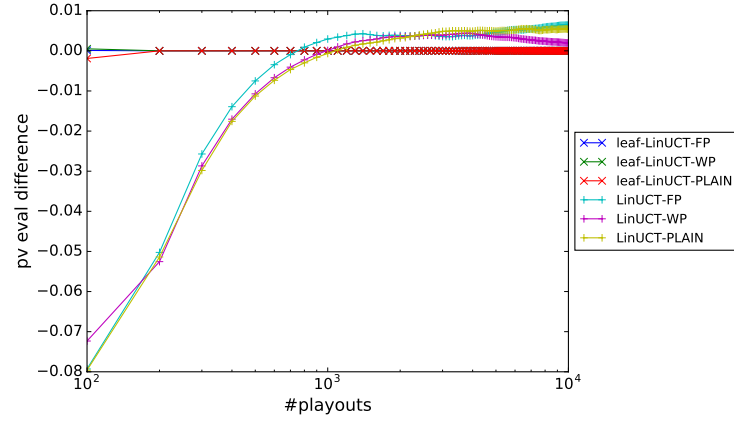


図 4.10: $\mathbf{x}_s^\top \theta^* - \mathbf{x}_s^\top \hat{\theta}$ の時間変化 (log-scale, depth = 4, branching factor = 4)

第 5 章

LinUCT の時間計算量向上

第 4 章 で提案した LinUCT は単純にアルゴリズムを適用すると特徴ベクトルの二乗に比例する時間計算量が各反復ごとに必要となる。ゲームにおける評価関数では大規模なもので数万から数百万 [5] の特徴ベクトルの次元を必要とするものがあり、そういった特徴ベクトル空間の設計を行うと現実的な時間での計算が終わらないという問題が生じる。そこで本章では確率的勾配降下法を用いた fLinUCB-GD [52] を LinUCT に応用し、特徴ベクトルの次元が数千次元まで拡大しても現実的な時間で探索をすることができることを示す。またそのときのアルゴリズムの性能についても議論する。

5.1 fLinUCT-GD

fLinUCB-GD は確率的勾配降下法を用いて報酬を予測する LinUCB アルゴリズムであり (2.4.2 項), 実世界の問題において LinUCB と同程度の性能を示すとされている [52]。よって前章で述べた LinUCT アルゴリズムに適用しても性能を保ったまま時間計算量を改善できると予想される。fLinUCT-GD の基本的なアルゴリズムを アルゴリズム 6 に示す。アルゴリズム中の `Eval()` は入力を用いてモンテカルロ木探索の選択の基準値を計算する関数である。この場合、モンテカルロ木探索の一反復にかかる時間計算量は特徴ベクトルの次元を d とした場合 $O(d)$ であり、LinUCT の場合の $O(d^2)$ と比較して大きく向上している。また全体の計算量は LinUCB を用いる LinUCT-PLAIN が一反復中の平均訪問局面数 (探索木の平均深さ) を \tilde{n} としたときに $O(T\tilde{n}d^2)$ であるのと比較して、fLinUCT-PLAIN は $O(T\tilde{n}d)$ で行うことが可能である。

また前章で述べた LinUCT アルゴリズムの変種も回帰予測の計算を確率的勾配降下法によって行うことが可能である。この章では主に計算速度に関して述べるため、LinUCT-RAVE を用いた fLinUCT-RAVE のみを fLinUCT-PLAIN とともに比較した。

5.2 実験環境

この章における実験は 表 5.1 の環境で行った。線形代数計算は Boost.uBLAS ^{*1} を用いて実装を行った。

^{*1} http://www.boost.org/doc/libs/1_55_0/libs/numeric/ublas/doc/index.htm

アルゴリズム 6 fLinUCT-GD

```
Initialization: Set  $\hat{\theta}_0$  arbitrarily
for  $t = 1, 2, \dots, T$  do
  Approximate  $\hat{\theta}_t$  using (2.10)
   $i \leftarrow 0$ 
   $path[i] \leftarrow$  root node
  while  $path[i]$  is not leaf nor terminal node do
    for  $a$  in all children of  $path[i]$  do
      Estimate confidence parameter  $\phi_{t,a}$  using (2.11)
       $p_a \leftarrow \text{Eval}(\mathbf{x}_a, \hat{\theta}_t, \phi_{t,a})$ 
    end for
     $path[i+1] \leftarrow \arg \max_a p_a$ 
     $i \leftarrow i+1$ 
  end while
  if  $path[i]$  should be expanded then
    Expand  $path[i]$ 
  end if
  Simulate rest of the game from  $path[i]$  and observe reward  $r_t$ 
  Store reward  $r_t$  and the feature vectors in  $path$ 
end for
```

表 5.1: 実験環境

OS	Debian 3.16.0-4-amd64
CPU	Intel®Xeon™E5645@2.40GHz (5.3 章)
	Intel®Core™i7-4790K@4.00GHz (5.4 章)
メモリ	16GB (5.4 章)
	48GB (5.3 章)
コンパイラ	g++ 4.9.2
g++ オプション	-O3 -DNDEBUG -march=native
Boost	1.55

5.3 Point Mutation Feature Tree における性能

はじめに各アルゴリズムの性能評価を point mutation feature tree を用いて行った．ここでは各手法の評価を誤答率と実行速度の二つの側面から行った．時刻 t において探索を打ち切ったとした場合に選ばれる着手について、それが最善手と異なることを誤答と定義する．時刻 t における誤答率はいくつかの実行結果の平均で算出する．各アルゴリズムの定数は後述の表 5.4 と同様であるが、モンテカルロ木探索における展開の閾値は 1 としている．

表 5.2: 各アルゴリズムの $t = 10,000$ 時点での誤答率 (分枝数 4, 深さ 4, () 内は 95% 信頼区間の幅)

特徴ベクトルの次元	8	32	128
LinUCT-PLAIN	24% (8.4%)	30% (9.0%)	16% (7.2%)
LinUCT-RAVE	11% (6.1%)	25% (8.5%)	16% (7.2%)
fLinUCT-PLAIN	28% (8.8%)	25% (8.5%)	18% (7.5%)
fLinUCT-RAVE	13% (6.6%)	15% (7.0%)	12% (6.4%)
UCT	16% (7.1%)	12% (6.4%)	13% (6.5%)

表 5.3: 各アルゴリズムの $t = 10,000$ 時点での誤答率 (分枝数 8 深さ 4, () 内は 95% 信頼区間の幅)

特徴ベクトルの次元	8	32	128
LinUCT-PLAIN	41% (9.6%)	34% (9.2%)	24% (8.4%)
LinUCT-RAVE	37% (9.3%)	34% (9.3%)	21% (8.0%)
fLinUCT-PLAIN	38% (9.5%)	30% (9.0%)	24% (8.4%)
fLinUCT-RAVE	24% (8.4%)	24% (8.3%)	22% (8.1%)
UCT	32% (9.1%)	29% (8.9%)	25% (8.5%)

木は (分枝数, 深さ) が (4, 4), (8, 4), 特徴ベクトルの次元が 8, 32, 128 であるような組み合わせの六種類の木を 100 本ずつ作成した。計測結果は一つの木に対して一回アルゴリズムを実行してすべての木について平均をとったものである。局面の特徴ベクトルの変異確率 p は 0.1 とした。表 5.2, 5.3 に $t = 10,000$ 時点での各アルゴリズムの誤答率を示す。図 5.1, 5.2 は 10,000 プレイアウトするのにかかった時間の平均である。

誤答率については、実験を行った回数が少ないことからどのアルゴリズムも有意に差があるとは言えないが、傾向として同じような性能を発揮することがわかった。また LinUCT, fLinUCT-GD とともに PLAIN よりも RAVE のほうが性能がよい傾向にある。また特徴ベクトルの次元で性能が異なるのは、次元によって加算される要素の数が異なり、評価値の分布が異なるからであると考えられる。実行時間を考えると、より少ない時間であるにもかかわらず fLinUCT-GD が LinUCT と同程度の結果を残すことがわかり、計算時間の面で有利であることが伺える。

5.4 囲碁における性能

fLinUCT-GD が UCT と比較してどのような性能となるかを調査するため、対戦実験を行った。本節の実験はすべて 9 路盤で行った。

また本稿で用いる特徴の次元数は 2000 次元程度としたため、もともとの LinUCB を用いる LinUCT-PLAIN, LinUCT-RAVE では現実的な時間で探索が終了しない。よって本稿ではこれらは扱わない。

5.4.1 実装

実装した UCT, fLinUCT-PLAIN, fLinUCT-RAVE は、通常の囲碁プログラムでの評価に近づけるため progressive widening [60] と特徴を用いたプレイアウト中の行動選択 [30] を用いている。Progressive

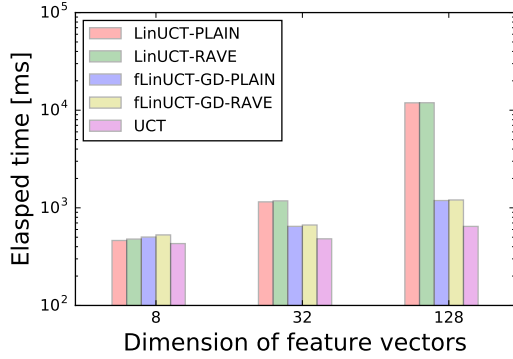


図 5.1: 分枝数 4, 深さ 4 の木における特徴ベクトルの次元と実行時間

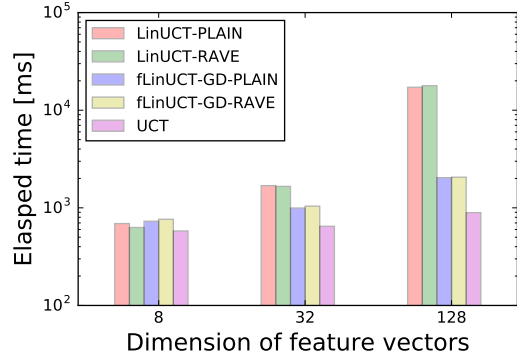


図 5.2: 分枝数 8, 深さ 4 の木における特徴ベクトルの次元と実行時間

表 5.4: 実験におけるパラメータ

パラメータ名	値	関係するアルゴリズム
γ_t	$\frac{1}{100+t}$ [52]	fLinUCT-GD
λ_t	$\frac{1}{t^{1-\alpha}}$ [52]	fLinUCT-GD
α	0.6 [52]	fLinUCT-GD
κ	1.0	fLinUCT-GD
k	1000	fLinUCT-RAVE
展開の閾値	10	MCTS

widening とプレイアウト中の行動選択確率 $\pi(s, a)$ で用いる特徴と重みは囲碁プログラム Libego *² が使用している 3×3 の石の配置パターンと, 8 近傍点のアタリの情報 (呼吸点の数が 1 または > 1) の特徴を使用した. プレイアウト中の行動選択確率は softmax 法で決定している.

各アルゴリズムのパラメータについては表 5.4 のとおり設定した.

5.4.2 特徴

囲碁においては先行研究において様々な特徴が用いられてきた ([61, 32] など). しかしそれらのほとんどは局面の特徴ではなく着手の特徴であり, 勝率の期待値が特徴ベクトルの線形結合で表されるという LinUCB の仮定に沿わないと考えられる. そこで今回は既存の研究で示されているような局所的な特徴ではなく, 盤面をそのまま表現するような特徴ベクトルの設計を行った. 具体的には以下のとおりである.

- 8 近傍の石の組み合わせ ($81 \times 8 \times 3 = 1944$ 次元)
- 定数 (1 次元)

8 近傍の石の組み合わせとは, 8 近傍の座標それぞれについて中心の石と同色, 異色, 空白もしくは盤

*² <https://github.com/lukaszlew/libego>. Accessed on 2018/12/10.

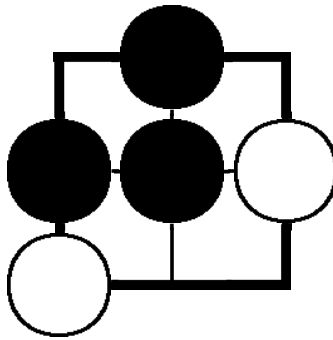


図 5.3: 特徴ベクトルの例: 中心の石と同色, 異色, 空白もしくは盤外の三つに各ビットが対応する. この例では $(0, 0, 1|1, 0, 0|0, 0, 0|1, 0, 0|0, 1, 0|0, 1, 0|0, 0, 0|0, 0, 0)$ が特徴ベクトルの部分ベクトルになる. 9 路盤ではこれが $9 \times 9 = 81$ 個存在する

外の 3 ビットずつの情報を持ち, 9 路盤においては 8 近傍 \times 81 点 \times 3 種類の合計 1944 次元の特徴をもつ特徴ベクトルである. 例として, 図 5.3 における 8 近傍の石の組み合わせを表すベクトルは, $(0, 0, 1|1, 0, 0|0, 0, 0|1, 0, 0|0, 1, 0|0, 1, 0|0, 0, 0|0, 0, 0)$ となる. 囲碁の局面の特徴から勝率を予測する試みは深層畳み込みニューラルネットワークを用いた研究において最近大きな成功が報告された [9]. 特徴の設計が適切であれば, 囲碁においても局面特徴からの勝率予測も有効と考えられる.

5.4.3 UCT との比較

これまで述べた二つのアルゴリズムについての性能評価として, 既存手法である UCT との対戦実験を行った.

はじめに, 人工木での実験と同様にプレイアウト回数による性能の違いを計測するため, プレイアウト回数を同じに設定したアルゴリズム同士の対戦を行い, 勝率と信頼区間を計測した. 次に高速化を施した fLinUCT-PLAIN, fLinUCT-RAVE が UCT と比較してどの程度の速さで動作するかを確認した.

プレイアウト回数での比較

fLinUCT-PLAIN, fLinUCT-RAVE と UCT とを同一プレイアウト回数の条件の元で対戦させた結果を表 5.5, 5.6 に示す.

fLinUCT-PLAIN は UCT と比較していずれのプレイアウト回数でも劣るという結果が得られた. しかしプレイアウト回数を増やすことによって勝率が向上するという結果が得られた. fLinUCT-RAVE と UCT との対戦では, いずれのプレイアウト数でも統計的な有意差は見られなかったが, fLinUCT-PLAIN と比較して性能が向上していることが確認できる.

実行時間での比較

表 5.7, 5.8 にそれぞれのアルゴリズムの一試合あたりの実行時間の比較と, UCT を 1 としたときの fLinUCT-PLAIN, fLinUCT-RAVE の実行時間を示す.

UCT と比較して, fLinUCTB-GD を用いた fLinUCT-PLAIN, fLinUCT-RAVE はおよそ定数倍のオー

表 5.5: 同一プレイアウト回数下での UCT に対する fLinUCT-PLAIN の対戦成績

プレイアウト数	勝率	信頼区間	対戦回数
1000	0.268	± 0.045	100
2000	0.258	± 0.044	100
4000	0.371	± 0.049	100
8000	0.433	± 0.050	100

表 5.6: 同一プレイアウト回数下での UCT に対する fLinUCT-RAVE の対戦成績

プレイアウト数	勝率	信頼区間	対戦回数
1000	0.440	± 0.068	100
2000	0.474	± 0.046	100
4000	0.502	± 0.049	100
8000	0.521	± 0.051	100

表 5.7: fLinUCT-PLAIN と UCT との一試合あたりの実行時間 (秒) の比較

プレイアウト回数	1000	2000	4000	8000
UCT	14.3	28.7	58.6	116.6
fLinUCT-PLAIN	32.5	68.8	148.4	316.9
fLinUCT-PLAIN/ UCT	2.27	2.40	2.53	2.78

表 5.8: fLinUCT-RAVE と UCT との一試合あたりの実行時間 (秒) の比較

プレイアウト回数	1000	2000	4000	8000
UCT	13.6	28.4	59.4	129.6
fLinUCT-RAVE	23.1	46.2	97.1	204.3
fLinUCT-RAVE/ UCT	1.70	1.63	1.63	1.57

ダーで計算を行うことが可能であることがわかる。fLinUCT-PLAIN と fLinUCT-RAVE を比較すると、fLinUCT-PLAIN のほうが時間がかかっていることがわかる。表 5.9 は開始局面から 1 秒間探索を実行した場合の平均プレイアウト回数を示している。この表からも fLinUCT-PLAIN と比較して fLinUCT-RAVE のほうがより多くのプレイアウトを実行することができている。これはそれぞれのアルゴリズムの探索木の形状の傾向によってこのような差が生まれると思われるが、詳細な調査は今後の課題である。

5.4.4 同一思考時間における比較

最後に同一の条件下での既存手法との比較を行った。fLinUCT-PLAIN, fLinUCT-RAVE と UCT, UCT-RAVE とを思考時間を 5 秒に揃えて対戦させた。表 5.10, 5.11 に二つのアルゴリズムの結果を示す。

表 5.9: 初期局面における一秒あたりの平均プレイアウト回数の比較 (100 回試行の平均)

アルゴリズム	プレイアウト回数/秒
UCT	2246.19
fLinUCT-PLAIN	632.82
fLinUCT-RAVE	952.15

表 5.10: 同一思考時間 (5 秒) における UCT に対する勝率

アルゴリズム	勝率	信頼区間	対戦回数
fLinUCT-PLAIN	0.173	0.038	100
fLinUCT-RAVE	0.270	0.044	100

表 5.11: 同一思考時間 (5 秒) における UCT-RAVE に対する勝率

アルゴリズム	勝率	信頼区間	対戦回数
fLinUCT-PLAIN	0.150	0.036	100
fLinUCT-RAVE	0.210	0.041	100

実行時間を同一にした場合は UCT, UCT-RAVE と比較して大きく劣ることがわかる. UCT との実行時間を考えると数倍のプレイアウトの差が存在し, その差を埋めるためより正確な評価が可能となる特徴の設計や, プレイアウト回数を増やせる次元数の少ない特徴の設計などを行う必要がある.

5.5 まとめ

本章では確率的勾配降下法を用いた fLinUCB-GD を利用し計算量を削減した LinUCT を実装し, ある程度大きな特徴空間であっても現実的な時間で探索が行えることを示した. また性能についても ridge regression を行うもとの LinUCT と遜色ないことを実験で示した.

囲碁においては既存手法である UCT, UCT-RAVE と比較して fLinUCT-GD の優位性を示すことができなかった. また同一計算時間を用いた場合には大きく劣ることが確認できた. 今後の課題として, 特徴空間の設計などを見直してより良い性能を目指すことが第一に挙げられる. 今回用いた特徴は単純な石の配置だけで, 勝敗に結びついていなかった可能性が高い.

今後は 第 4 章 で述べたようなアルゴリズムも fLinUCB-GD を利用して高速化し, ここで述べたような実験を行い評価を行う必要がある.

第 6 章

RankNet による効率的な評価関数の学習

これまでの章では、ゲームに関する知識が事前に得られないという前提で、探索中に知識を獲得しそれを探索に活かすアルゴリズムについて述べた。この章と次の章では、事前にゲームに関する情報を強いプレイヤーの棋譜という形で得られるようなゲームについて、そのゲームに関する知識を評価関数の学習という形で効率よく獲得する方法について論ずる。

そのような設定では、利用できる情報をうまく活用し、最大限情報が得られるようにすることでよりよい知識を獲得できることを示す。まず、通常の学習とは異なり、棋譜同士の比較を行うことでより効率的な学習が行えることを示す。具体的には、learning-to-rank 学習で用いられる RankNet をゲームの評価関数に応用し、RankNet によって得られた学習結果が通常の学習結果と比較して実際のゲームにおける強さの面で優れていることを示す。

6.1 RankNet による評価関数の学習

この節では 2.5.2 項 で述べた RankNet をゲームの評価関数の学習に応用する際の具体的なアルゴリズムについて示す。まず、利用可能なゲームの棋譜を $\langle s, \pi, r \rangle$ の局面集合に変換する。ここで利用するゲームの棋譜は十分強いプレイヤーの棋譜でもよいし、AlphaGo Zero のように自己対戦の棋譜であってもよい。 s はゲームの局面を予測器に入力可能な形式にしたもので、例えば AlphaGo などでは局面をテンソルの形にしたものである。 $\pi \in \mathbb{R}^d$ はその局面で取るべき行動を表したベクトルである。 d は対象となるゲームにおける取りうるすべての行動の数で、19 路盤の囲碁であれば 362 となる (パスを含む)。ベクトルの各要素は対応する番号の行動をその局面で取るべき確率であり、強いプレイヤーの棋譜からの学習などの教師あり学習では一つの要素だけが 1 でそれ以外が 0 のベクトルとなる。AlphaGo Zero における学習では、自己対戦中のモンテカルロ木探索の訪問回数に比例する値になっている。 r は報酬を表し、手番プレイヤーがその対戦で勝利したならば +1、敗北したならば -1、引き分けならば 0 となる。

その後変換した局面集合を三つの集合 W , L , および D に分割する。 W は $r = 1$ の棋譜、 L は $r = -1$, D は $r = 0$ の訓練対である。

学習に用いる訓練例は先に述べた $\langle s, \pi, r \rangle$ の対である。一つの例は部分集合 $W \cup D$ から、もう一つの例は $L \cup D$ から一様ランダムに抽出する。具体的には、ある一つの訓練例は $\langle s_i, \pi_i, s_j, \pi_j, S_{ij} \rangle$ となる。ここで i, j は抽出した二つのタプルを表し、順序はランダムに決定する (つまり i が $W \cup D$ かもしれないし、 $L \cup D$ かもしれない)。ラベル S_{ij} は i が j よりも優れているならば +1 を、 j が i よりも優れているならば -1 を表す。 i, j どちらも同じであれば (どちらも D から抽出された局面であれば) $S_{ij} = 0$ とする。

以上のように訓練例を生成し、学習を行う。学習する予測器は訓練例から (v_i, \mathbf{p}_i) と (v_j, \mathbf{p}_j) を計算し、以下の損失関数に従って勾配を計算する。

$$L_{\text{POLICY-RANKNET}} = \sum_{i,j} \underbrace{\frac{1}{2}(1 - S_{ij})\sigma(v_i - v_j) + \log(1 + e^{-\sigma(v_i - v_j)})}_{\text{RankNet loss}} - \underbrace{\frac{1}{2}(\boldsymbol{\pi}_i^\top \log \mathbf{p}_i + \boldsymbol{\pi}_j^\top \log \mathbf{p}_j)}_{\text{policy loss}}. \quad (6.1)$$

この式は RankNet による局面価値の損失と、着手予測に対する交差エントロピーの二つの項からなる。

以上の RankNet による評価関数の学習は多くのゲームに適用できると考えられるが、本章では主に囲碁を対象としてその評価を行う。また着手予測に関する損失を入れなくとも以上で述べた学習アルゴリズムは適用可能であるが、下記の実験で着手予測を含めたほうが対戦実験における性能がよいこと示す。これは AlphaGo Zero のように着手と局面価値両方を予測する予測器のほうが学習において正則化の効果が得られより強固な予測器が学習できるという知見を裏付ける。

関連する研究として、局面の比較という観点では 2.3.2 項 で述べた DeepChess によるチェスの学習が比較対象として挙げられる。DeepChess は入力が二つ存在する予測器を用いることで利用可能な局面の数に対しておおよそ二乗の数の訓練例をもって学習できる。しかしながら DeepChess が用いた予測器を現在広く使われている探索アルゴリズムとともに用いることは容易では無いと考えられる。事実 DeepChess が用いている探索アルゴリズムは変則的な $\alpha\beta$ 探索であり、囲碁で広く用いられているモンテカルロ木探索に単純に適用することは困難であると思われる。ここで提案した RankNet による学習では、予測器は単一の局面に対し単一の評価値と単一の着手予測を出力できればよく、DeepChess 方式の学習と比較してより一般的な予測器を扱える。またモンテカルロ木探索などの既存の探索アルゴリズムと組み合わせて用いることも容易である。

6.2 評価実験

この節では RankNet によって学習した予測器と既存のアルゴリズムによって学習した予測器との比較を行う。実験対象としたゲームは 9 路盤の囲碁である。

6.2.1 学習棋譜と実装

RankNet による学習も AlphaGo Zero と同様に自己対戦による強化学習が可能であると考えられる。しかしながら自己対戦による強化学習には膨大な量の計算資源が必要となるため、強いプレイヤーの棋譜による教師あり学習によって学習を行った。学習に用いた棋譜は aya の自己対戦棋譜であり、CGOS^{*1} においておおよそ 2,900 BayesElo rating の強さである。Aya による自己対戦棋譜を重複なしで 991,680 局分生成し、全部で 97,606,193 局面を学習局面として得た。自己対戦はコミ 7.0 で行ったため、棋譜中に引き分けが存在する。人間同士の対戦では引き分けをなくすためにコミ 7.5 で行われることが多いが、コミ 7.5 では白番が有利であると広く認識されており、また他のゲームにおいては引き分けが避けられないようなもの（チェスなど）が存在するためこの設定とした。

実験では学習に用いる棋譜の量による性能の変化を観察するため、データセットを 0.25M (250,000), 0.50M (500,000), 0.75M (750,000), および 1.00M (1,000,000) の 4 つ用意しそれぞれの棋譜量での性能を評価した。全 97,606,193 局面から validation 用の 1,000,000 局面を除いた 876,606,193 局面からランダムにそれぞれの量だけの局面を抽出してデータセットを用意した。表 6.1 にそれぞれのデータセットの内訳を示す。

^{*1} Computer GO Server <http://www.yss-aya.com/cgos/>. Accessed on 2018/12/10.

表 6.1: 各データセットの内訳

	$ W $	$ L $	$ D $	No. of possible combinations
0.25M	99,677	100,103	50,220	2.3×10^{10}
0.50M	200,202	199,526	100,272	9.0×10^{10}
0.75M	300,258	299,365	150,377	2.0×10^{11}
1.00M	400,384	399,251	200,365	3.6×10^{11}
Validation	400,646	399,556	199,798	3.6×10^{11}

表 6.2: 実験で用いたニューラルネットワーク

	training input	loss function	output
BASELINE	$\langle s, \pi, z \rangle$	式 (2.3)	$\langle p, v \rangle$
POLICY-RANKNET	$\langle s_i, \pi_i, s_j, \pi_j, S_{ij} \rangle$	式 (6.1)	$\langle p, v \rangle$
RANKNET	$\langle s_i, s_j, S_{ij} \rangle$	式 (2.16)	v

実験では BASELINE, POLICY-RANKNET, および RANKNET の三つの予測器を学習した. BASELINE は AlphaGo Zero で用いられたニューラルネットワークと同様の構造をしている. 20 層の residual block と value-head, policy-head から構成されている. 学習に用いる損失関数も AlphaGo Zero で用いられたものと同様であるが, 今回は教師あり学習であるため, 着手予測の教師例として one-hot のベクトルを与えている.

POLICY-RANKNET はニューラルネットワークの構造は BASELINE と同様であるが, 損失関数が本章で提案した 式 (6.1) となっている. 最後の RANKNET は RankNet 方式で学習するが, ニューラルネットワークの出力として局面価値予測しがなく, 着手予測を行わない.

表 6.2 に用いたネットワークをまとめたものを示す.

予測器に局面を入れる際には AlphaGo Zero と同様の符号化を行った. すなわち碁盤上の石の配置を 7 手前からそのまま符号化した $9 \times 9 \times 16$ の配列 (8 局面分 \times 黒・白の 2 色) と手番を表す 9×9 の配列を結合した $9 \times 9 \times 17$ の配列を入力する.

また RankNet による学習においては 1 epoch を 1,000,000 訓練例とみなした. これは 1 epoch を定義通り計算すれば 1.00M データセットの場合には 3.6×10^{11} 訓練例となるが現在の一般的な計算機でこの数の訓練例を計算することは困難であるため便宜的に定めた.

学習は 表 6.3 に示す通り, 二種類の設定を用いた. CONF1 は AlphaGo Zero において用いられた学習の設定とほぼ同じ [10] であり, CONF2 は DeepChess における設定 [43] である. しかし CONF1 のバッチサイズの大きさでは GPU メモリ容量の制約上 RankNet 方式の学習が困難であるため, CONF1 での学習を行わなかった. すなわち, 実験では BASELINE-CONF1, BASELINE-CONF2, POLICY-RANKNET-CONF2, および RANKNET-CONF2 の四種類のネットワークをそれぞれのデータセットで学習した.

表 6.3: 学習の設定

	CONF1	CONF2
Optimizer	Momentum SGD	Momentum SGD
Learning rate	0.02	0.01
Momentum	0.9	0.9
Batch size	1024	128
Learning rate decay factor	—	0.99
Learning rate decay steps	—	every 1 million examples
L_2 coefficient	10^{-4}	10^{-4}

6.2.2 学習結果

図 6.2–6.5 に学習中の validation set における各損失を示す。BASELINE と POLICY-RANKNET の policy loss (図 6.2a, 6.3a) は概ね大きな変動をせずデータセットの大きさに比例してある程度の epoch 数まで減少をしたのち損失が上昇する傾向にある。ここで、POLICY-RANKNET の 1 epoch は 1,000,000 訓練例であるため、epoch で比較すると 0.25M のデータセットでは BASELINE と比較して損失が大きく見えるが、図 6.5c で分かる通り学習に用いた訓練数で見た場合は同様の結果が伺える。

一方で BASELINE における value loss (図 6.2b) と POLICY-RANKNET における rank loss (図 6.3b) は大きく振動しているように見える。しかしスケールが policy loss と異なるため、大きく変動しているわけではなく、どのデータセットでも減少傾向が見られ、学習はどちらも成功していると考えられる。同様の傾向が RANKNET (図 6.4) にも見られる。

学習結果のまとめとして、どのモデルも損失関数の値は減少することが確認でき、学習自体は成功していると思われる。

6.2.3 対局実験

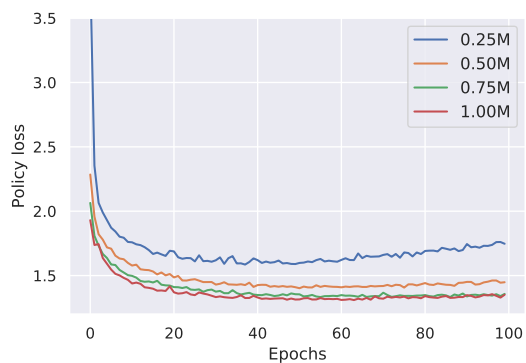
次に学習したニューラルネットワークの実際の対局における性能を比較した。対戦相手として Pachi [7] version 12.00-jowa^{*2} を用いた。Pachi は一手あたり 10,000 プレイアウトを行い、DCNN は使わない設定とした。

ニューラルネットワークで用いる重みは validation set での結果が最も良かった epoch の重みを用いた。表 6.4 に最良の重みの epoch 数を示す。

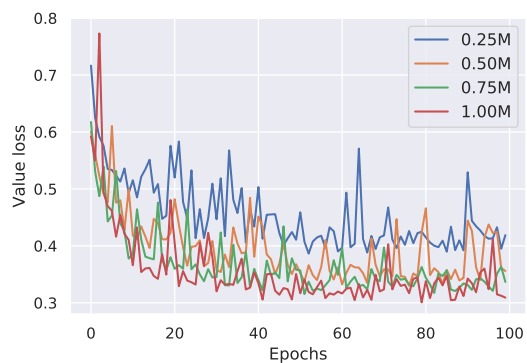
対局の際にはニューラルネットワークの局面価値予測による一手読みを用いて次の着手を決定した。つまりある局面においてすべての後者局面の局面価値の予測を行い、その値が最も高かった局面へ遷移する行動を次の行動とする。BASELINE と POLICY-RANKNET は着手予測も行うが、その値は対局に用いなかった。また対局の多様性を確保するために GoGui^{*3} に搭載されている 150 の定石を使用したのちにプレイヤー同士を対局させた。

^{*2} <https://github.com/pasky/pachi/releases>. Accessed on 2018/12/10.

^{*3} <https://sourceforge.net/projects/gogui/>. Accessed on 2018/12/10.

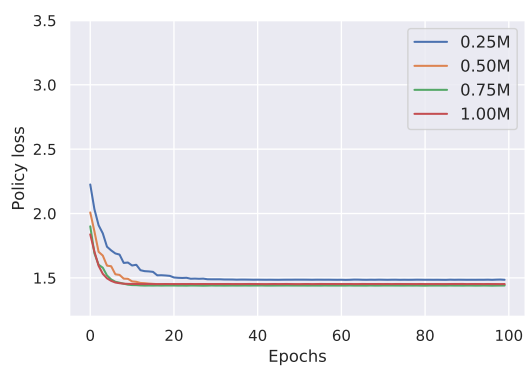


(a) Policy loss

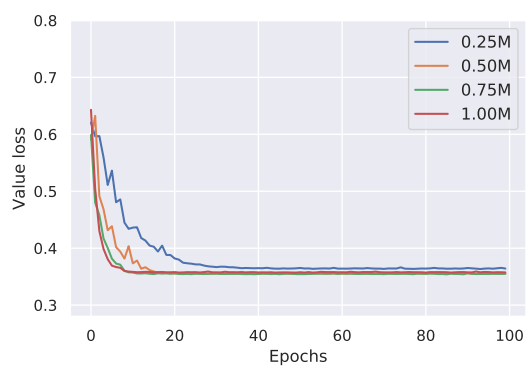


(b) Value loss

図 6.1: BASELINE-CONF1 の validation セットにおける損失

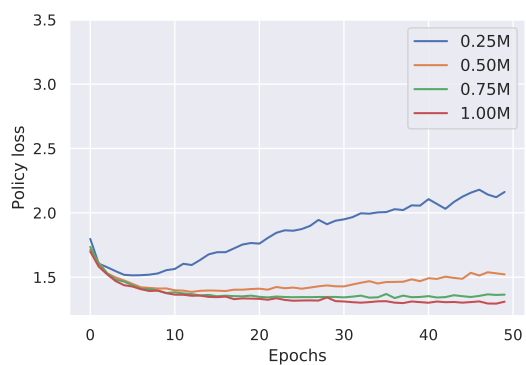


(a) Policy loss

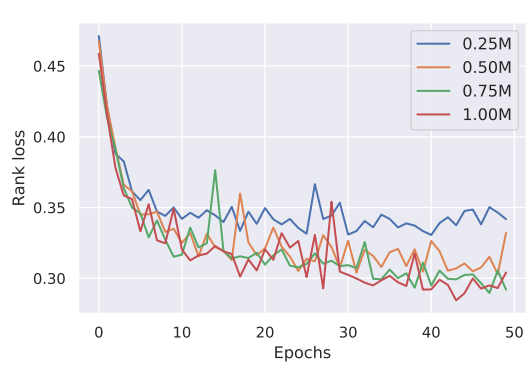


(b) Value loss

図 6.2: BASELINE-CONF2 の validation セットにおける損失



(a) Policy loss



(b) Rank loss

図 6.3: POLICY-RANKNET の validation セットにおける損失

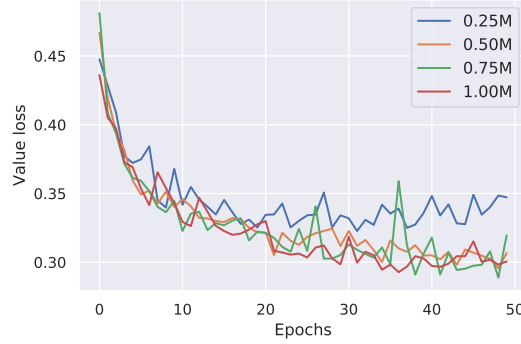


図 6.4: RANKNET の validation セットにおける損失

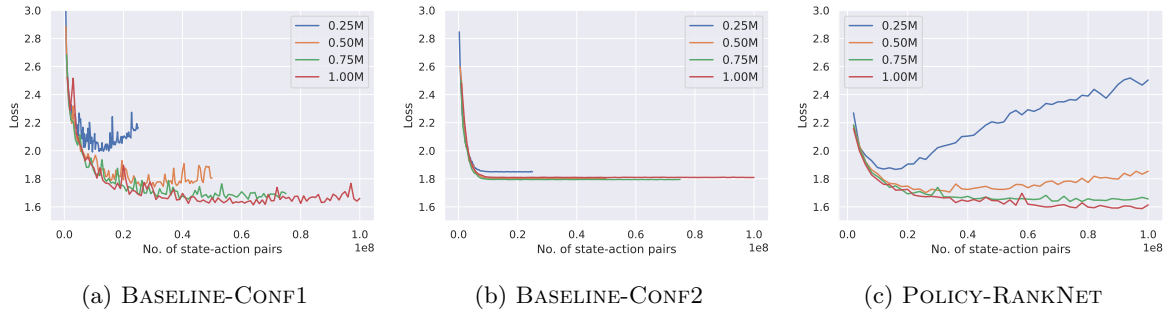


図 6.5: 学習に用いた訓練例の数と損失関数の値の関係 (validation セット)

表 6.4: 各ニューラルネットワークの最良の epoch 数

	0.25M	0.50M	0.75M	1.00M
BASELINE-CONF1	38	61	52	64
BASELINE-CONF2	62	49	28	54
POLICY-RANKNET	8	13	39	49
RANKNET	32	49	49	37

表 6.5 に 1,000 局の結果を示す。結果から、すべてのデータセットで POLICY-RANKNET が BASELINE よりもよい成績を取めた。また BASELINE-CONF2 は BASELINE-CONF1 と比較して、データ数を増やした場合における成績が悪化している。これは学習率の減衰スケジュールが不適當で、十分に学習を行うことができていなかったと推測される。一方で RANKNET はすべてのデータセットで顕著に性能が悪化していることがわかる。このことより着手予測とともに局面価値予測を行う学習は結果として強い正則化効果をもたらし、汎化性能を向上させることが示唆される。対局数から見てもこれらの差は有意であり、着手予測を同時に行う RankNet 方式の学習は有効に働いたと考えられる。

また POLICY-RANKNET と BASELINE を直接対局させた結果を表 6.6 に示す。それぞれのプレイヤーは決定的に着手を行うため、定石の数に先後入れ替えの 300 局の対局を行った。結果から POLICY-RANKNET が BASELINE よりも良い結果となった。特にデータセットの数が少ないときに差は顕著である。このことが

表 6.5: Pachi (10k playouts/move) に対する勝率 (標準誤差)

	BASILINE-CONF1	BASILINE-CONF2	POLICY-RANKNET	RANKNET
0.25M	8.4% (0.9)	7.3% (0.8)	14.8% (1.1)	2.8% (0.5)
0.50M	24.6% (1.4)	8.7% (0.9)	29.4% (1.5)	6.9% (0.8)
0.75M	29.8% (1.5)	7.7% (0.8)	43.7% (1.6)	5.5% (0.7)
1.00M	38.8% (1.6)	9.1% (0.9)	42.5% (1.6)	6.7% (0.8)

表 6.6: BASELINE に対する POLICY-RANKNET の勝率 (標準誤差)

	BASILINE-CONF1	BASILINE-CONF2
0.25M	70.4 % (2.9)	88.2 % (2.2)
0.50M	64.1 % (3.1)	89.9 % (1.9)
0.75M	57.3 % (3.1)	89.1 % (2.0)
1.00M	53.5 % (3.0)	91.4 % (1.7)

ら, RankNet を用いてどちらの局面が良いかどうかを予測する学習方式のほうがより効率的に学習でき, 単一の訓練例から多くの情報を抽出できると示唆している.

6.2.4 テストセットでの性能評価

それぞれのニューラルネットワークの性能を調査するため別のデータセットでの評価を行った. 評価に用いる指標は *sign accuracy*, *rank accuracy*, および *move prediction accuracy* の三つである. 評価したニューラルネットワークの重みは 6.2.3 項 と同様である.

Sign accuracy は以下のように定義する. テストセット中のデータ $\langle s, r_s \rangle$ について, 局面価値予測 $v_s = f(s) \in [-1, 1]$ が $r_s \in \{-1, 1\}$ と同じ符号であれば正解, そうでなければ不正解とし, 正解の数の割合を sign accuracy とする. テストセットには引き分け ($r_s = 0$) のものも存在するが, この指標を計算する際には除外している. 数式によって表現すると, ある局面集合 S に対する予測器 f の sign accuracy は

$$\text{SignAccuracy}(f; S) \equiv \frac{1}{|S|} \sum_{s \in S} \mathbf{1}[\text{sign}(v_s) = r_s]$$

となる. ここで $\mathbf{1}$ は括弧内の条件式が真であれば 1 を, 偽であれば 0 を返す.

Rank accuracy はある局面对 $\langle s_i, s_j \rangle$ に対して定義される. ここで s_i, s_j は学習の際に構成したようにどちらかがどちらかよりも優れている局面である. ある予測器 f が計算する局面価値 $v_s = f(s)$ について, s_i が優れているときに $f(s_i) > f(s_j)$ であれば正解, そうでなければ不正解とし, s_j が優れているときに $f(s_i) < f(s_j)$ であれば正解, そうでなければ不正解とする. 正解の全体に対する割合を rank accuracy と定義する. Rank accuracy の計算の際には s_i, s_j 両方とも引き分け局面であるような例は除外している.

Move prediction accuracy は着手予測のうち最も確率の高い着手がデータセット中の次の着手と等しければ正解, そうでなければ不正解としたときの正解の割合である.

テストセットは WWW 上で公開されている aya の自己対戦棋譜 *⁴ の中から 20,000 局を利用した．学習用の棋譜を生成した aya と，公開されている自己対戦棋譜を生成した aya とは異なるバージョンである．各指標の計算には 20,000 局のうち 1,048,576 局面を用いた．

図 6.6, 6.7 はそれぞれのニューラルネットワークの sign accuracy と rank accuracy である．図 6.8 は BASELINE と POLICY-RANKNET の move prediction accuracy を示している．結果から，どのニューラルネットワークの重みについてもこの指標のもとでは差が現れなかった．どのニューラルネットワークも局面の価値を正しく判別でき，着手予測も概ね高い正答率を示している．6.2.3 項 で示したとおり対局実験では有意に差が現れているため，ここで示した指標は実際の対局中の強さとは関係が薄いと考えられる．差が出なかった理由として (1) テストセットとして用いた局面はなんらかの偏りが存在する，(2) 比較したニューラルネットワークの差異はゲーム中の特定の状況もしくは序盤・中盤・終盤などの特定の段階で現れる，などが考えられる．(1) については，学習に用いた棋譜とテストに用いた棋譜はほとんど同じプログラムが生成しているため大いに考えうる．また興味深いことに move prediction accuracy については RankNet 方式のほうが単一の局面から学習したものよりわずかに高い正答率を示している．Move prediction accuracy については文献 [10] で報告されたものと概ね同じ値となっている．また BASELINE-CONF1 と BASELINE-CONF2 を比較すると，対戦実験のときとは異なりほとんど同じ結果となった．これも先ほど述べたような理由が原因と考えられる．

6.3 まとめ

この章では評価関数の学習として新たに RankNet を用いた学習方法を提案し，その性能について議論した．提案した RankNet 方式のアルゴリズムは AlphaGo Zero で用いられた着手予測と局面価値予測の両方を同時に行うニューラルネットワークを効率よく訓練できることを示し，その評価を対局実験によって行った．実験によって，着手予測と局面価値予測両方を用いるニューラルネットワークは学習を安定化し汎化性能を向上させ，RankNet 方式の学習によって限られた数の棋譜で学習が効率的に行えることを示した．

テストセットにおける評価では性能差を示すことができなかったが，今後はテストに用いる棋譜をより多様なものにすることで差異に関する知見が得られるものと思われる．

今後の課題として，提案した学習方法の理論的な解析があげられる．今回の実験では単一の局面の良さの学習よりも二つの局面の優劣の学習を行った方が対戦成績においては良い結果が得られた．この結果が得られた理由の一つとして局面の組み合わせによる訓練例の実質的な増加があげられる．しかし単にランダムに抽出した局面を比較するのではなく，より学習の効果が高いような局面の比較を行うなど別の方法も考えられる．また用いた損失関数についての比較も重要と思われる．今回提案したアルゴリズムの損失関数は既存の平均二乗誤差とは異なり，漸近的に線形になるものであったため，これらの違いによって結果に差異が生じたとも考えられる．

*⁴ <http://www.yss-aya.com/ayaself/ayaself.html>. Accessed on 2018/12/10.

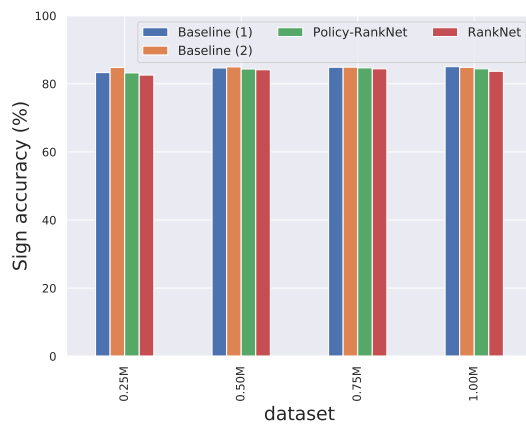


図 6.6: テストセットにおける Sign accuracy

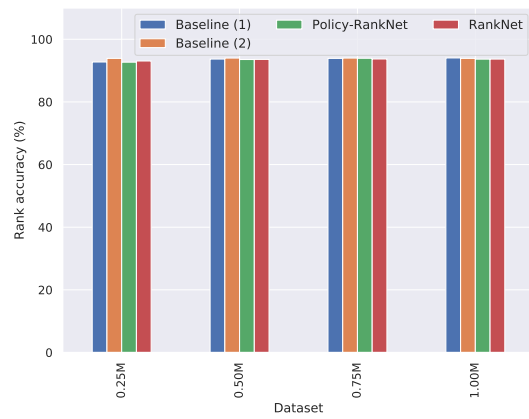


図 6.7: テストセットにおける Rank accuracy

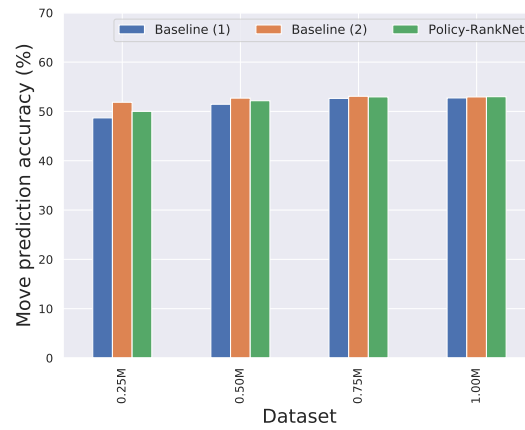


図 6.8: テストセットにおける Policy accuracy

第 7 章

Discriminator を用いたマルチタスク学習による効率的な評価関数の学習

この章では前章に続き事前にゲームに関する知見が強いプレイヤーの棋譜という形で利用できる場合により効率的な学習を行う方法について考察する。AlphaGo Zero が示したとおり、value head と policy head のマルチタスク学習はより良い汎化性能をもたらした。この知見は前章の RankNet を用いた学習における実験でも示されている。

この章では AlphaGo Zero が示したような value と policy だけでなく、別の学習目的を利用してマルチタスク学習を行なった場合にも学習した結果によりよい汎化性能をもたらすことを実験的に示す。具体的には 2.3.3 項 で述べた、GAN を模倣して評価関数を学習する方法を囲碁に対して実装して評価を行った。この手法では discriminator と呼ばれる、棋譜中に含まれる局面とランダムに生成した局面とを見分ける学習目的と、与えられた局面の価値を推定する学習目的とのマルチタスク学習を行う。

Discriminator を着手予測の代わりとしたマルチタスク学習の利点として、ニューラルネットワークの構成要素の数を減らすことが挙げられる。着手予測は一般的に対象のゲームのすべての局面における合法手を表現できるだけの数の出力を必要とする。例えば 19×19 の囲碁では 362 (パスを含む) あれば十分であるが、チェスだとその数は文献 [11] による実装だと 4762 必要となる。しかし discriminator はゲームの合法手順によらず単一のスカラー値を出力するため、ニューラルネットワークの構成要素数が小さくなる。

7.1 囲碁における Discriminator によるマルチタスク学習

この節では discriminator を用いた評価関数のマルチタスク学習について、囲碁に対して実験を行なった際の定式化を述べる。概ね 2.3.3 項 で述べた方法と同様であるが、いくつかの点について変更を加えている。

7.1.1 ネットワークの構成

この章で用いたニューラルネットワークの構成について詳説する。概ね 2.3.3 項 で述べた文献 [46] と同様である。ネットワークは単一の局面を符号化した s を入力として受け取り、一旦中間表現として $F(s)$ を計算する。そののち、中間表現 $F(s)$ から局面価値予測 $v(F(s))$ を出力する value head と、二つの中間表現 $F(s), F(s')$ から、 s' が s から遷移することが「もっともらしい」かどうかを出力する discriminator head $d(F(s), F(s'))$ を持つ。以降の節では $v(F(s)), d(F(s), F(s'))$ を単純に $v(s), d(s, s')$ と表記する。

中間表現を計算する feature extractor F は value head と discriminator head とで共有していることに注意する。AlphaGo Zero では policy network と value network が共通の residual blocks を共有しており、その際に正則化効果が得られたとされている [10]。同様に feature extractor を共有することでマルチタスク学習による正則化効果と汎化性能向上が期待できる。

局面価値予測を行う v の値域は $[-1, 1]$ とする。文献 [46] では値域を特に制限していなかったが、この章では AlphaGo, AlphaGo Zero, および AlphaZero で用いられた policy network との同時訓練との比較のために同様の値域に合わせた。局面価値予測には主に (1) 手番プレイヤーに対する価値予測と (2) 一方のプレイヤー（通常は先手番）に対する価値予測の二つの流儀があり、AlphaGo や AlphaGo Zero, Leela Zero などは (1) を、ELFOpenGo は (2) を採用している。この章の実験では実装の容易さから (2) を採用している。またもっともらしさを出力する d の値域も $[-1, 1]$ としている。 d の出力が $+1$ のとき、その後者局面 s' は実際に遷移する可能性が高く、 -1 のときにはありえそうにないと判断している。この値域は文献 [46] と同様である。

7.1.2 損失関数

学習には次の四つ組 $\langle s, s', s_r, z \rangle$ を用いる。ここで s は棋譜に含まれるある局面、 s' は棋譜中に実際に s から取られた行動 a を経た後者局面 s' 、 s_r は s における合法手集合のうちランダムに選択された行動によって遷移した局面 s_r である。また z は局面 s における先手番プレイヤーの報酬（勝利のとき $+1$ 、敗北のとき -1 、引き分けのとき 0 ）である。この章で用いた損失関数は value network と discriminator network それぞれに対応する項で構成されている。

$$L_{\text{All}}(s, s', s_r, z) = L_V(s, s', z) + L_D(s, s', s_r) \quad (7.1)$$

ここで、 L_D は式 (2.4) で定義される。これは 2.3.3 項と同様である。しかし value network の損失は以下のように変更している。

$$L_V(s, s', z) = ((v(s) - z)^2 + (v(s') - z)^2)/2$$

ここで s, s' 両方の局面価値予測の MSE を計算している。 s のみを用いて損失関数を計算しても訓練時における損失は training, validation set とともに減少するが、実際の対局において s のみを用いた場合性能が顕著に低下したため s' も含めて学習を行っている。 s_r に対しては局面価値予測の正確性を定義できないため除外している。しかし棋譜を生成したプレイヤーが最善プレイヤーであれば $v(s_r)$ を手番プレイヤーから見た報酬 -1 として損失を計算すればよいかもしれない*1。

また 2.3.3 項で述べた uniformity loss (式 (2.5)) は用いなかった。

7.2 実験による評価

この節では discriminator を用いたマルチタスク学習による評価関数の学習の性能を実験的に評価した。具体的には、discriminator および局面価値予測を同時に行うもの、着手予測および局面価値予測を同時に行うもの、ならびに局面価値予測のみを行うものの三つのニューラルネットワークの性能を対局実験にて比較した。題材としたゲームは 9 路盤の囲碁である。これは計算資源的に適度に複雑であり、ニューラルネットワー

*1 最善プレイヤーが着手しなかった手は少なくとも一手以上よりよい着手が存在するため、悲観的に考えて報酬を -1 とすれば間違いではない

表 7.1: 実験で用いたネットワーク構造の要約

ネットワーク構造	学習時の入力	損失関数	ネットワークの出力
VP-network	$\langle s, \pi, z \rangle$	式 (2.3)	$\langle p(s), v(s) \rangle$
VD-network	$\langle s, s', s_r, z \rangle$	式 (7.1)	$\langle v(s), d(s, s') \rangle$
V-network	$\langle s, z \rangle$	式 (2.1)	$v(s)$

クによる評価関数の作成に関する知見が十分存在し、かつ実装がしやすいために選択した。しかしながら本章で述べる結論は 19 路盤や 13 路盤の囲碁やチェス・将棋においても有効であると考えられる。

7.2.1 実験設定

ニューラルネットワークの構成

実験では上記三種類のニューラルネットワークを訓練した。以降それぞれ VD (value-discriminator), VP (value-policy), および V (value) ネットワークとする。VP ネットワークは単一の局面を入力として受け取り、出力としてその局面における着手予測とその局面の価値を出力する。VD ネットワークは局面価値予測の際には単一の入力を受け取り出力することができる。またある局面 s の後者局面 s' に対する discriminative value を出力する際には二つの局面を受け取り、中間表現 $F(s)$, $F(s')$ を計算したのちに discriminator head にその二つの中間表現を入力することで計算を行う。V ネットワークは policy head を除いた VP ネットワークである。

それぞれのネットワークの構成は可能な限り同じものにした。VP ネットワークはおおよそ AlphaGo Zero [10] と同様の構成をもつが、residual blocks の数を 16 とした (AlphaGo Zero は 20 もしくは 40)。VD ネットワークもそれに準ずるが、discriminator head として VP ネットワークの value head と同様の構造を用いた。Discriminator head の入力には $F(s)$ と $F(s')$ を結合したものを全結合層に入力している。ニューラルネットワークへ入力する際には局面を AlphaGo Zero と同様に符号化したものを用いた。すなわちある局面をその 7 手前からの石の配置を表現する $9 \times 9 \times 16$ の配列と、手番を表す 9×9 の配列を結合した $9 \times 9 \times 17$ の配列を入力とする。表 7.1 に三種類のネットワークをまとめたものを示す。

データセットと学習の設定

学習には WWW 上で公開されている aya の自己対戦棋譜^{*2}を用いた教師あり学習を行った。この aya の棋力は CGOS^{*3} 上で 2,529 BayesElo である。公開されている棋譜数はおおよそ 300 万局であるが、同一局内の相関を避けるため、一つの対局から一つの局面のみを学習と validation に用いた。この自己対戦は一つの対局中に一度だけ全合法手から一様ランダムに着手を行っているため、学習と validation にはそのランダムな着手のあとの局面を用いた。学習には 1,000,000 局面を、validation には 100,000 局面を用いた。

Discriminative model の学習には完全にランダムに着手したあとの局面 s_r が必要となるが、その局面は学習局面 s からランダムに着手したものを学習時に生成した。つまりある局面 s について、ある epoch で用いる s_r と別の epoch で用いる s_r は異なる可能性がある (ランダムに生成するため同一であることもある)。こ

^{*2} <http://www.yss-aya.com/ayaself/ayaself.html>. Accessed on 2018/12/10.

^{*3} <http://www.yss-aya.com/cgos/>. Accessed on 2018/12/10.

れにより学習に利用できる訓練例を増やすことが可能である。

学習はすべてのニューラルネットワークに対し同様の設定で行った。最適化には momentum SGD を用い、学習率を 0.01、モーメントを 0.9 して学習率減衰は行わなかった。バッチサイズは 1,024 とした。学習は 150 epochs まで行った。

7.2.2 学習結果

はじめに学習中の validation セットにおける損失関数の値について考察する。図 7.1 にそれぞれのニューラルネットワークの局面価値予測に関する損失関数の値を示す。図から分かる通りすべてのネットワークで損失関数の値が 0.2 程度に収束している。VP ネットワークと VD ネットワークは局面価値予測だけでなく別の目的関数が存在するが、二つとも同時に訓練しても単一の訓練の場合と同じくらいの精度で予測ができることが観測できる。

7.2.3 対局実験

次に学習した各ニューラルネットワークの対局における強さを観測した。対戦相手として Pachi (version 12.00-jowa) ^{*4} を用い、DCNN を使用せず一手ごとに 5,000 プレイアウト行うことで着手させた。

対局実験では Value, Policy, そして Disc プレイヤーという三種類のプレイヤーをそれぞれのネットワークごとに用意し、棋力を測定した。Value プレイヤーは現在の局面の後者局面すべてについて局面価値予測を行い、手番プレイヤーにとって最も良い評価となった後者局面へ遷移する行動を決定的に選択する。Policy プレイヤーは現局面の着手予測が出力する着手確率にしたがって確率的に行動を選択する。Disc プレイヤーは現在の局面と後者局面すべてに関し discriminator が出力するもっともらしさが最大の後者局面へ決定的に着手する。

ネットワークの種類三種類のうち五種類のプレイヤー (VP-Value, VP-Policy, VD-Value, VD-Disc, および V-Value) が存在するため、ニューラルネットワークは 10 epochs ごとに保存していたものを使用し、それぞれのプレイヤーと Pachi との対戦をそれぞれの重みについて 100 試合ずつ行った。図 7.2 に対局結果を示す。グラフ中の縦棒は標準誤差を示す。

Value プレイヤーの強さ

Value プレイヤーの強さを観察すると、VP-Value プレイヤーの強さと比較して VD-Value プレイヤーは学習序盤では性能が低い。しかし学習が進むと VD-Value プレイヤーは VP-Value プレイヤーと同程度の強さとなっている。対照的に V-Value プレイヤーは上述の二つのプレイヤーと比較して学習後半になっても対局においては性能が低い結果となった。

図 7.1 ではどのニューラルネットワークも損失関数の値は同程度となっていたが、対局実験においては損失関数の要素の数によって大きな差が生まれた。Validation セットでの損失の値が低かったのは、学習に用いたデータの分布とほとんど同じで汎化性能が高くなるとも損失関数の値を下げることができたからで、対局の相手であった Pachi が生成するような局面の分布についてはうまく予測ができなかったと思われる。このことから、実際の対局で有用な局面価値予測を行う汎化性能の高い評価関数を学習するためには複数の目的をもつ

^{*4} <https://github.com/pasky/pachi>. Accessed on 2018/12/10.

た学習を行うことが重要であると示唆される。

Policy および Disc プレイヤーの強さ

Policy および Disc プレイヤーは Value プレイヤーと比較して棋力が低いことがわかる。この差は着手予測および discriminative value が予測するもっともらしさが、局面価値予測とは性質の違うものを予測しているからと説明することができる。着手予測はある局面においてどの行動をとることが多かったかをもとに学習を行うため、局面価値予測が行う直接的な報酬とは出力するものの目的が異なる。よって着手予測による着手は必ずしも対局において勝利へと導く局面へと遷移するわけではなく、その差によって対局時の性能差が出たと思われる。同じ説明が discriminative value にも適用できる。

7.2.4 次の一手の正答率による評価

最後にそれぞれのプレイヤーの validation セットにおける次の一手の正答率について評価を行った。Validation セットにおけるある局面 s において、実際の着手によって遷移した後局面 s' のそれぞれのプレイヤーが用いる評価の値がその他の後局面のどれよりも（手番プレイヤーからみて）高ければその局面における出力を正解とする。つまり Value プレイヤーであれば $v(s')$ の値がすべての後局面のなかで最大であれば正解、そうでなければ不正解とする。Top-1 accuracy は validation セット中の局面数に対する正解数の割合とする。同様に Top- n accuracy はあるプレイヤーの用いる評価の値がすべての後局面のなかで n 番目に入っていれば正解としたときの割合とする。

図 7.3–7.5 にそれぞれのプレイヤーの validation セットにおける top-1, top-5, および top-10 accuracy を示す。この指標においても性能ごとに二つのグループに分けられる。Policy および Disc プレイヤーは棋譜中の次の着手を概ね正しく予測することができており、Value プレイヤーは比較して正答率が低い。また学習を進めるごとに正答率が減少傾向にある。このことから、実際の対局における性能と validation set における性能とはあまり相関がなく、実際に強いかどうかは実際に対局してみないとわからないことが伺える。また対局時において性能の良いプレイヤーの validation セットにおける正答率が低い理由として、より汎化した予測を得たとも考えられる。

Policy プレイヤーと同程度に Disc プレイヤーの正答率が高かった理由として、discriminative model が予測する局面遷移のもっともらしさは今回の教師あり学習の設定ではほとんど着手予測と同じものであるためであると思われる。

7.3 まとめ

この章では discriminator と局面価値予測とのマルチタスク学習が AlphaGo Zero で用いられた局面価値予測と着手予測のマルチタスク学習と同様に正則化効果をもたらし、汎化性能が向上することを示した。またこの章では 第 6 章 と同様に実際の対局で現れる性能の評価と、学習に用いたデータと同様の分布をもつ validation セットにおける性能の評価とが必ずしも関係しないことを示した。ここで得られた結果は実験で用いた 9 路盤囲碁だけでなくチェスや将棋などの他のゲームでも同様であると考えられる。

今後の課題としては学習によって得られた discriminator を探索に有効活用することが挙げられる。AlphaGo や AlphaGo Zero では学習した着手予測を用いてモンテカルロ木探索を前向き枝刈りし、探索効率を向上させている。実験で示したとおり、discriminator の出力結果は policy と同様の解釈が可能である。こ

れにより discriminator を利用しても探索の前向き枝刈りに利用できると予想される.

また value と policy と discriminator を組み合わせたマルチタスク学習も考えられる.

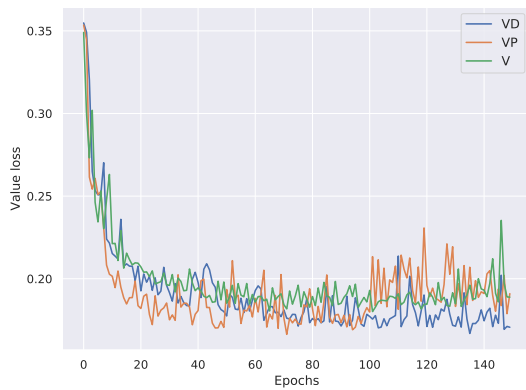


図 7.1: 学習時における validation セットの損失

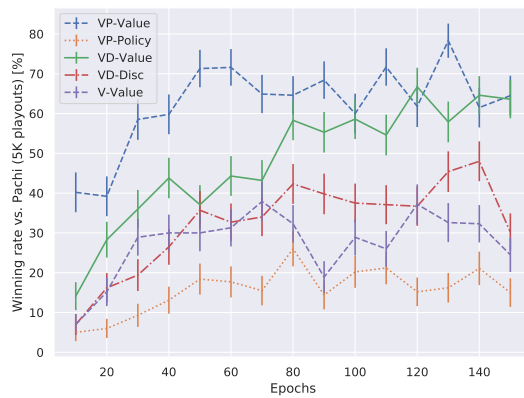


図 7.2: Pachi (5k playouts/move) に対する勝率 (value)

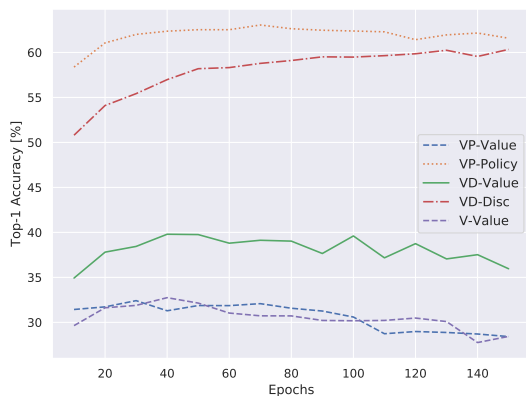


図 7.3: Validation セットにおける次の一手予測の Top-1 Accuracy

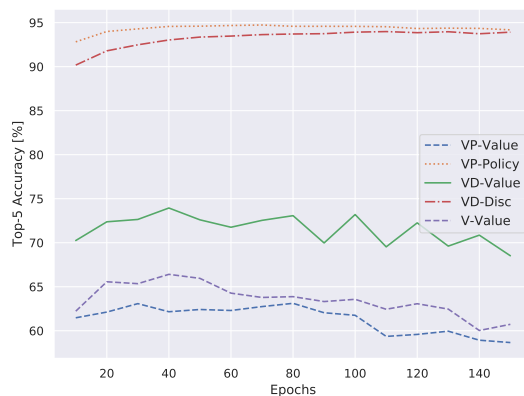


図 7.4: Validation セットにおける次の一手予測の Top-5 Accuracy

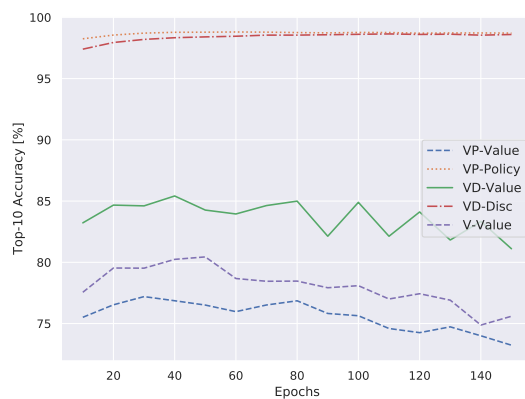


図 7.5: Validation セットにおける次の一手予測の Top-10 Accuracy

第 8 章

深層ニューラルネットワークと探索を組合せた判断根拠の可視化

本章では前二章で述べたようなアルゴリズムを用いて得られた知識による判断の根拠となったような入力部分を可視化する手法について述べる。これまでに述べてきたとおり人間を超えるような性能のコンピュータプレイヤーが出現している。これらのような人間を超える性能を持つ人工知能システムは今後さらに増えていき、現実社会がそれらに依存する割合は非常に高くなると予想される。しかし、人間を超えた人工知能システムによる判断の信頼性を検証することは困難であり、研究途上の段階であるといえる [55]。人工知能システムの判断の検証の重要な前段階として、その判断の根拠を提示して人間にとって理解しやすい形で可視化することが挙げられる。そのような可視化手法は人間の初心者への練習を手助けするようなシステムなど、幅広い応用先を持つと思われる。

本章では囲碁に対して訓練された深層ニューラルネットワークを対象に 2.6 節 で述べた Saliency Map [56] と SmoothGrad [62] を適用し、囲碁におけるそれらの有効性について議論を行う。特に、局面の着手判断や形勢判断の根拠が可視化できるかどうかを検証する。また深層ニューラルネットワークと探索アルゴリズムを組み合わせて得られる局面予測について、探索を考慮した判断根拠の可視化手法を提案し、実験的に評価を行った。図 8.1 に本稿で述べるアルゴリズムをある局面に対して適用したときの Saliency Map を示す。得られた Saliency Map は 8.3.1 項 で説明する方法によって 2 次元画像に変換して 0 から 1 に正規化したのち、0 を青、1 を緑とした場合の画像を表している。

8.1 囲碁の深層ニューラルネットワークに対する Saliency Map

本章で扱う囲碁に対して訓練された深層ニューラルネットワークは局面価値関数と方策予測があるものとする。これは現在標準的な実装であり、AlphaGo Zero [10] のほかに Leela Zero, ELFOpenGo [63] でも採用されている。

局面価値関数 $v: \mathcal{S} \rightarrow \mathbb{R}$ は局面 $s \in \mathcal{S}$ を実数値 $[-1, 1]$ へ写像する。これについて局面 s の Saliency Map を求めるには 2.6 節 で述べたとおりヤコビ行列

$$J_v(s) = \left(\frac{\partial v}{\partial s_1}, \dots, \frac{\partial v}{\partial s_n} \right)$$

を求める。

方策予測 $p: \mathcal{S} \rightarrow \mathbb{R}^d$ は局面を入力すると要素数 d の確率ベクトルを出力する。ここで d は着手予測の数

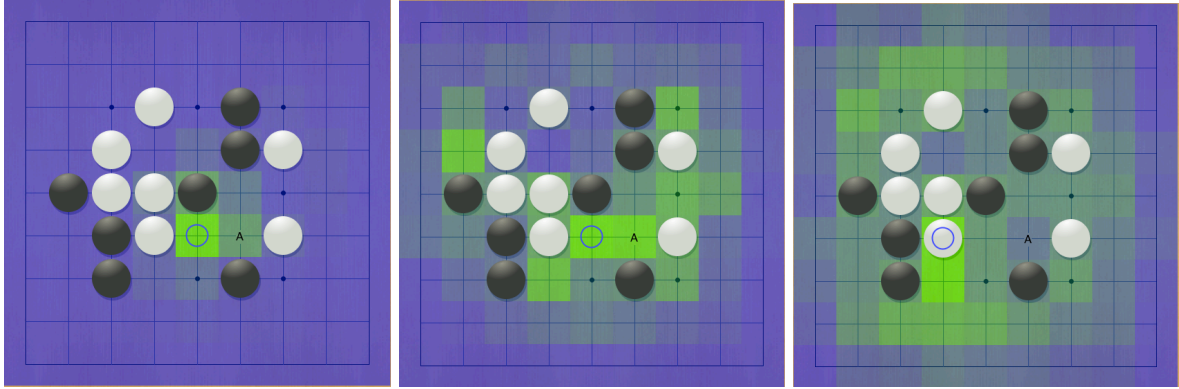


図 8.1: 得られた Saliency Map の例. 左から Poily, Value, MCTS による評価値に対する出力

で, 19×19 の囲碁であれば 362 となる. 複数候補が存在するため, 本章では $p(s) = \max \mathbf{p}(s)$, つまり予測された着手のうちもっとも確率が高い行動の出力についての判断根拠の可視化を考え,

$$J_p(s) = \left(\frac{\partial p}{\partial s_1}, \dots, \frac{\partial p}{\partial s_n} \right)$$

について議論する.

SmoothGrad を適用する際には入力にガウス雑音 $\mathcal{N}(\mathbf{0}, \sigma \mathbf{I})$ を加えて Saliency Map を計算する. このときの入力を解釈すると, 「ある座標に石が 1.21 個存在する」ような状況になりうるが, 本章では基本的に入力を画像とみなし, そのような状況についても考察する.

8.2 深層ニューラルネットワークとモンテカルロ木探索による価値予測結果に対する Saliency Map

モンテカルロ木探索によって根局面 s_0 の評価値の予測 $\hat{v}(s_0)$ を計算することができる. ここで $\hat{v}(s_0)$ はもとの局面価値予測 $v(s_0)$ よりも正確であると期待される. この改善された価値予測に対して Saliency Map を適用する手法を提案する.

簡単のため, AlphaGo Zero で採用されたような局面の展開の閾値が 1, プレイアウト回数 n モンテカルロ木探索を行ったとする. このときモンテカルロ木探索が出力する局面評価値は

$$\hat{v}(s_0) = \frac{1}{|T|} \sum_{s \in T} v(s) \quad (8.1)$$

となる. ここで T は探索後の探索木に存在する局面の集合で, $|T| = n + 1$ である. この \hat{v} を s_0 について偏微分すると Saliency Map が得られる. 式 (8.1) 右辺を s_0 で表すことを考える.

まず着手 \mathbf{a} によって一手進んだ局面について考える. 局面を $\mathbf{s} \in \mathbb{R}^d$ として表すことができるとする. このとき, 局面価値関数 $v: \mathbb{R}^d \rightarrow \mathbb{R}$ の \mathbf{s} における Saliency Map は

$$J_v(\mathbf{s}) = \left(\frac{\partial v}{\partial s_1}, \dots, \frac{\partial v}{\partial s_d} \right)$$

とヤコビ行列として表現できた. ここで状態遷移関数 \mathcal{T} を導入し, 後者局面 $\mathbf{t} = \mathcal{T}(\mathbf{s}, \mathbf{a})$ における v の \mathbf{s} に

おけるヤコビ行列は状態遷移関数が微分可能と仮定すれば

$$J_{v \circ \mathcal{T}}(\mathbf{s}) = J_v(\mathbf{t}) J_{\mathcal{T}}(\mathbf{s})$$

で求まる。ここで

$$J_{\mathcal{T}}(\mathbf{s}) = \begin{pmatrix} \frac{\partial t_1}{\partial s_1} & \cdots & \frac{\partial t_1}{\partial s_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial t_d}{\partial s_1} & \cdots & \frac{\partial t_d}{\partial s_d} \end{pmatrix}$$

である。 \mathbf{s} のある要素 s_i について考えると,

$$\begin{aligned} \frac{\partial}{\partial s_i}(v \circ \mathcal{T}) &= \left(\frac{\partial v}{\partial t_1}, \dots, \frac{\partial v}{\partial t_d} \right) \left(\frac{\partial t_1}{\partial s_i}, \dots, \frac{\partial t_d}{\partial s_i} \right)^\top \\ &= \sum_{k=1}^d \frac{\partial v}{\partial t_k} \frac{\partial t_k}{\partial s_i} \end{aligned} \quad (8.2)$$

となる。

簡単のため、囲碁の局面を $\mathbf{s} = (\mathbf{b}^\top, \mathbf{w}^\top, c)^\top \in \{0, 1\}^{2d+1}$ で表す。ここで $\mathbf{b} \in \{0, 1\}^d$ は黒石の配置を, $\mathbf{w} \in \{0, 1\}^d$ は白石の配置を, $c \in \{0, 1\}$ は手番を表す。また着手を $\mathbf{a} = (\mathbf{a}^b{}^\top, \mathbf{a}^w{}^\top)^\top \in \{0, 1\}^{2d}$ とする。 $\mathbf{a}^b, \mathbf{a}^w \in \{0, 1\}^d$ は着手に対応する要素が 1, それ以外が 0 のベクトルであり, 手番が黒であれば \mathbf{a}^b が, 手番が白であれば \mathbf{a}^w の要素が 1 になる。また状態遷移関数を

$$\mathcal{T}(\mathbf{s}, \mathbf{a}) = (\mathbf{b}^\top + \mathbf{a}^b{}^\top - \mathbf{u}(\mathbf{s}, \mathbf{a}), \mathbf{w}^\top + \mathbf{a}^w{}^\top - \mathbf{u}(\mathbf{s}, \mathbf{a}), 1 - c)^\top \quad (8.3)$$

と仮定する。ここで $\mathbf{u}(\mathbf{s}, \mathbf{a}) \in \{0, 1\}^{2d}$ は石の打ち上げが発生する際に, 打ち上げる石に対応する要素が 1, そうでなければ 0 という関数とする。 $\mathcal{T}(\mathbf{S})$ の黒石に対応するある要素 t_i ($1 \leq i \leq d$) については,

$$t_i = b_i + a_i^b - u_i(\mathbf{s}, \mathbf{a}) = s_i + a_i^b - u_i$$

となる。白石に対応する箇所も同様である。手番に対応する要素 t_{2d+1} は,

$$t_{2d+1} = 1 - c = -s_{2d+1}$$

である。このとき, この関数が s_i で微分可能であるとする, 黒石の要素 ($1 \leq i \leq d$) については

$$\begin{aligned} \frac{\partial t_i}{\partial s_j} &= \frac{\partial}{\partial s_j}(s_i + a_i^b - u_i(\mathbf{s}, \mathbf{a})) \\ &= \begin{cases} 1 + \frac{\partial a_i^b}{\partial s_j} - \frac{\partial u_i}{\partial s_j} & i = j \\ 0 + \frac{\partial a_i^b}{\partial s_j} - \frac{\partial u_i}{\partial s_j} & i \neq j \end{cases} \end{aligned}$$

となる。白石の要素も同様になる。また $i = 2d+1$ のときは,

$$\begin{aligned} \frac{\partial t_{2d+1}}{\partial s_j} &= \frac{\partial}{\partial s_j}(-s_{2d+1}) \\ &= \begin{cases} -1 & j = 2d+1 \\ 0 & j \neq 2d+1 \end{cases} \end{aligned}$$

であると考えることができる．ここで $\partial a_i^b/\partial s_j, \partial a_i^w/\partial s_j, \partial u_i/\partial s_j$ はつねに 0 であると仮定すれば，

$$\frac{\partial t_i}{\partial s_j} = \begin{cases} 1 & i = j \text{ かつ } i \neq 2d+1 \\ 0 & i \neq j \text{ かつ } i \neq 2d+1 \\ -1 & i = j = 2d+1 \end{cases}$$

と簡略化できる．これを用いると，式 (8.2) は，

$$\frac{\partial}{\partial s_i}(v \circ \mathcal{T}) = \begin{cases} \frac{\partial v}{\partial t_i} & i \neq 2d+1 \\ -\frac{\partial v}{\partial t_i} & i = 2d+1 \end{cases}$$

とみなせる．すなわち $i \neq d$ については単純に \mathbf{t} の微分になっており， $i = d$ のときは符号が反転している Saliency Map になっている．

以上をまとめると，以下の仮定

- 状態遷移関数 $\mathcal{T}(\mathbf{s}, \mathbf{a})$ が \mathbf{s} で微分可能
- \mathcal{T} が式 (8.3) で表される
- $\partial a_i^b/\partial s_j, \partial a_i^w/\partial s_j, \partial u_i/\partial s_j$ がつねに 0

を満たす場合は，一手進めたときの Saliency Map を

$$J_{v \circ \mathcal{T}}(\mathbf{s}) = \left(\frac{\partial v}{\partial t_1}, \dots, \frac{\partial v}{\partial t_{d-1}}, -\frac{\partial v}{\partial t_d} \right)$$

として記述できる． \mathbf{a} については \mathbf{s} に依存しない変数なので偏微分が 0 という仮定は妥当であると考えられる． \mathbf{u} については \mathbf{s} に依存すると考えられるため妥当ではないかもしれないが，打ち上げが発生しない局面の近傍については 0 として近似することができると思われる．二手進めたときには手番を表す要素が元に戻るため，着手後の局面についての Saliency Map そのものになる．

よってモンテカルロ木探索によって評価された局面評価値の Saliency Map は，

$$J_{\hat{v}}(s_0) = \frac{1}{|T_m|} \sum_{s \in T_m} \left(\frac{\partial v}{\partial s_1}, \dots, \frac{\partial v}{\partial s_{d-1}}, \frac{\partial v}{\partial s_d} \right) \quad (8.4)$$

$$+ \frac{1}{|T_o|} \sum_{s \in T_o} \left(\frac{\partial v}{\partial s_1}, \dots, \frac{\partial v}{\partial s_{d-1}}, -\frac{\partial v}{\partial s_d} \right) \quad (8.5)$$

と解釈することが可能である．ここで T_m は探索木中の局面集合のうち，探索開始局面と手番が同じ局面， T_o は手番が異なる局面である．式 (8.3) のような定式化は AlphaGo Zero や Leela Zero, ELFOpenGo で用いられている局面の符号化方式と同じであり，局面の履歴が含まれる場合でも同じ議論が可能である．

AlphaGo Zero では局面の符号化を「自分の石」と「相手の石」という方式で行っている [10] が，そのような場合では適切に特徴を並び替えることで同様の出力結果が得られる．またモンテカルロ木探索における展開の閾値が 1 より大きい数であった場合は加重平均をとる．

Minimax 探索によって局面を評価する場合には，探索によって得られた PV の葉局面について Saliency Map を適用すれば探索開始の根局面の判断根拠となるかもしれない．しかしモンテカルロ木探索の場合，根局面の評価値は探索中に訪れた全ての局面の評価値の加重平均となるため，PV の葉局面だけに対して Saliency Map を適用しても正確でないと予想される．この手法はモンテカルロ木探索の性質を勘案した手法である．

表 8.1: Pachi (10,000 playouts/move) に対する勝率

	勝率
Policy network (softmax)	13.4 %
Value network (一手読み)	62.9 %
MCTS (400 playouts/move)	91.0 %

8.3 重要な石の検出実験

既存手法である Saliency Map と SmoothGrad および 8.2 節 で提案した MCTS による評価値に対する Saliency Map を囲碁 DNN に適用した際に得られる出力に関して評価を行った。実験は 9 路盤用に学習を行った DNN を対象に行った。DNN は AlphaGo Zero と同様の構造を持つが、Residual Block の数を 16 に削減して実装した。入力に用いた特徴は 7 手前から現在の石の配置と手番の色のみを用いた $17 \times 9 \times 9$ の特徴を用いた。学習は公開されている aya の自己対戦棋譜 *¹ のうちの 2,000,000 局を用い、一局につき一局面を抽出 *² して教師例とした。学習後の DNN は Pachi (1 手 10,000 プレイアウト) に対して表 8.1 の対戦結果となった。

評価実験には学習に用いなかった対局を使用した。節 8.3.2, 8.3.3 では 100 対局 9,947 局面を使用した。8.3.4 章では 1000 対局を用い、それぞれの対局から一局面をランダムに抽出した 1,000 局面を使用した。

8.3.1 Saliency Map の結果の集約方法

Saliency Map や SmoothGrad によって得られる出力の次元は入力の次元と等しいが、これらの値の性質を個々に調査することは難しいため、出力結果を集約して次元を圧縮し、元の入力画像のピクセル数と等しくすることが行われている。例として文献 [56] では RGB 画像 $3 \times w \times h$ に対して得られた結果 $3 \times w \times h$ をピクセルごとに絶対値の最大のものを採用することで出力結果を集約している。

様々な集約方法が考えられるが、今回の実験ではそれぞれの座標ごとに、対応する出力の絶対値の総和を取ることによって 9×9 の値に集約し、集約した値をその座標の重要度として囲碁 DNN の判断結果に影響を与えると仮定した。

8.3.2 方策予測に対する Saliency Map

方策予測 (policy network) の出力に対して Saliency Map/SmoothGrad の出力が意味のあるものとなっているかどうかを検証するために実験を行った。実験では 8.3.1 項 で説明した集約方法により最も『重要』であるとされた座標の石が出力結果に与える影響を計測した。この実験手法は文献 [64, 64] で行われた評価手法を参考に考案した。局面 s_0 に対して Saliency Map/SmoothGrad を適用し、最も重要である座標を得たのち、その座標に石がある場合はその石を取り除く（なければ何もしない）ことで局面 s' を得る。 s_0, s' における方策予測の出力の変動を交差エントロピー $H(p(s_0), p(s'))$ の大きさによって計測した。この実験で

*¹ <http://www.yss-aya.com/ayaself/ayaself.html>. Accessed on 2018/12/10.

*² 一局に一回ランダムな着手があり、その直後の局面を採用した

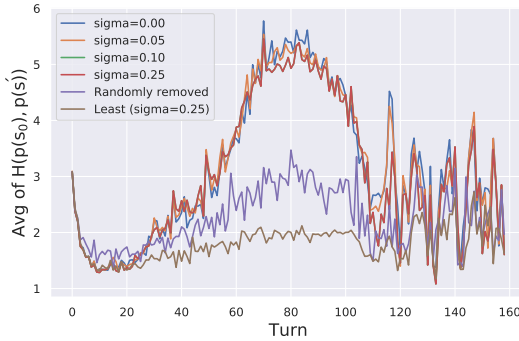


図 8.2: 方策評価の手数ごとの平均変動値
($H(p(s_0), p(s'))$)

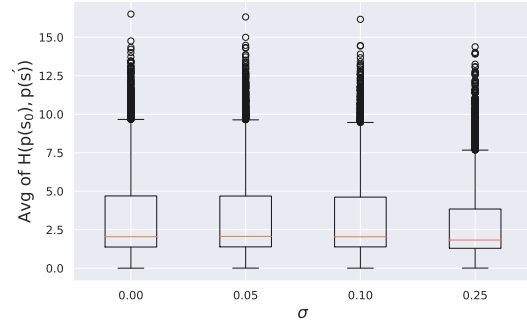


図 8.3: パラメータごとの方策評価の平均変動値
($H(p(s_0), p(s'))$)

用いている DNN の入力 は 7 手前からの石の配置の履歴が存在するため、石を取り除く場合には 7 手前から現在の座標すべてから石を除く．図 8.2 に、元の局面の手数ごとに交差エントロピーの平均をとったグラフを示す．SmoothGrad については $n = 128$ とし、 σ の値を 0.05, 0.10, 0.25 とし計測を行った． $\sigma = 0$ は Saliency Map に対応する．また比較のため石を取り除く座標を、ランダムに選択したものおよび出力結果が最小である（つまり重要度が最小の）ものについて計測した結果も示す．重要度最小のものは $\sigma = 0.25$ のときの SmoothGrad によって計測したものを掲載している．他の手法 (Saliency Map, $\sigma = 0.05, 0.10$ の SmoothGrad) によって得られた重要度最小のものでもほとんど同じものが得られたため結果からは除いている．

図から、交差エントロピーの大きさはランダムに石を取り除いた場合よりも Saliency Map/SmoothGrad を用いて選択したときのほうが有意に大きくなっていると読み取れる．また傾向として中盤から終盤において交差エントロピーの変動が大きい．序盤はどこの石を取り除いても大きく予測が外れ、100 手目以降はデータ数が少ないため傾向が不安定である．したがって、囲碁 DNN の方策予測に対して適用した結果は中盤から終盤においては方策予測に大きく関わるような石を重要であると認識していると考えられる．

Saliency Map と SmoothGrad の違いを見るため、計測した交差エントロピーを箱ひげ図で表したものが図 8.3 である．Saliency Map と SmoothGrad で結果にあまり差が見られず、どれも同じような出力結果が得られていると考えられる．

8.3.3 局面価値予測に対する Saliency Map

次に、局面価値予測 (value network) に対して 8.3.2 項 章と同様の実験を行った．今回は交差エントロピーではなく、局面間の価値予測の絶対値 $|v(s_0) - v(s')|$ を指標として用いた．図 8.4 に手数ごとに評価値の変動を平均をとったグラフを、図 8.5 に手法ごとに箱ひげ図を描画したグラフを示す．

全体の傾向として方策予測のときと同様にランダムに石を取り除いた場合よりも変動が大きい．序盤の傾向として、ランダムに取り除いたものや重要度最小の石を取り除いたものは予測の変動が大きく、Saliency Map/SmoothGrad で得られたものはほとんど変動しない．これはランダムに石を取り除くと学習データに含まれないような稀な局面に変化してしまい、そのような局面では局面評価値が大きく変化してしまっているためだと考えられる．中盤から終盤にかけては Saliency Map/SmoothGrad が認識した石の影響が大きく現れている．方策予測のときと同様、100 手目以降はデータ数が少ないため特定の傾向を見出すことは難しい．



図 8.4: 局面価値評価の手数ごとの平均変動値 ($|v(s_0) - v(s')|$)

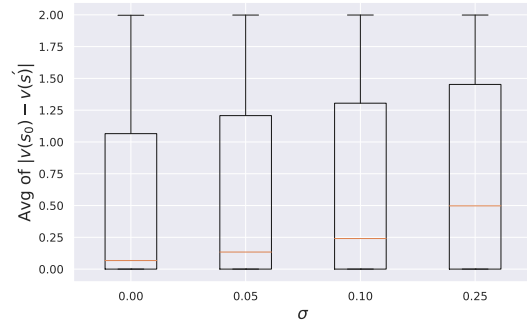


図 8.5: パラメータごとの局面価値評価の平均変動値 ($|v(s_0) - v(s')|$)

方策予測のときとは異なり，Saliency Map と SmoothGrad を比較した場合 σ が増えるにつれて評価値の変動がより大きくなっていることが2つの図から読み取れる．考えられる説明として，方策予測の場合は石の有無がより大きく影響を与えるために SmoothGrad による正規乱数の影響が大きく，その手法の利点である多数の類似した入力の平均による雑音の除去を打ち消してしまったということが挙げられる．一方で局面価値予測の場合はノイズを加えた場合でも価値予測に影響はほとんどなく，SmoothGrad の雑音除去が効果的に働いたと考えられるため，この実験で用いた指標に関しては SmoothGrad のほうが性能がよくなったと考えられる．

8.3.4 MCTS に対する Saliency Map

最後に 8.2 節 で提案した，MCTS によって得られた局面価値予測に対する Saliency Map に対して評価実験を行った．適用した局面には石の取り上げを含む場合があるが，Saliency Map の計算はすべて 式 (8.4)) で求めた．この実験では重要だと出力した座標に対して正規乱数によるノイズを加えた入力を用いて評価値の変動を計測した．以前の実験のように石を取り除いてしまうと探索する局面が大きく異なってしまうことを防ぐためである．図 8.6 に正規乱数の標準偏差を変えつつ評価値の変動の絶対値 $|\hat{v}(s_0) - \hat{v}(s')|$ を計測した結果を示す．先程の実験と同様，ノイズを加える座標をランダムに選択した場合と重要度が最も低いものにした場合を比較のために掲載する．雑音の影響が大きくなるにつれて，提案手法である MCTS に対する Saliency Map によって得られた座標の影響の度合いが大きくなることが確認できる．先述の実験と比較して評価値の変動が非常に小さいが，これは石を取り除く先述の実験とは異なり，雑音が入力に乗っているのみなので比較対象とはならないことに留意されたい．ランダムに取り除く場合と比較すると最重要とされた座標に雑音に乗せた場合は大きく変化しており，提案手法により重要な座標を検出できていると考えられる．

8.4 まとめ

本稿では Saliency Map と SmoothGrad を用いて囲碁について訓練された深層ニューラルネットワークの判断の根拠を可視化することを目的とした．またモンテカルロ木探索による局面価値予測に対して Saliency Map を適用する手法を考案した．提案手法では囲碁の状態遷移関数を簡素化し，石の打ち上げがない場合での妥当性を検証した．

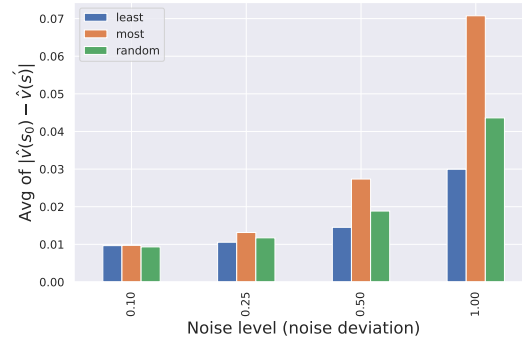


図 8.6: ノイズレベルごとの MCTS 評価値の平均変動値 ($|\hat{v}(s_0) - \hat{v}(s')|$)

評価実験では既存の文献で用いられた評価方法を踏襲して Saliency Map/SmoothGrad が『重視』する座標の妥当性を検証した。また既存研究と同様のアイデアにより提案手法を評価し、その効果を検証した。

今後の課題として、人間が用いる囲碁の特徴との対応づけがまず挙げられる。今回評価した指標はすべて単一の座標のみを対象としているが、普通人間が判断の根拠とするものは複数の座標のパターンである。このような対応づけができればより人間が DNN を解釈しやすくなるため有意義であると考えられる。またモンテカルロ木探索と組み合わせた場合における可視化は、遷移関数が微分可能であることや遷移関数の単純性などの仮定を置いていた。そのためこれらの要素について再度検証する必要がある。また入力に AlphaGo で用いられていたようにシチョウなどが入る [9] と容易に微分可能ではないため、これらについても議論する必要がある。

第 9 章

結論

本研究ではゲームを題材に、対象となるゲームの知識をゲーム木探索に利用するためにどのように効率よく獲得すればよいか、また獲得した知識をどのように評価するかについての研究を行った。本研究で対象とした知識とはゲーム木探索に利用される評価関数であり、評価関数をどのように効率的に学習するかについて議論を行なった。具体的には (1) 事前に強いプレイヤーの棋譜が利用できる状況で、その棋譜を利用した教師あり学習について効率的な学習方法を論じ、(2) 棋譜が利用できず事前に強化学習も行えないようなゲームを対象に、ゲームのプレイ中における木探索の際にゲームの知識を獲得するアルゴリズムについて性能評価と高速化を行なった。

第 3 章 では局面の特徴とその局面からの最終的な利得に関係性のあるゲームを模した人工的な評価の枠組みを考案し、その特性について議論した。そのような評価の枠組みとして P-game tree が存在するが、P-game tree は単一の評価値のみを提供する。この章では P-game tree を拡張した特徴ベクトル付き Incremental Random Game Tree を提案し、その性質について議論を行なった。局面の特徴を表す特徴ベクトルがそれぞれは局面に割り当てられ、特徴ベクトルによってある局面の評価値が決定される。その際、特徴ベクトルの割り当て方を二通り提案し、それぞれの異なる性質を論じた。

第 4 章 では事前に学習ができないような場合を念頭に、探索中に局面の特徴から報酬を予測しそれを探索に役立てるようなアルゴリズムについて記した。具体的には木探索アルゴリズムとしてモンテカルロ木探索を用い、モンテカルロ木探索のシミュレーションの報酬を予測する手法である。報酬の予測には線形回帰を用いる LinUCB と呼ばれる線形バンディット問題のアルゴリズムを利用し、局面の報酬を予測するとともにモンテカルロ木探索における次の局面の選択の際にも探索と知識の活用のジレンマをうまく扱う LinUCT アルゴリズムを提案した。評価においては第 3 章 で提案した特徴ベクトル付き Incremental Random Game Tree とともに、モンテカルロ木探索の性能評価で用いられるゲームでの性能を議論した。実験結果からは、対象のゲームにおいて特徴ベクトルと報酬に関係がある場合に、既存のモンテカルロ木探索アルゴリズムと比較して性能が良いことがわかった。

第 5 章 では第 4 章 で議論した LinUCT について近似を行うことにより高速化を施し、特徴の空間が大きい人工木や実際のゲームにおける性能を調査した。高速化は確率的勾配降下法を用いて行い、時間計算量を改善した。また高速化した LinUCT は、上述の特徴ベクトルと報酬に関連性が存在する場合においては近似を行わないアルゴリズムと比較して遜色ない性能となった。実際のゲームとしては囲碁を用いて性能を観測し、数千次元の特徴ベクトルを用いた場合でも現実的な動作時間で実行可能であることを示した。ただしアルゴリズムの正確さについては既存のモンテカルロ木探索と比較して優位性を認めることができなかった。これは局面の特徴ベクトルの線形結合では囲碁の局面を正確に評価できないためであると考えられる。

第6章では事前に多くの知見があり学習が効果的に行えるような場合に、より効率的な学習が行えるような新しいアルゴリズムを提案し、評価関数の精度の向上を達成した。最適化する関数として、二つの局面の優劣を比較するものを learning-to-rank 学習の定式化を元に作成し、それを利用して学習することにより既存の学習と比較して評価関数の精度が向上することを対局実験において示した。また提案した学習方法は利用できる棋譜の数が少ないときに既存の方法と比較して大きな優位性を示した。

第7章では第6章と同様の設定において既存の学習方法とは別のアルゴリズムで学習を行った場合での性能を評価した。この章では AlphaGo Zero で有用であると示されたマルチタスク学習について評価を行なった。具体的には AlphaGo Zero で行なっている方策予測と局面価値関数のマルチタスク学習と、ある局面がもっともらしいかどうかを判定する discriminator と局面価値関数のマルチタスク学習とを比較し、実験的に discriminator を用いるマルチタスク学習が方策予測を行う学習と同程度の性能を持つことを示した。今後の展望として、方策予測と discriminator、局面価値関数のマルチタスク学習の性能を比較することが挙げられる。

第8章では深層ニューラルネットワークを用いて学習した評価関数の判断根拠の可視化を目指し、実際に獲得した知識の評価を行った。既存手法である Saliency Map を囲碁に対して訓練された深層ニューラルネットワークに対して適用し、出力結果が意味のあるものとなっていることを確認した。また AlphaGo などで行われている標準的な探索手法である深層ニューラルネットワークとモンテカルロ木探索の組み合わせを用いた際の価値予測の判断根拠を可視化する新たな定式化を行い、ある程度の妥当性を示した。今後は遷移関数を用いたときの議論をより精緻化することが挙げられる。またチェスや将棋などのゲームではより複雑な遷移関数となっているため、これらについての議論も必要となる。

また本研究では「棋譜を利用した効率的な評価関数の学習」と「探索時における知識の獲得」という二つの側面からゲームの知識を得る方法を論じた。この二つの側面を組み合わせにより効率的にゲームの知識を獲得するような手法も考えられる。例えば棋譜が利用できるゲームにおいて効率的に学習を行ったのち、棋譜が利用できないがゲームの性質が似ているようなゲームに対して LinUCT アルゴリズムを元のゲームの知識を利用して実行するなどの zero-shot learning [65] に近い応用が考えられる。

本研究に関連する今後の展望としては、まず LinUCT の性能の調査を実際に GGP や GVGP などで行う必要がある。この場合、特徴の抽出を行う必要があり、さらなる研究が必要となる。またここで述べた LinUCT は局面価値関数を学習するものとなっているが、方策予測を学習するような別のアプローチも考えられる。

棋譜が利用可能な状況について、より効率的な学習アルゴリズムに関する研究も今後の課題である。例えば強化学習の文脈で擬似的な訪問回数を推定し、その訪問回数が少ないような状況をより重点的に学習するという手法がある [66]。このようなアルゴリズムは今回述べた教師あり学習についても適用することができると思われ、その性能について調査する価値がある。また、別の方向性として多数のエージェントを利用し学習データの多様性を確保して学習を行なった研究がある [67]。このように学習データに多様性を持たせることが学習の効率化に寄与していると考えられる。そのほか curriculum learning のように現在の状態に対してもっとも有効な学習データを与えることなどのアプローチが考えられ、例えばこれまでに学習した重みに対してもっとも敵対的な学習データを学習させて局所解に停滞しないようにするなどの方法が考えられる。

謝辞

金子知適准教授には研究だけでなく学生生活全般に渡る指導をしていただきました。田中哲郎准教授には Game Programming Seminar においてだけでなく様々な場面で研究に関する助言をしていただきました。山口和紀教授には KY ゼミや graco ゼミ，授業等において大変お世話になりました。その他 graco ゼミ，Game Programming Seminar，KY ゼミに参加された皆様とは研究に関する身のある議論をさせていただきました。また金子研究室の皆様には多岐にわたるご指導をしていただきました。

日本学術振興会には特別研究員 (DC2) として採用していただき，特別研究員奨励費 17J09685 として支援していただきました。またこの研究の一部は学際大規模情報基盤共同利用・共同研究拠点の採択課題として行われ (jh170038-DAH, jh180067-DAJ)，東京大学情報基盤センター Reedbush スーパーコンピュータシステムを利用しました。山下宏氏には 第 6 章 において学習棋譜を作成するために囲碁プログラムを快く提供していただきました。

以上の皆様に心より御礼申し上げます。

参考文献

- [1] A. L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: *IBM J. Res. Dev.* 3.3 (July 1959), pp. 210–229. ISSN: 0018-8646. DOI: 10.1147/rd.33.0210. URL: <http://dx.doi.org/10.1147/rd.33.0210>.
- [2] Gerald Tesauro. “Temporal Difference Learning and TD-Gammon”. In: *Commun. ACM* 38.3 (Mar. 1995), pp. 58–68. ISSN: 0001-0782. DOI: 10.1145/203330.203343. URL: <http://doi.acm.org/10.1145/203330.203343>.
- [3] Murray Campbell, A. Joseph Hoane, and Feng-hsiung Hsu. “Deep Blue”. In: *Artificial Intelligence* 134.1 (2002), pp. 57–83. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/S0004-3702\(01\)00129-1](https://doi.org/10.1016/S0004-3702(01)00129-1). URL: <http://www.sciencedirect.com/science/article/pii/S0004370201001291>.
- [4] Michael Buro. “Improving heuristic mini-max search by supervised learning”. In: *Artificial Intelligence* 134.1 (2002), pp. 85–99. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/S0004-3702\(01\)00093-5](https://doi.org/10.1016/S0004-3702(01)00093-5). URL: <http://www.sciencedirect.com/science/article/pii/S0004370201000935>.
- [5] Kunihiro Hoki and Tomoyuki Kaneko. “Large-scale Optimization for Evaluation Functions with Minimax Search”. In: *J. Artif. Int. Res.* 49.1 (Jan. 2014), pp. 527–568. ISSN: 1076-9757. URL: <http://dl.acm.org/citation.cfm?id=2655713.2655728>.
- [6] Rémi Coulom. “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search”. In: *Computers and Games*. Ed. by H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 72–83. ISBN: 978-3-540-75538-8.
- [7] Petr Baudis and Jean-Loup Gailly. “PACHI: State of the Art Open Source Go Program”. In: *Advances in Computer Games - 13th International Conference, ACG 2011, Tilburg, The Netherlands, November 20-22, 2011, Revised Selected Papers*. Ed. by H. Jaap van den Herik and Aske Plaat. Vol. 7168. Lecture Notes in Computer Science. Springer, 2011, pp. 24–38. ISBN: 978-3-642-31865-8. DOI: 10.1007/978-3-642-31866-5_3. URL: https://doi.org/10.1007/978-3-642-31866-5_3.
- [8] M. Enzenberger, M. Muller, B. Arneson, and R. Segal. “Fuego—An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.4 (Dec. 2010), pp. 259–270. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2010.2083662.
- [9] 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”. In: *Nature* 529 (Jan. 2016). Article. URL: <http://dx.doi.org/10.1038/nature16961>.
- [10] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. “Mastering the game of Go without human knowledge”. In: *Nature* 550 (Oct. 2017). Article. URL: <http://dx.doi.org/10.1038/nature24270>.
 - [11] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144. ISSN: 0036-8075. DOI: 10.1126/science.aar6404. eprint: <http://science.sciencemag.org/content/362/6419/1140.full.pdf>. URL: <http://science.sciencemag.org/content/362/6419/1140>.
 - [12] Hilmar Finnsson and Yngvi Björnsson. “Simulation-based Approach to General Game Playing”. In: *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1. AAAI’08*. Chicago, Illinois: AAAI Press, 2008, pp. 259–264. ISBN: 978-1-57735-368-3. URL: <http://dl.acm.org/citation.cfm?id=1619995.1620038>.
 - [13] R. D. Gaina, D. Pérez-Liébana, and S. M. Lucas. “General Video Game for 2 players: Framework and competition”. In: *2016 8th Computer Science and Electronic Engineering (CEECE)*. Sept. 2016, pp. 186–191. DOI: 10.1109/CEECE.2016.7835911.
 - [14] Richard A. Caruana. “Multitask Connectionist Learning”. In: *In Proceedings of the 1993 Connectionist Models Summer School*. 1993, pp. 372–379.
 - [15] S. Wan and T. Kaneko. “Building Evaluation Functions for Chess and Shogi with Uniformity Regularization Networks”. In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*. Aug. 2018, pp. 1–8. DOI: 10.1109/CIG.2018.8490455.
 - [16] Louis Victor Allis. “Searching for Solutions in Games and Artificial Intelligence”. PhD thesis. University of Limburg, 1994.
 - [17] H.Jaap van den Herik, Jos W.H.M. Uiterwijk, and Jack van Rijswijk. “Games solved: Now and in the future”. In: *Artificial Intelligence* 134.1 (2002), pp. 277–311. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/S0004-3702\(01\)00152-7](https://doi.org/10.1016/S0004-3702(01)00152-7). URL: <http://www.sciencedirect.com/science/article/pii/S0004370201001527>.
 - [18] 田中 哲朗. “「どうぶつしょうぎ」の完全解析”. In: 研究報告ゲーム情報学 (GI) 2009.3 (June 2009), pp. 1–8. ISSN: 09196072. URL: <https://ci.nii.ac.jp/naid/110007993265/>.
 - [19] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. “Checkers Is Solved”. In: *Science* 317.5844 (2007), pp. 1518–1522. ISSN: 0036-8075. DOI: 10.1126/science.1144079. eprint: <http://science.sciencemag.org/content/317/5844/1518.full.pdf>. URL: <http://science.sciencemag.org/content/317/5844/1518>.
 - [20] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. “Heads-up limit hold’em poker is solved”. In: *Science* 347.6218 (2015), pp. 145–149. ISSN: 0036-8075. DOI: 10.1126/

- science.1259433. eprint: <http://science.sciencemag.org/content/347/6218/145.full.pdf>.
URL: <http://science.sciencemag.org/content/347/6218/145>.
- [21] Stephen JJ Smith and Dana S Nau. “An analysis of forward pruning”. In: *AAAI*. 1994, pp. 1386–1391.
 - [22] Levente Kocsis and Csaba Szepesvári. “Bandit Based Monte-Carlo Planning”. In: *Machine Learning: ECML 2006*. Ed. by Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 282–293. ISBN: 978-3-540-46056-5.
 - [23] Richard E. Korf and David Maxwell Chickering. “Best-first minimax search”. In: *Artificial Intelligence* 84.1 (1996), pp. 299–337. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/0004-3702\(95\)00096-8](https://doi.org/10.1016/0004-3702(95)00096-8). URL: <http://www.sciencedirect.com/science/article/pii/0004370295000968>.
 - [24] Timothy Furtak and Michael Buro. “Minimum Proof Graphs and Fastest-Cut-First Search Heuristics”. In: *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*. Ed. by Craig Boutilier. 2009, pp. 492–498. URL: <http://ijcai.org/Proceedings/09/Papers/089.pdf>.
 - [25] Kazuki Yoshizoe, Akihiro Kishimoto, Tomoyuki Kaneko, Haruhiro Yoshimoto, and Yutaka Ishikawa. “Scalable Distributed Monte-Carlo Tree Search”. In: *Proceedings of the Fourth Annual Symposium on Combinatorial Search, SOCS 2011, Castell de Cardona, Barcelona, Spain, July 15.16, 2011*. Ed. by Daniel Borrajo, Maxim Likhachev, and Carlos Linares López. AAAI Press, 2011. URL: <http://www.aaai.org/ocs/index.php/SOCS/SOCS11/paper/view/4023>.
 - [26] Hiroshi Haramoto, Makoto Matsumoto, and Pierre L’Ecuyer. “A Fast Jump Ahead Algorithm for Linear Recurrences in a Polynomial Space”. In: *Sequences and Their Applications - SETA 2008*. Ed. by Solomon W. Golomb, Matthew G. Parker, Alexander Pott, and Arne Winterhof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 290–298. ISBN: 978-3-540-85912-3.
 - [27] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach, Global Edition*. 3rd. Pearson Education Limited, Apr. 2016. ISBN: 9781292153964. URL: <http://amazon.co.jp/o/ASIN/1292153962/>.
 - [28] Donald E. Knuth and Ronald W. Moore. “An analysis of alpha-beta pruning”. In: *Artificial Intelligence* 6.4 (1975), pp. 293–326. ISSN: 0004-3702. DOI: [https://doi.org/10.1016/0004-3702\(75\)90019-3](https://doi.org/10.1016/0004-3702(75)90019-3). URL: <http://www.sciencedirect.com/science/article/pii/0004370275900193>.
 - [29] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. “A Survey of Monte Carlo Tree Search Methods”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (Mar. 2012), pp. 1–43. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2012.2186810.
 - [30] Rémi Coulom. “Computing Elo Ratings of Move Patterns in the Game of Go”. In: *Computer Games Workshop*. Ed. by H. Jaap van den Herik, Mark Winands, Jos Uiterwijk, and Maarten Schadd. Amsterdam, Netherlands, June 2007. URL: <https://hal.inria.fr/inria-00149859>.
 - [31] David Silver and Gerald Tesauro. “Monte-Carlo Simulation Balancing”. In: *Proceedings of the 26th Annual International Conference on Machine Learning. ICML ’09*. Montreal, Quebec, Canada: ACM, 2009, pp. 945–952. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553495. URL: <http://doi.acm.org/10.1145/1553374.1553495>.

- [32] Tobias Graf and Marco Platzner. “Adaptive Playouts in Monte-Carlo Tree Search with Policy-Gradient Reinforcement Learning”. In: *Advances in Computer Games*. Ed. by Aske Plaat, Jaap van den Herik, and Walter Kusters. Cham: Springer International Publishing, 2015, pp. 1–11. ISBN: 978-3-319-27992-3.
- [33] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. “Finite-time Analysis of the Multiarmed Bandit Problem”. In: *Machine Learning* 47.2 (May 2002), pp. 235–256. ISSN: 1573-0565. DOI: 10.1023/A:1013689704352. URL: <https://doi.org/10.1023/A:1013689704352>.
- [34] Sylvain Gelly and David Silver. “Combining Online and Offline Knowledge in UCT”. In: *Proceedings of the 24th International Conference on Machine Learning*. ICML ’07. Corvallis, Oregon, USA: ACM, 2007, pp. 273–280. ISBN: 978-1-59593-793-3. DOI: 10.1145/1273496.1273531. URL: <http://doi.acm.org/10.1145/1273496.1273531>.
- [35] Sylvain Gelly and David Silver. “Monte-Carlo tree search and rapid action value estimation in computer Go”. In: *Artificial Intelligence* 175.11 (2011), pp. 1856–1875. ISSN: 0004-3702. DOI: <https://doi.org/10.1016/j.artint.2011.03.007>. URL: <http://www.sciencedirect.com/science/article/pii/S000437021100052X>.
- [36] Martin Müller. “Computer Go”. In: *Artif. Intell.* 134.1-2 (Jan. 2002), pp. 145–179. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(01)00121-7. URL: [http://dx.doi.org/10.1016/S0004-3702\(01\)00121-7](http://dx.doi.org/10.1016/S0004-3702(01)00121-7).
- [37] 美添 一樹 and 山下 宏. コンピュータ囲碁 - モンテカルロ法の理論と実践 -. Ed. by 松原 仁. 共立出版, Nov. 2012. ISBN: 9784320123274.
- [38] Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michèle Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. “The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions”. In: *Commun. ACM* 55.3 (Mar. 2012), pp. 106–113. ISSN: 0001-0782. DOI: 10.1145/2093548.2093574. URL: <http://doi.acm.org/10.1145/2093548.2093574>.
- [39] Nicol N. Schraudolph, Peter Dayan, and Terrence J. Sejnowski. “Temporal Difference Learning of Position Evaluation in the Game of Go”. In: *Advances in Neural Information Processing Systems 6*. Ed. by J. D. Cowan, G. Tesauro, and J. Alspector. Morgan-Kaufmann, 1994, pp. 817–824. URL: <http://papers.nips.cc/paper/820-temporal-difference-learning-of-position-evaluation-in-the-game-of-go.pdf>.
- [40] Chris J. Maddison, Aja Huang, Ilya Sutskever, and David Silver. “Move Evaluation in Go Using Deep Convolutional Neural Networks”. In: *CoRR* abs/1412.6564 (2014). arXiv: 1412.6564. URL: <http://arxiv.org/abs/1412.6564>.
- [41] Yuandong Tian and Yan Zhu. “Better Computer Go Player with Neural Network and Long-term Prediction”. In: *CoRR* abs/1511.06410 (2015). arXiv: 1511.06410. URL: <http://arxiv.org/abs/1511.06410>.
- [42] Ronald J. Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine Learning* 8.3 (May 1992), pp. 229–256. ISSN: 1573-0565. DOI: 10.1007/BF00992696. URL: <https://doi.org/10.1007/BF00992696>.
- [43] Omid E. David, Nathan S. Netanyahu, and Lior Wolf. “DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess”. In: *Artificial Neural Networks and Machine Learning –*

- ICANN 2016. Ed. by Alessandro E.P. Villa, Paolo Masulli, and Antonio Javier Pons Rivero. Cham: Springer International Publishing, 2016, pp. 88–96. ISBN: 978-3-319-44781-0.
- [44] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680. URL: <http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf>.
 - [45] Muthuraman Chidambaram and Yanjun Qi. “Style Transfer Generative Adversarial Networks: Learning to Play Chess Differently”. In: *CoRR* abs/1702.06762 (2017). arXiv: 1702.06762. URL: <http://arxiv.org/abs/1702.06762>.
 - [46] S. Wan and T. Kaneko. “Imitation Learning for Playing Shogi Based on Generative Adversarial Networks”. In: *2017 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*. Dec. 2017, pp. 92–95. DOI: 10.1109/TAAI.2017.17.
 - [47] Christopher D. Rosin. “Multi-armed bandits with episode context”. In: *Annals of Mathematics and Artificial Intelligence* 61.3 (Mar. 2011), pp. 203–230. ISSN: 1573-7470. DOI: 10.1007/s10472-011-9258-6. URL: <https://doi.org/10.1007/s10472-011-9258-6>.
 - [48] 本多 淳也 and 中村 篤祥. バンディット問題の理論とアルゴリズム (機械学習プロフェッショナルシリーズ). 講談社, Aug. 2016. ISBN: 9784061529175.
 - [49] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. “A Contextual-bandit Approach to Personalized News Article Recommendation”. In: *Proceedings of the 19th International Conference on World Wide Web. WWW '10*. Raleigh, North Carolina, USA: ACM, 2010, pp. 661–670. ISBN: 978-1-60558-799-8. DOI: 10.1145/1772690.1772758. URL: <http://doi.acm.org/10.1145/1772690.1772758>.
 - [50] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. “Contextual Bandits with Linear Payoff Functions”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 208–214. URL: <http://proceedings.mlr.press/v15/chu11a.html>.
 - [51] 中張 遼太郎, 水上 直紀, 浦 晃, 三輪 誠, 鶴岡 慶雅, and 近山 隆. “LinUCB の 1 人麻雀への適用”. In: *ゲームプログラミングワークショップ 2013 論文集*. Nov. 2013, pp. 114–117.
 - [52] Nathan Korda, L. A. Prashanth, and Rémi Munos. “Fast Gradient Descent for Drifting Least Squares Regression, with Application to Bandits”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence. AAAI'15*. Austin, Texas: AAAI Press, 2015, pp. 2708–2714. ISBN: 0-262-51129-0. URL: <http://dl.acm.org/citation.cfm?id=2886521.2886698>.
 - [53] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning (Adaptive Computation and Machine Learning series)*. The MIT Press, Nov. 2016. ISBN: 9780262035613.
 - [54] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. “Learning to rank using gradient descent”. In: *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML) 2005, Bonn, Germany, August*

- 7-11, 2005. Ed. by Luc De Raedt and Stefan Wrobel. Vol. 119. ACM International Conference Proceeding Series. ACM, 2005, pp. 89–96. ISBN: 1-59593-180-5.
- [55] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. “A Survey of Methods for Explaining Black Box Models”. In: *ACM Comput. Surv.* 51.5 (Aug. 2018), 93:1–93:42. ISSN: 0360-0300. DOI: 10.1145/3236009. URL: <http://doi.acm.org/10.1145/3236009>.
 - [56] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps”. In: *CoRR* abs/1312.6034 (2013). arXiv: 1312.6034. URL: <http://arxiv.org/abs/1312.6034>.
 - [57] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. “How to Explain Individual Classification Decisions”. In: *J. Mach. Learn. Res.* 11 (Aug. 2010), pp. 1803–1831. ISSN: 1532-4435. URL: <http://dl.acm.org/citation.cfm?id=1756006.1859912>.
 - [58] Michael Buro. “From Simple Features to Sophisticated Evaluation Functions”. In: *Computers and Games*. Ed. by H. Jaap van den Herik and Hiroyuki Iida. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 126–145. ISBN: 978-3-540-48957-3.
 - [59] Shipra Agrawal and Navin Goyal. “Analysis of Thompson Sampling for the Multi-armed Bandit Problem”. In: *Proceedings of the 25th Annual Conference on Learning Theory*. Ed. by Shie Mannor, Nathan Srebro, and Robert C. Williamson. Vol. 23. Proceedings of Machine Learning Research. Edinburgh, Scotland: PMLR, June 2012, pp. 39.1–39.26. URL: <http://proceedings.mlr.press/v23/agrawal12.html>.
 - [60] Kokoro Ikeda and Simon Viennot. “Efficiency of Static Knowledge Bias in Monte-Carlo Tree Search”. In: *Computers and Games*. Ed. by H. Jaap van den Herik, Hiroyuki Iida, and Aske Plaat. Cham: Springer International Publishing, 2014, pp. 26–38. ISBN: 978-3-319-09165-5.
 - [61] Shih-Chieh Huang, Rémi Coulom, and Shun-Shii Lin. “Monte-Carlo Simulation Balancing in Practice”. In: *Computers and Games*. Ed. by H. Jaap van den Herik, Hiroyuki Iida, and Aske Plaat. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 81–92. ISBN: 978-3-642-17928-0.
 - [62] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. “Smooth-Grad: removing noise by adding noise”. In: *CoRR* abs/1706.03825 (2017). arXiv: 1706.03825. URL: <http://arxiv.org/abs/1706.03825>.
 - [63] Yuandong Tian, Jerry Ma*, Qucheng Gong*, Shubho Sengupta, Zhuoyuan Chen, and C. Lawrence Zitnick. *ELF OpenGo*. <https://github.com/pytorch/ELF>. 2018.
 - [64] K. Ikeno and S. Hara. “Maximizing Invariant Data Perturbation with Stochastic Optimization”. In: *ArXiv e-prints* (July 2018). arXiv: 1807.05077 [stat.ML].
 - [65] Mark Palatucci, Dean Pomerleau, Geoffrey E Hinton, and Tom M Mitchell. “Zero-shot Learning with Semantic Output Codes”. In: *Advances in Neural Information Processing Systems 22*. Ed. by Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta. Curran Associates, Inc., 2009, pp. 1410–1418. URL: <http://papers.nips.cc/paper/3650-zero-shot-learning-with-semantic-output-codes.pdf>.

- [66] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. “Unifying Count-Based Exploration and Intrinsic Motivation”. In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett. Curran Associates, Inc., 2016, pp. 1471–1479. URL: <http://papers.nips.cc/paper/6383-unifying-count-based-exploration-and-intrinsic-motivation.pdf>.
- [67] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. Garcia Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel. “Human-level performance in first-person multiplayer games with population-based deep reinforcement learning”. In: *ArXiv e-prints* (July 2018). arXiv: 1807.01281.