RocAlphaGo に基づく囲碁アルゴリズム

山川雄史
愛知工業大学情報科学部情報科学科
〒470-0392 愛知県豊田市八草町八千草 1247
E-mail: k13139kk@aitech.ac.jp

伊藤 雅
愛知工業大学情報科学部情報科学科
〒470-0392 愛知県豊田市八草町八千草 1247
E-mail: mitoh@aitech.ac.jp

1 はじめに

人工知能の研究の一つであるコンピュータ囲碁は 1960 年代に研究が始まった．当初は研究が先行していたチェスや将棋で有効な探索手法が試されたが，探索範囲が広いことなどから囲碁では有効ではなかった．

2000 年代に，モンテカルロ法と木探索を組み合わせたモンテカルロ木探索 [1] が登場し，アマチュア段位レベルまで棋力が向上したが，プロにハンディなしで勝利することはできなかった．しかし，2015 年に Google DeepMind 社の AlphaGo と呼ばれる囲碁ソフトが，機械学習の分野で主に画像認識に用いられる畳み込みニューラルネットワーク (Convolutional Neural Network:以下 CNN) を木探索に組み合わせることにより，ヨーロッパの囲碁チャンピオンに互先で 5 戦 5 勝を達成した．

本研究ではこの AlphaGo の論文 [2] と，この論文をもとに AlphaGo の畳み込みニューラルネットワーク部分を再現した RocAlphaGo[3] を参考に CNN を用いた囲碁アルゴリズムを構築し，その棋力と性能を評価する．

2 コンピュータ囲碁のルール

本研究では中国ルールを基にデータ集計を行う．囲碁では一般に，相手に取られる事のない石を “活き石” と呼び，それ以外の相手に取られてしまう石を “死に石” と呼ぶ．活き石だけで囲まれた領域を “地” と呼ぶ．中国ルールでは（地 + 活き石）を数えて勝敗を判断する．

3 モンテカルロ木探索

3.1 モンテカルロ木探索の概要

モンテカルロ木探索は，モンテカルロ法と木探索を組み合わせたアルゴリズムである．単純なモンテカルロ法では深さ 1 の一手先読みしかできない．単純モンテカルロ法のゲーム木に UCB1 アルゴリズムと UCT (UCB applied to Trees) アルゴリズムを組み込んだ手法のことを一般にモンテカルロ木探索と呼んでいる．UCB とは Upper Confidence Bound の略である．ある局面から乱数を用いて終局までシミュレーションを行うことをプレイアウトと呼ぶ．

4 畳み込みニューラルネットワーク

畳み込みニューラルネットワークは，画像認識の分野で成果を上げている多層ニューラルネットワークである．CNN は畳み込み層とプーリング層の繰り返しおよび全結合層からなる．畳み込み層は，入力とニューラルネットワークの結合重みであるフィルタとの畳み込み演算を行う．畳み込み演算を行うことをフィルタリングと呼ぶ．フィルタの重みはランダムに初期化される．この重みはニューラルネットワークの出力を決定するものである．CNN の重みは，重みを w，誤差関数を E，0.01 程度の定数である学習率を η として，誤差逆伝播法を用いて

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$

で更新される．

プーリング層は，畳み込み層の出力を，特徴を保持したまま縮小する層である．特徴を保持する方法によって呼称が異なる．2×2 程度に区切り，それぞれの区切りに最大値出力を並べた Max Pooling，区切りに平均値出力を並べた Average Pooling などがよく知られている．

全結合層は，畳み込み層またはプーリング層の出力を入力として受け取り，通常の多層ニューラルネットワークと同様に活性化関数を用いて出力を行う層である．この層は最後の畳み込み層またはプーリング層の活性化関数をネットワーク全体の出力に用いる場合には省かれることもある．

5 AlphaGo

AlphaGo が用いる 3 つのネットワークと 2 つのポリシーについて簡単に述べる．

5.1 SL policy network

SL policy network(Supervised Learning policy network) は図 1 のように 13 層の CNN によって構成され，与えられた盤面から着手の確率分布を予測するネットワークである．出力は，交点ごとの確率分布であり，最も確率が高い交点が入力盤面に対する学習した着手点となる．このネットワークに有段者の着手を用いて学習を行うことで，盤面に対する有段者の着手を予測するネットワークが作成される．

学習データには，インターネット囲碁サイトである KGS[*1] における 6 段から 9 段の人間による棋譜 16 万棋譜，3000 万局面を用いる．ネットワークの学習は，棋譜から取得した盤面を特徴量に変換したベクトルを入力とし，その盤面への着手を教師データとする．変換する特徴を表 1 に示す．SL policy network では表 1 にある Player color 以外の特徴が用いられる．

ネットワーク構成は，13 層の畳み込み層が連続して配置され，プーリング層は使用しない．1 層目から 12 層目までの活性化関数は ReLU 関数であり，13 層目の畳み込み層の

[*1] KGS https://www.gokgs.com/

表 1　SL policy network および value network の特徴

特　徴	ビット数	意　味
Stone color	3	交点の石の色または空点
Ones	1	すべて 1
Turns since	8	打たれてからの手数
Liberties	8	石のダメの数
Capture size	8	着手によって取れる相手の石の数
Self-atari size	8	着手によって取られる自分の石の数
Liberties after move	8	石を置いた時のその石のダメの数
Ladder capture	1	シチョウで相手の石をとれる
Ladder escape	1	シチョウから逃げることができる
Sensibleness	1	合法手であり，自分の眼を埋めない
Zeros	1	すべて 0
Player color	1	現在の手番が黒である

活性化関数は softmax 関数である．softmax 関数により出力される確率分布がネットワークの出力となる．

ReLU 関数とは，入力 x に対して式（1）で定義される関数である．

$$ReLU(x) \equiv max(0, x) \tag{1}$$

一方の softmax 関数は入力要素数を N，入力を $x_i (i = 1, 2, \cdots, N)$ とすると

$$softmax(x_i) \equiv \frac{\mathrm{e}^{x_i}}{\sum_{k=1}^{N} \mathrm{e}^{x_k}}$$

で定義され，出力は確率分布となる．畳み込みフィルタの更新は，出力と教師データの交差エントロピーと確率的勾配降下法を用いて，誤差逆伝播法によって行われる．交差エントロピー $\mathrm{H}(P, Q)$ は，事象の数を N，2 つの確率分布を $P = \{p_i\}$, $Q = \{q_i\}$, $i = 1, 2, \cdots, N$ とすると，

$$H(P, Q) \equiv -\sum_{i=1}^{N} p_i \log_2 q_i \tag{2}$$

で定義される．

SL policy network における重みの更新量 Δw は，誤差関数 E を式（2）の H で置き換え，教師データを式（2）の P とし，SL policy network の出力を式（2）の Q とし，確率的勾配降下法における一回の学習データ数であるミニバッチを $k = 1, 2, \cdots, m$ とすると，

$$\Delta w = \frac{\eta}{m} \sum_{k=1}^{m} \frac{\partial \log q_n^k}{\partial w}$$

となる．ここで，教師データは正解を 1，不正解をすべて 0 で表す $1 - of - N$ 表現であり，正解データの添え字を n とした．

5.2　RL policy network

RL policy network(Reinforcement Learning policy network) は SL policy network 同士の強化学習によって得られるネットワークである．ネットワーク構成は SL policy network と同じである．学習の方法は SL policy network のフィルタの重みを初期値とし，自己対戦を行う．1 戦ごとに勝敗に応じてフィルタの重みを更新し，500 戦ごとに現在のフィルタの重みを対戦相手の一覧に追加する．AlphaGo では，このネットワークは次の value network の学習にのみ使用され実際の対戦中には使用されない．

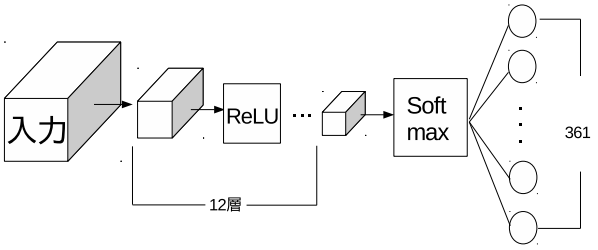

図 1　SL policy network の全体像

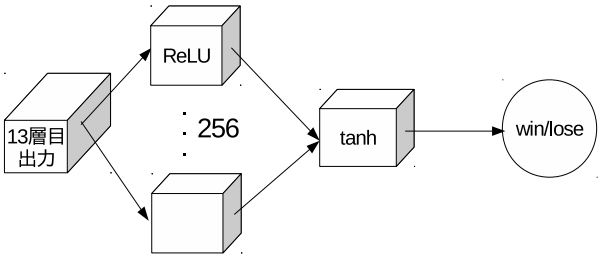

図 2　value network の全結合層

5.3　value network

value network は他のネットワークと異なり，与えられた盤面からゲームの勝敗指数を出力するネットワークである．これまで困難とされてきたコンピュータ囲碁の評価関数になり得るものである．出力は $[-1, 1]$ 区間の実数であり，値が大きいほど優勢であることを表す．

学習データは SL policy network および RL policy network によって作成される．まず，石の打たれていない初期盤面から SL policy network によってランダムな手数を打つ．SL policy network が出力した確率分布のうち，確率が高い交点に着手を行うことでゲームを進める．この時，最も確率が高い交点に着手すると常に同じ盤面になるので，確率に応じて選択されやすくなるようにランダム性を保持して着手を行う．次に合法手から一手ランダムに選択し打つ．ランダムに一手打たれた後の盤面を学習時の入力データとして記憶する．最後に RL policy network によって終局まで打つ．RL policy network の着手にランダム性はない．記憶した盤面を特徴量に変換したベクトルとそのゲームの終局時の勝敗を学習データとして，value network は学習を行う．変換する特徴は，表 1 のすべての特徴である．学習用盤面を生成するために SL policy network を用いることで学習用盤面は有段者同士の対戦で出現しやすい盤面となり，この盤面を学習した value network は実際の対戦で発生する盤面に対してよい予測ができる．

value network の構成は 15 層であり，SL policy network と同様に 13 層まで畳み込み層が連続して配置される．次に図 2 のように全結合層が配置される．14 層目は，256 個の ReLU 関数が置かれ，15 層目には 1 個の tanh 関数が置かれる．value network の学習は，SL policy network と同様に誤差逆伝播法を用いるが，誤差関数を出力とゲームの勝敗の平均二乗誤差として学習が行われる．

5.4　rollout policy および tree policy

rollout policy および tree policy は入力として盤面を受け取り，softmax 関数によって各交点の着手確率を出力する

表 2　rollout policy および tree policy の特徴

特　徴	意　味
Responce	レスポンスパターンに一致
Save atari	アタリを助ける手
Neigh bour	直前の着手の周り 8 目
Nakade	中手
Response pattern	直前着手が 12 目パターン
Non-response pattern	空点の周り 3×3 パターン
Self-atari	自分の石を取れる様にする手
Last move distance	2 手前までの，距離 17 まで
Non-response pattern	空点の周り距離 2 のパターン

ネットワークである．13 層の CNN を使用した SL policy network も同様に各交点の着手確率を出力するが，これら 2 つのネットワークは SL policy network と比較すると高速に動作する点が最大の特徴となっている．

ネットワークの入力は取得した盤面を表 2 の特徴に従って着手点ごとに重みとして数値化したものである．rollout policy は表 2 の上 6 つの特徴のみを使い，tree policy は 9 つ全ての特徴を用いる．tree policy は Non-response pattern を 2 つ保持するが，意味するパターンは異なる．1 つは空点周りの 3×3 パターンであり，もうひとつは空点周りの距離 2 のパターンである．

ネットワークの学習では，着手点ごとに特徴の重みの総和を softmax 関数に入力し，出力が最も大きい着手点と，学習データの着手点が合致しているかを調べて交差エントロピーを最小にするように学習する．

5.5　APV-MCTS

APV-MCTS は，Asynchronous Policy and Value Monte Carlo Tree Search の略であり，AlphaGo で用いられる木探索アルゴリズムである．RL policy network を除く 4 種類のネットワークを活用し，並列処理を行う．概略を図 3 に示す．APV-MCTS の動作について述べる．APV-MCTS では，$\{P(s,a), N_v(s,a), N_r(s,a), W_v(s,a), W_r(s,a), Q(s,a)\}$ の 6 つの値を木探索のノードごとに保持する．s は盤面，a は着手である．$P(s,a)$ は SL policy network および tree policy によって出力される交点ごとの着手確率である．SL policy network では，すでに石が置かれている交点にも着手確率が設定される．$N_v(s,a), N_r(s,a)$ はそれぞれ value network およびロールアウトによって評価された回数であり，$W_v(s,a), W_r(s,a)$ はそれぞれ value network およびロールアウトの評価である．

$Q(s,a)$ は盤面の value network とロールアウトによる評価であり

$$Q(s,a) \equiv (1-\lambda)\frac{W_v(s,a)}{N_v(s,a)} + \lambda\frac{W_r(s,a)}{N_r(s,a)},\ 0 \leqq \lambda \leqq 1$$

で定義される．ここで，λ は value network とロールアウトの評価の重みであり，$\lambda = 1$ であれば rollout，0 であれば value network の評価のみを使用する．論文 [2] では 0.5 が一番良いとされている．

APV-MCTS の探索手順では，まず，

$$a_t = \arg\max_a (Q(s_t,a) + u(s_t,a))$$

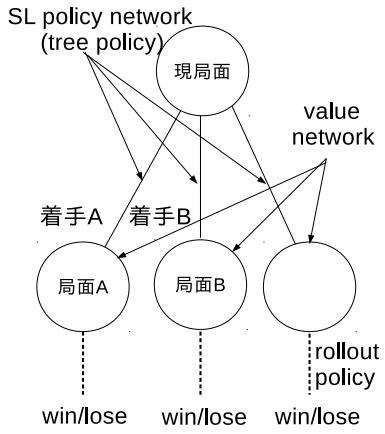

図 3　各ネットワークの用途

から得られる一手を選択し，探索木を降りる．ここで t は時間ステップであり，1 着手を 1 ステップとし，上限を終局面 s_T とする．$u(s,a)$ はボーナス値であり，

$$u(s,a) \equiv c_{puct} P(s,a) \frac{\sqrt{\sum_b N_r(s,b)}}{1 + N_r(s,a)} \tag{3}$$

で計算される．ここで，c_{puct} は 0.5 程度の定数であり，$\sum_b N_r(s,b)$ は探索木において盤面 s と同じ深さのノードのロールアウトによって評価した回数の和である．この式 (3) によって，探索回数が少ないノードのボーナス値は大きくなり，選択されやすくなる．これを葉ノード s_L にたどり着くまで繰り返す．また，枝を降りる際に，Virtual loss[4] を適用する．これは，並列木探索で用いられる手法であり，降りる枝の勝利数を意図的に減らすことで他の探索スレッドが自スレッドと同じ枝を探索し難くし，探索範囲を広げることができる．降りたノードが葉であり，value network によって一度も評価されたことがなければ，value network による処理キューに盤面 s_L を追加する．そして，rollout policy を使用して s_L から終局 s_T まで打たせ，ロールアウトによる評価

$$z_t = \pm r(s_T)$$

を得る．ここで，z_t はノード s_t におけるロールアウトの評価である．$r(s_T)$ はノード s_L の手番から見た終局時のゲームの勝敗であり，勝ちが 1，負けが -1 である．$\pm$ は手番によって勝敗が反転することを表す．ロールアウトの評価が得られたら，これまで降りてきたノードに対して，ノード s_L から順に $W_r(s,a)$ に評価値を加算し，$N_r(s,a)$ を 1 だけ増加させる．最後に，Virtual loss で変更した $W_r(s,a)$ および $N_r(s,a)$ を元に戻す．value network の評価も同様に，ネットワークによる評価が終了すれば，降りた葉ノード s_L から根ノードまで $W_v(s,a)$ および $N_v(s,a)$ を $W_r(s,a)$ や $N_r(s,a)$ と同様に更新する．

このようにしてあるノード s_t が複数回評価され，ロールアウトによって評価された回数 $N_r(s,a)$ が設定された閾値 n_{thr} を超えたとき，ノード s_t の合法着手を枝としてノード s_t に追加し，s_t を展開する．結果として，探索木は下に拡張されることになる．このとき，s_t の次の盤面を s' とすると，追加されたノードは

$N(s',a) = N_r(s',a) = 0,\ W(s',a) = W_r(s',a) = 0,$
$P(s',a) = p_\sigma(a \mid s')$

のように初期化される．ここで，$p_\sigma(a \mid s')$ は SL policy network による現在の盤面での着手の確率分布であり，value network と同様に処理キューに追加され，評価を待つ．

さらに，$p_\sigma(a \mid s')$ の処理を待つ間に tree policy $p_\tau(a \mid s')$ を使用して暫定的な着手確率を求め，探索を続ける．$p_\sigma(a \mid s')$ の評価が計算された時点で $P(s',a)$ を上書きする．ノードを展開する閾値 n_{thr} は SL policy network の処理キューに追加される盤面の数と実際に GPU によって処理される盤面のバランスが取れるように動的に変更される．最終的に，事前に設定された終了条件を満たした時点で一番多く選択されている着手を次の一手として選択する．

6 RocAlphaGo

RocAlphaGo とは，AlphaGo の畳み込みニューラルネットワークを再現したプロジェクトである．Python によって実装されている．RocAlphaGo では SL policy network などの CNN を使用したネットワークは開発されているが，rollout policy などは開発されていない．GTP による対局インターフェースも存在しているが，広く使われている囲碁 GUI ソフトの GoGui[*2] と盤面表現が異なる．GoGui の有用な機能を利用するには GoGui の盤面と整合性をとる必要があり，本研究では，RocAlphaGo が利用している Python ライブラリ pygtp に修正を施した．

7 数値実験

本研究では，2 つの数値実験を行った．まず，SL policy network と GNU Go Ver.3.8[*3] で対局を行い，その勝率で棋力を確認した．RocAlphaGo を参考に Python 言語を用いて SL policy network を作成し，学習データ数およびネットワークのフィルタ数を変動させ，19 路盤で 1000 局対戦させた．学習には，KGS の有段者の棋譜を利用した．学習は各棋譜数それぞれ，データを一巡することを 1 エポックとして，4 エポックずつ行った．木探索を用いず，SL policy network の出力である確率分布のうち最も高い確率を持つ候補手を着手点とした．勝率の変化を表 3 に示す．学習棋譜数 106888 棋譜以外では，フィルタ数を増やすと勝率が上昇することが分かる．

次に，4 つのネットワーク SL policy network，value network，rollout policy，tree policy を使用した木探索囲碁アルゴリズムを Python で実装し，GNU Go Ver.3.8 と対戦させた．SL policy network と value network の構築は RocAlphaGo を参考にしたが，rollout policy および tree policy については GoSample2[*4] と呼ばれるプロジェクトを参考にした．GoSample2 は Python ではなく C++ によって rollout policy などを再現するプロジェクトである． RocAlphaGo と GoSample2 では保持しているデータ構造や情報が異なるため，RocAlphaGo で利用可能な情報のみで GoSample2 のアルゴリズムが動作するよう Python で書き換えた．しかし，実装した rollout policy は非常に低速であり，十分な対戦データが収集できなかった．AlphaGo は初期盤面から 1 秒間に 1 万回のロールアウトを行うが，本研究で実装した rollout policy では 1 秒間に 1 回のロールアウトしか実行できなかった．

表 3　SL policy network の棋譜数とフィルタによる勝率

学習棋譜数	フィルタ数		
	128	196	256
53334 棋譜	41.4%	55.6%	69.6%
106668 棋譜	59.3%	58.7%	56.9%
160000 棋譜	58.6%	64.0%	65.4%

8 おわりに

本研究では，RocAlphaGo に基づく囲碁アルゴリズムを GNU Go と対戦させた．学習棋譜数やフィルタ数を増加させた際，棋力向上を 53334 棋譜と 160000 棋譜で確認できた．106668 棋譜では確認できなかった．学習エポック数が足りず，学習精度にかなりの誤差が生じたと推察される．SL policy network 以外のネットワークや木探索を利用した囲碁アルゴリズムも実装したが，rollout policy の動作に時間を要し十分なデータを取得するには至らなかった．今後の課題である．

謝 辞

尚，本研究の一部は，文部科学省 科研費 基盤研究 (C) No. 16K00510 の助成を得た．ここに謝意を表する．

参考文献

[1] Rémi Coulom, “Computing Elo Ratings of Move Patterns in the Game of Go”, Computer Games Workshop 2007 (CGW2007), Amsterdam, The Netherlands, 2007.

[2] David Silver, Aja Huang, et al., “Mastering the game of Go with deep neural networks and tree search”, Nature, Vol. 529, pp. 484 –489, January 2016.

[3] RocAlphaGo, https://github.com/Rochester-NRT/RocAlphaGo

[4] Guillaume M.J.-B. Chaslot, Mark H.M. Winands, and H. Jaap van den Herik, “Parallel Monte-Carlo Tree Search”, in Proceedings of the 6th International Conference on Computers and Games (CG2008), Springer, *Computers and Games*, Vol. LNCS5131, pp. 60–71, 2008.

[*2] GoGui https://sourceforge.net/projects/gogui/
[*3] GNU Go http://www.gnu.org/software/gnugo/
[*4] GoSample2 https://github.com/TadaoYamaoka/GoSample2