Miacis アピール文書

迫田真太郎

平成 31 年 3 月 26 日

1　概要

Miacis は深層ニューラルネットワークによる評価関数を持ち、モンテカルロ木探索によってゲーム木探索を行う AlphaZero[1] ライクな将棋ソフトです。強化学習のみによる学習を行っており、現在までのところランダムに初期化したパラメータから約 32 時間の学習で Lesserkai[2] に対して勝率 9 割程度になることが確認できています。

2　使用フレームワーク、参考にしたもの等

大会ルールで定められるライブラリは利用していません。学習も平手初期局面からの自己対局による強化学習を行っており、棋譜や定跡等の外部データへの依存もありません。

深層学習フレームワークとしては PyTorch[3] の C++API(LibTorch) を利用し、C++のみで学習から推論までを行います。

モンテカルロ木探索の実装においては「将棋 AI で学ぶディープラーニング」[4] を基本とし、直列的な効率化についてはねね将棋 [5] およびそれに関する山岡さんのブログ記事 [6] を参考にさせていただきました。

将棋の盤面を管理する部分のプログラムに関しては去年まで作っていた将棋ソフト海底 [7] をそのまま利用しており、海底の実装においてはれさぴょん [8]、技巧 [9]、Apery[10]、やねうら王 [11]、Novice[12] などのソフトを参考にさせていただきました。

3　現状

プログラムの詳細はほぼ AlphaZero と同様です。畳み込み層を 2 回用いる Residual ブロックを多数重ねたネットワークにより方策と状態価値を同時に学習し、探索ではプレイアウトを用いないモンテカルロ木探索を行っています。

冒頭で述べたように Lesserkai に対して勝率約 9 割となるモデルの学習、探索が可能になっています。この検証対局は 1 手 1 秒、使用 GPU は GTX1080ti 一枚で行いました。モンテカルロ木探索の探索速度は平手初期局面で 15,000 局面/秒ほどとなっています。探索中にはモンテカルロ木探索とは別にルート局面に対して簡単な詰み探索を行っており、7 手詰くらいまでは読み切れるはずです。

学習では AlphaZero アルゴリズムをそのまま実行できるほどの計算資源を持っていないため、計算量やメモリ使用量を抑える変更を行っています。現在のところ 100,000 ステップまでの学習によって学習の成否を判断しており、その条件の下での AlphaZero との比較を表 1 に示します。この表に記載していないもの、たとえば各探索におけるルート局面に対するディリクレノイズの付与などは AlphaZero と同様に行っています。以下は表中の各項目について表中の各項目についての簡単な解説と所感になります。

3.1　計算量の削減

ブロック数、フィルタ数を少なくすることで計算量を抑えています。これは単に学習時の自己対局生成速度を上げるための変更ですが、ブロック数とフィ

表 1: AlphaZero(100,000 ステップまで) と Miacis の学習比較

内容	AlphaZero	Miacis
計算資源	第 1 世代 TPU 5000 基 第 2 世代 TPU 16 基	RTX2080ti 2 基
ブロック数	19	10
フィルタ数	256	64
バッチサイズ	4096	64
学習率	0.2	0.02
学習時間	約 2 時間	約 33 時間
リプレイバッファサイズ	1,000,000 局	1,000,000 局面
Policy の教師	探索回数を正規化した分布	実際に選択した指し手の Onehot 分布
推定 Elo レート	3600 ほど	1100 ほど

ルタ数についての簡単な検証が select766 氏によって行われており [13]、探索速度が上がることによって多少のモデル精度の低下は補えるのではないかと考えています。

3.2 学習スレッドと自己対局スレッドの速度バランス

学習の速度と自己対局によるデータ生成速度のバランスが重要であると感じており、バッチサイズを下げ、学習スレッドは 1 ステップごとに 1 秒間スリープすることで相対的に自己対局側の速度が高まるようにしています。バッチサイズを小さくする場合、学習率も小さくしなければ学習が上手く進まないようです。また 1 ステップごとにスリープを挟むため学習時間は非常に多くなっています。

3.3 使用メモリの削減

使用メモリ量にも制約があるためリプレイバッファサイズを小さくする、Policy の教師を実際に選択された手の Onehot 分布にするなどで対処しています。リプレイバッファの縮小による悪影響には次の段落で触れます。Policy の教師を Onehot 分布にすることに関してはその影響かは断定できませんが、いくらか Policy が偏りやすいようには見えます。たとえば学習したモデルの平手初期局面でのネットワークの出力 Policy は小数点第 2 位以下を切り捨てして

- ▲２六歩:96.8%
- ▲７六歩:2.4%
- ▲５八金右:0.1%
- その他:0.0%

となっています。AlphaZero の論文中 Fig.3 には平手初期局面から 2 手後の局面パターン割合が学習の進行と共にどう変化するか示されており、そこから読み取る限りこれほど極端な状況にはなっていないようです。

3.4 最終的な性能

推定 Elo レーティングの推定は、AlphaZero については論文 [1] 中の Fig.1 にあるグラフから 100,000 ステップ時点の値を読み取り、Miacis については uu-unuuun 氏のレーティングサイト [14] から Lesserkai のレートを 713 としてそれに対して 9 割 (+381 程度) であることから算出しました。AlphaZero と完全に同じではありませんが、似たアーキテクチャを用いている論文 [15] では、やはり Actor(自己対局スレッ

ド) は多ければ多いほうがよく、リプレイバッファも大きい方が性能が上がると報告されているため、そのあたりを犠牲にしている Miacis では性能が悪化しているのだと考えられます。

4　今後の展望

ここまでの実装で性能は悪くともおおむね学習は進むプログラムができたことが確認できました。ここから選手権当日までは AlphaZero アルゴリズムを単に模倣するだけでなく、計算資源が少なくとも精度の高い学習ができるように改良する方法を模索していきたいと考えています。今のところ案として考えているのは以下の 3 つとなります。

4.1　データ選択方法の改良

AlphaZero アルゴリズムでは自己対局スレッドがデータを一度リプレイバッファに貯めて、学習スレッドがリプレイバッファからランダムにデータを選んで学習を行います。この学習スレッドによるデータの選択方法として、ランダムではなく前回の学習における損失によって重み付けを行う優先順序付き経験再生 (Prioritized Experience Replay)[16] という手法があり、アタリゲームなどで高い性能を示しています。先日山岡さんのブログで「弱点の克服」[17] として記述された、評価値が高い状態から負けた局面を初期局面集に追加して学習する手法もいくらか似た効果があるのではないかと感じられ、将棋でも有効なのではないかと思われます。

4.2　深層学習の改良

深層強化学習の学習では得られる表現 (最終層手前の層の活性) の相関が強く、そのため学習が上手く進まないと主張する論文 [18] があります。これによると表現の相関を弱めるように正則化を加えることで広く性能が向上するそうです。この手法をそのまま適用して性能が向上するかはわかりませんが、一般の深層 (強化) 学習の知見を利用したアプローチも探していくべきだと感じています。

4.3　探索部の改良

ゲーム木探索 (特に MCTS) と強化学習の類似性を指摘する論文 [19] について以前自分のブログで触れました [20]。この論文ではそこから導き出されるものとして Sarsa-UCT(λ) という MCTS の改良案を示しており、プレイアウトを使う探索で簡単なゲームに適用した場合には、特に探索回数が少ない状態において性能が上がるとされています。これが直接的に将棋の AlphaZero 的なプレイアウトを行わない探索にも有効であるかはわかりませんが、MCTS にはまだ改良の余地が残されているのではないかとも考えています。

参考文献

[1] David Silver, Thomas Hubert, Julian Schrittwieser et. al., "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," Science, vol.362, issue 6419, pp.1140-1144, 2019.

[2] 「将棋所:ダウンロード」, http://shogidokoro.starfree.jp/download.html

[3] 「PyTorch」, https://pytorch.org/

[4] 山岡忠夫 (2018)「将棋 AI で学ぶディープラーニング」マイナビ出版.

[5] 「select766/neneshogi: NEural NEtwork Shogi」, https://github.com/select766/neneshogi

[6] 「将棋 AI の進捗 その 22(探索と評価の直列化) - TadaoYamaoka の日記」, 将棋 AI の進捗その 22(探索と評価の直列化)-TadaoYamaoka の日記

[7] 「SakodaShintaro/kaitei_history」, https://github.com/SakodaShintaro/kaitei_history

[8] 「コンピュータ将棋のアルゴリズム」, http://usapyon.game.coocan.jp/ComShogi/index.html

[9] 「gikou-official/Gikou: 将棋ソフト「技巧」」, https://github.com/gikou-official/Gikou

[10] 「HiraokaTakuya/apery: a USI Shogi engine.」, https://github.com/HiraokaTakuya/apery

[11] 「yaneurao/YaneuraOu: shogi engine(AI player), stronger than Bonanza6 , educational and tiny code(about 2500 lines) , USI compliant engine , capable of being compiled by VC++2017」, https://github.com/yaneurao/YaneuraOu

[12] 「kumasan1011/Novice_mini: Novice mini is a USI Shogi Program.」, https://github.com/kumasan1011/Novice_mini

[13] 「CNN の複雑さと nps と棋力【コンピュータ将棋】 - select766 's diary」, https://select766.hatenablog.com/entry/2019/03/09/123505

[14] 「shogi-engines」, http://www.uuunuuun.com/

[15] Daniel Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt and David Silver, "Distributed Prioritized Experience Replay," International Conference on Learning Representations, 2018.

[16] Tom Schaul, John Quan, Ioannis Antonoglou and David Silver, "Prioritized Experience Replay," International Conference on Learning Representations, 2016.

[17] 「将棋 AI の進捗 その 28(弱点の克服) - TadaoYamaoka の日記」, http://tadaoyamaoka.hatenablog.com/entry/2019/03/23/114849

[18] Borislav Mavrin, Hengshuai Yao and Linglong Kong, "Deep Reinforcement Learning with Decorrelation," arXiv:1903.07765, 2019.

[19] Tom Vodopivec, Spyridon Samothrakis and Branko Šter, "On Monte Carlo Tree Search and Reinforcement Learning," Journal of Artificial Intelligence Research vol.60 pp.881-936, 2017.

[20] 「On Monte Carlo Tree Search and Reinforcement Learning を読んだ - 水たまり」, https://tokumini.hatenablog.com/entry/2019/03/21/112149