第３３回世界コンピュータ将棋選手権「アストラ将棋」アピール文書

令和５年３月
恒岡 正年

1. 全体の構成
　dlshogi の探索部、学習部を利用。
2. 独自に実装した部分
　Network 構造、学習方法、model の生成の工夫。（詳細は後述）
3. 開発動機
　ディープラーニングの特に学習部に興味があり、「強い将棋ソフトの創り方」という書籍の内容をトレースする事で開発をはじめた。
　したがって、自前の model を作る事に注力している。
4. 主な開発内容
　独自構造の model の作成・学習方法
　棋譜生成：6000po 程度、及び 15 秒+F1.2 秒程度の物を中心に現在 20 数万の棋譜作成
　学習には、自己生成した棋譜以外に上記書籍の学習用データを利用
　AobaZero の棋譜も一時期利用したが、現在は未使用。
　探索パラメータの調整(手作業・optuna 利用)
5. 探索ロジックの検討(UctSearch.cpp の改造)
　標準の UCB 値の計算式は n が十分に大きい時に最良の計算式となるが、限られた試行回数では必ずしも最適な方法ではないと思う。
　オリジナルの計算式を独自の式と置き換える。(実装は終了。)
　探索パラメータの再調整が必要になる。
(3/30 追記:調整の途中だが better なパラメータセットを１つ見つけたので本番で使えそう。予選と本選で使い分けるかもしれない。)
6. 今後の開発予定(~4 月末)
　指し手選択ロジックの追加(後述)
　思考時間制御(後述)
　定跡作成

---

7. Network 構造

Network を次の 3 つに分ける。

(1)データ入力部 (2)ResNET 部 (3)データ出力部

1）データ入力部

3 つの情報の合流位置とデータのサイズを振って実験した。

合流位置に関してはオリジナルの物よりも優れた構造は見つけられなかった。(同等の物は有った。)

データサイズは、9x9 以外にも 11x11、11x9 を試した。Play Out 数固定での評価では有意に強くなるが、探索速度の低下が大きく探索時間一定の条件では弱くなった。ネットワーク層数またはカーネル数を増やす方が良い。

将来的にネットワーク層数またはカーネル数を増やしても改善効果が小さくなった場合に、データサイズ 11x11 を試してみたい。

2）ResNET 部

オリジナルの ResNET に限らず何通りかの形状のネットワークを試したが、最終的にオリジナルと同じ Conv2d() 2 層の ResNET にした。

データサイズは 9x9。(11x11、11x9 も試したが不採用)

カーネルサイズは 3x3 のみ評価。

データ入力側と出力側でカーネル数が異なる構造を採用。

ResNET 部の層数は 25。

3）データ出力部(Policy/Value Network の分岐後)

オリジナルの構造に Conv2D()を追加している。

ResNET を何層か追加する実験も行ったが、優位性は認められなかった。

4）その他

活性化関数は 主に ReLU を、最も出力に近い Conv2D()にのみ SiLU を使っている。

8. 学習方法

1）学習率(lr)：0.2 から始めて 0.000000012 まで等比数列的に減少させ学習した。減少率は主に 0.85。最後の 6 エポックは学習率を固定値とした。

2）バッチサイスは 512 から初めて GPU メモリの許す限り順次大きくしている。

3）マクロバッチを実装し、最終的には 12 倍まで学習単位を大きくして

いる。8 倍～12 倍に最適値が有りそう。今回の model は 12 倍を採用。
(FP16 だとこれより大きくすると精度が足りない。(学習率) x (バッチサイズ) x (マクロバッチ倍率) が一定値を超えると収束付近で学習効果が無くなる様に思う。)

４）小さめの学習セットで 1 エポックの学習を 3 ～ 5 回実行し、最も D 値 (=Loss-(PolicyAcc+ValueAcc)*0.5)が小さくなった物を採用し以降の学習を行っている。初期値の分布の素性の良さそうな物を選ぶ。

５）25 エポックまでに 3 回、入力に近い部分の ResNET を入れ替えて学習する。これにより収束を早める効果を狙っている。

６）26 エポック以降も D 値をモニターし、D 値の減少が期待値以下の場合に、意図的にイレギュラーなデータを与える。データの与え方は 2 種類の手法を用意した。

特定の次元が局所解に陥っている可能性を考慮しそこから脱出するのを期待している。

9. model の生成の工夫

学習済のモデルは、最も Loss 値が低い、または最も Accuracy が高い物が勝率が高いモデルとは限らない。これは、Loss 値や Accuracy 値は出現頻度の高い局面での正解率に強く依存し、出現頻度の低い局面での正解率の寄与割合が低いためと考えた。将棋の勝敗は出現頻度の低い局面で正しい手を指せるかにも依存する。

すべてのモデルでベンチマークを取るのを回避しつつ、この問題の影響を低減する工夫を行った。

10. その他(今後の予定)

1）指し手を決定する前に候補手を 2 つに絞り、どちらを選ぶかのロジックを追加する。訪問数が最大の物を選ぶのではなく、勝率も勘案する様にする。思考時間の終盤でその時点での本線に問題がある事を発見した時に手遅れになる場合があるのを少しでも回避したい。

この様な場合通常は思考時間を延ばすが、持ち時間を使い切っている時に対応できない＆持時間の消費が増える。そのため思考時間を増やさずに対応できる様にしたい。
python-dlshogi 版で実装しており有効と考えている。

2）一手の思考時間を、現在までの手数・評価値を考慮して決定する様にする。中盤の重要な局面で思考時間を長くする様にしたい。
これも python-dlshogi 版で実装しており有効と考えている。

-以上-