第３５回世界コンピュータ将棋選手権

「あすとら将棋」アピール文書

あすねこ

- 開発内容：WCSC34の延長線上の開発
- 自己生成棋譜＝＞～118万棋譜(2025/03末時点, 千日手・持将棋を除く)
  - 30~150手を利用する場合、0.77億局面(重複を除く)
  - GCTのhcpeファイルとマージして合計 2.7億局面(重複を除く)を学習に利用
- rtx4090に最適化されたNetwork: AstraNET(自称)
- 教師データの学習時選別（新規）
- ReEvalによる追加学習（新規）
- USB計算式の変更（新規）
- 定跡作成：定跡の追加～合計50万局面(2025/04/E 時点)...（継ぎ足し）
- 手数＆評価値による探索エンジンの切り替え

WCSC35での開発内容

- WCSC34 で行った事を踏襲
- 色々なmodel 構造を試して学習可能性・速度・性能を評価した（基本的な理解を深める実験を行った）
- 学習時の工夫
  - 複数回の異なる乱数初期値での試行＆選別（学習開始時）
  - 学習時のネットワークのブロック順序の入れ替え（学習初期）
  - 異なる特徴を持つ学習データ・学習条件の交互使用（学習末期,lr が小さい事）
  - 学習時に教師データを評価し取捨選択
  - ReEvalによる追加学習
- 学習後の追加工(学習済のmodelの加重平均＆乱数による揺らぎの付加：後述)
- 探索部：UCB値計算式の変更
- 探索エンジンの切り替え(Ponderと両立できなかった)

生成棋譜

- ベンチマークを兼ねた棋譜生成
  - NNUE vs. NNUE（5%位）
  - NNUE vs. DL（85%位）
  - DL vs. DL（10%位）
- DL系は 7000po~25000po程度で順次大きくしてきた
- NNUE系は 当初 3M nodes, 最近は 6~9M nodes
- NNUE系のエンジンは4種類利用

  （Hao,Tanuki-dr4,Daigorilla5,Suiden3)
  - dl側の勝率が55%~60%になる様に node数/po数を調整
  - 複数の対戦相手でベンチを行うのは重要
  - 特定の対戦相手のみで棋譜生成すると、指し手がその相手に似てくる

生成棋譜(cont.)

- Tanuki-dr4の棋譜が最も多い
- 複数AIの意見が分かれる局面では、Tanuki-dr4と同じ手を指す場合が多い
  - modelは、教師データの影響を強く受けている
  - 良いmodelを得るためには、良質な教師データを用意する事が重要では？
- 棋譜の質は重要！（数だけではない！）
- 指し手は教師データの影響を強く受けており、将棋の神様には ほど遠い（たどり来て未だ森林限界）

- **AstraNET(自称)のrtx4090への最適化（後述）**

- **教師データの学習時選択**
  - 学習中のmodelと意見の合わない学習データの削除(～5%)
  - どのようなデータを不採用とするかが難しい
  - 学習時間が概ね２倍になる
  - 効果は有ったがそれ程大きくはなかった=> ReEvalの方が良い

- **ReEvalによる追加学習**
  - 山岡氏のコード使用
  - 効果あり(劇的という程ではなかった)
    - 元になる model の規模と学習中のmodel の規模が大きくは違わないためではないか
    - 探索を行わない ReEvalの限界の可能性もある

定跡作成

- WCSC34 と同じ２段階作成(詳細はwcsc34のアピール文書で)
  - 先手が単調に勝った棋譜を沢山集める(R4400以上)
  - 最も多く選択された手は先手にとって悪い進行ではない可能性が高い
  - 2段階目でその進行が正しいか検証する（後手も同様）
- WCSC34 で使用した定跡に棋譜を追加
- 2025/04/末 時点で50万局面程度
- 後手番では相手により２種類の定跡を使い分ける予定

以下、詳細

1)AstraNET(自称)について

2)Modelの追加工について

3)スレッド生成並列化による高速化

4)UCB値計算式の変更

AstraNETの概要

- rtx4090の場合、kernel数256のResNETに対して同一の性能であれば70%程度高速
- 高性能GPUでの効果が大きい（効果は使用するGPUにより変わる）
  - 計算ユニットが多い(＝並列化の粒度が大きい)場合効果大
  - データ転送速度が速い（3090, 4090 & PCIE-4以上のMB) 場合効果大
- 適用する対象(将棋・囲碁等)にも依存する（次項）
- ブロック数が12ブロック以上で問題なく学習できる事を確認

  それ未満で学習できるかは未確認

- 2023年9月頃から採用

AstraNETの考え方(kernel数)

1) 過剰な表現力の削減

kernel数はデータ量に応じて最適な値がある

将棋のデータ量は9x9=81なので256だと大きすぎる（推測）

囲碁だと19x19(将棋の約４倍)なので256でも問題ないのでは（推測）

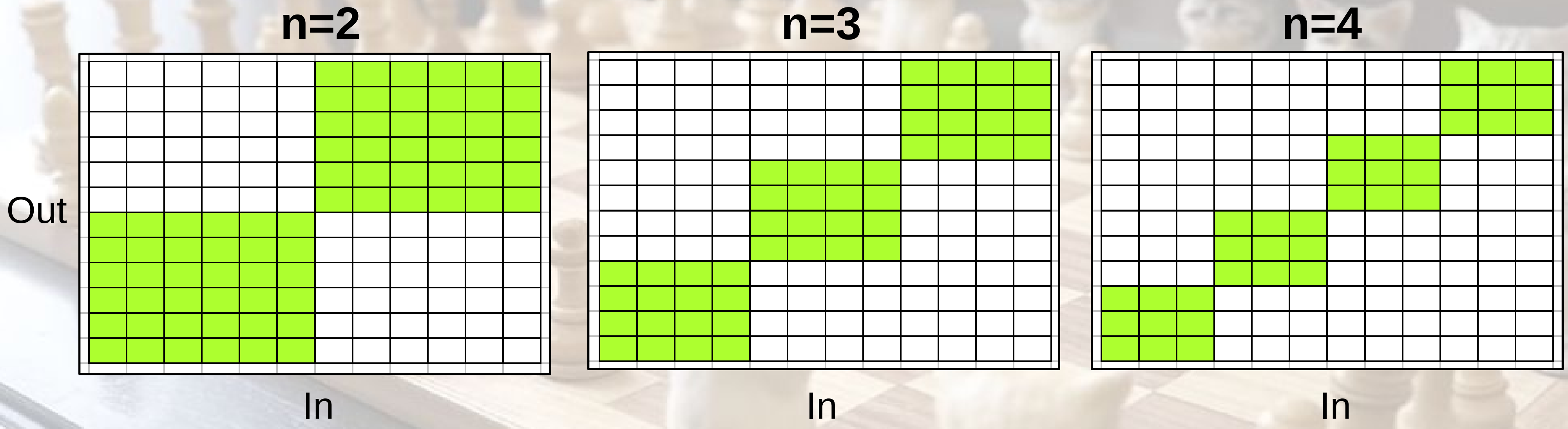
2) NVIDIAのGPUの並列化最小単位はGPU毎に異なる

使用するGPUで効率的に計算できるkernel数を採用する

rtx4090 では128が好適

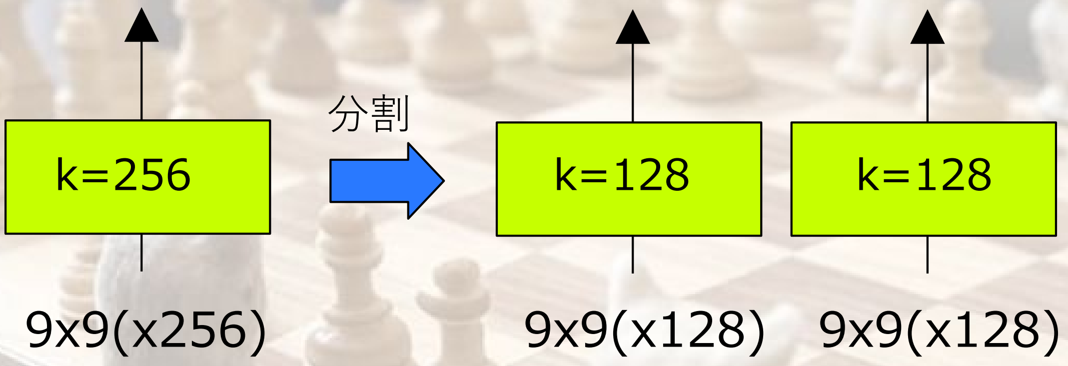
(1)と(2)の両方を満たすkernel数でネットワークを構成する

AstraNETの考え方(計算量の削減)



Conv2D()の対角部分(**緑色の部分**)のみを計算する事により計算量を削減する

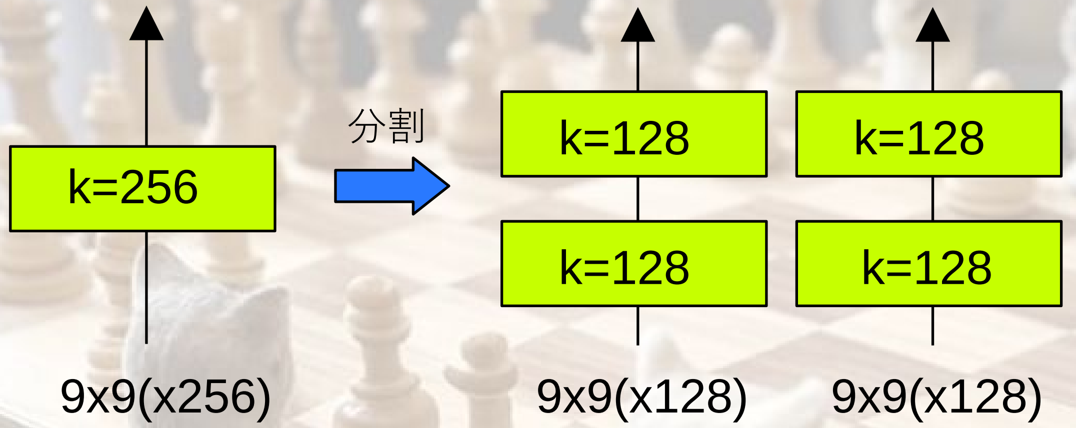
AstraNET(１)



データフロー：同じ
計算量：1/2
フィルター数：同じ
層数：同じ
非線形変換：1回(同じ)

大きなkernel数のConv2D()を、使用する<u>GPUに最適なkernel数で構成されたResNET</u>で表現する。RTX4090では <u>k=128</u>。

AstraNET(２)



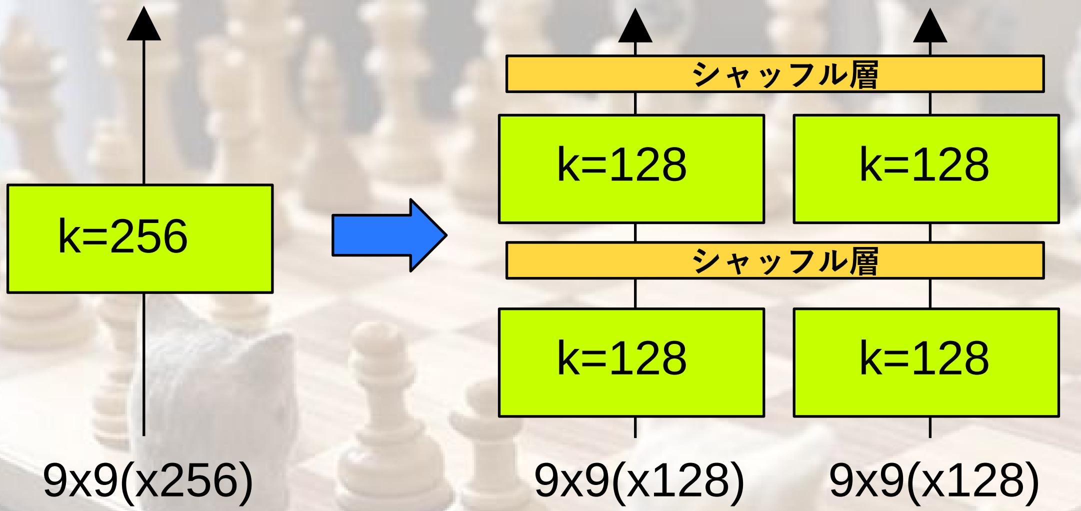
データフロー：同じ
計算量：同じ
フィルター数：2倍
層数：２倍
非線形変換：2回(2倍)

・同じ計算量で、フィルター数を２倍、非線形変換の回数を２倍にする事ができ表現能力が増す

・層数が２倍になるので、元の3x3のフィルターよりも広範囲の5x5の情報を得ることができる

・ただしこの構造は、層数が多くなるにしたがって学習終盤にLossが十分に下がらないのではないかという懸念がある

AstraNET(３)



データフロー：同じ
計算量：同じ
(シャッフル層分を除く)
フィルター数：2倍
層数：２倍
非線形変換：2回(2倍)

必要に応じて シャッフル層を追加する(必ずしも毎層入れる必要はない)
シャッフルの仕方は、何通りか考えられる
(シャッフルの仕方によりスピード(nps)と必要GPUメモリ量が変わる)

シャッフル層のオーバーヘッドが発生するが、rtx4090の場合 k=256と比較して
同じ層数でほぼ同じ性能が出て1.7倍程度高速だった

AstraNET(cont.)

前項では n=2 の例を示したが、n=2,3,4,6 で上手く学習できる事を確認

a) (9x9,k=128x2,36b) : n=2 ...k=256 x 36 block 相当、前項の例
b) (9x9,k=128x3,24b) : n=3
c) (9x9,k=128x4,18b) : n=4
d) (9x9,k=128x6,12b) : n=6 ...k=768 x 12 block 相当
を試し学習できた

ただ n が大きくなるにしたがって学習が難しくなる様
強さは a)~d)で大きくは変わらなかった
厳密には b), d) は少しnpsが低下しその分弱かった（ a と c は nps はほぼ同じ ）
n が小さい場合 nvidiaのハードの制御機構に起因して 2 の冪乗以外で並列化の効率が落ちる

＊入力データ量が少ないと、大きすぎる n は逆効果になりそう
dlshogi の入力データ量の場合 n=6 までは問題ない
＊GPUの種類により 最も効率よく計算できる k の値は変わると思われる(後述)

AstraNET(cont.)

- 今回使用する予定の model (V318)は
(9,k=128x4,6b)+(9,k=128x3,4b)+(9,k=128x2,24b)
と多段構造にした
- 入力側のデータを多く(k=512相当)し、計算量を増やさずに層数を増やすため
- 入力側は各ResNET毎に異なるデータを受け取り、総計で34blocksを持つ

SM数(左)と並列実行に好適なkernel数の関係

- RTX5090…………..170　　k=170 (9x9の将棋には大きすぎる？)
- RTX4090…………..128　　k=128 (256 は大きすぎる？)
- RTX4080_16GB…76　　k=76 (or 152 ?)
- RTX4080_12GB…60　　k=60 or 120
- RTX3090..…………82　　k=82 (164 は大きすぎる？)
- RTX3080_10GB…68　　k=68 or 136

kernel 数がこれらの数より少し小さいのは問題ないが、少し大きい場合は計算の並列化への悪影響が大きいので避けるべき

類似例を見つけた...(2025/03/03)

- ShuffleNet, ResNeXt(Grouped Convolution) の考え方に近い
  - データをグループに分けて計算量を減らす
  - ShuffleNetと同じ理由でデータを混ぜる
  - 実装方法＆シャッフルの仕方は異なる
  - kernel数をGPUのSM数に限定して高速化するのはオリジナル

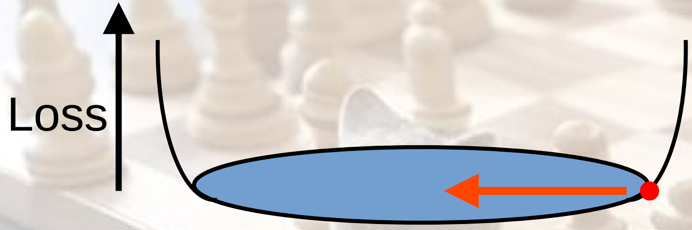
- 2023/09頃から採用して、wcsc34 もこの構造を採用した

<u>**追加工**</u> --学習終了後の派生modelの生成--

1）勾配が緩やかになった時点で学習を終了する

2）そのため学習終了時には、Loss値の盆地の端にいる

3）盆地の中央方向に移動したい



Loss値の盆地

・未知の局面でよりロバストになる事を期待

・盆地の底にはまだ緩い勾配が残っている可能性があるが、通常の学習では効率が悪い

・到達場所が極所解であれば、近傍により良い場所がある場合がある

・算術演算により良いmodelの周辺のmodelを作成し、ベンチマークでbetterな物であるか確認する

・より良い物が見つからなくなるまで繰り返す

Modelの追加工

- ２つ(複数)のmodelの加重平均により派生modelを作成する
  より良いmodelに成る可能性がある
  - ノイズ成分を減少させる効果
  - 学習終了時には「盆地」の端にいる可能性が高い
    中央方向に移動する可能性がある
- 学習済modelを定数倍する
  - 1.001倍~1.004倍が良い場合が多かった
    (結構センシティブ：FP8等は将棋AIでは無理？)
- model に乱数を付加する事により派生model を生成する

これらを実施する事により、学習終了時のmodelからR+50程度の向上が期待できる

Modelの追加工(cont.)

- 乱数は正規分布よりも一様分布がよかった
- 乱数の与え方は２通り試した
    - 読み出し単位で同じ乱数を使う
    - すべての数値に異なる乱数を使う
- 一様乱数の大きさのレンジは +/- 1.001 ~ +/- 1.004
  この数値を掛ける(学習の精度が良い場合は小さな数字が良い)
- 学習が十分に上手くいっている場合は、追加工による伸びしろは少ない

スレッド生成部の並列化による高速化

- スレッド生成部の並列化を採用した（wcsc34 でも採用した）
- 計算ユニットが多いGPUでは、計算ユニットがデータ転送待ちにならないような工夫が必要
- rtx3090/4090と今回検討したmodelの組み合わせでは効果あり
  - rtx4090 では30%高速化するmodelもあった
- 推論時のGPU使用率が90%以上にならない場合は効果がある可能性あり
- 効果の程度はHWのスペックに依存する（MBのPCIEの世代も重要）
- 効果がない場合もある

スレッド生成部の並列化(cont.)

- dlshogi では、GPUが１枚の場合は次の様な設定でスレッド生成部を並列化できる
- 効果は使用する GPU,MB と model のサイズにより異なる
- 特にGPU-MB間のバスの転送速度が重要(rtx3090, rtx4090でPCI-E4.0以上のMB必要)
- 概ねnpsの向上が10%以上であれば試す価値がある　それ以下では弱くなる可能性が大(*1)

DfPn_Min_Search_Millisecs	300	DNN_Batch_Size	128
DNN_Batch_Size2	128	DNN_Batch_Size3	0
DNN_Batch_Size4	0	DNN_Model	V324-064-3-d_971548.or
DNN_Model2	V324-064-3-d_971548.or	DNN_Model3	V324-064-3-d_971548.or
DNN_Model4	V324-064-3-d_971548.or	Draw_Ply	0
UCT_Threads	2	UCT_Threads2	2
UCT_Threads3	0	UCT_Threads4	0

図：設定の例 (128x2x2)

GPUが１枚のPCでは GPUIDは0となる
同時に GPUID=1を使う設定にしても
　GPUID=0として扱われる
この時 スレッド生成部が並列化される

*1: 本手法は標準的な探索に比べ探索の初期に探索ツリーの更新が遅れる
　そのため以降の探索に悪影響を及ぼす局面が存在するためと思われる

UCB値計算式の変更

- 次の一手問題を検討した時に問題によっては 実行毎に探索結果・nps が大きく異なる場合がある（結果が２値的に分かれる）
- 探索初期の乱数の出方に依存して、探索の集中する枝が変わるのが原因ではないかと推測　＜＝これを対策したい
- 特定の条件下で、勝率項の重みが小さければ安定した探索ができるのではないか

- 一手１秒程度の探索で勝率の向上を確認
(より長時間、例えば一手10秒思考での効果は未確認)
- ２番目の候補の探索回数が増えていた

UCB値計算式の変更(cont.)

- ucb_value = q + c * u * rate; を変形し
  ルート：　　　ucb_value = f(q) + c * u * rate;
  ルート以外：　ucb_value = q + k * c * u * rate; 　k は定数
- f(q) は q の２次式、または３次式 ...比較した範囲では２次式が良かった
- 境界条件：f(0)=0, f(1)=1, 0 <= f’(1) < 1

- ７個の探索パラメータが先に調整されている事
- Temperatureの再調整が必要になるかもしれない(小さめが良くなる可能性あり)
- 探索パラメータは７つあり調整が困難である事に対し、パラメータが１～２になるので調整が容易
  （探索パラメータの最適化が理想的にできているのであれば、この変更は必要ないかもしれない）

まとめ

- 今回使用する予定の model (V318)は (9,k=128x4,6b)+(9,k=128x3,4b)+(9,k=128x2,24b) と、多段構造
- wcsc34 から R+120 程度 (@ rtx4090) 向上
  - 定跡有りで、floodgate基準で R4500位@rtx4090（推測）
- AstraNET、及び「追加工」の詳細を説明した
- スレッド生成部の並列化を説明した
- UCB値計算式の変更を説明した
- 手数と評価値によるEngine の切り替えを実装した（詳細略:ページが足りない）