1H-02

畳み込みニューラルネットワークにおける Cython を用いた高速化

Speedup Using Cython for Convolutional Neural Network

西村 太輔 †
Daisuke Nishimura
吉田 明正 †
Akimasa Yoshida

1 はじめに

機械学習に広く用いられている Python プログラムの高速化手法として，変数の静的型付けを行う Cython，JIT コンパイラライブラリである Numba や PyPy，GIL を解除して並列化を行う threading や multiprocessing 等が知られている．Cython を用いた機械学習高速化手法の一例として，Cython を用いた一部計算の高速化研究 [1]，CNN の一部を Cython で実装した方法 [2] が挙げられる．本稿では画像分類の教師あり学習を対象とし，全体を Cython 実装し静的型付けの特性を活かして高速化を実現する．また Cython には並列処理を容易に実装するために prange 特殊関数が用意されている．本研究では予備評価として Cython コンパイラによってコンパイルされた C プログラムに OpenMP[3] を組み込み，従来より高い並列性を引き出している．性能評価では，6 コアの CPU で提案手法のプログラムを実行したところ高い実行性能が達成され，提案手法の有効性が確認された．

2 Python プログラムの Cython による高速化

本稿で利用する Cython は変数の静的型付け，GIL の取り外しにより可能となる並列処理によって高速化を実現する．本章では Cython の仕組みについて述べる．

2.1 Cython の仕組み

Cython 言語は Python のスーパーセットであるため，Python インタープリタが Cython を直接インポート・実行することはできない．そのため Cython コンパイルパイプラインを用いてコンパイルする必要がある．Cython ソースコードがコンパイルされ Python モジュールとなるまで 2 ステージ存在する．まず第 1 ステージでは cython コンパイラにより Cython ソースコードを C もしくは C++ソースコードに変換する．その後第 2 ステージで標準 C，C++コンパイラにより，第 1 ステージで作成した C もしくは C++ソースコードを共有ライブラリにコンパイルし，拡張モジュールとして扱うことができるようになる．通常このコンパイルは，小さな Python スクリプトを実行することで行われる．第 1 ステージは Cython の cythonize コマンド，第 2 ステージは Python の distutils によりコンパイルされる．

2.2 変数の静的型付け

Python が遅い原因として，変数や属性に代入されたオブジェクトを読み込む度に，そのオブジェクトがどの型に属しているかを調べる必要のある，動的型付け言語であることが挙げられる．それに対し Cython は，numpy 配列やメモリビューに型情報・次元数を指定することで高速化を実現している．numpy 配列は numpy に対応しているがローカル変数としてしか使用できない．一方メモリビューは一部の numpy に対応してないが，モジュールレベルの変数に対応している．Cython を使用するにあたり，両者の特性を把握し適切に用いることが必要となる．

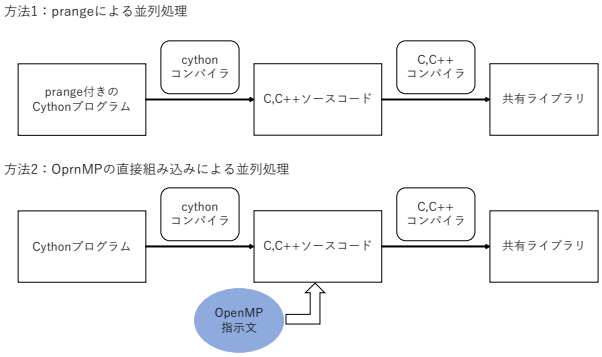


図 1 Cython の並列処理手法.

2.3 Cython を用いた並列処理

prange 特殊関数 [4] はループ並列処理を容易に実装できるが，注意点や制約もある．例えば prange は OpenMP API を用いて実装されているため，コンパイルする際-fopenmp コマンドを追加するよう Python スクリプトに変更を加える必要がある．またループ並列処理内では Python オブジェクトを扱えない．これは Python オブジェクトのメモリ管理のために必要な GIL を解除したことに起因する．そのためループ内では型付きメモリビューを用いる場合が多い．これは C レベルで動作するため，Python のオーバーヘッドを最小限に抑えることができ極めて高速である．

3 Cython と OpenMP を用いた並列処理

本章では Cython における高速化手法と性能評価プログラムの詳細について述べる．

3.1 並列コードの生成方法

本節では prange を用いた並列手法 (図 1 の方法 1) に対し，cython コンパイラにより出力された C コードに直接並列処理を組み込む手法 (図 1 の方法 2) を提案する．prange 特殊関数を使用した際 C コードに多くの例外処理が追加されるため，本手法により並列化による恩恵を最大限受けられるようになると推測する．注意点として prange を用いる場合と同様に Python オブジェクトを扱えないため，並列適応箇所にはメモリビューもしくは numpy 配列を使う必要がある．

†明治大学大学院 先端数理科学研究科 ネットワークデザイン専攻
Graduate School of Advanced Mathematical Sciences, Meiji University

表 1　作成した CNN の構成.

層番号	層の種類	操作	出力サイズ	出力枚数
1	畳み込み層	3 × 3,32 フィルター	224 × 224	32
2	畳み込み層	3 × 3,32 フィルター	224 × 224	32
-	プーリング層	max プーリング	112 × 112	32
3	畳み込み層	3 × 3,64 フィルター	112 × 112	64
4	畳み込み層	3 × 3,64 フィルター	112 × 112	64
-	プーリング層	max プーリング	56 × 56	64
5	畳み込み層	3 × 3,128 フィルター	56 × 56	128
6	畳み込み層	3 × 3,128 フィルター	56 × 56	128
-	プーリング層	max プーリング	28 × 28	128
7	全結合層	1024 ユニット	1024	-
-	ドロップアウト層	確率 50%	1024	-
8	全結合層	1024 ユニット	1024	-
-	ドロップアウト層	確率 50%	1024	-
9	全結合層	6 ユニット	6	-

表 2　使用したデータセット.

	トレーニング データセット	テスト データセット
airplanes	400	400
Moterbikes	400	400
Faceseasy	220	215
watch	120	120
Leopards	100	100
bonsai	65	65
合計	1305	1300

3.2　数値積分プログラムにおける並列処理

円周率を求める台形積分プログラムを用いて，prange を用いた従来の並列処理手法と提案手法の性能評価を行う．数値積分の分割数 N は 200,000,000 に設定した．

3.3　CNN プログラムにおける Cython 実装による高速化

画像分類畳み込みニューラルネットワーク [5] を用いて，Cython の静的型付けによる高速化について性能評価を行う．本稿で作成した 9 層 CNN の詳細は表 1 の通りであり，フレームワークを用いずに，Python と Cython により実装した [6]．データセットには Caltech101 に含まれる 6 カテゴリをデータ拡張し約 2600 枚用いた．拡張後のトレーニングデータセット，テストデータセットのサンプル数を表 2 に示す．

4　Intel Corei7-8700 上での性能評価

本性能評価で使用したマシンは，プロセッサとして Intel Corei7-8700(3.2GHz 6 コア) を搭載しており，メモリ 48GB，OS は Ubuntu 16.04.6LTS で構成される．使用した Python はバージョン 3.7.3，Cython は 0.29.12 である．

4.1　数値積分プログラムを用いた性能評価

Cython の並列処理の性能評価を行うにあたり，参考として C 言語プログラムも合わせた 3 種類の比較を行った．性能評価結果は図 2 の通りとなり，1 スレッド時における prange を用いた計算結果が 4008[ms] であったのに対し提案手法は 421[ms] であり，9.5 倍の計算速度向上であった．また 6 スレッド時における prange を用いた計算結果は 720[ms] であったのに対し提案手法は 75[ms] であり，9.6 倍の計算速度向上であった．これにより，提案手法は並列性を保ちつつ C 言語と同等の処理速度が得らており，従来の prange 特殊関数と比べて高い実効性を達成できることが確認された．

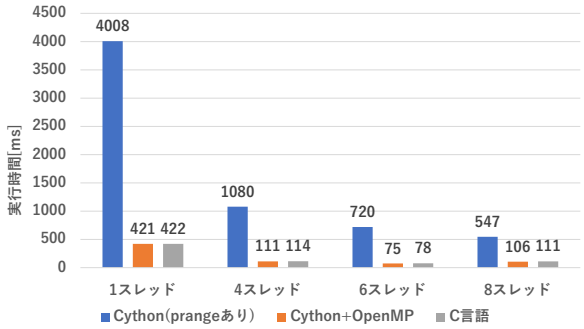
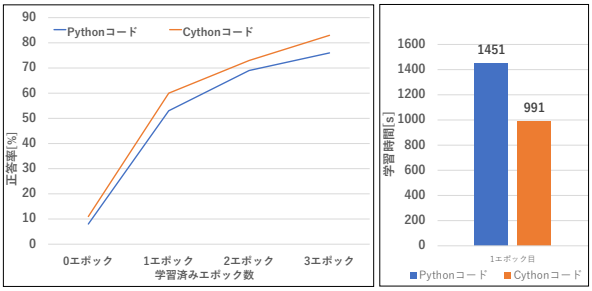


図 2　数値積分プログラムにおける並列処理時間.



(a)テストデータの正答率遷移　(b)1エポック目の学習時間

図 3　CNN プログラムにおける性能評価.

4.2　CNN プログラムにおける性能評価

構築した CNN を用いて Python コードと Cython コードの性能評価を行った．なおバッチサイズは 64 で毎エポック 1280 データの学習，最適化アルゴリズムには AdaGrad，学習係数は 0.01 とした．学習済みエポック数とテストデータの正答率遷移を示した図 3(a) から，Cython 実装による分類精度低下は確認されなかった．また図 3(b) より Python コードの 1 エポック目の学習時間は 1451[s]，Cython コードの 1 エポック目の学習時間は 991[s] であり，1.46 倍の高速化の実現に成功した．これにより，提案手法である Cython 実装による CNN の高速化は高い実効性能を達成できることが確認された．

5　おわりに

本稿では，機械学習で広く用いられている Python プログラムの高速化を達成するため，Cython による実装，および Cython における並列処理の効率性をより高める手法を提案した性能評価の結果，数値積分プログラムによる Cython を用いた並列処理の性能評価は，従来手法の 9.6 倍，画像分類 CNN による性能評価では，Python プログラムの 1.46 倍の速度向上が得られた．性能評価の結果，提案手法の有効性が確認された．

参考文献

[1] 関谷 翠，大沢 和樹，長沼 大樹，横田 理央．低ランク近似を用いた深層学習の行列積の高速化，研究報告ハイパフォーマンスコンピューティング (HPC)，Vol.2017-HPC-158,No.24,pp.1-7,2017.

[2] 伊藤 有人，伊藤 雅．少資源環境下における RocAlphaGo の改良とその棋力検証，情報処理学会第 80 回全国大会講演論文集,Vol.2018,No.1,pp.53-54,2018.

[3] OpenMP．https://www.openmp.org/, 2018.

[4] Kurt W．Smith，中田秀基 (監訳)，長尾高弘 (訳)．Cython C との融合による Python の高速化．オライリージャパン，2015.

[5] 藤田一弥，高原歩．実装 ディープラーニング．オーム社，2016.

[6] 我妻幸長．はじめてのディープラーニング．SB クリエイティブ，2018.