1H-01

強化学習のPythonプログラムにおけるNumbaを用いた高速化

Speedup Using Numba for Python Programs of Reinforcement Learning

中村 優志† 吉田 明正†
Yuji Nakamura Akimasa Yoshida

1 はじめに

Python言語は機械学習分野においてライブラリが充実しており，広く用いられている．しかしながらPython言語はインタプリタ言語であるため，C/C++言語などのプログラムに比べて実行時間が長いことが指摘されている．これらの問題を解決するためにJITコンパイラ技術を利用したNumbaフレームワークが開発されている[1][2].

本稿ではPython言語で実装された強化学習プログラムにNumbaフレームワークを適用し高速化を実現する．Numbaにはループ並列処理を行うためにprangeが用意されており，本手法では強化学習のエージェントレベルの並列化にprangeを適用する．Intel Corei7上での性能評価の結果，提案手法の有効性が確認された．

2 PythonのNumbaフレームワーク

本稿で利用するNumbaは，LLVM[1]コンパイラライブラリを用いて実行時にPython関数を最適化された機械語に変換し，高速化するフレームワークである．Numbaでコンパイルされた PythonはCまたはFOTRANに近い速度が出る．

2.1 Numbaの利用法

Numbaを Pythonで使用するには Numbaデコレーターを関数の前につける必要がある．Numbaデコレーターをつけた関数は実行時に機械語にコンパイルされる．

2.2 Numbaの実行モード

Numbaには Objectモードと No-Pythonモードの2種類の実行方法がある．Objectモードはコンパイルに失敗したときに実行されるモードで，コンパイルされない従来通りの方法である．コンパイルされないので高速化はされない．No-Pythonモードは実行時のコンパイルされ高速化される．No-Pythonモードを強制するには@jit(nopython=True) または@njitのデコレーターをつける．使用例を (図1) に示す．しかし，No-Pythonモードでコンパイルに失敗場合 Objectモードにはならずエラーとなる．

```
1 from numba import jit
2 @jit(nopython=True)
3 def test():
4         ...
```

図1 Numbaの No-Pythonモードのプログラム．

†明治大学大学院先端数理科学研究科
Graduate School of Advanced Mathematical Sciences, Meiji University

2.3 Numbaによる並列処理

Numbaには並列オプション[3]の prangeがあり，これを適用すると関数の一部で自動的に並列化が実行される．現時点でこの機能はCPUのみで機能する．使う場合は@jit(nopython=True,parallel=True) と書き，forループの rangeを prangeにする．使用例を (図2) に示す．

```
1 from numba import jit,prange
2 @jit(nopython=True,parallel=True)
3 def test():
4     for i in prange ():
5         ...
```

図2 Numbaによる並列プログラム．

2.4 Numbaにおける Numpyの制約

現在 Numbaは Pythonのすべての機能やライブラリををサポートしていない．それゆえ，前節で説明したデコレータを使用するだけでは No-Pythonモードで実行する場合エラーが出る可能性がある．よって事前に Numbaがサポートしていない部分を書き直す必要がある．

Numpyは Pythonにおいて数値計算を効率的に行うための拡張モジュールである．Pythonにはない型付きの多次元配列や，それらを操作する高水準の数学関数ライブラリを使用することが可能となる．但し，Numba上で Numpyを使う場合は注意が必要である．例えば，Numba上で Numpy配列を使用する場合，2次元配列等を1次元配列にする必要がある．

3 強化学習における Numbaによる高速化

本章では，強化学習アルゴリズムとして Q学習を取り上げ，強化学習への Numbaの適用方法について述べる．

3.1 Q学習のアルゴリズム

強化学習とは，ある環境内におけるエージェントが現在の状態を観測し行動を決定する機械学習の1つである．エージェントは行動を選択することで環境から報酬を得る．この報酬が最も多く得られるような方策を学習する．

強化学習には様々なアルゴリズム[4]があるが，本稿では Q学習を用いた．Q学習とは報酬経験だけを頼りにエピソードの終了まで待たずに状態価値観数や行動価値観数を更新する手法の1つである．Q学習では各行動の有効性を示す Q値という値を持たせ，エピソードが行動するたびに Q値を更新する．Q値は次式によって更新される．

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha\ reward(s,a) + \gamma\ maxQ(s',a')$$

ここで，sは状態，aは行動，Q(s,a) は Q値，αは学数率，γは時間割引率，reward(s,a) は状態 sにおいて行

動 a を実行した際に得られる報酬，である．また s' は状態 s において行動 a を実行した後の遷移した状態であり，max Q(s',a') は s' における Q 値の最大値である．これを繰り返すことで学習が進行する．

3.2 Q 学習における Numba の適用

Q 学習において最も時間のかかる部分は学習を繰り返している部分である．この部分を高速化することで Q 学習全体が高速化されるのは自明である．よってこの部分に Numba を適用し最適化することで Q 学習の高速化が達成される．適用方法は 2 章で述べた図 1 の Numba 逐次コードである．本稿で実装した Q 学習のプログラムは，学習を繰り返すループと 1 回の学習で試行を繰り返すループの 2 重ループで構成されている．試行を繰り返すループにはデータ依存があるため並列処理をする事は出来ない．しかし，エージェントレベルでは並列性があるため並列処理を行うことが可能である．よって Numba 並列を適用するのは学習を繰り返すループとなる．適用方法は 2 章で述べた図 2 の Numba 並列コードである．

4 Numba フレームワークによる高速化の性能評価

本性能評価では，まず 1000*1000 の行列積を求めるプログラムで性能評価を行った．次に Q 学習による迷路問題 [5] のプログラムを用いて性能評価を行った．本稿では Python 配列コードを用いたプログラムと，Numpy 配列コードを用いたプログラムの 2 種類を用いて行った．2 章で説明した Numpy の Numba 上での制約を考慮し配列を用いる際は，Python 配列コードのプログラムと Numpy 配列コードのプログラム両方で配列の 1 次元化を行っている．

4.1 性能評価環境

本性能評価では，表 1 に示すマシンを使用する．

表 1　性能評価に用いるマシン．

CPU コア	Intel Core i7-8750H 2.20GHz 6 コア
メモリ	16GB
OS	Windows10 home
Python	3.7.7
Numpy	1.18.4
Numba	0.51.2

4.2 行列積における Numba の性能評価

本節では，行列積を用いて性能評価を行う．行列積を求める方法は，Numba 並列を用いるために for 文によるループを用いて実装した．Python 配列コードと Numpy 配列コードを用いた，Numba なし，Numba 逐次，Numba 並列でそれぞれ実行し速度を比較する．

結果は表 2 のようになった．Python 配列コードの Numba なしを基準に，Numpy 配列コードの Numba なしで 0.18 倍，Python 配列コードの Numba 逐次で 4.32 倍，Python 配列コードの Numba 並列で 18.70 倍，Numpy 配列コードの Numba 逐次で 22.88 倍，Numpy 配列コードの Numba 並列で 78.31 倍の高速化が達成された．本稿の性能評価の結果，Python 配列コードの Numba なしより Numpy 配列コードの Numba なしが低速なのは今回実装したプログラムでは Numpy に用意されている行列積の関数を用いなかったためだと考えられる．

表 2　行列積による性能評価．

	Numba なし	Numba 逐次	Numba 並列
Python 配列コード	110.788[s] (1.00 倍)	25.648[s] (4.32 倍)	5.925[s] (18.70 倍)
Numpy 配列コード	617.310[s] (0.18 倍)	4.843[s] (22.88 倍)	1.415[s] (78.31 倍)

4.3 迷路問題の Q 学習における Numba の性能評価

本節では，Q 学習による迷路問題を用いて性能評価を行う．今回用いた迷路問題は 39*39 の迷路である．ここで，エージェント数＝1 万，最大ステップ数＝1 万として実行時間を測定する．

結果は表 3 のようになった．Python 配列コードの Numba なしを基準に，Numpy 配列コードの Numba なしで 1.11 倍，Python 配列コードの Numba 逐次で 11.70 倍，Python 配列コードの Numba 並列で 45.08 倍，Numpy 配列コードの Numba 逐次で 75.62 倍，Numpy 配列コードの Numba 並列で 56.38 倍の高速化が達成された．

表 3　迷路問題による性能評価．

	Numba なし	Numba 逐次	Numba 並列
Python 配列コード	151.510[s] (1.00 倍)	12.953[s] (11.70 倍)	3.361[s] (45.08 倍)
Numpy 配列コード	137.049[s] (1.11 倍)	2.004[s] (75.62 倍)	2.687[s] (56.38 倍)

5 おわりに

本稿では，Python プログラムの Numba を用いた高速化を提案した．Intel Corei7-8750H 2.20GHz 6 コア上での性能評価の結果，行列積では Python に比べて Numba を適用したものは最大で 78 倍の高速化を達成した．迷路問題の Q 学習では Python に比べて Numba を適用したものは最大で 75 倍の高速化を達成した．以上の結果から，提案手法の有効性が確認された．

参考文献

[1] Siu Kwan Lam, Antoine Pitrou, Stanley Seibert. Numba: a LLVM-based Python JIT compiler, LLVM '15: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, 2015.

[2] Gilbert Gutabaga Hungilo, Gahizi Emmanuel, Pranowo. Performance comparison in simulation of Mandelbrot set fractals using Numba, AIP Conference Proceedings 2217, 2020.

[3] かくあき． Python 科学技術計算入門， 翔泳社，2020.

[4] 曽我部東馬．強化学習アルゴリズム入門，オーム社，2019.

[5] 伊藤一之．実装強化学習，オーム社，2018.