分散強化学習におけるデータ保存ライブラリの設計と実装に関する研究

1255104　仮谷 拡晃　【 高度プログラミング研究室 】

Design and Implementation of a Data Storage Library for Distributed Reinforcement Learning

1255104　KARIYA, Hiroaki　【 High-Level Programming Lab. 】

1　はじめに

強化学習は環境の状態を観測し意思決定を行うエージェントが，環境に対して行動を選択し，その行動の結果得られる報酬が最大となるような方策を学習する手法である [1]．強化学習の手法として，経験の生成や環境の探索を行う計算資源のスケールアウトにより分散並列処理する分散強化学習があり，A3C [2] や Ape-X DQN [3] などが考案されている．分散強化学習では複数のプロセスが学習データを非同期かつ並列に生成するが，生成されたデータは検証や解析などの追試研究のために保存されていることが望ましい．

そこで，本研究では強化学習向けの分散並列処理フレームワークである Ray [4] を利用して Python 上で動作するデータ保存ライブラリの設計と実装を行う．非同期に生成される学習データを保存するためのライブラリのモデルを提案する．実装したライブラリの性能を評価するために，OpenAI Gym が提供する環境において学習を行う Ape-X DQN に対してデータ保存処理を追加した時の実行時間を調べる．

2　Ray

Ray [4] は Python で分散処理を含んだ強化学習アプリケーションを実装するためのフレームワークである．タスクの並列処理とアクターベースの処理を統一されたインタフェースで記述することにより既存のコードから大きく変更することなく複数サーバから成るクラスタで動作するアプリケーションを実装できる．

Ray ではリモート関数を呼び出した際にタスクの結果を示す ObjectRef オブジェクトを返す．ObjectRef オブジェクトはタスクの完了を待たずに他のリモート関数へと渡すことが可能である．これにより，データの依存関係を保ったまま複数のタスクを非同期かつ並列に処理できる．

本研究では Ray の持つ特性を利用してライブラリの実装を行う．

3　提案モデル

本研究で提案するデータ保存モデルを図 1 に示す．アクター $\{Actor_1, Actor_2, \ldots, Actor_n\}$ が生成する経験

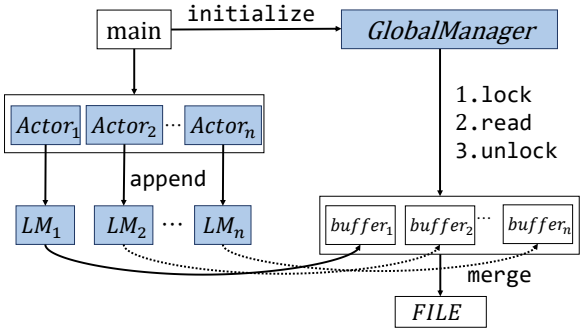

図 1　モデル概要

をそれぞれバッファ $\{buffer_1, buffer_2, \ldots, buffer_n\}$ へと保存する LocalManager とバッファに保存された経験を統合し時系列順にソートして *FILE* に書き込む GlobalManager で構成される．LocalManager は経験の生成を行うアクターごとに割り当てられる．データ保存を追加することによる性能の低下を緩和するために LocalManager と GlobalManager は Ray のアクターによりリモート化し並列にファイルへの書き込み処理を行う．

LocalManager と GlobalManager の起動，バッファへのデータ追加，*FILE* への出力及び読み込みは以下の関数を用いて行う．

`initialize`: GlobalManager (起動していない場合は Ray クラスタ) の起動，初期化を行う．

`activate_manager`: LocalManager を起動し，オブジェクト ID と実体を GlobalManager へ渡す．

`append`: アクターが生成した経験を LocalManager が持つバッファに対して追加する．中間データ及び最終的な出力である *FILE* のサイズを小さくするために zlib により圧縮，pickle によりシリアライズ化を行ってからデータの追加を行う．

`merge`: この関数を呼び出すと GlobalManager が $\{buffer_1, buffer_2, \ldots, buffer_n\}$ 内のデータを取得し，時系列順にソートして *FILE* へ書き込む．

`finitialize`: 実行中に生成された中間ファイルの削除，未処理の学習データの統合を行う．

4 実験

実装したライブラリの性能を評価するために Ape-X DQN で OpenAI Gym が提供している環境である CartPole-v1 と BreakoutDeterministic-v4 の学習に対してデータ保存処理を追加した際の実行時間を計測した．実験環境は同一ネットワーク内に配置された同じ構成の計算機を 2 台用いて Ray クラスタを構築し，100 ステップを 1 ロールアウトとしてアクター数 20 で学習を行った．学習終了時に経験の保存を行う方法 (これを一括出力と呼ぶ) とネットワーク更新時に保存を行う方法 (これを都度出力と呼ぶ) の 2 通り行い実行時間を比較する．

表 1 と表 2 は各環境についてロールアウトの回数を 2000 から 10000 でそれぞれ 10 回実行した際の実行時間の中央値である．CartPole 環境において，学習終了時に保存する一括出力では最小で 1.9%，最大で 7.4% ，ネットワーク更新時に保存する都度出力では 最小で 11.2%，最大で 22.7% の性能低下となった．Breakout 環境において，一括出力では 最小で 16.5%，最大で 21.7%，都度出力では最小で 15.8%，最大で 19.6% と約 20%前後の性能低下となった．また，ロールアウトの回数とデータを保存するタイミングの違いによる性能低下率にほとんど差は確認できなかった．

5 考察

実験の結果から CartPole 環境の一括出力では性能低下を抑制しつつ学習過程で生成されたデータを保存することができた．一方，都度出力では 11.2 ～ 22.7% 程実行速度が低下した．都度出力ではネットワーク更新時に経験を時系列順にソートしてファイルへ書き込む `merge` 関数を実行していることから，そのオーバーヘッドが 1 度のネットワーク更新に対して大きかったと考えられる．

表 1　CartPole 環境における実行時間 [sec]

rollout	データ保存なし	一括出力	都度出力
2000	15.21	15.71	16.92
4000	25.99	27.21	30.68
6000	38.36	40.10	44.60
8000	48.18	51.76	59.12
10000	61.98	63.16	71.81

表 2　Breakout 環境における実行時間 [sec]

rollout	データ保存なし	一括出力	都度出力
2000	92.46	112.53	110.11
4000	185.03	215.58	214.24
6000	268.95	321.77	321.13
8000	357.00	427.45	421.66
10000	444.50	530.07	531.69

Breakout 環境では一括出力，都度出力ともに約 20% の性能低下が確認された．性能低下の原因として 1 ステップあたりの経験のデータサイズの大きさが考えられる．このライブラリで保存対象としている 1 ステップあたりの学習データは時刻 t における状態，行動，報酬，行動後の状態，終了判定

$$S = \{s_t, a_t, r_t, s_{t+1}, d_t\}$$

の 5 つの値で表される．CartPole 環境における s_t, s_{t+1} は 1×4 の ndarray でるが，Breakout 環境では $210 \times 160 \times 3$ の ndarray で表現された画像となっており，一度に追加するデータサイズが大きくなっている．リモート関数に対してオブジェクトを入力として与えると Ray が内部に持つ Global Control Store (GCS) と呼ばれる共有メモリに対するコピーが発生する [4]．そのため，リモート関数 `append` を実行する際に発生する GCS へのコピーがボトルネックとなり性能低下を招いたと考えられる．

6 まとめ

本研究では，分散強化学習の学習過程で生成されるデータを効率よく保存するための Ray を使用して python 上で動作するライブラリのモデルの提案と開発を行い，Ape-X DQN の学習に対してデータ保存処理を追加した際の実行時間を計測した．

実験結果から経験のデータサイズが小さい環境では比較的効率良くデータ保存を行えるが，状態空間のデータサイズが大きい環境では GCS に対するコピーにより性能低下を招いているため，リモート関数実行時のデータ処理を改善する必要がある．

参考文献

[1] R. S. Sutton and A. G. Barto, “Reinforcement learning: An introduction”, The MIT Press (1998)，三上 貞芳, 皆川 雅章 共訳, “強化学習”, 森北出版 (2000).

[2] V. Mnih et al., “Asynchronous methods for deep reinforcement learning”, International Conference on Machine Learning 2016 (2016).

[3] D. Horgan et al., “Distributed prioritized experience replay”, International Conference on Learning Representations 2018 (2018).

[4] P. Moritz et al., “Ray: A distributed framework for emerging AI applications”, 13th USENIX Symposium on Operating Systems Design and Implementation, pp. 561–577 (2018).