2P-03

DCGANにおけるPyTorch Distributed Data Parallelライブラリを用いた並列処理

Parallelization of DCGAN Using PyTorch Distributed Data Parallel Library

冨田 雄大† 吉田 明正†
Yuta Tomita Akimasa Yoshida

1 はじめに

近年注目されている深層学習技術の1つにGANがある．GANはデータの特徴を学習することによって，実在しないデータを生成する．また，隠れ層に畳み込みを用いたDCGAN[1]では，より鮮明な画像の生成が可能になっているが，GANに比べて学習時間が長いことが指摘されている．

最近では複数GPUを搭載した並列システム[2]が普及しており，PyTorchのDistributedDataParallelモジュール[3]の利用により，複数GPUを容易に扱うことができる．従来，GPUを用いて処理時間の高速化を図った新しいGANアルゴリズム[4]は提案されているが，PyTorch記述されたDCGANに対しての速度向上は提案されていない．

本手法では2つのネットワークを持つDCGANに対して，DistributedDataParallelモジュールを適用して高速化を実現する．NVIDIA Quadro RTX6000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 DCGANのアルゴリズム

本章では敵対的生成ネットワーク並びにDeep Convolutional Generative Adversarial Networks(DCGAN)のアルゴリズムについて述べる．

2.1 敵対的生成ネットワーク

敵対的生成ネットワークとは通称，Generative Adversarial Networks（GAN）と呼ばれる．生成器と識別器の二つのネットワークを持つ．このネットワークが競い合うようにして，学習が進んでいく．生成器では，与えられたランダムなノイズを基に本物らしい画像を生成する．識別器では，本物画像または偽物画像を入力し真偽を判定する．識別器による判定の誤差を基にして，識別器と生成器のパラメータを調整する．生成器は識別器を誤認させるように，識別器は生成器が作り出した画像を正しく判別できるように競い合うようにして学習が進んでいく．

2.2 DCGAN

Deep Convolutional GAN(DCGAN)は各ネットワークの中間層に畳み込み層を用いている．従来，中間層には全結合層が用いられてきたが，生成される画像が鮮明ではないという問題点があった．畳み込み層を用いることで，学習を高度化しこの問題を解決した．一方，層が多層化したことなどによって，トレーニング時間の増加が見られる．本稿では，複数GPUを用いることによって，トレーニングの高速化を図る．

†明治大学総合数理学部ネットワークデザイン学科
Department of Network Design, School of Interdisciplinary Mathematical Sciences, Meiji University

3 PyTorchのデータ並列モジュール

PyTorchは2016年にFacebookから発表された機械学習ライブラリである．柔軟にネットワークを記述できることから，多く利用されている．PyTorchにはデータ並列学習をするためのモジュールが2つ用意されている．従来は，CNN等の1つのネットワークを持つものに利用されてきた．本稿では，2つのネットワークをもつDCGANアルゴリズムにPyTorchのデータ並列モジュールを適用する．

3.1 DataParallelモジュール

PyTorchデータ並列モジュールの1つが，DataParallelモジュールである．これは，シングルプロセス・マルチスレッド・シングルマシンで動作する．モデルをカプセル化して実行するだけなので，プログラムの書き換えが容易である．反復ごとにモデルを複製する．入力を分散して，出力で集約を行う．

3.2 DistributedDataParallelモジュール

PyTorchデータ並列モジュールのもう1つが，DistributedDataParallelである．これは，マルチプロセス・シングルマシン・マルチマシンで動作する．プロセスによって制御される．DataParallelモジュールに比べて，プログラムの書き換えが多く発生する．勾配のみがGPU間で伝播するため，ネットワーク間の通信が少ない．

4 PyTorch実装によるDCGANの並列処理

本稿では，DCGANをPyTorchを用いて記述した．PyTorchを用いることによって，容易にGPUを扱うことができ，プログラム全体の高速化が見込まれる．それでもなお処理時間の多くかかるDCGANアルゴリズムに対し，PyTorchが提供している2つのGPU並列モジュールを用いてさらなる高速化を図る．

4.1 DataParallelによる並列化

DCGANのプログラムをPyTorch DataParallelモジュールを用いて，2GPUデータ並列による高速化を図る．DCGANのGenerator(生成器)とDiscriminator(識別器)のネットワークをDataParallelモジュールにのせている．

実行の流れは，メインGPUがデータローダーを保持し，バッチの分割を行う．分割したバッチをそれぞれのGPUに転送する．加えて，各GPUにパラメータを共有する．それぞれのGPUでフォワードパスを行う．出力されたアウトプットはメインGPUに送られて，損失値を計算する．損失値をそれぞれのGPUに送信し，各GPUで勾配計算を行う．各GPUからメインGPUに勾配値を送り，メインGPUでパラメータを更新する．これを繰り返し行う．

表 1　性能評価に用いる並列システム.

CPU	Intel(R) Xeon(R) W-2265 CPU @ 3.50GHz
GPU	NVIDIA Quadro RTX6000 × 2
メモリ	125GiB
OS	Ubuntu 18.04.4 LTS
Python	3.9.7
Cudatoolkit	10.2.89
Pytorch	1.10.0

表 2　実行時間一覧.

実行環境	実行時間 [s]
逐次	6503
1GPU	238
DataParallel-2GPU	238
DistributedDataParallel-2GPU	145

4.2　DistributedDataParallel による並列化

DCGAN のプログラムを PyTorch DistributedDataParallel モジュールを用いて, 2GPU データ並列による高速化を図る. DataParalle モジュールの時と同様, Generator(生成器) と Discriminator(識別器) のネットワークを DistributedDataParallel モジュールにのせる.

データローダーは事前に分割され, 各 GPU が持つようにする. パラメータの共有は初回のみ実行される. フォワードパス, 損失値の計算, 勾配計算は各 GPU で行われる. そのため, この間に通信は発生しない. 各 GPU 間で得られた勾配をメイン GPU で集約し, 均一化した勾配を各 GPU に送り, パラメータを更新する. これを繰り返し行っていく.

5　マルチ GPU 上での性能評価

本稿では MNIST の手書き数字画像 6 万枚をトレーニング画像として, 各エポックごとの処理時間, トレーニング全体の処理時間を逐次実行,1GPU 実行, DataParallel による 2GPU 実行, DistributedDataParallel による 2GPU 実行に分けて評価を行う.

性能評価に用いる並列システムの構成は, 表 1 に示す通りである.

5.1　DataParallel モジュールを用いた性能評価

DataParallel モジュールを用いた性能評価 PyTorch で記述した DCGAN[5] に DataParallel モジュールを用いて, 2GPU 実行による性能評価を行った. DataParallel モジュールを用いた実行時間は表 2 に示した通り 238[s] であった. 逐次実行時では 6503[s] なのに対し, 図 1 に示した通り 27.2 倍の高速化が確認 された. 一方, 1GPU 実行時と比較すると速度向上は得 られなかった. これは, DataParallel モジュールが反復ごとにモデルを複製するために, オーバヘッドが増加したことが原因と考えられる.

5.2　DistributedataParallel モジュールを用いた性能評価

DistributedDataParallel モジュールを用いた性 能評価 PyTorch で記述した DCGAN[5] に対し, DistributedDataParallel モジュールを用いて 2GPU 実行による性能評価を行った. DistributedDataParallel モジュールを用いた実行時間は表 2 に示した通り, 145[s] であった. 逐次実行時では 6503[s] なのに対し, 図 1 に示した通り, 44.8 倍の速度向上, 1GPU 実行時では 238[s] に対し 1.64 倍

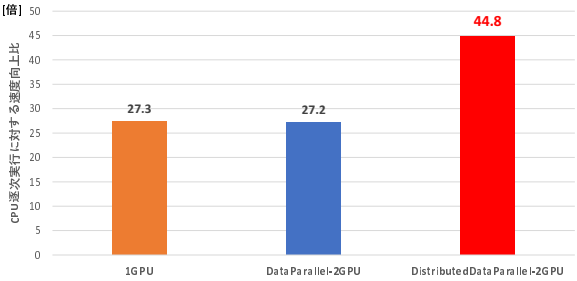

図 1　NVIDIA Quadro RTX6000 上での性能評価.

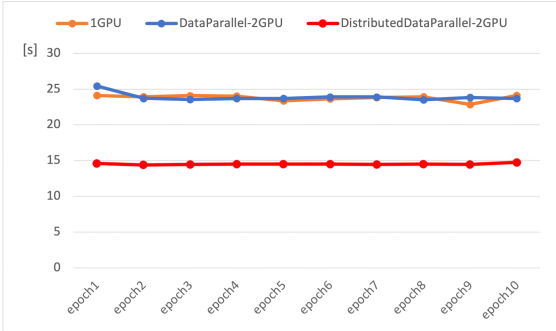

図 2　エポック別処理時間比較.

の速度向上であった. 図 2 に示したエポックごとの処理時間でも, 各エポックごとに大きな誤差は見られなかった. これにより, 提案手法では高い実効性を達成できることが確認された.

6　おわりに

本稿では, NVIDIA Quadro RTX6000 上で DCGAN に対し PyTorch DistributedDataParallel モジュールを利用したデータ並列による高速化を提案した. MNIST の手書き数字画像を用いた DCGAN に対して DistributedDataParallel を用いた実装を行ない, 実行速度の計測を行なった.

上記の実装で CPU 逐次実行時と比べて, 2GPU 実行時に最大 44.8 倍の速度向上が得られた. また, 1GPU 実行時と比べて, 2GPU 実行時に最大 1.64 倍の速度向上が得られた. これらの結果から, 提案手法の有効性が確認された.

参考文献

[1] Alec Radford, Luke Metz, Soumith Chintala.
Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks, 2016.

[2] 山田泰永.
ディープラーニング向け GPU 環境選択の実務, 2018.

[3] Shen, Yanli, Rohan, et al.
Pytorch Distributed : Experiences on Accelerating Data Parallel Training, 2020.

[4] Haibo Chen, Tao Jia, Jing Tang..
A research on generative adversarial network algorithm based on GPU parallel acceleration, 2019.

[5] 毛利拓也, 大郷友海, 嶋田宏樹, 大政孝充, むぎたろう, 寅蔵, もちまる.
PyTorch による画像生成/画像返還のための GAN ディープラーニング実装ハンドブック秀和システム, 2021.