

Mesh TensorFlow を用いた MNIST 学習の性能評価

鵜野圭介[†] 大島聡史[‡] 片桐孝洋[‡] 永井亨[‡]名古屋大学 情報学部 コンピュータ科学科[†] 名古屋大学 情報基盤センター[‡]

1. はじめに

分散学習はデータ並列とモデル並列の2通りがある。それぞれに異なる長所があり、長所は、データ並列は学習の高速化、モデル並列は大規模モデルの学習が挙げられる。しかし現状では、データ並列の方が適用のしやすさや実装が容易であるために、データ並列に比べ、モデル並列を扱うライブラリが少ない。

Mesh TensorFlow[1]は、2019年に提案された分散型深層学習用のフレームワークである。モデル並列によって大規模ニューラルネットワーク学習を行うといった特徴を持つ。しかし、BERT型の構造を持つニューラルネットワークのみにしか使用できないこともあり、性能評価事例が多くない。

そこで本研究では、GPU スパコン上で Mesh TensorFlow の性能評価を行う。性能評価のサンプルコードではパラメータとして、バッチサイズやレイアウトルール等が存在するため、それらを考慮した性能評価を行う。

2. Mesh TensorFlow

本研究では、深層学習フレームワークの1つである TensorFlow 上のレイヤーとして実装された Mesh TensorFlow を用いる。

Mesh TensorFlow のターゲットモデルとして扱われる言語モデルのパラメータは、数億を軽く超える。そのため、データ並列トレーニングを行うことは難しい。そこで Mesh TensorFlow を用いることでモデル並列トレーニングを可能とし、より高度な計算が可能となる。

Mesh TensorFlow では、プロセッサセットを n 次元のメッシュとして表示し、分割されたテンソルをメッシュに配置する。その後バッチをプロセッサの行に、中間層(隠れ層)をプロセッサの列に分割して計算が行われる。バッチを列に中間層を行に分割することも可能であり、レイアウトルールと呼ばれる。

計算の過程で中間層のサイズ等、複数のパラメータが存在するが、バッチサイズは学習回数に直結するため、結果に影響が表れやすい。

したがって本研究では MNIST データを活用したサンプルコードを実行するにあたり、バッチサイズの値を変化させて性能評価を行った。

3. 性能評価

3.1 実験方法

本実験では MNIST データを用いて学習の性能評価を行った。MNIST モデルは 0 から 9 までの手書き数字の画像データで構成されている。それぞれ 8bit グレースケールからなる 28×28 ピクセルの画像データであり、6 万枚の訓練用データと 1 万枚のテスト用データが入ったデータセットである。

ソースコードは Mesh TensorFlow のリポジトリ下にあるサンプルコードを用いて行い、バッチサイズを変更することで学習の正解率と損失の変化を観測した。

実行する際のバッチジョブスクリプトでは環境変数である `CUDA_VISIBLE_DEVICES` を `{OMPI_COMM_WORLD_LOCAL_RANK}` に設定することで各プロセスにおいて異なる GPU を使用した。

実験には名古屋大学情報基盤センターに設置されたスーパーコンピュータ「不老」Type II サブシステムを使用し、1 ノード 4GPU で実行した。ハードウェア構成、実行環境を表 1、表 2 に示す。

表 1 「不老」Type II サブシステムのハードウェア構成

CPU	Intel Xeon Gold 6230, 2.10GHz, 20 コア, ノードあたり 2 基
GPU	NVIDIA Tesla V100 SXM2 版, ノードあたり 4 基
メモリ	メインメモリ 384GiB/ノード デバイスメモリ 32GiB/GPU
ノード数	221 ノード
総理論演算性能	7.489PFLOPS (CPU 0.594PFLOPS, GPU 6.895PFLOPS)

Performance evaluation of MNIST learning using Mesh TensorFlow

[†] Keisuke Uno, Department of Computer science, School of Informatics, Nagoya University

[‡] Satoshi Ohshima, Takahiro Katagiri, Toru Nagai, Information Technology Center, Nagoya University

表2 実行環境

Python	Ver. 3.6.8
TensorFlow	Ver. 2.6.0
CUDA	Ver. 11.2.1
CUDNN	Ver. 8.1.1
GCC	Ver. 8.4.0
OpenMPI	Ver. 4.1.2

3. 2 実験結果

バッチサイズを変化させながら MNIST データを学習させて評価を行った。正解率、損失の結果をそれぞれ図1、図2に示す。

図1より、どのバッチサイズも正解率が収束するまでの epoch 数は変わらないが、バッチサイズが小さいほど初期から正解率が高いことが見て取れる。バッチサイズ10と250で比較すると初期正解率に0.03以上の差があり、epoch数を増加させて十分に学習させると差は0.005程度まで小さくなった。

同様に図2よりバッチサイズが小さくなるほど初期損失も小さくなることが見て取れる。しかしバッチサイズ10の場合のみ、epoch数5を境に過学習が始まっていることが分かる。結果として他のバッチサイズはepoch数20で0.05程度に収まるのに対し、バッチサイズ10の場合は損失が約2倍となった。

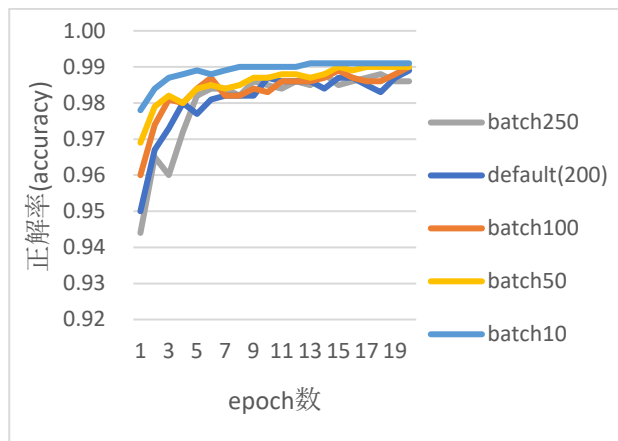


図1 各バッチサイズにおける正解率

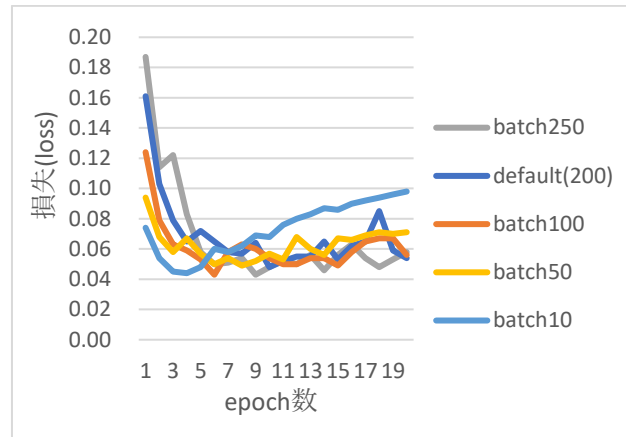


図2 各バッチサイズにおける損失

4. 考察とまとめ

本稿では、Mesh TensorFlow を用いてバッチサイズを変化させながら MNIST データを学習させた。実験結果から、バッチサイズが大きい場合に比べ、小さく設定したときに高パフォーマンスを発揮する傾向があることがわかった。

また図2において、バッチサイズ10の場合に過学習が早く起こった原因としては、図1からも見て取れるように、学習効率が高かったためと推察される。ここでは1ノード4GPUを用いて実験を行ったが単一GPU、複数ノードにて実行した場合、結果は異なると考えられる。GPUの数の変化は一度に扱うバッチサイズの変化に直結するため、単一GPUの場合は約0.25倍、複数ノードの場合は約ノード数倍のバッチサイズを扱えると考えられる。

今後の課題として、より高効率な学習を行うための最適条件を模索するために、他パラメータと同時にバッチサイズを変化させたときの挙動を評価することが挙げられる。また今回はMNISTデータを用いたが、更に精度の低いデータセットを用いた場合の正解率、損失についても調査していくことを考えている。さらに、ノード数を増やすことで、1ノードのメモリ量を打破し、どれだけ大規模な問題が扱えるかの検証を行っていく必要がある。

謝辞 本研究は JSPS 科研費 JP19H05662 の助成を受けたものです。

参考文献

- [1] Mesh TensorFlow -Model Parallelism Made Easier,
<https://github.com/tensorflow/mesh>
 (最終閲覧日 2021/12/21)