Vol.2018-HPC-165 No.2
2018/7/30

K-FACと分散処理による深層学習の高速化

辻 陽平[1,2,a)]　大沢 和樹[1]　横田 理央[1]　松岡 聡[3,1]

概要：機械学習の手法である深層学習は，モデルの認識精度の高さから現在多くの研究開発が行われている．とくに画像認識は様々なタスクの中でも非常に活発な分野である．深層学習では選択するモデルやデータセットによって長時間の計算が必要になる．計算時間を短縮する一つの方法として，分散処理が挙げられる．深層学習の分散処理は，いくつかの仮定のもとで，十分にスケールするが，同時に学習結果である認識精度が悪化するという問題がある．既存の研究からこの精度悪化の問題を回避する手法はいくつか提案されている．最適化手法である自然勾配法も有効な手段として議論されている．我々はこの自然勾配法の近似手法である K-FAC を分散で実行できるよう深層学習フレームワーク上に実装した．既存の分散実装ではスケールがしにくいという問題を同期的な集団通信と部分的なモデル並列により対処した．また分散環境で実行し計算性能と認識精度の評価を行った．

1. はじめに

近年，深層学習の研究開発が非常に活発となっており，その認識精度の高さと適用範囲の広さから，すでに実社会への応用も始まっている．

深層学習は古典的な機械学習に比べモデルのパラメータ数が多く，学習の時間・空間計算量が大きいという問題がある．初期の深層学習研究はこの問題のために実用的なモデルの学習が困難であった．しかし，GPU を汎用計算に利用する GPGPU により大幅に計算時間が短縮され，現在ではほとんど深層学習フレームワークが GPU を用いた計算に対応している．しかし，より認識精度や汎化性能が高いモデルや学習方法への要求により 1 つの GPU では計算資源が十分でない場合が発生した．分散深層学習はこのような文脈で現れ，現在，深層学習の計算を高速化する場合の一つの選択肢となっている．

分散深層学習の並列方法には大きく分けて，モデル並列とデータ並列の 2 つの方法がある．モデル並列はモデルのパラメータ $\mathbf{w}$ を複数のプロセス間で分散させて保持し，正伝播や逆伝播の際に通信を行う．一方データ並列では全てのプロセスがモデルのパラメータ $\mathbf{w}$ のコピーを保持し，各プロセスが異なるデータに対して計算を行い，定期的に同期を行う．この場合，通信は同期の際に発生する．より一般的なのはデータ並列であり，十分なデータを各プロセスに与えることが可能であるならば，すなわち弱スケーリングが可能ならば，この並列方法はスケールすることが知られている．

分散深層学習は計算の高速化ができるが，認識精度の視点では必ずしも良い選択肢とは言えない．深層学習の最適化手法，確率的勾配降下法（SGD），ではミニバッチごとに勾配計算を行うが，プロセス数を増やすことによってミニバッチサイズ（バッチサイズ）が増加する．バッチサイズの増加に伴い，認識精度が悪化することが知られており [1]，この精度悪化の抑制とその理由の解明が分散深層学習の研究において重要なテーマとなっている．

本研究では大規模な分散深層学習の計算に伴う認識精度の悪化を抑えるために最適化手法の自然勾配法（Natural Gradient Descent)[2], [3] を近似した K-FAC（Kronecker-Factored Approximate Curvature）[4] を用いる．この K-FAC を大規模な分散環境でもスケールするように実装し，その計算性能の測定と既存の最適化手法との比較・議論を行った．

本論文の構成は次にようになっている．2 節では本研究の背景である，分散深層学習，自然勾配法，そして K-FAC について述べる．3 節では既存の分散 K-FAC 実装と我々の実装について述べる．4 節では我々の実装の実験結果とその評価を行う．5 節では分散深層学習の関連研究について述べる．そして 6 節で，まとめと今後の課題について述べる．

[1] 東京工業大学
[2] 産業技術総合研究所
[3] 理化学研究所　計算科学研究センター
[a)] tsuji.y.ae@m.titech.ac.jp

ⓒ 2018 Information Processing Society of Japan

Vol.2018-HPC-165 No.2
2018/7/30

2. 背景

2.1 分散深層学習での精度の悪化

深層学習では式 (1) に示す確率的勾配降下法を用いてモデル $f : x \mapsto y$ のパラメータ $\mathbf{w}$ の最適化を行う．ただし l は誤差関数，B はミニバッチ，η は学習率，t は反復数である．

$$
\begin{aligned}
L &\leftarrow \frac{1}{|B|} \sum_{(x,y)\in B} l\left(y, f(x; \mathbf{w}_t)\right) \\
\Delta \mathbf{w}_t &\leftarrow \frac{\partial L}{\partial \mathbf{w}} \qquad (1) \\
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta \Delta \mathbf{w}_t
\end{aligned}
$$

n プロセスの分散深層学習のデータ並列では，それぞれのプロセスが異なるミニバッチ $B^{(1)}, \cdots, B^{(n)}$ に対して更新ベクトル $\Delta\mathbf{w}^{(1)}, \cdots, \Delta\mathbf{w}^{(n)}$ を計算する．パラメータの更新時には，すべてのプロセスが同じ算術平均 $\Delta\mathbf{w}_t = \sum_i \Delta\mathbf{w}^{(i)}/n$ を用いる．データ並列は通常，各プロセスのバッチサイズ $|B^{(i)}|$ を固定し，プロセス数 n を大きくする．すなわち弱スケーリングを行う．n プロセスを用いた場合，一般的に n 倍の高速化を見込める．分散深層学習はこのような利点があるが，モデルの認識精度を悪化させてしまう問題がある．

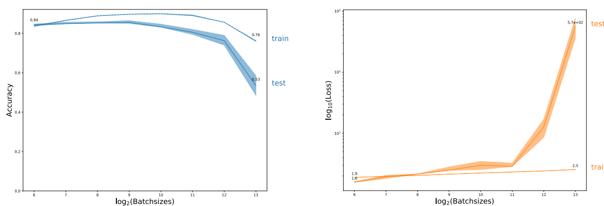

図 1 バッチサイズがモデルの認識精度にどのような影響を与えるかプロットした図．左右どちらの図も x 軸は合計のバッチサイズの対数 $\log_2(|B^{(i)}|)$ である．左図の y 軸は精度（accuracy），右図の y 軸は誤差の対数（$\log_{10}(\mathrm{loss})$）である．いずれの実験も CIFAR10 データセットに Chainer の VGG モデルを 30 エポック学習させている．薄色の箇所は複数回実験した最大と最初の間を塗ったものである．

図 1 にあるように学習するエポック数を固定して，バッチサイズを増加させることによって精度が悪化していることが実際の学習で確認できる．精度悪化を防ぐ既存の方法として，linear learning rate scale [5]，learning rate warm up [6]，LARS [1]，Ghost batch normalization [7] などがある．

2.2 自然勾配法と K-FAC

自然勾配法は情報幾何の分野で提案された反復最適化手法の一つである．以下では簡単のために反復数 t を省略して記号を表記する．

オンライン型の自然勾配法では更新ベクトル $\Delta\mathbf{w}$ をフィッシャー情報行列 $\mathbf{F}$ の逆行列を用いて式 (2) のように計算する．

$$
\Delta\mathbf{w} \leftarrow \mathbf{F}^{-1}\mathcal{G}_{\mathbf{w}} \qquad (2)
$$

ただし、$\mathcal{G}_{\mathbf{w}}$ は勾配であり、$\frac{\partial L}{\partial \mathbf{w}}$ と等しい．フィッシャー情報行列 $\mathbf{F}$ は次のように計算される．

$$
\begin{aligned}
\mathbf{F} &= \mathop{\mathbb{E}}_{(x,y)}\Big[(\nabla \log p(y \mid x, \mathbf{w}) \\
&\qquad\qquad (\nabla \log p(y \mid x, \mathbf{w})^{\top}\Big] \qquad (3) \\
&= \mathop{\mathbb{E}}_{(x,y)}\left[\mathcal{G}_{\mathbf{w}}\mathcal{G}_{\mathbf{w}}^{\top}\right] \qquad (4)
\end{aligned}
$$

ここで $\log p(y \mid x, \mathbf{w})$ はモデルの対数尤度関数である．分類タスクの典型的なニューラルネットワークの場合、対数尤度関数は誤差関数のマイナスと等しくなる*1．したがって、式 (3) から (4) は等式として変形できる．

フィッシャー情報行列 $\mathbf{F}$ はモデルのパラメータ数 N に対して、N^2 の要素数があるため、愚直な方法ではメモリの確保とその計算が困難となる．例として ResNet-50 では $N \approx 25550000$ であるので、すべての値を単精度型で保持した場合、合計でおよそ 2.3 ペタバイトの容量が必要となる．したがって近似による空間・時間計算量の削減は必須といえる．近似の手法としては、フィッシャー情報行列 $\mathbf{F}$ 及びその逆行列を近似するものと、更新ベクトル $\mathbf{F}^{-1}\mathcal{G}_{\mathbf{w}}$ を近似するものがある．K-FAC（Kronecker-Factored Approximate Curvature）[4] は前者の近似手法の一つである．本節では全結合層の K-FAC の計算方法についてのみ簡単に説明する．畳み込み層の計算方法は [8] に詳しく書かれている．まずはじめにいくつかの記号を用意する．

- $\mathbf{s}_\ell$：ℓ 層の活性化関数の入力、$W_\ell^{\top}\mathbf{a}_{\ell-1}$ と等しい
- $\mathbf{a}_\ell$：ℓ 層の活性化関数の出力、$\ell + 1$ 層の入力になる
- $\mathbf{g}_\ell$：$\mathbf{s}_\ell$ による誤差関数の勾配、$\frac{\partial L}{\partial \mathbf{s}_\ell}$ と等しい

ただし、W_ℓ は ℓ 層の重み行列である．$\mathbf{s}_\ell, \mathbf{a}_\ell$ はニューラルネットワークの順伝播時に計算され、$\mathbf{g}_\ell$ は逆伝播時に計算される．K-FAC の目標はフィッシャー情報行列の逆行列 $\mathbf{F}^{-1}$ を近似することであるが、そのためにまず $\mathbf{F}$ を対角ブロック近似する*2．この際、各ブロックはニューラルネットワークの各層に対応する．ℓ 層のブロックを $\mathbf{F}_\ell$ と表記する．ℓ 層のパラメータ W_ℓ の勾配が $\mathcal{G}_{W_\ell} = \mathbf{g}_\ell\mathbf{a}_{\ell-1}^{\top}$ であることに注意すると*3、$\mathbf{F}_\ell$ は式 (8) のように近似できる．

*1 分類タスクのニューラルネットワークではソフトマックス交差エントロピー関数が誤差関数 l として用いることが一般的である．このときに $-\log p(y \mid x, \mathbf{w}) = L(y, f(x; \mathbf{w}))$ となることが示せる．

*2 Martens ら [4] は対角ブロック近似とともにブロック三重対角近似を紹介しているが、ここでは対角ブロック近似のみ説明する．

*3 $\mathcal{G}_{\mathbf{w}}$ は N 次元縦ベクトルであったが、表記の簡略のために $\mathcal{G}_{W_\ell}$ は ℓ 層の重み行列 W_ℓ と同じ次元数の行列となっている．そのため式 (5) で vec によって縦ベクトルへの変形を行っている．

ⓒ 2018 Information Processing Society of Japan

$$\mathbf{F}_\ell = \underset{(x,y)}{\mathbb{E}} \left[\mathrm{vec}\left(\mathcal{G}_{W_\ell}\right) \mathrm{vec}\left(\mathcal{G}_{W_\ell}\right)^\top \right] \tag{5}$$

$$= \underset{(x,y)}{\mathbb{E}} \left[\mathbf{g}_\ell \mathbf{g}_\ell^\top \otimes \mathbf{a}_{\ell-1} \mathbf{a}_{\ell-1}^\top \right] \tag{6}$$

$$\approx \underset{(x,y)}{\mathbb{E}} \left[\mathbf{g}_\ell \mathbf{g}_\ell^\top \right] \otimes \underset{(x,y)}{\mathbb{E}} \left[\mathbf{a}_{\ell-1} \mathbf{a}_{\ell-1}^\top \right] \tag{7}$$

$$= G_\ell \otimes A_{\ell-1} \tag{8}$$

式 (6) から式 (7) への近似は $\mathbf{g}_\ell$ と $\mathbf{a}_{\ell-1}$ の独立性の近似を用いている．実際の計算時には、G_ℓ と $A_{\ell-1}$ の期待値計算はミニバッチによる平均によって近似する．これらを用いて ℓ 層の更新ベクトル $\Delta \mathbf{w}_\ell$ は式 (9) のように近似できる．

$$\begin{aligned} \Delta \mathbf{w}_\ell &= \mathbf{F}_\ell^{-1} \mathrm{vec}\left(\mathcal{G}_{W_\ell}\right) \\ &\approx \left(G_\ell \otimes A_{\ell-1}\right)^{-1} \mathrm{vec}\left(\mathcal{G}_{W_\ell}\right) \\ &= G_\ell^{-1} \otimes A_{\ell-1}^{-1} \mathrm{vec}\left(\mathcal{G}_{W_\ell}\right) \\ &= \mathrm{vec}\left(A_{\ell-1}^{-1} \mathcal{G}_{W_\ell} G_\ell^{-1}\right) \end{aligned} \tag{9}$$

式 (9) をすべての層 ℓ について計算すれば更新ベクトルが計算可能である．この近似により、例として ResNet50 では容量を 700 メガバイト程度に抑えることができる．

3. 分散 K-FAC

3.1 設計

K-FAC によって，実用的に計算が不可能であった自然勾配法が近似的に計算可能になったが，依然 G_ℓ や $A_{\ell-1}$ の逆行列計算を必要とし，単純な確率的勾配降下法に比べ計算量が多い．このような計算も分散処理によって並列計算を行い高速化することができる．Ba ら [9] は深層学習フレームワーク TensorFlow[10] を用いて分散処理に対応した K-FAC を実装し評価を行った．彼らの分散方法は次の特徴がある．

- パラメータサーバ方式のデータ並列である
- 勾配 $\mathcal{G}_\mathbf{w}$ と $G_\ell, A_{\ell-1}$ の計算に異なるミニバッチを用いている
- 逆行列計算 $G_\ell^{-1}, A_{\ell-1}^{-1}$ とその適用は複数反復に一回しか行わない

図 **2** に彼らの設計を示す．パラメータサーバ方式はマスタースレーブ型のプロセス分散方法であるが，この方式ではマスターを端点とする通信がボトルネックになりやすく，大規模な環境でスケールしないという問題がある．また，役割が異なる複数のスレーブが存在するため，全体の規模をスケールさせたときの適切な割当数を決定するのが難しい．逆行列 G^{-1}, A^{-1} が計算時間のボトルネックになるため，毎反復逆行列計算を行わず，複数反復に一度だけ行うように設計している．すなわち非同期に $\mathbf{F}^{-1}$ を適用しているのである．そのため古い情報の入った $\mathbf{F}^{-1}$ を更新式に適用させており，精度への悪化が懸念される．

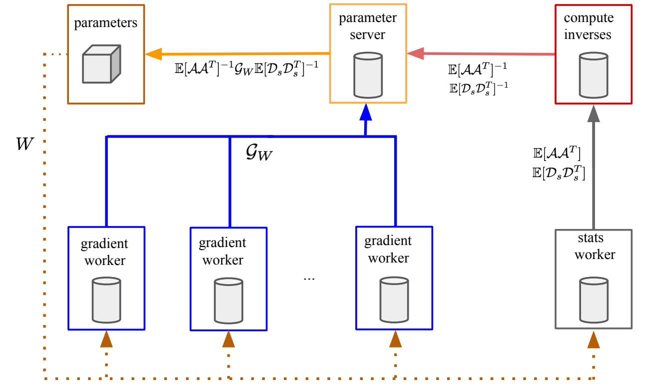

図 **2** Ba ら [9] から引用．青い gradient worker が勾配 $\mathcal{G}_\mathbf{w}$ を計算，灰色の stats worker が A, G を計算，赤色の compute inverse が A^{-1}, G^{-1} を計算する．gradient worker と stats worker はオレンジ色のパラメータサーバから最新パラメータ $\mathbf{w}$ を取得し，異なるミニバッチを用いてそれぞれの計算を行う．

我々はこれらの問題に対し，大規模な分散環境であってもスケールする同期的な分散 K-FAC を設計した．概略図を図 **3** に示す．我々の設計では Ba らの設計と比較して次のような特徴がある．

- 全ワーカ方式のデータ並列である
- 勾配 $\mathcal{G}_\mathbf{w}$ と $G_\ell, A_{\ell-1}$ の計算にすべて同じミニバッチを用いている
- 逆行列計算 $G_\ell^{-1}, A_{\ell-1}^{-1}$ とその適用を毎イテレーション行っている

通信はすべて集団通信を用いることで，プロセスが増加しても通信がボトルネックにならないよう設計した．さらに各層の逆行列計算を複数プロセスに分担させることで，モデル全体の更新ベクトルを高速に計算し，毎反復 $\mathbf{F}^{-1}$ を適用できるようにした．

3.2 実装

分散 K-FAC の実装は汎用性をもたせるために既存の深層学習フレームワーク上に実装した．具体的には深層学習フレームワーク Chainer[11] とその分散実装 ChainerMN[12] を利用した．ChainerMN は Chainer のラッパーとして作られており，通信などに関わる部分のみが記述してある．さらに ChainerMN は全ワーカ型の通信方法をデフォルトで採用しており，分散学習に対応している他のフレームワークに比べスケールしやすいという報告がある．

K-FAC を様々なニューラルネットワークに適用するために，Chainer の Optimizer クラスを継承したクラスとして K-FAC を実装した．アルゴリズム 1 に簡略した疑似コードを示した．4,5,6 行目が K-FAC 特有の計算となる．

通常の確率的勾配降下法を分散処理する場合，各プロセスが自身が割り当てられたミニバッチ $B^{(i)}$ を用いて計算した勾配 $\mathcal{G}_\mathbf{w}^{(i)}$ の算術平均を求める必要がある．ChainerMN ではこの計算を毎反復の勾配計算のあとに Allreduce 通信

ⓒ 2018 Information Processing Society of Japan

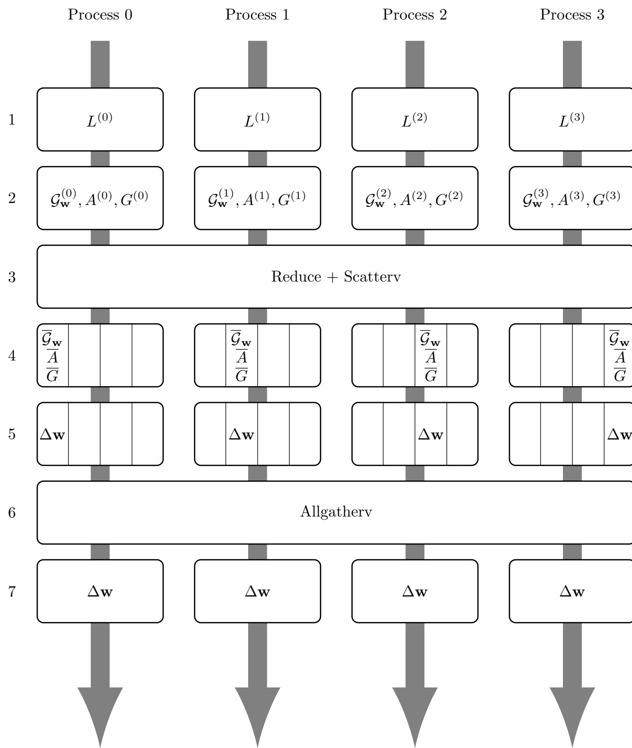

図 3　提案する同期的な分散 K-FAC 実装．縦に伸びている灰色の矢印が各プロセスの時間方向を表している．以下 i ステップ目というのは左側の数字を指し示すこととする．1 ステップ目では順伝播，2 ステップ目では逆伝播の計算を行っている．3 ステップ目で Reduce と Scatterv の集団通信を行っている．これによって各プロセスが 1 層分の勾配ベクトル計算に必要な $\overline{\mathcal{G}}_{\mathbf{w}_\ell}, \overline{G}_\ell, \overline{A}_{\ell-1}$ の層ごとのミニバッチ平均を持つことができる．4 ステップ目から 5 ステップ目では逆行列計算と更新ベクトル計算が行われる．6 ステップ目では各プロセスが計算した各層の更新ベクトルを全てのプロセスが受けてれるよう Allgatherv 通信を行う．通信後にはすべてのプロセスがすべての層の更新ベクトル $\Delta W_\ell \mathrm{vec}(A_{\ell-1}^{-1}\mathcal{G}_{W_\ell}G_\ell^{-1})$ を持つことができる．7 ステップ目で更新ベクトルから各プロセスが自分のパラメータ $\mathbf{w}$ を更新し，1 イテレーションが完了する．

アルゴリズム 1

1: **while** ¬ 学習完了 **do**
2: 　正伝播
3: 　逆伝播（$\mathbf{g}, \mathbf{a}$ の計算）
4: 　G, A の計算
5: 　G^{-1}, A^{-1} の計算
6: 　$\Delta\mathbf{w}$ の計算
7: 　$\mathbf{w} \leftarrow \mathbf{w} - \eta\Delta\mathbf{w}$
8: **end while**

を行うことで実現している．我々の設計した分散 K-FAC では Allreduce ではなく，Reduce + Scatterv と Allgatherv を通信として用いるために別に実装が必要であった．既存のライブラリの場合，集団通信では MPI または NCCL を用いることができる．

図 4 から分かるよう通信するデータが大きい場合，NCCL の方がより高速に通信を完了することができる．この結果

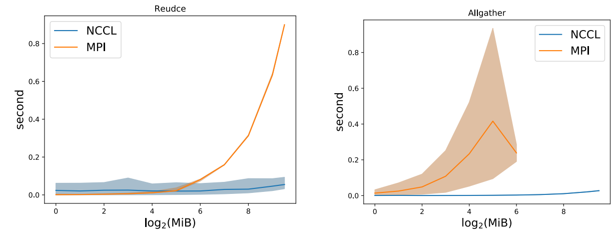

図 4　MPI と NCCL の通信完了までにかかる時間の比較．左図は Reduce，右図は Allgather による測定である．すべての実験は TSUBAME3.0 を 2 ノード用いている．x 軸は通信のデータサイズの log をとったものである．いずれの図も NCCL がより良い結果を出していることが分かる．薄色の箇所は複数回実験した最大と最初の間を塗ったものである。

から通信には NCCL を採用した．NCCL には vector 通信（Scatterv や Allgatherv）などが実装されていないため，他の通信を用いて同じ結果を得る必要がある．実験によって，Reduce，Allgatherv，Allreduce の 3 つの通信がいずれもほぼ同等の性能を出すことが分かったので我々の実装の Reduce+Scatterv，Allgatherv いずれにも Allreduce を用いた．

4. 評価

4.1 実行環境

実行環境として，東京工業大学，学術国際情報センターの TSUBAME3.0 を利用した．表 **1** に実行時のハードウェア環境とソフトウェア環境を示す．

表 1　TSUBAME3.0 の 1 ノードあたりの構成と実行環境

CPU	Intel Xeon E5-2680 v4 × 2CPU
周波数	2.4 GHz
物理コア数/CPU	14
スレッド数/CPU	28
メモリ/ノード	256 GiB
GPU	NVIDIA(R) Tesla(R) P100 × 4
インターコネクト	Intel(R) Omni-Path HFI 100Gbps × 4
MPI	Open MPI 2.1.2
CUDA	8.0
NCCL	2.2.13
Chainer	5.0.0b
ChainerMN	1.3.0

4.2 計算性能

図 **5** にスケーリングを測定した結果を示す．我々の分散 K-FAC 実装では逆行列計算にモデル並列を用いているので，線形よりもよくスケールしていることがわかる．現在の実装ではモデル並列部分の最大並列数はモデルの層数と決定しまっている．したがって，プロセス数がこの最大並列数を超えた場合，分散の確率的勾配降下法と同じようなスケーラビリティとなることが予想される．

ⓒ 2018 Information Processing Society of Japan

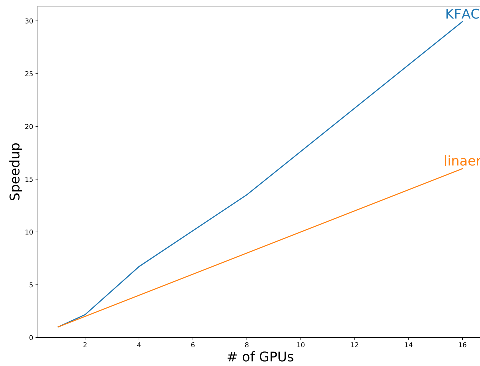

図 5　スケーラビリティを示した図．x 軸が GPU 数，y 軸が 1GPU に対する速度上昇を表している．オレンジ色の線は線形スケール，水色は実際の K-FAC のスケールを表している．図からわかるように線形よりも良いスケールを得ることができる．実験のデータセットは CIFAR10，モデルは VGG を用いている．

4.3　学習精度

図 6 に 1GPU での K-FAC の学習曲線を示す．K-FAC が Adam よりもエポック数に対してい早い良い収束をしていることがわかる．この実験では 1GPU しか使用していないために計算時間に対する収束は Adam よりも遅い結果となってしまっている．

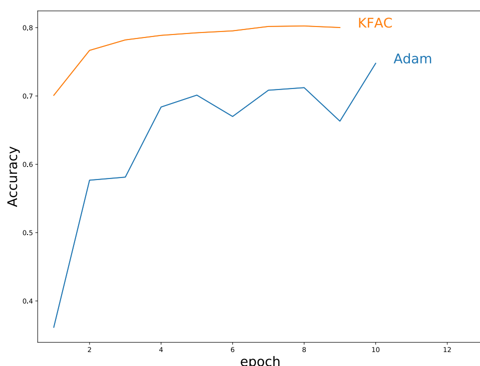

図 6　異なる最適化手法に対する収束性の違いを示す図．K-FAC が Adam よりも早く収束していることがわかる．x 軸はエポック数，y 軸は精度を表している．y 軸の開始点が 0 でないことに注意する．GPU 数以外の実験設定はすべて図 5 と同じである．

5.　関連研究

5.1　分散深層学習における精度悪化

Krizhevsky ら [5] はバッチサイズに比例して学習率を増加させる LR linear scale によって AlexNet をバッチサイズ 1024 でほぼ精度の減少なしに学習させた．

Goyal ら [6] は Krizhevsky らの LR linear scale に加え学習率を小さいものから徐々に大きくする LR warm up を提案した．彼らは少ない精度悪化で ResNet50 をバッチサイズ 8192 で 1 時間で学習を完了させた．

You ら [1] は上記の手法ではバッチサイズによる精度の悪化は十分に抑えられないとし，Layer-wise Adaptive Rate Scaling (LARS) を提案した．LARS では各層のパラメータ $\mathbf{w}_\ell$ とその勾配 $\frac{\partial L}{\partial \mathbf{w}_\ell}$ のノルムによって動的に学習率を変化させる．彼らは LARS を用いて ResNet-50 を 32K で精度悪化なしに学習を完了した．

Hoffer ら [7] は汎化性能の悪化はバッチサイズによるものではなく，少ない反復数によるものが原因であると実験的に示した．さらに分散深層学習において各プロセスが少ないバッチサイズによってバッチ正規化を行う Ghost-BN を提案した．

Keskar ら [13] は大きなバッチサイズの汎化性能の悪化は “sharp-minima” に落ちることが原因であると提唱した．さらに既存の精度悪化を抑制する手法がどれだけ鋭さ（sharpness）に影響するか評価した．

Smith ら [14] は深層学習の汎化性能への影響への議論を行い，さらに “noise scale” と呼ばれる指標を提案した．実験的に noise scale によって sharp-minimizer から抜け出せるかが決定することを示した．

6.　まとめと今後の課題

本研究では大規模な分散深層学習で問題となる精度の悪化を自然勾配法の近似手法である K-FAC を用いることによって抑制することを試みた．また K-FAC を分散で実行が可能となるよう深層学習フレームワーク上に実装し，評価を行った．

今後の課題として，大きく 2 つのテーマが挙げられる．まず 1 つはロードアンバランスの解消である．逆行列計算の対象となる行列はニューラルネットワークの層と対応しているため大小様々な次元数の行列が存在する．並列化を行ったとしても逆行列の計算時間は最大の次元数を持つ行列に律速されるため，1 つの行列を 1 つのプロセスに割り当てるのではなく，大きな行列を複数のプロセスで並列で実行できるよう実装するのがロードバランスとして良い．続いて 2 つ目はより近似精度の良い手法への改良である．本研究で我々が目標としたのは自然勾配法というフィッシャー情報行列を用いる最適化手法であり，K-FAC はあくまでその近似である．より大規模な分散環境を想定して，より精度の良い近似手法をとることが考えられる．

謝辞　本研究の一部は，JST CREST(JPMJCR1303, JP-MJCR1687) 及び，産総研・東工大実社会ビッグデータ活用オープンイノベーションラボラトリの支援による．

ⓒ 2018 Information Processing Society of Japan

参考文献

[1] You, Y., Gitman, I. and Ginsburg, B.: Large Batch Training of Convolutional Networks, pp. 1–8 (online), available from ⟨http://arxiv.org/abs/1708.03888⟩ (2017).

[2] Amari, S.-I.: Natural gradient works efficiently in learning, *Neural computation*, Vol. 10, No. 2, pp. 251–276 (1998).

[3] Amari, S. and Douglas, S.: Why natural gradient?, *1998. Proceedings of the 1998 IEEE*, Vol. 9, pp. 1213–1216 (online), DOI: 10.1109/ICASSP.1998.675489 (1998).

[4] Martens, J. and Grosse, R. B.: Optimizing Neural Networks with Kronecker-factored Approximate Curvature, *CoRR*, Vol. abs/1503.05671 (online), available from ⟨http://arxiv.org/abs/1503.05671⟩ (2015).

[5] Krizhevsky, A.: One weird trick for parallelizing convolutional neural networks, *CoRR*, Vol. abs/1404.5997 (online), available from ⟨http://arxiv.org/abs/1404.5997⟩ (2014).

[6] Goyal, P., Dollár, P., Girshick, R. B., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y. and He, K.: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour, *CoRR*, Vol. abs/1706.02677 (online), available from ⟨http://arxiv.org/abs/1706.02677⟩ (2017).

[7] Hoffer, E., Hubara, I. and Soudry, D.: Train longer, generalize better: closing the generalization gap in large batch training of neural networks, *NIPS*, (online), available from ⟨http://arxiv.org/abs/1705.08741⟩ (2017).

[8] Grosse, R. and Martens, J.: A Kronecker-factored Approximate Fisher Matrix for Convolution Layers, *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, JMLR.org, pp. 573–582 (online), available from ⟨http://dl.acm.org/citation.cfm?id=3045390.3045452⟩ (2016).

[9] Ba, J., Grosse, R. and Martens, J.: Distributed Second-Order Optimization Using Kronecker-Factored Approximations, *Iclr 2017*, No. 2008, pp. 1–17 (2017).

[10] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015). Software available from tensorflow.org.

[11] Tokui, S., Oono, K., Hido, S. and Clayton, J.: Chainer: a Next-Generation Open Source Framework for Deep Learning, *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS)* (2015).

[12] Akiba, T., Fukuda, K. and Suzuki, S.: ChainerMN: Scalable Distributed Deep Learning Framework, *Proceedings of Workshop on ML Systems in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)* (2017).

[13] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M. and Tang, P. T. P.: On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, pp. 1–16 (online), DOI: 10.1227/01.NEU.0000255452.20602.C9 (2016).

[14] Smith, S. L. and Le, Q. V.: A Bayesian Perspective on Generalization and Stochastic Gradient Descent, No. 2016, pp. 1–13 (online), available from ⟨http://arxiv.org/abs/1710.06451⟩ (2017).

ⓒ 2018 Information Processing Society of Japan