Vol.2018-HPC-163 No.14
2018/3/1

フィッシャー情報行列のクロネッカー因子分解を用いた深層ニューラルネットワークの分散学習

大友 広幸[†1,a)]　大沢 和樹[†1,b)]　横田 理央[†1,c)]

概要：深層ニューラルネットワークの画像認識モデルの学習では，訓練データに対する誤差関数を最小にすることを目的とする．パラメータを反復的に探索するために，ミニバッチを用いた確率的勾配降下法（SGD）による学習が広く用いられている．近年では，大規模計算資源を活用した学習高速化のアプローチとして，巨大なミニバッチを分割して計算を分散化させる大規模分散学習が注目されているが，SGD による巨大なミニバッチでの学習においては，モデルの認識精度が劣化することが報告されている．一方で，フィッシャー情報行列で定まるリーマン計量を用いて誤差関数の自然勾配を求める自然勾配法（NGD）では，SGD より大幅に少ない反復数でモデルが学習される上に巨大なミニバッチでの学習においても，モデルの認識精度の劣化を抑えられるという特長を持っている．膨大な数のパラメータを持つモデルの NGD による学習では巨大なフィッシャー情報行列の逆行列計算が学習全体のボトルネックとなるため，この計算を近似し，NGD を擬似的に実現する手法が提案されてきている．本研究ではクロネッカー因子分解を用いてフィッシャー情報行列の逆行列を近似的に計算する学習手法（K-FAC）に注目し，K-FAC による大規模分散学習における学習の高速化と画像認識モデルの認識精度について調査した．

Ootomo Hiroyuki[†1,a)]　Oosawa Kazuki[†1,b)]　Yokota Rio[†1,c)]

1. はじめに

近年，画像認識をはじめとした多くの分野で深層学習を用いた手法が広く用いられている．年々深層ニューラルネットワークの構造は複雑さを増しており，強化学習を用いて深層ニューラルネットワークを設計する研究 [1] も行われている．

一方，深層ニューラルネットワークの学習においてはパラメータをユークリッド空間上の点と仮定した確率的勾配降下法を用いた最適パラメータ探索が主流となっている．これに対して Amari が考案した自然勾配法 [3] ではパラメータの集合がリーマン多様体上にあるとし，リーマン計量であるフィッシャー情報行列の逆行列を用いて最適パラメータを探索する．フィッシャー情報行列の大きさはニューラルネットワークの大きさに依存し，ニューラルネットワークの大きさが大きくなっている昨今，フィッシャー情報行列を精度良く求めることはメモリ容量や計算量の観点から困難である．そのためフィッシャー情報行列の逆行列を近似的に求める様々な手法が提案されている．

本研究では Martens らが提案したクロネッカー因子分解を用いた近似手法 K-FAC[4] を用いた深層ニューラルネットワークの分散学習において，ミニバッチサイズを変化させた場合や分散数を変化させた場合，フィッシャー情報行列の逆行列計算を行うノードの数を変化させた場合での精度と収束の傾向を調査した．

[1] 情報処理学会
IPSJ, Chiyoda, Tokyo 101–0062, Japan
[†1] 現在，東京工業大学
Presently with Tokyo Institute of Technology
[a)] ootomo.h@rio.gsic.titech.ac.jp
[b)] oosawak@rio.gsic.titech.ac.jp
[c)] rioyokota@gsic.titech.ac.jp

2. 関連研究

2.1 確率的勾配降下法

ニューラルネットワークでは入力 $x \in \mathbb{R}^d$ に対し出力 $y = f(x) \in \mathbb{R}^{d'}$ を計算する．例えば 3 層ニューラルネットワークであれば，入力層と中間層 1 層と出力層からなり，入力層と中間層及び中間層と出力層の間で計算が行われる．$l+1$ 層ニューラルネットワークを考える．i 層目では入力 $a_{i-1} \in \mathbb{R}^{d_{i-1}}$ に対し出力 $a_i \in \mathbb{R}^{d_i}$ を計算する．この計算は重み行列 $W_i' \in \mathbb{R}^{d_i \times d_{i-1}}$，バイアス $b_i \in \mathbb{R}^{d_i}$，非線形な活性化関数 ϕ_i を用いて

$$s_i = W_i' a_{i-1} + b_i \tag{1}$$
$$a_i = \phi_i(s_i) \tag{2}$$

と書ける．ただし，$a_0 = x, a_l = y$ である．本論文では便宜上 $\overline{a}_i = \left[1\ a_i^T\right]^T$，$W_i = [b_i\ W_i']$ として式 1 を

$$s_i = W_i \overline{a}_{i-1} \tag{3}$$

と書く．

ニューラルネットワークの学習では出力 y と期待される出力 $z \in \mathbb{R}^{d'}$ との不一致度を測る損失関数 $L(y, z)$ を定義し，その期待値を目的関数とする最小化問題を解くために反復的にニューラルネットワークのパラメータ W を更新する．損失関数は，例えば訓練サンプルを用いて学習する画像認識の分野では交差エントロピーが広く用いられている．W をユークリッド空間上の点と仮定すると，勾配 $\nabla L = \frac{\partial L}{\partial W}$ は L が増加する方向を向いている．したがって反復数 t でのパラメータ $w^{(t)}$ を勾配の逆方向，すなわち $-\nabla L$ 方向へ更新すれば L を減少させることができる．したがってパラメータの更新式は

$$w^{(t+1)} = w^{(t)} - \epsilon \nabla L \tag{4}$$

と表される．ϵ は学習率と呼ばれ，更新の幅を決める係数である．学習率はミニバッチサイズなどを考慮して決め，反復数などに応じて途中変更なパラメータである．

確率的勾配降下法 (SGD;Stochastic Gradient Descent) ではミニバッチを用いて ∇L を計算し，反復的にパラメータを更新する．∇L の計算方法としては，出力 y と期待される出力 z の誤差を出力層から入力層にかけて伝播させることで ∇L を計算する誤差逆伝播法 [2] が主流である．

2.2 自然勾配法

SGD ではパラメータ W をユークリッド空間上の点と仮定し勾配を計算した．これに対し自然勾配法 (NGD;Narutal Gradient Descent) ではパラメータ W の集合をリーマン計量がフィッシャー情報行列 F で定まるリーマン多様体と見なし勾配を計算する．フィッシャー情報行列は正定値対称行列であるため逆行列が存在し，NGD のパラメータの更新式は

$$w^{(t+1)} = w^{(t)} - \epsilon F^{-1} \nabla L \tag{5}$$

で表される．

SGD を用いたニューラルネットワークの学習では損失関数 L の期待値を最小化するパラメータを探索するが，鞍点に陥り学習が停滞することがある．NGD は SGD と比べて鞍点に陥りにくいことや反復数に対して学習が速いこと，大きなミニバッチで学習を行っても精度が落ちにくいという特徴がある [5]．一方，昨今のニューラルネットワークではパラメータ数が増加しており，F 及び F^{-1} を正確に計算することはメモリ消費量や計算量の観点から困難である．そこで F^{-1} をスパース行列近似 [6]，低ランク近似 [7]，クロネッカー因子分解を用いた近似 [4]，などを用いて計算する手法が考案されている．

2.3 K-FAC

K-FAC(Kronecker-Factored Approximate Curvature) はクロネッカー因子分解を用いた近似手法として Martens らにより考案された [4]．K-FAC では 2 段階の近似を行うことでメモリ消費量と計算量を抑えている．

2.3.1 近似理論

行列 M に対し $[M]_{i,j}$ は M の (i, j) 成分を，ベクトル v に対し $[v]_i$ は v の i 番目の成分を指すとする．vec を $M \in \mathbb{R}^{m \times n}$ を $v \in \mathbb{R}^{mn}$ に $[v]_{i+nj} = [M]_{i,j}$ を満たしながら変換するベクトル化関数と定義する．ニューラルネットワークの全層でのパラメータを並べたベクトル θ を

$$\theta = \left[\mathrm{vec}(W_1)^T\ \mathrm{vec}(W_2)^T \cdots \mathrm{vec}(W_l)^T\right]^T \tag{6}$$

とする．$p(y|x, \theta)$ を入力 x に対する出力 y の確率密度関数とし，行列 u に対し

$$Du = -\frac{d \log p(y|x, \theta)}{du} \tag{7}$$

を定義すると，フィッシャー情報行列の定義から

$$F = E\left[\frac{d \log p(y|x, \theta)}{d\theta} \frac{d \log p(y|x, \theta)}{d\theta}^T\right] \tag{8}$$
$$= E\left[D\theta D\theta^T\right] \tag{9}$$

である．式 6 より

$$D\theta = \left[\mathrm{vec}(DW_1)^T\ \mathrm{vec}(DW_2)^T \cdots \mathrm{vec}(DW_l)^T\right]^T \tag{10}$$

であるから $1 \le i, j \le l$ として $F_{i,j} \in \mathbb{R}^{d_{i-1}d_i \times d_{j-1}d_j}$ を

$$F_{i,j} = E[\mathrm{vec}(DW_i)\mathrm{vec}(DW_j)^T] \tag{11}$$

と定義すると式 9 は

$$F = \begin{bmatrix} F_{1,1} & F_{1,2} & \cdots & F_{1,l} \\ F_{2,1} & F_{2,2} & \cdots & F_{2,l} \\ \vdots & \vdots & \ddots & \vdots \\ F_{l,1} & F_{l,2} & \cdots & F_{l,l} \end{bmatrix} \tag{12}$$

と書ける．$g_i = Ds_i$ とすると，誤差逆伝播法の過程より $\mathrm{vec}(DW_i) = g_i \overline{a_{i-1}}^T$ である．クロネッカー積 $\otimes$(付録 A.1) を用いると，$\mathrm{vec}(uv^T) = v \otimes u$ であることから式 11 は

$$\begin{aligned} F_{i,j} &= E\left[\mathrm{vec}(DW_i)\mathrm{vec}(DW_j)^T\right] \\ &= E\left[(\overline{a}_{i-1} \otimes g_i)(\overline{a}_{j-1} \otimes g_j)^T\right] \\ &= E\left[(\overline{a}_{i-1} \otimes g_i)(\overline{a}_{j-1}^T \otimes g_j^T)\right] \\ &= E\left[\overline{a}_{i-1}\overline{a}_{j-1}^T \otimes g_i g_j^T\right] \end{aligned}$$

と書ける．ここで $F_{i,j}$ に対して 1 段階目の近似を行い

$$\begin{aligned} F_{i,j} &= E\left[\overline{a}_{i-1}\overline{a}_{j-1}^T \otimes g_i g_j^T\right] \\ &\approx E\left[\overline{a}_{i-1}\overline{a}_{j-1}^T\right] \otimes E\left[g_i g_j^T\right] \\ &= \overline{A}_{i-1,j-1} \otimes G_{i,j} \equiv \tilde{F}_{i,j} \end{aligned}$$

を得る．ただし $\overline{A}_{i,j} = E\left[\overline{a}_i\overline{a}_j^T\right] \in \mathbb{R}^{d_i \times d_j}, G_{i,j} = E\left[g_i g_j^T\right] \in \mathbb{R}^{d_i \times d_j}$ であり，クロネッカー因子と呼ぶ．これにより F は

$$\tilde{F} = \begin{bmatrix} \tilde{F}_{1,1} & \tilde{F}_{1,2} & \cdots & \tilde{F}_{1,l} \\ \tilde{F}_{2,1} & \tilde{F}_{2,2} & \cdots & \tilde{F}_{2,l} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{F}_{l,1} & \tilde{F}_{l,2} & \cdots & \tilde{F}_{l,l} \end{bmatrix} \tag{13}$$

と近似される．

2 段階目の近似として $\tilde{F}$ の非対角ブロックを零行列とする．これにより $\tilde{F}$ は

$$\breve{F} = \mathrm{diag}(\tilde{F}_{1,1}\ \tilde{F}_{2,2}\ \cdots\ \tilde{F}_{l,l}) \tag{14}$$

と近似される．ここでクロネッカー積の性質より

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{15}$$

が成り立つため，

$$\begin{aligned} \breve{F}^{-1} &= \mathrm{diag}(\tilde{F}_{1,1}^{-1}\ \tilde{F}_{2,2}^{-1}\ \cdots\ \tilde{F}_{l,l}^{-1}) &(16)\\ &= \mathrm{diag}(\overline{A}_{0,0}^{-1} \otimes G_{1,1}^{-1}\ \cdots\ \overline{A}_{l-1,l-1}^{-1} \otimes G_{l,l}^{-1}) &(17) \end{aligned}$$

が従い，クロネッカー因子 $A_{i,i}^{-1}, G_{i,i}^{-1}$ を計算することで $\breve{F}^{-1}$ を計算できる．

2.3.2 数値計算

K-FAC の計算機上での計算は次の 3 つに大分できる．

- 勾配計算 (grad)：
  学習データを用いて各層における a_i, g_i 及び ∇L を計算する．
- クロネッカー因子更新 (cov)：
  クロネッカー因子 $\overline{A}_{i,i}, G_{i,i}$ は $a_i a_i^T, g_i g_i^T$ の期待値であるが，それらの真の分布は訓練データからではわかり得ない．そこで反復毎に移動平均をとることで経験的に推定する．
- クロネッカー因子逆行列計算 (inv)：
  cov で計算したクロネッカー因子 $\overline{A}_{i,i}, G_{i,i}$ の逆行列 $\overline{A}_{i,i}^{-1}, G_{i,i}^{-1}$ を計算する．

3. 実験

実験設定

実験には図 1 に示すニューラルネットワークを用い，CIFAR-10[*1]を学習データとしてを用いた．また，実験を行うにあたって次の 3 種類のノードを用意し，図 2 に示す同期型データ並列とした．

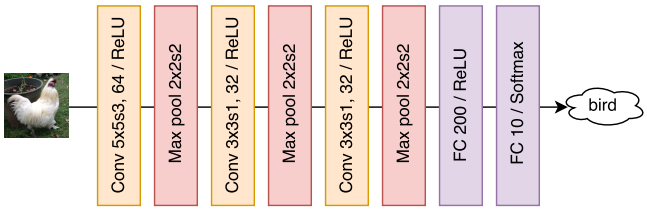

図 1　ニューラルネットワーク構成図

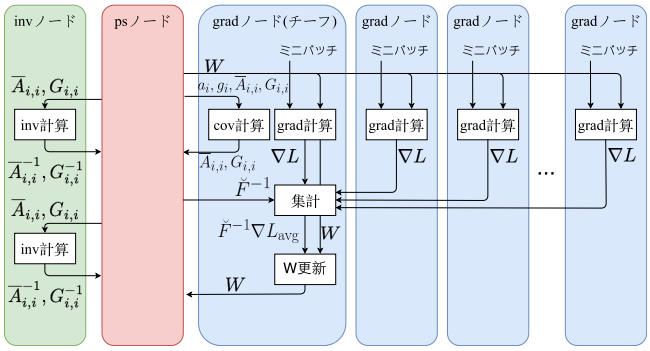

図 2　同期型データ並列

- パラメータサーバー [12] ノード (ps ノード)
  ニューラルネットワークのパラメータ及び K-FAC の保持及び共有のみを行う．
- 勾配計算，クロネッカー因子更新ノード (grad ノード)
  - ps ノードからパラメータ W を取得し，grad 計算を行う．
  - うち 1 ノード (チーフ) は cov 計算を行い，各 grad ノードが計算した勾配 ∇L を集計し平均をとったものを ∇L_{avg} とし，$\breve{F}^{-1}\nabla L_{\mathrm{avg}}$ を用いて ps ノードのパラメータを更新する．
- 逆行列計算ノード (inv ノード)
  ps ノードから $\overline{A}_{i,i}, G_{i,i}$ を取得し inv 計算を行う．計算した $\overline{A}_{i,i}^{-1}, G_{i,i}^{-1}$ で ps ノードのそれらを更新する．grad ノードとは独立しており，常時 inv 計算をし続ける．

実装には TensorFlow[*2] 1.4.0 を用い，ノード間の通信は Google が開発している RPC(Remote Procedure Call) である gRPC[*3]を用いた．

各実験におけるノード数と各 grad ノードが計算するミニバッチサイズなどの分散計算のための設定は表 2 に示す．

平均損失 $\overline{L}$ を

$$\overline{L} = \mathrm{avg}_{3-\mathrm{Experiments}}\left[\mathrm{avg}_{\mathrm{Mini-Batch}}\left[L(y,z)\right]\right] \tag{18}$$

のようにミニバッチでの損失関数値の平均をとり，さらに 3 回実験を行った際の平均と定義する．

[*1] https://www.cs.toronto.edu/ kriz/cifar.html
[*2] https://www.tensorflow.org
[*3] https://grpc.io

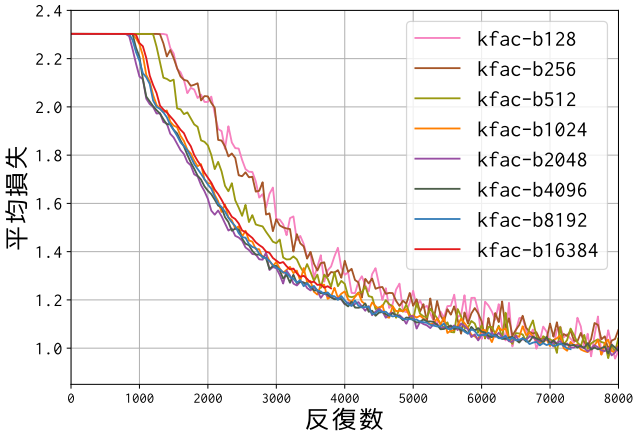

図 3　ミニバッチサイズ変更実験

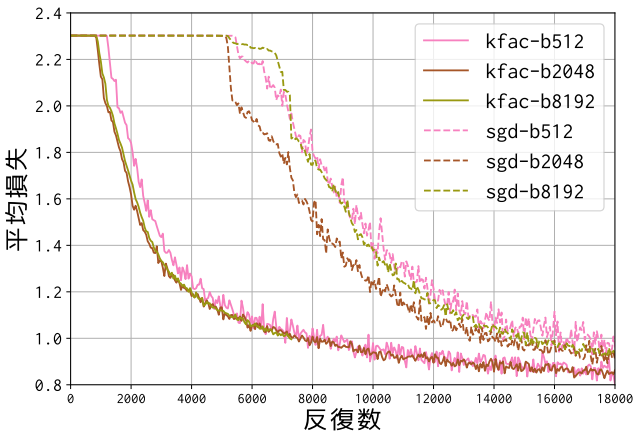

図 4　ミニバッチサイズ変更実験 (SGD との収束の比較)

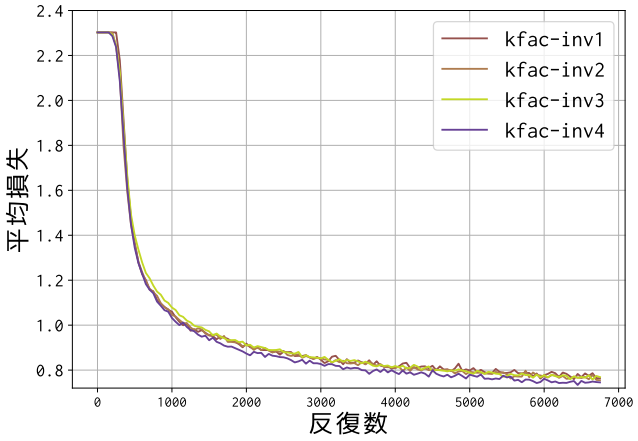

図 5　inv ノード数変更実験

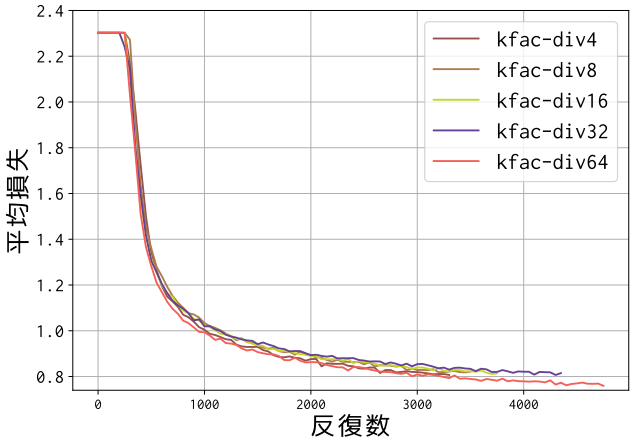

図 6　ミニバッチ分割数変更実験 (反復数比較)

実験 1:ミニバッチサイズ変更

大きいサイズのミニバッチでの損失関数値の減少傾向を調査するため，ミニバッチサイズを 128 から 16384 まで 2 倍ずつ変化させた際の平均損失の減少傾向を比較した (図 3)．横軸は反復数，縦軸は平均損失を表している．ラベルは kfac-b[ミニバッチサイズ] である．

ミニバッチサイズが 1024 以上のときは平均損失の減少傾向に差は見られなかった．ミニバッチサイズが 128,256,512 のときはミニバッチサイズが小さいほど平均損失の減少が遅くなった．以上より K-FAC を用いた場合でもミニバッチサイズを大きくするとある程度までは平均損失の減少速度は速くなるが，それ以上収束の傾向が変わらなくなるミニバッチサイズが存在すると考えられる．

また，同じネットワークをバッチサイズを 512，2048，8192 として SGD と K-FAC で学習した場合の平均損失の減少傾向を比較した (図 4)．横軸は反復数，縦軸は平均損失を表している．ラベルは [kfac/sgd]-b[ミニバッチサイズ] である．SGD ではバッチサイズが 2048 で最も平均損失の減少が速く，512 と 8192 では遅くなった．これに対し K-FAC では，バッチサイズを大きくしても平均損失の減少傾向が劣化することはなかった．これにより SGD に対する K-FAC の優位性を確認できた．

実験 2:inv ノード数変更

分散計算で inv ノード数 1，2，3，4 と変化させた際の平均損失の減少傾向を比較した (図 5)．横軸は反復数，縦軸は平均損失を表している．ラベルは kfac-inv[inv ノード数] である．

inv ノード数を変化させても平均損失の減少傾向に差がないことが分かる．これはこのニューラルネットワークでは inv ノードが 1 台で十分な頻度 (1 反復に 1 回以上) の inv 計算を行えているからであると考えられる．

より多くのパラメータ W をもつニューラルネットワークの学習では，1 回の grad 計算より inv 計算にかかる時間の方が長くなり，十分な頻度の inv 計算を行うためには inv ノードを複数台用意する必要が出てくると考えられる．

実験 3:ミニバッチ分割数変更

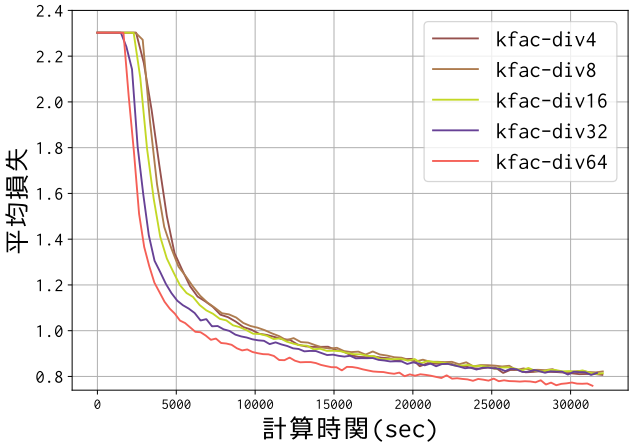

図 7　ミニバッチ分割数変更実験 (計算時間比較)

大きさ 16384 のミニバッチを 4，8，16，32，64 分割した際の平均損失の減少傾向を比較した (図 6，7)．横軸は図 6 では反復数，図 7 では計算時間であり，縦軸は平均損失である．ラベルは kfac-div[分割数] である．

反復数で比較した場合，分割数を変化させても平均損失の減少傾向は変わらなかった．計算時間で比較した場合，分割数を大きくすることで平均損失の減少速度は速くなった．一般に分割数を大きくすると 1 つの grad ノードの計算量は減少し，全体の通信量は増加する．分散学習をする際はこれを考慮して分割数を決めるべきであり，今回用いたニューラルネットワークと学習データでは更に大きい分割数でも高速化できる可能性がある．

4. おわりに

本研究ではパラメータ W の数が比較的少ないニューラルネットワークを用いて K-FAC を用いた分散学習の有効性と分散計算のための設定を確かめることができた．

一方でさらなる K-FAC を用いた分散学習の調査として以下のような課題がある．

より多くのパラメータ W をもつニューラルネットワークでの実験

AlexNet[8], VGG[10], GoogLeNet[9], ResNet[11] と言った今回実験に用いたニューラルネットワークより多くのパラメータ W をもつニューラルネットワークを K-FAC を用いて学習した場合の精度と収束性を調査する．本研究より十分な頻度で inv 計算を行うために必要な inv ノードの数はニューラルネットワークのパラメータ W の要素数によって決めるものと考えられるため，これらのニューラルネットワークでどれほどの inv ノードが必要になるかを検証する．また，大きなニューラルネットワークでの分散学習ではパラメータを共有する際の通信コストが大きくなるので，通信時間と計算時間を総合的に考えた場合の最適な分散計算のための設定を調査する必要がある．

より大きな学習セットでの実験

最適なミニバッチサイズは学習データのクラス分類数にも依存する．本研究では 10 クラス分類の CIFAR-10 を用いたが，ILSVRC12[*4]で用いられた 1000 クラス分類用のデータセットなどより多くのクラス分類を持つ学習データを用いた場合の最適なミニバッチサイズを調査し，それに対する最適な分散設定を調査する必要がある．

計算機に合わせた実装の改良

本研究では全ノードが gRPC を用いて通信を行った．しかし 1 ノード内に複数 GPU を持つ計算機を用いた場合では NVIDIA NVLink[*5]などの高速な通信手段を備えている場合があり，これを有効に活用することでさらなる高速化が期待できる．

K-FAC の分散学習を考えると，複数 GPU を持つノードでは例えば grad ノードと inv ノードの掛け持ちをさせることができ，掛け持ちのさせ方によって通信時間に変化が見られると考えられる．

謝辞　本研究は JST，CREST の支援を受けたものである．

参考文献

[1] Barret Zoph , Vijay Vasudevan, Jonathan Shlens, Quoc V. Le: *Learning Transferable Architectures for Scalable Image Recognition*, arXiv 1707.07012
[2] David E. Rumelhart, Geoffrey E. Hinton , Ronald J. Williams: *Learning representations by back-propagating errors*, Nature 323
[3] Amari Shun-ichi : *Natural Gradient Works Efficiently in Learning*, Neural Computation 1998
[4] James Martens ,Roger Grosse : *Optimizing Neural Networks with Kronecker-factored Approximate Curvature*, ICML 2015
[5] Jimmy Ba, Roger Grosse, James Martens : *Distributed Second-Order Optimization using Kronecker-Factored Approximations*, ICLR 2017
[6] Roger Grosse, Ruslan Salakhudinov : *Scaling up Natural Gradient by Sparsely Factorizing the Inverse Fisher Matrix*, PMLR 2015
[7] Nicolas Le Roux, Pierre-Antoine Manzagol, Yoshua Bengio : *Topmoumoute online natural gradient algorithm*, NIPS 2007
[8] Alex Krizhevsky and Ilya Sutskever and Geoffrey E. Hinton : *ImageNet Classification with Deep Convolutional Neural Networks*, NIPS 2012
[9] Christian Szegedy ,Wei Liu ,Yangqing Jia ,Pierre Sermanet ,Scott Reed ,Dragomir Anguelov ,Dumitru Erhan ,Vincent Vanhoucke ,Andrew Rabinovich : *Going Deeper with Convolutions*, CVPR 2015
[10] Simonyan, K. and Zisserman, A.: *Very Deep Convolutional Networks for Large-Scale Image Recognition*. CoRR 2014
[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun:

*4 http://www.image-net.org/challenges/LSVRC
*5 http://www.nvidia.co.jp/object/nvlink-jp.html

Deep Residual Learning for Image Recognition arXiv 1512.03385

[12] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, Bor-Yiing Su : *Scaling Distributed Machine Learning with the Parameter Server*, OSDI'14

分散計算のための設定

今回実験に使用した計算機は表 1 の 2 種類である．

計算機	略号	項目	型番
TSUBAME3.0[*4]	T	CPU	Intel Xeon E5-2680 v4[*5]
		GPU	NVIDIA Tesla P100[*6]
reki	R	CPU	Intel Xeon E3-1220 v3 [*7]
		GPU	NVIDIA GeForce GTX 1080 [*8]

表 1 使用計算機

実験は次に示す設定で行った．

実験	ラベル	#p	#g	#i	#b	計算機
1	kfac-b128	1	1	1	128	R
	kfac-b256	1	1	1	256	R
	kfac-b512	1	1	1	512	R
	kfac-b1024	1	2	1	512	R
	kfac-b2048	1	4	1	512	T
	kfac-b4096	2	8	1	512	T
	kfac-b8192	2	16	1	512	T
	kfac-b16384	4	32	1	512	T
	sgd-b512	1	1	-	512	R
	sgd-b2048	1	4	-	512	R
	sgd-b8192	2	8	-	1024	R
2	kfac-inv1	1	3	1	2048	T
	kfac-inv2	1	3	2	2048	T
	kfac-inv3	1	3	3	2048	T
	kfac-inv4	1	3	4	2048	T
3	kfac-div4	1	4	1	4096	T
	kfac-div8	2	8	1	2048	T
	kfac-div16	2	16	1	1024	T
	kfac-div32	4	32	1	512	T
	kfac-div64	7	64	1	256	T

表 2 #p，#g，#i，#b はそれぞれ ps ノード数，grad ノード数，inv ノード数，1 ノードが計算するミニバッチサイズを表す．

[*4] http://www.gsic.titech.ac.jp/tsubame3
[*5] https://ark.intel.com/ja/products/91754/Intel-Xeon-Processor-E5-2680-v4-35M-Cache-2_40-GHz
[*6] http://www.nvidia.co.jp/object/tesla-p100-jp.html
[*7] https://ark.intel.com/ja/products/75052/Intel-Xeon-Processor-E3-1220-v3-8M-Cache-3_10-GHz
[*8] http://www.nvidia.co.jp/graphics-cards/geforce/pascal/jp/gtx-1080

付　　録

A.1 クロネッカー積

$A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m' \times n'}$ に対するクロネッカー積は

$$A \otimes B = \begin{bmatrix} A_{1,1}B & A_{1,2}B & \cdots & A_{1,n}B \\ A_{2,1}B & A_{2,2}B & \cdots & A_{2,n}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1}B & A_{m,2}B & \cdots & A_{m,n}B \end{bmatrix} \quad \text{(A.1)}$$

と定義され，その大きさは $mm' \times nn'$ である．