Pip-SGD: 分散型ディープネットトレーニングのための分散型パイプライン SGD フレームワーク

Youjie Li†, Mingchao Yu\*, Songze Li\*, Salman Avestimehr\*,

Nam Sung Kim†, Alexander Schwing†.

イリノイ大学アーバナ・シャンペーン校 †University of Illinois at Urbana-Champaign 南カリフォルニア大学

# アブストラクト

ディープネットの分散学習は，メモリ消費量や計算量など，今日のコンピューティングの課題に対処するための重要な手法です．古典的な分散アプローチは，同期または非同期に関わらず，パラメータサーバアーキテクチャに基づいています．すなわち，ワーカーノードが勾配を計算し，

それがパラメータサーバに伝達され，更新されたパラメータが返されます．最近では、AllReduce を用いた分散学習も人気があります。これらの操作の多くは魅力的に見えますが、壁時計のようなトレーニング時間の改善についてはほとんど報告されていません。本論文では、AllReduce ベースのセットアップを慎重に分析し、ネットワークのレイテンシー、バンド幅、クラスタサイズ、計算時間を含むタイミングモデルを提案し、2 の幅を持つパイプライン化されたトレーニングが、同期トレーニングと非同期トレーニングの両方の長所を兼ね備えていることを実証しました。

具体的には、4 ノードの GPU クラスタで構成されたセットアップにおいて、従来のアプローチと比較して、ウォールクロックタイムのトレーニングが最大 5*.*4 倍向上することを示しました。

# はじめに

ディープネット[25, 3]は，コンピュータビジョン，自然言語処理，計算生物学，ロボット工学などの分野で広く用いられている．深層ネットは，さまざまなデータセットから階層的に抽象化された表現を自動的に抽出することで，領域やタスクを超えて素晴らしい成果を上げている．人気の高まりは 2010 年代に集中し、ImageNet データセットで素晴らしい結果が実証されました[22, 42]。それ以降，深層ネットはより多くのタスクに応用されている．代表的な例としては，場所の認識[53]，Atari社のゲームのプレイ[34, 35]，囲碁 [45]などがある．これらの手法に共通しているのは，深層ネットの多くの層に燃料を供給するために大規模なデータセットを使用することである．

重要なのは、ここ数年で、層の数、より一般的には計算ツリーの深さが、LeNet [26]の数層から数 100〜1000 層にまで大幅に増加していることです[14, 24]。計算グラフの複雑化に伴い、学習時間が増加し、処理されるデータ量も増加しています。従来、GPU などのアクセラレータを使用しても、計算性能の向上が必要な処理に追いつかないことがありました。

そのため、アクセラレータ以外にも、複数のコンピュータで計算を行う並列化が盛んに行われています。しかし、ネットワークインターフェースの帯域が限られている中で、計算ノード間で大量のデータをやり取りするためには、頻繁に通信を行う必要があります。このため、学習時間のかなりの部分がデータ通信に費やされることになり、並列化のメリットが大きく損なわれてしまいます。また，アクセラレータ[16, 7, 38, 52, 48, 49, 44] を適用すると，通信時間はそのままで計算時間が短縮されるため，通信にかかる時間の割合はさらに増加します．

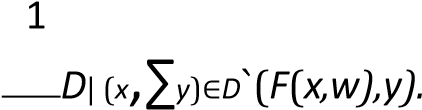
32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

マシン間の並列化を利用するために，人気の高い MapReduce パラダイムを始めとする様々なアプローチが開発されている[9, 51, 19, 37]．その利点にもかかわらず，ディープネットの通信負荷の高い学習は，パラメータサーバアーキテクチャ[28, 27, 15]に依存したカスタム実装[8, 6, 36, 20]に基づいていることが多く，中央のサーバがワーカーからの勾配を集約し，更新された重みを同期または非同期に分配する．最近の研究では，ノード間でグローバルな同期を行う分散型アーキテクチャの使用が提案されている[12, 33]．しかし，上述の手法に共通して，分散型ディープネットの学習のタイミング解析に関する報告はほとんどない．

本論文では、通信と計算の間の壁クロック時間のトレードオフを分析します。この目的のために、レイテンシー、クラスタサイズ、ネットワークバンドウィズ、モデルサイズなどの一連のパラメータに基づいてトレーニング時間を評価するモデルを開発する。我々のモデルの結果に基づいて、パイプライン化されたトレーニングとバランスの取れた通信を備えたフレームワークである Pipe-SGD を開発し、[23, 15]の証明を調整することでその収束特性を示す。また、AllReduce ベースのフレームワークにどのような種類の圧縮を効率的に組み込むことができるかを示す。最後に、10GbE ネットワークを備えた 4 つのノードからなる GPU クラスタ上で、我々の提案するアプローチのスピードアップを評価し、精度を落とすことなく、従来の集中型および分散型のアプローチと比較して、ウォールクロックタイムのトレーニングを 3*.*2 倍から 5*.*4 倍に向上させることができた。

# 背景

深層ネットの一般的な学習：深層ネットの学習では，入力データ *x* が与えられたときの予測器 *F*(*x,w*)のパラメータ *w* を求めることになる．この目的のために，与えられたデータ *x* と現在の *w* に対する予測器の出力 *F*(*x,w*)を，グランドトゥルースのアノテーション *y* と比較する損失関数`(*F*(*x,w),y*)を最小化する．データセット *D* = {(*x,y*)}が与えられた場合、*w* を見つけることは形式的に次のように要約されます。

min *fD*(*w*) := | (1) *w*

式(1)で与えられた目的をパラメータ *w* に合わせて最適化することは，例えば*∂ fwD* を用いた勾配降下法では，予測値 *F*(*x,w*)とその評価が複雑になるだけでなく，困難を伴います。

導関数だけでなく、データセット｜*D*｜のサイズも重要です。その結果、確率的勾配降下法（SGD）が一般的な手法として登場しました。データセットのサブセット *B* を無作為にサンプリングし、これをミニバッチと呼ぶこともあります。データセット *D* 全体で勾配を計算するのではなく、ミニバッチのサンプルを使って勾配を近似する、つまり*∂* *f ∂ fwB* を仮定する。しかし、現在のデータセットや予測器では、1 台のマシンで勾配 *∂∂fwB* を計算することはまだ難しい。例えば、セマンティック・イメージ・セグメンテーションのトレーニングでは、20 サンプル以下のミニバッチ・サイズ｜*B*｜が一般的である

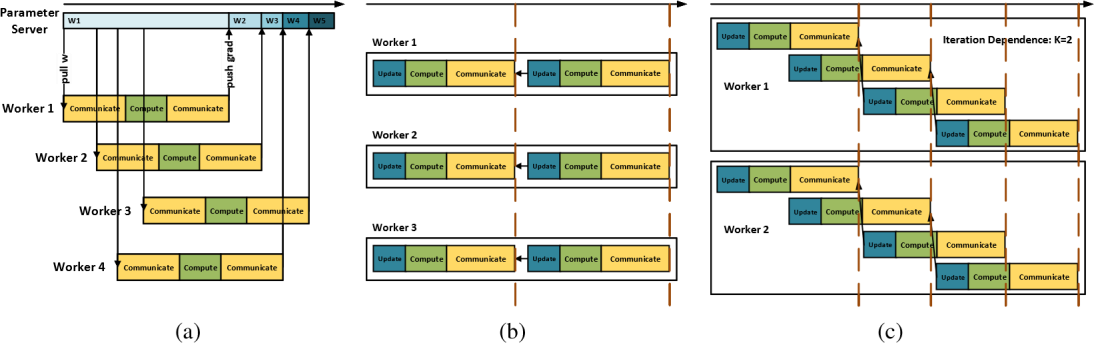
[5]。

ディープネットの分散学習：より大きなモデルを学習したり，ミニバッチサイズを大きくしたりするには，複数の計算ノード上での分散学習が用いられます [8, 15, 6, 27, 28, 36, 16]．分散学習を促進するための一般的なアーキテクチャとして，パラメータサーバフレームワークがあります[15, 27, 28]．パラメータサーバは，現在のパラメータのコピーを保持し，ワーカーノードのグループと通信します．ワーカーノードは，それぞれ小さなミニバッチを操作して，取得したパラメータ *w* に基づいてローカルグラデーションを計算します．パラメータサーバは，すべてまたは一部の勾配を取得すると，負の勾配の方向を用いてパラメータを更新し，その後，最新の値をワーカーと共有する．

各ワーカーが独立してサーバーから *w* を取得し、自分のローカルグラデーションを計算し、結果をプッシュバックする非同期更新が可能で、図 1(a)に示されています。非同期のため、最小限の同期オーバーヘッドは勾配の陳腐化と引き換えになります。遅延ステップの数を制限する遅延制御の方法が存在する[15]。しかし，陳腐化した勾配は学習速度を著しく低下させる可能性があるので注意が必要である．

重要なのは、これらのフレームワークはすべて、通信のボトルネックとなる集中型の計算トポロジーに基づいており、クラスタサイズが大きくなるにつれて学習時間が長くなることです。また、ネットワークの混雑により、勾配の押し付け、更新、*W* の引き抜きにかかる時間は、クラスタサイズに比例します。

そのため、最近では、分散型トレーニングフレームワークが、同期型と非同期型の両方で人気を博しています[30, 31]。しかし、これらのアプローチは、分散型ワーカーが完全に同期型（図1(b)のように）または完全に非同期型であることを前提としており、反復ごとに長い実行時間に対処するか、制御できない勾配の陳腐化を支払う必要があります。 タイムタイムタイム

 図 1：異なる分散学習フレームワークの比較。(a)非同期学習のパラメータサーバ、(b)分散型同期学習、(c)分散型パイプライン学習。

分散学習における圧縮。モデルサイズが大きくなり，クラスタサイズが大きくなると，分散学習システムの通信オーバーヘッドが学習時間を支配し，例えば，高速ネットワーク環境でも 80 ∼ 90%に達する[29, 10]．通信時間を短縮するために，近年，様々な圧縮アルゴリズムが提案されている[43, 46, 11, 4, 50, 33, 2]．その中には，スカラ量子化によって通信する勾配の精度を 1 ビットに落とすことに焦点を当てたものや，転送する勾配の量を減らすことに焦点を当てたものがある．しかし，ほとんどの圧縮技術は，壁時計型の学習時間を報告することなく，高い圧縮率や精度の低下の少なさを達成することにのみ重点を置いている．

実際には、通信プロセスの知識がない状態での圧縮は、通常、逆効果であり[29]、すなわち、総学習時間がしばしば増加します。これは、AllReduce が、後述の Sec.3.2 で述べるように、クラスタサイズに線形な最悪の複雑さで、転送された勾配を繰り返し圧縮・解凍する必要があるマルチステップ・アルゴリズムであることに起因する。

# 分散型パイプラインドストキャスティック勾配降下法

概要前述の問題（中央サーバのネットワーク輻輳、同期学習の長い実行時間、非同期学習の陳腐な勾配）を解決するために、我々は図 1（c）に示す新しい分散型学習フレームワーク「Pipe-SGD」を提案する。このフレームワークは、AllReduce によってノード間の通信をバランスさせ、ローカルトレーニングの反復をパイプライン化することで、通信時間を隠します。

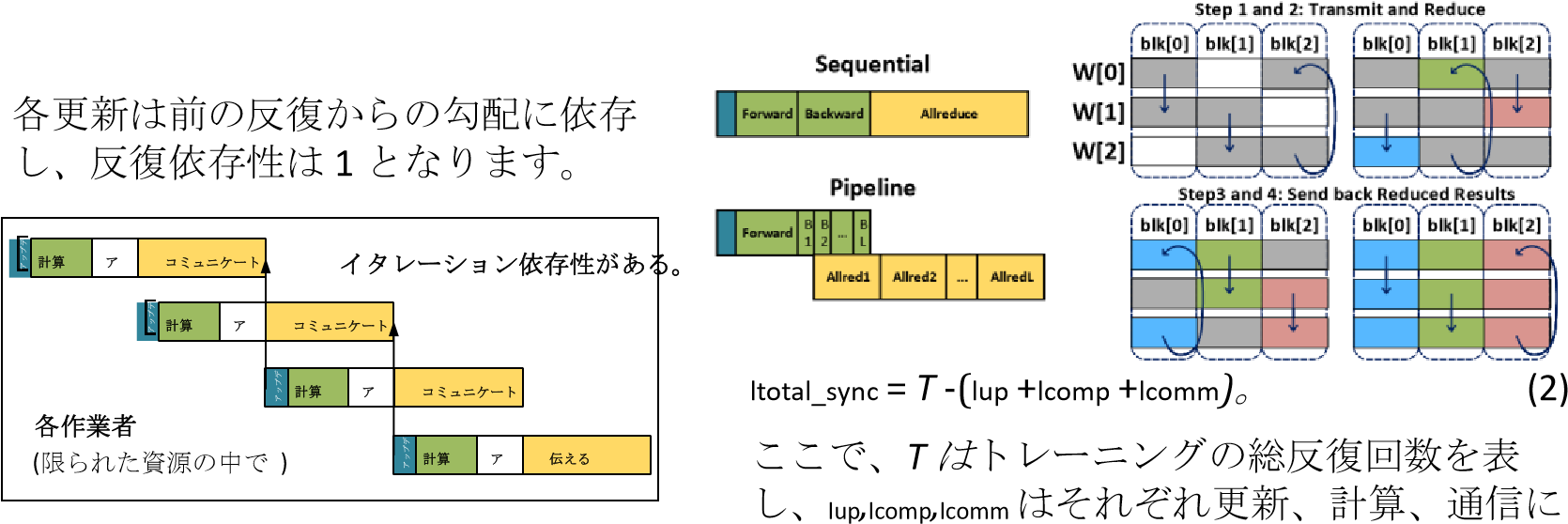
我々は、様々な通信アプローチを用いて、異なるリソース条件下での壁時計の列車時間のタイミングモデルを解析することにより、Pipe-SGD を開発しました。提案した Pipe-SGD は、勾配更新が 1 反復だけ遅れ、各反復にかかる時間がワーカー上のローカルな計算に支配される場合に最適であることを発見した。さらに、非可逆圧縮により、精度に影響を与えることなく通信時間をさらに短縮できることがわかった。

局所的なパイプライントレーニング、バランスのとれた通信、圧縮により、通信時間はもはやクリティカルパスの一部ではなく、すなわち計算によって完全にマスクされ、クラスタサイズが大きくなるにつれてエンドツーエンドのトレーニング時間が線形に高速化されます。最後に、[23, 15]の証明を調整することで、凸型および強凸型の目的に対する Pipe-SGD の収束性を証明する。

3.1 タイミングモデルと分散型 **Pipe-SGD**

タイミングモデルトレーニングのウォールクロック実行時間を解析するために、分散型同期 SGD に基づくタイミングモデルを提案します。各学習反復は、モデルの更新、勾配

の計算、および勾配の通信という 3 つの主要な段階で構成されます。古典的な同期

SGD(図 1(b))では、ワーカーの局所的

な反復を順次実行します。すなわち、

かかる時間を表します。同期式 SGD では、すべてのステージの実行時間の合計に依存するため、エンドツーエンドのトレーニング時間が長くなることが明らかになりました。

(a )(b )(c)のような

図 2：Pipe-SGD のタイミングモデル：(a)各ワーカーが限られたリソースを持つ場合、(b)シーケンシャル vs.パイプラインによるグラデーション通信、(c)グラデーション通信の一例。

Ring-AllReduce。

一方、Pipe-SGD では、反復依存性を *K* に緩和しています。つまり、各更新は、*K* 番目の最後の反復の勾配にのみ依存します。これにより、図 1(c)に示すように、グローバルに同期した通信を維持しながら、隣り合う反復間のインターリーブが可能になります。計算資源（CPU、GPU、その他のアクセラレータ）と通信資源（通信リンク）の両方が無制限または数・帯域幅ともに豊富であるという理想的な条件を仮定した場合、Pipe-SGD の総実行時間は

ltotal\_pipe = *T/K* -(lup +lcomp +lcomm*)*。(3) ここで、*K* は反復依存性または勾配の陳腐性を表します。しかし、今日の分散システムでは、各ワーカーノード上の計算資源と通信資源の両方が厳しく制限されているため、理想的な資源の仮定は実際には成り立ちません。その結果、分散学習のタイミングモデルは、図 2(a)に示すように、通信または計算のどちらかのリソースに縛られたものとなり、すなわち、総実行時間は

ltotal\_pipe = *T* -max(lup +lcomp*,*lcomm*),* (4) ここで、総実行時間は、*K* に関係なく（*K*≧2 の場合）、計算資源と通信資源のどちらかだけで決まります。また、勾配の更新は常に(*K* -1)反復分だけ遅れるため、*K > 2* を増加させても害にしかならず、すなわち、リソースが限られた Pipe-SGD の最適値は K = *2* となります。つまり、非同期更新で達成可能な最小のスタリングになります。また、分散型学習システムでは、一般的に計算量に制限のある設定、すなわち、lup +lcomp *>* lcomm が望ましいとされています。これを実現するための圧縮技術については、項 3.2 で説明します。パイプライン化されたイテレーションの実行に加えて、各イテレーション内のパイプライン化されたグラデーション通信についても解析し、トレインタイムの短縮を図っています。グラデーションの計算、すなわちバックワードパスとグラデーションの通信は、多くの場合、厳密に連続して実行されます（図 2（b）参照）。しかし、パイプライン化されたグラジエント通信、すなわち、グラジエントを計算した直後に通信を行うことは可能です。ここでも資源が限られていると仮定し、図2(b)で逐次グラデーション通信とパイプライン化されたグラデーション通信を比較します。

この 2 つのアプローチの詳細なタイミングを分析するために、通信のタイミングモデル [47]を使用する。勾配の通信は AllReduce オペレーションであり、すべてのワーカーからの勾配ベクトルを集約し、要素ごとに和のリダクションを行い、その結果を全員に送り返す。実際には、基礎となるアルゴリズムはもっと複雑である[47]。例えば、最速の AllReduce アルゴリズムの 1 つである Ring-AllReduce は、バランスの取れた通信により、ワーカー間でまとめて勾配の集約を行う。図 2（c）に示すように、各ワーカーは勾配ベクトル全体のブロックのみを隣のワーカーに送信し、受信したブロックに対して和の削減を行う。この「送信とリダクション」は、各ワーカーで勾配ブロックが完全にリダクションされるまで、すべてのワーカーで並行して実行されます（ブロックごとに異なります）。その後、完全にリダクションされたブロックは、仮想リング上の残りのワーカーに送り返されます。このアプローチは、すべてのノードのネットワーク帯域幅を最適に利用します。

47]の Ring-AllReduce モデルを採用することで、限られたリソースを仮定した場合の、逐次勾配通信を行う Pipe-SGD の総実行時間を以下のように求めることができます。

トータルパイプバック

、

2

（

*P*

アルゴリズム

1

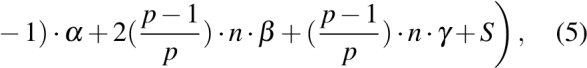
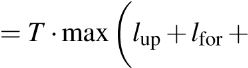
：各ワーカーの分散型

Pipe

-

SGD

学習アルゴリズム。



1: 同じモデル *w*[0]、学習率 *γ*、反復依存度 *K*、反復回数 *T* で初期化する 2: for *t* = 1*,...,T* do

3:イテレーション[*t* -*K*]で圧縮されたフォーマットで集約されたグラデーション gcsum ができるまで待つ。

4: 勾配を圧縮する gsum[*t* -*K*] ← Decompress(gcsum[*t* -*K*])

5: *w*[*t*]の更新 ← *w*[*t* -1]-*γ* -gsum[*t* -*K*].

6: 学習データのバッチ *B* を読み込む

7: 現在の損失 *fB* を計算するフォワードパス

8: バックワードパスで勾配を計算 glocal[*t] ←*

*∂∂wfB*[*t*].

9: グラデーションの圧縮 gclocal[*t*] ← Compress(glocal[*t*])

10: local gradient gclocal[*t*]を ready とする 11:

end for

各ワーカーの通信スレッドに

1: 繰り返し[1*-K,*1*-K*+1*,...,*0]の集約された勾配 gcsum をゼロとして初期化し、準備完了としてマークする 2: for *t* = 1*,...,T* do

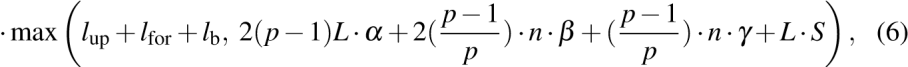
3:ローカルグラデーション gclocal[*t*]ができるまで待つ

4: AllReduce gcsum[*t*]←∑gclocal[*t*]です。

5: 集約された勾配 gcsum[*t*] を ready とする 6: end for

ここで、lfor と lback はフォワードパス時間とバックワードパス時間、*p* は作業者数、*α* はネットワークレイテンシー、*n* はモデルサイズ（バイト）、*β* はバイト転送時間、*γ* はバイトサムリダクション時間、*S* はグローバル同期時間を示す。

同様に、パイプライン化された勾配通信を用いた Pipe-SGD の総実行時間は、以下のように求められます。

ltotal\_pipe\_p = *T* 

ここで、*L* はグラデーションセグメントの数を表し、lb は最初のセグメントが取るバックワードパスの時間を表します。

式(5)と式(6)に基づいて、次のことが言えます：パイプラインシステムが通信結合のままであれば、パイプラインによるグラデーション通信よりも逐次的なグラデーション通信の方が好ましい（式(5)は正の *L* のため式(6)よりも小さい）。実際には、大規模なモデルの分散学習は、しばしば通信拘束され、逐次交換が最良の選択肢となります。

要約すると、我々のタイミングモデルに基づいて、次のことがわかります。*K* = 2 の場合、

**Pipe-SGD** が最適であること、システムが（圧縮後に）計算量に拘束されること、逐次勾配通信が使用されること。なお、我々のモデルは Ring-AllReduce に基づいて導き出されたものですが、この結論は、再帰的 2 倍化、再帰的半減と 2 倍化、ペアワイズ交換など、他の AllReduce アルゴリズムにも当てはまります。[47].

分散型パイプライン SGD: タイミングモデルを参考に、図 1(c)に示す分散型 Pipe-SGD フレームワークを開発しました。このフレームワークでは、ワーカー上の隣接する学習反復が *K* = 2 の幅でインターリーブされますが、各反復内の実行は厳密に逐次的に行われます。分散化されたワーカーは、各反復の後、勾配通信の同期をとりながら、パイプライン化されたトレーニングを並行して実行します。我々のフレームワークの同期性により、勾配の更新は常に *K* -1 反復分遅れるため、無秩序な陳腐化ではなく、決定論的な陳腐化が実施されます。我々の最適な設定では、遅延更新の反復回数は、従来の非同期パ

ラメータサーバートレーニング[15, 31, 1]の *O*(*p*)（*p* はクラスタサイズ）と比較して、1 となります。重要なのは、我々のフレームワークは、非同期アプローチの利点である、エンドツーエンドの実行時間を短縮するためにトレーニングの反復をインターリーブすることができるという点です。また、パラメータサーバのアーキテクチャとは異なり、我々はヘッドノードを混雑させません。代わりに、我々のケースでは、すべてのワーカーはバランスのとれた方法で勾配の一部を集約することにのみ責任を負うので、通信と集約の操作時間ははるかにスケーラブルになります。

より正式には、Alg.1 の各ワーカーに対する我々の実装のアルゴリズム構造を概説します。

1.具体的には、各ワーカーは、計算用と通信用の 2 つのスレッドを持ち、前者のスレッドは、*K* 番目の最後の反復の集約された勾配を消費し、通信されるべきローカル勾配を生成し、後者のスレッドは、ローカル勾配を交換し、前者のスレッドによって消費される集約された結果をバッファリングします。

1

2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ring-AllReduce** におけるブロック転送タイムライ | | | | | |
| .COMM |  | | |  |  |
| .REDUCE | パイプライン化された **Ring-AllReduce** | | | |
| .COMM |  |  |  |  |
| .REDUCE | | | |

におけるブロック転



**Ring**

**-**

**AllReduce + Light Compression**



1



.COMM



.REDUCE



2



.COMM



1



.REDUCE



2



デコンプ



パイプライン化された

**Ring**

**-**

**AllReduce + Light Compression**



サム



コンプ



タイムライ

1

2

(a) (b)

図 3：AllReduce におけるパイプライン化。(a)ネイティブな Ring-AllReduce とパイプライン化された Ring-AllReduce でのブロック転送、(b)軽量圧縮によるブロック転送。

3.2 パイプ**-SGD**の圧縮特性

通信時間をさらに短縮するために、我々は非可逆圧縮を分散型 Pipe-SGD フレームワークに統合した。従来のパラメータサーバや、ネットワーク上でパラメータを転送する最近の分散型フレームワーク[8, 6, 27, 28, 36, 16, 15, 31, 30]とは異なり、我々のアプローチでは勾配のみを通信します。そして、勾配はモデルパラメータよりも非可逆圧縮に対してはるかに耐性があることを経験的に確認しました。反復ごとにパラメータの精度を下げると、学習したモデルの最終的な精度が直接損なわれるため、これは直感的に理解できます。重要なのは、Sec.3.1 で述べたように、通信オーバーヘッドの圧縮が Pipe-SGD の最適な設定に寄与することです。一旦、Pipe-SGD が完全に計算に拘束されてしまえば、クラスタサイズの増加に伴い、エンドツーエンドのトレーニング時間を線形に高速化することができます。分析的には、式(4)で示されるタイミングモデルを使ってスケーリング効率を導き出すことで、この観察結果を示します。次のように仮定します。

2) *p* 人のワーカーを持つ Pipe-SGD クラスタでは、シングルノードと同じバッチサイズを各ワーカーに使用する[12]。3) シングルノードと Pipe-SGD は、データセットに対して同じエ

ポックで学習する。2)と 3)から、Pipe-SGD に必要な反復回数の合計は Tsingle*/p* であることがわかります。

Pipe-SGD は、同じ数のサンプルを学習しながら、バッチサイズが *p* 倍になります。このことから、Pipe-SGD のスケーリング効率 *SE* は以下のように求められます。

シングル・シングル・シングルシングル、シングル、シングル ツィングル

実際のスピードアップ ltotal\_pipe

最大

(

lup

+

lcomp

*,*

lcomm

)

-

*p*

lup

+

lcomp

SE ==== *.* (7)

理想的なスピードアップ  *pp* max(lup +lcomp*,* lcomm)

このように、我々のシステムが圧縮された通信で計算に拘束されるようになれば、PipeSGD はクラスタの規模に応じて、つまり*SE*=1 に応じて、リニアなスピードアップを達成できることを示しました。

Ring-AllReduce の適用性を維持するために、我々は 2 つの単純な圧縮アプローチを選択した：トランケーションとスカラ量子化。トランケーションでは、各グラデーションの浮動小数点値の下位の仮数ビットを落とします。スカラー量子化では、各勾配値を限られたビット数の整数に離散化し、その量子化範囲は勾配ベクトルの最大要素によって決定される。これらの圧縮手法は単純であるため、簡単に並列化することができ、オーバーヘッドを最小限に抑えることができます。

圧縮自体は計算量が多く、導入された計算オーバーヘッドが圧縮通信の利点を上回ることがあることに注意してください。特に、AllReduce ベースの通信では、データの転送と縮小のために複数のステップを実行することを考えると（図 2（c）参照）、圧縮と解凍を繰り返し呼び出すことが必要になります。そのため、提案されている多くの複雑な圧縮技術[43, 46, 11, 4, 50, 33]は、通信に最適な AllReduce の設定では失敗することが多く、結果としてウォールクロック時間が長くなってしまいます。これらの理由から、AllReduce に組み込まれた圧縮は、浮動小数点の切り捨てや我々の要素ごとの量子化のように、軽く、速く、並列化が容易でなければならない。

実際、*AllReduce* 内でのパイプライン化は、複雑な圧縮の重いオーバーヘッドを軽減するのに役立ちます。

しかし、その利点はまだ限られているかもしれません。PipeSGD のように学習反復をパイプライン化するのではなく、図 3(a)に示すように、*AllReduce* 内でパイプライン化することで、各 AllReduce プロセス内で勾配の伝達とリダクションを行うことができます。図 3 （a）のように、各 AllReduce 処理の中でグラデーションの通信とリダクションをパイプライン化することで、通信時間がリダクション時間よりも長くなり、リダクション時間を短縮することができます。圧縮が使用されると（図 3（b）のように）、2 段階のパイプラインは（解凍、合計、圧縮）と（圧縮された通信）になり、軽い圧縮オーバーヘッドを完全に隠すことができます。複雑な圧縮もパイプライン化された AllReduce の恩恵を受

ける可能性がありますが、複雑な圧縮にかかる時間が通信時間を上回ることが多いため、改善は限定的です。例えば、パイプライン化された AllReduce に[50]を実装したところ、圧縮オーバーヘッドは、圧縮前の通信時間の 1*.*6 ∼ 2*.*3 倍、圧縮後の通信時間の 25*.*6 ∼ 36*.*8 倍であり、この場合、重いオーバーヘッドをマスクすることはできません。完全なマスキングのためには、圧縮時のオーバーヘッドが圧縮後の通信時間よりも小さいことが必要である。残りの部分では、ネイティブな AllReduce による軽い圧縮（切り捨て／量子化）のみを考慮する。

3.3コンバージェンス

Pipe-SGD の収束性を証明するために、パラメータサーバベースの非同期トレーニング[15,

23]からの導出を適用します。凸型の目標に対する Pipe-SGD の収束率は、以下のようになります。

SGD による *qK* は、8*FL T* となります。ここで、*K* = 2、*F* と *L* はそれぞれ勾配距離とリプシッツ連続性を表す定数です。また、強く凸な関数に対する Pipe-SGD の収束を示

すことができ、勾配降下の速度が *O*(log *TT* )であることがわかります。これらの率は[15,

23]と一致している。ページ数の制限があるため、詳細は補足資料に譲る。

# 実験評価

本節では，3 つのデータセットを用いた 4 つのベンチマークにおいて，我々のアプローチの有効性を実証する．MNIST [26]，CIFAR100 [21]，ImageNet [42]の 3 つのデータセットを用いて，4 つのベンチマークで我々のアプローチの有効性を実証する．これらのデータセットの特徴を簡単に説明した後，測定基準と設定方法を説明し，最後に実験結果と分析を示す．データセットと Deep Net のアーキテクチャ

* MNISTMNIST データセットは，6 万枚のトレーニング画像と 1 万枚のテスト画像で構成されており，それぞれの画像には 10 種類の数字が表示されています．画像は 28×28 ピクセルの大きさで，数字は画像の中央に配置されている．我々は古典的な 3 層パーセプトロンである MNIST-MLP を用い、両方の隠れ層を 500 次元とし、グローバルバッチサイズを 100 とした。
* ImageNet です。今回の実験では，ImageNet チャレンジの 1,281,167 個の学習例と 50,000 個の検証例を使用しました．各例は，256×256 ピクセルのカラー画像で，1000 個のクラスのうちの 1 つに属しています．古典的な AlexNet [22]と ResNet [14]を使用し，どちらもグローバルバッチサイズを 256 としました．
* CIFAR100: CIFAR100 データセットは，50,000 個の学習例と 10,000 個のテスト例から構成され，100 個のクラスがあります．このデータセットのベンチマークには，[32]のシンプルな AlexNet スタイルの CIFAR100 アーキテクチャが用いられています．このアーキテクチャは，3 つの畳み込み層と 2 つの完全連結層，そしてソフトマックス層から構成されています．詳細なパラメータは[32]を参照してください。重要なのは、この 5 層の CIFAR100-CNN を凸最適化ベンチマークである CIFAR100-Convex に適合させ、我々の収束の証明と一致させていることです。この凸型最適化は、最後の完全連結層のみを学習し、それ以前のすべての層のパラメータを固定することで実現しています。

メトリクスとセットアップエンド・ツー・エンドのトレーニング，すなわち，異なる設定で同じ回数の反復を行う場合のウォールクロックタイムを測定しています．各ベンチマークについて，エンド・ツー・エンドのトレーニング時間と詳細なタイミングの内訳を用いて，我々が提案したタイミングモデルを評価しました．実際の収束性を評価するために，テスト/検証の精度をトレーニング時間に対してプロットしました．また，テスト/検証セットでの最終的なトップ 1 精度も報告しています．設定には，GPU「Titan XP」[40]と Xeon CPU「E5-2640」 [17]で構成された 4 つのノードからなるクラスタを使用しました．また，従来の中央集権

的な設計をサポートするために，パラメータサーバとして追加のノードを採用しまし

た．すべてのノードは 10Gb イーサネットで接続されています．CUDA 8.0 [39]，MKL 2018 [18]，OpenMPI 2.0 [41]を用いて，C++による分散学習フレームワークを実装し，パラメータサーバ方式と Pipe-SGD 方式をサポートしています．結果と分析

同期 SGD を用いたパラメータサーバ（PS-Sync）、分散型同期 SGD（D-Sync）、Pipe-SGD という 3 つの異なるフレームワークの性能を評価した。また、D-Sync と Pipe-SGD における

AllReduce 通信には、16 ビットの切り捨て（T）と 8 ビットの量子化（Q）という圧縮スキームを適用した。評価結果は図 4 にまとめられており、最初の 2 列は収束性能を、3 列目は最終的な精度を示した詳細なタイミングブレークダウンを示している。

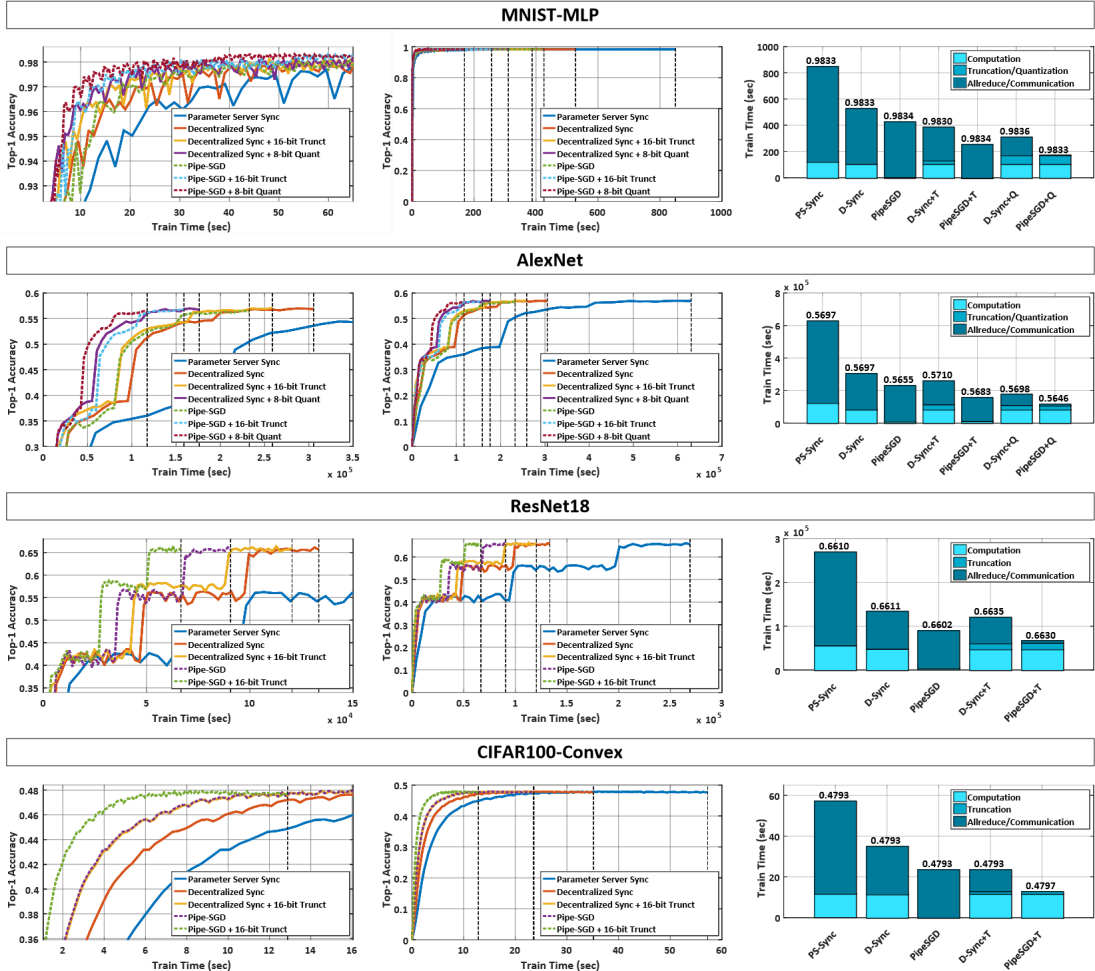


図 4：実験結果。各行は異なるベンチマークを示している．左側の 2 列は，テスト/検証の精度による収束とウォールクロックでのトレーニング時間の関係を示しており，1 列目は 2 列目の挿入図である．右端の列は，エンドツーエンド・トレーニングの詳細なタイミングを示しています．なお，テスト/検証セットの最終的なトップ 1 精度は，バーの上にラベル付けされています．

収束。図 4 から、分散型アプローチ、すなわちD-Sync と Pipe-SGD は、圧縮を行わなくてもパラメータサーバよりもはるかに速く収束し、Pipe-SGD はこれらのフレームワークの中で、特に圧縮を適用した場合に最も速く収束することがわかります。例えば、CIFAR100Convex の収束曲線を見ると、D-Sync は PS-Sync に比べて約 40%速く、Pipe-SGD は D-Sync に比べてさらに 37%速いことがわかります。この Pipe-SGD の優位性は、圧縮（この場合はトランケーション）によってさらに高まり、同じ圧縮方式の D-Sync よりもさらに 46%速い収束を示しています。このように、Pipe-SGD は大差をつけて勝利しています。

タイミングブレークダウン。図 4 より、集中型と分散型の比較では、非圧縮時の通信時間が 50％短縮されており、バランス型通信の有効性が示されています。圧縮を適用すると、さらに短縮されることがわかります。しかし、D-Sync の実際の改善は、圧縮のオーバーヘッドが D-Sync のクリティカルパスで支払われるため、切り捨てのための 2 倍、量子化のための 4 倍の圧縮係数を考慮すると理想的ではありません。これに対し、当社の Pipe-SGD は、パイプライン化されているため、このオーバーヘッドを計算と一緒に隠すことができます。

MNIST ベンチマークにおける「D-Sync+T」と「PipeSGD+T」の比較。量子化によって通信がさらに削減されると、システムは計算量に拘束されるようになり、Pipe-SGD は代わりに通信を隠すように切り替わり、Pipe-SGD の最適な設定に達します。この最適な設定は、 ResNet18 や CIFAR100-Convex など、通信時間が支配的でないモデルでは、最も単純な切り捨てによっても達成されます。その結果、D-Sync と比較して 2*.*0〜3*.*2 倍、PS-Sync と比較して 4*.*0〜5*.*4 倍の高速化を実現しました。なお、これらの高速化は、クラスタサイズを変えずに、同じクラスタ内で異なるアプローチを比較した結果です。

精度。Pipe-SGD の 1iteration staled update と非可逆圧縮の潜在的な欠点を考慮して、図 4 に示すように、end-to-end トレーニング後の最終テスト/検証の精度も評価しました。興味深いことに、最適な設定である「PipeSGD +T/Q」では、ベースラインの D-Sync と比較して、AlexNet のみがトップ 1 の精度を 0*.*005 低下させ、他のベンチマークはわずかに精度が向上していることがわかります。AlexNet や ResNet のような大規模な非凸モデルで最良の精度を得るために、[33]と同様のウォームアップスキームを採用しました。つまり、5 番目のエポックまでパイプライン学習をオンにせず、それ以前は初期段階での望ましくない勾配変化を避けるために D-Sync 学習に固執します。ウォームアップ期間はエポックの総数に比べてわずかなので、システムの性能はほとんどの場合、Pipe-SGD の恩恵を受けます。なお、小さいモデル、特に凸型のモデル（例：CIFAR100-Convex）では、ウォームアップは必要ありません。

# 関連作品

Li ら[27, 28]は，分散学習のためのパラメータサーバフレームワークを提案し，計算ノード間の通信コストを削減するために，ゼロでないパラメータ値のみの交換，インデックスリストのローカルキャッシュ，送信するメッセージのランダムスキップなどのいくつかのアプローチを提案した．Abadi ら[1]も、ディープネットの学習のためにモデルとデータの並列処理を組み込んだ集中型フレームワーク TensorFlow を提案しています。どちらの作品も、通信効率を向上させるために非同期設定をサポートしていますが、勾配更新の陳腐化を制御していません。Ho ら[15]は、SSP を提案しています。これも集中型の非同期フレームワークですが、グラデーションのスタールネスを制限しています。SSP の主なアイデアは、1）各ワーカーが自分の反復インデックスを持つ、2）最も遅いワーカーと最も速いワーカーが *S* 反復以内でなければならない、そうでなければ、最も遅いワーカーが追いつくまで、最も速いワーカーは待たされる、というものです。ただし、この境界 *S* は、パラメータサーバの陳腐な更新に直接適用されるのではなく、ワーカー間の反復ドリフトに適用されます。その結果、制限内の各ワーカーは、更新を非同期的にサーバーにコミットすることができるため、最後の勾配更新が大きくスタリングされてしまいます。最悪の場合、陳腐化の度合いはクラスタサイズの線形となります。

Lin ら[33]は、同期フレームワークの勾配集約法として AllReduce を採用していますが、ウォールクロック時間のメリットについてはほとんど報告されていません。特に、完全な同期設計では、すべてのワーカーの中で実行時間が最も長くなるという問題があります。

また、Lian らは AD-PSGD[31]を提案しています。これは、完全に非同期な方法で分散したワーカー上で SGD 処理を並列化するものです。ワーカーは完全に独立して動作し、訓練された重みを交換するために近隣のノードのセットと通信するだけです。つまり、各反復において各ワーカーのローカルモデルを置き換えるために近隣のモデルが平均化されます。しかし、この方法では、制御できない陳腐さがあり、実際には、クラスタサイズと各反復にかかる時間に応じて増加します。また、このような通信方法では、各ワーカーがローカルグラフの中心ノードとして動作する必要があり、ローカル通信のボトルネックとなります。その結果、各ワーカーは長い反復時間に悩まされ、重み更新の陳腐さをさらに高めることになります。Lian ら[31]は、自分たちのフレームワークと完全同期型のデザインを壁時計時間で比較していますが、ネットワークの速度がほぼ同じであれば、性能は同程度であることがわかります。

最近、独立した研究である[13]も、DNN 学習のための分散パイプラインシステムを提案した。これは、DNN のレイヤーを異なるマシンに分割し、最初のマシンに連続したミニバッチを注入することで、マシンの実行をパイプライン化するものです。この方法では、マシン間で通信されるのは一部の層の活性化と勾配だけなので、通信負荷が軽減されます。しかし，異なるマシン間で作業負荷のバランスをとるためには，複雑なメカニズム（プロファイリング，パーティショニングアルゴリズム，複製されたステージなど）が必要であり，そうしないと計算資源が空回りしてしまいます．さらに、[13]では、ステージ数に線形なウェイト更新のスタール性に悩まされる可能性があります。これは、モデルのパイプライン化の効果を制限し、スピードアップを妨げる要因となります。

# 結論

我々は、ネットワーク遅延、モデルサイズ、バイト転送時間などを考慮した、分散型ディープネットトレーニングのための厳密なタイミングモデルを開発しました。このタイミングモデルと、限られたネットワーク帯域などの現実的なリソースの仮定に基づき、スケーラビリティを評価し、計算時間や通信時間の高速化をマスクできるパイプライン型学習フレームワーク「Pipe-SGD」を開発しました。10Gb リンクで接続された 4 ノードの

GPU クラスターにおいて、提案手法の有効性を示しました。また、Pipe-SGD のウォールクロック時間を厳密に評価した結果、従来の手法と比較して最大 5*.*4 倍の改善を達成することができました。

# 感謝の気持ちを込めて

本研究は，NSF からの助成金（IIS 17-18221，CNS 17-05047，CNS 15-57244）に一部支えられています。

CCF-1763673 および CCF-1703575）。)本研究は、3M および IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR)の支援を受けています。また、この資料は、米国国防総省高等研究計画局（DARPA）の契約番号 HR001117C0053 の支援を受けた研究に一部基づいています。本資料に記載されている見解、意見、調査結果は著者のものであり、米国国防総省または米国政府の公式見解または政策を示すものと解釈されるべきではありません。

# リファレンス

1. M.Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard,

M.Kudlur、J.Levenberg、R.Monga、S.Moore、D.G.Murray、B.Steiner、P.A.Tucker、V.Vasudevan。

P.Warden, M. Wicke, Y. Yu, and X. Zhang.TensorFlow: A System for Large-Scale Machine Learning.In *OSDI*,

2016.

1. D.Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic.QSGD: Gradient Quantization and Encoding via

Communication-Efficient SGD.In *NIPS*, 2017.

1. Y.Bengio, A. Courville, and P. Vincent.表現学習。A Review and New Perspectives.*PAMI*, 2013.
2. C.-Y.C.Y.Chen, J.Choi, D.Brand, A.Agrawal, W.Zhang, and K.Gopalakrishnan.AdaComp : Adaptive Residual Gradient Compression for Data-Parallel Distributed Training.In *AAAI*, 2018.
3. L.-C.Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille.Deep Convolutional Nets and Fully

Connected CRFs によるセマンティック画像セグメンテーション.In *ICLR*, 2015.

1. T.Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman.Project Adam: Building an efficient and scalable deep learning training system.In *OSDI*, 2014.
2. H.Cui, H. Zhang, G. R. Ganger, P. B. Gibbons, and E. P. Xing.GeePS: Scalable Deep Learning on Distributed GPU with a GPU-Specialized Parameter Server.In *EuroSys*, 2016.
3. J.Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng.Large Scale Distributed Deep Networks.In *NIPS*, 2012.
4. J.Dean and S. Ghemawat.MapReduce:大規模クラスターでのシンプルなデータ処理。

*Communications of the ACM*, 2008.

1. N.Dryden, N. Maruyama, T. Moon, T. Benson, A. Yoo, M. Snir, and B. V. Essen.Aluminum:An Asynchronous, GPU-Aware Communication Library Optimized for Large-Scale Training of Deep Neural Networks on HPC Systems.In *MLHPC*, 2018.
2. N.Dryden, T. Moon, S. A. Jacobs, and B. V. Essen.Communication Quantization for Data-Parallel Training of Deep Neural Networks.In *MLHPC*, 2016.
3. P.Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He.Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.In *CVPR*, 2017.
4. A.Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, and P. B. Gibbons.PipeDream:Fast and Efficient Pipeline Parallel DNN Training.in *arXiv:1806.0337v1*, 2018.
5. K.He, X. Zhang, S. Ren, and J. Sun.画像認識のための深い残差学習．In *CVPR*, 2016.
6. Q.Q.Ho, J.Cipar, H.Cui, S.Lee, J.K.Kim, P.B.Gibbons, G.A.Gibson, G.R.Ganger, and E.P.Xing.Stale Synchronous Parallel Parameter Server を用いたより効果的な分散型 ML.in *NIPS*, 2013.
7. F.F. N. Iandola, K. Ashraf, M. W. Moskewicz, and K. Keutzer.FireCaffe:Compute Clusters 上でのディープニューラルネットワークトレーニングのほぼ直線的な高速化。In *CVPR*, 2016.
8. イ ン テ ル ・ コ ー ポ レ ー シ ョ ン Xeon CPU E5, https://www.intel.com/content/www/us/en/products/processors/xeon/e5processors.html, 2017.
9. インテル・コーポレーション Intel Math Kernel Library, https://software.intel.com/en-us/mkl, 2018.
10. M.Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly.Dryad:逐次的なビルディングブロックからの分散データ並列プログラム．*ACM SIGOP*, 2007.
11. H.Kim, J. Park, J. Jang, and S. Yoon.Deepspark:A spark-based distributed deep learning framework for commodity clusters. *arXiv:1602.08191 [cs*], 2016.
12. A.Krizhevsky.小さな画像から複数の層の特徴を学習する, 2009.
13. A.Krizhevsky, I. Sutskever, and G. E. Hinton.深層畳み込みニューラルネットワークによる画像分類.In

*NIPS*, 2012.

1. J.Langford, A. J. Smola, and M. Zinkevich.Slow Learners are Fast.In *NIPS*, 2009.
2. G.Larsson, M.Maire, and G.Shakhnarovich.FractalNet:残留物のない超深層ニューラルネットワーク。

In https://arxiv.org/abs/1605.07648, 2016.

1. Y.LeCun, Y. Bengio, and G. E. Hinton.深層学習です。*Nature*, 2015」に掲載されています。
2. Y.LeCun, L. Bottou, Y. Bengio, and P. Haffner.勾配ベースの学習の文書認識への応用．*IEEE*, 1998.
3. M.Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-

Y.Su.Parameter Server による分散機械学習のスケーリング。In *OSDI*, 2014.

1. M.Li, D. G. Andersen, A. J. Smola, and K. Yu.Parameter Server を用いた通信効率の良い分散機械学習。

In *NIPS*, 2014.

1. Y.Y.Li, J.Park, M.Alian, Y.Yuan, Z.Qu, P.Pan, R.Wang, A.G.Schwing, H.Esmaeilzadeh, and N.S.Kim.A NetworkCentric Hardware/Algorithm Co-Design to Accelerate Distributed Training of Deep Neural Networks.In *MICRO*, 2018.
2. X.Lian, C. Zhang, H. Zhang, C. Hsieh, W. Zhang, and J. Liu.Can Decentralized Algorithms Outperform Centralized Algorithms?A Case Study for Decentralized Parallel Stochastic Gradient Descent.In *NIPS*, 2017.
3. X.X.Lian, W.Zhang, C.Zhang, and J.Liu.Asynchronous Decentralized Parallel Stochastic Gradient Descent.in *arXiv:1710.06952v3*, 2018.
4. R.Liao, A. Schwing, R. Zemel, and R. Urtasun.Learning Deep Parsimonious Representations.In *NIPS*, 2016.
5. Y.Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally.Deep Gradient Compression:Reducing the Communication Bandwidth for Distributed Training.In *ICLR*, 2018.
6. V.Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller.Playing Atari with Deep Reinforcement Learning.In *NIPS Deep Learning Workshop*, 2013.
7. V.Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K.

Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra,

S. Legg, and D. Hassabis.深層強化学習による人間レベルの制御。*Nature*, 2015」に掲載されています。

1. P.Moritz, R. Nishihara, I. Stoica, and M. I. Jordan.SparkNet:Spark でディープネットワークをトレーニングする。In *ICLR*, 2016.
2. D.G. Murray, R. Isaacs F. McSherry, M. Isard, P. Barham, and M. Abadi.Naiad:A Timely Dataflow System.In *SOSP*, 2013.
3. Nvidia.GPU ベースのディープラーニング推論。A Performance and Power Analysis.In *Whitepaper*, 2015.[39] NVIDIA Corporation.NVIDIA CUDA C プログラミングガイド, 2010.
4. NVIDIA Corporation.TITAN Xp, https://www.nvidia.com/en-us/design-visualization/products/titan-xp/, 2017.
5. OpenMPI コミュニテ ィ OpenMPI:A High Performance Message Passing Library, https://www.openmpi.org/, 2017.
6. O.Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei.ImageNet Large Scale Visual Recognition Challenge.*IJCV*, 2015.
7. F.Seide, H. Fu, J. Droppo, G. Li, and D. Yu.1-Bit Stochastic Gradient Descent and Its Application to DataParallel Distributed Training of Speech DNNs.In *INTERSPEECH*, 2014.
8. H.Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Misra, and H. Esmaeilzadeh.From HighLevel Deep Neural Models to FPGAs.In *MICRO*, 2016.
9. D.Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser,

I.Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I.

Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis.Deep Neural Networks and

Tree Search で囲碁のゲームをマスターする。*Nature*, 2016」に掲載されています。

1. N.Strom.コモディティ GPU クラウドコンピューティングを用いたスケーラブルな分散型 DNN トレーニング.In *INTERSPEECH*, 2015.
2. R.Thakur, R. Rabenseifner, and W. Gropp.MPICH における集団通信操作の最適化。*IJHPCA*, 2005.
3. Q.Q. Wang, Y. Li, and P. Li.液体状態機械に基づくパターン認識を、FPGA 上で Firing-Activity Dependent Power Gating and Approximate Computing を用いて実現しました。In *ISCAS*, 2016.
4. Q.Q. Wang, Y. Li, B. Shao, S. Dey, and Peng Li.FPGA 上の近似演算を用いたエネルギー効率の高い並列ニューロモーフィック・アーキテクチャ.*Neurocomputing*, 2017.
5. W.Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li.TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning.In *NIPS*, 2017.
6. M.Zaharia, M. Chowdhury, Michael J. Franklin, S. Shenker, and I. Stoica.Spark:ワーキングセットによるクラスタコンピューティング。In *HotCloud*, 2010.
7. C.C.Zhang, P.Li, G.Sun, Y.Guan, B.Xiao, and J.Cong.Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks.In *FPGA*, 2015.
8. B.Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva.Learning Deep Features for Scene Recognition using Places Database.In *NIPS*, 2014.