Vol.2017-ARC-227 No.25
2017/7/27

可変データビット幅を持つDNNと そのアクセラレータアーキテクチャの検討

石井 潤[1,a)]　坂本 龍一[1]　近藤 正章[1]

概要：近年，DNN（Deep Neural Network）において，重みや入力値のビット幅を縮小させる量子化を行うことで，記憶領域と計算資源のコストを削減する研究が多く行われている．これまでネットワーク全体や層毎で量子化ビット幅を最適化する手法が提案されているが，各層内のニューロン毎に量子化ビット幅を最適化することについては，まだ検討されていない．本稿では，ネットワークの各層と層内のニューロン毎にビット幅を最適化する手法と，そのネットワークモデルを効率的に実行可能なハードウェア機構を検討する．また，予備評価として，ニューロン毎に個別の量子化をした場合のデータサイズと画像認識精度への影響についての予備評価を行う．MNIST ベンチマークの手書き文字認識を行う 3 層のニューラルネットワークでの評価の結果，ニューロン毎の量子化を適用することで層毎の量子化に比べても高い精度を達成しつつ，よりデータを圧縮できる可能性があることがわかった．

1. はじめに

深層ニューラルネットワーク (Deep Neural Network: DNN）は，多層のニューラルネットワークを用いた機械学習モデルであり，コンピュータビジョン [8]，音声認識 [11]，自然言語処理 [11] などの様々な分野でそれぞれ高い性能が報告されている．近年の DNN モデルは，認識精度を向上させるためにネットワークモデルが大規模になっており，必要な計算量やメモリ消費量が増加する傾向にある．

畳み込みニューラルネットワーク (Convolutional Neural Network: CNN) は，畳み込み層とプーリング層など，特に画像処理を指向した演算を行う層を含む DNN の一種であり，主に画像認識に使用される．CNN は，畳み込み層では計算量が多く CPU 演算処理がボトルネックとなり，また全結合層ではパラメータ数が多く，メモリ転送がボトルネックになる傾向があることが知られている．例えば，画像認識コンペティション ILSVRC2015 の勝者である ResNet-152[6] は，約 230MB のモデルサイズをもち，224x224 サイズの画像 1 枚を分類するための推論計算に約 113 億回の浮動小数点演算を実行する必要がある．

このような計算量およびメモリアクセス量の増大は，消費エネルギーの増加を招く．計算資源や電力資源が限られる組み込み機器において，大規模な CNN を実行する際にはその演算およびメモリアクセスに要する消費エネルギーの削減と，高い性能を両立することが課題となる．

これまで，組み込み機器向けの低消費電力な DNN 専用アクセラレータが多く提案されてきた．DaDianNao[1] は，チップ内に大容量の eDRAM を配置することで，オフチップアクセス回数を削減し，処理の高速化と低消費電力化を実現したアクセラレータである．Eyeriss[2] は，計算量の多い畳み込み層を高電力効率で処理することを狙ったアクセラレータである．演算ユニットを二次元アレイ状に配置し，データの再利用性や並列性を考慮してデータフローを最適化することで，畳み込み層での計算効率を高めるとともにオフチップメモリアクセスを削減し，高速化，低消費電力化を実現する．EIE[5] は，パラメータ数が多い全結合層におけるメモリアクセスのボトルネックを解決し，高電力効率を狙うアクセラレータである．圧縮したネットワークモデル [4] を使用することによりメモリアクセスを低減し，また圧縮に伴う複雑なデータ構造の処理を効率的に行うハードウェア機構を持つことで，CPU や GPU などの汎用ハードウェアよりも高速，低消費電力な処理を実現可能である．

一方で，学習済みネットワークを圧縮することにより，演算負荷や消費電力を削減する手法も多数提案されている．枝刈り [3] はシナプスやニューロンの重要度をもとに，重要度の低いものを削除することでネットワークモデルのパラメータ数を削減する手法である．Han らは，この枝刈り手法によって認識精度を落とさずにモデルサイズを 10%程度まで圧縮することに成功している [3]．前述の EIE では，枝刈りにより削除されたパラメータを 0 で表現し，Zero

[1] 東京大学 大学院情報理工学系研究科
Graduate School of Information Science and Technology, The University of Tokyo
[a)] ishii@hal.ipc.i.u-tokyo.ac.jp

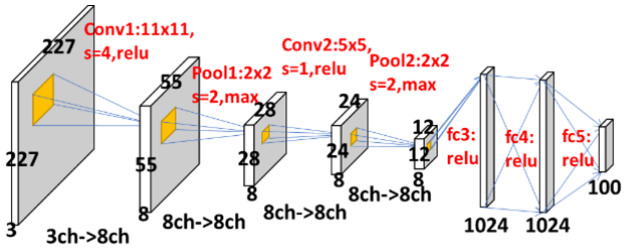

図 1 7 層畳込みニューラルネットワーク

skipping[5] によって計算を省略することで計算回数を低減している．

量子化 [16] は，単精度浮動小数点数などで表現された値を，より低精度な数値表現で近似することで，計算負荷やメモリ消費量の削減を行う手法である．DNN の計算では，推論時に必要な認識精度を得るための各データにはそれほど高い精度が必要でないことがわかっている．そこで，入力や重みの量子化ビット幅を削減することで，認識精度の低下を抑えながらデータあたりの計算に必要なハードウェア面積やメモリ消費量，あるいはメモリアクセスの際のデータ転送量を削減することができる．これまで量子化手法として，ネットワークモデル全体で同一の量子化ビット幅を用いるもの [16]，畳み込み層や全結合層などの層の種類毎に個別の量子化ビット幅を用いるもの [4]，さらには，各層で異なるビット幅で量子化を行うことで，認識精度を低下させずにさらにメモリ消費量を削減するもの [13] などが提案されている．また，各ビットを 2 のべき乗に対応する値として非線形に量子化を行う手法も提案されている [12][10]．

本稿では，さらに計算負荷やデータ転送量を削減するために，ニューロン毎に異なる量子化ビット幅を適用することを検討する．ニューロンに接続されるシナプスの重みの値のレンジは，各ニューロン毎に大きく異なるため，重みの値の分散が大きいニューロンの重みには多数のビット幅を，重みの値の分散が小さいニューロンの重みには少数のビットを割り当てるよう量子化することで，従来の量子化の手法に比べてさらにデータ量を圧縮できる (あるいは同じデータ量で精度を向上させることができる) と期待される．本稿では，ニューロン毎に個別の量子化を行ったネットワークモデルを効率的に実行できるハードウェア機構を提案する．またニューロン毎に個別の量子化を行った場合のデータサイズと画像認識の精度について予備評価を行う．

2. DNN と量子化手法

2.1 深層ニューラルネットワークの概要

本節では，深層ニューラルネットワーク，特に CNN の各層の計算について概要を述べる．CNN は，主に畳込み層，プーリング層，全結合層という 3 種類の層の組み合わせで構成される．それぞれの演算は式 (1)(2)，式 (3)，式 (4)(5) のように定義される．また，$f(\cdot)$ は活性化関数と呼ばれ，ReLU 関数 $f(x) = \max(0, x)$ や sigmoid 関数 $f(x) = 1/(1 + \mathrm{e}^{-1})$ がよく用いられる．

$$a_{n_o}[i,j] = \sum_{n_i}\sum_{p}\sum_{q} \omega_{n_i,n_o}[i,j]x[i+p,j+q] + b_{n_o} \quad (1)$$

$$y_{n_o}[i,j] = f(a_{n_o}[i,j]) \quad (2)$$

$$y[i,j] = \max_{p,q}(x[i+p,j+q]) \quad (3)$$

$$a[i] = \sum_{j} \omega[i,j]x[j] + b_i \quad (4)$$

$$y[i] = f(a[i]) \quad (5)$$

例として，図 1 に 7 層 CNN のネットワーク構造を示す．ここでは，消費電力の大きい畳み込み層と全結合層について主に説明する．なお，どちらも入力値 x に重み w をかけてバイアス b を足したものの総和を取る積和演算が主な演算となる．

畳み込み層では，式 (1) で表される畳み込み演算の結果を活性化関数にかけて出力を得る．三重の総和を計算するため，計算回数が大きくなりやすい傾向がある．227×227 の RGB 画像に 11×11 のフィルタをストライド幅が 5 で畳込み，8 枚の出力マップを得る畳込み層の演算を考えると，積和演算回数は 293 万回である．

全結合層では式 (4) で表される行列ベクトルの積和演算の結果を活性化関数にかけて出力を得る．入力サイズ j と出力サイズ i に比例して，重みパラメータの容量が大きくなることがわかる 4096 入力 4096 出力の全結合層を考えると，重みパラメータの容量は 67MB にもなる．

2.2 量子化手法

DNN における量子化とは，浮動小数点数から固定小数点数への変換や，1 データのビット幅を縮小させることで，ハードウェアコストやメモリ消費量を低減する手法である．量子化にともない，表現可能なデータの範囲が縮小するため，データの精度が低下し，それにより計算精度が低下する．DNN は一般に計算精度を低下させても認識精度が低下しにくいことが知られている．DNN による分類問題は，認識対象が各クラスに属する確率を計算し，その順位に基いて分類の予測がなされることが通常である．したがって，確率同士の大小関係が重要であり絶対値は重要でないため，単精度浮動小数点数ほどの高い計算精度は必要ない [16]．したがって，認識精度の低下を 0%から 10 数%程度に抑えながら，ビット幅を例えば 32 ビットから 16 ビット以下のより小さいビット幅に低減させる手法が多く提案されている．

DNN の推論計算時において，量子化の対象は重みのみの場合もあれば，重みと入力値の両方である場合もある．前者はメモリ消費量の削減が主な効果で，計算時には一般的にデータがデコードされ，フル精度で計算が行われる．

後者は積和演算における 2 つの入力の双方が量子化されるため，メモリ消費量の削減だけでなく，演算器をコストの小さな演算器に置き換えることができる．

2.2.1 線形な量子化

量子化手法は，浮動小数点数を固定小数点数で近似したり，入力や重みデータの最大値に合わせてスケーリングを行い小数点の位置を決めた後，下位のビットを削減することで行われることが一般的である．

DNN の量子化はモデル全体あるいは層の種類ごとに一律のビット幅が用いられることが多い．画像認識コンペティション ILSVRC2012 の勝者である AlexNet という CNN モデルにおいて，認識精度の劣化なしで，モデル全体で入力と重みの両方を 9 ビットに量子化できることが報告されている [13]．また，層毎にビット幅の異なった量子化をすることで，モデル全体を量子化したときのビット幅よりも大きくビット幅を削減する手法が提案されている [13]．各層の入力や重みの値の分散の違いなどから，層毎で最適なビット幅が異なると考えられている．

また，Binarized Neural Network (BNN) で知られるように，学習した重みを 2 値，3 値で再学習して，メモリ消費量とハードウェアコストを大幅に削減する手法も報告されている [7], [14], [18]．2 値化ネットワーク [14] は，認識精度の低下を AlexNet で 11 ポイント程度に抑えつつ，入力と重みの双方を 2 値化している．重みを 2 値にすることでメモリ消費量を 1/32 に削減し，積和演算の積を XNOR，和を Population Count というビットレベルの極めて単純な演算に帰着させることで，処理を高速・低消費電力に実行できる．3 値化ネットワーク [18] は，認識精度の低下を 1%以内に抑えながら，重みを-1, 0, +1 の 3 値，すなわち 2 ビットで表現することで，重みの容量を 1/16 に削減する．ハードウェアで実行する際，重みが 0 のときは計算をスキップし，0 以外のときは符号のみに乗算を適用することで処理を高速化できる．

2.2.2 非線形な量子化

前節で述べた量子化手法は，刻み幅が一律であるが，一律でない刻み幅を採用することで，少ないビット長で大きな範囲のデータを表現することができるため，ビット幅をさらに削減する手法が提案されている．

Weight-Sharing[4] は，重みを K-means クラスタリングによって数種類に分類する量子化手法である．AlexNet において，認識精度の低下なしに畳み込み層で 8 ビット，全結合層で 4 ビットまでビット幅を削減できると報告されている．この手法は線形な量子化よりもさらにビット幅を削減できる一方，重みの値をインデックスから真値にデコードする計算が必要になり，オーバーヘッドが大きいという欠点がある．

対数量子化手法 [10] は，量子化の刻み幅に対数を用いた手法である．底に 2 を用いた量子化をすることで，値が 2 の冪になるため，乗算ビットシフトに帰着させることでハードウェアコストを削減できる．AlexNet を対数量子化した実験によると，認識精度の低下は 3.7 ポイントで，重みをと入力を 5 ビットまで削減できると報告されている．

3. ニューロン毎の量子化

3.1 ニューロン毎の量子化手法

本節では，ニューロン毎に異なる量子化を行い，ビット幅を最適化するための手法について述べる．

本手法は，まず層毎に量子化を行い最適なビット幅を設定し，その後ニューロン毎の量子化を行うというステップからなる．層毎の最適なビット幅の探索は，[13] の手法に従い，第 1 層から順番に th を満たす最小のビットを求めていく．また，ニューロン毎の最適化は本稿では全結合層に限定して行うものとする．

量子化により認識精度は低下する可能性があるが，許容できる認識精度の低下割合の最小値を th とする．例えば，$th = 0.99$ のときはフル精度の 99%までの認識精度低下を許容することを示す．以下に，ある層におけるビット幅最適化の手順を示す．

(1) 重みのビット幅を 16 ビット (初期値) にする
(2) 推論計算を行った場合の認識精度を求める
(3) 重みパラメータのビット幅を 1 小さくし認識精度を求める
(4) 認識精度が th を満たす場合は，再び (3) から処理を行い，満たさない場合は (5) に移る
(5) 現在のビット幅に 1 を加えたものを当該層の最適なビット幅とする
(6) この層で求めた最適なビット幅を利用し，次層以降でも同様の処理を (1) から行う

この手法では，最適なビット幅の組み合わせが求まるとは限らない．ただし，上層から伝播される情報による精度の劣化を防ぎながら下層へと探索を行っていくため妥当な手法であると考えられる．

次に，ニューロン毎のビット幅の最適化について述べる．まず，上記で述べた層毎の最適なビット幅が求まっていると仮定する．ニューロン毎にも層毎のビット幅最適化の場合と同様に，th を満たす範囲で徐々にビット幅を縮小させるという貪欲法的な手法を用いる．まず，各層で何割のニューロンのビット幅を縮小してもよいかのパラメータを r と定める．ある層について，r の割合だけニューロンに接続される重みのビット幅を 1/2 倍にし th を満たす最大の r を求める．この際，どのニューロンのビット幅を削減するかの選択方法は，本稿では以下に示す単純な方法をまずは検討する．

対象の問題のテストデータセットに対する推論計算において，ニューロン毎の出力を記憶する．フル精度の場合と量子化後の精度での差が大きいニューロンは，それに接続する重みの精度が不足しているものと考え，大きいビット幅を与える．それ以外は小さいビット幅を与える．本稿で

は，大きいビット幅のデータを長ビットデータと呼び，小さいビット幅のデータを短ビットデータと呼ぶことにする．先に述べたように，各層でそれぞれの割合は $1-r:r$ となる．以下にビット幅を 1/2 にするニューロンの選択手順をまとめる．

(1) $r = 0.1$ とする
(2) テストデータに対してフル精度で推論処理を行い各ニューロンの出力値を記憶する
(3) 層毎の量子化で求めたビット幅で推論処理を行い各ニューロンの出力値を記憶する
(4) ニューロン毎にフル精度と量子化後の各出力値の差を求め，それを加算する
(5) 差の平均が下位 r の割合のニューロンを接続する重みをさらに縮小できるニューロンとして選択する
(6) ニューロン毎に量子化した場合での認識精度を求める
(7) 認識精度が th を満たす場合は，r の値を増加させて再度 (2) から繰り返す
(8) 認識精度が th を満たさない場合は，先の r の値で選択されたニューロン毎の量子化を採用する

3.2 3 層ニューラルネットワークの例

本節では，3 層ニューラルネットワークでのニューロン毎の量子化の具体例を示す．なお，以降では第 1 層から第 3 層までのビット幅を (n_{l1}, n_{l2}, n_{l3}) と表記して説明する．

まず，学習済みの 3 層のニューラルネットワークを用意する．このとき，層ごとのビット幅は (32, 32, 32) である．次に，層ごとの量子化を行うと，例えば重みのビット幅は (6, 4, 4) になる，

次に，第 1 層についてニューロン毎の量子化を行う．誤差が小さい方から割合 r のニューロンの重みを半分の精度である 3 ビットにする．$r = 0.1, 0.2, 0.3...$ と r の値を大きくしていき，例えば $th = 0.99$，つまり認識精度がベースラインの 99%を上回る最大の r を探索する．例えば $r = 0.1$ のとき，重みの 90%が 6 ビット，10%が 3 ビットとなり，認識精度がベースラインの 99%を上回っていたとすると，r の値を 0.2 にして再度ニューロン毎の量子化を行い，認識精度を求める．これを繰り返すと，例えば r の値が 0.7 になったとき，認識精度がベースラインの 99%を下回ったとすると，最適な割合 r は 0.6 となる．

最適な割合 $r = 0.6$ が見つかったら，その層に対して 0.4:0.6 の割合で長ビットデータは 6 ビット，短ビットデータは 3 ビットの量子化を行う．当該層での割合 r と量子化するニューロンが求められたら，次の層で同様の計算を繰り返す．

4. ニューロン毎の量子化のためのアーキテクチャ拡張

4.1 ベースのハードウェア構成

我々は，高い消費電力効率と様々なネットワークモデルを実行可能な柔軟性を備える DNN 推論用のアクセラレータチップを検討している [19]．本アクセラレータは，図 2 に示すように，マイクロコントローラと SIMD 型積和演算器を主な構成要素とするコアを複数搭載したマルチコアアクセラレータである．各コアは，命令メモリ (inst)，ストリームバッファ (sbuf)，データメモリ (dmem)，ルックアップテーブル (lut)，データ出力用メモリ (omem) の 5 つのメモリを持つ．基本メモリ構成としてはコア毎に別々のアドレス空間を持つ分散メモリシステムであるが，CNN を始めとした多層のニューラルネットワークを複数コアで実行する際，演算結果をコア間で共有する必要があるため，出力用メモリの omem はコア間で共有する．

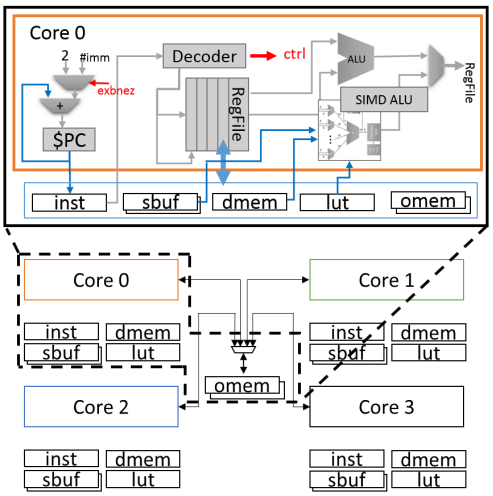

図 2 4 コア構成のアクセラレータ

コアは回路規模の小さなマイクロコントローラと SIMD 型積和演算器から構成される．マイクロコントローラは 16 ビット固定長の命令セットにより動作する．命令長が短いために実装可能な命令の機能は単純なものに限られるが，命令デコーダや制御回路も単純化されるため，小型で高電力効率なコントローラとなっている．一方で，DNN 向けアクセラレータでは，膨大な積和演算を効率よく実行できることが重要である．そこで，本研究のコアには SIMD 型積和演算器と独自のカスタム SIMD 算術命令を実装している．

メモリ構成

各コアが持つ 5 つのメモリ (inst, sbuf, dmem, lut, omem) は 32 ビットのアドレス空間に割り付けられており，load/store 命令によって全てのメモリとレジスタファイル間でデータをやり取りできる．inst は命令メモリで，sbuf と dmem は処理対象データ用メモリである．sbuf と dmem はデータを直接 SIMD 型積和演算器に供給するため，本アーキテクチャでは 64 ビット幅のデータバスを持つ．sbuf は再利用性の低いストリームデータ用のバッファとして，dmem は再利用性の高いデータ用のバッファとして用いる．sbuf 側は一般的にデータ転送量が多く，メモリボトルネックの原因となりやすいため，sbuf 側のみダブ

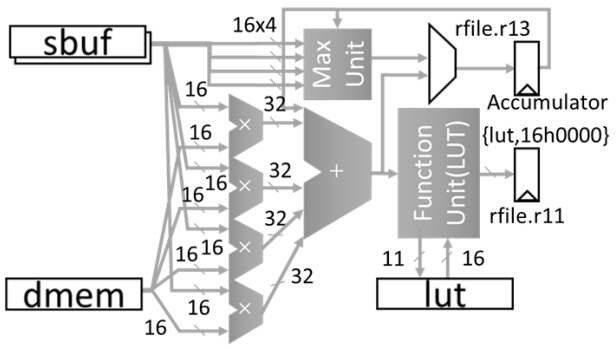

図 3 SIMD 型積和演算器

ルバッファリングを行うことで演算処理とデータ転送をオーバーラップさせ，実行時間の削減を狙う．なお，lut はニューラルネットワークの活性化関数に利用し，omem は出力データ用である．

以上のようなメモリ構成をとる目的は，畳込み層と全結合層のデータ再利用性に関する特性の違いに対応するためである．畳込み層では学習済み重みパラメータの再利用性が高く各層の入力データの再利用性が低いが，全結合層では逆転し各層の入力データの再利用性が高く学習済み重みパラメータの再利用性が低いという違いがある．

マイクロコントローラ

本アーキテクチャは 4 段パイプラインのインオーダー実行で，MIPS に近い形式の 16 ビット固定長命令セットを解釈実行する．マイクロコントローラの主要な役割は，SIMD 型積和演算器の制御やループの制御，メモリへの load/store である．レジスタファイルは 32 ビット 16 本であるが，そのうち 4 本を SIMD 型積和演算器の演算結果が格納される特殊レジスタに，1 本をプログラムカウンタに割り当てているため，汎用レジスタは 11 本となる．演算器は 32 ビット長で論理算術演算が可能である．

SIMD 型積和演算器

SIMD 型積和演算器の基本構成を図 3 に示す．SIMD 型積和演算器は 16 ビット長データ 4 並列で演算を行うことができ，実行可能な演算はテーブルルックアップ付きの積和演算と MAX 演算である．処理対象データはレジスタファイルを介さずに，sbuf と dmem から直接演算器に供給され，データバスは 64 ビット幅である．ルックアップテーブル (lut) はニューラルネットワークの活性化関数に利用する．DNN アクセラレータの先行研究では活性化関数に ReLU 関数のみをサポートするものもあるが，汎用性の観点からルックアップテーブルによる実装を採用している．SIMD 型積和演算器の演算結果は，積和演算や MAX 演算の場合はレジスタファイルの 13 番レジスタに自動的に保存され，ルックアップテーブルの場合は 11 番レジスタに保存される．

積和演算の詳細な動作としては，sbuf と dmem それぞれから 16 ビット固定小数点形式サイズ 4 のベクトルデータが SIMD 型積和演算器に投入され，その内積演算結果が 13 番レジスタにアキュムレートされる．なお，4 つの乗算器はマスクレジスタによって制御可能である．一方，MAX 演算の場合は，sbuf から供給された 16 ビット固定小数点形式データ 4 つと現在の 13 番レジスタの値の MAX 演算結果を 13 番レジスタに保存する．こちらも sbuf から供給された 4 つのデータに対しマスクレジスタによる制御が可能である．前述のマルチサイクルのカスタム SIMD 算術命令は，積和演算や MAX 演算を sbuf と dmem アドレスをインクリメントしながら指定回数連続実行する．

4.2 アーキテクチャの拡張

ネットワーク全体や層毎に量子化ビット幅を変更するような粗粒度なビット幅の最適化の場合は，例えばその実行前にハードウェアのモードを変更したり，ハードウェアを再構成することで，実現することが可能である．一方で，ニューロン毎に重みのデータのビット幅を変えるような細粒度なビット幅の最適化を行う場合には，重みのデータをチップ内のバッファに転送する際に，ビット幅の異なるデータが混在する可能性もあり，モードの切り替えや各データのビット幅の判定を効率良く行うことが重要となる．本稿では，そのためのアーキテクチャを，前節で述べたアクセラレータを拡張することで実現する．なお，基本的なコンセプトやハードウェアの拡張方法は，アクセラレータアーキテクチャとは比較的独立であるため，他のアーキテクチャにも適用可能であると考えられる．

図 4 に拡張部分のハードウェア構成を示す．これは，図 3 の sbuf と dmem，SIMD 演算器部分に相当する．まず，sbuf の 1 エントリのビット幅は半分になり，各演算器へと供給するビット幅も半分になる．また，sbuf の各エントリへのフラグビットを示す xflag と呼ぶ領域を新たに sbuf に追加する．xflag は，値が 0 であれば通常の演算モード用のデータが格納され，値が 1 の場合は 2 倍のビット幅を利用する 2 倍ビット長演算モード用のデータであることを示す．また，SIMD 演算器の入力部にはラッチを設ける．

sbuf からのデータをロードする際に xflag をチェックし，その値が 0 の場合，すわなち通常の演算モード用のデータがロードされた場合は，図 5 に示すように各演算器に対応するラッチの上位半分のビット部に sbuf からのデータを格納する．また，下位ビット部には 0 を挿入する．一方で，xflag の値が 1 の場合，すわなち 2 倍ビット長演算モード用のデータであった場合は，図 6 に示すように，ラッチの上位半分のビット部にいったんロードされたデータを格納し，次のサイクルで下位ビット部に格納するべきデータを sbuf からロードし，先のデータと連結して 1 データとする．その後演算が実行される．

例えば，各演算器の入力ビット幅が 16 ビットの場合は，sbuf 上の 1 エントリに保存されるデータ長は 8 ビット ×SIMD 演算器数となり，各演算器には 8 ビットのデータが供給される．したがって，通常モードでは 8 ビットデー

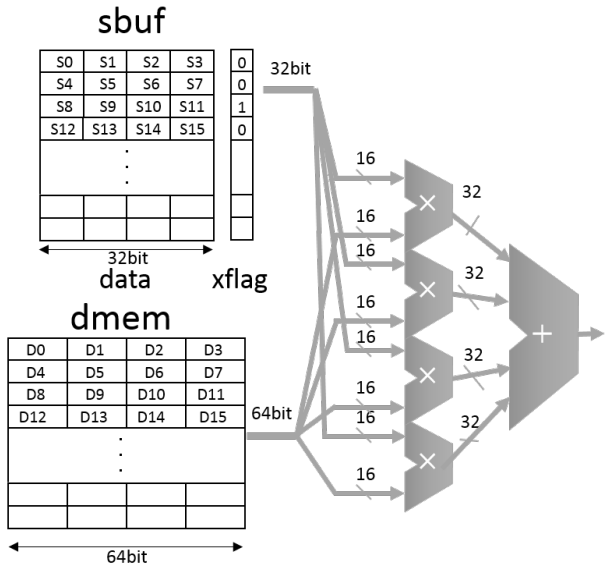

図 4 アーキテクチャの拡張

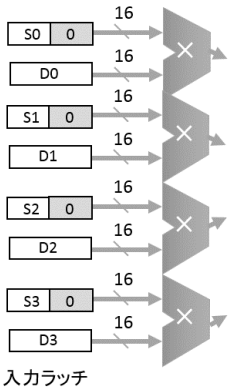

図 5 通常の演算 (xflag = 0)

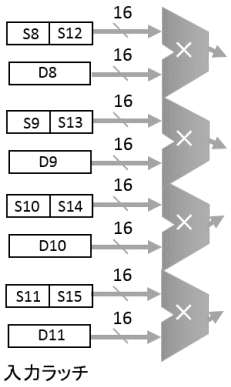

図 6 2 倍ビット長演算モード (xflag = 1)

タと下位 8 ビットに 0 が拡張された 16 ビットデータに対して演算が行われる．一方で，2 倍ビット長演算モードであれば，8 ビットデータを 2 回ロードして 16 ビットデータを生成しそれに対して演算が行われる．

2 倍ビット長演算モードの際には，データを 2 回ロードする必要があり，演算スループットが悪化するが 2 倍ビット長演算モードを利用するデータの割合が全体に対して少ない場合には大きな問題とならないと考えられる．一方で，多くのデータのビット幅を削減できれば，特に全結合層においてメモリボトルネックの影響が緩和されるため，その効果は大きい．オーバーヘッドの低減方法などを含めたアーキテクチャのさらなる拡張については，今後さらに検討する予定である．

5. 予備評価

5.1 評価手法

ニューロン毎にビット幅を最適化することによるデータ量の削減効果と，画像認識精度への影響を確認するために予備評価を行う．3.1 節で述べた手法に従い，まず層毎のビット幅最適化を行い，その結果を元にしてさらにニューロン毎のビット幅最適化を行う．

評価に用いたネットワークモデルは，ディープラーニングフレームワーク Chainer[15] の手書き文字認識サンプルのネットワークであり，3 層のパーセプトロンである．各層同士は全結合しており，ノード数は，入力層が 784，2 つの中間層である第 1 層，第 2 層がそれぞれ 1000，1000，出力層である第 3 層が 10 である．活性化関数は ReLU 関数である．データセットには，MNIST[9] を用いた．

本稿では，初期評価として，実際に固定小数点で量子化や演算を行うのではなく，以下に示す方法で重みの量子化と固定小数点数の精度をエミュレートした．

まず，学習は 32 ビットの浮動小数点数で行われるため，32 ビットの浮動小数点数データに量子化したいビット数分に対応する 2 のべき乗数を掛けてから整数に変換し，再度浮動小数点数に変換する．実際の計算は 32 ビットの浮動小数点演算で行った．本評価での量子化の手順を，以下に示す．なお，小数点以下のビット幅を n ビットとする．

(1) 重みパラメータ (float 32-bit) に 2^n を乗算する
(2) 重みパラメータを int 16-bit に変換する
(3) 重みパラメータを float 32-bit に変換する
(4) 重みパラメータを 2^n で除算する

なお，学習済みの重みの値は全て 1 未満であり，小数点以下のみを保持すればよいため，小数部のみ量子化すると想定する．そのため，小数点以下のビット幅が n ビット，小数点以上は 0 ビット，符号ビットを 1 ビットとした．また，提案ハードウェアを利用することを想定し，ニューロン毎の量子化を行う場合はさらに xflag として余分に 1 ビットが必要であるとしてデータサイズを算出した．

5.2 評価結果

評価結果を表 1 に示す．1 行目と 2 行目は層毎に量子化を行った場合を示している．また，1 列目から順に 1 層目，2 層目，3 層目のビット幅，32 ビット浮動小数点数での認識精度に対する量子化による精度の割合，認識精度の絶対値，各層のビット幅が (4,3,3) の場合における 1 層目のデータ量の変化，各層のビット幅が (4,3,3) の場合におけるネットワーク全体のデータ量の変化，各層が 32 ビット浮動小数点数の場合における 1 層目のデータ量の変化，各層が 32 ビット浮動小数点数の場合におけるネットワーク全体のデータ量の変化を示している．さらに，値が 0 でない 1 層目の重みの数，その割合，ネットワーク全体について値が 0 でない重みの割合も示している．

表 1 量子化の精度のデータの圧縮率

	bit width of l1	bit width of l2	bit width of l3	accuracy loss rate (vs fp32)	accuracy	compress rate of layer1 (vs 4, 3, 3)	compress rate of model (vs 4, 3, 3)	compress rate of layer 1 (vs fp32)	compress rate of model (vs fp32)	#non-zero weights of layer1	non-zero rate of layer1	non-zero rate of model
layer-wise	4	3	3	99.11%	0.9737	100.0%	100.0%	15.6%	13.9%	299093	38.10%	24.00%
	3	3	3	93.50%	0.9185	75.0%	89.7%	12.5%	9.4%	97342	12.40%	12.07%
r=0.9	8/4	3	3	99.11%	0.9737	108.0%	103.9%	16.9%	14.4%	339170	43.26%	26.20%
	6/3	3	3	96.52%	0.9482	86.0%	93.1%	13.4%	12.9%	148736	18.97%	15.58%
	4/2	3	3	73.68%	0.7238	64.0%	82.3%	10.0%	11.4%	42053	5.36%	9.64%
	2/1	3	3	11.25%	0.1105	42.0%	71.4%	6.6%	9.9%	2050	0.26%	7.41%
r=0.8	8/4	3	3	99.11%	0.9737	116.0%	107.9%	18.1%	15.0%	379366	48.39%	28.44%
	6/3	3	3	98.37%	0.9664	92.0%	96.1%	14.4%	13.3%	200281	25.55%	18.46%
	4/2	3	3	94.89%	0.9322	68.0%	84.2%	10.6%	11.7%	75168	9.59%	11.48%
	2/1	3	3	13.75%	0.1351	44.0%	72.4%	6.9%	10.0%	3726	0.48%	7.50%
r=0.7	8/4	3	3	99.22%	0.9747	124.0%	111.8%	19.4%	15.5%	420970	53.70%	30.76%
	6/3	3	3	98.77%	0.9703	98.0%	99.0%	15.3%	13.7%	253118	32.29%	21.40%
	4/2	3	3	97.24%	0.9553	72.0%	86.2%	11.3%	12.0%	107303	13.69%	13.27%
	2/1	3	3	14.16%	0.1391	46.0%	73.4%	7.2%	10.2%	5035	0.64%	7.57%
r=0.6	8/4	3	3	99.36%	0.9761	132.0%	115.8%	20.6%	16.1%	463471	59.12%	33.13%
	6/3	3	3	99.24%	0.9749	104.0%	102.0%	16.3%	14.1%	305653	38.99%	24.33%
	4/2	3	3	98.61%	0.9687	76.0%	88.2%	11.9%	12.2%	138843	17.71%	15.03%
	2/1	3	3	14.92%	0.1466	48.0%	74.4%	7.5%	10.3%	6199	0.79%	7.64%
r=0.5	8/4	3	3	99.38%	0.9763	140.0%	119.7%	21.9%	16.6%	506497	64.60%	35.53%
	6/3	3	3	99.26%	0.9751	110.0%	104.9%	17.2%	14.5%	359026	45.79%	27.31%
	4/2	3	3	98.85%	0.9711	80.0%	90.2%	12.5%	12.5%	169596	21.63%	16.75%
	2/1	3	3	14.61%	0.1435	50.0%	75.4%	7.8%	10.5%	7042	0.90%	7.69%
r-0.4	8/4	3	3	99.42%	0.9767	148.0%	123.6%	23.1%	17.1%	550212	70.18%	37.96%
	6/3	3	3	99.38%	0.9763	116.0%	107.9%	18.1%	15.0%	413022	52.68%	30.32%
	4/2	3	3	99.10%	0.9736	84.0%	92.1%	13.1%	12.8%	199102	25.40%	18.39%
	2/1	3	3	0.1475	0.1449	52.0%	76.4%	8.1%	10.6%	7715	0.98%	7.72%

層毎の量子化の結果では，小数点以下のビット幅を $(4,3,3)$ まで縮小しても，フル精度 (32 ビットの浮動小数点) で計算した場合と比べて 99%以上の認識精度が達成できている．$(3,3,3)$ に縮小すると，認識精度の低下率が大きくなるため，層毎の量子化として $(4,3,3)$ を基準にする．

次に，ニューロン毎の量子化に対する結果について議論する．表 1 中，3 行目以降は 1 層目に関して，ニューロン毎に量子化した場合の結果を示している．$r = 0.9$ は，3.1 節で述べたように，1 層目のニューロン中全体の 90%分を短ビットデータに，10%分を長ビットデータにした場合である．さらに，1 層目のビット幅で 8/4 と書かれているのは，長ビットデータのビット幅/短いビットデータのビット幅を表している．

表より長ビットデータの割合が大きいほど，また長ビット/短ビットデータのビット幅が大きいほど，高い精度が達成できる傾向にあることがわかる．ベースラインである $(4,3,3)$ の場合と比較すると，単ビット幅の割合が 90%〜50%の場合には，ベースラインよりも圧縮率が高くかつ認識精度を上回るものがない．しかし，短ビットデータの割合を 40%にした際に，長ビット/短ビットデータのビット幅が 4/2 のときは，フル計算精度での認識精度の 99%以上を維持しつつ，ベースラインと比べてのデータ圧縮率が 84%となっている．本結果より，ニューロン毎の量子化手法を用いることで，層毎の量子化と比較しても精度を犠牲にせずに，さらにビット幅を最適化できる余地があると考えられる．

6. まとめ

本稿では，可変データビット幅を持つ DNN とそのアクセラレータアーキテクチャについて検討を行った．特に，ニューロン毎に異なるビット幅に量子化する手法と，それを効率的に実行可能なアーキテクチャを提案した．MNIST ベンチマークの手書き文字認識を行う 3 層のニューラルネットワークに対し，ニューロン毎の量子化を適用した際の予備評価を行った結果，層毎の量子化に比べても高い精度を達成しつつ，よりデータを圧縮できる可能性があることを示した．ただし，長ビットデータが多いと，提案アーキテクチャでは演算スループットが低下する恐れがあるため，アーキテクチャをさらに改善するなどして，効果的にニューロン毎の量子化を行った DNN が実行できるアクセラレータを検討する必要がある．その他に，AlexNet などの画像認識ベンチマークで評価を行うことや，ノードやパラメータの枝刈り手法と組み合わせて量子化を行うこと，さらに実際のハードウェア環境を模倣したシミュレータなどで推論にかかる実行時間や消費エネルギーを見積もることなどが今後の課題である．

謝辞 本研究の一部は JSPS 科研費基盤研究 (S)25220002 の助成によるものである．

参考文献

[1] Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., and Temam, O.: *Dadiannao: A machine-learning supercomputer*, Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (pp. 609-622) (2014).

[2] Chen, Y. H., Krishna, T., Emer, J. S., and Sze, V.: *Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks*, IEEE Journal of Solid-State Circuits, 52(1), 127-138 (2017).

[3] Han, S., Pool, J., Tran, J., and Dally, W.: *Learning both weights and connections for efficient neural network*, Advances in Neural Information Processing Systems (pp. 1135-1143) (2015).

[4] Han, S., Mao, H., and Dally, W. J.: *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, arXiv preprint arXiv:1510.00149 (2015).

[5] Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., and Dally, W. J. *EIE: efficient inference engine on compressed deep neural network* Proceedings of the 43rd International Symposium on Computer Architecture (pp. 243-254) (2016).

[6] He, K., Zhang, X., Ren, S., and Sun, J.: *Deep residual learning for image recognition*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 770-778) (2016).

[7] Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y.: *Binarized neural networks*, Neural Information Processing Systems (pp. 4107-4115) (2016).

[8] Krizhevsky, A., Sutskever, I., and Hinton, G. E.: *Imagenet classification with deep convolutional neural networks*, In Advances in neural information processing systems (pp. 1097-1105) (2012).

[9] LeCun, Y.: *The MNIST database of handwritten digits*, http://yann. lecun. com/exdb/mnist/.

[10] Lee, E., Miyashita, D., Chai, E., Murmann, B. and Wong, S. *LogNet: Energy-efficient Neural Networks using Logarithmic Computation*, International Conference on Acoustics, Speech and Signal Processing (2017).

[11] Mikolov, T., Karafiat, M., Burget, L., Cernocky, J., and Khudanpur, S.: *Recurrent neural network based language model*, In Interspeech (Vol. 2, p. 3) (2010).

[12] Miyashita, D., Lee, E. H., and Murmann, B.: *Convolutional neural networks using logarithmic data representation*, arXiv preprint arXiv:1603.01025 (2016).

[13] Moons, B., De Brabandere, B., Van Gool, L., and Verhelst, M. (2016, March): *Energy-efficient ConvNets through approximate computing*, Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on (pp. 1-8) (2016).

[14] Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A.: *Xnor-net: Imagenet classification using binary convolutional neural networks*, European Conference on Computer Vision (pp. 525-542). Springer International Publishing (2016).

[15] Tokui, S., Oono, K., Hido, S., Clayton, J.: *Chainer: a Next-Generation Open Source Framework for Deep Learning*, In Workshop on Machine Learning Systems at Neural Information Processing Systems (2015).

[16] Vanhoucke, V., Senior, A., and Mao, M. Z.: *Improving the speed of neural networks on CPUs*, Deep Learning and Unsupervised Feature Learning NIPS Workshop (Vol. 1, p. 4) (2011).

[17] Zhou, A., Yao, A., Guo, Y., Xu, L., and Chen, Y.: it Incremental network quantization: Towards lossless cnns with low-precision weights, arXiv preprint arXiv:1702.03044 (2017).

[18] Zhu, C., Han, S., Mao, H., and Dally, W. J.: *Trained Ternary Quantization*, arXiv preprint arXiv:1612.01064 (2016).

[19] 高田, 石井, 坂本, 近藤, 中村, 大久保, 小島, 天野, : スケーラブルなディープラーニング向けアクセラレータチップの設計と評価, 情報処理学会研究報告, Vol. 2016-ARC-223, No. 1, pp. 1-6, (2016).