卒業論文

題目

ディープラーニングアクセラレータの演算精度低減の研究

指導教員

近藤利夫教授

2017年

三重大学 工学部 情報工学科
コンピュータアーキテクチャ研究室

川合悠生 (413812)

内容梗概

Deep Learningを利用した多層のニューラルネットワーク(以下NN)が，高い認識性能を発揮することが2012年に発表されて以来，車両検出，歩行者認識などの画像認識や，音声認識などへの応用に向けて，多くの企業や研究者が活発に研究を行っている．その中で，GoogleはASIC(Application Specific Integrated Circuit:特定用途向けIC)を利用することで，従来と比較し，消費電力あたりの性能を10倍に高めるASICのDeep Learning専用プロセッサであるTPU(Tensor Processing Unit)を開発している．この種の専用ハードウェアの性能コスト比改善には，演算の精度を抑えて必要トランジスタを減らすことが消費電力あたりの性能を高めるには必須である．

そこで，本論文では構成ゲートの大半を占める乗算器の演算精度をNNの層ごと，または学習途中で動的に切り替える新たな演算精度低減手法を提案し，12ビット精度の演算の割合を、平均で先行研究より20 %から30 %まで高められることを明らかにしている．

Abstract

Multi-layered neural network(NN) has been announeced in 2012 and it can provide a high recognition performance by using Deep Learning. in recent years, many companies and researchers are actively researching application for image recognition, such as vehicle detection, pedestrian recognition, and speech recognition. Especially, Google has been developed the TPU(Tensor Processing Unit) by using ASIC(Application Specific Integrated Circuit), which is a dedicated processor for Deep Learning. The increase of the performance per power consumption for the TPU is ten times that of the conventional method. However, in order to improve the performance cost ratio of this dedicated hardware, it is essential to reduce the number of required transistors while suppressing the computation accuracy to improve the performance per power consumption.

In this study, we proposed a method that switch over the operation precision of the multiplier during learning or switch over the operation precision of the multiplier that occupy the majority of constituent gates for each layer of NN. The evaluation results show that the performance of the operation with 12bit are improved by 20 % to 30 % than the previous study.

目 次

図 目 次

表 目 次

1　まえがき

1.1　背景

現在車両検出，歩行者認識などの画像認識や，音声認識などに．ニューラルネットワーク (以下 NN) を多層に用いた Deep Learning が用いられている．脳を模した計算アルゴリズムである現在の NN の前身は，1940 年代から研究が行われてきたが，NN の脳を模したモデルの実装が複雑な上，演算器の性能が NN の実装にはあまりに低かったため，応用は一部に限られてきた．しかし，2012 年に行われた ILSVRC2012(ImageScaleVisual RecognitionChallenge) では画像認識の分野で，FisherVector などを用いた従来法の結果を，Deep Learning による認識の手法が 10 %ほど多く上回った．2012 年以前では数%の改善しかされなかったため，この結果は多くの注目を集めた．また同年に Google が深層学習で YouTube にアップロードされた動画を使って 1 週間ほど学習させたところ，猫の概念を教えてないにもかかわらず，猫に反応するニューロンが生成され，注目を集めた．それらの発表により，過学習が発生しやすい，パラメータの調整が難しい，といった問題点から敬遠されていた Deep Learning の研究が多く行われるようになった．それらに続き，従来の方法を大きく上回る認識率を Deep Learning が達成されたことから，次々と高速性や，高精度を謳

うニューラルネットモデルや専用ハードウェアが提案されてきた．その結果，近年，Google や Facebook といった大手企業が力を注いできたこともあり,Deep Learning の研究・開発は著しく進んでいる．たとえば，Google は Deep Learning 専用プロセッサである TPU(Tensor Processing Unit) を開発し，使用していることを発表した [1]．この TPU は Deep Learning のために開発された ASIC(Application Specific Integrated Circuit:特定用途向け IC) で，従来の GPU，CPU などと比較し，消費電力あたりの性能 10 倍を示した．TPU が消費電力あたりの性能が GPU などと比べて高いのは，演算の精度を必要最低限に抑えることで，計算に必要となるトランジスタを減らしているからである．このように，演算の精度を抑え必要トランジスタを減らすことは，消費電力あたりの性能を高めるには必須である．

1.2 研究目的

Deep Learning を学ぶ上で重要な点は良い構造をした NN のモデル，モデルに与えるデータ量，そして GPU などの集積回路の 3 点と言われている．現在，検索エンジンや音声認識等で Deep Learning の技術は幅広く応用されていて，将来的には産業用ロボットや車の自動運転技術にも応

用が期待されるため，大量生産を可能とする高並列化，ハードウェア規模削減は必要不可欠であると考える．その中で，Deep Learning 用のハイスペック GPU の研究・開発が盛んで，演算精度 8bit を示している．Deep Learning の学習には多く行列演算が含まれるため，一般的には GPU が用いられるが，中には省電力やレイテンシが小さいなどの FPGA，TPU などの整数演算ベース専用構成の利点を生かした研究も行われている．その整数演算ベース専用構成は構成ゲートの大半を乗算器が占める．整数演算ベース専用構成の研究の中でも，性能を高め，演算の精度を抑えることで必要トランジスタを減らす研究もされているが，特に FPGA の演算精度の研究は 16bit ほどにとどまっている [2]．本研究では，演算精度を動的に切り替えることで平均の演算精度 16bit 以下に抑える演算精度低減法を明らかにする．また，演算精度低減法を利用したハードウェア規模削減法を示すことで，FPGA などの整数演算ベース専用構成を高並列化し，GPU にも引けをとらない性能を発揮させることを提案する．

2 Deep Learning

NN は動物の脳の神経回路の仕組みを模したモデルで，1940 年代から研究が行われ，2 度程，ブームと終焉を繰り返してきた．1980 年代に発見

された NN 学習方法の誤差逆伝播法は，2 度目のブームのきっかけとなった．しかし，このブームも 90 年代後半には次の 3 つの理由から終焉した．第一に，誤差逆伝播法による学習は 3 層程度の NN ではうまく行えるが，それ以上に層が増えるとうまく学習できず過学習してしまうということ，第二に，層の数やユニット数をどのように選ぶかが他の機械学習と比べ難しいということ，第三に，当時の計算機の能力では現実的な問題を扱い得る規模のネットワークを扱えなかったということが挙げられる．その後，2006 年のプレトレーニングという学習手法の提案により，多層の NN の学習（Deep Learning）が可能になった．NN の構造には畳み込み NN，多層パーセプトロン等がある．本章では NN の構造と学習，および本研究で用いた畳み込み NN について簡単に説明する．

2.1 ニューロンモデル

ニューロンモデルとは脳の神経回路であるニューロンを模したユニットである．実際のニューロンには単純作用するものから複雑な作用するまで様々存在するが，現在 NN に用いられているモデルは以下の図 2.1 ような単純化したユニットとなる．ニューロンモデルは神経回路のモデルであるので，入力出力が存在する．出力は $y = f(\sum_i w_i x_i + b)$ となり，入

力 x と重み w の積和にバイアス b を加算したものに活性化関数を適用する．その活性化関数には

$$\text{ロジスティック関数} \qquad f(x) = \frac{1}{1+\exp(-x)} \tag{1}$$

$$\text{双曲線正接関数} \qquad f(x) = tanh(x) \tag{2}$$

$$\text{打ち切り線形関数} \qquad f(x) = max(0, x) \tag{3}$$

がよく用いられる．近年では効率的に学習が進められるため，中でも打ち切り線形関数が一般的に用いられている．

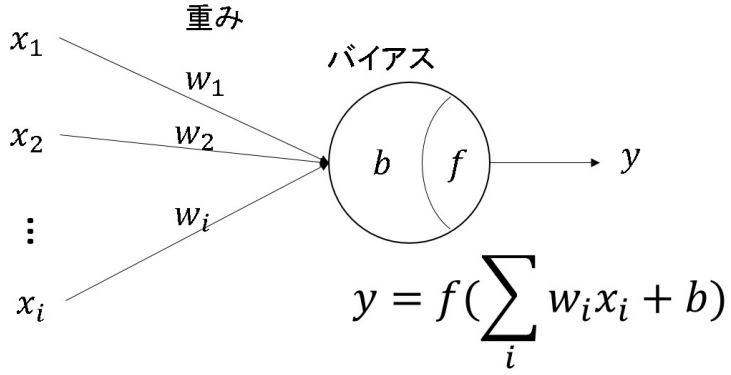

図 2.1: ニューロンモデル

2.2 ニューラルネットワーク

先ほどまでのニューロンを多数組み合わせることで形成さされたものをニューラルネットワーク (NN) という．NN には大きく 2 種類存在する．

1 つは階層型ネットワークと呼ばれるもので，ネットワークが層にわかれており，信号の流れは一方向のみなっている．信号は同じ層に 2 度入ることはなく，入力層に与えられた信号が同じであれば，出力層から出る信号は一意である．

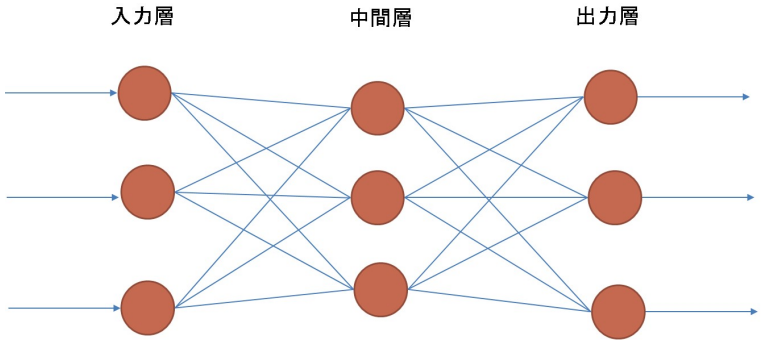

図 2.2: 階層型ニューラルネットワーク

もう 1 つは相互結合型ネットワークと呼ばれるもので，それぞれのニューロンが相互に結合していて信号の流れは一方向でなく，一度出力された信号であってもまた返ってくることもある．

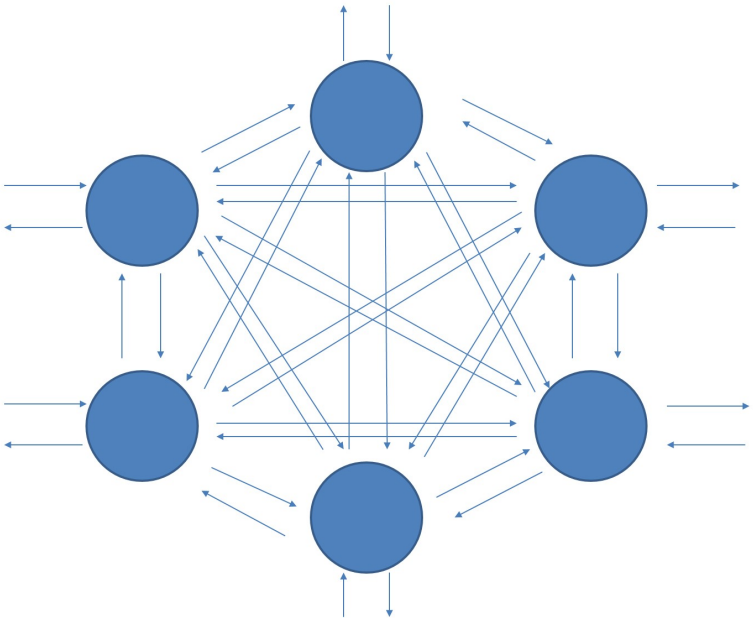

図 2.3: 相互結合型ニューラルネットワーク

2.3 畳み込みニューラルネットワーク

畳み込み NN(CNN:Convolutional Neural Network) は畳み込み層，プーリング層を繰り返し，その後に全結合層を持つ構成をとる．

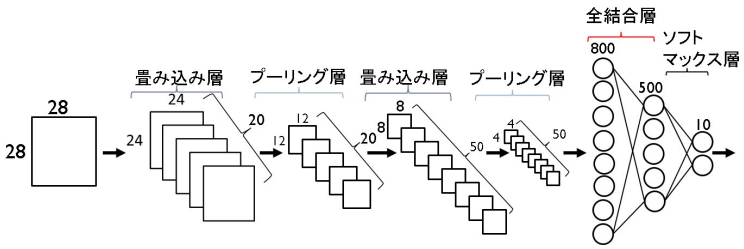

図 2.4: 畳み込みニューラルネットワーク

CNN は数ある種類の多層ニューラルネットワークの中でもプレトレー

ニングを行わず，ランダムな初期値から学習を開始しても充分な性能を発揮することが示されている．畳み込み層，プーリング層を繰り返すことで特徴量を自動抽出する．畳み込み層では入力された画像に重みフィルタを畳み込んだ結果である特徴マップを得て，プーリング層では得た特徴マップの解像度を落とすことで余分な情報を捨てることを行っている．それを繰り返した後，全結合層に繋ぎ，確率で表すために出力層でsoftmax 関数を用いる．

2.3.1 畳み込み層

畳み込み層では入力画像に対してフィルタを畳み込む処理を行い，特徴マップと呼ばれる出力画像を得る動作が行われる層である．画像，フィルタサイズをそれぞれ $n_x * n_y$, $n_w * n_w$ としたとき，特徴マップのサイズは $n'_x = n_x - n_w + 1$, $n'_y = n_y - n_w + 1$ となる．

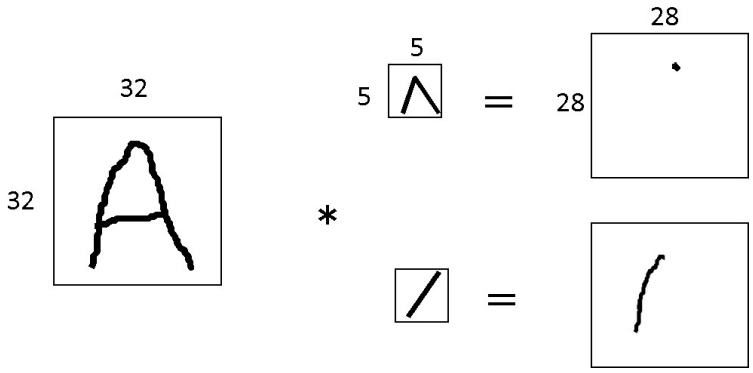

図 2.5: 畳み込み処理

2.3.2 プーリング層

プーリングとは抽出した特徴から認識に余分な情報を捨て，認識に必要な情報を保った新たな表現に変換することである．画像の解像度を落とすことで余分な情報を捨てる．プーリング層ではそのような動作が行われる．プーリングには以下が存在する．

$$\text{平均プーリング} \qquad h'_i = \frac{1}{|P_i|} \sum_{j \in P_i} h_j \tag{4}$$

$$\text{マックスプーリング} \qquad h'_i = \max_{j \in P_i} h_j \tag{5}$$

$$Lp\,\text{プーリング} \qquad h'_i = (\frac{1}{|P_i|} \sum_{j \in P_i} h_j^P)^{\frac{1}{P}} \tag{6}$$

平均プーリングとは $k-1$ 層での小領域 P_i でのデータの平均を k 層でのニューロン i の値とする式．マックスプーリングとは $k-1$ 層での小領域 P_i でのデータの最大値を k 層でのニューロン i の値とする式．平均プーリ

ングとマックスプーリングの中間的方法としてLpプーリングが存在する．

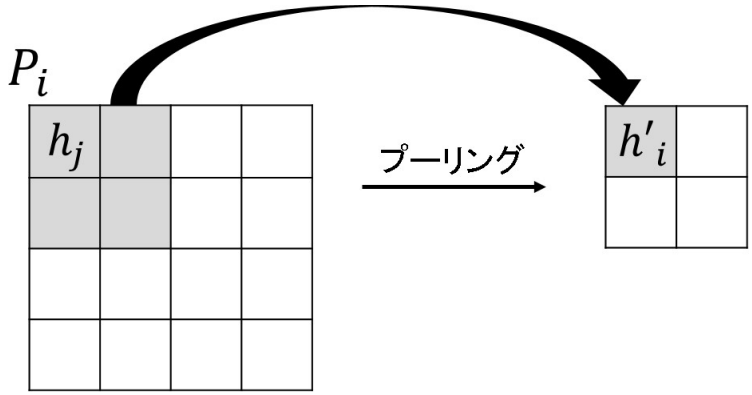

図 2.6: プーリング処理

2.4 Deep Learningの学習アルゴリズム

Deep Learningとは多層のNNを用いた機械学習の手法である．Deep Learningの学習とは，ユニットのパラメータ(重み、バイアス)を調整し，訓練データを入力したときの出力を望みの出力に近づけることである．実際の出力と望みの出力の差は，交差エントロピー

$$C = \sum_i d_i \log p_i \tag{7}$$

で示すことができる．ここではd_iは教師データで1つだけが1をとり，それ以外はすべて0となる．このCをパラメータを調整し小さくすること

でが学習が行われる．このとき，勾配降下法と呼ばれるアルゴリズムが用いられる．

2.4.1 勾配降下法

勾配降下法とは関数の勾配から極小値を探索するアルゴリズムである．

$$\begin{aligned} w_{ij} &\leftarrow w_{ij} + \Delta w_{ij} \\ \Delta w_{ij} &= -\epsilon \frac{\partial C}{\partial w_{ij}} \end{aligned} \tag{8}$$

以上の式のようにコスト C のパラメータに関する勾配を計算し，連続的にパラメータを更新していく．ϵ は学習係数と呼ばれ学習の幅を決める．学習係数が大きすぎると発散してしまい，小さすぎると反復回数が増え学習進行が遅くなる．

2.4.2 誤差逆伝搬法

勾配降下法ではコスト C のパラメータに関する勾配 $\frac{\partial C}{\partial w_{ij}}$ を計算する必要がある．C はニューラルネットワークの最終出力によって定義される．出力層の重み w_{ij} に関する勾配は，

$$\frac{\partial C}{\partial w_{ij}} = (p_i - d_i) h_j \tag{9}$$

で求められる．各層の出力は直前の層の出力の関数になっていて，各パラメータは $f(f(f))$ のような活性化関数の入れ子の形になって計算が難しいため，誤差逆伝搬法が用いられる．ある中間層を含む 3 層のユニットのインデックスを，上から順に l, i, j で表し，簡単化のためバイアスを省きユニット i への入力を

$$x_i = \sum_j w_{ij} h_j \tag{10}$$

と書く．このとき，w_{ij} についての勾配 $\frac{\partial C}{\partial w_{ij}}$ は，微分の連鎖法則より

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial x_i} \frac{\partial x_i}{\partial w_{ij}} \tag{11}$$

と書ける．ここでこの右辺第 1 項を $\delta_i \equiv \frac{\partial C}{\partial x_i}$ と置く．第 2 項は $x_i = \sum_j w_{ij} h_j$ の関係から

$$\frac{\partial x_i}{\partial w_{ij}} = h_j \tag{12}$$

と計算できる．一番上の層のユニット l への入力 x_l は、$x_l = \sum_i w_{li} h_i = \sum_i w_{li} f(x_i)$ と与えられるように x_i の関数である．C は各 $x_l (l = 1, 2, \ldots)$ の関数であるとみることもでき，微分の連鎖法則より δ_i は

$$\delta_i = \frac{\partial C}{\partial x_i} = \sum_l \frac{\partial C}{\partial x_l} \frac{\partial x_l}{\partial x_i} \tag{13}$$

と展開できる．この右辺第1項を$\delta_l \equiv \frac{\partial C}{\partial x_l}$と置く．第2項は$x_l = \sum_i w_{li} f(x_i)$の関係から

$$\frac{\partial x_l}{\partial x_i} = f'(x_i) w_{li} \tag{14}$$

と計算され，以上より

$$\delta_i = f'(x_i) \sum_l \delta_l w_{li} \tag{15}$$

と書ける．この式より一番上の層のδ_lが与えられると，中央のδ_iを計算できる．一番上の層が出力層であったとすると，このユニットのδ_lは$\delta_l = p_l - d_l$で計算できる．式(15)のインデックスを読み替えて，出力層から入力層へ向かって順番に用いるとすべての層でδ_iを計算できることになる．δ_iが得られることで，式(11)より，勾配$\frac{\partial C}{\partial w_{ij}}$は

$$\frac{\partial C}{\partial w_{ij}} = \delta_i h_j \tag{16}$$

と計算できる．δ_iを誤差と考えると，以上の計算は誤差δ_iを出力層から入力層へ逆方向に計算していくことに相当し，これが誤差逆伝播法と呼ばれる．

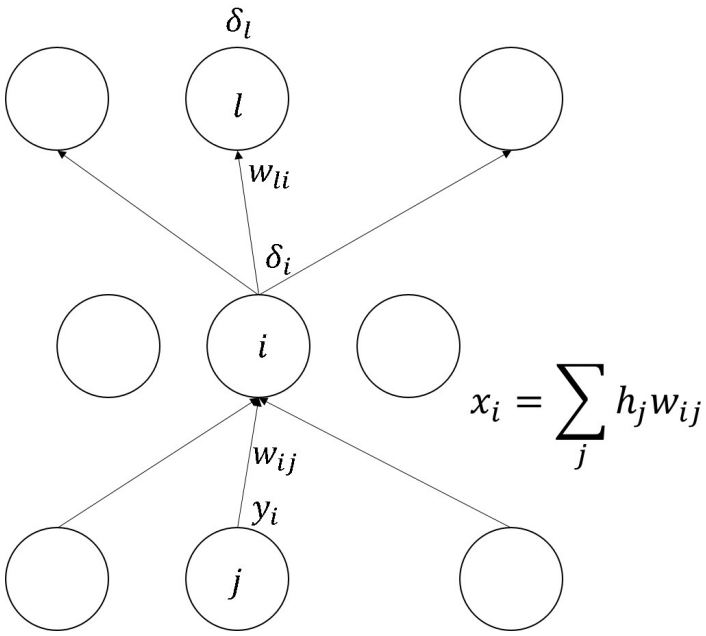

図 2.7: 誤差逆伝搬法中間層の処理

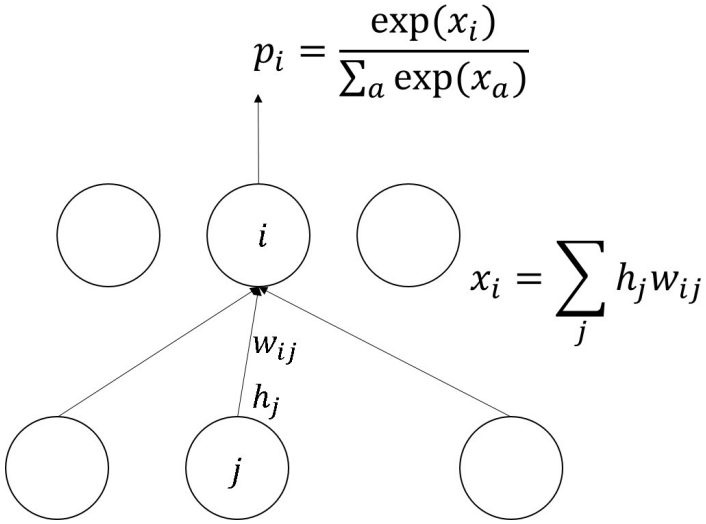

図 2.8: 誤差逆伝搬法出力層の処理

3　Deep Learinig システム構成上の問題点

Deep Learning では大量のデータを行列演算で扱う処理が多いことから，大量のデータを並列で高速に処理できる GPU が用いられてきた．近年では整数演算ベース専用構成の，省電力，レイテンシが小さいなどといった利点を活かした専用ハードウェアの研究も，多少ながら進んでいる．しかし，整数演算ベース専用構成は，GPU と比べて一般的に演算処理能力に劣る．

3.1　GPU による構成

GPU は大量のデータを並列，高速に処理できるという特徴がある．NN のユニットの演算は単純な積和であり，同一層の各ユニットに依存関係はないため，限りはあるが，並列な処理が可能である．GPU は高速な処理が可能だが，浮動小数点しか扱うことができないため，消費電力が増えていることや，レイテンシが大きいなどといった問題点がある．また，従来の GPU はディープラーニング専用構成をとっていないため，並列処理可能部分を並列化しやすい NN を構成する必要があるため，汎用性に欠ける [3].

3.2 ASICによる構成

ASIC（Application Specific Integrated Circuit、特定用途向け集積回路）は消費電力が小さく，レイテンシが小さいという特徴があり，固定小数点を扱うこともできる．固定小数点演算では情報落ちが起こらず，演算を高速にすることができる．初期コストがとてもかかるASICだが，実装面積や，大量生産時のコストを低減するために作られたもので，消費電力，性能，１チップあたりの単価においてGPUより圧倒的なパフォーマンス性能を誇る．GoogleやAppleも注目しており，Googleでは囲碁AIの「AlphaGo（アルファ碁）」などに使用，AppleではiOSデバイス向けに自社のカスタムASICを制作している．しかし，一般的にGPUに比べて演算処理能力に劣る．その低い演算処理能力を補うには高並列化が必要で，低コスト・高並列化の両立にはハードウェア規模低減が必要不可欠である．ASICの専用ハードウェアは構成ゲートの大半を乗算器が占めるため，乗算時の乗数，あるいは被乗数のビット幅を抑えることでハードウェア規模の低減が可能である．近年ではASICの専用ハードウェアを用いた研究は進んでいるものの，必要最低限の固定小数点演算精度がどれだけかの研究は充分には行われていない．

4　提案手法

本研究で使用したCNN(Convolutional Neural Network)は，畳み込み層，プーリング層，全結合層など，計算方法が違う層が多く集まり，構成されている．また，前述の通り，学習フェーズごとに精度の必要な箇所，そうでない箇所が存在する．それらの計算部分ごとに必要最低限のビット幅に切り替える．本研究室の先行研究では，演算ビット幅の動的な切り替えを行っていたが，16bitでの演算の割合が12bitでの演算よりも多くなっているという問題がある．本研究では，12bitでの演算の割合を減らす演算精度低減法を提案し，更なるハードウェア規模低減法を示す．

4.1　層ごとに適したビット切り替え

NNの学習時の演算精度は，16bit幅で充分な認識精度が示されている．しかし，NNには種類の異なる層が存在しており，それぞれ，学習に必要な演算精度が異なる場合がある．そこで，それぞれの層ごとに演算ビット幅を静的に変更し，適したビット幅に操作することで効率化を図る．

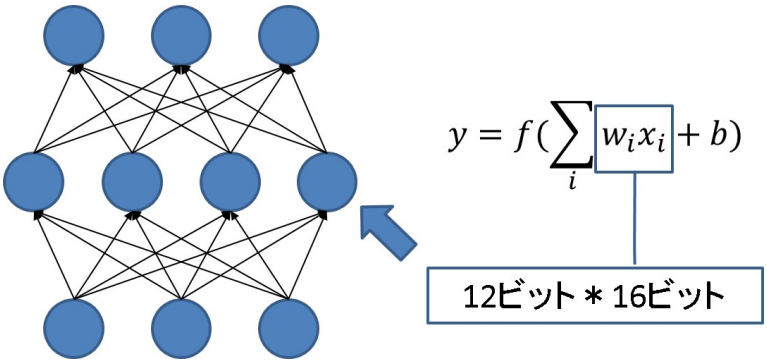

図 4.9: 層ごとに適したビット切り替え

4.2 学習フェーズごとに適したビット切り替え

演算ビット幅を静的に操作し，固定するだけでな，学習の進行によって動的に操作し，更なる効率化を図る．本研究では，勾配降下法に則り、適したビット幅で学習させる．勾配降下法の，勾配の大きい場合は大きく降下，勾配の小さい場合は小さく降下するという特徴を考慮する．具体的には，学習の前半ではビット幅を小さくして粗く学習させ，学習の中盤から後半にかけては細かく学習させる．極小値付近ではさらにビット幅を小さくして再度粗く学習させる.（極小値を超えてしまいそうな場合は，再度細かく学習させる.）閾値は 12 → 16 に操作の場合は最低エラー率が 5epoch 連続で更新されなかった場合，16 → 12 に操作の場合は最低エラー率が 3epoch 連続で更新された場合としている．

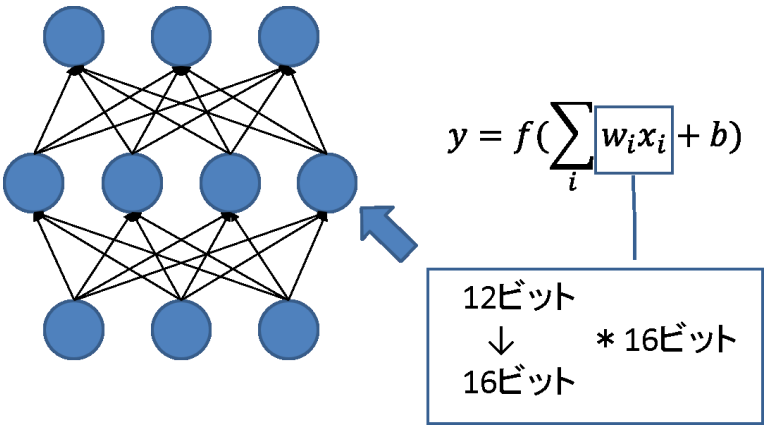

図 4.10: 学習フェーズごとに適したビット切り替え

4.3 実装方法

プログラミング言語 Python と，実行時コンパイルによる高速化，GPU のサポートによる高速化，自動微分などの特徴を持ち，Deep Learning の実装に適している数値計算のライブラリ Theano を用いた．プログラムは Deep Learning Tutorials や先行研究などを参考にし，演算ビット幅を変更できるように修正した．実験は GPU で実行したため，浮動小数点での実行のみしか行えなかった．そこで，演算ビット幅で十分表現できる有効数字に丸めることで代わりとした．しかし，この方法では 1bit 単位での操作ができないという問題がある．

4.4 動的な演算ビット幅切り替えを活かすハードウェア規模低減法

現在，データセンターという分散するIT機器を集約設置し効率よく運用するために作られた共用施設が存在し，さまざまな場面で応用されている．特にインターネット用のサーバや，データ通信，固定・携帯・IP電話などの装置を設置・運用することに特化している建物が多い．サーバを収容する専用のデータセンターでは，効率が悪い夜間・休日の空調運転を抑えることができ，最も少ないエネルギーでサーバを稼働することできる．Deep Learning専用のデータセンターを想定すると，不特定多数のユーザが，そこに設置される多数のアクセラレータを共有することで，それぞれのDeep Learningの高速な実行が可能になる．多数のアクセラレータを，並列に設置可能になるので，演算ビット幅の切り替えが個々のアクセラレータで，動的に演算幅を12bit，16bit間で切り替える代わりに，12bit専用のアクセラレータ，16bit専用のアクセラレータを必要に応じて使用することで実現される．本研究では，先行研究より12bit演算の割合を10 %増やすことができたため，ハードウェア規模低減法を示すことができたといえる．

5　性能評価

5.1　評価方法

性能評価には手書き文字データセットの MNIST を使用した．MNIST は 0〜9 の手書き数字の 28*28 白黒画像 7000 枚で構成される．表 5.1 に今回利用した MNIST の構成を示す．表中の CONV，POOL，FULLY，SOFTMAX，RELU はそれぞれ畳み込み層，プーリング層，全結合層，ソフトマックス層，活性化関数層を，M_i, R_i, C_i, M_o, R_o, C_o は入出力のマップ数，縦幅，横幅を，K_r, K_c はフィルタの縦幅，横幅を表す．全結合層，ソフトマックス層にはフィルタが存在せず，画像ではなく画素値のデータを扱うため，K_r, K_c, R_i, C_i, R_o, C_o は表示していない．

表 5.1: MNIST の構成

MNIST	M_i, R_i, C_i	K_r, K_c	M_o, R_o, C_o
INPUT	-	-	1,28,28
CONV	1,28,28	5,5	20,24,24
POOL	20,24,24	2,2	20,12,12
CONV	20,12,12	5,5	50,8,8
POOL	50,8,8	2,2	50,4,4
FULLY	800,-,-	-	500,-,-
SOFTMAX	500,-,-	-	10,-,-

5.2 結果と考察

演算ビット幅を操作した演算精度 (8 回学習を実行した平均) を表 5.2 に示す．表 5.2 から，ソフトマックス層では演算回数が多くないため，12bit 演算にしてもあまり精度が落ちないことがわかった．そこで，本研究と先行研究では畳み込み層と活性化関数層を動的操作，ソフトマックス層を静的操作でビット幅の割合を示すための実験をした．全学習時間の中の，各演算器ビット幅学習時間の占める割合，その 8 回学習させた内訳とその平均を表 5.3，比較対象として 12bit から 16bit へ 1 度のみ操作する先行研究 [4] の結果を表 5.4 に示す．先行研究と比較し，精度を落とすことなく学習できる 12bit 演算の割合を 10 %ほど高めることができた．また，12bit 演算で極小値辺りの収束を迎える場合は，極小値を超えてしまうおそれがあり，正確な学習ができない．そのため，極小値付近では 16bit 演算で調整し，学習させる必要がある．本実験では GPU で実行したため，浮動小数点しか扱うことができなかった．そのため，有効数字を丸めることで，その値を十分表せるビット幅に丸めたことにしている．小数点以下 4 桁に丸めると 16bit，小数点以下 3 桁に丸めると 12bit として実験を行っている．

表 5.2: MNIST 結果

変更した層 bit	エラー率
CONV 12 → 16 RELU 12 → 16 SOFTMAX 12	1.13 %
CONV 12 → 16 RELU 12 → 16 SOFTMAX 12 → 16	1.16 %
CONV 12 → 16 RELU 16 SOFTMAX 12	1.26 %
CONV 12 → 16 RELU 12 SOFTMAX 12	1.45 %
CONV 12 RELU 12 → 16 SOFTMAX 12	1.25 %
CONV 16 RELU 16 SOFTMAX 16	1.05 %

表 5.3: 学習回数毎の演算ビット利用率 (%)

学習回数		1	2	3	4	5	6	7	8	平均
演算器	12bit	51	30	22	46	39	32	20	28	33.5
ビット幅	16bit	49	70	78	54	61	68	80	72	66.5

表 5.4: 先行研究の学習回数毎の演算ビット利用率 (%)

学習回数		1	2	3	4	5	6	7	8	平均
演算器	12bit	13	31	15	34	37	17	20	16	22.9
ビット幅	16bit	87	69	85	66	63	83	80	84	77.1

6　あとがき

本研究では，演算精度 16bit 以下に抑えることを目標に，認識精度を落とすことなく 12bit で実行可能な演算の割合を増やすことのできる，演算精度低減法を示した．具体的には，それぞれの層に必要なビット幅を明らかにすることで，先行研究よりも 12bit 演算の割合を 10 %以上増加させた．また，本研究の演算精度低減法を用いたハードウェア規模低減法を示した．今後は，実際に整数演算ベース専用構成を設計することで，さらに細かいビット幅変更の実験やハードウェア規模をどこまで削減できるかを検討していく必要がある．また，本論文の実験においては使用した NN のモデルは CNN のみであったが，NN のモデルを変更することで，層の総数と演算回数の比較をし，それぞれの層に必要なビット幅の調査を明確に行うことができる．本論文の実験では，Deep Learning 専用のデータセンターを想定して実験を行ったが，実際に多くのユーザが共有することのできるアクセラレータで学習を試すことができれば，さらなる改善案を示すことができるはずである．

謝辞

本研究を進めるにあたり様々なご指導，ご助言をいただいきました近藤利夫教授，佐々木敬泰助教，深沢祐樹研究員に深く感謝いたします．また，様々な面でお世話になったコンピュータアーキテクチャ研究室の皆様に感謝いたします.

参考文献

[1] 日経コンピュータ, 米 Google が深層学習専用プロセッサ「TPU」公表、「性能は GPU の 10 倍」と主張, ＜ http://itpro.nikkeibp.co.jp/atcl/ncd/14/457163/052001464/＞ (2016/12/10 アクセス)

[2] S.Fupta,*et al.*, ”Deep Learning with Limited Numerical Precision ”,ICML-15,pp.1737-1746,2015.

[3] NVIDIA, ディープラーニング最新動向と技術情報＜ https://images.nvidia.com/content/APAC/events/deep-learning-day-2016 -jp/NVIDIA-DeepLearning-Intro.pdf ＞ (2017/1/30 アクセス)

[4] 小西佑弥, ”ディープラーニングにおける演算ビット幅低減の研究 ”, 三重大学卒業論文,2016 年 3 月

A　プログラムリスト

```
############################
```

#5 回連続最低エラー率の更新がなければ丸め幅変更

```
#########################
                if this_validation_loss >= best_validation_loss:
                    count += 1
                    count2 = 0
                    print "best_error %.3f, this_error %.3f, count %d count12 %
                    if count3 == 1:
                        count16 +=1
                    if count3 == 0:
                        count12 +=1
                else:
                    count = 0
                    count2 +=1
                    print "best_error %.3f, this_error %.3f, count %d count12 %
                    if count3 == 1:
                        count16 +=1
```

```
                    if count3 == 0:
                        count12 +=1
                if count == 5:
                    conv_digit = 4
                    relu_digit = 4
                    print "digit updated"
                    count3 = 1
############################
#3回連続最低エラー率の更新があれば丸め幅変更
########################
                if count2 == 3:
                    if count3 == 1:
                        if count4 == 0:
                            conv_digit = 3
                            relu_digit = 3
                            print "digit rollback"
                            count3 = 0
                            count4 = 1
```

##############################

B　評価用データ

評価用データには手書き数字データセット MNIST を用いた.