卒業論文

題目

ディープラーニングにおける演算ビット幅低減の研究

指導教員

近藤利夫 教授

2016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

小西佑弥(412825)

内容梗概

その認識精度の高さから Deep Learning による認識が注目を集めている．Deep Learning には膨大な学習が必要であり，高速に学習を行うため大量のデータを並列に高速に処理することが可能な GPU が用いられている．しかし GPU では浮動小数点演算しか扱えず，消費電力，レイテンシが大きいという問題がある．そこで近年，より省電力で高速な処理が可能な固定小数点演算に対応する FPGA を利用した専用ハードウェアの研究が行われている．

この固定小数点ベースのハードウェアでは構成ゲートの大半を乗算器が占めるため，ゲート規模がおおよそ乗数，被乗数のビット幅の積に比例して増加する．従って，これら被演算数のビット幅を必要最小限に抑えられればハードウェア規模の低減が可能になる．実際，各層のビット幅固定の条件で，16 ビットまでの低減に成功している．本研究では，1 層のハードウェアの小型化，高速化を目指し，演算ビット幅をニューラルネットワークの層毎に静的及び動的に最小限化する手法を試み，畳み込みニューラルネットワークでは演算ビット幅を全層 16 ビットとする従来手法と比較して，乗算器の規模がおよそ 69%低減されることを明らかにした．

Abstract

Recognition by DeepLearning attracts attention, because of its high recognition accuracy. Lots of learning is necessary for Deep Learning, and GPU which can parallel process a large amount of data to learn it fast is used. However, GPU can process only floating point arithmetic and has a problem that is a large power consumption and high latency. Therefore, in recent years, the dedicated hardware which used FPGA which can process the fixed point arithmetic that low power consumption and high-speed processing are possible than floating point arithmetic is studied. Because a multiplier for most of the configuration gates with this fixed-point-based hardware, the gate scale increase in proportion to the product of the bit width of a multiplier and multiplicand. Therefore, the reduction of the hardware scale is enabled if make the bit width of a multiplier and multiplicand into necessity minimum. Actually, under conditions of the bit width fixation of each layer, there is a study succeeding for reduction to 16 bits. In this study, aiming at reduced and high-speed hardware, make the operation bit width into a necessity minimum in every layer of the neural network statically and dynamically. In Convolutional Neural Network, it is shown that reduce the multiplier scale 69% in comparison with the conventional technique that operation bit width was 16 bits in all layers.

目 次

1 はじめに **1**
1.1 背景 1
1.2 研究目的 2

2 ニューラルネットワークと Deep Learning の概要 **3**
2.1 ニューロンモデル 4
2.2 多層パーセプトロン Multi-Layer Perceptron 5
2.3 畳み込みニューラルネットワーク Convolutional Neural Network 6
2.3.1 畳み込み層 6
2.3.2 プーリング層 7
2.4 Deep Learning 8
2.4.1 勾配降下法 9
2.4.2 誤差逆伝播法 10

3 Deep Learning システム構成上の問題点 **13**
3.1 GPU による構成 13
3.2 FPGA による構成 14

4 提案手法 **15**
4.1 静的ビット切り詰め 15
4.2 動的ビット切り詰め 16
4.3 実装 17

5 性能評価 **19**
5.1 データセット 19
5.2 評価 19
5.3 考察 19

6 おわりに **22**
6.1 まとめ 22

謝辞 **23**

参考文献 **23**

A プログラムリスト **25**
A.1 MNIST のプログラム変更点 25
A.2 CIFAR-10 プログラム変更点 28

B 評価用データ **32**

図 目 次

2.1 ニューロンモデル 4
2.2 多層パーセプトロン 5
2.3 畳み込み処理例 6
2.4 プーリング処理例 7
2.5 誤差逆伝播法の出力層での計算 10
2.6 誤差逆伝播法の中間層での計算 10
4.7 静的なビット切り詰めの例 15
4.8 静的なビット切り詰めのエラー率の推移例 15
4.9 動的なビット切り詰めの例 16
4.10 動的なビット切り詰めのエラー率の推移例 16

表 目 次

4.1 MNIST の構成 18
4.2 CIFAR-10 の構成 18
5.3 MNIST 結果 20
5.4 CIFAR-10 結果 20

1　はじめに

1.1　背景

文字認識をはじめとする物体認識法の一つに，ニューラルネットワーク（以下 NN）がある．NN の研究は 1940 年代から行われており，最近提案されたプレトレーニング，RBM といった学習手法や生成モデルにより多層の NN の学習が可能になった．この多層の NN を用いた機械学習の方法論を Deep Learning と呼ぶ．検索エンジンや音声認識等で利用されており将来的には自動車の自動運転技術や産業用ロボットの作業の効率化等への応用も期待されている．しかし，Deep Learning には膨大な学習が必要であり，大量の学習用データ確保や学習時間の長大さが問題視されている．このため Deep Learning には，これまで大量のデータを並列に処理できる GPU が用いられてきた．しかし，近年は消費電力やレイテンシの低減のため FPGA による専用ハードウェアの研究も進んでおり，GPU と同程度の認識性能，速度で消費電力がおよそ半分の FPGA ハードウェアが実現されている．

1.2 研究目的

GPU では浮動小数点演算しか扱えないのに対して，FPGA では低消費電力の固定小数点演算が扱えるからである．固定小数点用のハードウェアの規模は構成ゲートの大半を乗算器が占めるため，おおよそ乗数，被乗数のビット幅の積に比例して増加する．従って，これら被演算数のビット幅を必要最小限に抑えられればハードウェア規模の低減が可能になる．当然，NN の学習に必要なビット幅の研究は行われてきているが高々16 ビット程度までの低減に留まっている [1]．本研究では演算ビット幅を NN の層毎に最小限化し，平均でどこまで低減できるかを明らかにする．評価は多層パーセプトロン（Multi-Layer Perceptron：MLP）と畳み込みニューラルネットワーク（Convolutional Neural Network：CNN）の両者を比較して行う．

2　ニューラルネットワークとDeep Learningの概要

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ており何度かのブームと終焉を繰り返してきた．1980年代にはNNの学習方法である誤差逆伝播法の発見によりブームが起きた．しかし，このブームは90年代後半頃には次の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）が可能になり，2012年には一般物体認識コンテストILSVRCでDeep Learningの手法が従来の手法に大きく差をつけて優勝した．NNの構造には多層パーセプトロン，畳み込みニューラルネットワーク等の種類がある．本章ではニューラルネットワークの構造と学習について簡単に説明する．

2.1 ニューロンモデル

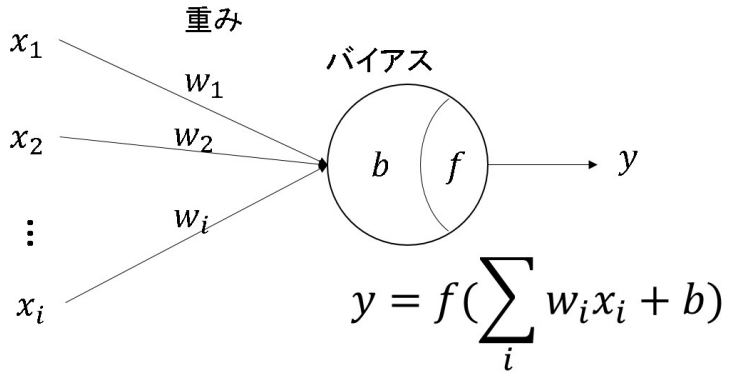

図 2.1: ニューロンモデル

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り，図 2.1 のように入力と重みの積和にバイアスを加算したものに活性化関数を適用し出力

$$y = f(\sum_i w_i x_i + b) \tag{1}$$

を求める．活性化関数には

$$\text{ロジスティック関数} \qquad f(x) = \frac{1}{1+\exp(-x)} \tag{2}$$

$$\text{双曲線正接関数} \qquad f(x) = tanh(x) \tag{3}$$

$$\text{打ち切り線形関数} \qquad f(x) = max(0, x) \tag{4}$$

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打ち切り線形関数が一般的に用いられている．

2.2 多層パーセプトロン Multi-Layer Perceptron

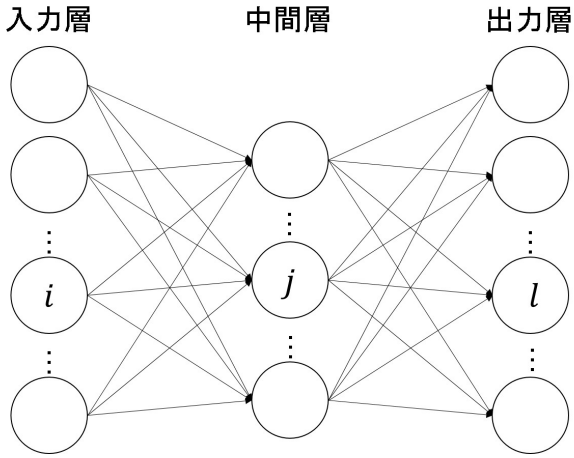

図 2.2: 多層パーセプトロン

多層ニューラルネットワークには構成によっていくつかの種類がある．本研究ではMLP およびCNN を使用している．MLP は非線形の多クラス識別器であり，図2.2 のようにユニットを階層状に並べすべてのユニットを結合したもので，入力層，中間層，出力層で構成されている．第 k 層 i 番目の出力 h_i^k はベクトルを用いて

$$h_i^k = f(b_i^k + \boldsymbol{w}_i^{kT}\boldsymbol{h}^{k-1}) \tag{5}$$

で表される．出力層では活性化関数としてsoftmax 関数

$$p_i = softmax_i(w_i x_i + b_i) = \frac{\exp(w_i x_i + b_i)}{\sum_j \exp(w_j x_j + b_j)} \tag{6}$$

が用いられる．出力は入力がそのクラスに属する確率となり，出力層において最大の値を与えるユニットのクラスが入力のクラスとなる．

2.3 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，プーリング層を繰り返した後に全結合層の構造を持つ．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している．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る．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．その後，全結合層に繋ぎ出力層では softmax 関数を用いる．

2.3.1 畳み込み層

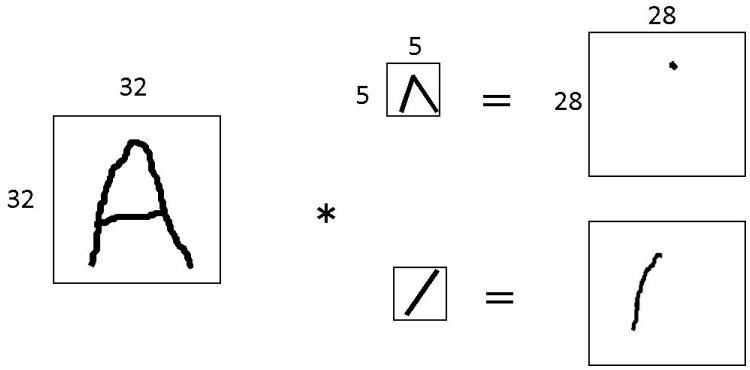

図 2.3: 畳み込み処理例

畳み込み層では図 2.3 のように，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．画像，フィルタサイズをそれぞれ $n_x \times n_y$, $n_w \times n_w$ としたとき，特徴マップのサイズは $n' = n_x - n_w + 1, n'_y = n_y - n_w + 1$ となる．

2.3.2 プーリング層

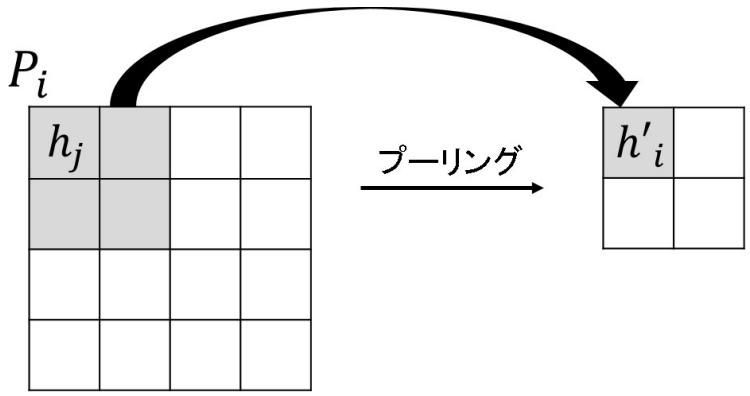

図 2.4: プーリング処理例

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を言う．ここでは図2.4 のように，画像の解像度を落とすことで余分な情報を捨てている．プーリングにはいくつかの種類があり，平均プーリング，マックスプーリング，Lp プーリング等が存在する．

$$\text{平均プーリング} \qquad h'_i = \frac{1}{|P_i|} \sum_{j \in P_i} h_j \tag{7}$$

$$\text{マックスプーリング} \qquad h'_i = \max_{j \in P_i} h_j \tag{8}$$

$$Lp\,\text{プーリング} \qquad h'_i = (\frac{1}{|P_i|} \sum_{j \in P_i} h_j^P)^{\frac{1}{P}} \tag{9}$$

平均プーリングとは式 (7) のように，$k-1$ 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．マックスプーリングとは式 (8) のように，$k-1$ 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．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式 (9) の Lp プーリングがある．

2.4 Deep Learning

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．望みの出力と実際の出力との差は交差エントロピー

$$C = \sum_i d_i \log p_i \tag{10}$$

で測る．ここで 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行われる．このときに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．次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．

$$\begin{aligned} w_{ij} &\leftarrow w_{ij} + \Delta w_{ij} \\ \Delta w_{ij} &= -\epsilon \frac{\partial C}{\partial w_{ij}} \end{aligned} \tag{11}$$

ϵ は学習係数と呼ばれ学習の幅を決める．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．

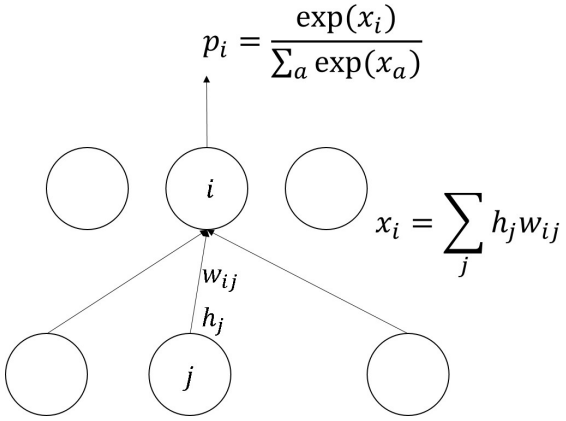

図 2.5: 誤差逆伝播法の出力層での計算

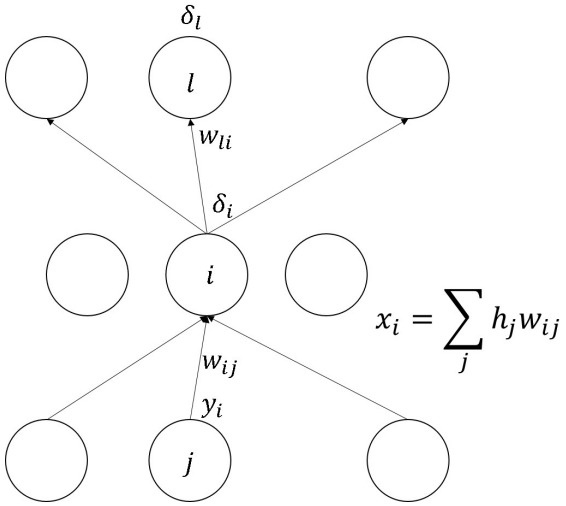

図 2.6: 誤差逆伝播法の中間層での計算

2.4.2 誤差逆伝播法

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，式 (10)，図 2.5 の様に NN の最終出力に対して定義され,

この出力層の重み w_{ij} に関する勾配は,

$$\frac{\partial C}{\partial w_{ij}} = (p_i - d_i)h_j \tag{12}$$

で求められる．各層の出力はは直前の層の出力の関数になっており，各パラメータは f(f(f $\cdots$)) のような活性化関数の入れ子の形になり計算が難しい．そこで誤差逆伝播法が利用される．図 2.6 のように，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

$$x_i = \sum_j w_{ij} h_j \tag{13}$$

と書く．このとき，図 2.6 で，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

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial x_i}\frac{\partial x_i}{\partial w_{ij}} \tag{14}$$

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

$$\frac{\partial x_i}{\partial w_{ij}} = h_j \tag{15}$$

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 $x_l (l = 1, 2, ...)$

の関数であるとみることもでき，微分の連鎖法則により，δ_i は

$$\delta_i = \frac{\partial C}{\partial x_i} = \sum_l \frac{\partial C}{\partial x_l}\frac{\partial x_l}{\partial x_i} \tag{16}$$

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

$$\frac{\partial x_l}{\partial x_i} = f'(x_i) w_{li} \tag{17}$$

と計算され，以上をまとめると

$$\delta_i = f'(x_i) \sum_l \delta_l w_{li} \tag{18}$$

と書ける．この式より図 2.6 の一番上の層の δ_l が与えられると，中央の δ_i を計算できる．図 2.6 の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 (18) 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 が得られれば，式 (14) より，欲しかった勾配 $\frac{\partial C}{\partial w_{ij}}$ は

$$\frac{\partial C}{\partial w_{ij}} = \delta_i h_j \tag{19}$$

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，これは誤差逆伝播法と呼ばれる．

3　Deep Learningシステム構成上の問題点

これまで，Deep Learningは大量のデータを行列演算で扱う処理が多いと言う特徴から，同じく大量のデータを並列に高速で処理できるGPUが用いられてきた．しかし近年では，FPGAによる専用ハードウェアの研究が進んでいる．理由として，FPGAはGPUに比べ省電力，レイテンシが小さい，さらなる高速化のため等が挙げられる．しかし一般に，FPGAは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に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無く並列に処理できる．高速な処理が可能だが，消費電力が大きい，レイテンシが大きい，浮動小数点しか扱えないという問題がある．マルチGPUという複数台のGPUを用いる技術もあるが，並列処理できる部分が限られており，高速化するにはマルチGPUを意識して並列化しやすいようなNNを構成する必要がある．

3.2 FPGA による構成

GPU に比べ演算処理能力に劣るが消費電力，レイテンシが小さい，固定小数点が扱えるという利点がある．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・高並列の両立にはハードウェア規模低減が不可欠である．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られればハードウェア規模の低減が可能である．しかし，必要最低限の固定小数点演算精度がどれだけかの研究が十分行われているとは言えない状況である．

4　提案手法

4.1　静的ビット切り詰め

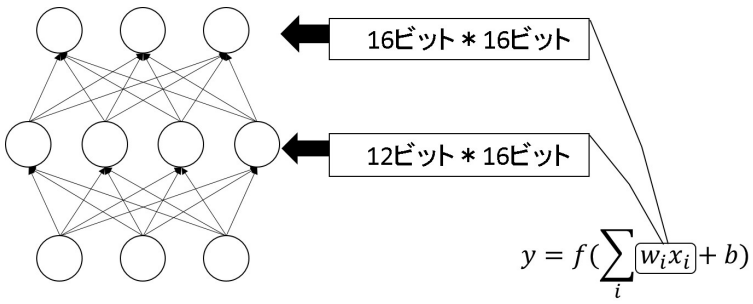

図 4.7: 静的なビット切り詰めの例

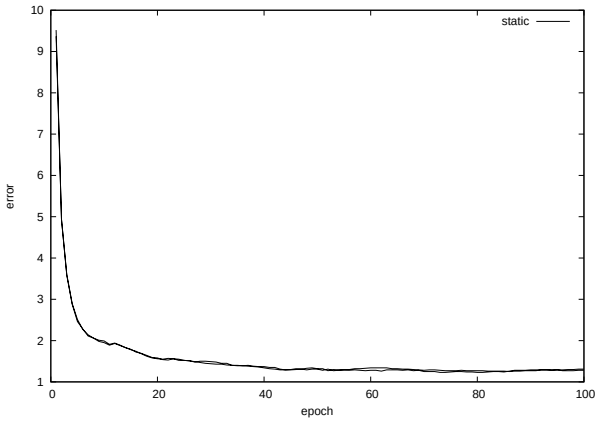

図 4.8: 静的なビット切り詰めのエラー率の推移例

学習時の演算精度は 16 ビット幅で十分な認識精度が発揮されるという報告 [1] がされている．NN の層にはいくつかの種類があり，それぞれ必要な演算精度が異なる可能性がある．そこで，NN 全体の演算ビット幅をまとめて操作するのではなく，層毎にそれぞれ操作し適したビット幅に

丸めることでさらなる効率化を図る．図 4.7 に示すように NN の層毎に乗算器の演算ビット幅を丸める．学習の回数とエラー率の推移は図 4.8 のようになる．

4.2 動的ビット切り詰め

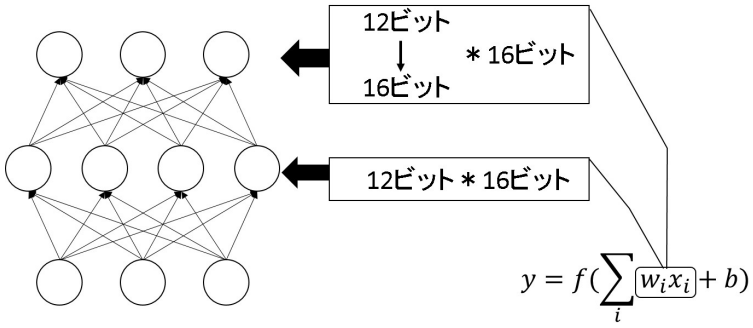

図 4.9: 動的なビット切り詰めの例

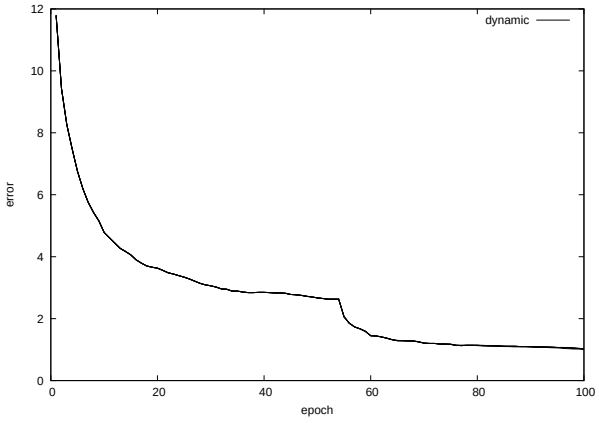

図 4.10: 動的なビット切り詰めのエラー率の推移例

演算ビット幅を静的に固定するのではなく学習の進行によって大きさ

を操作することでさらなる効率化を目指す．図 4.9 に示すように NN の層毎に乗算器の演算ビット幅を動的に丸める．勾配降下法の勾配の大きい場合は大きく降下し，極小値に近づき勾配が小さくなってくると小さく降下するという特徴を考慮し，学習の前半ではビット幅を小さくし粗い学習を行い，学習の後半ではビット幅を大きくし細かい学習を行う．この閾値は最低エラー率の更新が 5epoch 連続で起きなかった場合とする．学習の回数とエラー率の推移は図 4.10 のようになる．

4.3 実装

プログラミング言語 Python とそのライブラリ Theano を用いて，手書き文字データセット MNIST，一般物体データセット CIFAR-10 に対して実験を行った．プログラムは Deep Learning Tutorials[2] 等を参考に演算ビット幅を変更できるよう改造し使用した．なお実験には GPU を用いたため演算ビット幅を調整できなかった．そのため，その演算ビット幅で十分表現できる有効数字桁に丸めることで代わりとした．使用した NN の構成は表 4.1，表 4.2 に示す．表中の CONV，POOL，FULLY，SOFTMAX 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 4.1: 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,-,-

表 4.2: CIFAR-10 の構成

CIFAR-10	M_i, R_i, C_i	M_o, R_o, C_o
INPUT	-	3,32,32
FULLY	3072,-,-	1000,-,-
FULLY	1000,-,-	500,-,-
SOFTMAX	500,-,-	10,-,-

5　性能評価

5.1　データセット

性能評価には MNIST，CIFAR-10 の 2 種のデータセットを使用した．MNIST[3] は手書き数字のデータセットで 0〜9 の手書き数字の 28*28 白黒画像 70000 枚で構成される．CIFAR-10[4] は一般物体のデータセットで飛行機，自動車，鳥，猫，鹿，犬，蛙，馬，船，トラックの 10 種の 32*32 カラー画像 60000 枚で構成される．

5.2　評価

表 4.1，表 4.2 の構成で，演算ビット幅を静的もしくは動的に操作したときの平均ビット幅，認識精度を表 5.3，表 5.4 に示す．表中の→はビット幅を動的に操作したことを表す．符号ビットを含む整数部には，DATE では 6 ビット，層のパラメータでは 2 ビットを割り当てている．比較のため 32 ビット浮動小数での結果も示している．

5.3　考察

表 5.3 で示すように CNN の構成では DATA のビット幅を 6 ビット，つまり，整数に丸めても認識精度は落ちていない．それに対して，表 5.4 の MLP では認識精度が落ちてしまい，うまく演算ビット幅を小さくできて

表 5.3: MNIST 結果

CONV(bit)	32	16	12 → 16	12	12 → 16	12	16	12 → 16
FULLY(bit)	32	16	12 → 16	12 → 16	12	12	16	12 → 16
SOFTMAX(bit)	32	16	12	12 → 16	12 → 16	12	16	12
DATA(bit)	32	16	16	16	16	16	6	6
演算平均 (bit)	32	16	15.2	15.2	15.2	14	11	9.0
精度 (%)	99.0	98.9	98.8	98.7	98.4	98.4	98.9	98.7

表 5.4: CIFAR-10 結果

FULLY(bit)	32	16	12 → 16	12 → 16	12	16
SOFTMAX(bit)	32	16	12 → 16	12	12 → 16	16
DATA(bit)	32	16	16	16	16	6
演算平均 (bit)	32	16	15.5	15.5	14.0	11
精度 (%)	56.1	55.6	54.6	54.2	46.8	22.4

いない．この違いは，CNN の構成では畳み込み層，プーリング層を用いて特徴の抽出を行っているため，端数を丸めてもそれほど影響を及ぼさないからだと考えられる．CNN については演算ビット幅が最大で 9 ビットまで削減できており，単純に 16 ビットと比較すると乗算演算はおよそ 69%低減できている．各層のパラメータについては，どちらの結果においても全結合層のビット幅は丸めると精度が落ちてしまうが，ソフトマックス層についてはそこまで精度は落ちないという結果になっている．しかし，ソフトマックス層は演算数が少ないためそこまで高速化には繋がら

ない．また，本研究では浮動小数点で固定小数点を実現した．浮動小数点で正確な固定小数点を表現しようとすると変換に時間がかかる．そこで、時間の関係上，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た．小数点以下 4 桁に丸めると 16 ビット，小数点以下 3 桁に丸めると 12 ビットとして実験を行った．この方法では 1 ビット単位での細かな実験が行えないという欠点がある．これは、実際に FPGA で設計し実験を行うことで解決できる．

6　おわりに

6.1　まとめ

本研究では Deep Learning の FPGA による専用ハードウェアの規模削減のため，演算ビット幅を層毎に静的及び動的に操作した．CNN での MNIST の認識ではデータを整数に丸めても認識できることがわかった．結果，平均で 9 ビットまでの低減に成功した．

実際に FPGA で設計することで，さらに細かい実験やどこまで規模を削減できるかの確認が今後の課題である．

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] S.Fupta，A.Agrawal，K.Gopalakrishnan and P.Narayanan,“ Deep Learning with Limited Numerical Precision“ ，ICML-15，pp.1737–1746，Feb 2015.

[2] Deep Learning「Deep Learning Tutorials」，http://deeplearning.net/tutorial/，2016 年 1 月 29 日アクセス.

[3] MNIST handwritten digit database「THE MNIST DATABASE of handwritten digits」，http://http://yann.lecun.com/exdb/mnist/，2016 年 2 月 29 日アクセス.

[4] CIFAR-10 and CIFAR-100 datasets「The CIFAR-10 dataset」, https://www.cs.toronto.edu/ kriz/cifar.html, 2016年2月29日アクセス.

A　プログラムリスト

A.1　MNIST のプログラム変更点

```
#####################
#各層での出力を得る
#####################
  train_layer0 = theano.function(
      [index],
      layer0.output,
      givens={
          x: train_set_x[index * batch_size: (index + 1) * batch_size],
          })
  train_layer1 = theano.function(
      [layer1_input],
      layer1.output.flatten(2)
      )
  train_layer2 = theano.function(
      [layer2_input],
      layer2.output,
```

```
        )
    train_layer3 = theano.function(
        [index, layer3_input],
        layer3.negative_log_likelihood(y),
        updates=updates,
        givens={
            x: train_set_x[index * batch_size: (index + 1) * batch_size],
            y: train_set_y[index * batch_size: (index + 1) * batch_size]
            })
####################
#各層での丸め
####################
    for i in xrange(8):
        #softmax 層
        if i == 0:
     params[i].set_value(np.round(params[i].get_value(), softmax_digit))
        #ReLu 層
        elif i == 2:
```

```
                params[i].set_value(np.round(params[i].get_value(), relu_digit))
            #conv層
            elif i == 4 or i == 6:
                params[i].set_value(np.round(params[i].get_value(),conv_digit))


    ####################
    #動的操作
    ####################
if this_validation_loss >= best_validation_loss:
    count += 1
                print "best_error %.3f, this_error %.3f, count %d"%(best_validatior
            else:
                count = 0
                print "best_error %.3f, this_error %.3f, count %d, best_error updat
if count == 5:
                conv_digit = 3
                relu_digit = 3
                softmax_digit = 3
```

```
            print "digit updated"
```

A.2 CIFAR-10プログラム変更点

```
######################
#動的操作
######################
    this_error = 1 - erV
    if this_error >= best_error:
        count += 1
        print "best_error %.3f, this_error %.3f, count %d"%(best_error, this_er
    else:
        count = 0
        print "best_error %.3f, this_error %.3f, count %d ,best_error update"%(
        best_error = this_error
    if count == 5:
        softmax_digit = 4
        relu_digit = 4
        print "softmax_digit,relu_digit = 4"
```

```
####################
#各層での丸め
####################
 for i, layer in enumerate(mlp.Layers):
     #softmax
     if i == 2:
         layer.W.set_value( np.round( layer.W.get_value(), softmax_digit ) )
         layer.b.set_value( np.round( layer.b.get_value(), softmax_digit ) )
     #ReLu
     else:
         layer.W.set_value( np.round( layer.W.get_value(), relu_digit ) )
         layer.b.set_value( np.round( layer.b.get_value(), relu_digit ) )


####################
#各層で出力を得る
####################
    def training( self, XL, tL, data_digit ):
```

```
X    = T.dmatrix( 'X' )

t    = T.dmatrix( 't' )


Y0, Z0 = self.Layers[0].output( X )

Y1, Z1 = self.Layers[1].output( Z0 )

Y2, Z2 = self.Layers[2].output( Z1 )

cost = T.mean( _T_cost( Z2, t ) )


updatesList = []

for layer in self.Layers:

    gradW = T.grad( cost, layer.W )

    Wnew = layer.W - 0.1 * gradW

    updatesList.append( ( layer.W, Wnew ) )


    if layer.withBias:

        gradb = T.grad( cost, layer.b )

        bnew = layer.b - 0.1 * gradb
```

```
        updatesList.append( ( layer.b, bnew ) )


train_layer0 = theano.function( [X], Z0 )

train_layer1 = theano.function( [Z0], Z1 )

train_layer2 = theano.function( [Z1, X, t], cost, updates = updatesList


output_layer0 = train_layer0( XL )

output_layer0 = np.round( output_layer0, data_digit )


output_layer1 = train_layer1( output_layer0 )

output_layer1 = np.round( output_layer1, data_digit )


cost = train_layer2( output_layer1, XL, tL )



return cost
```

B　評価用データ

MNIST，CIFAR-10 の 2 種のデータセットを評価用データとして用いた.