特集
Special Feature

［知能コンピューティング― AI とハードウェアの出会い―］

4 機械学習に適したハードウェア・ハードウェアに適した機械学習アルゴリズム

高前田伸也　東京大学

なぜコンピュータアーキテクチャの研究者が機械学習の研究に取り組むのか

この特集は「AI とハードウェアの出会い」というお題目をいただいて執筆するものであるが，この章ではまず「コンピュータアーキテクチャの研究者と AI の出会い」について述べたいと思う．筆者の主な研究分野はコンピュータアーキテクチャである．コンピュータアーキテクチャとは，計算速度，電力効率，信頼性，コスト，使いやすさなど，さまざまな指標で「優れた」コンピュータの構成とその実現方法を追求する分野である．ところが，すべての指標で優れた方式を実現することは容易ではなく，こちらを立てればあちらが立たないことが多い．たとえば，高速化のために並列ハードウェアを導入すれば，当然のように消費電力は増え，さらに並列化のオーバヘッドで電力効率は低下する．また，並列ハードウェアを使いこなすためには，並列プログラミングが必要となり使いやすさが低下する．だからこそ，複数の指標を並立する，「チラチラと見え隠れする，針の穴のような突破口を突く」[1]ことの喜びが大きい研究分野であると個人的には思う．多くの大学の情報系学科におけるコンピュータアーキテクチャの講義（たとえば，筆者が所属する東京大学理学部情報科学科であれば「計算機構成論」）では，マイクロプロセッサ（いわゆる CPU）の基本的な仕組みや，プログラムを効率的に処理するための方式や概念，たとえばパイプライン，キャッシュメモリ，分岐予測，命令レベル並列性，アウトオブオーダ実行といった事柄が教示されていると思われる．これらの講義で取り扱われる CPU の高度化技術は，アプリケーションごとに得意不得意は多少あるものの，特定のアプリケーションに限定されるものではなく，プログラム全般を効率的に処理するための汎用的なものである．コンピュータアーキテクチャの研究も同様に，CPU を中心とした汎用的な技術に関するものが主流で，優れた CPU の内部構成方式（マイクロアーキテクチャ）に関する研究は花形であると個人的に思う．

ところが，コンピュータアーキテクチャとその周辺分野の国際学会においては，今回のテーマでもある「AI のためのハードウェア」に関する発表が多く見受けられるようになってきた．たとえば，2021 年 10 月に開催されたコンピュータアーキテクチャのトップ会議の 1 つである MICRO（International Symposium on Microarchitecture）2021 では，21 の論文発表セッションのうち，4 つがアプリケーション・ドメインに特化したアクセラレータに関するものであり，その多くの論文は機械学習処理の効率化に関するものである（なお「マイクロアーキテクチャ」と冠したセッションは 2 つのみであった）．FPGA（Field Programmable Gate Array，利用者が回路構成を変更可能なやわらかいデジタル集積

回路）の原理と利活用に関する国際学会においても，従来は多様性に富んだアプリケーションの高速化事例が発表されていたが，最近はディープラーニングの高速化や軽量化に関する論文の割合が増している．なぜ，コンピュータアーキテクチャの研究者がここまで活発に機械学習に関する研究を取り組むようになったのであろうか．

理由 1：ディープラーニングのための計算能力の要求

理由の 1 つは，ディープラーニングの急速な普及により，より高速かつ高効率にその計算を行うコンピュータシステムの必要性が高まってきたからである．ディープラーニングが普及した理由の 1 つとしてコンピュータの高速化が挙げられるが，その中心となったハードウェアは GPU（Graphics Processing Unit）である．名前のとおり，元々 GPU は画像処理のためのハードウェアであったが，大量の演算器で構成される並列アーキテクチャにより行列計算を CPU よりも高効率に処理できるため，たちまちディープラーニングでは欠かせないハードウェアとなった．

一方で，優れたディープラーニングモデルの学習と推論には大きな計算コストを要することが問題視されはじめている．たとえば，OpenAI が開発した自然言語処理向けのディープラーニングモデルである GPT-3 は，その学習には単一 GPU 換算で 355 年分という膨大な計算コストを必要とすると知られている [2]．また，ディープラーニングモデルの Transformer の学習に要するエネルギー量を二酸化炭素量に換算すると，乗用車 5 台分の製造から廃棄までの間に排出する二酸化炭素量に匹敵すると言われている [3]．今後の持続的なディープラーニング技術の進展には，エネルギー効率に優れる環境負荷の少ないコンピュータシステムの創出が強く求められている．そこで，GPU よりもさらにディープラーニングの計算を高効率に処理可能なハードウェアが求められるようになってきた．有名な例では，Google は TPU（Tensor Processing Unit）と呼ばれるディープラーニングに特化した専用チップを独自に開発し，実際のサービスで活用している．Microsoft は，多数の FPGA で構成される Brainwave と呼ばれるディープラーニングプラットフォームを運用している．また，大企業以外でも DNN に特化したハードウェアの開発の事例が見られる．たとえば，Cerebras はウェハースケールの巨大なチップによるディープラーニングシステムを提供するなど，競争の激しい領域である．このように，機械学習，特にディープラーニングの計算を高速・高効率に処理するためのコンピュータシステム・ハードウェアへの要求が高まるのに伴い，そのための研究が活発に行われるようになった．

理由 2：マイクロプロセッサの性能向上の困難さ

関連する別の理由としては，従来のマイクロプロセッサの改善だけでは要求される効率を達成することが困難になってきたことが挙げられる．マイクロプロセッサは汎用であるが故，どのようなアプリケーションに対してもそれなりの高速性を達成するためのハードウェア機構を有しており，そこに多くのトランジスタと電力を消費している．しかし，ポラックの法則と呼ばれる，投入するトランジスタの数を増やし，マイクロプロセッサのコアに複雑なハードウェア機構を投入したとしても，得られる性能は複雑性の平方根程度にしかならない，という悲観的な法則が知られている．もちろん，マイクロプロセッサの研究者と開発者は，複雑性が低く性能向上の利得が大きいハードウェア方式の研究開発を今も進めており，ありがたいことにその性能は伸び続けているが，針の穴のような突破口を新たに見つけては突き続けるような苦労があると思われる．また，これまでのマイクロプロセッサの性能は，半導体の集積密度が 2 年間で 2 倍にとなるというムーアの法則に牽引され伸び続けてきた．これまでは，関係各位の努力によりなんとかトランジスタの微細化を継続してきたが，物理的および産業的な理由により，その限界が近いという見方が強い．つまり，多くの

トランジスタと電力を投入したとしても，それに見合った性能向上が得にくく，その上，トランジスタの微細化すら期待できなくなるという状況で，コンピュータの性能を向上させ続けるという難しい課題に立ち向かわなければならない．ところが，後述のとおり，機械学習，ディープラーニング計算の効率化という目的に対しては，マイクロプロセッサの汎用性は必ずしも必要ではないため，その汎用性を捨て，ディープラーニングに特化したシンプルかつ高並列なハードウェアにトランジスタと電力を投入するアーキテクチャへとシフトするのは自然である．

理由 3：面白いから

もう 1 つの理由は，単純に「面白い」からではないかと筆者は考えている．従来のマイクロプロセッサに関する研究は，命令セットアーキテクチャと呼ばれるハードウェアがソフトウェアに提供するインタフェースを汎用に保ったまま変更することなく，ハードウェアのマイクロアーキテクチャの工夫により，速度や電力効率を改善することを目指すものが主流であった．ソフトウェア側から見れば，ソフトウェアをほとんど変更することなく高速化の恩恵を受けることができ，同じソフトウェアをそのまま異なるプロセッサ上で実行できるポータビリティが提供される点で好ましいアプローチである．一方，ハードウェア側から見れば，汎用性の死守という大きな制約の下で戦うことを強いられる．もちろん，制約があるからこそ面白いという見方ができるが，できることが限定されてしまうのは事実である．

ところが，ディープラーニング自体はさまざまな用途で利用できる「汎用的な」技術であるが，そのディープラーニングというドメインに絞ることで，汎用性の死守という足かせは大幅に緩和される．ディープラーニングにおけるプログラミングは，モデルの計算グラフを記述し，学習データによりモデルの振る舞いを規定するという，データドリブン・プログラミングと捉えることができる．利用者からすれば，記述したディープラーニングモデルに対応する計算が，期待どおりかつ高効率に行われれば，CPU だろうが GPU だろうが専用チップだろうが構わないのである．もちろん，従来の命令セットアーキテクチャに縛られる必然性はまったくない．つまり，死守すべき汎用性のレイヤを，従来の命令セットアーキテクチャから，ディープラーニングというレイヤに移すことで，従来とは異なるハードウェアとソフトウェアの間のインタフェースの定義が許容されやすくなる．そして，コンピュータアーキテクチャの研究が，より自由な発想が強く求められるようになり，今までよりも面白いものになったと言える．

コンピュータアーキテクチャの研究者が行う機械学習の研究

コンピュータアーキテクチャの研究者が行う機械学習研究は，理論的な側面よりもシステム化を意識したものが多い．たとえば，機械学習，特にディープラーニングの計算に特化したハードウェア・回路の構造や，ハードウェアを意識したディープラーニングのアルゴリズムに関するものである．前者は，畳み込みなどの計算パターンの特性を活かした演算器やメモリシステムの構成を採用することで，データの読み書きや移動の量を削減し，与えられたディープラーニングのモデルの計算を短時間かつ少ないエネルギーで処理できるようにするものである．後者は，演算や値の冗長性や類似性を活用した計算省略手法や，量子化や枝刈りといったモデル表現の簡略化に関するものが多数見受けられる．より優れたディープラーニングシステムの実現には，前者と後者を併せて，ディープラーニングの計算の特徴を捉えたハードウェアの仕組みと，ハードウェアにおける得失を考慮したディープラーニングのアルゴリズムやモデルを同時に考えることが好ましい．筆者はコンピュータアーキテクチャの専門家（のはず）であるため，コンピュータアーキテクチャの知見を活かして，ディープラーニングの計算アルゴリズムを変化させたときにハードウェアにどのような変化

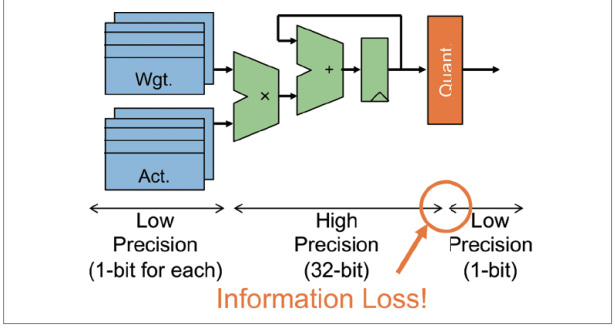
が起こるかを想像して，ハードウェアにとっては負担が大きくないにもかかわらず，精度向上の恩恵のあるアルゴリズムに関する研究に取り組んでいる．前置きが長くなったが，以下，筆者がこれまでに取り組んできた，ハードウェア指向のディープラーニングに関する研究についていくつか紹介する．

Dither NN/DeltaNet：回路中の一時情報を活用する二値化ニューラルネットワークの高精度化手法

GPU を用いたニューラルネットワークの計算では，モデルのパラメータ（ウェイト，重み）と活性値（アクティベーション）を浮動小数により表現し，計算に用いることが一般的である．浮動小数の代わりに整数を用いてこれらを表現することで，ハードウェア・回路を小型化と低消費電力化しつつ，適切なビット幅を選定することで浮動小数と比較してもほぼ変わらない精度が達成できることが知られている．このように数値表現を整数で近似することを量子化（Quantization）と呼ぶ．この中でも，1 つの数値を 1 ビットのみを用いて表現することを二値化（Binarization）と呼ぶ．メモリ量を削減することができ，乗算を単一の Exclusive NOR（XNOR）で実現できるため回路面積および消費エネルギーの点で優れており，最小のビット幅で数値を表現している点が美しいと筆者は感じる．しかし，表現可能な数値状態の数に制限があるため，達成可能な認識精度が低いことが知られている．ここで，推論ハードウェアに入出力される数値は 1 ビットに制限したまま，ハードウェアに内在する多ビットの数値情報を活用することで，要求メモリ量とハードウェア複雑性を増やすことなく認識精度を向上させるハードウェア指向のニューラルネットワークのアルゴリズムの Dither NN[4] と DeltaNet を紹介する．

Dither NN と DeltaNet は共通して，入出力の値を 1 ビットで表現しても，ハードウェア内部には多ビットの数値表現が必ず出現し，それらは最終的には捨てられていることに注目して，捨てられる情報を低コストに活かして二値化ニューラルネットワークの精度を向上するハードウェア指向のアルゴリズムである．図 -1 に二値化ニューラルネットワークのアクセラレータ回路の簡略化した構成を示す．メモリから読み出された値の各ビットが活性値や重みに対応し，ビット値 '1' が +1，ビット値 '0' が -1 に対応する．このとき，+1/-1 のみが入力される乗算は論理演算の XNOR で実現され，積算は乗算結果の popcount（'1' の数の数え上げ）に対応する．このとき，popcount の結果を保持するためには複数ビットの加算回路とレジスタが必要となる．二値化ニューラルネットワークにおける活性化関数には Sign 関数（入力が正なら +1，負なら -1 を返す関数．バイナリ表現の場合には入力が正なら 1，負なら 0 を返す）を用いることが多く，popcount の結果と閾値との大小比較で実現され，最終的に 1 ビットの値が出力され，適宜メモリに書き込まれる．つまり，メモリ上に配置される値には 1 ビットの値のみを用いるが，計算回路の内部では複数ビットの整数表現を用いる．

図 -2 に示すように，Dither NN は画像処理におけるディザリング（入力値を少ない階調表現に変換する際の量子化誤差が，領域全体で平均的に小さくなるように切り上げ・切り捨てを選択する手法）に着想を得た方式である．中間色を表現可能なグレースケール画像を，単一の閾値により二値化すると白つぶれ・黒つぶれが発生する．ここで，二値化時の量子化誤差を隣接ピクセルに拡散し，その後に二値化することで，各ピクセルは通常の二値化と同様に

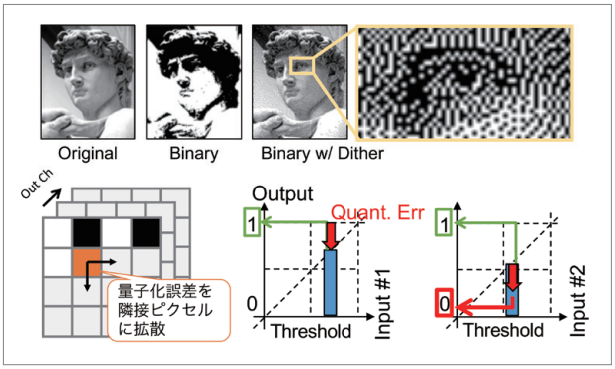


■図 -1　二値化ニューラルネットワークハードウェアにおける数値表現の変化

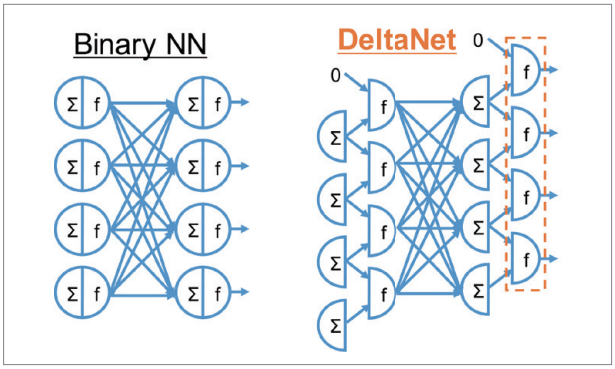
白黒のみで表現されるが，画像全体では白黒の濃淡により疑似的に中間色を表現することが可能になる．Dither NN はこの誤差拡散法に基づくディザリングを二値化ニューラルネットワークの活性化関数に適用したものである．二値化ニューラルネットワークでは，各ピクセルの活性化前の値を求めた後，各ピクセルに対して Sign 関数を適用するが，Dither NN では Sign 関数により発生する量子化誤差を隣接ピクセルに繰り越す．量子化誤差の量に応じて，隣接のピクセルの活性化後の値が変化し，各チャネルの空間方向で疑似的に中間値を表現することが可能になる．このとき，実際の二値化ニューラルネットワークハードウェアにおける量子化誤差の繰越は，pop-count の値を保存する中間レジスタの値から Sign 関数のための閾値を減算し，同一の回路で隣接ピクセルの計算を行うだけでよい．つまり，新たに外部に対するデータの読み書きはいっさい発生せず，回路構成の変更はきわめて小さい．このように，二値化ニューラルネットワークのハードウェアに対する利点を損なうことなく Dither NN は実現できる．FPGA を対象とした評価では，Dither NN の適用による回路規模の増加量は 0.3% であり，VGG-9 をベースとするモデルにおいて CIFAR-10 に対する精度は 1.3% 向上することを確認した．

関連する技術として，差分二値化によるハードウェア指向ニューラルネットワークである DeltaNet も紹介する．二値化ニューラルネットワークの精度低下の要因の 1 つは，活性化関数適用前の値の強弱が Sign 関数により失われることにある．たとえば，+1/-1 で活性値と重みが表現される二値化ニューラルネットワークにおいて，活性化関数適用前の値がどちらも正で異なる絶対値を持つ場合に，どちらも Sign 関数を適用すると +1 に変換される．ここで，片方がもう一方よりも大きな絶対値を持つということは，何らかの特徴がもう一方の特徴よりも強く検出されていることになるが，強弱に関する情報は Sign 関数により失われている．DeltaNet は，図 -3 に示すように，活性化関数適用前に隣接の活性化前の値との差分を求め，その差分を二値化する方式である．つまり，どちらの特徴がより大きいかを後続の層に伝播し，部分的に値の順序を保存することが可能になる．Dither NN と同様に FPGA を対象とした評価では，DeltaNet の適用による回路規模の増加は 3% であり，GoogLeNet をベースとするモデルにおいて CIFAR-10 に対する精度は 1.2% 向上した．

一般に，精度向上の恩恵のある方式はハードウェアの負荷が大きく，こちらを立てればあちらが立たないことになる．しかし，ハードウェアの内部に潜んでいる情報を，回路的には簡単な仕組みで活用することができれば，両方を立てることができる．残念なことに，上記の手法を用いたとしても浮動小数により表現されるモデルの精度には及ばないのが現状である．しかし，このような突破口を積み重ねることで，精度とハードウェア効率の両面で優れた究極的な技術を実現したいと筆者は考えている．



■図 -2　ディザリングに基づく二値化ニューラルネットワーク Dither NN

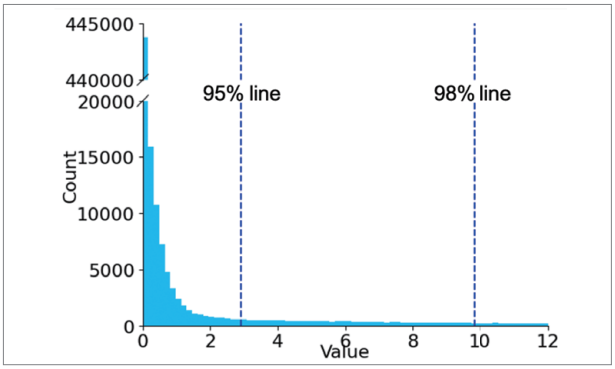


■図 -3　差分二値化関数を用いる二値化ニューラルネットワーク DeltaNet

ASBNN：アーキテクチャとアルゴリズムの協調によるベイジアンニューラルネットワークの高速化

ニューラルネットワークは一般にデータを用いて学習するものであるため，学習データとは性質の異なる入力に対しては特に誤った認識をしてしまうことがしばしばある．信頼される AI システムには，「分からないこと」が分かることが好ましい．そのための方法の 1 つとして，重みや活性値に確率分布を用いるベイジアンニューラルネットワーク（Bayesian Neural Network）と呼ばれる技術がある．出力の確率分布を観察することで，その出力の確信度を知ることができるという特徴があり，確信度に基づく後続処理が可能になるという利点がある．しかし，出力の確率分布を得るためには，同一のモデルに対して，モデルの重みの確率分布から値を複数回サンプリングして，ニューラルネットワークの順伝播計算を繰り返す必要がある．そのため，一般のニューラルネットワークよりも計算コストが単純に増加するという課題が存在する．ここでは，ベイジアンニューラルネットワークに現れる値の特性を活かして，出力の品質を保ったまま計算を高速化するアルゴリズムとハードウェアアーキテクチャの ASBNN[5] を紹介する．

ベイジアンニューラルネットワークでは，順伝播の計算ごとに重みの値を新たにサンプリングし，各層の計算を進める．このとき，順伝播ごとに各層の出力はどの程度変化するかを図 -4 のヒストグラムに示す．横軸が初回の順伝播との差分の絶対値，縦軸がその出現回数である．観察の結果，各層の出力の多くは初回の順伝播と似た値をとっており，ごく少数の値のみが大きく異なることが分かる．ASBNN はこの特性を活かして，ニューラルネットワークの計算中に出現するスパース性（ゼロの割合）を増やし，計算回数を削減する方式である．



■図 -4　ベイジアンニューラルネットワークのサンプリングごとの活性値の変化量

ASBNN では，初回にサンプリングを用いずに各層の出力を求める．各層の出力 $\mathrm{output}_{(\mathrm{k,\,first})}$ は，$\mathrm{input}_{(\mathrm{k,\,first})}$ を各層の初回の活性値，$\overline{\mathrm{W_k}}$ をそれぞれの重みが正規分布に従うとしたときの平均，$\overline{\mathrm{b_k}}$ をそれぞれのバイアスを正規分布に従うとしたときの平均とすると，以下の式で与えられる．

$$\mathbf{output_{(k,first)}} = Conv(\mathbf{input_{(k,first)}}, \overline{\mathbf{W_k}}, \overline{\mathbf{b_k}})$$

$\Sigma_{(W,\,k)}$ と $\Sigma_{(b,\,k)}$ がそれぞれの重みとバイアスの標準偏差とし，ε_w と ε_b を標準正規分布に従うとすれば，以下のとおり，k 回目にサンプリングされた重みとバイアスは，平均値のみに従い決定的に定まる項と，確率的な項に分解できる．

$$\mathbf{W_k} = \overline{\mathbf{W_k}} + \boldsymbol{\epsilon_w} * \boldsymbol{\Sigma_{(W,k)}},\ \mathbf{b_k} = \overline{\mathbf{b_k}} + \boldsymbol{\epsilon_b} * \boldsymbol{\Sigma_{(b,k)}}$$

以上に基づき，k 回目の出力 $\mathrm{output_k}$ は，以下の式のように近似することができる．

$$\begin{aligned}
&\mathbf{output_k} = Conv(\mathbf{input_k}, \mathbf{W_k}, \mathbf{b_k}) \\
&\simeq Conv(\mathbf{input_k}, \mathbf{W_k}, \overline{\mathbf{b_k}}) \qquad (1) \\
&= Conv(\mathbf{input_k}, \overline{\mathbf{W_k}}, \overline{\mathbf{b_k}}) + Conv(\mathbf{input_k}, \boldsymbol{\epsilon_w} * \boldsymbol{\Sigma_{(W,k)}}, \mathbf{0}) \\
&= Conv(\mathbf{input_{(k,first)}}, \overline{\mathbf{W_k}}, \overline{\mathbf{b_k}}) \\
&\quad + Conv(\mathbf{input_k} - \mathbf{input_{(k,first)}}, \overline{\mathbf{W_k}}, \mathbf{0}) \\
&\quad + Conv(\mathbf{input_k}, \boldsymbol{\epsilon_w} * \boldsymbol{\Sigma_{(W,k)}}, \mathbf{0}) \\
&= \mathbf{output_{(k,first)}} + Conv(\mathbf{input_k} - \mathbf{input_{(k,first)}}, \overline{\mathbf{W_k}}, \mathbf{0}) \\
&\quad + Conv(\mathbf{input_k}, \boldsymbol{\epsilon_w} * \boldsymbol{\Sigma_{(W,k)}}, \mathbf{0}). \qquad (2)
\end{aligned}$$

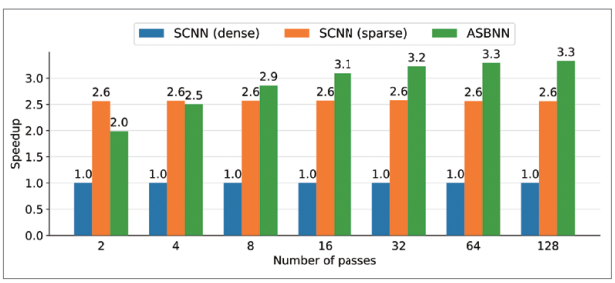
式（1）では，バイアスの標準偏差は重みのそれと比べて小さいため，サンプリングせずに平均のみを用いて計算することで得られる．畳み込みの出力は，式（2）のように，平均のみを用いて求めた初回の結果と，それとの差分に分解することができる．ここで，式（2）の第 2 項は初回の活性入力と各回の活性入力の差分に対して重みの平均を用いて畳み込みを行ったもの，第 3 項は各回の活性入力に対

して，重みの標準偏差と標準正規分布により畳み込みを行ったものとなる．ASBNN では，活性入力の差分が小さければ，第 2 項の計算を省略し，活性入力の値そのものが小さければ第 3 項の計算を省略することにより，高速化を達成する．つまり，各層の計算において，入力値（＝前層の出力）が初回の順伝播と大きく異なるところに限定して差分を計算するというアルゴリズムである．また，本アルゴリズムに対応する，差分個所のみを効率的に計算する，スパース構造に特化したハードウェアアーキテクチャを組み合わせることで，精度を低下することなく計算コストを大幅に削減することに成功している．図 -5 に ASBNN による速度向上率を示す．従来の密行列を対象とするアクセラレータおよびスパース行列に適した CNN アクセラレータ SCNN と比べて，サンプリング回数が 128 回のときに，それぞれ 3.3 倍，1.27 倍の速度向上を達成している．ここで，アルゴリズムとハードウェアをコデザインすることで，片方だけでは到達できないところに達することができる点が，研究を進める上で面白いところである．

NNgen：ディープラーニングのモデル特化ハードウェアを生成するオープンソースコンパイラ

ディープラーニングは，サーバなどの高性能なコンピュータシステムだけではなく，組み込みシステムと呼ばれる，低消費電力かつ小型なシステムにも用いられる．そのような場面での高速化方法として，Google の Edge TPU などの専用チップ（いわゆる AI チップ）を用いるほかに，利用者が設計・再構成可能なハードウェアである FPGA を用いる方法がある．FPGA を用いることで，画像の前処理回路などといったディープラーニング以外の回路と，周辺デバイスに接続される入出力インタフェース回路を 1 つのチップ上に集積することが可能になり，データの入力から最終出力までの遅延を削減することができるというメリットが存在する．FPGA 上にディープラーニングの処理回路を実現する方法としては，FPGA ベンダが提供する AI ツールを用いることで，一般的な深層学習フレームワークを用いた学習で得られたモデルを FPGA 上にデプロイすることができる．しかし，FPGA ベンダが提供するツールでは他社の FPGA には実装できず，改変は容易ではない．また，必ずしも深層学習モデルの特徴を踏まえたハードウェア構成ではなく，ターゲットとなる FPGA や混載される回路の規模と，要求速度や精度に応じた回路を実現することは一般に難しい．

NNgen[6] は，学習済みモデルから，そのモデルに特化したハードウェアアクセラレータのハードウェア記述（Verilog HDL）を生成する，オープンソースの高位合成コンパイラである．NNgen を用いることで，ハードウェア記述を一切することなく，FPGA 上にニューラルネットワークの処理システムを実現することができる．NNgen のコンパイラ構成は図 -6 のとおりである．NNgen は，ONNX（Open Neural Network eXchange）形式の学習済みモデルや，Define and Run 方式で記述された NNgen 形式のモデル記述から，Verilog HDL で記述されたニューラルネットワークのハードウェアアクセラレータを自動的に生成する，Python により実装された高位合成コンパイラである．図 -6 に，NNgen 形式のモデル記述の例を示す．入力データに対応する 'placeholder'，学習済み重みに対応する 'variable'，'conv2d' や 'max_pool' といったオペレータを組み合わせて計算グラフを構築する．計算グラフ中のオペレータに対して並列度などのアトリビュートを指定することで，同一の計算グラフから異なる構成

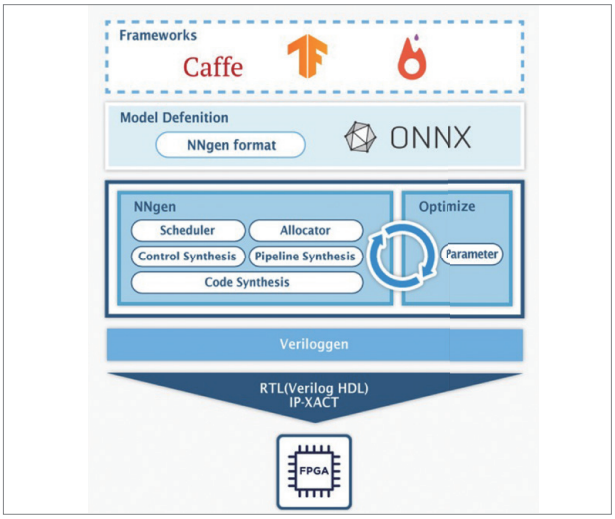


■図 -5　ASBNN による速度向上率

のハードウェアを生成することができる．そして，'to_ipxact' および 'to_verilog' といったメソッドにより，計算グラフが実際のハードウェア記述へとコンパイルされる．また，ONNX 形式のモデルを入力とすることもできる．

浮動小数の演算回路は多くの回路資源を要するため，重みと活性値を数ビットの整数で表現することが好ましい．NNgen は学習済みのモデルを整数量子化する機能を有しており，利用者が独自の量子化を適用することも可能である．また，NNgen の計算グラフは，ハードウェア化することなく，計算グラフと等価なソフトウェアの関数として実行することができる．これを活用することで，量子化前後のニューラルネットワーク振る舞いの変化をあらかじめ評価することができ，許容できる量子化結果が得られた後にハードウェア化すればよい．

コンピュータアーキテクチャの研究者であっても，ハードウェアの開発はしんどい．そのしんどい個所をソフトウェアによる自動化で解決することで，コンピュータアーキテクチャを意識した機械学習の研究がより加速されるものと筆者は考えている．現在，NNgen により生成されたニューラルネットワークアクセラレータは実際の製品でも利用されており，今後もさらなる高速化と回路規模の軽量化に向けて開発を進めていく．



■図 -6　ニューラルネットワークハードウェアの自動合成コンパイラ NNgen のアーキテクチャ

コンピュータアーキテクチャと機械学習の分野連携？

筆者の場合には，コンピュータアーキテクチャの研究者がもがきながらも，楽しく機械学習の研究を行っているわけであるが，逆に，機械学習の理論やアルゴリズムが専門の研究者がコンピュータアーキテクチャの研究を行うことも楽しいのではないかと思う．もちろん，より応用に近い研究者が機械学習の方式とコンピュータアーキテクチャを同時に考えてもよい．つまりは，優れた機械学習システムが実現できれば，誰が何をやってもよいわけである．異なる専門を持つ研究者同士が，相手の言っていることがよく分からないながらも交流することで，今は見つかっていない機械学習システムのあるべき姿が見えてくるかもしれない．そのような場所に積極的に飛び込んでもがくことが，コンピュータアーキテクチャと機械学習システムの進化に重要であると信じて，引き続きあれこれ取り組んでいきたい．また，読者の皆様にもお付き合いいただければ幸いである．

参考文献

1）しげの秀一：頭文字 D，11 巻，講談社（1998 年 4 月）．
2）OpenAI's GPT-3 Language Model : A Technical Overview, https://lambdalabs.com/blog/demystifying-gpt-3/
3）Strubell, E., Ganesh, A. and McCallum, A. : Energy and Policy Considerations for Deep Learning in NLP, 57th Annual Meeting of the Association for Computational Linguistics (ACL), Florence, Italy (July 2019).
4）Ando, K., Oba, Y., Hirose, K., Uematsu, R., Kudo, T., Ikebe, M., Asai, T., Takamaeda-Yamazaki, S. and Motomura, M. : Dither NN : An Accurate Neural Network with Dithering for Low Bit-Precision Hardware, The 2018 International Conference on Field-Programmable Technology (FPT'18) (Dec. 2018).
5）Fujiwara, Y. and Takamaeda-Yamazaki, S. : ASBNN : Acceleration of Bayesian Convolutional Neural Networks by Algorithm-hardware Co-design, The 32nd IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2021) (July 2021).
6）NNgen : A Fully-Customizable Hardware Synthesis Compiler for Deep Neural Network, https://github.com/NNgen/nngen

（2021 年 12 月 6 日受付）

■高前田伸也（正会員）　shinya@is.s.u-tokyo.ac.jp

2014 年東京工業大学博士課程修了．博士（工学）．2014 ～ 2016 年奈良先端大助教，2016 ～ 2019 年北海道大学准教授．2019 年から現職．