［機械学習工学］

2 機械学習応用システムの開発・運用環境

今井健男 | ぼのたけ／国立情報学研究所

太田満久 | （株）ブレインパッド

背景と歴史

現在機械学習がここまで実システムに利用されるようになったのは，各種機械学習アルゴリズムが成熟したことやそれらの精度が向上したこともあるが，何よりも，各アルゴリズムを簡易に利用して訓練・推論するためのフレームワークやライブラリ，そしてさまざまな環境にデプロイ[☆1]し，実運用するための環境が整備されたことによるところが大きい．本稿では，機械学習による訓練・推論を行うための，こうしたフレームワークやライブラリ等の現在の潮流について紹介したい．

しかしその詳細に入る前に，ここで機械学習応用システムの典型的な開発・運用フローに軽く触れておく．詳細は本特集に掲載されている丸山『1. 機械学習工学の狙いと展開』の「ライフサイクル各局面における課題」の節や本橋『6. 機械学習応用システムのプロジェクト管理と組織』を参照いただきたいが，大まかに言えば，要求定義や目的設計などの段階を経た後は以下のような流れになる．

1. PoC（Proof of Concept）フェーズ：試行錯誤によるモデルの設計と訓練を行い，実現したいアプリケーションが機械学習によって十分な精度をもって実現できるかを試作・評価する．
2. 開発フェーズ：PoC を経て得られた訓練済みモデルを元にアプリケーションプログラムを構築する．
3. 運用フェーズ：アプリケーションプログラムを用いてシステムを実稼働させる．

これら各フェーズにおいて利用環境に求められる要件には，主に以下のようなものがある（が，これだけに限らない）．

- PoC フェーズにおける，アルゴリズムの設計・選択のしやすさ．機械学習にもさまざまなアルゴリズムが存在し，あるいは深層学習では，ディープニューラルネットワーク（DNN）の設計に多くの選択の余地がある．PoC ではアルゴリズムを入れ替えたり，同じアルゴリズムでもパラメータを調整したりしながら訓練を何度も行う必要があるため，アルゴリズムの設計・選択は容易かつ効率的に行えることが理想である．
- 特に深層学習応用システムでの PoC フェーズにおいて，深層学習の訓練に要する多大な計算能力．とりわけ深層学習を用いたシステムにおいては，GPGPU（General Purpose Graphics Processing Unit）などの高性能プロセッサを（多くの場合複数）積んだサーバにおいて訓練をすることが現在一般的である．PoC フェーズにおいてはこの計算が何度も発生することになる．
- 運用フェーズにおいて，深層学習の推論に要する計算能力．推論時も訓練時ほどではないにせよ，訓練済みモデルに基づいた多量の行列計算（テンソル計算）が求められ，よって運用フェーズにおいてもこうした高性能計算環境が必要となる．

☆1 アプリケーションプログラムを運用環境に移行させ，有効化させること．「配備」「配置」「展開」などの訳があてられる場合もある．

- 運用フェーズにおける機械学習・深層学習応用システムの高速処理を実現するための，運用環境・ハードウェアに合わせた訓練済みモデルの最適化．汎用のソフトウェアであればコンパイラがこの機能を担っている．
- 特に SaaS（Software as a Service）における開発フェーズから運用フェーズへのスムーズなデプロイを実現する手段．SaaS の場合，訓練もサービス上での推論もクラウド上で行い，PoC 環境で訓練したモデルをクラウド上でシームレスに運用環境にデプロイしたい，と考えるのは自然である．

このように多くの要件を満たさなければならないので，一概に機械学習応用システムの開発環境・運用環境といっても，要求されるフレームワーク・ツールの機能は多岐にわたる．

以上を踏まえてこれまでの歴史を振り返ると，特に深層学習における PoC でのアルゴリズム設計のしやすさを主眼として，2015 年頃に Google や Facebook，Microsoft といった世界的 IT 企業が次々に自社開発のフレームワークをオープンソース（OSS）化し，これが深層学習の「民主化」を促した．

そして，訓練・推論それぞれの計算能力を向上させるため，深層学習専用アクセラレータチップの開発も 2015 年頃を端緒に多くの企業が手がけており，現在も開発競争が続いている．これに併せ，訓練済みモデルを最適化してさまざまな環境にデプロイするコンパイラ（モデルコンパイラ，バックエンド）の開発も 2016 年頃から盛んになってきている．また現在の機械学習応用システムの運用はクラウド上で，SaaS 的に行われるのが主流であり，そのためのツール等も近年普及してきている．

本稿では以降，まずは，主に PoC フェーズにおいて用いられるフレームワークを紹介し，次に特に深層学習用コンパイラの研究開発に大きく寄与することとなった中間表現について触れた後，深層学習用コンパイラを紹介する．主に運用フェーズで用いられるフレームワークやハードウェアについて紹介する．最後に今後の展望を述べる．なお，本稿で触れるものは，特に近年開発と利用が盛んである深層学習向けのものが中心となることに留意されたい．

フレームワーク

前述の通り，機械学習，特に深層学習の実用化・民主化に貢献したのは，世界的 IT 企業が中心となって自社製のフレームワークを 2015 年を端緒に相次いで OSS 化したからであった．フレームワークの導入によって，一般的な機械学習ではインタフェースを統一してさまざまなアルゴリズムを自由に選択し，深層学習ではネットワークの容易な設計が行えるようになった．さらにいえば，特に深層学習における GPGPU 等を用いた高性能計算をフレームワークの内部に隠蔽することで，DNN の訓練・推論がきわめて容易になった．これにより，特に PoC フェーズでのモデルの試作・訓練が誰でも気軽に行えるようになった．本章ではフレームワークや関連ライブラリの詳細を，PoC フェーズ内での手順を踏まえて紹介する．

PoC と EDA

現実のデータは非常に複雑な構造を持っていることが多い．そのため，与えられたデータを適切にモデル化するには，与えられたデータの構造や特徴を把握する必要がある．PoC フェーズは，あらかじめ仮定していたモデルの構築から始めるのではなく，まずはデータをさまざまな観点で集計・可視化することから始めるのが常道となっている．これは統計学者の J. W. Tukey によって提唱された考え方で，EDA（Exploratory Data Analysis，探索的データ解析）と呼ばれるアプローチである．

EDA に始まる PoC フェーズは，試行錯誤の繰り返しとなる．そのため，その都度コードの実行結果を可視化し，確認できる対話型の実行環境が好ま

れて使われている．Jupyter Notebook は対話型の実行環境の代表例である．Jupyter Notebook を使うと，実行可能なコードや Markdown 形式のテキスト，コードの実行結果等を含むドキュメントを Web ブラウザ上で作成し，共有することができる．また，Matplotlib などの可視化ライブラリを用いることで，実行結果を随時可視化しながら分析を進めることもできる．Google, Microsoft, Amazon といった大手クラウドベンダも Jupyter Notebook をもとにした機能を提供していることからも分かるように，Jupyter Notebook は機械学習コミュニティに広く受け入れられているが，複数のコードを任意の順番で実行でき，実行済みのコードを書き換えることも可能なため，実行結果の再現性の担保が難しいという課題がある点には注意が必要である．

データの前処理

現実の生データには欠損値や異常値が含まれることがよくある．そのため，EDA と同時に，不要なデータの削除や補完といった前処理を行うのが一般的である．高い精度を実現するには，アルゴリズムそのものよりも前処理が重要であるとも言われており，前処理は欠かせない工程となっている．前処理には，Pandas や NumPy といった Python ライブラリが使われることが多い．Pandas は統計解析向けの言語である R 言語のデータフレームという概念を Python に導入し，高機能なデータ処理機能を提供する．NumPy は Pandas も内部的に利用している行列演算のライブラリで，大部分を C 言語で記述することで，Python の欠点である実行速度の遅さを解消し，高速な演算を可能にしている．

機械学習の訓練

次に，前処理の終わったデータを使って，機械学習の訓練を行う．Python には多くの機械学習ライブラリが存在しており，さまざまな機械学習アルゴリズムを利用することができる．中でもデファクトスタンダードとなっているのが Scikit-learn である．Scikit-learn を使うことで，さまざまな機械学習アルゴリズムを共通のインタフェースを通じて利用できる．

ただし，機械学習の中でも深層学習に絞ってみると，少し状況が異なる．深層学習においては，ネットワークアーキテクチャを柔軟に設計できる必要があるが，Scikit-learn にはその機能はない．そのため，深層学習においては Scikit-learn 以外の専用のライブラリを利用するのが一般的である． 深層学習向けのライブラリは 2015 年から 2017 年にかけて多く公開されている（表 -1）．

これらの中で，2018 年時点でユーザ数が最も多いと考えられるのは Google が中心となって開発している TensorFlow である．TensorFlow は，事前にネットワークアーキテクチャを計算グラフとして定義しておき，あとでまとめて計算を実行する Define and Run と呼ばれる方式を採用している．深層学習の訓練は，一般に計算量が非常に大きいため，GPU など CPU 以外の高速な演算装置を利用することが多い．その場合演算装置間でデータの移動が発生するが，Define and Run 方式ではあらかじめ計算グラフが得られているため，データの移動を最小限に押さえるなどの高速化が期待できる．一方，Define and Run 方式は独特の記述方法に慣れる必要があったり，計算グラフの定義に間違いがあった

■表 -1　主な深層学習用フレームワーク

公式リリース時期	フレームワーク名	主たる開発元
2015 年 1 月	Torch7	Facebook
2015 年 3 月	Keras	Google
2015 年 4 月	CNTK	Microsoft
2015 年 5 月	neon	Nervana（現：Intel）
2015 年 6 月	Chainer	Preferred Networks
2015 年 11 月	TensorFlow	Google
2015 年 12 月	Apache MXNet	DMLC
2016 年 6 月	Neural Network Libraries	Sony
2017 年 1 月	PyTorch	Facebook
2017 年 4 月	Caffe2	Facebook

としても，エラーは計算の実行時に起こるため，デバッグが難しいなどの問題もあった．

一方で，Preferred Networks の開発した Chainer や Facebook の PyTorch などが採用しているのは，計算グラフの定義と計算処理を同時に行う，Define by Run と呼ばれる方式である．Define by Run 方式は，Define and Run 方式と比較して高速化が難しいと言われていたが，ネットワーク構造が動的に変化するようなアルゴリズムを構築できる点や，挙動が直感的でデバッグが容易な点など，Define and Run 形式にはないメリットも多いことが分かってきた．そのため，近年は Define by Run 方式を採用するライブラリが増えており，2018 年には前述の TensorFlow も Define by Run 方式でも記述できるように拡張されている．

アルゴリズムの選択

機械学習の精度はデータによって変わるため，どのアルゴリズムを選べばよいか，一概には決められない．より高い精度を求めるには，同一の訓練データを使ってさまざまなアルゴリズムをさまざまなパラメータで訓練し，最も性能の良いものを採用する必要がある．

機械学習の訓練にはある程度の時間がかかる．すべてのパターンを網羅的に試すに越したことはないが，実際には時間制約があるため，それは不可能である．そのため，性能の良いアルゴリズムやパラメータを効率良く見つけだす技術の研究が盛んである．従来は，深層学習のネットワーク構造を固定したまま，最適なパラメータを見つけ出す手法の研究が多かったが，2017 年頃からは，ネットワークアーキテクチャの決定まで自動化する NAS(Network Architecture Search) とよばれる技術が進歩しており，一部のタスクにおいては，人間の設計したネットワークアーキテクチャを超えて最高精度を実現している．

ビッグデータ向けの環境

機械学習の訓練では，単一の計算機では処理できない膨大な量のデータが必要となることがある．そのため，分散型ファイルシステムにデータを保存し複数台の計算機を用いて前処理や訓練を行う分散処理基盤が利用されている．EDA や前処理については，HDFS（Hadoop Distributed File System）などの分散型ファイルシステム上のファイルに対して，Hadoop MapReduce や Spark などのフレームワークを用いたり，Presto や Impala などのクエリエンジンを用いることが多い．訓練については，アルゴリズムや分散方法によって精度や計算時間が大きく異なるため，フレームワークごとに最適化された実装が提供されていることが多く，それを利用することになる．

中間表現

2017 年前半頃までの深層学習フレームワークは，DNN の設計・訓練・デプロイをすべて 1 つのフレームワークで完結させるものがほとんどだった．これは利用者が簡易に DNN 環境を構築できるというメリットがある反面，利用フレームワークが対応しない環境へのデプロイが困難であったり，あるフレームワークで実装して効果の確認された DNN を別のフレームワークに移植しても効果を容易に再現できなかったりするなど，さまざまな弊害もあった．

そうした中，2017 年 9 月に Facebook と Microsoft が中心となって ONNX（Open Neural Network Exchange,「オニキス」と発音）の公開をアナウンスした．これは DNN を表現するための標準フォーマットであり，フレームワーク間で訓練済みモデルのインポート／エクスポートを促すものである．

以降，ONNX をサポートするフレームワークは急速に増え，また本稿執筆時点（2018 年 9 月）では 21 社がパートナー企業として ONNX の開発とサポートを表明している．さらにいえば，ONNX

はGitHub上で仕様とサポートツールがすべてオープンソースとして公開されており，サポート企業以外の企業，あるいは大学や個人であっても，仕様の策定に参加したり，ONNX対応ツールを開発したりすることが自由にできる．こうした背景から，DNNの標準フォーマットとしては事実上のデファクトスタンダードとなっている．

一方，ONNXと同様の中間表現フォーマットとしてNNEF（Neural Network Exchange Format）がある．これはOpenCVなどの規格を定めたKhronosグループが提案するもので，2017年12月にバージョン1.0の仕様書と関連ソースコードがオープンソースで公開された．ONNXがフレームワーク間のモデルの相互運用に重きを置く一方，NNEFはそれだけでなく，訓練済みモデルを各推論デバイス・ハードウェア向けに変換する変換仕様の標準化を目的にしている点が大きく異なる．

コンパイラ，バックエンド

後述のさまざまな運用環境の登場により表出してきたのが，PoCフェーズでのDNN設計・訓練環境と，運用フェーズにおける推論の実現とを分離して扱いたいという問題意識である．ユーザが使い慣れたDNN設計・訓練環境を使いながら，さまざまな運用環境へ機械学習応用システムを自由にデプロイし，それぞれの環境で効率的な推論を実現したい，というニーズが見られるようになった．

つまり，従来はDNNの設計・訓練・推論環境へのデプロイを統一的に1つのフレームワークで行っていたが，こうしたフレームワーク1つでは対応しきれないあらゆる運用環境・デバイスに対応し，運用環境のそれぞれにDNNを最適化させて，運用フェーズでの効率的な推論を実現したい，と考えられるようになった．

こうした経緯から，設計・訓練は取り扱わず，運用環境へのデプロイのみを主眼として，訓練済みモデルの最適化と推論用の各種デバイス・ハードウェアへの対応を目的としたコンパイラ（グラフコンパイラ，バックエンドなどとも呼ばれる）の開発も多く提案されるようになった（表-2）．特に前述のONNXやNNEFの登場を契機として，さまざまなフレームワークで訓練したモデルをこれら中間表現を介して容易にインポートできるようになってから，盛んに開発が進んでいる．

これらコンパイラの中核となるのは，高性能計算（いわゆるHPC）などで従来培われてきた並列コンパイラの技術，特にループ最適化の技術である．推論時にDNN内で行われる計算の多くはテンソルの積和演算であり，愚直にプログラムに変換すると多重ループがDNNの層ごとに構成されることになる．この多重ループをいかにターゲットデバイスごとに最適化するかがコンパイラでの最適化の骨子となる．

そのため，現在提案されているコンパイラ内で多く採用されているのがHalide[1]である．Halideは本来画像処理向けコードの最適化用フレームワークで，処理アルゴリズムと，そのアルゴリズムを最適化するループ変形処理（「スケジューリング」）を別々に書くことができる．スケジューリングを変更するだけで，同じアルゴリズムに対し，元のコードを変更しないまま容易に並列化・ベクトル化・タイリングなどのループ最適化を施すことができる．

■表-2　主な深層学習用コンパイラ

公式リリース時期	コンパイラ名	開発元
2016年11月	Nervana Graph Compiler	Intel
2017年8月	TVM	Univ. of Washington
2017年10月	PlaidML	Vertex.ai
2017年10月	TACO	MIT
2017年11月	DLVM	Univ. of Illinois
2018年2月	Tensor Comprehensions	Facebook
2018年4月	TIRAMISU	MIT
2018年5月	PyTorch Glow	Facebook
2018年8月	ONNC	Skymizer / BITMAIN

さらには，特に GPU 向けへのスケジューリング自動化が現在研究開発のトレンドとなっている．端緒は Mullapudi らによる Halide のスケジューリング自動化の研究[2] だが，ほかに自動スケジューリングを採用した深層学習向けコンパイラとしては Facebook による Tensor Comprehensions（TC），また Chen らによる AutoTVM がある．TC は遺伝的アルゴリズムによって，AutoTVM は XGBoost もしくは GRU を使った機械学習によってスケジューリングの自動最適化を行う．

運用環境・ハードウェア

運用フェーズにおいては，いかに機械学習の推論を高速・効率的に実行しながら，機械学習応用システムのスムーズな運用を行うかが至上命題となる．現在，実社会における機械学習応用システムは大きく分けて SaaS・クラウドとエッジデバイスに分類でき，それぞれ効率的な推論に求められる要件や問題意識も変わってくる．本章では以降，SaaS とエッジデバイスでのそれぞれにおける課題や現在提供されている技術・ツールを述べる．ハードウェア（チップ）は両者にまたがるものであるが，便宜上エッジデバイスの節で触れる．

SaaS

機械学習応用システムとモデルの再訓練

機械学習応用システムの対象としているデータは，さまざまな要因により統計的な性質が変化していくことがある．原則として，訓練済みモデルは訓練時に用いたデータと同じ統計的な性質を持ったデータが入力されることを想定しているため，データに統計的な性質の変化があった場合には，再訓練を行い，訓練済みモデルを更新する必要がある．訓練済みモデルの更新のタイミングは，データの変化の度合いによって決める必要があり，一般に，ほかのモジュールとは更新のタイミングが異なる．

そのため，訓練済みモデルは，コードとは別に管理する必要がある点に注意が必要である．たとえば，Cloud ML Engine，Azure Machine Learning，Amazon SageMaker といったクラウドサービスでは，訓練ジョブが投げられるたびにバージョンが付与され，過去の訓練済みモデルに遡れるようになっている．

機械学習モデルの更新

機械学習の性能は訓練データに依存して決まるため，再訓練したからといって必ずしも精度が良くなるとは限らない．そのため，何らかの方法で再訓練前後の精度を比較し，更新すべきか否か判断する必要がある．

訓練済みモデルの評価方法には，大きく分けてオフライン評価とオンライン評価の 2 種類がある．オフライン評価では，与えられたテストデータに対する精度を比較するのに対し，オンライン評価では，実際にシステムに組み込み，精度指標やビジネス指標を評価する．

オフライン評価は，運用環境に影響を与えずに同じデータを使って精度評価ができるというメリットがあるが，必ずしも欲しい指標が得られるとは限らない．たとえば，商品の推薦システムにおいては，商品を推薦したために購買に至ったのかどうかを過去データから直接知る術はない．そのため，推薦の有無を考慮に入れない購買確率の予測精度など，別の指標を使った精度指標を用いた評価とせざるを得ない．

一方，オンライン評価は実際のシステムに組み込んでいるため，推薦結果を直接評価できる反面，システムへの影響を考慮に入れなければならない．このように，オフライン評価とオンライン評価ではそれぞれ特性が違うため，両者を組み合わせた運用を行うのが一般的である．

オンライン評価方法の一種である A/B テストでは，トラフィックをランダムに 2 つに分け，それぞれ再訓練前後の機械学習モデルの推論結果を適用す

る．そのまましばらく運用し，性能の良かったほうを採用する．Amazon SageMaker や ML Kit Firebase などのクラウドサービスでは，A/B テストのための機能が提供されている．

統合的な環境

PoC から運用まで，一気通貫で管理する仕組みも登場している．Kubeflow は 2017 年に公開された Kubernetes をもとにした機械学習プラットフォームで，Jupyter Notebook による PoC 環境，分散リソースを用いた訓練やハイパーパラメータチューニング，推論の API（Application Programming Interface）による公開機能などがある．Kubernetes をもとにしているため，訓練や推論を容易にスケールさせることができる．機械学習，特に深層学習の場合は，推論にも相応の計算コストがかかるため，オートスケールの機能が重視されるケースも多い．実際，各クラウド事業者の提供する機械学習サービスでも，オートスケールをうたっていることが多い．

ハードウェア・エッジデバイス

深層学習フレームワークは初期段階から現在に至るまで，GPGPU を利用するものが多い．GPU が比較的廉価で普及していたこともあり，また DNN の推論には GEMM（GEneral Matrix Multiplication）演算などを応用したテンソルの計算を多用するため，GPU による並列演算がきわめて有効であった．

しかし一方で，より効率的な計算能力を求めるため，深層学習の演算に特化した専用アクセラレータチップの開発も 2014 年頃から始まり，特に 2017 年以降加速した（表 -3）．最も有名なのは，かの D. Patterson が設計の指揮を執った Google の Tensor Processing Unit（TPU）であろう．2014 年から 2015 年に開発されたとされるバージョン v1 は推論のアクセラレーションのみが可能であったが，2017 年に発表された v2 では訓練のアクセラレーションも可能となり，2018 年に発表された 3.0 では 100 ペタ flops と，数値のみの比較でいえばスーパーコンピュータ「京」の 10 倍の処理性能を誇る．

TPU は主にサーバに搭載することを念頭に置いたチップであるが，家庭用の PC やモバイルデバイス，エッジデバイスでの使用を意識したものも登場している．組込みデバイスでは使用電力・発熱・容積等の問題から，一般的な GPU をそのまま使用できない環境も多く，厳しい使用制約や貧弱な利用可能リソースの中でも十分な性能を出せる専用の小規模チップへのニーズは今なお高い．

またこうした動きに伴い，モデル圧縮の研究も盛んに行われている．一般的な訓練済み深層学習モデルのサイズは数百メガバイトから，大きいもので数ギガバイト，場合によっては 10 ギガバイトを超えることもある．これほどの規模のモデルはエッジデバイスに収めて処理を行うのがそもそも困難な場合が多く，圧縮して省メモリ化と高速化の両方を狙うのがモデル圧縮である．

モデル圧縮には剪定（pruning）や蒸留（distillation）といった手法が存在するが，特に多く用いられているのは量子化（quantization）である．一般的なフレームワークでは計算をデフォルトで 32 ビット浮動小数点にて行うものが多いが，これを低ビットの整数や固定小数点演算に置き換える手法である．

多くの調査（文献 3）など参照）により，単純な量子化では 8 ビットまでビット数を減らしても精度

■表 -3　深層学習専用アクセラレータチップ

発表時期	名称	開発元
2014 年 9 月	Myriad 2	Movidius（現：Intel）
2015 年頃	TPU v1	Google
2016 年 8 月	HPU	Microsoft
2017 年 5 月	TPU v2	Google
2017 年 8 月	Brainwave	Microsoft
2017 年 8 月	Myriad X	Intel Movidius
2017 年 9 月	NVDLA	NVIDIA
2017 年 10 月	NNP	Intel Nervana
2018 年 7 月	Edge TPU	Google

劣化がたかだか 1% 程度で済むことが知られており，現在多くのフレームワーク・コンパイラ・専用チップでは 8 ビット整数あるいは 8 ビット固定小数点での演算をサポートするものが多い．

そして，さらに小規模の ASIC（Application Specific Integration Circuit, 特定用途向け集積回路）や廉価な FPGA（Field Programmable Gate Array）でのアクセラレーションを可能とするため，2 値や 3 値といった非常に小さいビット幅での量子化を行う研究も盛んに行われている．2 値や 3 値では，メモリフットプリントを劇的に小さくできるだけでなく，積和演算を XNOR（eXclusive Not OR）と popcount（1 の値を持つビットの数を数える処理）で置換できるので，計算そのものも劇的に高速化できる．いかに精度劣化を抑えつつ超低ビットでの量子化を実現するかがこうした研究の焦点となっている．

今後の展望

最後に，今後の開発・運用環境の研究開発がどのような方向で進んでいくか，筆者らの個人的な見解を述べておきたい．

PoC で用いられる機械学習フレームワーク・深層学習フレームワークの開発競争はほぼ終焉を迎えたといってよい．よほど新規の機能が見出されない限り，現在あるフレームワークが拡充される方向で開発が進むのみであろう．

一方，すでに世間の潮流は計算の高性能化を見込んだハードウェアやコンパイラの開発にシフトしており，競争が激化しつつある．特にハードウェアは，従来の GPGPU から移行すべくさまざまなものが提案されている段階であり，まだデファクトスタンダードとなるチップは登場していない．また SaaS ベースの運用環境においてもさまざまなサービスやフレームワーク・ミドルウェアが登場しつつある段階であり，今後新たなツール群が提供される可能性をまだ残している．

チップやコンパイラ，また運用環境におけるミドルウェアの開発が進んでいるところを見るに，すでに汎用には枯れてしまったチップやシステムソフトウェアの研究開発が，機械学習応用システム・深層学習応用システムの世界で今再び注目を浴び，技術の再構築が行われている，といっても過言ではない．現在はこういった潮流の中心は巨大 IT 企業や企業主体のオープンソースコミュニティであるが，ぜひ（特に国内の）アカデミアからも参入して，新たな風を吹かせてほしい，と筆者らは期待している．

参考文献

1) Ragan-Kelley, J., Adams, A., Paris, S., Levoy, M., Amarasinghe, S. and Durand, F. : Decoupling Algorithms from Schedules for Easy Optimization of Image Processing Pipelines, ACM Transactions on Graphics, 31(4), 1-12 (2012), https://doi.org/10.1145/2185520.2185528
2) Mullapudi, R. T., Adams, A., Sharlet, D., Ragan-Kelley, J. and Fatahalian, K : Automatically Scheduling Halide Image Processing Pipelines, ACM Transactions on Graphics, 35 (4), 1-11 (2016), https://doi.org/10.1145/2897824.2925952
3) Cheng, Y., Wang, D., Zhou, P. and Zhang, T. : A Survey of Model Compression and Acceleration for Deep Neural Networks (2017), arXiv. http://arxiv.org/abs/1710.09282

（2018 年 9 月 3 日受付）

■**今井健男（正会員）** bonotake.oshigoto@gmail.com

2000 年東京大学大学院理学系研究科情報科学専攻修士課程修了．（株）東芝，LeapMind（株）勤務を経て，2018 年よりフリーランスエンジニア 兼 国立情報学研究所 特任研究員．現在は主に深層学習用コンパイラの研究開発を行っている．日本ソフトウェア科学会機械学習工学研究会 運営委員．

■**太田満久** mitsuhisa.ohta@brainpad.co.jp

（株）ブレインパッド アナリティクスサービス本部 副本部長．2010 年京都大学大学院理学研究科物理学・宇宙物理学専攻博士後期課程修了．博士（理学）．同年，（株）ブレインパッドに開発エンジニアとして入社し，後に機械学習エンジニアとしてアルゴリズム開発や先行研究に従事．日本ディープラーニング協会試験委員．日本ソフトウェア科学会機械学習工学研究会 運営委員．