深層学習コンパイラの概要と最近の動向

今井 健男
Idein株式会社 / 国立情報学研究所

xSIG 2019

自己紹介

今井 健男
Idein株式会社 エンジニア
兼 国立情報学研究所 特任研究員

- 経歴
  - ソフトウェア工学、特に形式手法、プログラム解析の研究
    - 株式会社東芝（2000〜2017）
    - マサチューセッツ工科大学 客員研究員（2007）
  - 深層学習コンパイラの研究開発
    - LeapMind株式会社（2017〜2018）
    - 国立情報学研究所（2018〜）
    - Idein株式会社（2019〜）
- 書籍 (共訳)
  - D. Jackson, 『抽象によるソフトウェア設計』
  - B. Pierce, 『型システム入門』
- その他活動
  - 日本ソフトウェア科学会 機械学習工学研究会（MLSE）
    発足メンバー / 現 運営委員

本日のあらまし

- 深層学習コンパイラの成り立ち
- 深層学習のおさらい
- 深層学習コンパイラの基礎
- 現在のコンパイラ技術開発動向
- コンパイラの等価性に関して

深層学習コンパイラの成り立ち

深層学習の興り

- 2006年 Hintonら[1]が深い階層のニューラルネットワークに対する高速な学習アルゴリズムを提案
- 2012年 KrizhevskyらのAlexNet[2]がILSVRC 2012にて従来の結果を大きく改善する結果で優勝
- 2015年 人間の画像認識誤り率とされる5.1%を2つの研究[3][4]が凌駕
- 2016年 AlphaGoが囲碁の世界的トップ棋士を破る

[1] Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, *18*(7), 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

[2] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Proceedings of NIPS 2012.*

[3] He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. http://arxiv.org/abs/1502.01852

[4] Ioffe, S., & Szegedy, C. (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. http://arxiv.org/abs/1502.03167

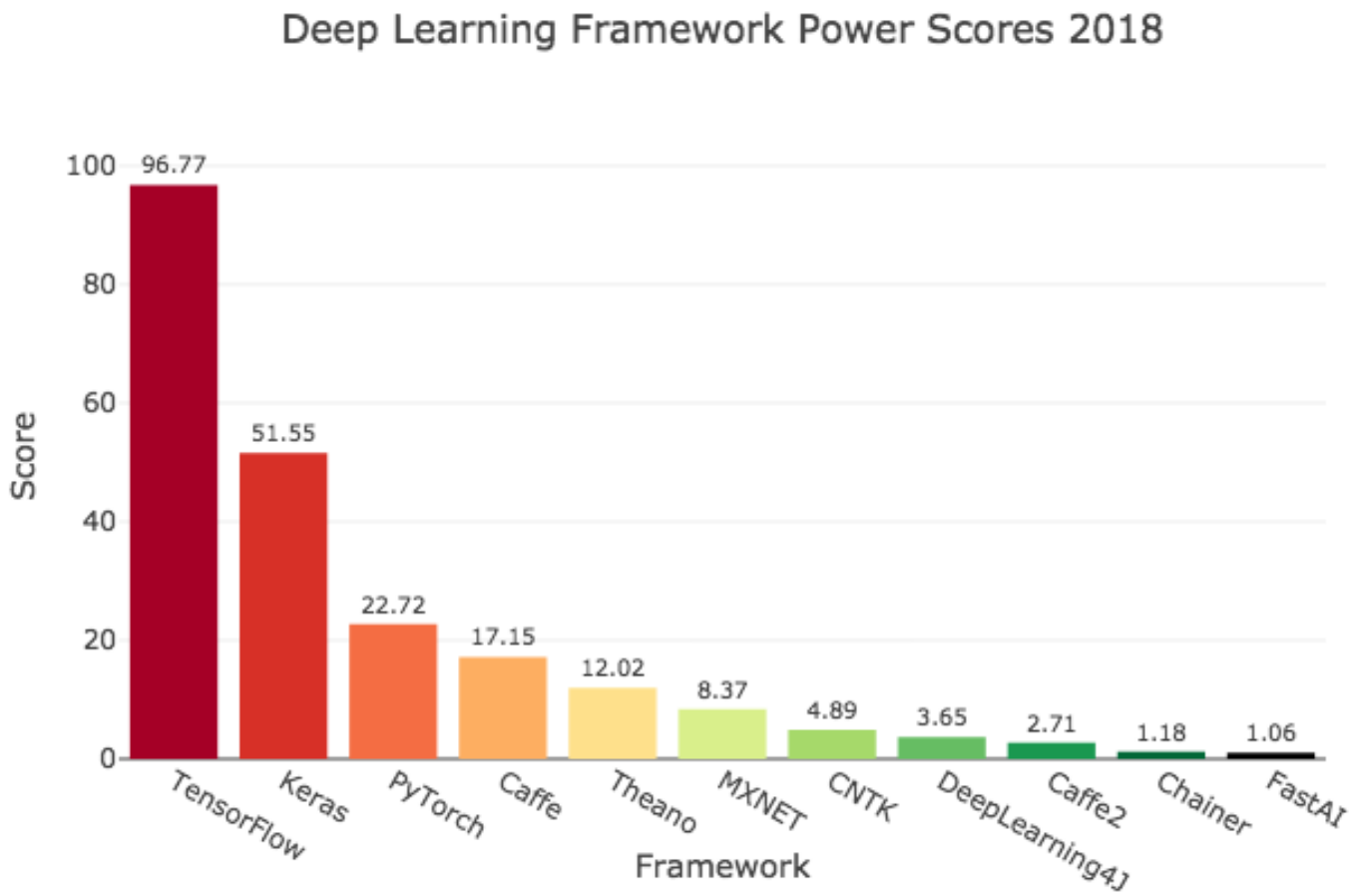
深層学習フレームワークの登場

- ニューラルネットワークの設計、学習・推論の実行を統一的な記述で行えるようにしたライブラリ
- 通常、**オペレータ**をつなげて**グラフ**を作ることでニューラルネットワークを表現
- 多くは汎用言語の埋込み型DSLとして実現
  - 現在の主流フレームワークは全てPythonがベース言語
- 多くはバックエンドとしてGPUの存在を仮定
  - NVIDIA GPU + CUDA を使用するものがほとんど

フレームワークによる
深層学習の民主化

2008.01　Theano (Univ. of Montreal)
2013.09　Caffe (Univ. of Montreal)
2015.03　Keras (Google)
2015.04　CNTK (Microsoft)
2015.04　mxnet（DMLC)
2015.06　Chainer (Preferred Networks)
2015.11　TensorFlow (Google)
2017.01　PyTorch (Facebook)
2017.04　Caffe2 (Facebook)

参考：フレームワークの人気度



引用）https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a

フレームワークの長所と短所

長所

- GPUを用いた複雑で煩雑な実装を隠蔽、抽象化
  → ネットワーク設計から訓練・推論までをワンストップで実行可能

短所

- 様々な実行環境にデプロイしづらい
  - 専用AIチップ、組み込みディープラーニングと相性があまり良くない
- 訓練も推論も特定のフレームワークに特化してしまう
  - PyTorchでは良好な精度が出たのにTensorFlowで再現しない…など

→ 訓練環境と推論環境を分離するニーズ

深層学習コンパイラ の登場

深層学習コンパイラ とは：

- （様々な）フレームワークで作られた
  訓練済み Deep Neural Network (DNN) を入力にとり
- （通常複数の）最適化をDNNに施して計算を
  高速化・効率化した上で
- 所望の（様々な）ハードウェア／プラットフォームでの
  動作に必要な目的コードを出力するもの

入力がグラフを成すことから「グラフコンパイラ」とも

※ 以降、単にコンパイラと書く場合がありますがいずれも同じものです

主なコンパイラOSS (統廃合されたもの含む)

公式リリース時期	名称	開発元
2016.11	nGraph	Intel Nervana
2016.11	TensorFlow XLA	Google
2017.8	TVM	Univ. of Washington
2017.10	PlaidML	Vertex.ai
2017.11	DLVM	Illinois Univ.
2018.2	Tensor Comprehension	Facebook
2018.4	TIRAMISU	MIT
2018.5	Glow	Facebook
2018.8	ONNC	Skymizer

更に：中間表現標準化の動き

ONNX（"オニキス"）

- 2017年9月、MicrosoftとFacebookが提唱
- 現在 28社 が公式に参画

NNEF

- 2017年12月、Khronosグループが仕様を公開
- 32社 がWGに参画

→ 訓練環境と推論環境の分離がさらに促進
コンパイラがより開発しやすい状況に

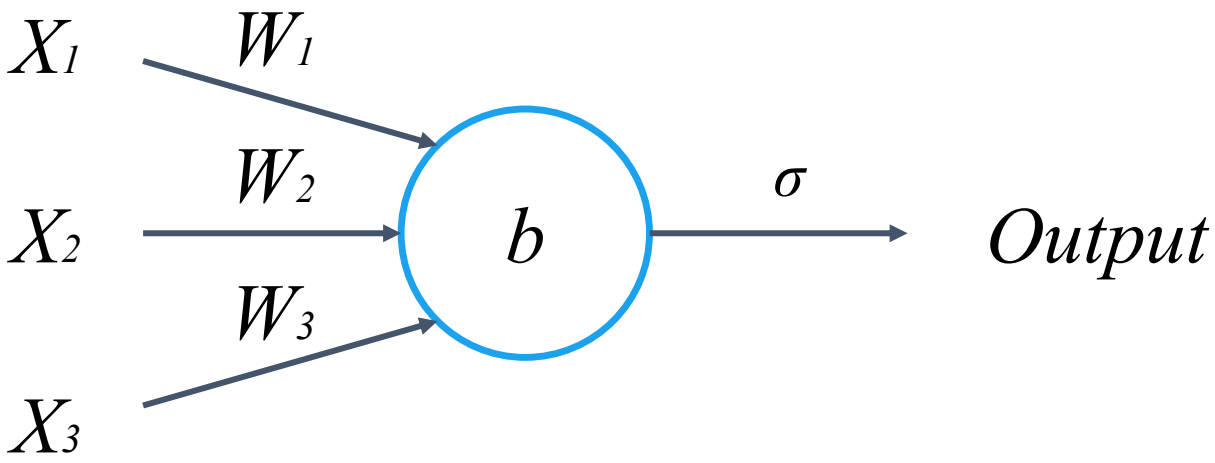
深層学習のおさらい

深層学習の基礎（１）
ニューロン / オペレータ

- ニューラルネットの構成単位
- 基本となるのはテンソルの積和演算

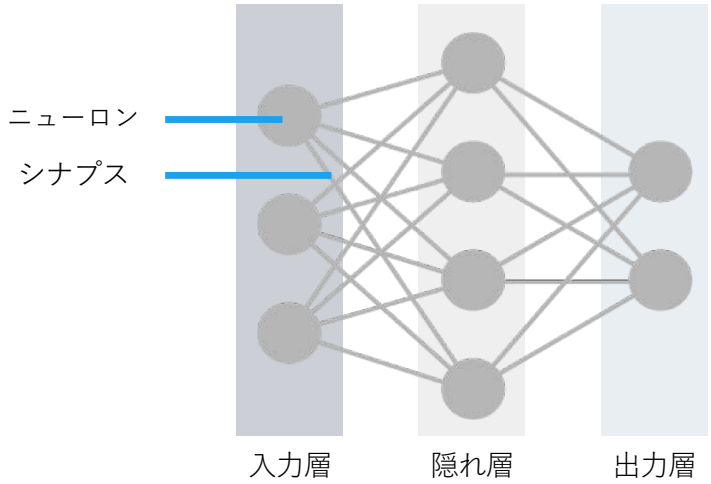
図は全結合層と呼ばれる層にあるオペレータ：

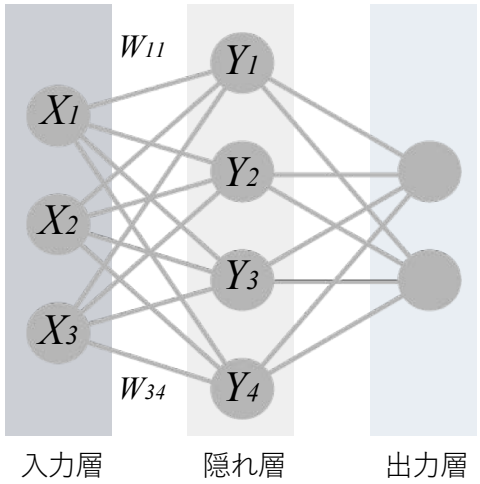
$$Output = \sigma(v), \quad v = \sum_{k=1}^{n} W_k X_k + b$$



深層学習の基礎（２）
深層ニューラルネットワーク、DNN

- ニューロン（オペレータ）を多層に重ねたものを**ニューラルネットワーク**と呼ぶ
- 特に中間層（隠れ層と呼ばれる）が２層以上あるニューラルネットワークを**深層ニューラルネットワーク（Deep Neural Network, DNN）**と呼ぶ
- DNNを使った機械学習を**深層学習（deep learning）**と呼ぶ





DNNのカテゴリ分け

FFNN

Feed-Forward Neural Network

循環構造のない、n層目の出力がn+k層目（$k \in \{1, \cdots\}$）の入力になるようなDNN

RNN

Recurrent Neural Network

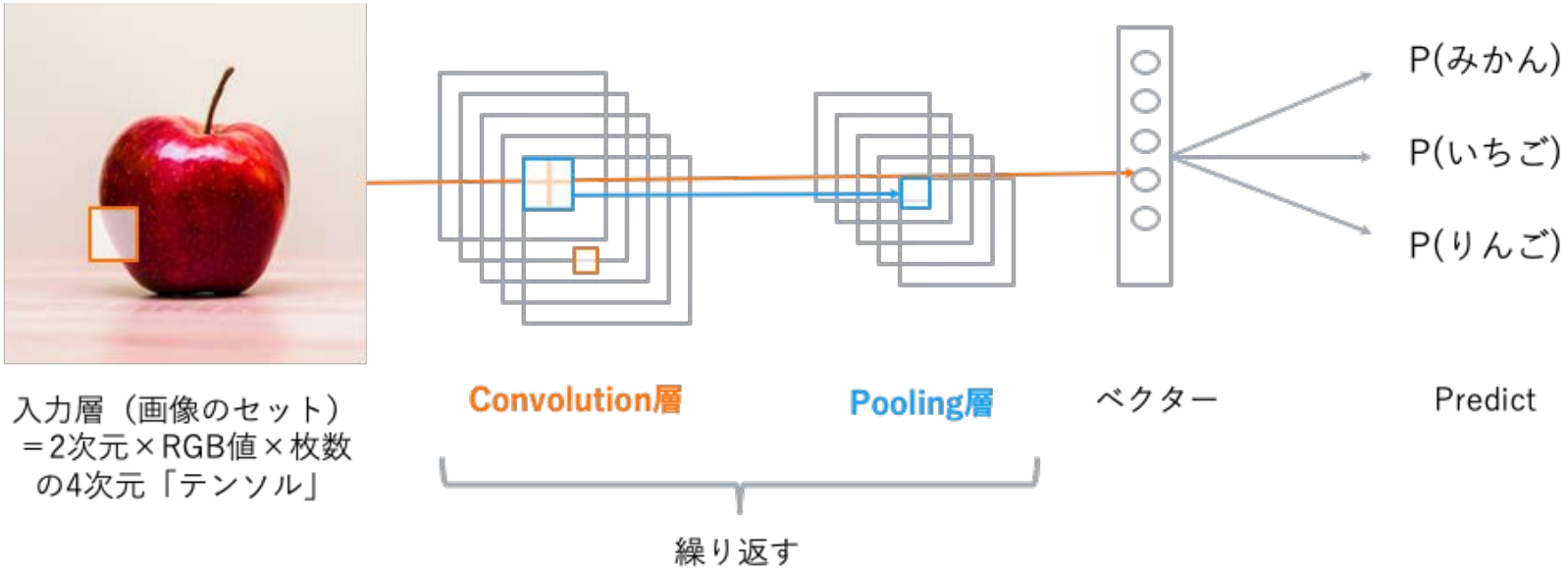
循環構造のあるDNN

- FFNN（のうち、特に後述するCNN）は画像認識に用いる
- RNNは自然言語処理や音声認識、時系列解析に用いる

Convolutional Neural Network (CNN)

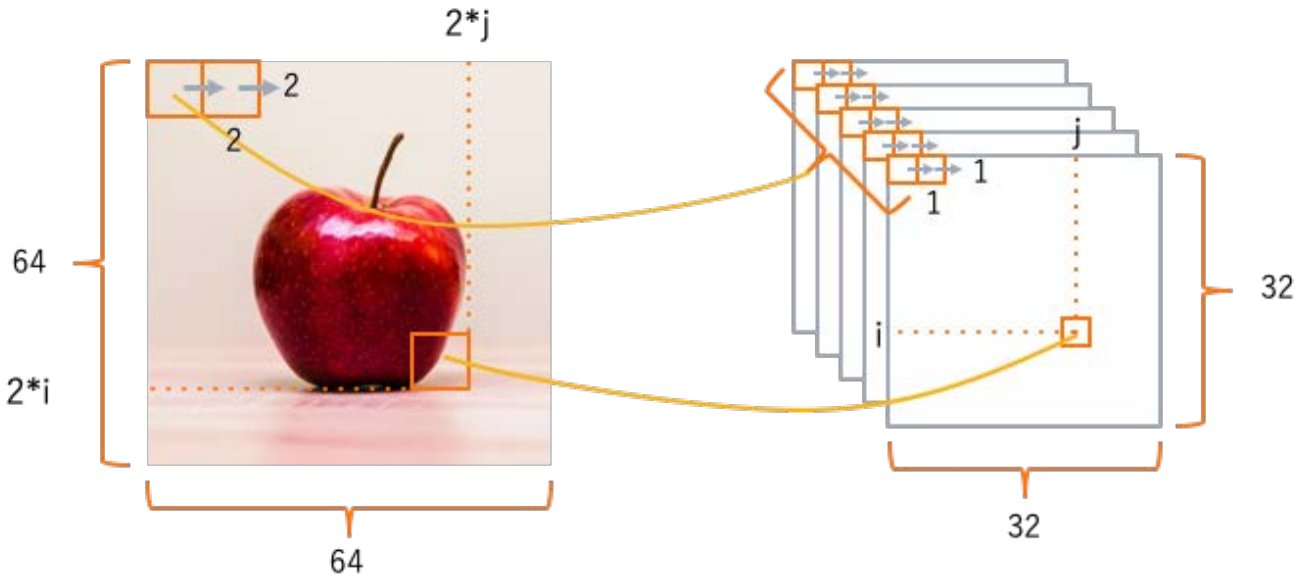
Convolutional Neural Network(CNN)は、主に画像認識で用いられるDNNの一種。

最初のinputが画像の場合、画像のRGB値と重みで内積計算をして、特徴量を抽出する**Convolution（畳み込み）**と特徴量をまとめる**Pooling**を繰り返し、画像の特徴量を取得して予測モデルを作る



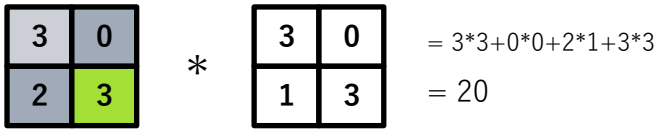
Convolutionの概要

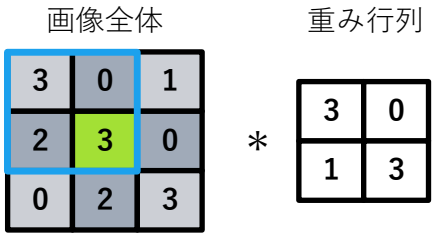
64×64の画像上で2×2のパッチを2pixelずつ滑らせ、
各パッチから5つの特徴量を取り出すConvolutionの例



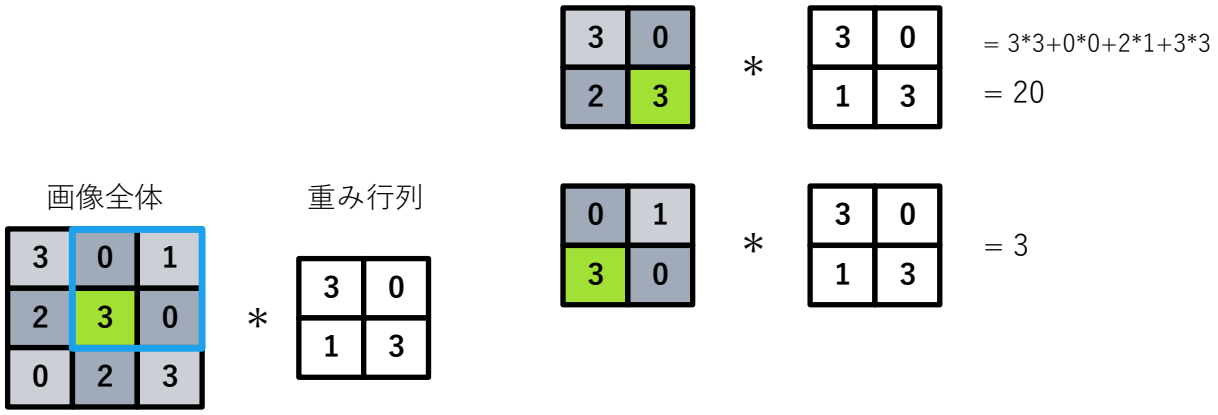
隣り合うパッチから抽出した特徴量は、
出力でも隣り合っているようにデータを保持する

Convolutionの概要

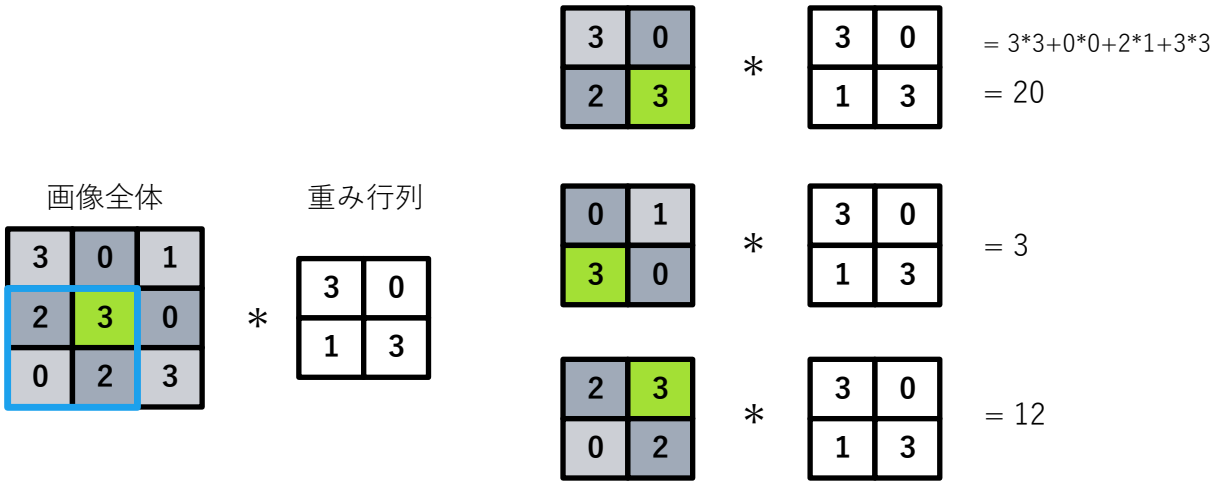




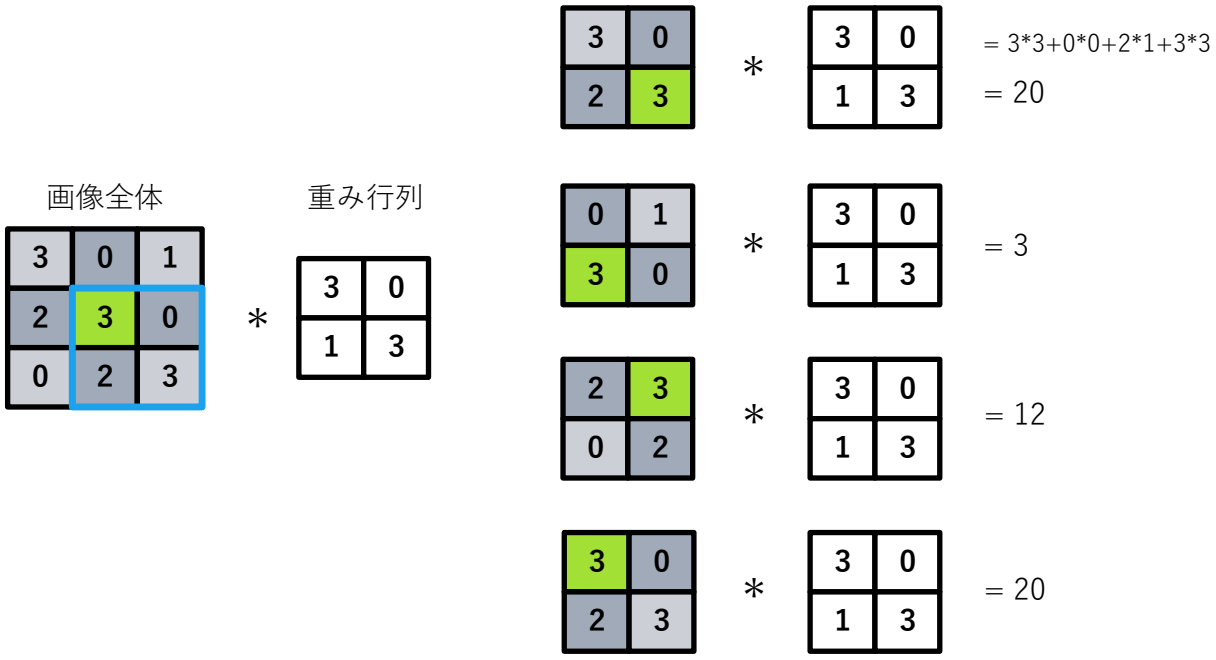
Convolutionの概要



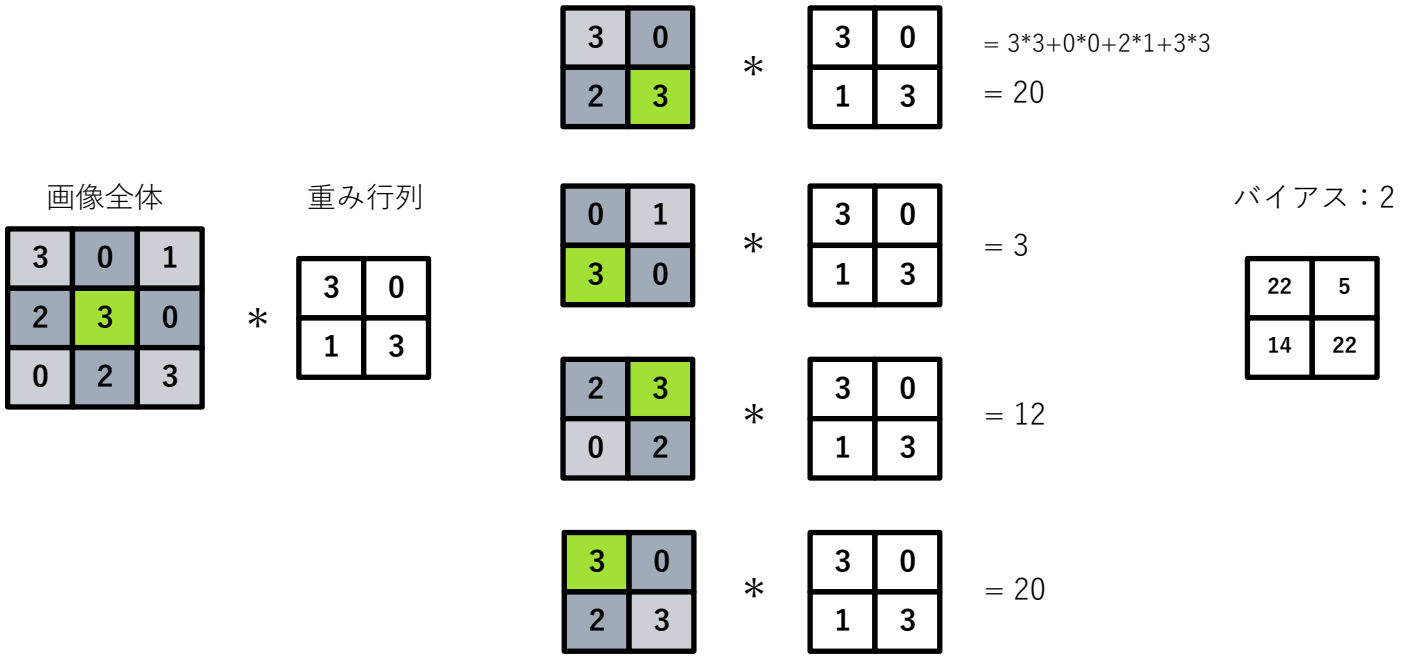
Convolutionの概要



Convolutionの概要

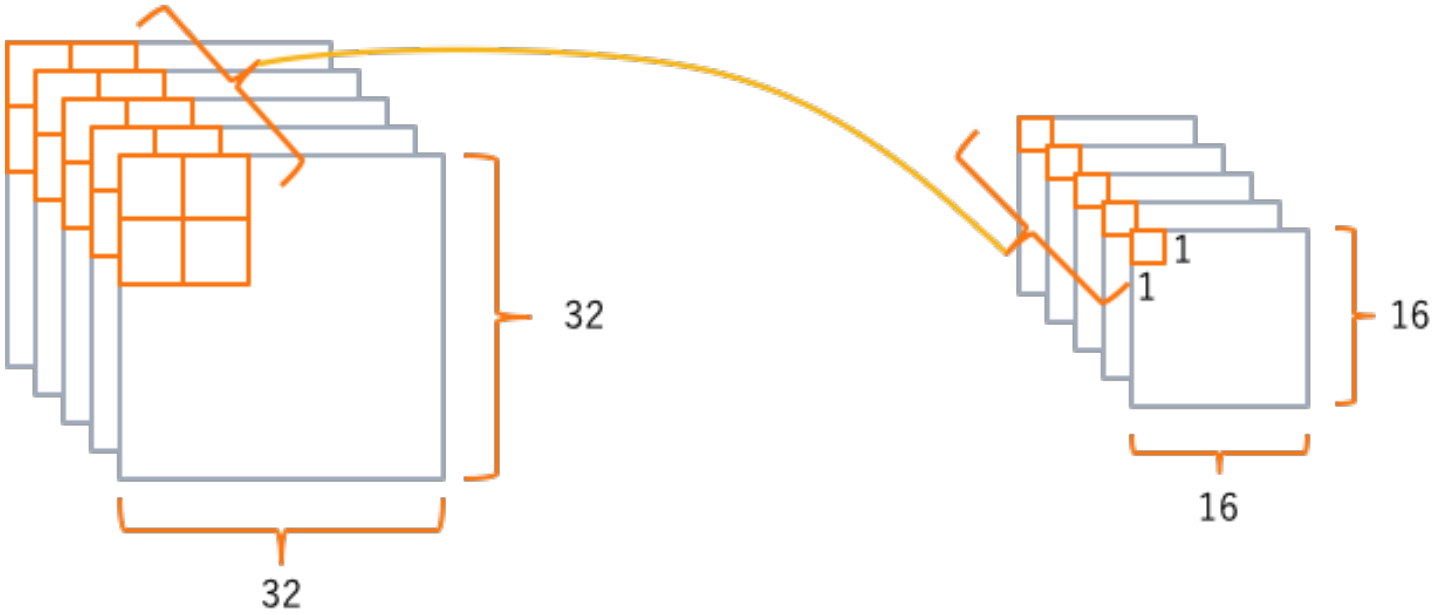


Convolutionの概要



Poolingの概要

認識精度・処理効率向上のためのダウンサンプリング



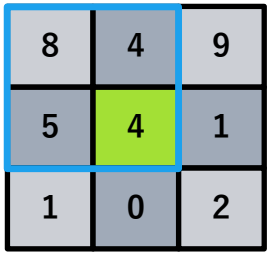
2×2の特徴量をまとめるPoolingの例

Poolingの概要

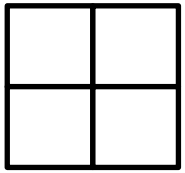
2×2 max pooling（最大値を用いたpooling）の例

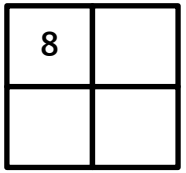
Max() = 8

Convの結果



サイズ：2×2

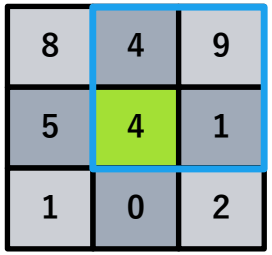




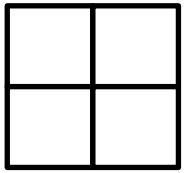
Poolingの概要

2×2 max pooling（最大値を用いたpooling）の例

Convの結果



サイズ：2×2

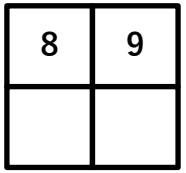


Max(

) = 8

Max(

) = 9



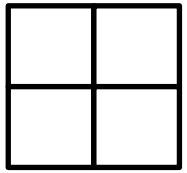
Poolingの概要

2×2 max pooling（最大値を用いたpooling）の例

Convの結果

8	4	9
5	4	1
1	0	2

サイズ：2×2

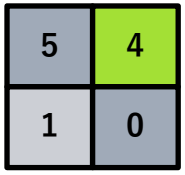


Max() = 8

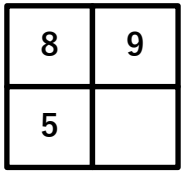
8	4
5	4

Max() = 9

4	9
4	1

Max() = 5

5	4
1	0



8	9
5	

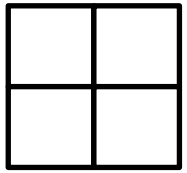
Poolingの概要

2×2 max pooling（最大値を用いたpooling）の例

Convの結果

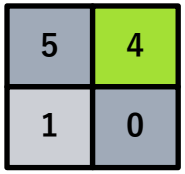
8	4	9
5	4	1
1	0	2

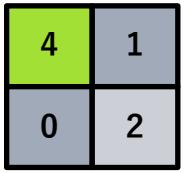
サイズ：2×2



Max() = 8

Max() = 9

Max() = 5

Max() = 4

8	9
5	4

深層学習コンパイラの基礎

深層学習コンパイラの特徴

通常のコンパイラに比べ、より大胆な最適化が用いられる

- 数値表現・ビット幅の変更
  - float32 から float16, int16, int8 への変更など
- 浮動小数点演算に対して誤差の生じる代数的規則の適用
  - 結合則に従った計算順序の変更など
- メモリレイアウトの変更
- etc.

- 理由
  - 深層学習推論が本来「ある一定の精度」でしか結果が保証されない
    - 80%の精度で計算されていたものが79%になったことを気にする人はあまりいない
  - 制御フロー・データフローが極めて単純で、変換しやすい

実際どれくらい変わるのか？（１）

- モデル：あるsegmentationのモデル、fp32, 320x200
- ハード：Raspberry Pi 3, CPU (Cortex-A53) の1コア利用
- NEONによるSIMD化利用

Operator fusion	アルゴリズム	Loop unroll	メモリレイアウト	実行時間（ms）
×	direct	×	NCHW	733
○	direct	×	NCHW	495
○	im2col	×	NCHW	286
○	im2col	○	NCHW	175
○	im2col	○	NHWC	166

実際どれくらい変わるのか？（２）

- モデル：MobileNet V2 1.0 224x224, 1000クラス分類
- ハード：STM32H7 (Cortex-M7)

float32のまま：**9.0秒**

float32のままで最適化：**3.1秒**

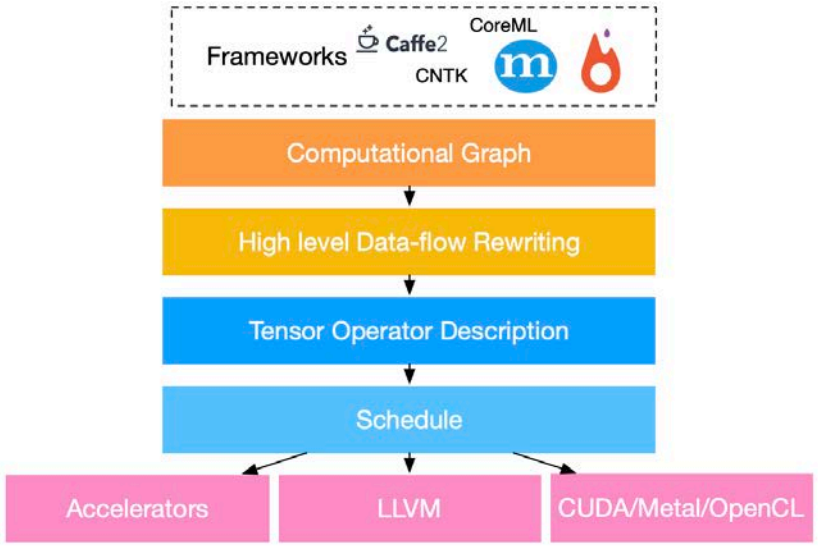
- Tightly Coupled Memory (TCM) の活用
- Cortex-M7 のSIMD命令の活用
- アラインメント等を意識したメモリ配置の最適化
- etc.

int16に変換して**1.1秒**

Deep learning コンパイラの構成

基本は2層構成
（中間表現＋最適化）

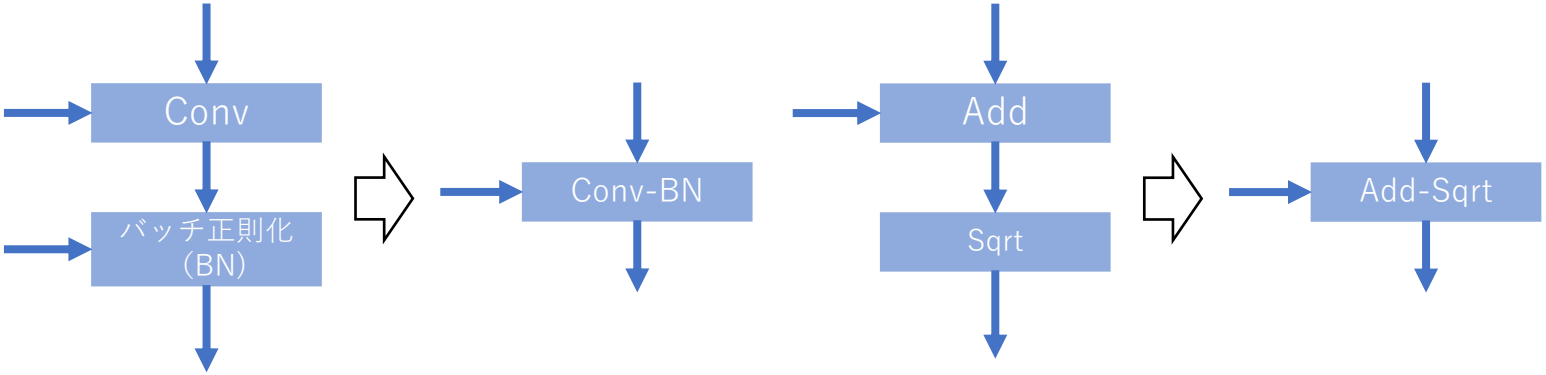
- **グラフレベル**
  - ネットワーク設計を表現
- **オペレータレベル**
  - テンソル計算を表現



TVMの例
（arXiv:1810.00952v1 より引用）

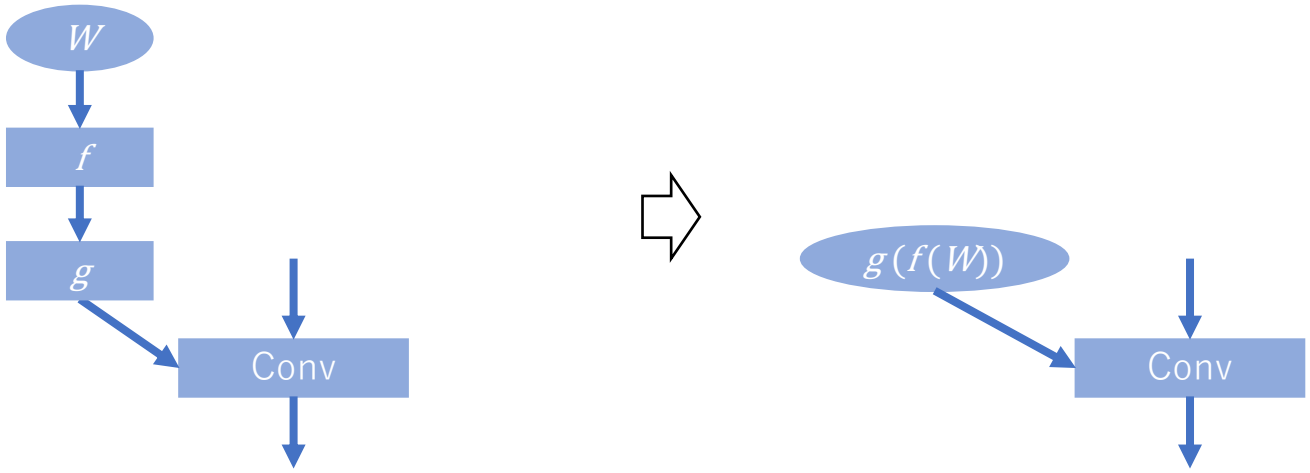
グラフレベル最適化（１）
Operator Fusion

- 複数のオペレータを1つにまとめる
  - 1つのオペレータ演算は多重ループで構成されており、これは
    いわゆるループ融合にあたる
- 特定のパターンでのオペレータの組み合わせはまとめても
  意味が保持される



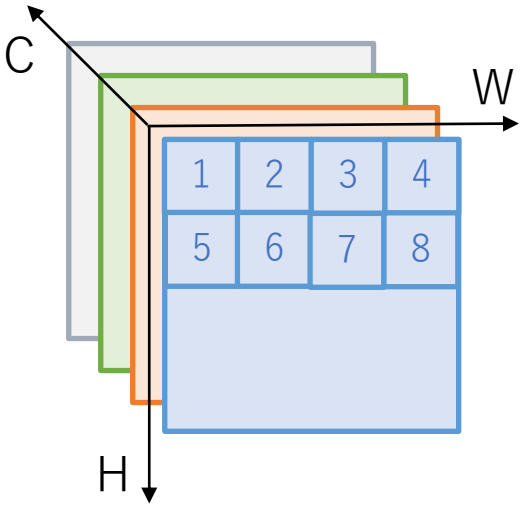
グラフレベル最適化（２）
Constant Folding

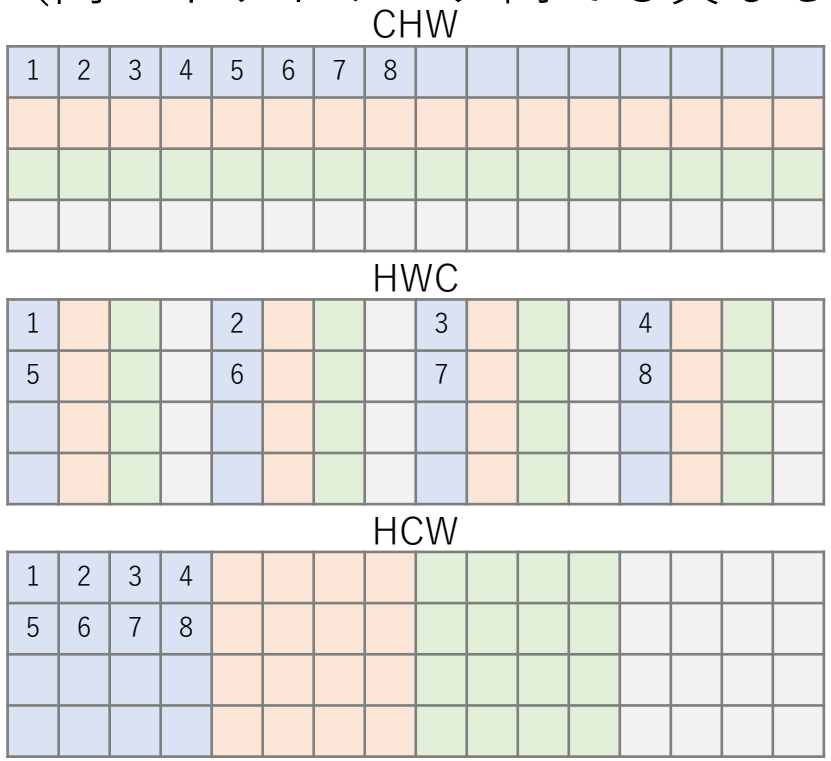
- いわゆる定数畳み込み／定数伝播
- 学習したパラメータは推論時には定数として扱えるため、それに連なる、値の確定するオペレータは予め計算してグラフを縮約できる



グラフレベル最適化（３）
Data Layout Transformation

- メモリレイアウトを変更する
- 2D CNNの場合、理論上は 4! だけレイアウトが存在
  - TensorFlowはNHWC, ONNX, PyTorch, ChainerはNCHW を採用
- どのハードウェアでどの演算を行うかによって
  最適なレイアウトは変わる（同一ネットワーク内でも異なる）





オペレータレベル最適化（１）
アルゴリズム最適化

Convolutionなど、よく利用されるオペレータは効率的な演算方法が多く提案されている

2D Convolution のアルゴリズム（入力 HxW、カーネル 3x3の場合）

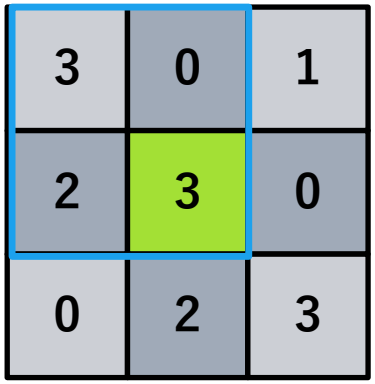
手法	概要	演算回数	メモリ使用量
direct	定義通り	–	–
im2col	行列乗算に変換	同じ	入力が9倍
Winograd	乗算回数削減	4/9程度	重みが16/9倍
FFT	乗算回数削減	最大2/9程度	重みがHxW/9倍

im2colによるconvolutionの高速化

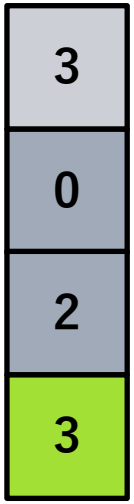
入力のうち畳み込み1回分を列に持つ行列を生成、
MVM（Matrix-Vector Multiplication）に落とし込む

カーネル 2x2 の場合

元の入力



生成行列

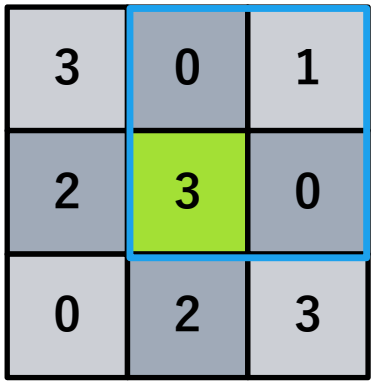


im2colによるconvolutionの高速化

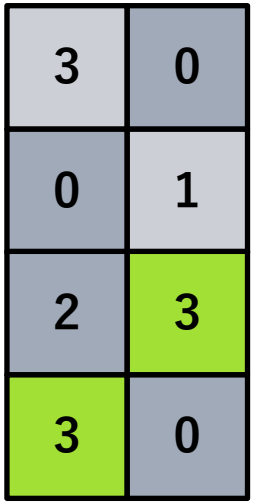
入力のうち畳み込み1回分を列に持つ行列を生成、
MVM（Matrix-Vector Multiplication）に落とし込む

カーネル 2x2 の場合

元の入力



生成行列

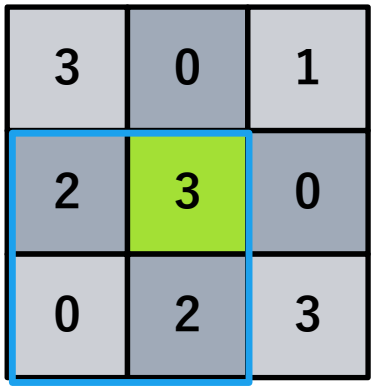


im2colによるconvolutionの高速化

入力のうち畳み込み1回分を列に持つ行列を生成、
MVM（Matrix-Vector Multiplication）に落とし込む

カーネル 2x2 の場合

元の入力



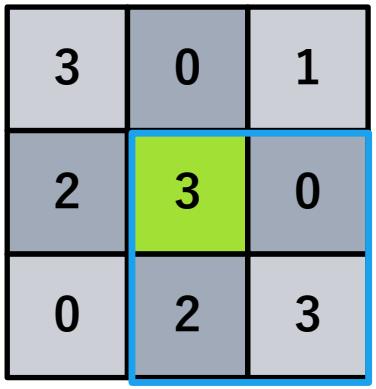
生成行列

im2colによるconvolutionの高速化

入力のうち畳み込み1回分を列に持つ行列を生成、
MVM（Matrix-Vector Multiplication）に落とし込む

カーネル 2x2 の場合

元の入力



生成行列

3	0	2	3
0	1	3	0
2	3	0	2
3	0	2	3

オペレータレベル最適化（２）
多重ループ最適化

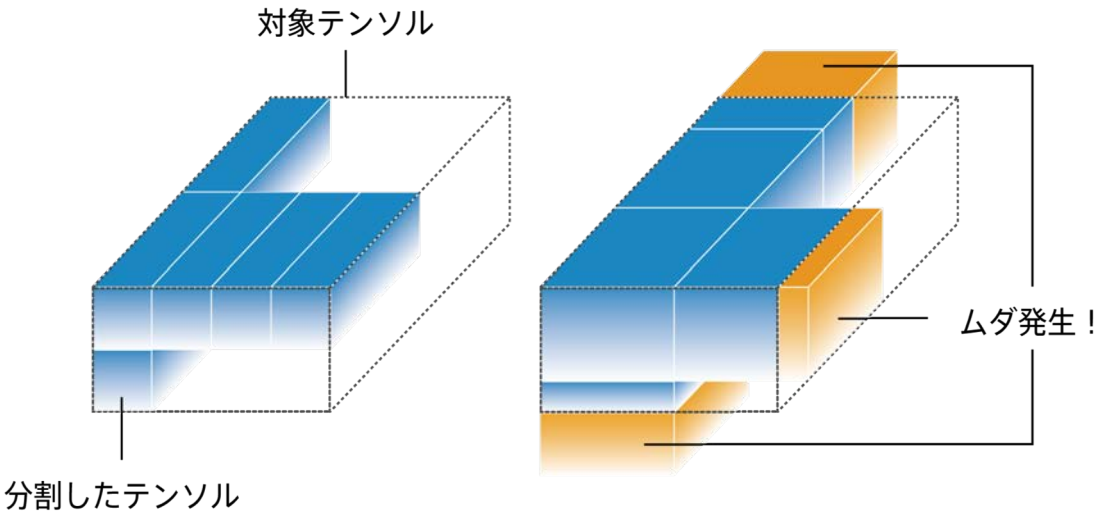
- オペレータごとの演算はテンソル計算であり、
  愚直にコード化すれば多重ループによって表現される
- よって、ハードウェアに合わせた最適化を施すことで
  かなりの高速化効果を得られる
  - アクセス順序、レジスタ数、キャッシュサイズ、etc.
- 高性能計算の分野で培われた技術がほぼそのまま流用できる

例)

- 並列化
- リオーダリング（または「ループ交換」）
- タイリング（または「ブロック化」）
  - ループを特定サイズに分割
- テンソル化
  - 一定の大きさのテンソルに分割、特別処理を施す
- ソフトウェアパイプライニング

ループの最適分割（１）

ループの分割サイズによって対象テンソルの処理効率性が変わる

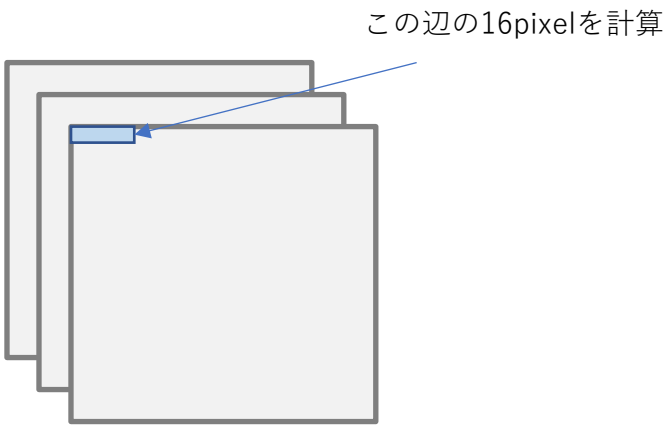


適切なループの分割　　不適切なループの分割

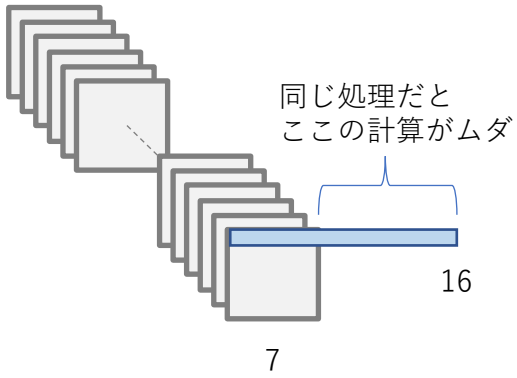
ループの最適分割（２）

テンソルのサイズはネットワーク内でも変化するため、
同じオペレータであっても最適解は異なる

長さ16のSIMD命令が使えるとして、



先頭に近い層：224 x 224 x 3



末尾に近い層：7 x 7 x 1280

Halide

- **アルゴリズム** と **スケジュール** の分離によって簡易にループ最適化を記述
- 本来は画像処理用だが、深層学習コンパイラにも多く転用されている
  - TensorComprehensions では本家Halideを取り込み
  - TVM では独自拡張したIRを採用

```
Func blur_3x3(Func input) {
  Func blur_x, blur_y;
  Var x, y, xi, yi;

  // The algorithm - no storage or order
  blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
  blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;

  // The schedule - defines order, locality; implies storage
  blur_y.tile(x, y, xi, yi, 256, 32)
        .vectorize(xi, 8).parallel(y);
  blur_x.compute_at(blur_y, x).vectorize(x, 8);

  return blur_y;
}
```

http://halide-lang.org/ より引用

深層学習固有の最適化（or 等価変換）

剪定 Pruning

- 影響の少ないニューロン（ノード）やシナプス（エッジ）を間引く処理

量子化 Quantization

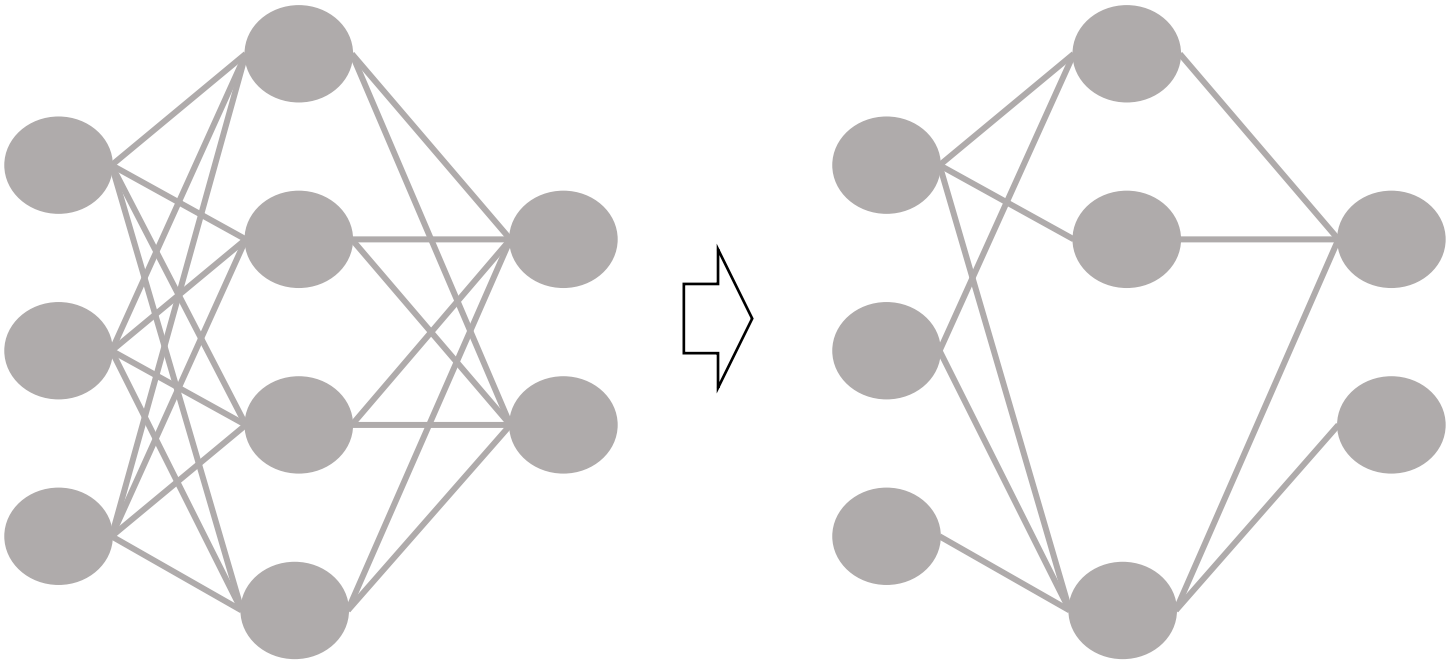
- 通常 32bit float でなされるネットワーク内部の計算/データを低ビット幅の int/fixed-point に置き換える処理

いずれも、

- 明らかな精度劣化を伴う
- 学習しながら、あるいは再学習とともに用いられる事も多く、その点でコンパイラの最適化と厳密にはいえない面もある（コンパイラのみで行われることもある）

剪定（Pruning）

目的に大きく影響しないオペレータ（ニューロン）、あるいは接続（シナプス）を刈る。
これにより、メモリフットプリントの圧縮と計算回数の削減が同時に図れる。
剪定により**テンソルが疎になる**ので、疎な行列をうまく扱えるハードウェアとともに用いると効果は大きい。
（GPUではあまり旨味がない）

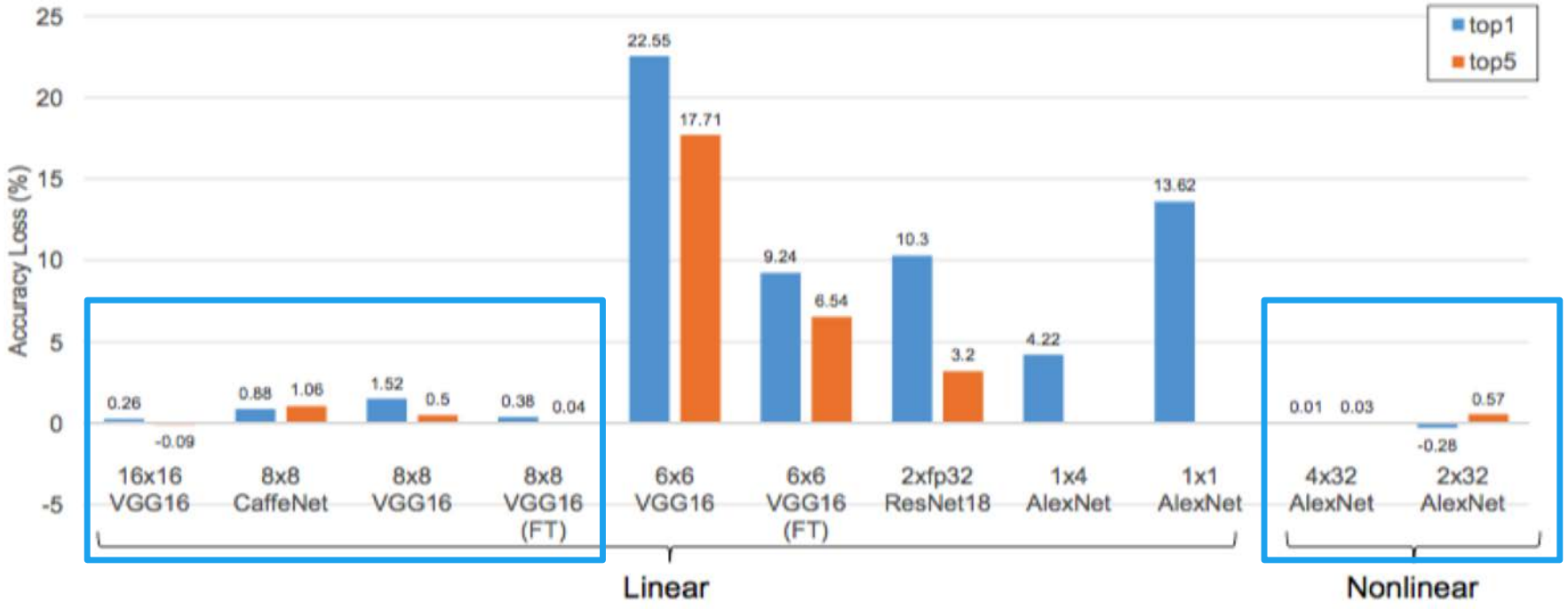


量子化（Quantization）

- DNNが保持するパラメータを低ビット幅に変換
  - 通常float32で計算されるところ、int8等に変換するのでこの名がある
- 全体でのデータ量を削減できる他、必要な計算処理ユニットのサイズを圧縮する効果がある
  - 特に1~2bitにすると乗算をXNORゲートで代用できるため、専用ハードウェアで用いた場合は計算効率は飛躍的に向上する
  - が、ビット幅を小さくするほど精度劣化は当然激しくなる

量子化によるビット幅と各ネットワークの精度低下の関連性

出典：A Survey of FPGA Based Neural Network Accelerator:　https://arxiv.org/abs/1712.08934

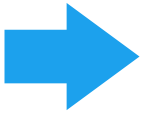


8ビット量子化であれば、1%程度の精度劣化に留まる

精度がどこまで重要か？

画像分類の場合、多少の精度劣化は影響しない

1. りんご 92.6%
2. みかん 35.3%
…

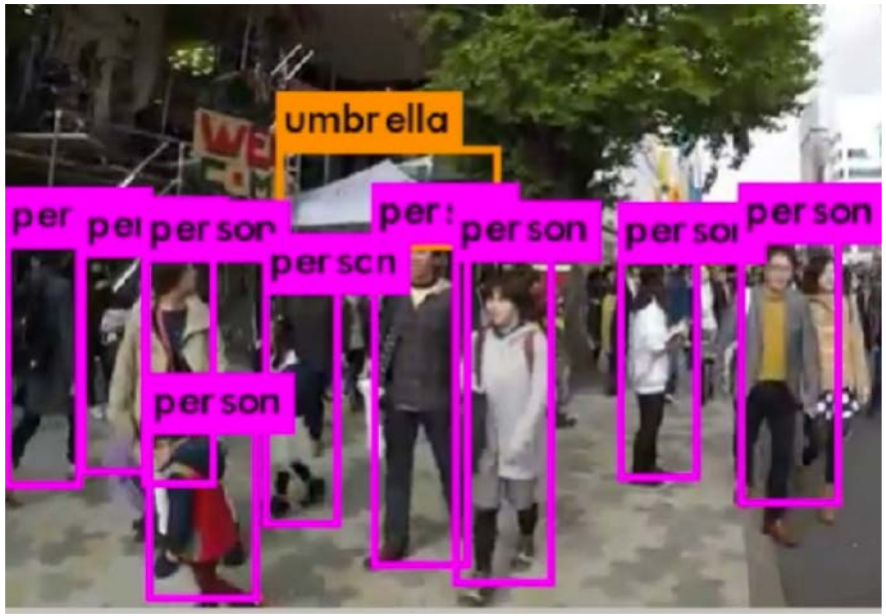


1. りんご 75%
2. みかん 52%
…

精度がどこまで重要か？

一方、たとえば物体検出の場合、
精度／誤差は領域の特定誤差に直接影響する

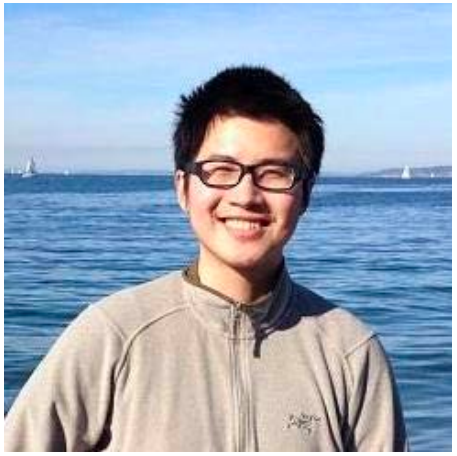
物体を覆う矩形(bounding box)が正しく描けなくなる



現在の技術開発動向
～TVMを中心に～

TVM

- ワシントン大学のグループが中心になって開発している深層学習コンパイラ
  - 作者：Tianqi Chen
    - 機械学習アルゴリズム XGBoost 作者
    - 深層学習フレームワークMXNet の初期開発者
  - OSSのコンパイラでは最も人気
    - GitHub stars: 3,509 (2019/5/27 現在)
    - Amazonが自社サービスに組み込み



TVMにおける
現在の研究開発方針

自動最適化

AutoTVM 機械学習を用いたオペレータ自動最適化

ハードウェア

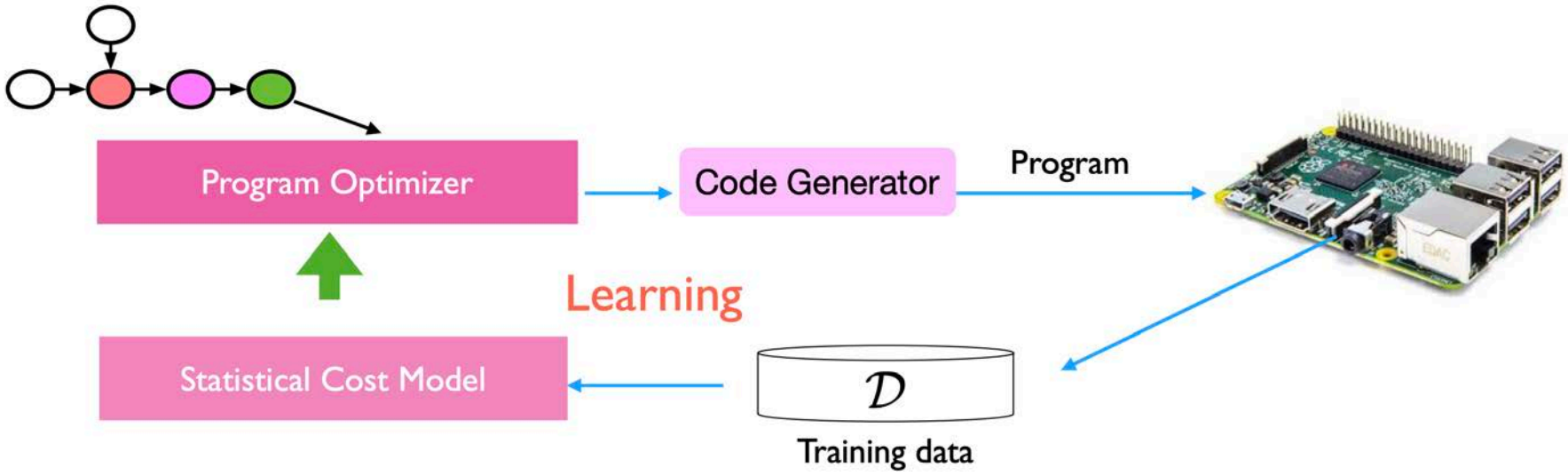
VTA（”ヴィータ”） TVM専用のAIチップ標準デザイン

中間言語

Relay 次期グラフレベル中間言語

AutoTVM

オペレータ最適化におけるスケジュールの自動化



AutoTVM

- 機械学習を使った最適パラメータ学習
  - TVMではGRU or XGBoost を採用
- 端緒はHalideでの自動スケジュール[1]
- TensorComprehensions でも同様の試み

参考文献）
[1] Mullapudi, R. T., Adams, A., Sharlet, D., Ragan-Kelley, J., & Fatahalian, K. (2016). Automatically scheduling halide image processing pipelines. *ACM Transactions on Graphics*, *35*(4), 1–11.

VTA

TVM専用チップ設計フレームワーク

2階層のISA

- CISCベースのマルチサイクル命令
  （DENSE, ALU, LOAD, STORE）
- RISCベースのマイクロ命令

TVMと連動したlatency hiding

- DNNコンパイル時に命令列内の依存関係を解析し
  その情報を用いたパイプラインを構成

- 現在はFPGAベース（エッジ、クラウド）
- 動作レベルシミュレータも

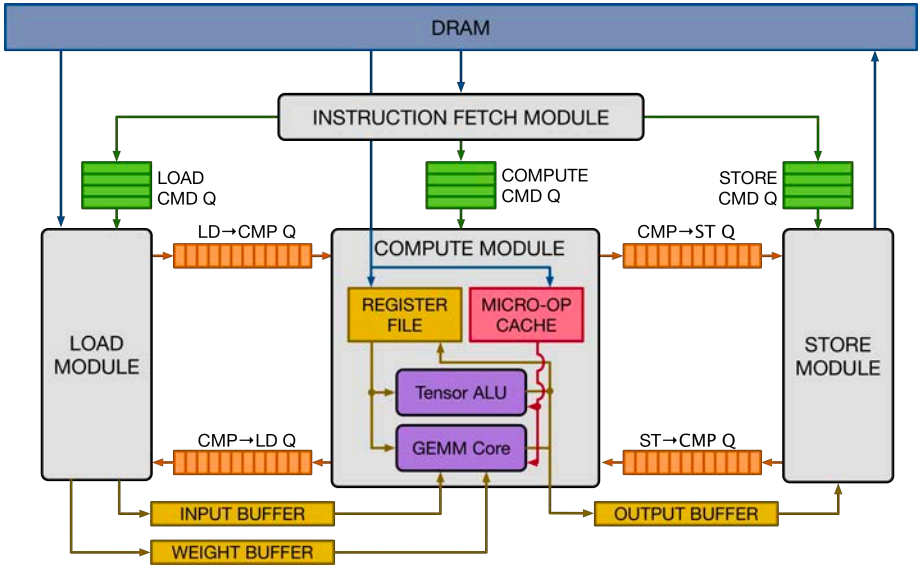
TVM + VTA スタック

https://sampl.cs.washington.edu/tvmconf/slides/Thierry-Moreau-VTA.pdf より

VTAアーキテクチャ

4つのモジュールから構成

- フェッチ
- ロード
- 計算（ALU or GEMM）
- ストア



Relay

- 深層学習フレームワーク／コンパイラの
  グラフレベル中間言語
- 静的型付き関数型言語
  - グラフ最適化 から （関数型）プログラム最適化 へ
  - shape のチェックを型検査で
- 自動微分可能な高階関数を採用
  - **Differential programming language**
    （『微分可能プログラミング言語』？）
    → TVM自身で学習も可能に

詳細：『TVMの次期グラフIR Relayの紹介』
https://www.slideshare.net/bonotake/tvmir-relay

参考：POPL2018 Keynote

Attending ▾ Program ▾ Tracks ▾ Organization ▾ Search Series ▾ Sign in Sign up

POPL 2018 (series) / Research Papers /

Some Principles of Differential Programming Languages

Track POPL 2018 Research Papers

When **Thu 11 Jan 2018 08:30 - 09:30 at Bunker Hill / Watercourt** - Keynote-II Chair(s): Andrew Myers

Abstract Languages for learning pose interesting foundational challenges. We look at one case: the foundations of differentiable programming languages with a first-class differentiation operator. Graphical and linguistic facilities for differentiation have proved their worth over recent years for deep learning (deep learning makes use of gradient descent optimisation methods to choose weights in neural nets). Automatic differentiation, also known as algorithmic differentiation, goes much further back, at least to the early 1960s, and provides efficient techniques for differentiation, e.g., via source code transformation. It seems fair to say, however, that differentiable programming languages have begun to appear only recently. It seems further fair to say that, while there has certainly been some foundational study of differentiable programming languages, there is ample opportunity to do more.

We investigate the semantics of a simple first-order functional programming language with reals, product types, and a first-class reverse-mode differentiation operator (reverse-mode differentiation generalises gradients). The semantics employs a standard concept of real analysis: smooth multivariate functions with open domain. It turns out that such partial functions fit well with programming constructs such as conditionals and recursion; indeed they form a complete partial order, and so standard fixed-point methods can be applied. From a

Gordon Plotkin
University of Edinburgh, UK

https://popl18.sigplan.org/details/POPL-2018-papers/76/Some-Principles-of-Differential-Programming-Languages より

参考：MLIR (Multi-Level Intermediate Representation)

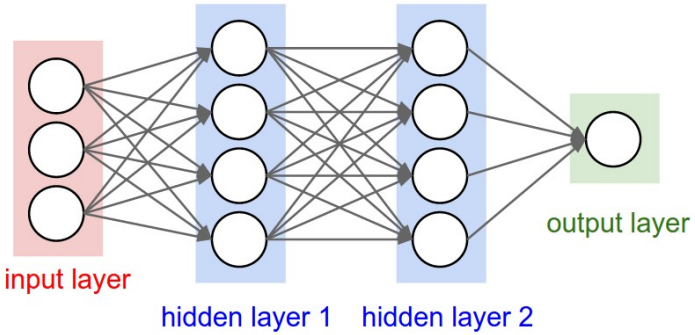
- Google / TensorFlowのグループが2019年から新たに提案している、深層学習コンパイラ向け中間表現、およびこれを使ったコンパイラフレームワーク
- 型付き、SSA（静的単一代入）形式の言語
  - スカラーの他、ベクター、テンソルの型をshape付きで与えられる

```
func @some_func(%arg0: !random_dialect<“custom_type”>) -> !another_dialect<“other_type”> {
  %result = “custom.operation”(%arg0) :
              (!random_dialect<“custom_type”>) -> !another_dialect<“other_type”>
  return %result : !another_dialect<"other_type">
}
```

- 複数レベルのIRを統一的に記述できることを目標
  - cf. TVM ... 現在は Relay と Halide IR の2種類が存在
    - TVMでも統一の動きがある模様

コンパイラの等価性について

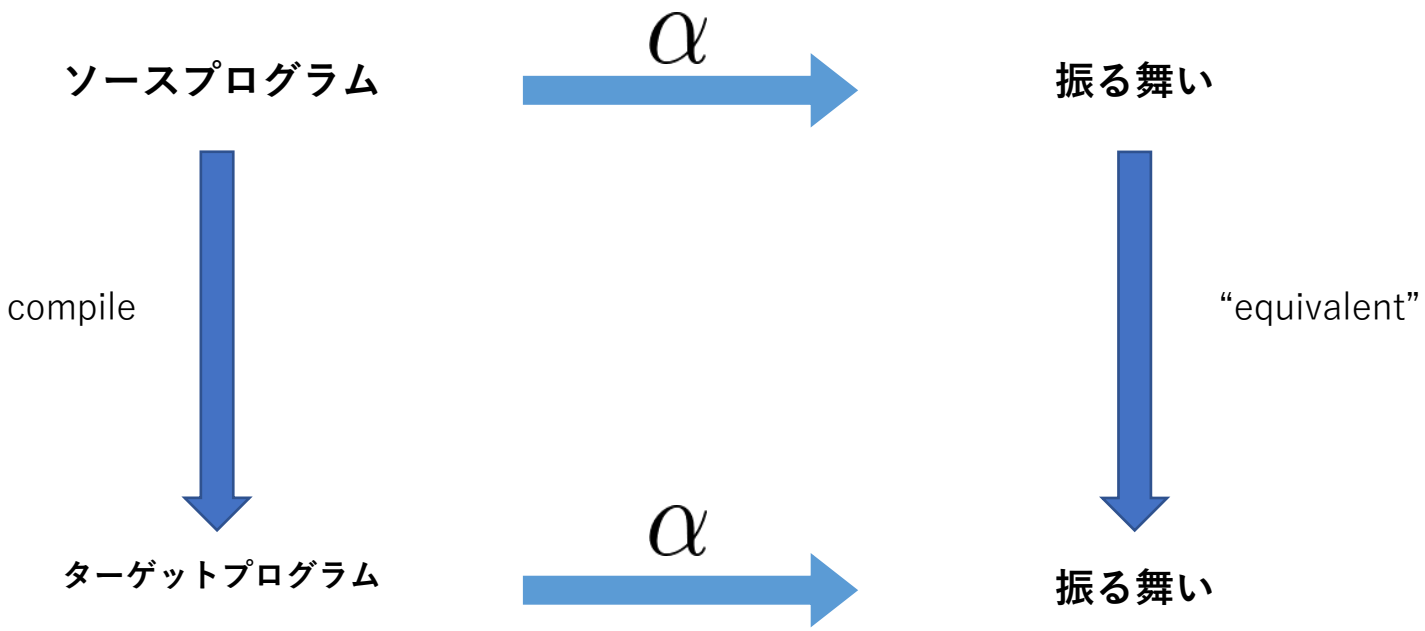
背景



- 深層学習コンパイラはときに精度劣化を伴う最適化をする
- つまり、コンパイル前後で違う結果が出力されうるが、それが許容されうる
- しかも、出力結果は離散値（カテゴリ値）
  - 例えば
    コンパイル前 -> 猫 に分類
    コンパイル後 -> 犬 に分類
- このようなコンパイラをテストするにはどうすれば？？

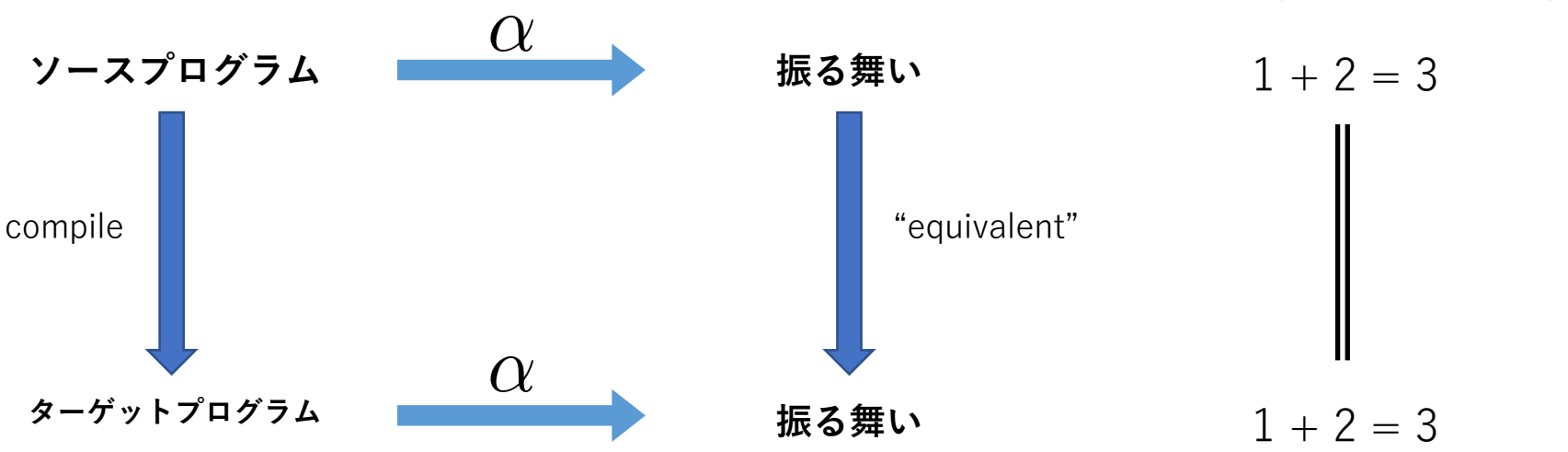
『正しいコンパイラ』とは？

普通のコンパイラ



『正しいコンパイラ』とは？(cont'd)

普通のコンパイラ



あるテスト（= 入出力のサンプル）

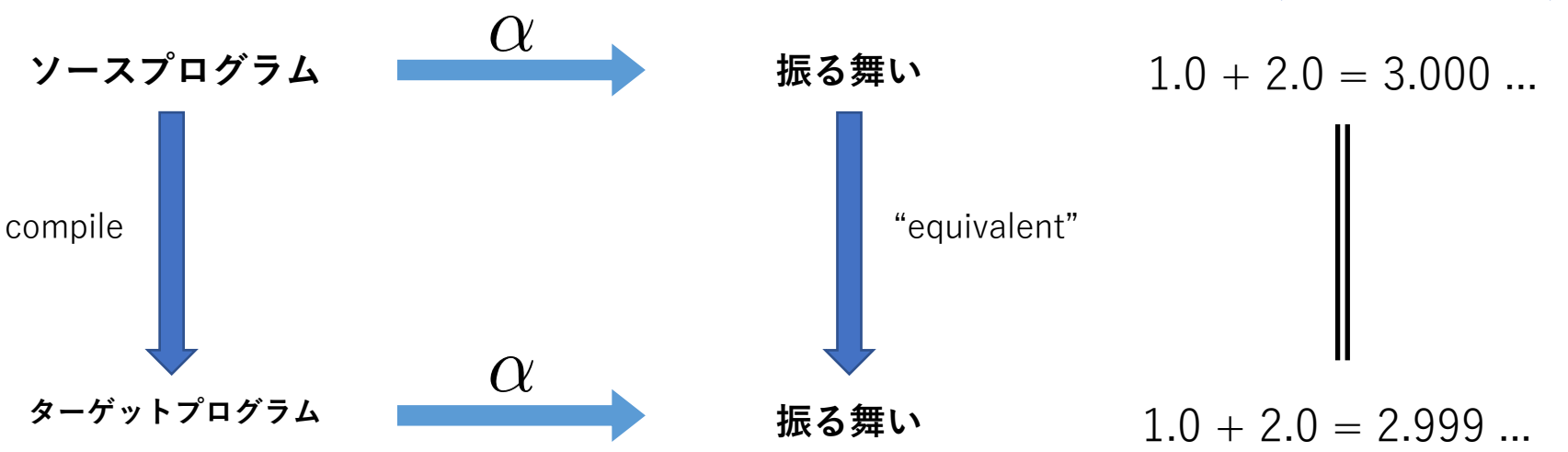
1 + 2 = 3

1 + 2 = 3

全ての入力に対して、計算結果は等しい

『正しいコンパイラ』とは？(cont'd)

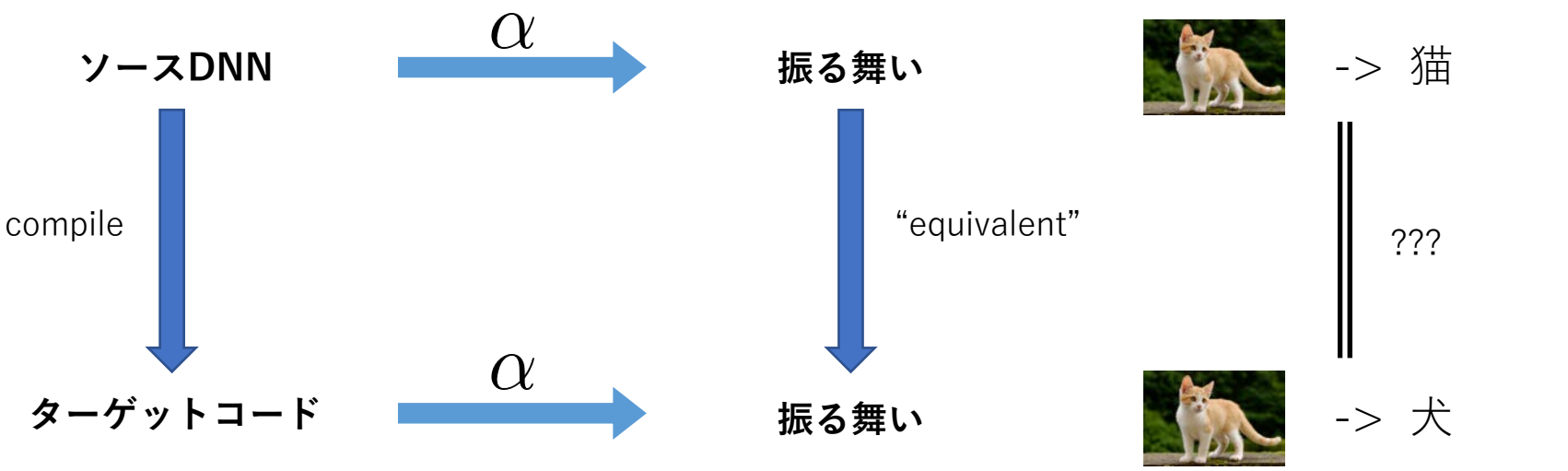
普通のコンパイラ



全ての入力に対して、計算結果は *許容誤差を無視すれば* 等しい

『正しいコンパイラ』とは？(cont'd)

深層学習コンパイラ



計算結果が異なる入力が存在するかもしれない

深層学習コンパイラの正しさに至るための課題

1. DNNの振る舞いとは何か？
   - あるいは、抽象化関数 α をどう定義すればよいか？
   - この振る舞いはコンパイル前後で保存されなければならない
2. DNNとコンパイル後のコードとの間の等価性とは何か？
3. その等価性をどう確かめればよいか？
   - テストケース1つでは確かめられない
   - 個々のテストケースではなく、テストセット全体での（統計的）評価が必要？
   - 精度でしか評価する方法はないのか？
4. その等価性に従えば、どのような最適化が許容されるのか？
   - 例えば、量子化は良いのか？ 何ビットまで許容されうるのか？

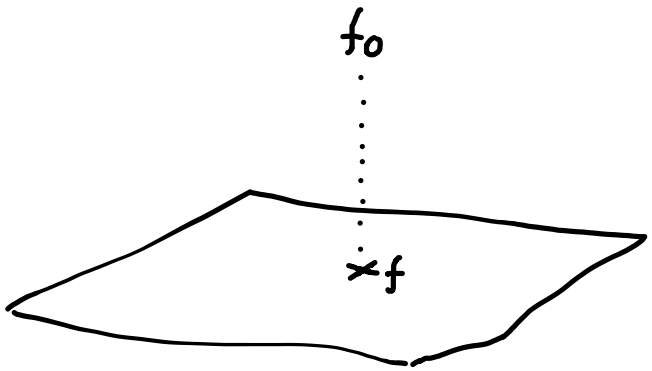
1. DNNの振る舞いとは何か？

- 教師あり学習において、DNNは関数 $\mathbf{f}$ であって、別の関数 $\mathbf{f}_0$ を近似するものとみなせる
- ここで $\mathbf{f}_0$ は教師データによって *暗黙的に* 定義される
- つまり、DNNの振る舞いはあまりにも不定であやふやなもの

1. DNNの振る舞いとは何か？(cont'd)

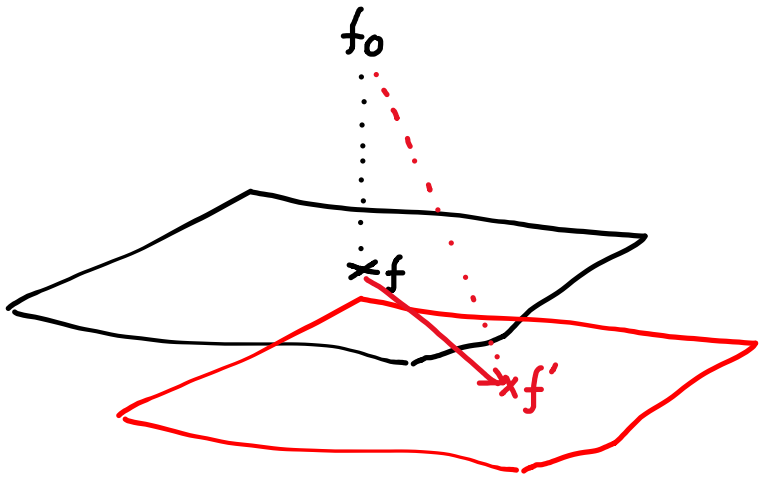
数学的（だがinformal）に考えるならば、

- 目標関数 $\mathbf{f}_0$ はある関数空間の中の1点
- 深層学習でのアルゴリズム／ネットワークはこの空間内の『平面』を定義するもの
- 学習済みDNNが表す関数 $\mathbf{f}$ は（理想的には）その平面内で $\mathbf{f}_0$ に最も近い点に相当する
  - 実際には学習が常に大局最適な解を導くとは限らないが、その近傍にあることはある程度期待して良い



では、コンパイラとは？

- 平面から他の平面への写像 m
- m によって f から写された点 f' はどうであれば『正しい』か？
  - fからの『距離』が小さければ良い？ それとも、
  - f_0からの『距離』が小さければ良い？
  - それらの『距離』を測る手段は？



4. どのような最適化が許容されるのか？

- 深層学習での等価性を評価するためには、通常のコンパイラで考えるような等価性から基準を緩和する必要がありそう
  - 全体で正しければ、個々の入力では*間違っても良い*
- 裏を返せば、緩和された等価性の下ではより大胆な最適化を採用できる
  - 今までは、基準もなく闇雲な最適化が使われてきた
  - 基準が定まれば、採用できる／できない最適化が分別でき、*安心して*最適化を適用できる

ご清聴
ありがとうございました