StrassenNets: Deep Learning with a Multiplication Budget

Michael Tschannen*

ETH zürich

michaelt@nari.ee.ethz.ch

13 July 2018

Joint work with Aran Khanna* and Anima Anandkumar*

*work done at Amazon AI

Motivation

Outstanding predictive performance of deep neural networks (DNNs) comes at the cost of **high computational complexity and high energy consumption**.

Motivation

Outstanding predictive performance of deep neural networks (DNNs) comes at the cost of **high computational complexity and high energy consumption**.

Known solutions

- Architectural optimizations
  [Iandola et al. 2016, Howard et al. 2017, Zhang et al. 2017]
- Factorizations of weight matrices and tensors
  [Denton et al. 2014, Novikov et al. 2015, Kossaifi et al. 2017, Kim et al. 2017]
- Pruning of weights and filters
  [Liu et al. 2015, Wen et al. 2016, Labedev et al. 2016,]
- Reducing numerical precision of weights and activations
  [Courbariaux et al. 2015, Rastegari et al. 2016, Zhou et al. 2016, Lin et al., 2017]

Motivation

Our approach: **Reducing the number of multiplications as a guiding principle**

Motivation

Our approach: **Reducing the number of multiplications as a guiding principle**

- This strategy led to many **fast algorithms**
  - Strassen's matrix multiplication algorithm
  - Winograd-filter based convolution [Gray & Lavin 2016]

Motivation

Our approach: **Reducing the number of multiplications as a guiding principle**

- This strategy led to many **fast algorithms**
  - Strassen's matrix multiplication algorithm
  - Winograd-filter based convolution [Gray & Lavin 2016]
- DNNs with $\{-1, 0, 1\}$-valued weights have **60% higher throughput** on FPGA than on GPU, while being **2.3$\times$ better in performance/watt** [Nurvitadhi et al. 2017]

Motivation

Our approach: **Reducing the number of multiplications as a guiding principle**

- This strategy led to many **fast algorithms**
  - Strassen's matrix multiplication algorithm
  - Winograd-filter based convolution [Gray & Lavin 2016]
- DNNs with $\{-1, 0, 1\}$-valued weights have **60% higher throughput** on FPGA than on GPU, while being **2.3$\times$ better in performance/watt** [Nurvitadhi et al. 2017]
- Multiplications take **up to 32$\times$ more cycles** than additions on (low-end) MCUs

Motivation

Our approach: **Reducing the number of multiplications as a guiding principle**

- This strategy led to many **fast algorithms**
  - Strassen's matrix multiplication algorithm
  - Winograd-filter based convolution [Gray & Lavin 2016]
- DNNs with $\{-1, 0, 1\}$-valued weights have **60% higher throughput** on FPGA than on GPU, while being **2.3$\times$ better in performance/watt** [Nurvitadhi et al. 2017]
- Multiplications take **up to 32$\times$ more cycles** than additions on (low-end) MCUs
- Additions are **more area-efficient** and hence **much less energy consuming** (3–30$\times$ [Horowitz 2014]) than multiplications on ASIC

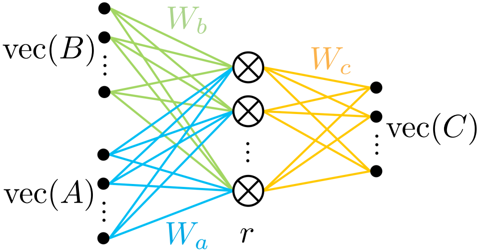
Casting matrix multiplications as 2-layer sum-product networks (SPNs)

A large fraction of arithmetic operations in DNNs are due to matrix multiplications

Casting matrix multiplications as 2-layer sum-product networks (SPNs)

A large fraction of arithmetic operations in DNNs are due to matrix multiplications

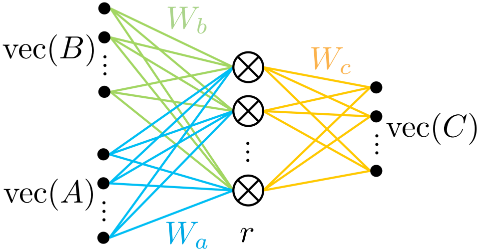
$$C = AB \quad \Longleftrightarrow$$



Casting matrix multiplications as 2-layer sum-product networks (SPNs)

A large fraction of arithmetic operations in DNNs are due to matrix multiplications

$$C = AB \qquad \Longleftrightarrow$$

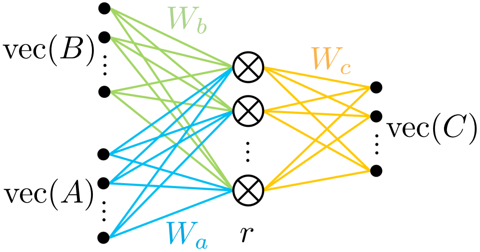


- A is $k \times m$, B is $m \times n$: Ternary ($\{-1, 0, 1\}$) W_a, W_b, W_c exist if $r \geq nmk$

Casting matrix multiplications as 2-layer sum-product networks (SPNs)

A large fraction of arithmetic operations in DNNs are due to matrix multiplications

$$C = AB \quad \Longleftrightarrow$$

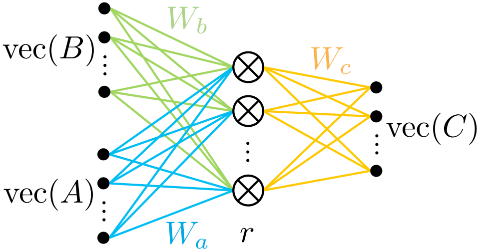


- A is $k \times m$, B is $m \times n$: Ternary ($\{-1, 0, 1\}$) W_a, W_b, W_c exist if $r \geq nmk$
- A, B are 2×2: Strassen's algorithm: Ternary W_a, W_b, W_c for $r = 7$

Casting matrix multiplications as 2-layer sum-product networks (SPNs)

A large fraction of arithmetic operations in DNNs are due to matrix multiplications

$$C = AB \quad \Longleftrightarrow$$



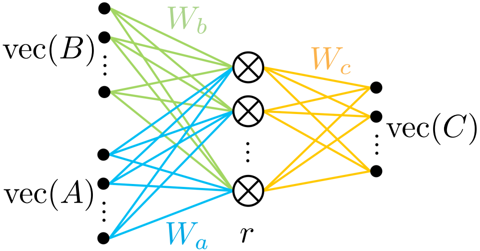
Change assumptions

- A fixed, B distributed on low-dimensional "manifold": Can realize **approximate** multiplication for $r \ll nmk$

Casting matrix multiplications as 2-layer sum-product networks (SPNs)

A large fraction of arithmetic operations in DNNs are due to matrix multiplications

$$C = AB \quad \Longleftrightarrow$$

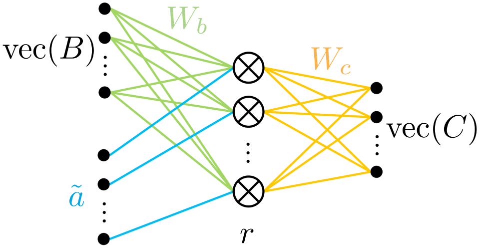


Idea: Associate A with the weights/filters and B with the activations/feature maps and learn W_a, W_b, W_c with $r \ll nmk$ end-to-end. Alternatively, learn $\tilde{a} = W_a \text{vec}(A)$ from scratch.

Casting matrix multiplications as 2-layer sum-product networks (SPNs)

A large fraction of arithmetic operations in DNNs are due to matrix multiplications

$$C = AB \quad \Longleftrightarrow$$



Idea: Associate A with the weights/filters and B with the activations/feature maps and learn W_a, W_b, W_c with $r \ll nmk$ end-to-end. Alternatively, learn $\tilde{a} = W_a \mathrm{vec}(A)$ from scratch.

Application to 2D convolution

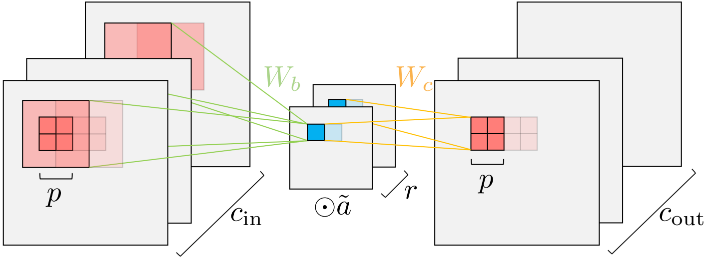
- Write convolution as matrix multiplication (`im2col`)
  $\rightarrow$ impractically large W_a, W_b, W_c

Application to 2D convolution

- Write convolution as matrix multiplication (`im2col`)
  $\rightarrow$ impractically large W_a, W_b, W_c
- Compress computation of $c_{\text{out}} \times p \times p$ outputs from $c_{\text{in}} \times (p-1+k) \times (p-1+k)$ inputs

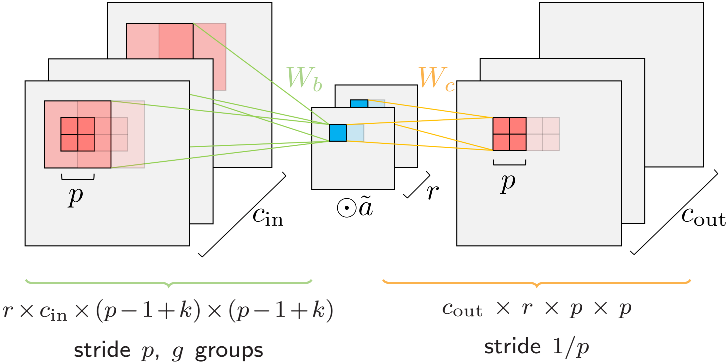
Application to 2D convolution

- Write convolution as matrix multiplication (`im2col`)
  $\rightarrow$ impractically large W_a, W_b, W_c
- Compress computation of $c_{\text{out}} \times p \times p$ outputs from $c_{\text{in}} \times (p-1+k) \times (p-1+k)$ inputs



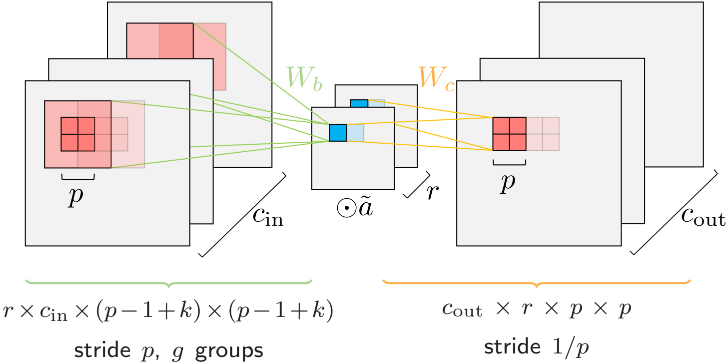
Application to 2D convolution

- Write convolution as matrix multiplication (`im2col`)
  $\rightarrow$ impractically large W_a, W_b, W_c
- Compress computation of $c_{\text{out}} \times p \times p$ outputs from $c_{\text{in}} \times (p-1+k) \times (p-1+k)$ inputs



Application to 2D convolution

- Write convolution as matrix multiplication (`im2col`)
  $\rightarrow$ impractically large W_a, W_b, W_c
- Compress computation of $c_{\text{out}} \times p \times p$ outputs from $c_{\text{in}} \times (p-1+k) \times (p-1+k)$ inputs $\rightarrow$ **multiplication reduction by a factor of $\mathbf{c_{in} c_{out} k^2 p^2 / r}$**

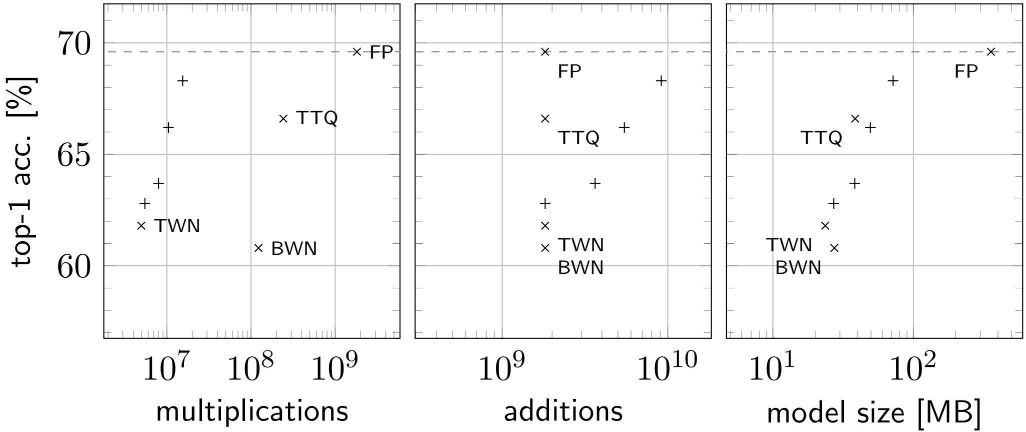


Training

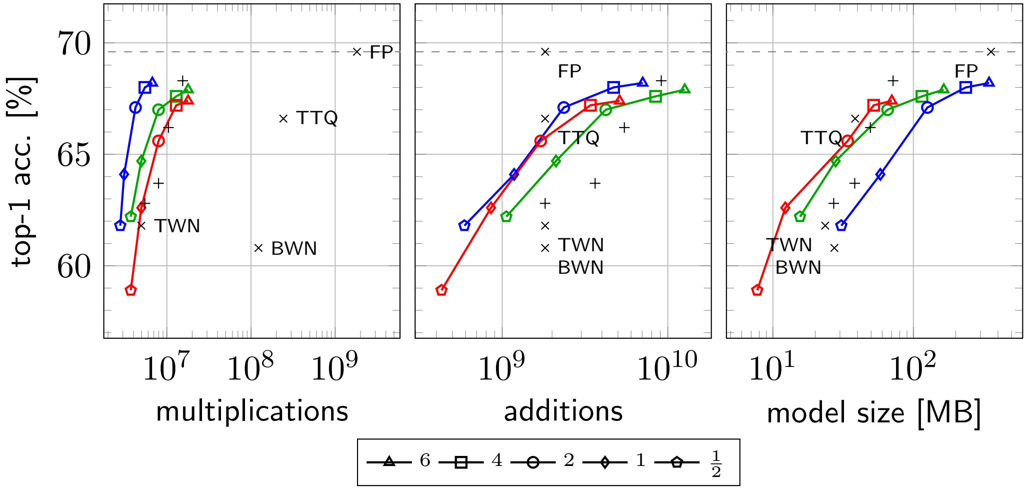
- SGD with momentum
- Quantize $(W_a), W_b, W_c$ with method described by [Li et al. 2016]
  - Quantization in the forward pass
  - Straight-through gradient estimator for backward pass
  - Gradient step on full-precision weights
- Pretraining with full-precision weights
- Knowledge distillation [Hinton et al. 2015]

$$\mathcal{L}_{\mathrm{KD}}(f_{\mathrm{S}}, f_{\mathrm{T}}; x, y) = (1-\lambda)\mathcal{L}(f_{\mathrm{S}}(x), y) + \lambda \mathrm{CE}(f_{\mathrm{S}}(x), f_{\mathrm{T}}(x))$$

Experiment: ResNet-18 on ImageNet

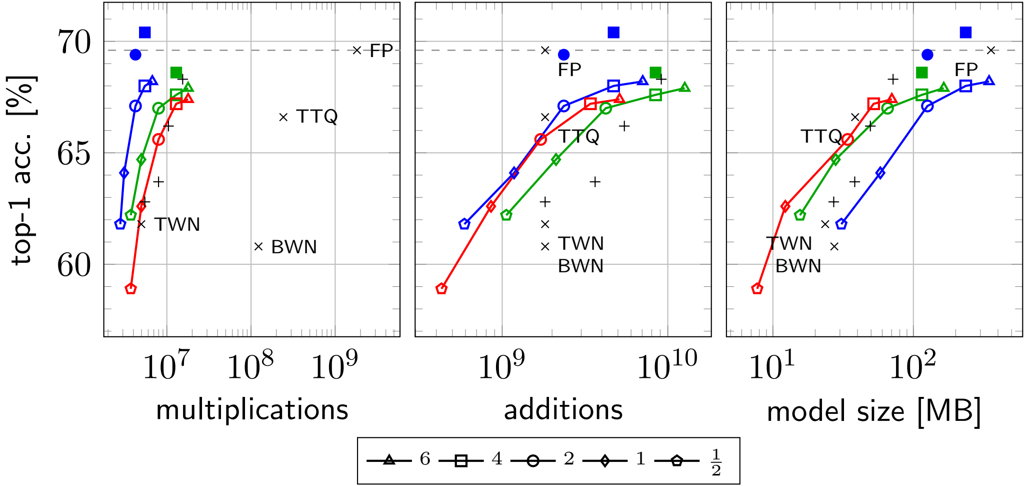


Experiment: ResNet-18 on ImageNet



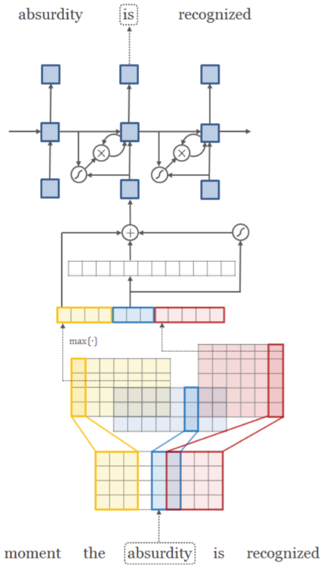
blue: $p = 2$, $g = 1$; green: $p = 1$, $g = 1$; red: $p = 1$, $g = 4$; marker type: r/c_{out}

Experiment: ResNet-18 on ImageNet



blue: $p = 2,\ g = 1$; green: $p = 1,\ g = 1$; red: $p = 1,\ g = 4$; marker type: r/c_{out}

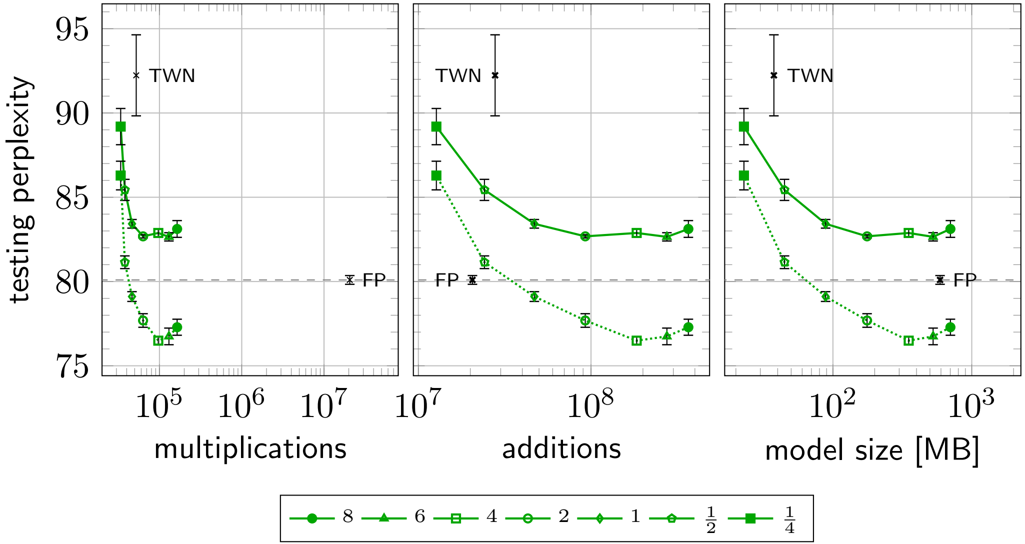
Experiment: Character-CNN language model on Penn Tree Bank



Compact model proposed by [Kim et al. 2016]

- Word-level decoder
- 2-layer LSTM, 650 units
- 2-layer highway network, 650 units
- Convolution layer, 1100 filters
- Character-level embedding

Experiment: Character-CNN language model on Penn Tree Bank



Rediscovering Strassen's algorithm

Learn to multiply 2×2 matrices using 7 multiplications
$W_a, W_b \in \{-1, 0, 1\}^{7\times4}$, $W_c \in \{-1, 0, 1\}^{4\times7}$ $\rightarrow$ solution space size $3^{3\cdot4\cdot7} = 3^{84}$

- L2-loss, 100k synthetic training examples, 25 random initializations:

$$W_a = \begin{pmatrix} -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & 0 & 1 & 0 \\ -1 & -1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}, W_b = \begin{pmatrix} -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ -1 & -1 & -1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & -1 & 0 \end{pmatrix}, W_c = \begin{pmatrix} 1 & 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 1 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Summary & Outlook

- Proposed and evaluated a **versatile framework** to learn fast approximate matrix multiplications for DNNs end-to-end
- Over **99.5% multiplication reduction** in image classification and language modeling applications while **maintaining predictive performance**
- Method can learn **fast exact** 2×2 **matrix multiplication**

Summary & Outlook

- Proposed and evaluated a **versatile framework** to learn fast approximate matrix multiplications for DNNs end-to-end
- Over **99.5% multiplication reduction** in image classification and language modeling applications while **maintaining predictive performance**
- Method can learn **fast exact 2×2 matrix multiplication**
- Application to more layer types (e.g., group equivariant convolutions, deformable convolutions)
- MCU/FPGA/ASIC implementation, end-to-end integration with hardware platforms
- Learning fast exact transforms

Thank you!

Poster #99

michaelt@nari.ee.ethz.ch

Code: `http://bit.ly/2Akmerp`

Pseudocode

```
W_B = Quantize(W_B)
W_C = Quantize(W_C)
conv_out = Conv2d(
               data=in_data,
               weights=W_B,
               in_channels=c_in,
               out_channels=r,
               kernel_size=p − 1 + k,
               stride=p,
               groups=g)
mul_out = Multiply(
               data=conv_out,
               weights=a_tilde)
out_data = ConvTranspose2d(
               data=mul_out,
               weights=W_C,
               in_channels=r,
               out_channels=c_out,
               kernel_size=p,
               stride=p)
```