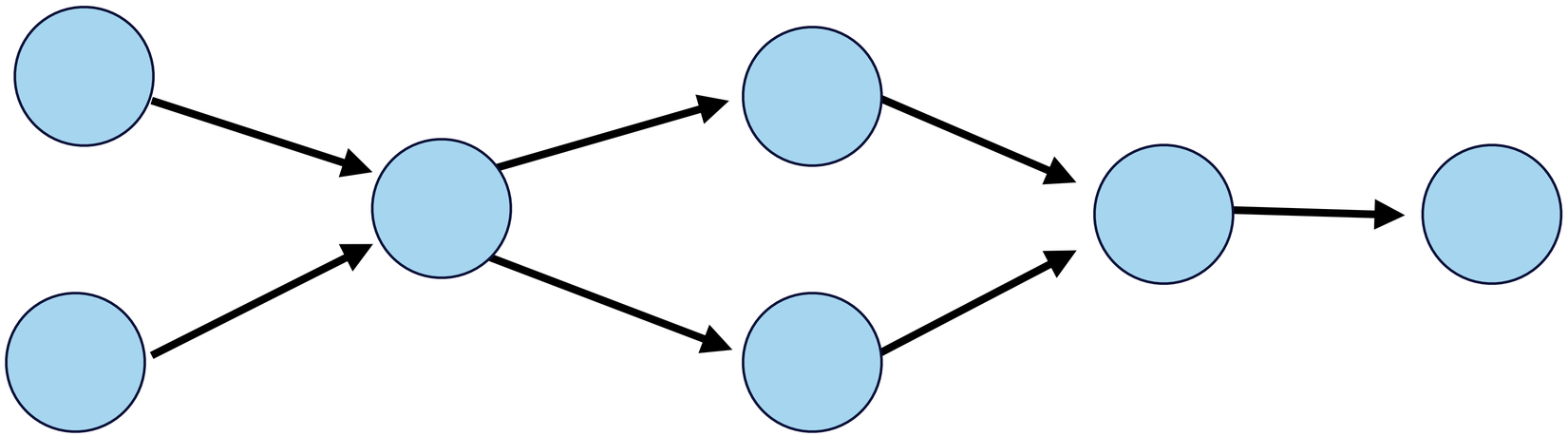
Deep Learning Systems

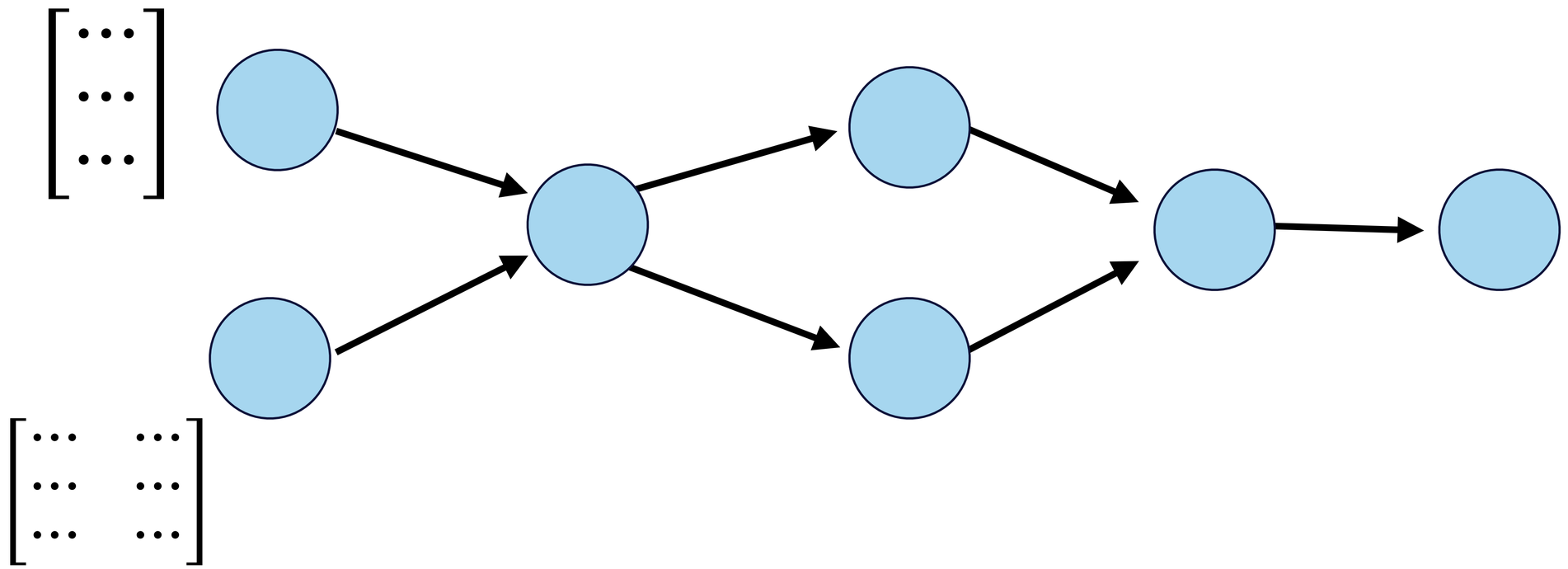
UCSD CSE 291 Differentiable Programming

Tzu-Mao Li

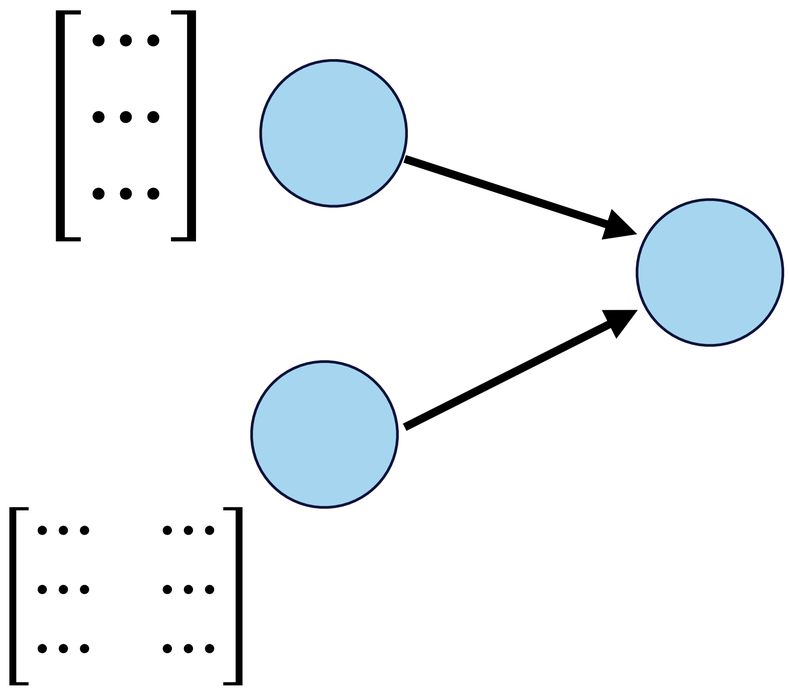
Recall: computational graph



Deep learning systems:
each of the node is a multi-dimensional array



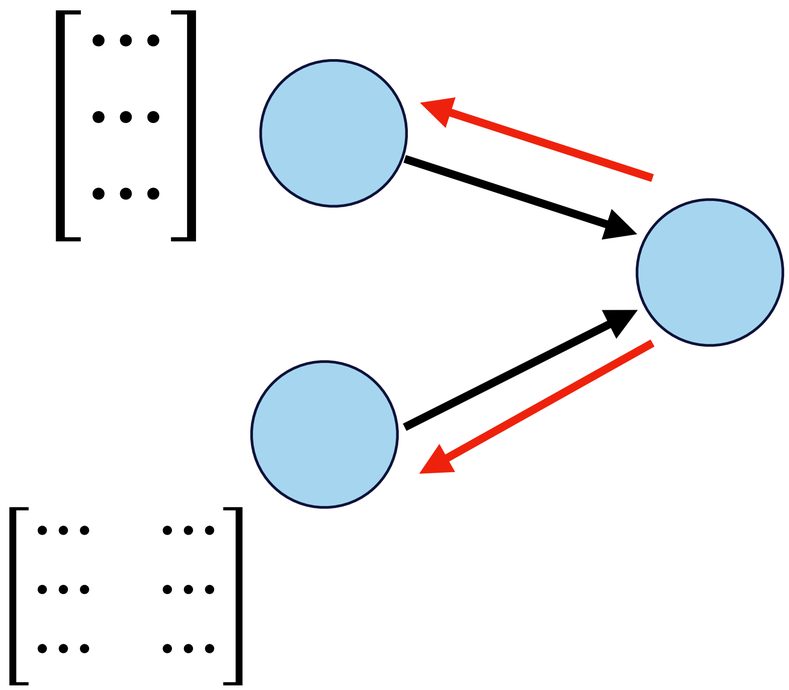
How to deep learning



Implement a bunch of operators
on multidimensional arrays

```
torch.add
torch.matmul
torch.nn.Conv2d
torch.nn.ReLU
torch.fft
…
```

How to deep learning



Implement a bunch of operators
on multidimensional arrays

```
torch.add
torch.matmul
torch.nn.Conv2d
torch.nn.ReLU
torch.fft
…
```

```
torch.add
torch.matmul
torch.nn.ConvTransposed2d
torch.nn.ReLU
torch.ifft
…
```

and their adjoints

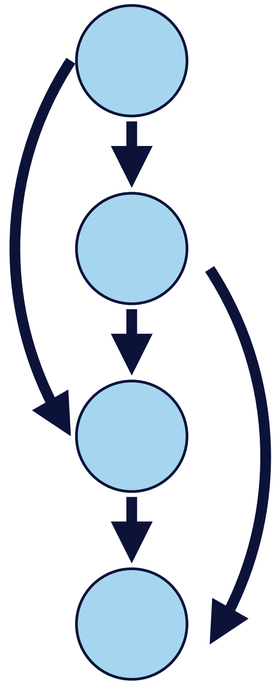
How to deep learning

Use taping to record the forward computation

we always record the operations,
optionally, compute the values (aka “eager mode”)

```
x = torch.tensor(…)
y = 2 * x
for i in range(10):
  y = torch.sin(y)
```

taping
(trace)

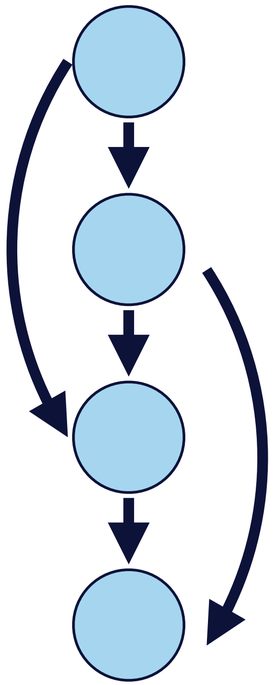


How to deep learning

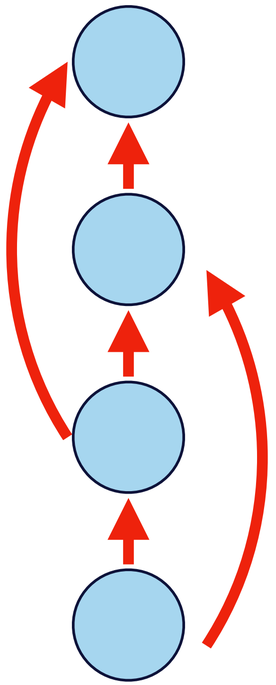
run the tape backwards to compute gradients

```
x = torch.tensor(…)
y = 2 * x
for i in range(10):
  y = torch.sin(y)
```

taping
(trace)



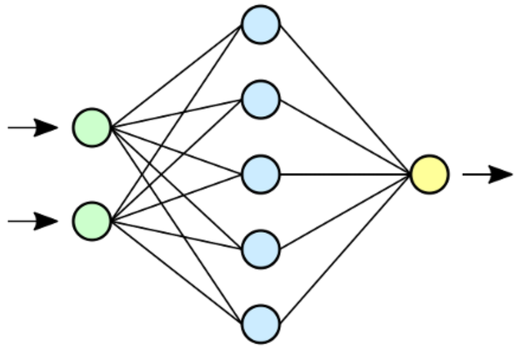
reverse
mode



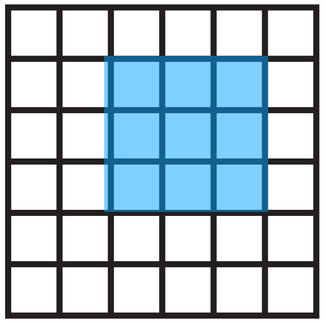
voila! we have a deep learning framework

Why does the deep learning framework strategy work?

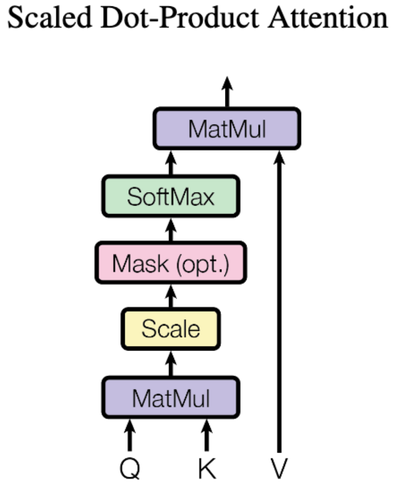
- observation 1: most deep learning operations are matrix multiplications or element wise operations



"multilayer perceptron"
matrix multiplication

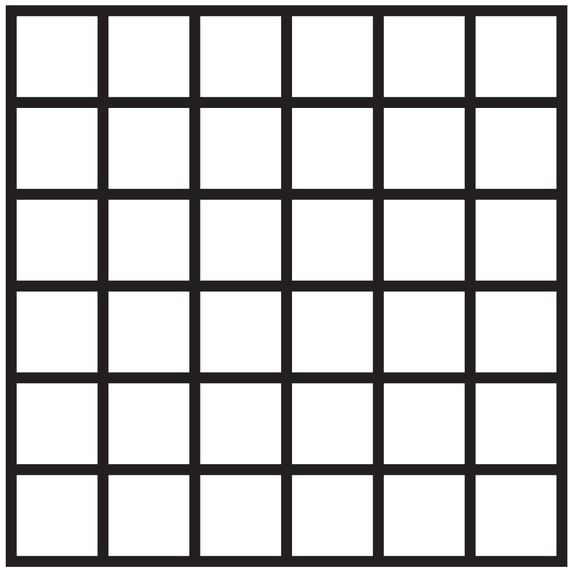


convolution layer
matrix multiplication
(how?)

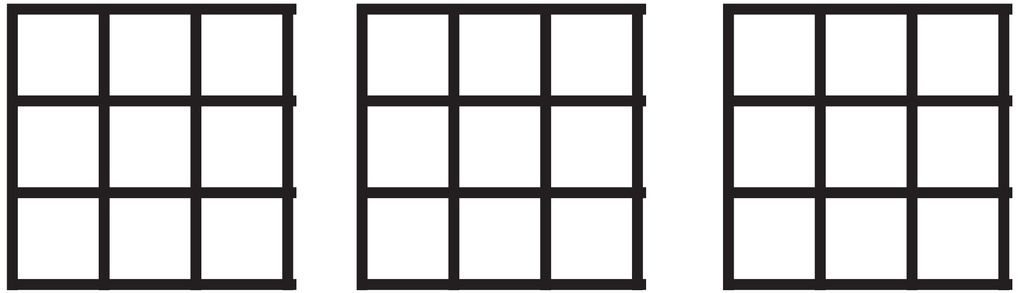


attention
matrix multiplication

Sidetrack: how to make convolution a matrix multiplication?

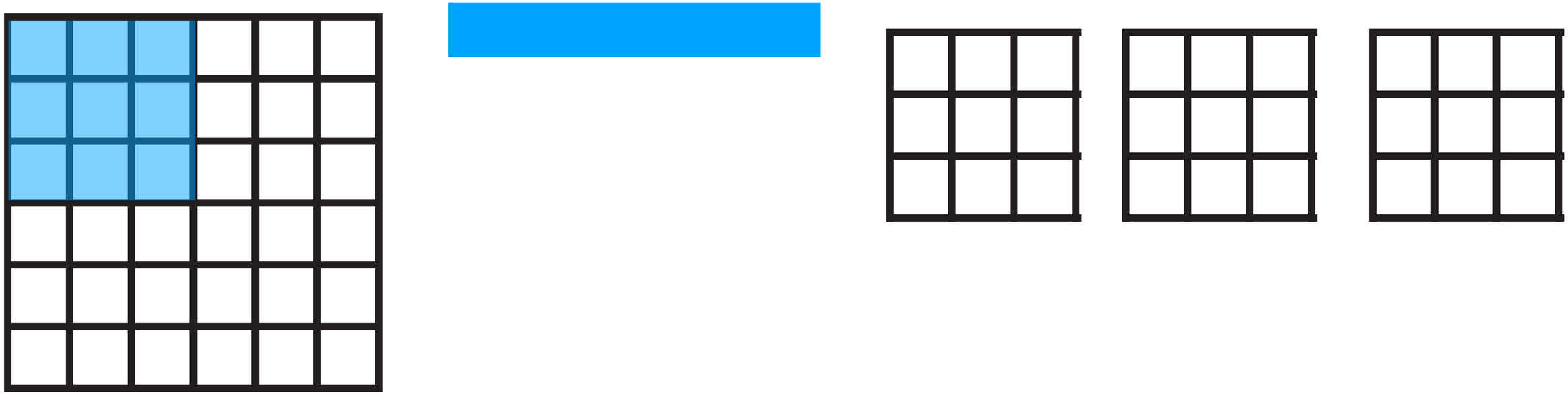


image



kernels

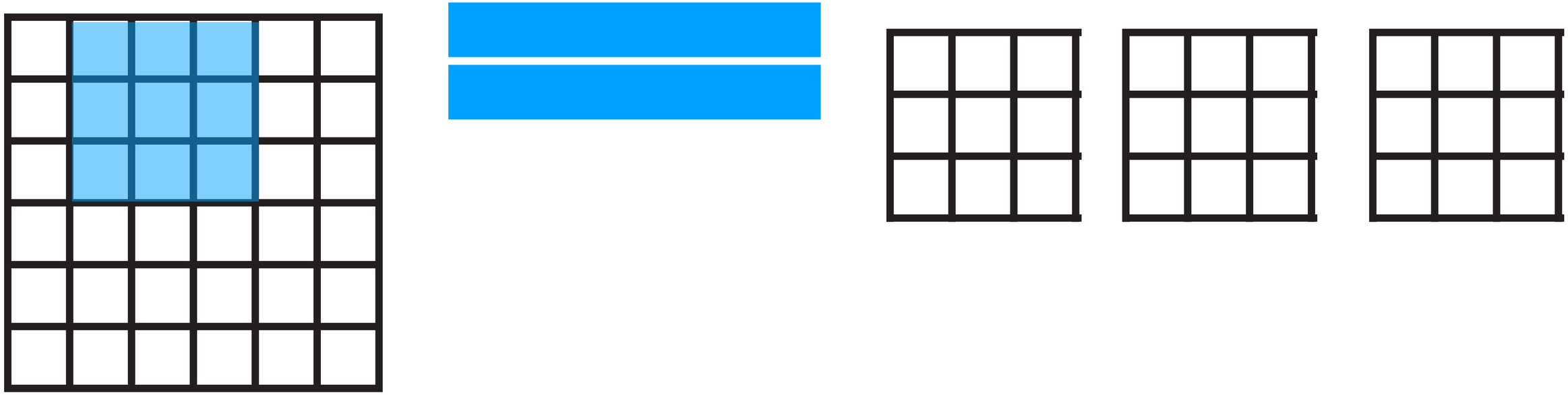
Sidetrack: how to make convolution a matrix multiplication?



image

kernels

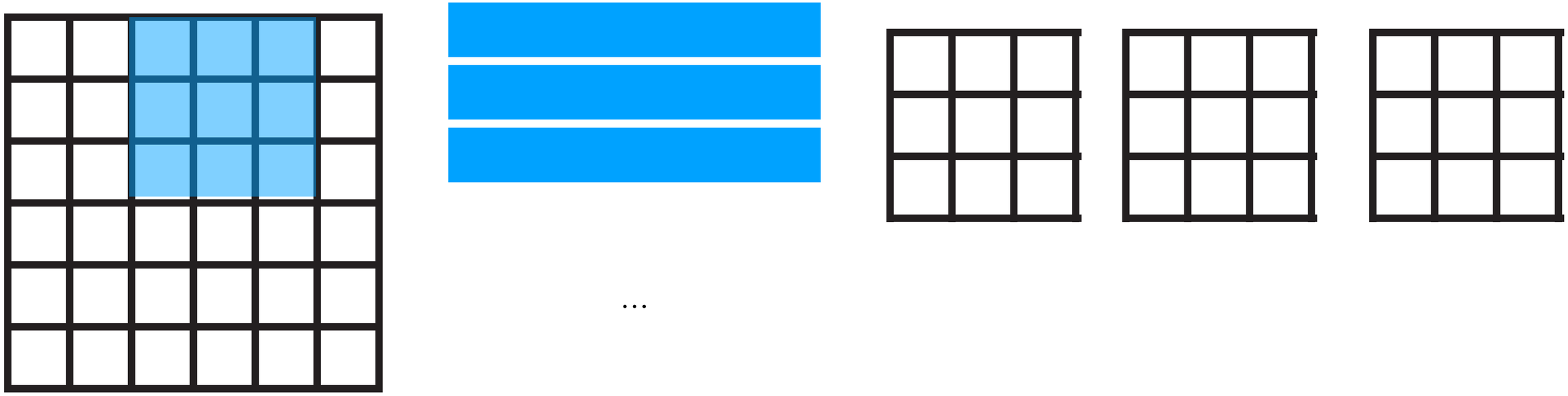
Sidetrack: how to make convolution a matrix multiplication?



image

kernels

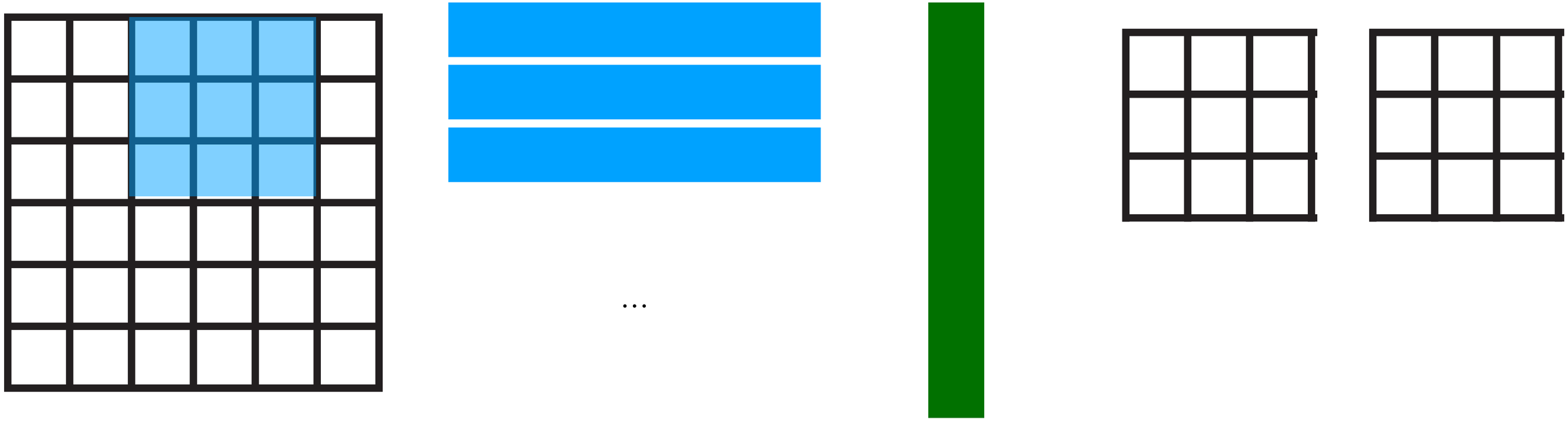
Sidetrack: how to make convolution a matrix multiplication?



image

kernels

Sidetrack: how to make convolution a matrix multiplication?



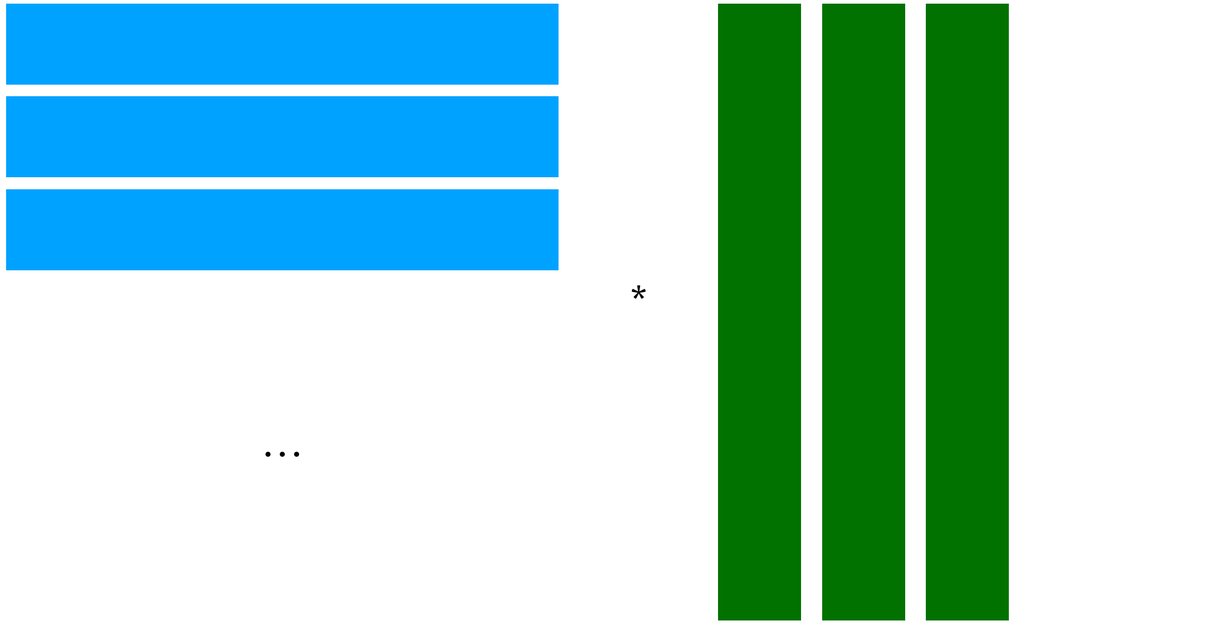
image

kernels

Sidetrack: how to make convolution a matrix multiplication?

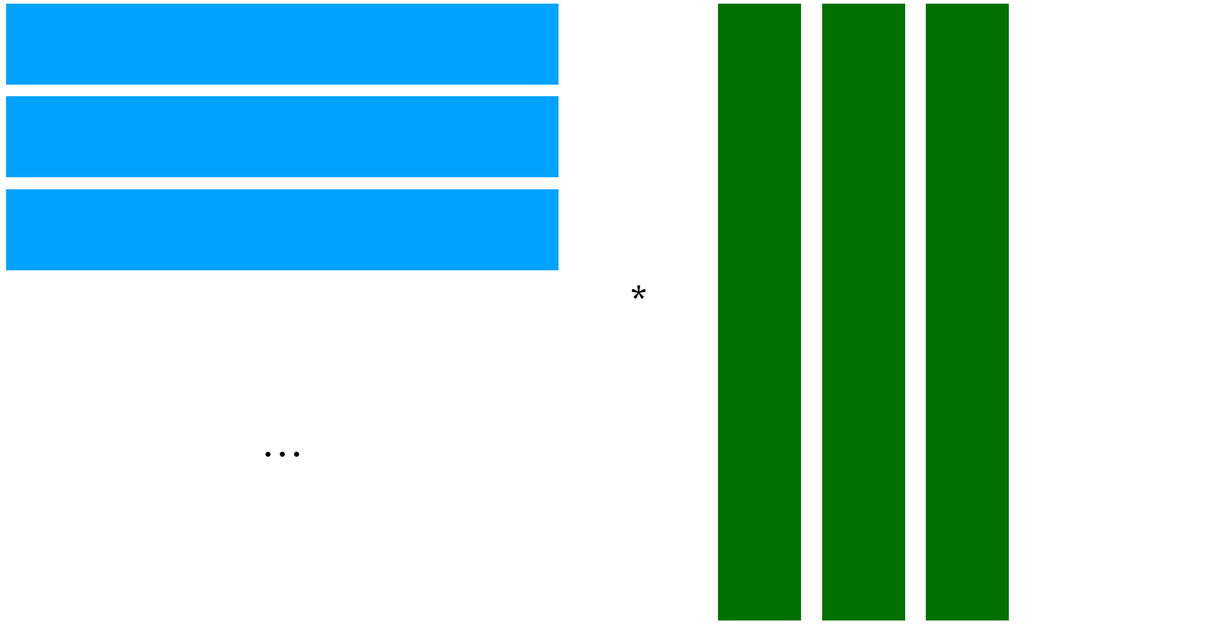
Sidetrack: how to make convolution a matrix multiplication?

this conversion is often called “im2col”



Sidetrack: how to make convolution a matrix multiplication?

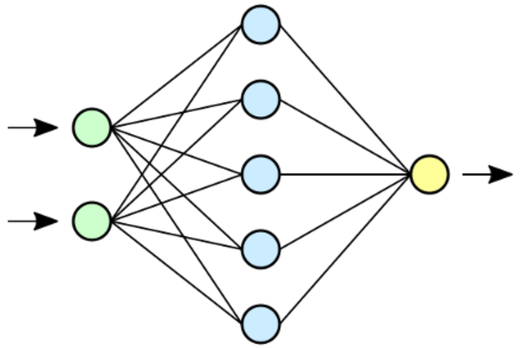
this conversion is often called “im2col”



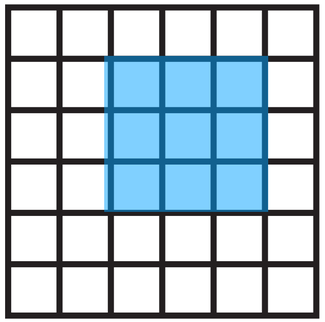
a specialized routine to perform this matrix multiplication is called “Winograd convolution”

Why does the deep learning framework strategy work?

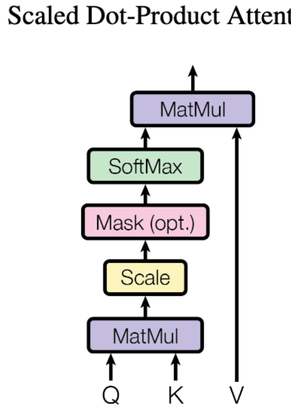
- observation 1: most deep learning operations are matrix multiplications or element wise operations
- therefore if we have an optimized matmul routine, we optimize all deep learning



"multilayer perceptron"
matrix multiplication



convolution layer
matrix multiplication



attention
matrix multiplication

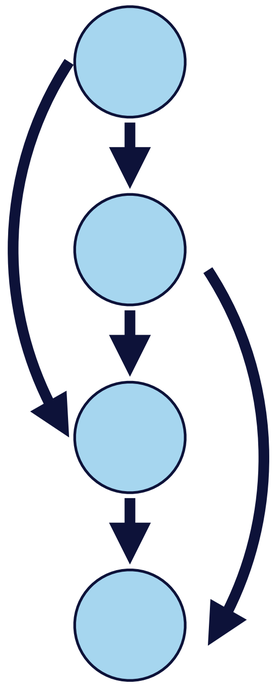
Why does the deep learning framework strategy work?

- observation 1: most deep learning operations are matrix multiplications or element wise operations
  - therefore if we have an optimized matmul routine, we optimize all deep learning

- observation 2: deep learning computation has high arithmetic intensity, thus we do not need to aggressively fuse the computations
  - it's further helped by "batching": we can process a huge amount of data at once to increase arithmetic intensity

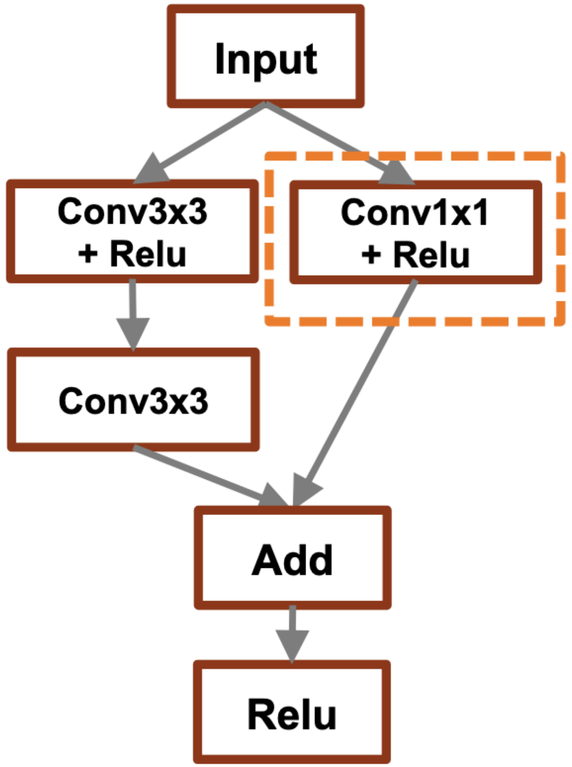
Bells and whistles: graph optimization

given a computational graph of multidimensional arrays (can be the adjoint code)
we can often optimize it



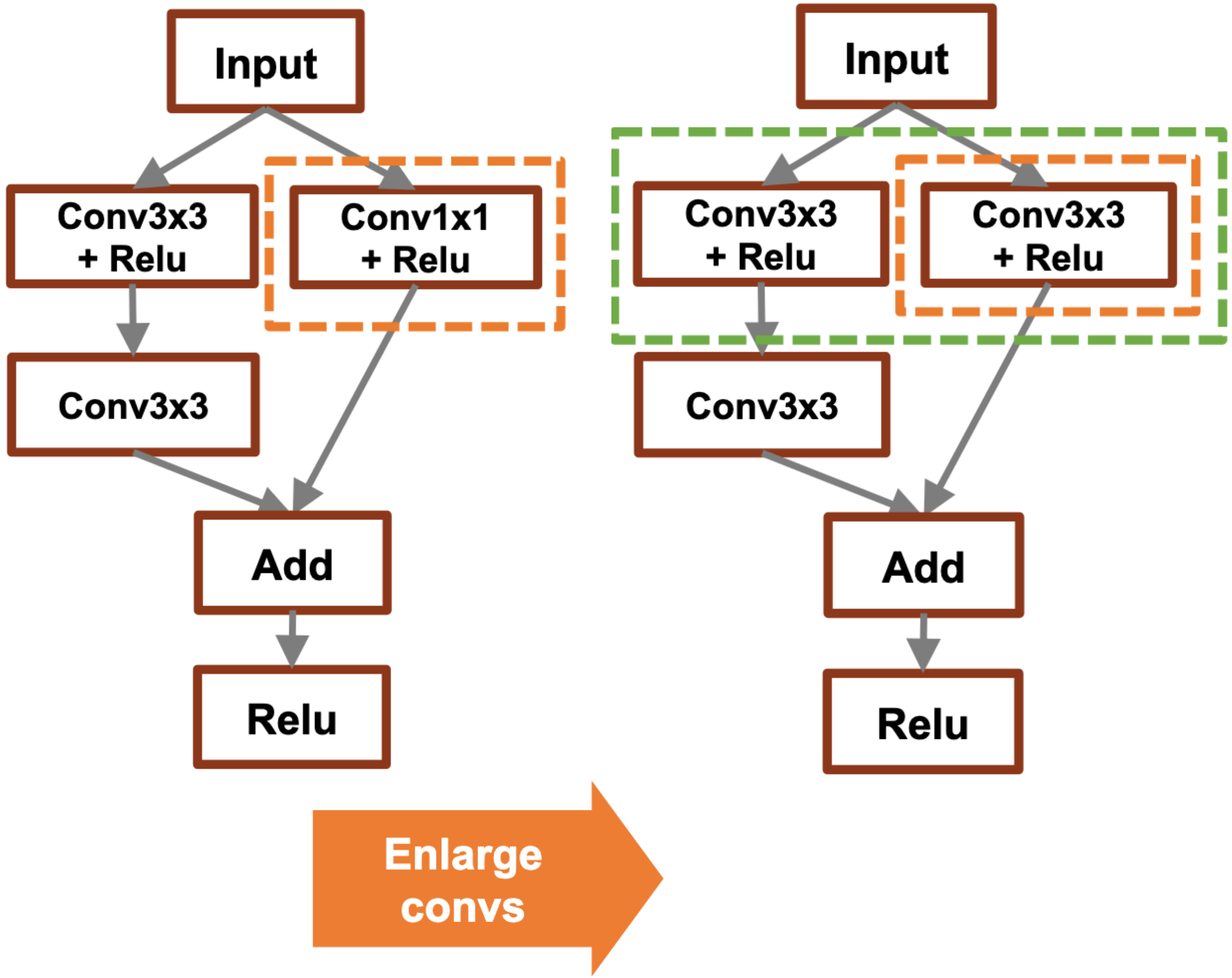
how?

Bells and whistles: graph optimization



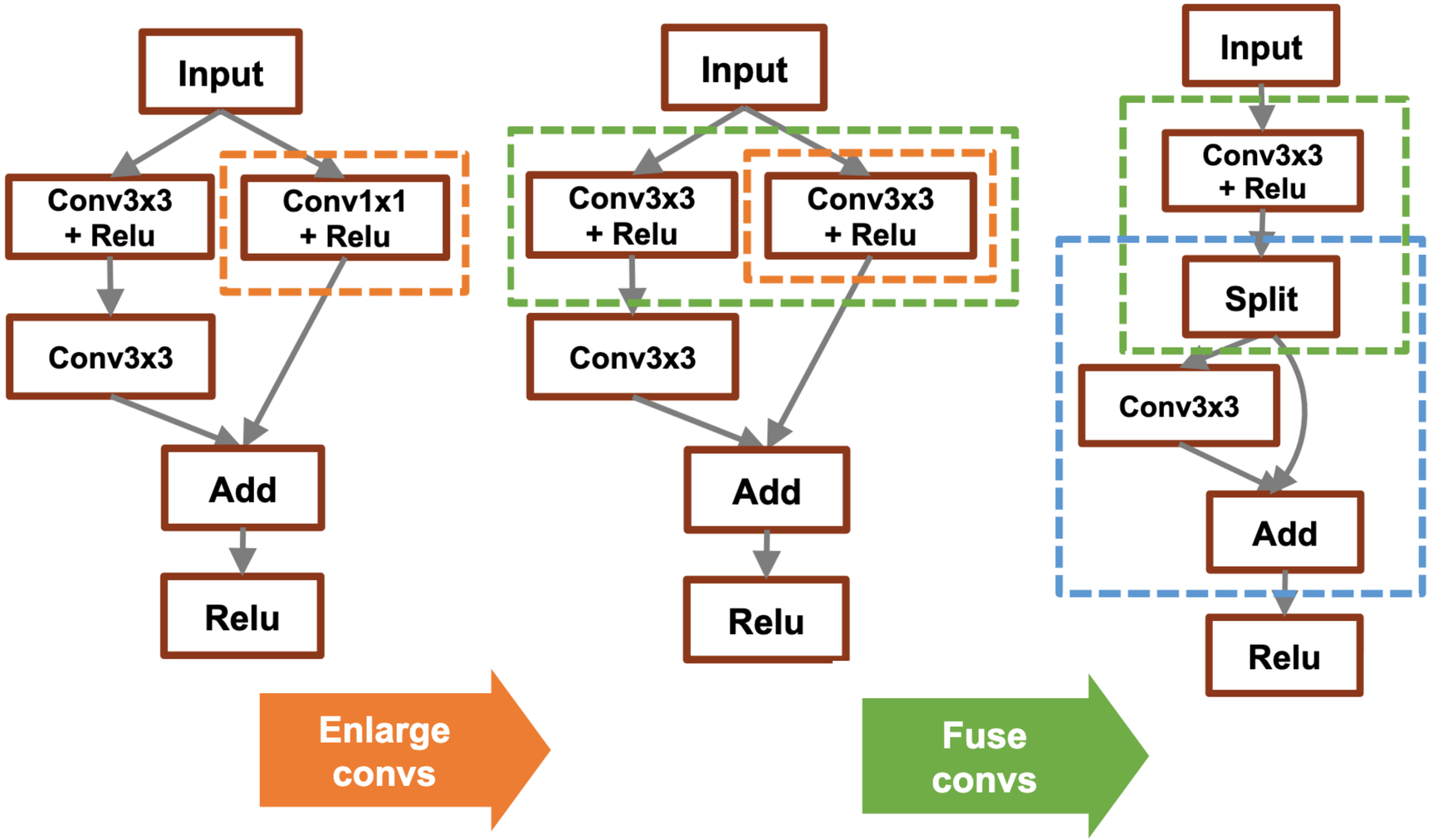
from https://sampl.cs.washington.edu/tvmconf/slides/2019/Zhihao-Jia-TASO.pdf

Bells and whistles: graph optimization



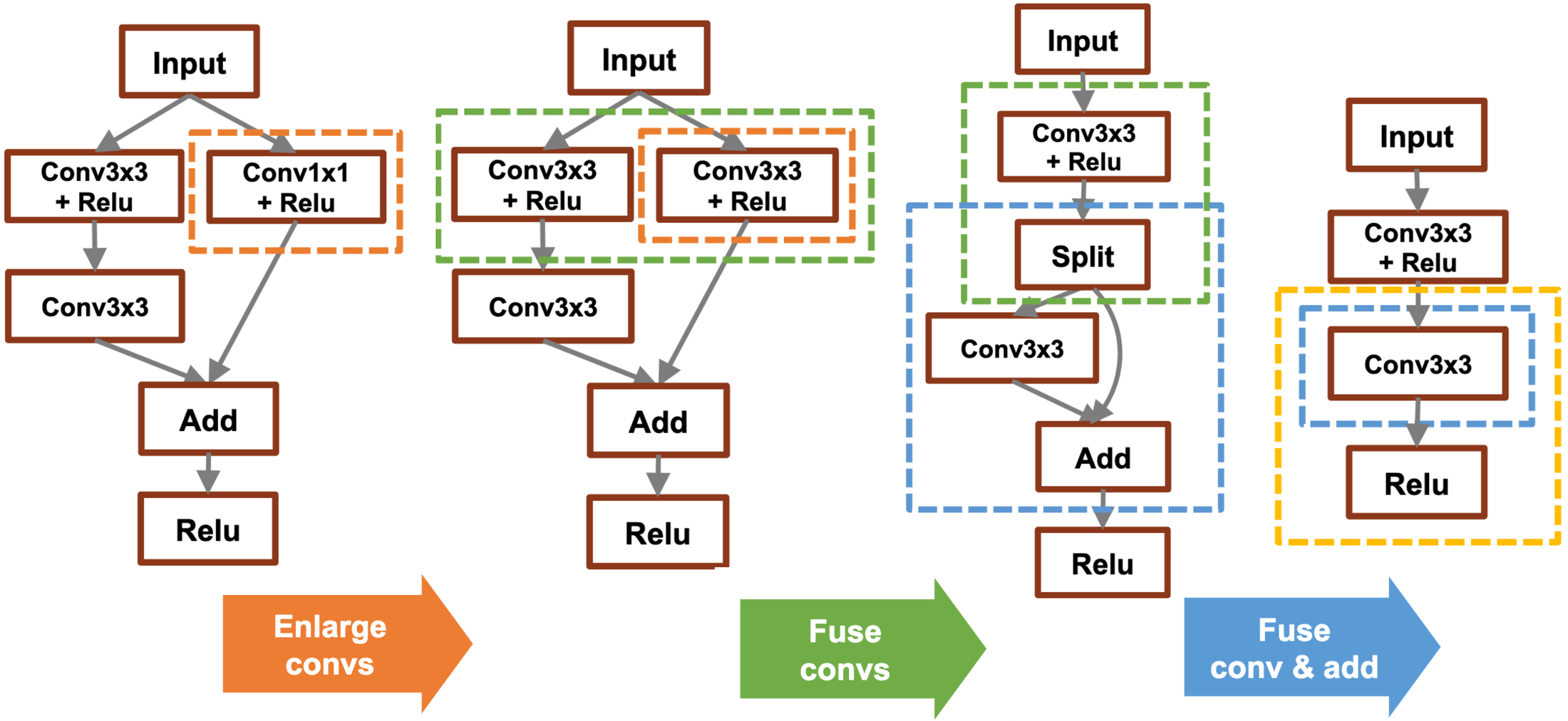
from https://sampl.cs.washington.edu/tvmconf/slides/2019/Zhihao-Jia-TASO.pdf

Bells and whistles: graph optimization



from https://sampl.cs.washington.edu/tvmconf/slides/2019/Zhihao-Jia-TASO.pdf

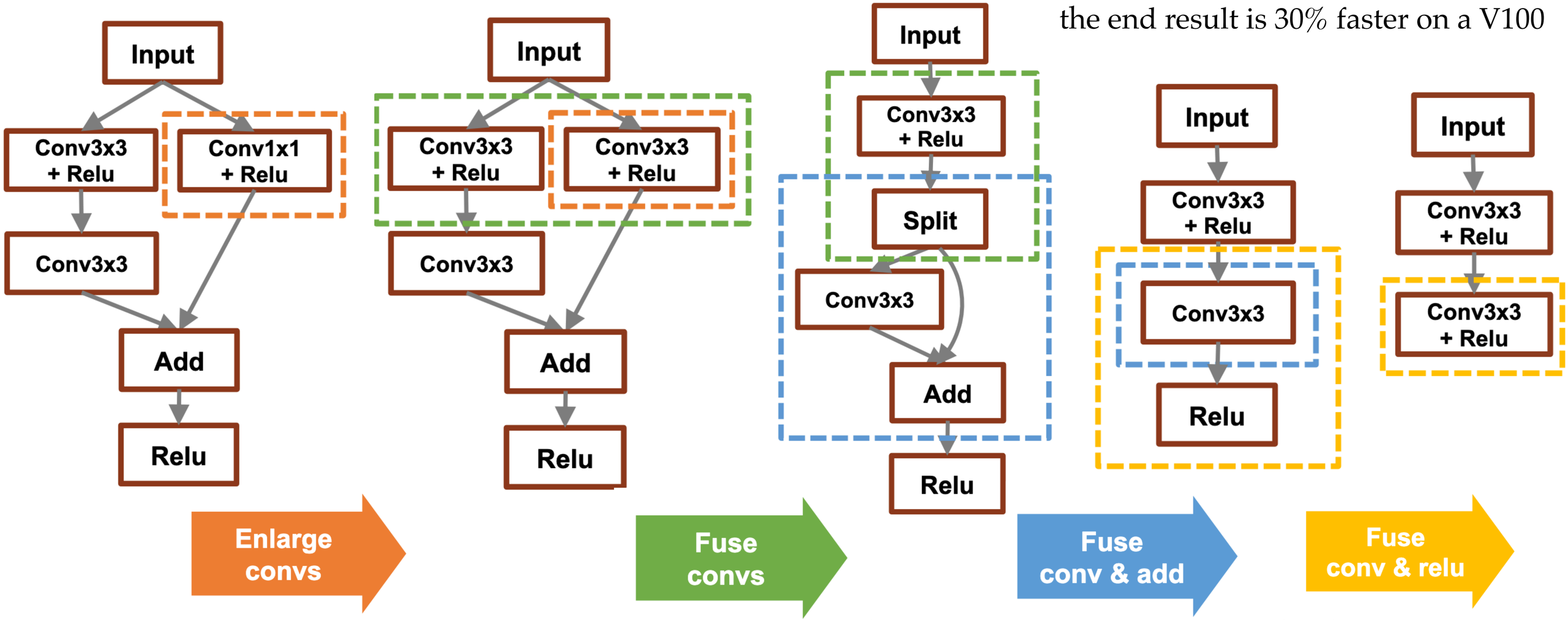
Bells and whistles: graph optimization



from https://sampl.cs.washington.edu/tvmconf/slides/2019/Zhihao-Jia-TASO.pdf

Bells and whistles: graph optimization

the end result is 30% faster on a V100



from https://sampl.cs.washington.edu/tvmconf/slides/2019/Zhihao-Jia-TASO.pdf

Bells and whistles: graph optimization

first, prepare a set of “basic” algebraic rules

Operator Property	Comment
$\forall x, y, z.\ \texttt{ewadd}(x, \texttt{ewadd}(y, z)) = \texttt{ewadd}(\texttt{ewadd}(x, y), z)$	ewadd is associative
$\forall x, y.\ \texttt{ewadd}(x, y) = \texttt{ewadd}(y, x)$	ewadd is commutative
$\forall x, y, z.\ \texttt{ewmul}(x, \texttt{ewmul}(y, z)) = \texttt{ewmul}(\texttt{ewmul}(x, y), z)$	ewmul is associative
$\forall x, y.\ \texttt{ewmul}(x, y) = \texttt{ewmul}(y, x)$	ewmul is commutative
$\forall x, y, z.\ \texttt{ewmul}(\texttt{ewadd}(x, y), z) = \texttt{ewadd}(\texttt{ewmul}(x, z), \texttt{ewmul}(y, z))$	distributivity
$\forall x, y, w.\ \texttt{smul}(\texttt{smul}(x, y), w) = \texttt{smul}(x, \texttt{smul}(y, w))$	smul is associative
$\forall x, y, w.\ \texttt{smul}(\texttt{ewadd}(x, y), w) = \texttt{ewadd}(\texttt{smul}(x, w), \texttt{smul}(y, w))$	distributivity
$\forall x, y, w.\ \texttt{smul}(\texttt{ewmul}(x, y), w) = \texttt{ewmul}(x, \texttt{smul}(y, w))$	operator commutativity
$\forall x.\ \texttt{transpose}(\texttt{transpose}(x)) = x$	transpose is its own inverse
$\forall x, y.\ \texttt{transpose}(\texttt{ewadd}(x, y)) = \texttt{ewadd}(\texttt{transpose}(x), \texttt{transpose}(y))$	operator commutativity
$\forall x, y.\ \texttt{transpose}(\texttt{ewmul}(x, y)) = \texttt{ewmul}(\texttt{transpose}(x), \texttt{transpose}(y))$	operator commutativity
$\forall x, w.\ \texttt{smul}(\texttt{transpose}(x), w) = \texttt{transpose}(\texttt{smul}(x, w))$	operator commutativity
$\forall x, y, z.\ \texttt{matmul}(x, \texttt{matmul}(y, z)) = \texttt{matmul}(\texttt{matmul}(x, y), z)$	matmul is associative
$\forall x, y, w.\ \texttt{smul}(\texttt{matmul}(x, y), w) = \texttt{matmul}(x, \texttt{smul}(y, w))$	matmul is linear
$\forall x, y, z.\ \texttt{matmul}(x, \texttt{ewadd}(y, z)) = \texttt{ewadd}(\texttt{matmul}(x, y), \texttt{matmul}(x, z))$	matmul is linear
$\forall x, y.\ \texttt{transpose}(\texttt{matmul}(x, y)) = \texttt{matmul}(\texttt{transpose}(y), \texttt{transpose}(x))$	matmul and transpose
$\forall s, p, c, x, y, w.\ \texttt{conv}(s, p, c, \texttt{smul}(x, w), y) = \texttt{conv}(s, p, c, x, \texttt{smul}(y, w))$	conv is bilinear
$\forall s, p, x, y, w.\ \texttt{smul}(\texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, x, y), w) = \texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, \texttt{smul}(x, w), y)$	conv is bilinear
$\forall s, p, x, y, z.\ \texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, x, \texttt{ewadd}(y, z)) = \texttt{ewadd}(\texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, x, y), \texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, x, z))$	conv is bilinear
$\forall s, p, x, y, z.\ \texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, \texttt{ewadd}(x, y), z) = \texttt{ewadd}(\texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, x, z), \texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, y, z))$	conv is bilinear
$\forall s, c, k, x, y.\ \texttt{conv}(s, \texttt{P}_{\texttt{same}}, c, x, y) = \texttt{conv}(s, \texttt{P}_{\texttt{same}}, c, x, \texttt{enlarge}(k, y)),$	enlarge convolution kernel
$\forall s, p, x, y.\ \texttt{conv}(s, p, \texttt{A}_{\texttt{relu}}, x, y) = \texttt{relu}(\texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, x, y))$	conv with $\texttt{A}_{\texttt{relu}}$ applies relu
$\forall x.\ \texttt{relu}(\texttt{transpose}(x)) = \texttt{transpose}(\texttt{relu}(x))$	operator commutativity
$\forall s, p, x, k.\ \texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, x, \texttt{C}_{\texttt{pool}}(k)) = \texttt{pool}_{\texttt{avg}}(k, s, p, x)$	pooling by conv. with $\texttt{C}_{\texttt{pool}}$
$\forall k, x.\ \texttt{conv}(1, \texttt{P}_{\texttt{same}}, \texttt{A}_{\texttt{none}}, x, \texttt{I}_{\texttt{conv}}(k)) = x$	identity kernel
$\forall x.\ \texttt{matmul}(x, \texttt{I}_{\texttt{matmul}}) = x$	identity matrix
$\forall x.\ \texttt{ewmul}(x, \texttt{I}_{\texttt{ewmul}}) = x$	ewmul identity
$\forall a, x, y.\ \texttt{split}_0(a, \texttt{concat}(a, x, y)) = x$	split definition
$\forall a, x, y.\ \texttt{split}_1(a, \texttt{concat}(a, x, y)) = y$	split definition
$\forall x, y, z, w.\ \texttt{concat}(0, \texttt{concat}(1, x, y), \texttt{concat}(1, z, w)) = \texttt{concat}(1, \texttt{concat}(0, x, z), \texttt{concat}(0, y, w))$	geometry of concatenation
$\forall a, x, y, w.\ \texttt{concat}(a, \texttt{smul}(x, w), \texttt{smul}(y, w)) = \texttt{smul}(\texttt{concat}(a, x, y), w)$	operator commutativity
$\forall a, x, y, z, w.\ \texttt{concat}(a, \texttt{ewadd}(x, y), \texttt{ewadd}(z, w)) = \texttt{ewadd}(\texttt{concat}(a, x, z), \texttt{concat}(a, y, w))$	operator commutativity
$\forall a, x, y, z, w.\ \texttt{concat}(a, \texttt{ewmul}(x, y), \texttt{ewmul}(z, w)) = \texttt{ewmul}(\texttt{concat}(a, x, z), \texttt{concat}(a, y, w))$	operator commutativity
$\forall a, x, y.\ \texttt{concat}(a, \texttt{relu}(x), \texttt{relu}(y)) = \texttt{relu}(\texttt{concat}(a, x, y))$	operator commutativity
$\forall x, y.\ \texttt{concat}(1, \texttt{transpose}(x), \texttt{transpose}(y)) = \texttt{transpose}(\texttt{concat}(0, x, y))$	concatenation and transpose
$\forall x, y, z.\ \texttt{concat}(1, \texttt{matmul}(x, y), \texttt{matmul}(x, z)) = \texttt{matmul}(x, \texttt{concat}(1, y, z))$	concatenation and matrix mul.
$\forall x, y, z, w.\ \texttt{matmul}(\texttt{concat}(1, x, z), \texttt{concat}(0, y, w)) = \texttt{ewadd}(\texttt{matmul}(x, y), \texttt{matmul}(z, w))$	concatenation and matrix mul.
$\forall s, p, c, x, y, z.\ \texttt{concat}(0, \texttt{conv}(s, p, c, x, z), \texttt{conv}(s, p, c, y, z)) = \texttt{conv}(s, p, c, \texttt{concat}(0, x, y), z)$	concatenation and conv.
$\forall s, p, c, x, y, z.\ \texttt{concat}(1, \texttt{conv}(s, p, c, x, y), \texttt{conv}(s, p, c, x, z)) = \texttt{conv}(s, p, c, x, \texttt{concat}(0, y, z))$	concatenation and conv.
$\forall s, p, x, y, z, w.\ \texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, \texttt{concat}(1, x, z), \texttt{concat}(1, y, w)) = \texttt{ewadd}(\texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, x, y), \texttt{conv}(s, p, \texttt{A}_{\texttt{none}}, z, w))$	concatenation and conv.
$\forall k, s, p, x, y.\ \texttt{concat}(1, \texttt{pool}_{\texttt{avg}}(k, s, p, x), \texttt{pool}_{\texttt{avg}}(k, s, p, y)) = \texttt{pool}_{\texttt{avg}}(k, s, p, \texttt{concat}(1, x, y))$	concatenation and pooling
$\forall k, s, p, x, y.\ \texttt{concat}(0, \texttt{pool}_{\texttt{max}}(k, s, p, x), \texttt{pool}_{\texttt{max}}(k, s, p, y)) = \texttt{pool}_{\texttt{max}}(k, s, p, \texttt{concat}(0, x, y))$	concatenation and pooling
$\forall k, s, p, x, y.\ \texttt{concat}(1, \texttt{pool}_{\texttt{max}}(k, s, p, x), \texttt{pool}_{\texttt{max}}(k, s, p, y)) = \texttt{pool}_{\texttt{max}}(k, s, p, \texttt{concat}(1, x, y))$	concatenation and pooling

TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions

Zhihao Jia
Stanford University
zhihao@cs.stanford.edu

Oded Padon
Stanford University
padon@cs.stanford.edu

James Thomas
Stanford University
jjthomas@stanford.edu

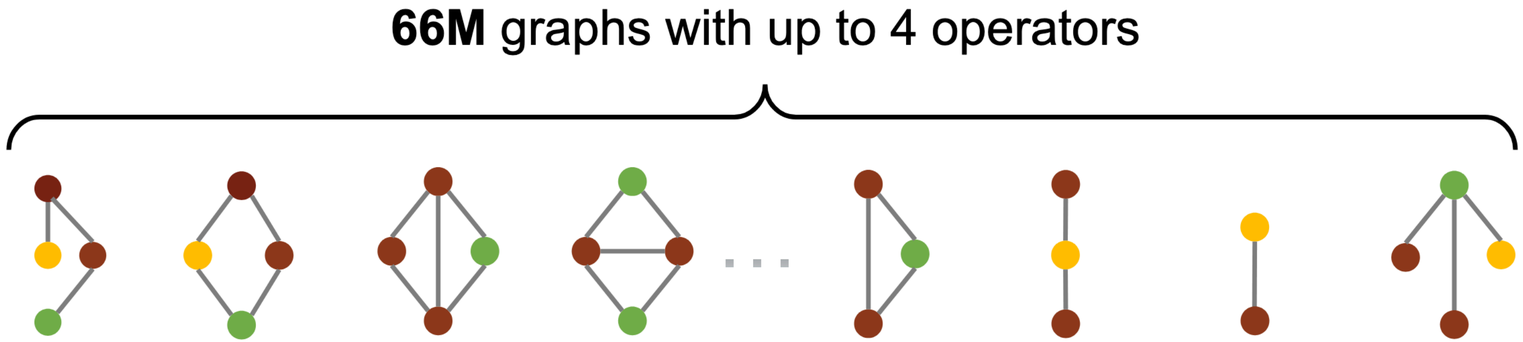
Todd Warszawski
Stanford University
twarszaw@stanford.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

Alex Aiken
Stanford University
aiken@cs.stanford.edu

Bells and whistles: graph optimization

next, enumerate a bunch of "potential" graph rewrites



check whether they are correct using the algebraic rules (using "theorem provers" such as Z3)

TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions

Zhihao Jia
Stanford University
zhihao@cs.stanford.edu

Oded Padon
Stanford University
padon@cs.stanford.edu

James Thomas
Stanford University
jjthomas@stanford.edu

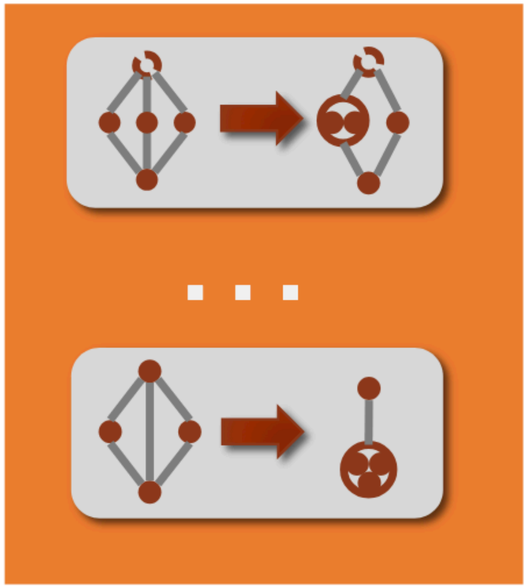
Todd Warszawski
Stanford University
twarszaw@stanford.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

Alex Aiken
Stanford University
aiken@cs.stanford.edu

Bells and whistles: graph optimization

get a list of graph rewrite rules
to rewrite computation into equivalent forms



Verified
Substitutions

TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions

Zhihao Jia
Stanford University
zhihao@cs.stanford.edu

Oded Padon
Stanford University
padon@cs.stanford.edu

James Thomas
Stanford University
jjthomas@stanford.edu

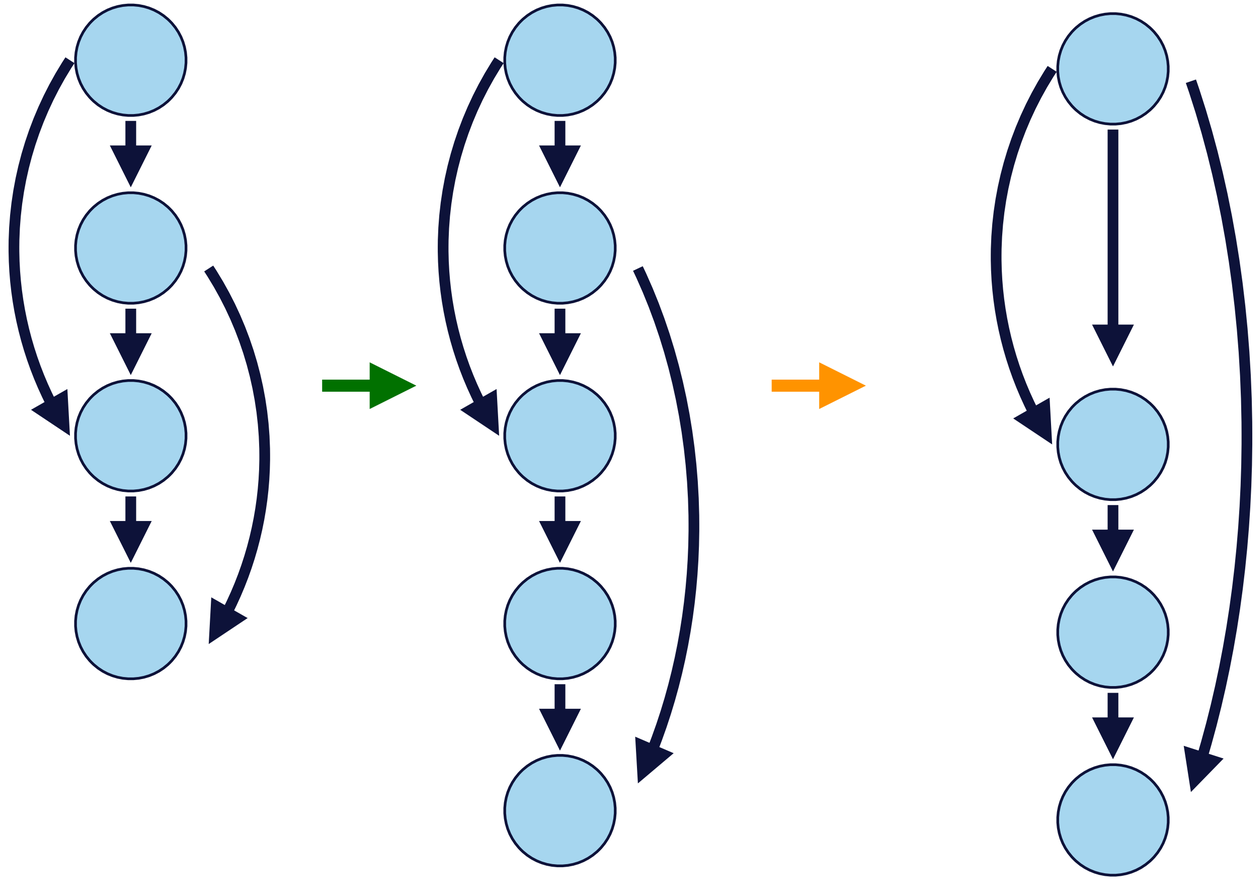
Todd Warszawski
Stanford University
twarszaw@stanford.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

Alex Aiken
Stanford University
aiken@cs.stanford.edu

Bells and whistles: graph optimization

given an input computational graph,
apply a sequence of rewrites to
minimize a “cost model”
(e.g., measure how much time each
operator costs in a hardware)



TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions

Zhihao Jia
Stanford University
zhihao@cs.stanford.edu

Oded Padon
Stanford University
padon@cs.stanford.edu

James Thomas
Stanford University
jjthomas@stanford.edu

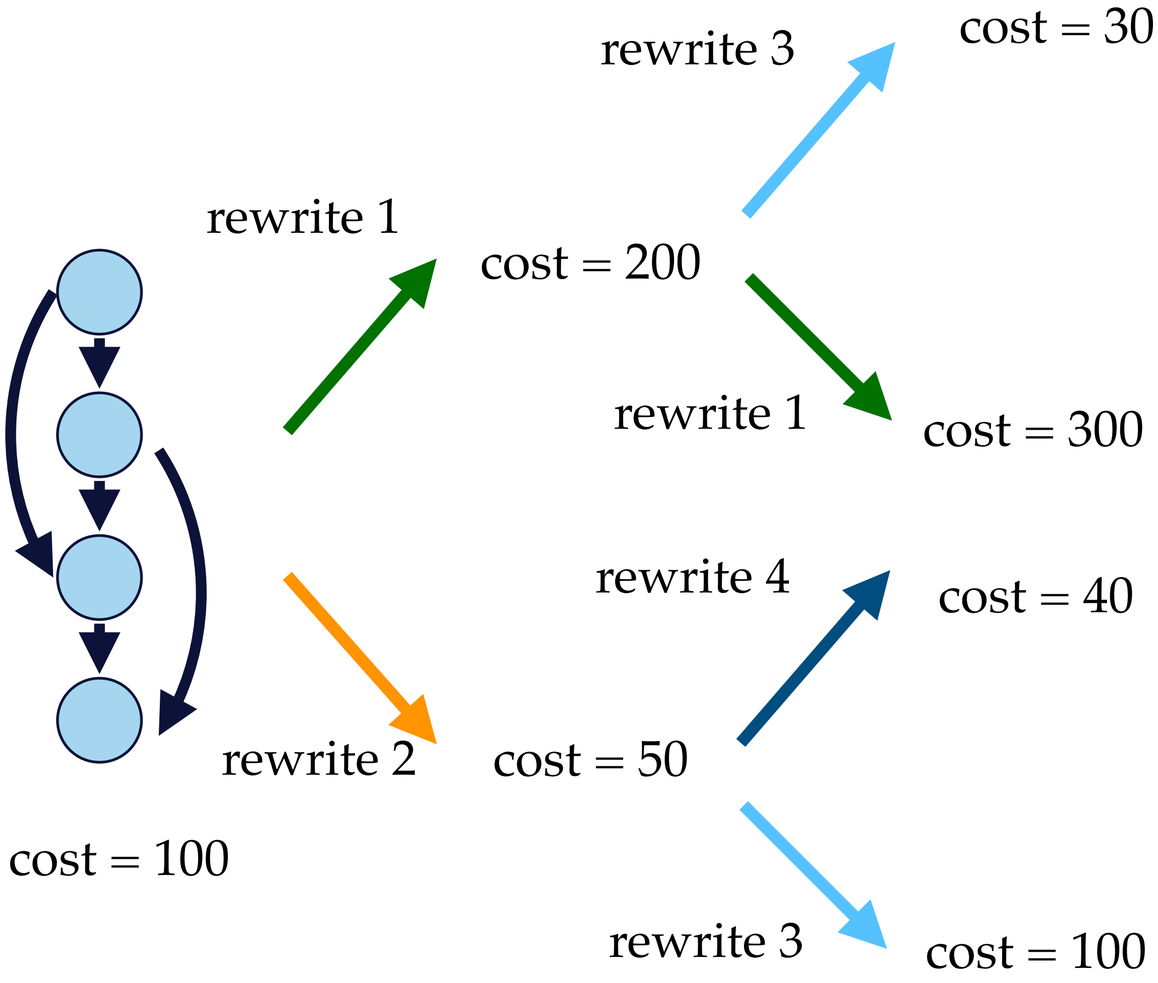
Todd Warszawski
Stanford University
twarszaw@stanford.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

Alex Aiken
Stanford University
aiken@cs.stanford.edu

Bells and whistles: graph optimization

the cost minimization
can be casted as a
discrete search problem
(can be solved by
A*, beam search, MCTS, etc)



TASO: Optimizing Deep Learning Computation with Automatic Generation of Graph Substitutions

Zhihao Jia
Stanford University
zhihao@cs.stanford.edu

Oded Padon
Stanford University
padon@cs.stanford.edu

James Thomas
Stanford University
jjthomas@stanford.edu

Todd Warszawski
Stanford University
twarszaw@stanford.edu

Matei Zaharia
Stanford University
matei@cs.stanford.edu

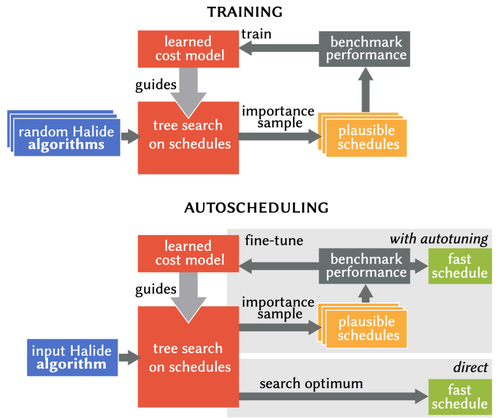
Alex Aiken
Stanford University
aiken@cs.stanford.edu

A few other cool papers in this domain

Learning to Optimize Halide with Tree Search and Random Programs

ANDREW ADAMS, Facebook AI Research
KARIMA MA, UC Berkeley
LUKE ANDERSON, MIT CSAIL
RIYADH BAGHDADI, MIT CSAIL
TZU-MAO LI, MIT CSAIL
MICHAËL GHARBI, Adobe
BENOIT STEINER, Facebook AI Research
STEVEN JOHNSON, Google
KAYVON FATAHALIAN, Stanford University
FRÉDO DURAND, MIT CSAIL
JONATHAN RAGAN-KELLEY, UC Berkeley

We present a new algorithm to automatically schedule Halide programs for high-performance image processing and deep learning. We significantly improve upon the performance of previous methods, which considered a limited subset of schedules. We define a parameterization of possible schedules much larger than prior methods and use a variant of beam search to search over it. The search optimizes runtime predicted by a cost model based on a



Exocompilation for Productive Programming of Hardware Accelerators

Yuka Ikarashi*
MIT CSAIL, USA

Gilbert Louis Bernstein*
UC Berkeley, USA

Alex Reinking
UC Berkeley, USA

Hasan Genc
UC Berkeley, USA

Jonathan Ragan-Kelley
MIT CSAIL, USA

Verified Tensor-Program Optimization Via High-Level Scheduling Rewrites

AMANDA LIU, Massachusetts Institute of Technology, USA
GILBERT LOUIS BERNSTEIN, University of California, Berkeley, USA
ADAM CHLIPALA, Massachusetts Institute of Technology, USA
JONATHAN RAGAN-KELLEY, Massachusetts Institute of Technology, USA

Bells and whistles: Triton

```
@triton.jit
def add_kernel(x_ptr,  # *Pointer* to first input vector.
               y_ptr,  # *Pointer* to second input vector.
               output_ptr,  # *Pointer* to output vector.
               n_elements,  # Size of the vector.
               BLOCK_SIZE: tl.constexpr,  # Number of elements each program should process.
               # NOTE: `constexpr` so it can be used as a shape value.
               ):
    pid = tl.program_id(axis=0)  # We use a 1D launch grid so axis is 0.
    block_start = pid * BLOCK_SIZE
    offsets = block_start + tl.arange(0, BLOCK_SIZE)
    mask = offsets < n_elements
    x = tl.load(x_ptr + offsets, mask=mask)
    y = tl.load(y_ptr + offsets, mask=mask)
    output = x + y
    tl.store(output_ptr + offsets, output, mask=mask)
```

basically CUDA, but operates at “block” instead of “thread” level
this enables much easier warp-level operations
PyTorch 2 generates Triton code from the computational graph

https://triton-lang.org/main/getting-started/tutorials/01-vector-add.html

Jax

- basic idea: construct computational graph where all graph operators are numpy functions
- reuse TensorFlow's graph optimizer and code generator (XLA)
- profit

A unique idea in Jax: vmap

adds extra dimensionality to the inputs of a function

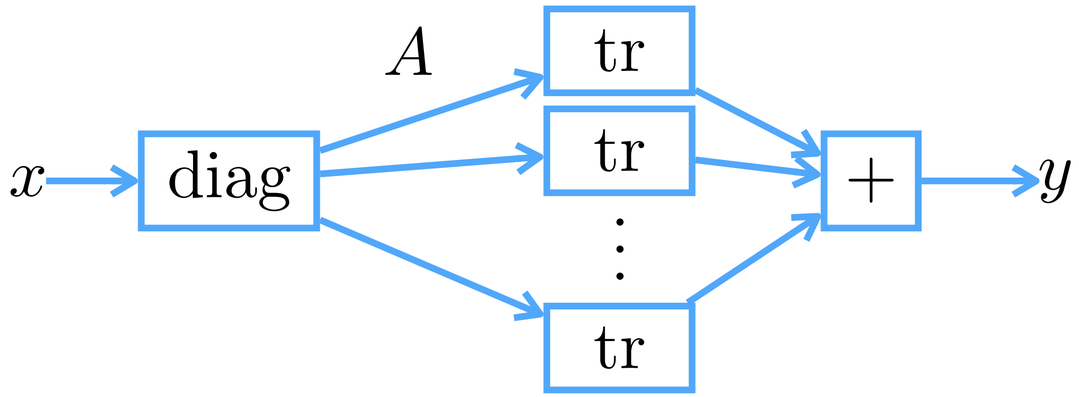
vector-vector dot product

```
vv = lambda x, y: jnp.vdot(x, y) # ([a], [a]) -> []
mv = vmap(vv, (0, None), 0) # ([b,a], [a]) -> [b]
mm = vmap(mv, (None, 1), 1) # ([b,a], [a,c]) -> [b,c]
```

matrix-vector multiplication

matrix-matrix multiplication

A fun trivia:
do TensorFlow / PyTorch obey cheap gradient principle?

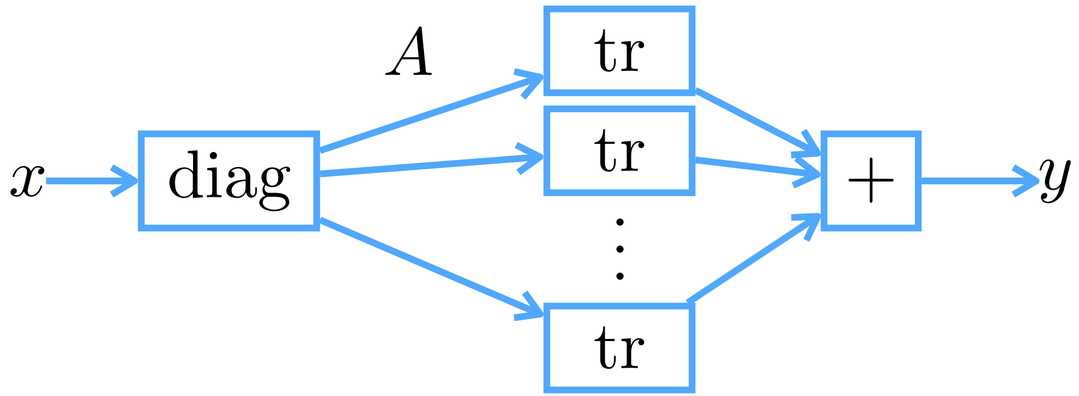


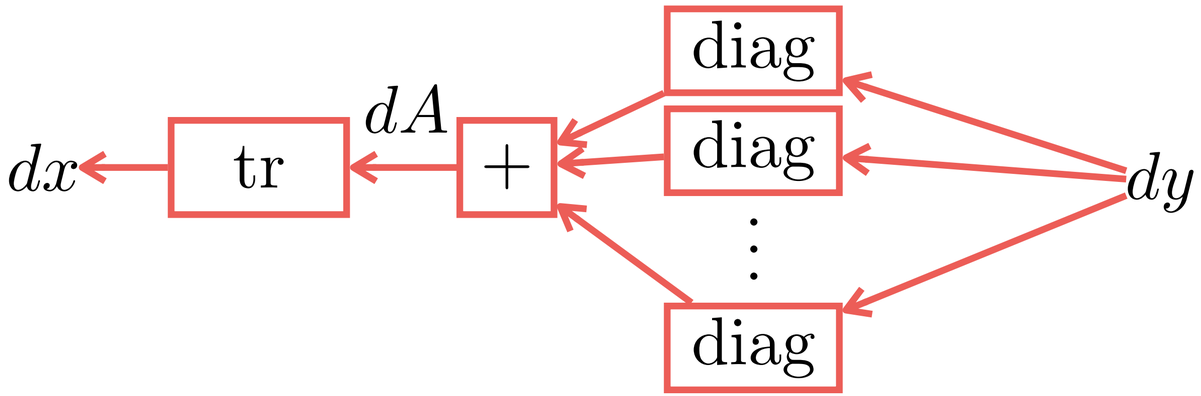
$$\textbf{let}\ \ A : \mathbb{R}^{n\times n} = \mathrm{diag}(x)\ \ \textbf{in}\ \ \underbrace{\mathrm{tr}(A) + \mathrm{tr}(A) + \cdots + \mathrm{tr}(A)}_{k}$$

$$\mathrm{diag}(x) = \begin{bmatrix} x & 0 & 0 & \\ 0 & x & 0 & \cdots \\ 0 & 0 & x & \\ & \vdots & & \ddots \end{bmatrix}$$

$$\mathrm{tr}(A) = \sum_{i=0}^{n} A_{ii}$$

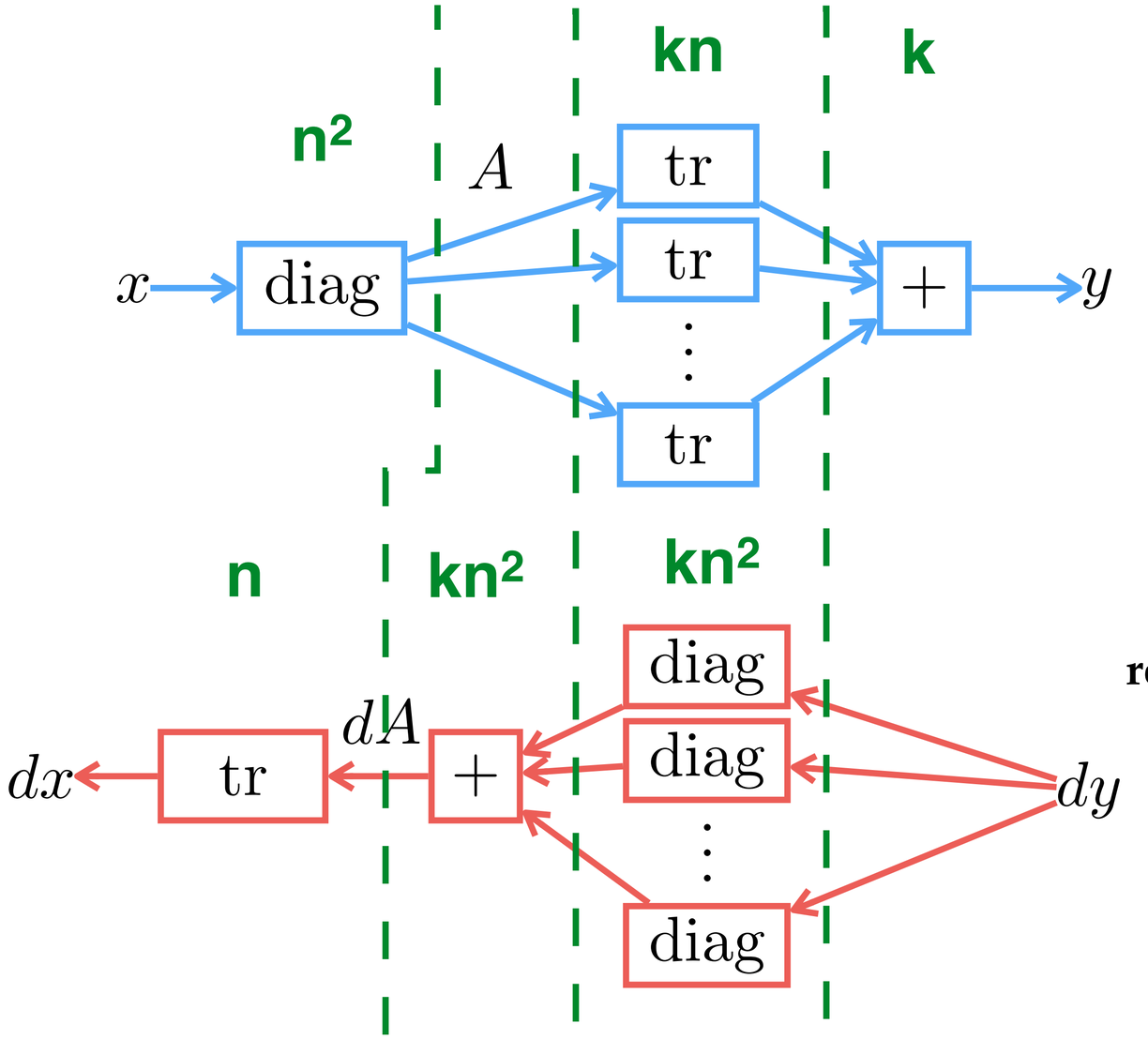
A fun trivia:
do TensorFlow / PyTorch obey cheap gradient principle?





reverse pass

A fun trivia:
do TensorFlow / PyTorch obey cheap gradient principle?

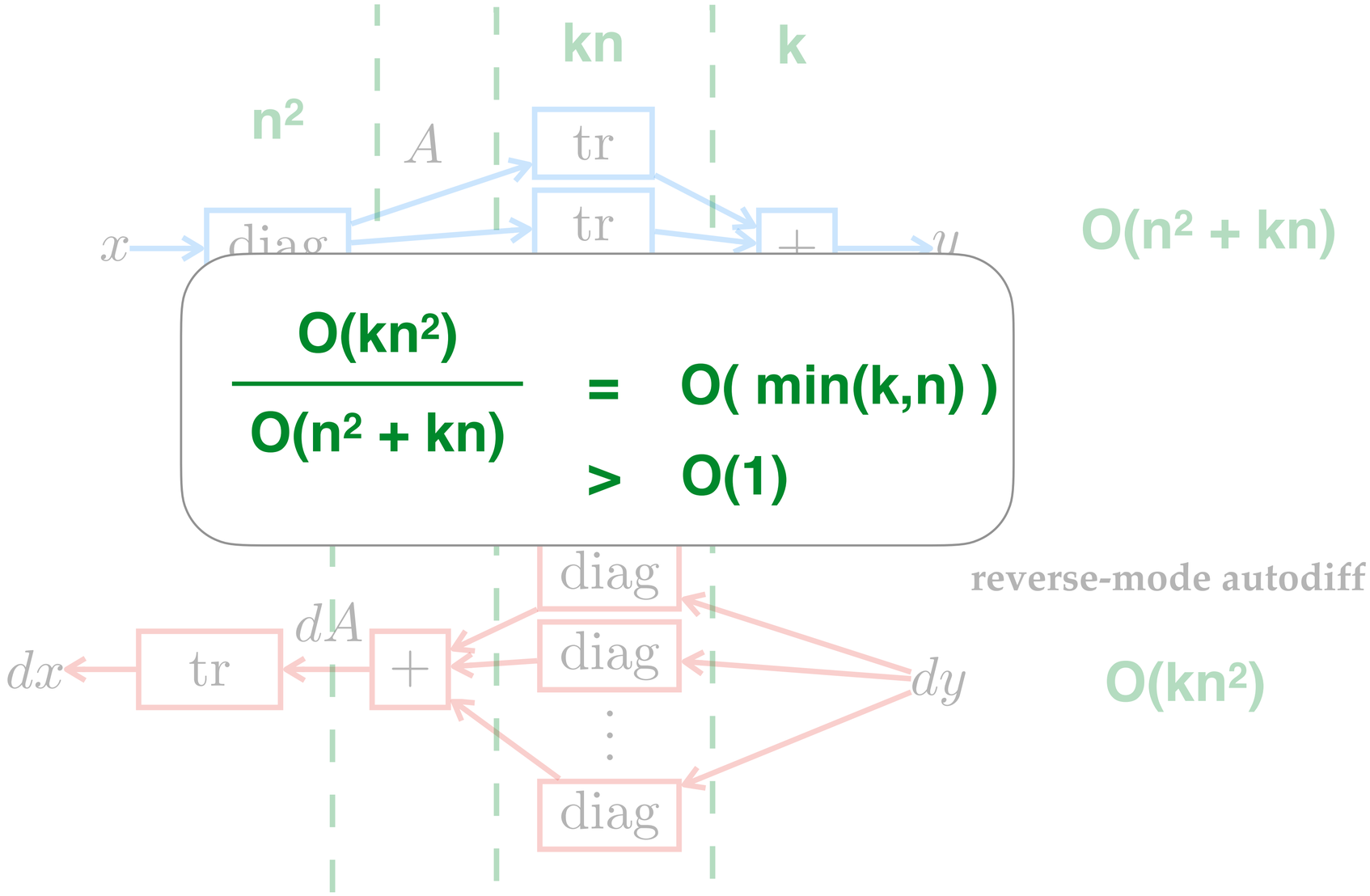


O(n² + kn)

reverse-mode autodiff

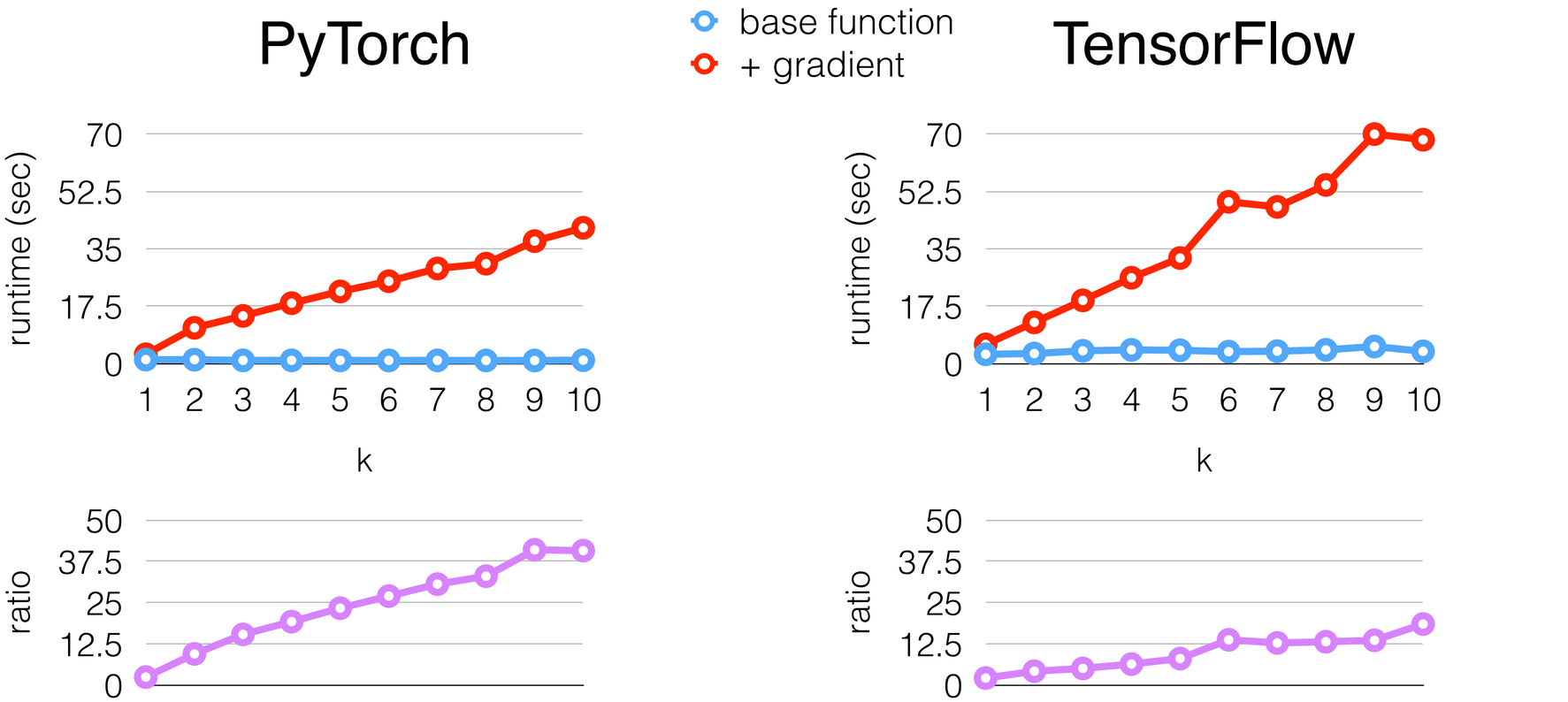
O(kn²)

Tensorflow / PyTorch's reverse mode **violates** cheap gradient principle



reverse-mode autodiff

Empirical verification



$$\textbf{let}\ \ A : \mathbb{R}^{n \times n} = \mathrm{diag}(x)\ \ \textbf{in}\ \ \underbrace{\mathrm{tr}(A) + \mathrm{tr}(A) + \cdots + \mathrm{tr}(A)}_{k}$$

n = 20,000

What's wrong?

- the adjoint of a trace (O(N) operation) requires creating a N^2 matrix
- autodiff needs to be **sparsity-aware**, even when dealing with dense arrays

$$\mathrm{tr}(A) = \sum_{i=0}^{n} A_{ii}$$

$$\mathrm{diag}(x) = \begin{bmatrix} x & 0 & 0 & \\ 0 & x & 0 & \cdots \\ 0 & 0 & x & \\ & \vdots & & \ddots \end{bmatrix}$$

this needs to be sparse

Sparsity-aware, pure functional autodiff

- the adjoint of a trace (O(N) operation) requires creating a N^2 matrix **k** times

- autodiff needs to be **sparsity-aware**, even when dealing with dense arrays

Differentiating A Tensor Language

GILBERT BERNSTEIN, University of California, Berkeley
MICHAEL MARA, Stanford University
TZU-MAO LI, Massachusetts Institute of Technology
DOUGAL MACLAURIN, Google
JONATHAN RAGAN-KELLEY, Massachusetts Institute of Technology