Convolutions and More as Einsum: A Tensor Network Perspective with Advances for Second-Order Methods

Felix Dangel
Vector Institute (Canada)

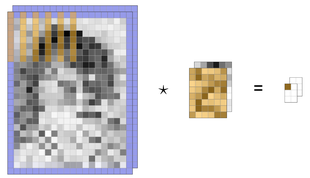
VECTOR INSTITUTE

**We derive einsum implementations of various operations related to convolution.
This simplifies the transfer of algorithmic ideas & speeds up non-traditional routines.**

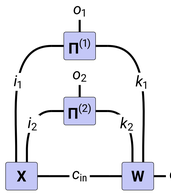
Overview & Motivation: Provide a Simplified Perspective onto Convolutions

Despite their simple intuition, convolutions are more tedious than fully-connected layers.

Convolution



Tensor networks



Evaluation

```
einsum("c_in i1 i2, k1 o1 i1, k2 o2 i2, c_out c_in k1 k2 -> c_out o1 o2", X, Pi1, Pi2, W)
```

Concept	for MLPs	for CNNs
Approximate Hessian diagonal	1989	2023
Kronecker-factored curvature (KFAC, KFRA, KFLR)	2015, 2017, 2017	2016, 2020, 2020
Kronecker-factored quasi-Newton methods (KBFGS)	2021	2022
Neural tangent kernel (NTK)	2018	2019
Hessian rank	2021	2023
Gradient descent learning dynamics	2014	2023

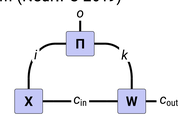
To reduce this complexity gap, we use the einsum formulation of convolution (NeurIPS 2019).

Contribution: Convolutions and More as Einsum

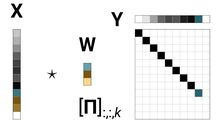
We derive the tensor networks (einsum implementations) for various operations related to convolution (many more in the paper!).

• Starting point: convolution as einsum (NeurIPS 2019)

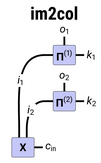
$$Y_{c_{\text{out}},o} = \sum_{c_{\text{in}},k} X_{c_{\text{in}},i(k,o)} W_{c_{\text{out}},c_{\text{in}},k} = \sum_{c_{\text{in}},k} \sum_{i} X_{c_{\text{in}},i} \Pi_{i,o,k} W_{c_{\text{out}},c_{\text{in}},k}$$



Main idea: Make dependencies explicit through index pattern tensor Π.

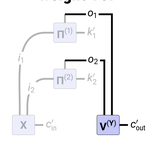


im2col

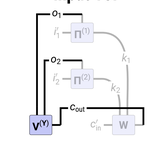


• Convolution
- im2col for arbitrary dimensions
→ **KFAC for ConvNd**
- Easy-to-randomize VJPs
→ **stochastic backpropagation**

weight VJP



input VJP



• Transpose convolution (input VJP)
- Equivalent of im2col (does not exist in APIs)
→ **KFAC for ConvTransposeNd**

Background: Tensor Networks in a Nutshell

Tensor networks are diagrams that represent tensor multiplications and translate to einsum expressions.

Operation	Diagram
Scalar	c
Vector	c — i
Matrix	i — C — j
Tensor	i — C — j, k
Matricize	c — (i,j) ← i, j
Flatten	C — i, j → (i,j)
Trace	C — i

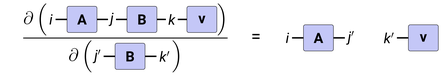
Operation	Diagram
i — AB — k	i — A — j — B — k
i — A ⊙ B — j	i — A, B — j
(i,k) — A ⊗ B — (j,l)	(i,k) — A — j, B — (j,l)
diag(A) — i	A — i
i — diag(a) — i	i — a — i

They are (1) easier to parse than index-heavy equations, . . .

. . . (2) fun to work with through graphical manipulation, . . .

• vmap by adding legs

• Differentiate by removing tensors



. . . and (3) automatically efficient thanks to contraction optimizers (opt_einsum).

Application: Improving an ICML 2024 Second-order Optimizer

Non-traditional convolution-related operations can be more efficient with einsum.

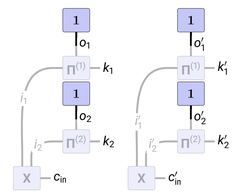
• Example: KFAC-reduce approximation (NeurIPS 2023)

State of the art (standard API)

X
→ ⟦X⟧ (im2col)
→ $1^\top$⟦X⟧ (average)
→ ($1^\top$⟦X⟧)$^\top$ ($1^\top$⟦X⟧)

Time: **9.87 ms**
Extra memory: **3.07 GiB**

Tensor network



Time: **2.69 ms (3.7 x)**
Extra memory: **0 MiB**

(first convolution of ConvNeXt-base with (32, 3, 256, 256) input, on an A40)

• Example: SINGD optimizer (ICML 2024) on CNNs

Net	Optimizer	Per-iteration [s]	Peak memory [GiB]
ResNet18	SGD	$1.17 \cdot 10^{-1}$ (1.00 x)	3.62 (1.00 x)
	SINGD	$1.94 \cdot 10^{-1}$ (1.67 x)	4.54 (1.25 x)
	SINGD+TN	$1.56 \cdot 10^{-1}$ (1.33 x)	3.63 (1.00 x)
VGG19	SGD	$6.90 \cdot 10^{-1}$ (1.00 x)	14.1 (1.00 x)
	SINGD	1.02 (1.48 x)	32.1 (2.28 x)
	SINGD+TN	$8.01 \cdot 10^{-1}$ (1.16 x)	16.1 (1.14 x)

Our einsum implementation reduces the overhead of a second-order method w.r.t. SGD by 50 % in time and up to 100 % in memory.

Try it yourself: `pip install einconv`