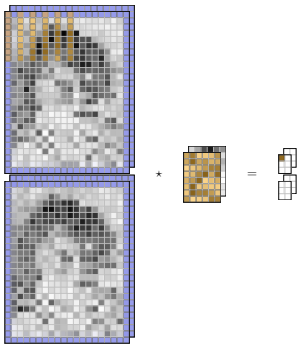
Convolutions and More as Einsum: A Tensor Network Perspective

Felix Dangel

November 29, 2024

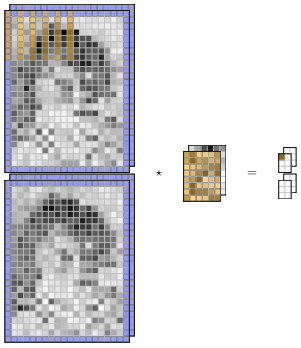
Conceptual Overview

Convolution

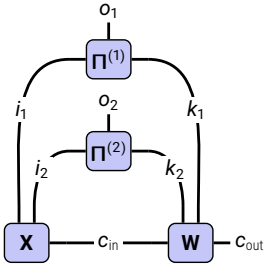


Conceptual Overview

Convolution

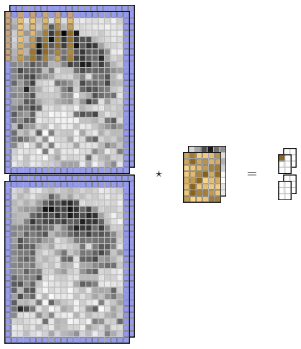


Tensor networks

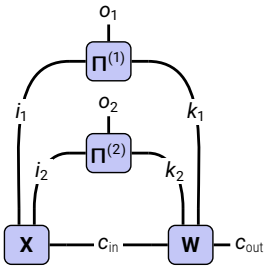


Conceptual Overview

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)
```

Convolutions Are More Tedious than Linear Layers

- Personal example: Vector-Jacobian products (VJPs)

```
def _weight_jac_t_mat_prod(
    self,
    module: Linear,
    g_inp: Tuple[Tensor],
    g_out: Tuple[Tensor],
    mat: Tensor,
    sum_batch: int = True,
    subsampling: List[int] = None,
) -> Tensor:
    """Batch-apply transposed Jacobian of the output w.r.t. the weight.

    Args:
        module: Linear layer.
        g_inp: Gradients w.r.t. module input. Not required by the implementation.
        g_out: Gradients w.r.t. module output. Not required by the implementation.
        mat: Batch of ``V`` vectors of same shape as the layer output
            (``[N, *, out_features]``) to which the transposed output-input Jacobian
            is applied. Has shape ``[V, N, *, out_features]`` if subsampling is not
            used, otherwise ``N`` must be ``len(subsampling)`` instead.
        sum_batch: Sum the result's batch axis. Default: ``True``.
        subsampling: Indices of samples along the output's batch dimension that
            should be considered. Defaults to ``None`` (use all samples).

    Returns:
        Batched transposed Jacobian vector products. Has shape
        ``[V, N, *module.weight.shape]`` when ``sum_batch`` is ``False``. With
        ``sum_batch=True``, has shape ``[V, *module.weight.shape]``. If sub-
        sampling is used, ``N`` must be ``len(subsampling)`` instead.
    """
    d_weight = subsample(module.input0, subsampling=subsampling)

    equation = f"vn...o,n...i->v{'' if sum_batch else 'n'}oi"
    return einsum(equation, mat, d_weight)
```

Convolutions Are More Tedious than Linear Layers

- Personal example: Vector-Jacobian products (VJPs)
- Other algorithmic & theoretical ideas

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

Convolutions Are More Tedious than Linear Layers

- Personal example: Vector-Jacobian products (VJPs)
- Other algorithmic & theoretical ideas

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

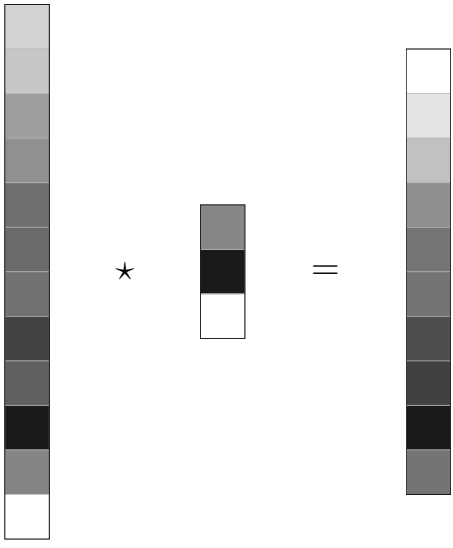
This talk: A simplifying perspective of convolutions through tensor networks

Convolutions — A Visual Walkthrough

A Simple Convolution

$\mathbf{Y} = \mathbf{X} \star \mathbf{W}$ with

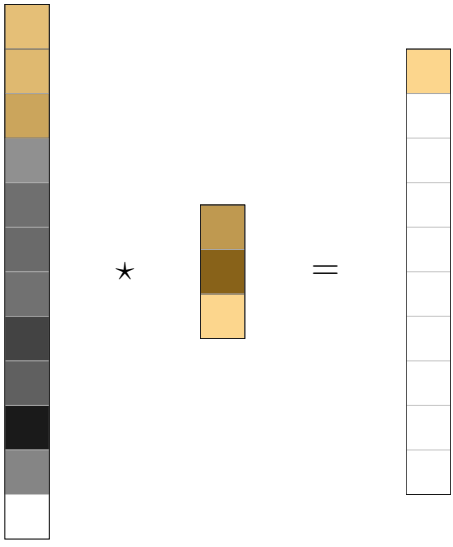
- $\mathbf{X} \in \mathbb{R}^I$
- $\mathbf{W} \in \mathbb{R}^K$
- $\mathbf{Y} \in \mathbb{R}^O$



A Simple Convolution

$\mathbf{Y} = \mathbf{X} \star \mathbf{W}$ with

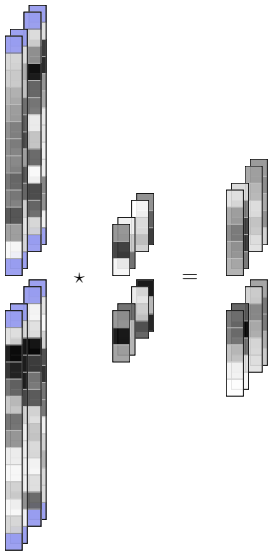
- $\mathbf{X} \in \mathbb{R}^I$
- $\mathbf{W} \in \mathbb{R}^K$
- $\mathbf{Y} \in \mathbb{R}^O$



Putting Everything Together in One Dimension...

$\mathbf{Y} = \mathbf{X} \star \mathbf{W}$ with

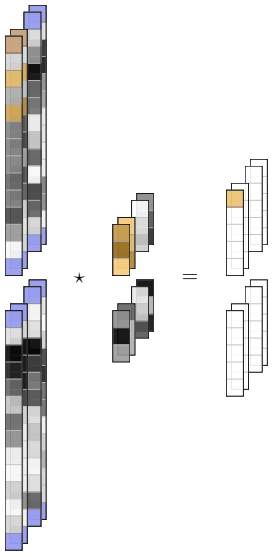
- $\mathbf{X} \in \mathbb{R}^{N \times C_{\text{in}} \times I}$
- $\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}/G \times K}$
- $\mathbf{Y} \in \mathbb{R}^{N \times C_{\text{out}} \times O}$



Putting Everything Together in One Dimension...

$\mathbf{Y} = \mathbf{X} \star \mathbf{W}$ with

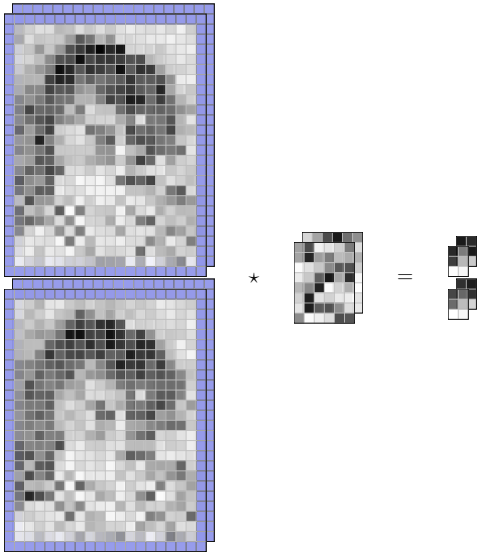
- $\mathbf{X} \in \mathbb{R}^{N \times C_{\text{in}} \times I}$
- $\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}/G \times K}$
- $\mathbf{Y} \in \mathbb{R}^{N \times C_{\text{out}} \times O}$



. . . and in Two Dimensions

$\mathbf{Y} = \mathbf{X} \star \mathbf{W}$ with

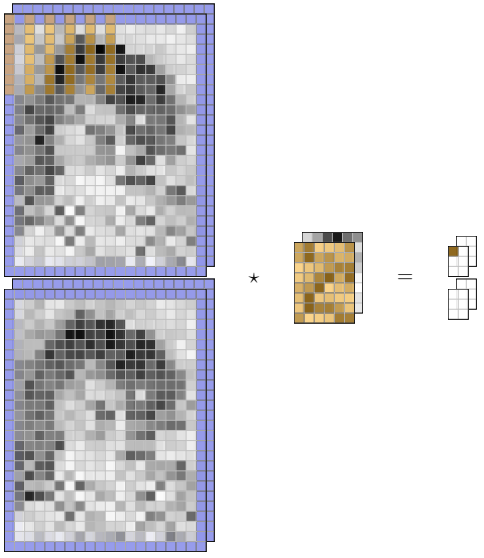
- $\mathbf{X} \in \mathbb{R}^{N \times C_{\text{in}} \times I_1 \times I_2}$
- $\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}/G \times K_1 \times K_2}$
- $\mathbf{Y} \in \mathbb{R}^{N \times C_{\text{out}} \times O_1 \times O_2}$



. . . and in Two Dimensions

$\mathbf{Y} = \mathbf{X} \star \mathbf{W}$ with

- $\mathbf{X} \in \mathbb{R}^{N \times C_{\text{in}} \times I_1 \times I_2}$
- $\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}}/G \times K_1 \times K_2}$
- $\mathbf{Y} \in \mathbb{R}^{N \times C_{\text{out}} \times O_1 \times O_2}$



Tensor Multiplication & Tensor Networks

Tensor Multiplication Unifies Vector & Matrix Multiplications

(Disclaimer: Tensor = multi-dimensional array; sorry)

Tensor Multiplication Unifies Vector & Matrix Multiplications

(Disclaimer: Tensor = multi-dimensional array; sorry)

Consider two vectors $\boldsymbol{a}, \boldsymbol{b}$

Product	**Notation**
Inner	$c = \boldsymbol{a}^\top \boldsymbol{b}$
Element-wise	$\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b}$
Outer/Kronecker	$\boldsymbol{C} = \boldsymbol{a}\boldsymbol{b}^\top = \boldsymbol{a} \otimes \boldsymbol{b}^\top$

Tensor Multiplication Unifies Vector & Matrix Multiplications

(Disclaimer: Tensor = multi-dimensional array; sorry)

Consider two vectors $\boldsymbol{a}, \boldsymbol{b}$

Product	**Notation**	**Index notation**
Inner	$c = \boldsymbol{a}^\top \boldsymbol{b}$	$c = \sum_i a_i b_i$
Element-wise	$\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b}$	$c_i = a_i b_i$
Outer/Kronecker	$\boldsymbol{C} = \boldsymbol{a}\boldsymbol{b}^\top = \boldsymbol{a} \otimes \boldsymbol{b}^\top$	$C_{i,j} = a_i b_j$

Tensor Multiplication Unifies Vector & Matrix Multiplications

(Disclaimer: Tensor = multi-dimensional array; sorry)

Consider two vectors $\boldsymbol{a}, \boldsymbol{b}$ and two matrices $\boldsymbol{A}, \boldsymbol{B}$

Product	**Notation**	**Index notation**
Inner	$c = \boldsymbol{a}^\top \boldsymbol{b}$	$c = \sum_i a_i b_i$
Element-wise	$\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b}$	$c_i = a_i b_i$
Outer/Kronecker	$\boldsymbol{C} = \boldsymbol{a}\boldsymbol{b}^\top = \boldsymbol{a} \otimes \boldsymbol{b}^\top$	$C_{i,j} = a_i b_j$
Inner	$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$	
Element-wise	$\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B}$	
Kronecker	$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B}$	

Tensor Multiplication Unifies Vector & Matrix Multiplications

(Disclaimer: Tensor = multi-dimensional array; sorry)

Consider two vectors $\boldsymbol{a}, \boldsymbol{b}$ and two matrices $\boldsymbol{A}, \boldsymbol{B}$

Product	**Notation**	**Index notation**
Inner	$c = \boldsymbol{a}^\top \boldsymbol{b}$	$c = \sum_i a_i b_i$
Element-wise	$\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b}$	$c_i = a_i b_i$
Outer/Kronecker	$\boldsymbol{C} = \boldsymbol{a}\boldsymbol{b}^\top = \boldsymbol{a} \otimes \boldsymbol{b}^\top$	$C_{i,j} = a_i b_j$
Inner	$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$	$C_{i,k} = \sum_j A_{i,j} B_{j,k}$
Element-wise	$\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B}$	$C_{i,j} = A_{i,j} B_{i,j}$
Kronecker	$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B}$	$C_{(i,k),(j,l)} = A_{i,j} B_{k,l}$

Tensor Multiplication Unifies Vector & Matrix Multiplications

(Disclaimer: Tensor = multi-dimensional array; sorry)

Consider two vectors $\boldsymbol{a}, \boldsymbol{b}$ and two matrices $\boldsymbol{A}, \boldsymbol{B}$

Product	Notation	Index notation	In	Out
Inner	$c = \boldsymbol{a}^\top \boldsymbol{b}$	$c = \sum_i a_i b_i$	$(i), (i)$	$()$
Element-wise	$\boldsymbol{c} = \boldsymbol{a} \odot \boldsymbol{b}$	$c_i = a_i b_i$	$(i), (i)$	(i)
Outer/Kronecker	$\boldsymbol{C} = \boldsymbol{a}\boldsymbol{b}^\top = \boldsymbol{a} \otimes \boldsymbol{b}^\top$	$C_{i,j} = a_i b_j$	$(i), (j)$	(i,j)
Inner	$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$	$C_{i,k} = \sum_j A_{i,j} B_{j,k}$	$(i,j), (j,k)$	(i,k)
Element-wise	$\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B}$	$C_{i,j} = A_{i,j} B_{i,j}$	$(i,j), (i,j)$	(i,j)
Kronecker	$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B}$	$C_{(i,k),(j,l)} = A_{i,j} B_{k,l}$	$(i,j), (k,l)$	$((i,k),(j,l))$

We can infer the summations given the index signature

A Mathematical Definition of Tensor Multiplication [e.g. Laue et al., 2020]

Given two tensors $\mathbf{A}, \mathbf{B}$ with index tuples $S_\mathbf{A}, S_\mathbf{B}$

$$C := *_{(S_\mathbf{A}, S_\mathbf{B}, S_\mathbf{C})}(\mathbf{A}, \mathbf{B}) \quad \Leftrightarrow \quad [\mathbf{C}]_{S_\mathbf{C}} = \sum_{(S_\mathbf{A} \cup S_\mathbf{B}) \setminus S_\mathbf{C}} [\mathbf{A}]_{S_\mathbf{A}} [\mathbf{B}]_{S_\mathbf{B}} \,,$$

indices not present in the output are summed out; $S_\mathbf{C}$ satisfies $S_\mathbf{C} \subseteq S_\mathbf{A} \cup S_\mathbf{B}$.

A Mathematical Definition of Tensor Multiplication [e.g. Laue et al., 2020]

Given two tensors $\mathbf{A}, \mathbf{B}$ with index tuples $S_\mathbf{A}, S_\mathbf{B}$

$$C := *_{(S_\mathbf{A}, S_\mathbf{B}, S_\mathbf{C})}(\mathbf{A}, \mathbf{B}) \quad \Leftrightarrow \quad [\mathbf{C}]_{S_\mathbf{C}} = \sum_{(S_\mathbf{A} \cup S_\mathbf{B}) \setminus S_\mathbf{C}} [\mathbf{A}]_{S_\mathbf{A}} [\mathbf{B}]_{S_\mathbf{B}} \,,$$

indices not present in the output are summed out; $S_\mathbf{C}$ satisfies $S_\mathbf{C} \subseteq S_\mathbf{A} \cup S_\mathbf{B}$.

[Example] Matrix-matrix multiplication $\boldsymbol{C = AB}$ as tensor multiplication

$$\begin{aligned} S_A &= (i, j) \\ S_B &= (j, k) \\ S_C &= (i, k) \end{aligned}$$

A Mathematical Definition of Tensor Multiplication [e.g. Laue et al., 2020]

Given two tensors $\mathbf{A}, \mathbf{B}$ with index tuples $S_\mathbf{A}, S_\mathbf{B}$

$$C := *_{(S_\mathbf{A}, S_\mathbf{B}, S_\mathbf{C})}(\mathbf{A}, \mathbf{B}) \quad \Leftrightarrow \quad [\mathbf{C}]_{S_\mathbf{C}} = \sum_{(S_\mathbf{A} \cup S_\mathbf{B}) \setminus S_\mathbf{C}} [\mathbf{A}]_{S_\mathbf{A}} [\mathbf{B}]_{S_\mathbf{B}} \,,$$

indices not present in the output are summed out; $S_\mathbf{C}$ satisfies $S_\mathbf{C} \subseteq S_\mathbf{A} \cup S_\mathbf{B}$.

[Example] Matrix-matrix multiplication $\boldsymbol{C = AB}$ as tensor multiplication

$$\begin{array}{rcl} S_\boldsymbol{A} & = & (i,j) \\ S_\boldsymbol{B} & = & (j,k) \\ S_\boldsymbol{C} & = & (i,k) \end{array} \quad \Longrightarrow \quad (S_\boldsymbol{A} \cup S_\boldsymbol{B}) \setminus S_\boldsymbol{C} = (i,j,k) \setminus (i,k) = (j)$$

A Mathematical Definition of Tensor Multiplication [e.g. Laue et al., 2020]

Given two tensors $\mathbf{A}, \mathbf{B}$ with index tuples $S_\mathbf{A}, S_\mathbf{B}$

$$C := *_{(S_\mathbf{A}, S_\mathbf{B}, S_\mathbf{C})}(\mathbf{A}, \mathbf{B}) \quad \Leftrightarrow \quad [\mathbf{C}]_{S_\mathbf{C}} = \sum_{(S_\mathbf{A} \cup S_\mathbf{B}) \setminus S_\mathbf{C}} [\mathbf{A}]_{S_\mathbf{A}} [\mathbf{B}]_{S_\mathbf{B}} \,,$$

indices not present in the output are summed out; $S_\mathbf{C}$ satisfies $S_\mathbf{C} \subseteq S_\mathbf{A} \cup S_\mathbf{B}$.

[Example] Matrix-matrix multiplication $\boldsymbol{C = AB}$ as tensor multiplication

$$\begin{aligned} S_A &= (i,j) \\ S_B &= (j,k) \\ S_C &= (i,k) \end{aligned} \quad \Longrightarrow \quad (S_A \cup S_B) \setminus S_C = (i,j,k) \setminus (i,k) = (j)$$

$$\Longrightarrow \boldsymbol{C} = *_{((i,j),(j,k),(i,k))}(\boldsymbol{A}, \boldsymbol{B}), \qquad \text{or} \qquad \texttt{C = einsum("ij,jk->ik", A, B).}$$

Many libraries provide $*_{(\ldots)}(\ldots)$ as `einsum`.

A Mathematical Definition of Tensor Multiplication [e.g. Laue et al., 2020]

General case: Given N tensors $\mathbf{A}_1, \ldots, \mathbf{A}_N$ with index tuples $S_{\mathbf{A}_1}, \ldots, S_{\mathbf{A}_N}$

$$C := *_{(S_{\mathbf{A}_1}, \ldots, S_{\mathbf{A}_N}, S_{\mathbf{C}})}(\mathbf{A}_1, \ldots, \mathbf{A}_N) \quad \Leftrightarrow \quad [\mathbf{C}]_{S_{\mathbf{C}}} = \sum_{(S_{\mathbf{A}_1} \cup \cdots \cup S_{\mathbf{A}_N}) \setminus S_{\mathbf{C}}} [\mathbf{A}_1]_{S_{\mathbf{A}_1}} \cdots [\mathbf{A}_N]_{S_{\mathbf{A}_N}},$$

indices not present in the output are summed out; $S_{\mathbf{C}}$ satisfies $S_{\mathbf{C}} \subseteq S_{\mathbf{A}_1} \cup \cdots \cup S_{\mathbf{A}_N}$.

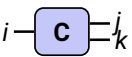
[Example] Matrix-matrix multiplication $\boldsymbol{C = AB}$ as tensor multiplication

$$\begin{array}{rcl} S_{\boldsymbol{A}} & = & (i,j) \\ S_{\boldsymbol{B}} & = & (j,k) \\ S_{\boldsymbol{C}} & = & (i,k) \end{array} \quad \Longrightarrow \quad (S_{\boldsymbol{A}} \cup S_{\boldsymbol{B}}) \setminus S_{\boldsymbol{C}} = (i,j,k) \setminus (i,k) = (j)$$

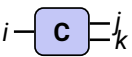
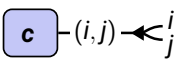
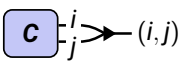
$$\Longrightarrow \boldsymbol{C} = *_{((i,j),(j,k),(i,k))}(\boldsymbol{A}, \boldsymbol{B}), \qquad \text{or} \qquad \texttt{C = einsum("ij,jk->ik", A, B).}$$

Many libraries provide $*_{(\ldots)}(\ldots)$ as `einsum`.

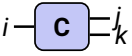
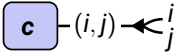
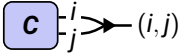
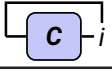
Tensor Networks: Graphical Notation for Tensor Multiplications

Operation	Diagram
Scalar	
Vector	
Matrix	
Tensor	

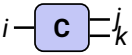
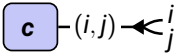
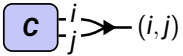
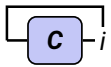
Tensor Networks: Graphical Notation for Tensor Multiplications

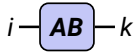
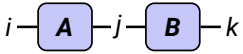
Operation	Diagram
Scalar	
Vector	
Matrix	
Tensor	
Matricize	
Flatten	

Tensor Networks: Graphical Notation for Tensor Multiplications

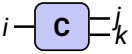
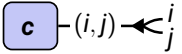
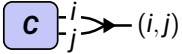
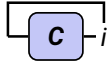
Operation	Diagram
Scalar	
Vector	
Matrix	
Tensor	
Matricize	
Flatten	
Trace	

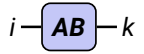
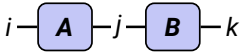
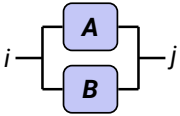
Tensor Networks: Graphical Notation for Tensor Multiplications

Operation	Diagram
Scalar	
Vector	
Matrix	
Tensor	
Matricize	
Flatten	
Trace	

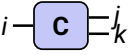
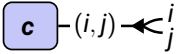
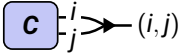
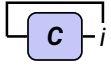
Operation	Diagram
	

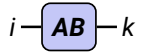
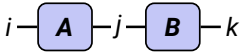
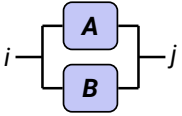
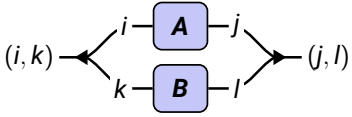
Tensor Networks: Graphical Notation for Tensor Multiplications

Operation	Diagram
Scalar	
Vector	
Matrix	
Tensor	
Matricize	
Flatten	
Trace	

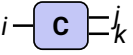
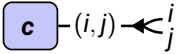
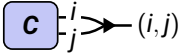
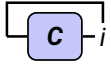
Operation	Diagram
	
	

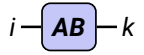
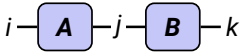
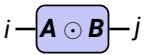
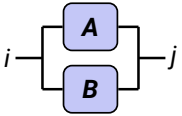
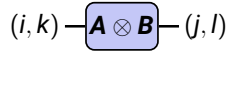
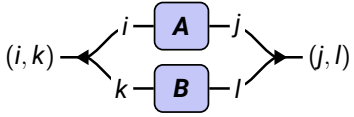
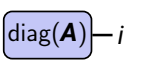
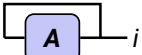
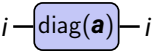
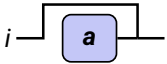
Tensor Networks: Graphical Notation for Tensor Multiplications

Operation	Diagram
Scalar	
Vector	
Matrix	
Tensor	
Matricize	
Flatten	
Trace	

Operation	Diagram
	
	
	

Tensor Networks: Graphical Notation for Tensor Multiplications

Operation	Diagram
Scalar	
Vector	
Matrix	
Tensor	
Matricize	
Flatten	
Trace	

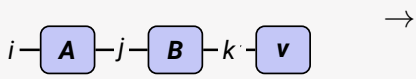
Operation	Diagram
	
	
	
	
	

Benefits of TN/`einsum` Abstraction

Function Transformations with Tensor Networks

[Example] Batching/`vmap`-ing: adding legs

$$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{v}) \mapsto \boldsymbol{ABv}$$

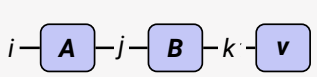


$\rightarrow$

Function Transformations with Tensor Networks

[Example] Batching/`vmap`-ing: adding legs

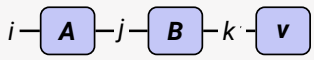
$$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{v}) \mapsto \boldsymbol{A}\boldsymbol{B}\boldsymbol{v} \qquad \{(\boldsymbol{A}_n, \boldsymbol{B}_n, \boldsymbol{v}_n)\}_{n=1}^{N} \mapsto \{\boldsymbol{A}_n \boldsymbol{B}_n \boldsymbol{v}_n\}_{n=1}^{N}$$



Function Transformations with Tensor Networks

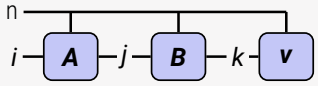
[Example] Batching/vmap-ing: adding legs

$$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{v}) \mapsto \boldsymbol{ABv}$$



$\rightarrow$

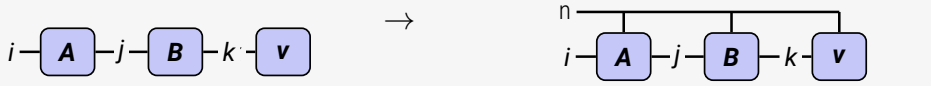
$$\{(\boldsymbol{A}_n, \boldsymbol{B}_n, \boldsymbol{v}_n)\}_{n=1}^{N} \mapsto \{\boldsymbol{A}_n \boldsymbol{B}_n \boldsymbol{v}_n\}_{n=1}^{N}$$



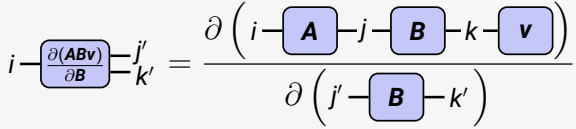
Function Transformations with Tensor Networks

[Example] Batching/`vmap`-ing: adding legs

$$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{v}) \mapsto \boldsymbol{ABv} \qquad \{(\boldsymbol{A}_n, \boldsymbol{B}_n, \boldsymbol{v}_n)\}_{n=1}^{N} \mapsto \{\boldsymbol{A}_n \boldsymbol{B}_n \boldsymbol{v}_n\}_{n=1}^{N}$$



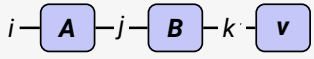
[Example] Differentiation: removing arguments



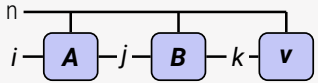
Function Transformations with Tensor Networks

[Example] Batching/`vmap`-ing: adding legs

$$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{v}) \mapsto \boldsymbol{ABv} \qquad \{(\boldsymbol{A}_n, \boldsymbol{B}_n, \boldsymbol{v}_n)\}_{n=1}^{N} \mapsto \{\boldsymbol{A}_n \boldsymbol{B}_n \boldsymbol{v}_n\}_{n=1}^{N}$$



→



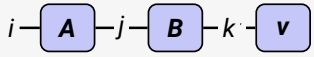
[Example] Differentiation: removing arguments

$$i \boxed{\frac{\partial(\boldsymbol{ABv})}{\partial \boldsymbol{B}}} \begin{matrix} j' \\ k' \end{matrix} = \frac{\partial \left(i \boxed{\boldsymbol{A}} j \boxed{\boldsymbol{B}} k \boxed{\boldsymbol{v}} \right)}{\partial \left(j' \boxed{\boldsymbol{B}} k' \right)} = i \boxed{\boldsymbol{A}} j' \qquad k' \boxed{\boldsymbol{v}}$$

Function Transformations with Tensor Networks

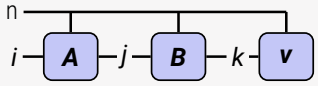
[Example] Batching/vmap-ing: adding legs

$$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{v}) \mapsto \boldsymbol{ABv}$$

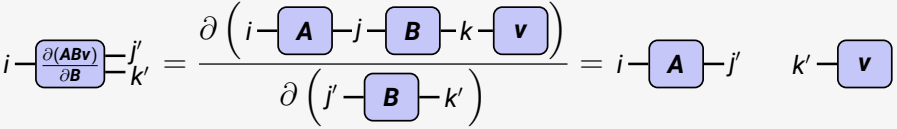


$\rightarrow$

$$\{(\boldsymbol{A}_n, \boldsymbol{B}_n, \boldsymbol{v}_n)\}_{n=1}^{N} \mapsto \{\boldsymbol{A}_n \boldsymbol{B}_n \boldsymbol{v}_n\}_{n=1}^{N}$$



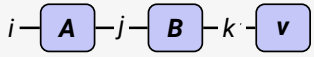
[Example] Differentiation: removing arguments



Function Transformations with Tensor Networks

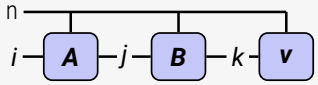
[Example] Batching/vmap-ing: adding legs

$$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{v}) \mapsto \boldsymbol{ABv}$$

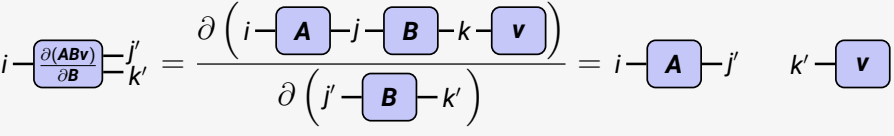


$$\rightarrow$$

$$\{(\boldsymbol{A}_n, \boldsymbol{B}_n, \boldsymbol{v}_n)\}_{n=1}^{N} \mapsto \{\boldsymbol{A}_n\boldsymbol{B}_n\boldsymbol{v}_n\}_{n=1}^{N}$$



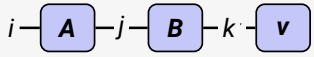
[Example] Differentiation: removing arguments



Function Transformations with Tensor Networks

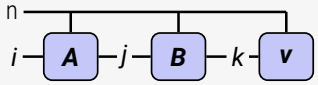
[Example] Batching/vmap-ing: adding legs

$$(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{v}) \mapsto \boldsymbol{ABv}$$

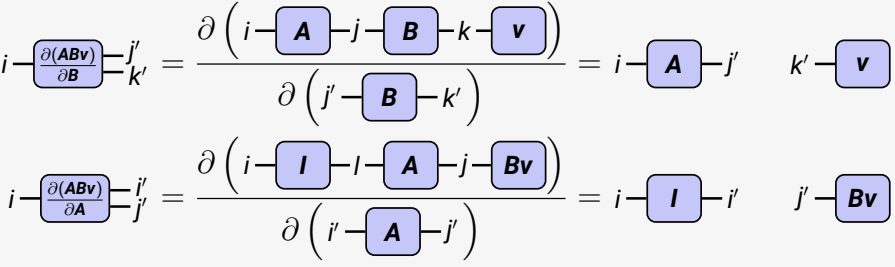


$$\rightarrow$$

$$\{(\boldsymbol{A}_n, \boldsymbol{B}_n, \boldsymbol{v}_n)\}_{n=1}^{N} \mapsto \{\boldsymbol{A}_n \boldsymbol{B}_n \boldsymbol{v}_n\}_{n=1}^{N}$$



[Example] Differentiation: removing arguments



Convolutions as Tensor Networks

Convolution as Tensor Network [Hayashi et al., 2019]

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

Convolution as Tensor Network [Hayashi et al., 2019]

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

Convolution as Tensor Network [Hayashi et al., 2019]

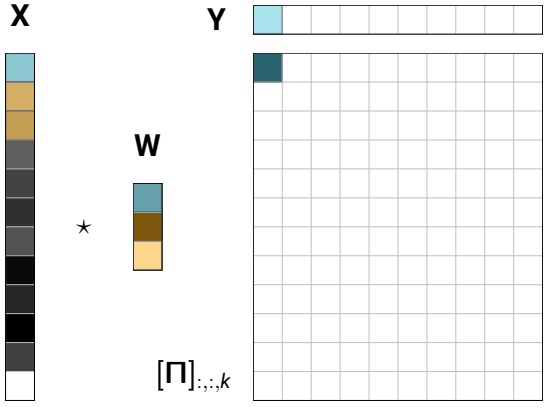
Index pattern $\Pi(I, K, S, P, D)$ captures the convolution's connectivity

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

Convolution as Tensor Network [Hayashi et al., 2019]

Index pattern $\Pi(I, K, S, P, D)$ captures the convolution's connectivity

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

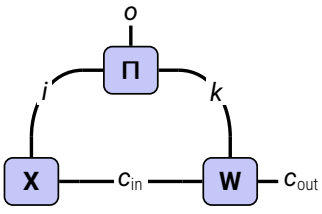


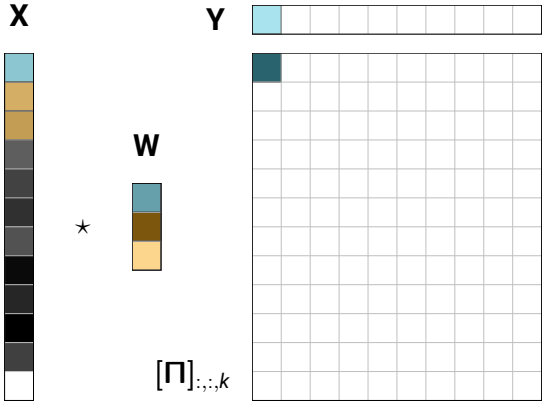
Convolution as Tensor Network [Hayashi et al., 2019]

Index pattern $\Pi(I, K, S, P, D)$ captures the convolution's connectivity

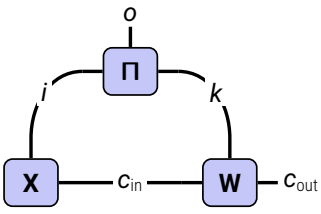
$$Y_{c_{\text{out}},o} = \sum_{c_{\text{in}},k} X_{c_{\text{in}},i(k,o)} W_{c_{\text{out}},c_{\text{in}},k}$$

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



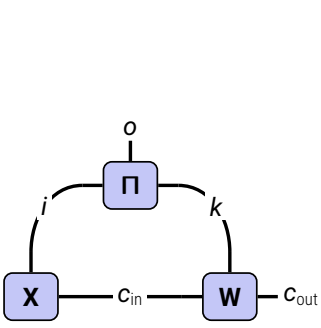


Higher-dimensional Convolutions [Hayashi et al., 2019]

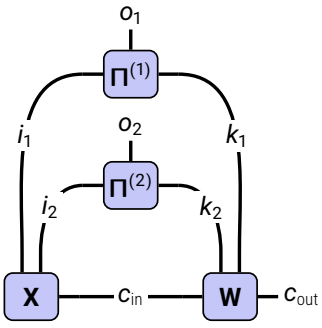


1d

Higher-dimensional Convolutions [Hayashi et al., 2019]

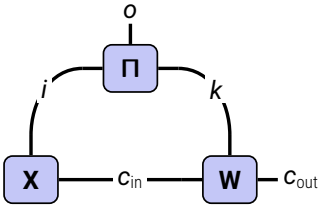


1d

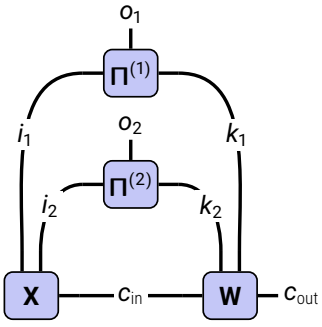


2d

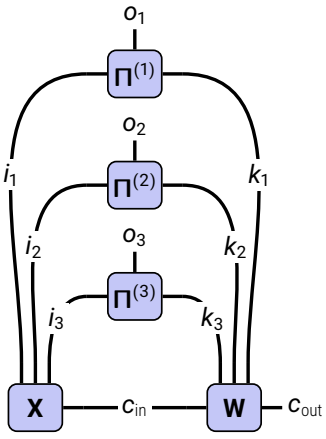
Higher-dimensional Convolutions [Hayashi et al., 2019]



1d

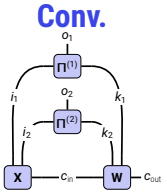


2d



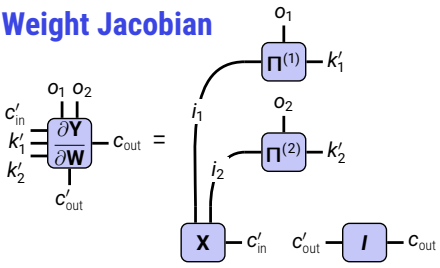
3d

Derivatives & Autodiff Routines

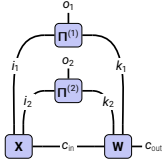


Derivatives & Autodiff Routines

Weight Jacobian

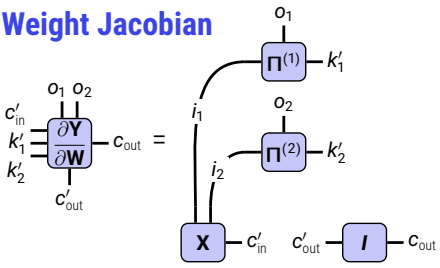


Conv.

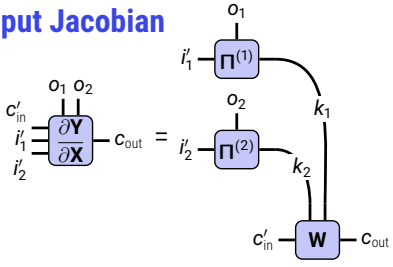


Derivatives & Autodiff Routines

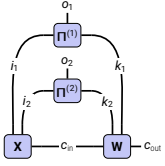
Weight Jacobian



Input Jacobian

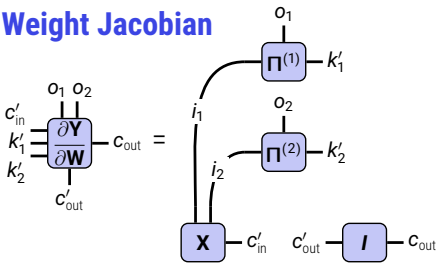


Conv.

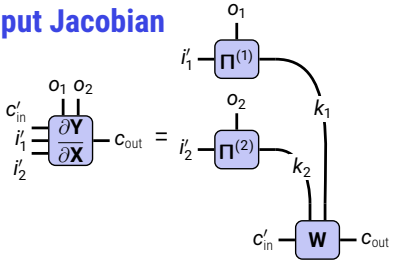


Derivatives & Autodiff Routines

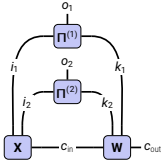
Weight Jacobian



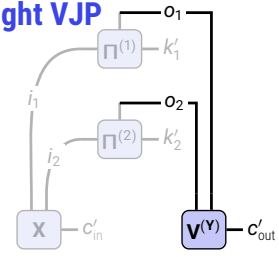
Input Jacobian



Conv.

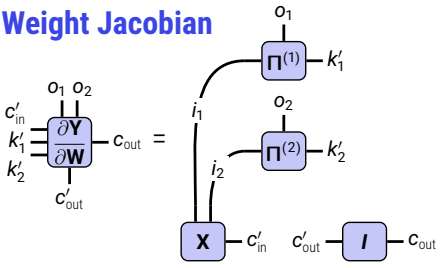


Weight VJP

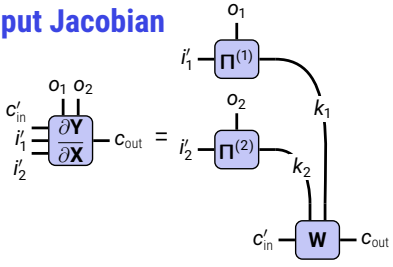


Derivatives & Autodiff Routines

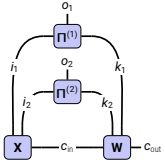
Weight Jacobian



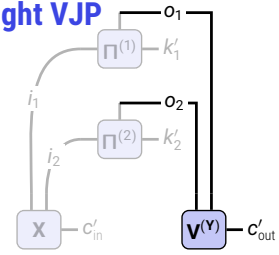
Input Jacobian



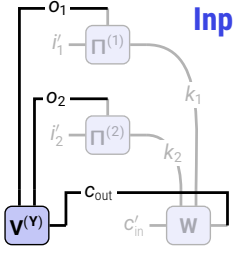
Conv.



Weight VJP



Input VJP



Use Cases

Improving KFAC-reduce [Eschenhagen et al., 2023, NeurIPS]

State of the art

x

Improving KFAC-reduce [Eschenhagen et al., 2023, NeurIPS]

State of the art

$\mathbf{X}$

$\rightarrow [\![\mathbf{X}]\!]$

Improving KFAC-reduce [Eschenhagen et al., 2023, NeurIPS]

State of the art

$\mathbf{X}$

$\rightarrow [\![\mathbf{X}]\!]$

$\rightarrow \mathbf{1}^\top [\![\mathbf{X}]\!]$

Improving KFAC-reduce [Eschenhagen et al., 2023, NeurIPS]

State of the art

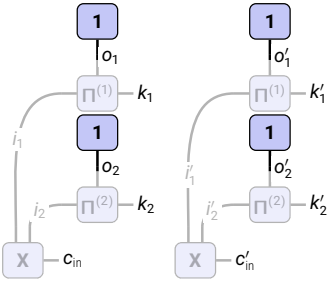
$$\mathbf{X}$$
$$\rightarrow [\![\mathbf{X}]\!]$$
$$\rightarrow \mathbf{1}^\top [\![\mathbf{X}]\!]$$
$$\rightarrow \left(\mathbf{1}^\top [\![\mathbf{X}]\!]\right)^\top \left(\mathbf{1}^\top [\![\mathbf{X}]\!]\right)$$

Improving KFAC-reduce [Eschenhagen et al., 2023, NeurIPS]

State of the art

$$\mathbf{X}$$
$$\to [\![\mathbf{X}]\!]$$
$$\to \mathbf{1}^\top [\![\mathbf{X}]\!]$$
$$\to \left(\mathbf{1}^\top [\![\mathbf{X}]\!]\right)^\top \left(\mathbf{1}^\top [\![\mathbf{X}]\!]\right)$$

Tensor Network



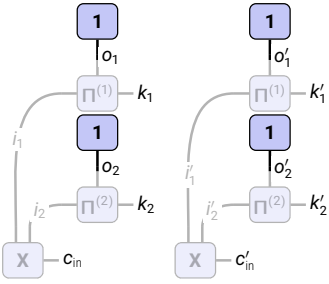
Improving KFAC-reduce [Eschenhagen et al., 2023, NeurIPS]

State of the art

$$
\begin{aligned}
&\mathbf{X} \\
&\rightarrow [\![\mathbf{X}]\!] \\
&\rightarrow \mathbf{1}^\top [\![\mathbf{X}]\!] \\
&\rightarrow \left(\mathbf{1}^\top [\![\mathbf{X}]\!]\right)^\top \left(\mathbf{1}^\top [\![\mathbf{X}]\!]\right)
\end{aligned}
$$

Time: **9.87 ms**

Tensor Network



Time: **2.69 ms (3.7 x)**

(`features.1.0.block.0` convolution of ConvNeXt-base with (32, 3, 256, 256) input)

Improving KFAC-reduce [Eschenhagen et al., 2023, NeurIPS]

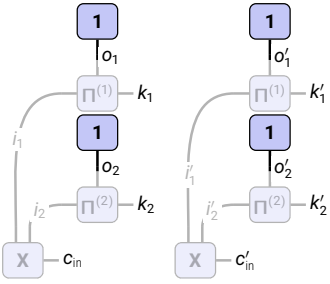
Non-conventional operations can be much cheaper with tensor networks.

State of the art

$$
\begin{aligned}
&\mathbf{X} \\
&\rightarrow [\![\mathbf{X}]\!] \\
&\rightarrow \mathbf{1}^\top [\![\mathbf{X}]\!] \\
&\rightarrow \left(\mathbf{1}^\top [\![\mathbf{X}]\!]\right)^\top \left(\mathbf{1}^\top [\![\mathbf{X}]\!]\right)
\end{aligned}
$$

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

Tensor Network



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

(`features.1.0.block.0` convolution of ConvNeXt-base with (32, 3, 256, 256) input)

Improving Second-Order Optimizers

[Lin, Dangel, Eschenhagen, Neklyudov, Kristiadi, Turner, and Makhzani, 2024, ICML]

ImageNet GPU benchmark (NVIDIA A40) with (128, 3, 256, 256) inputs

Net	Optimizer	Per-iteration [s]	Peak memory [GiB]
ResNet18	SGD	$1.17 \cdot 10^{-1}$ (1.00 x)	3.62 (1.00 x)
VGG19	SGD	$6.90 \cdot 10^{-1}$ (1.00 x)	14.1 (1.00 x)

Improving Second-Order Optimizers

[Lin, Dangel, Eschenhagen, Neklyudov, Kristiadi, Turner, and Makhzani, 2024, ICML]

ImageNet GPU benchmark (NVIDIA A40) with (128, 3, 256, 256) inputs

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)**
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 uses KFAC-reduce and diagonal pre-conditioners which are updated every step)

Improving Second-Order Optimizers

[Lin, Dangel, Eschenhagen, Neklyudov, Kristiadi, Turner, and Makhzani, 2024, ICML]

ImageNet GPU benchmark (NVIDIA A40) with (128, 3, 256, 256) inputs

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)**

(SINGD uses KFAC-reduce and diagonal pre-conditioners which are updated every step)

Significantly reduces the computational gap of second-order methods.

Convolutions and More as `einsum`

- TN perspective simplifies the transfer of algorithmic ideas
- Enables flexible/faster implementations of black box routines
- Relies on automatically efficient evaluation inside `einsum`

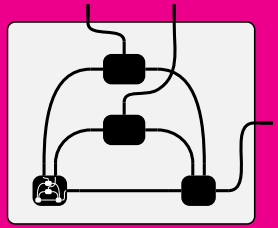
Convolutions and More as `einsum`

- TN perspective simplifies the transfer of algorithmic ideas
- Enables flexible/faster implementations of black box routines
- Relies on automatically efficient evaluation inside `einsum`

Try it out!

```
from einconv.expressions import kfac_reduce
from torch import einsum

# create the tensor network
equation, operands, shape = kfac_reduce.
    einsum_expression(..., simplify=True)
# evaluate it
einsum(equation, *operands).reshape(shape)
```



`pip install einconv`

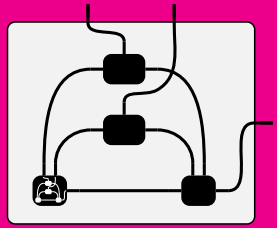
Convolutions and More as `einsum`

- TN perspective simplifies the transfer of algorithmic ideas
- Enables flexible/faster implementations of black box routines
- Relies on automatically efficient evaluation inside `einsum`

Try it out!

```
from einconv.expressions import kfac_reduce
from torch import einsum

# create the tensor network
equation, operands, shape = kfac_reduce.
    einsum_expression(..., simplify=True)
# evaluate it
einsum(equation, *operands).reshape(shape)
```



pip install einconv

Paper: arxiv/2307.02275

Code: github.com/f-dangel/einconv

Thank you 🙏 questions?

References I

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. **Advances in neural information processing systems (NeurIPS)**, 2019.

Suzanna Becker and Yann Lecun. Improving the convergence of back-propagation learning with second-order methods. 1989.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In **International Conference on Machine Learning (ICML)**, 2017.

Felix Dangel, Frederik Kunstner, and Philipp Hennig. BackPACK: Packing more into backprop. In **International Conference on Learning Representations (ICLR)**, 2020.

Mohamed Elsayed and A. Rupam Mahmood. HesScale: Scalable computation of hessian diagonals. 2023.

Mohamed Elsayed, Homayoon Farrahi, Felix Dangel, and A. Rupam Mahmood. Revisiting scalable hessian diagonal approximations for applications in reinforcement learning. In **International Conference on Machine Learning (ICML)**, 2024.

References II

Runa Eschenhagen, Alexander Immer, Richard E. Turner, Frank Schneider, and Philipp Hennig. Kronecker-factored approximate curvature for modern neural network architectures. In **Advances in Neural Information Processing Systems (NeurIPS)**, 2023.

Donald Goldfarb, Yi Ren, and Achraf Bahamou. Practical quasi-newton methods for training deep neural networks, 2021.

Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. **Quantum**, 2021.

Roger Grosse and James Martens. A kronecker-factored approximate Fisher matrix for convolution layers. In **International Conference on Machine Learning (ICML)**, 2016.

Kohei Hayashi, Taiki Yamaguchi, Yohei Sugawara, and Shin-ichi Maeda. Exploring unexplored tensor network decompositions for convolutional neural networks. In **Advances in Neural Information Processing Systems (NeurIPS)**, 2019.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks, 2020.

Sören Laue, Matthias Mitterreiter, and Joachim Giesen. A simple and efficient tensor calculus. In **AAAI Conference on Artificial Intelligence**, 2020.

References III

Wu Lin, Felix Dangel, Runa Eschenhagen, Kirill Neklyudov, Agustinus Kristiadi, Richard E. Turner, and Alireza Makhzani. Structured inverse-free natural gradient descent: Memory-efficient & numerically-stable KFAC. In **International Conference on Machine Learning (ICML)**, 2024.

Linjian Ma, Matthew Fishman, Miles Stoudenmire, and Edgar Solomonik. Approximate contraction of arbitrary tensor networks with a flexible and efficient density matrix algorithm. 2024.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In **International Conference on Machine Learning (ICML)**, 2015.

Hannah Pinson, Joeri Lenaerts, and Vincent Ginis. Linear cnns discover the statistical structure of the dataset using only the most dominant frequencies. In **International Conference on Machine Learning (ICML)**, 2023.

Yi Ren, Achraf Bahamou, and Donald Goldfarb. Kronecker-factored quasi-newton methods for deep learning, 2022.

Alex Rogozhnikov. Einops: Clear and reliable tensor manipulations with einstein-like notation. In **International Conference on Learning Representations (ICLR)**, 2022.

References IV

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks, 2014.

Sidak Pal Singh, Gregor Bachmann, and Thomas Hofmann. Analytic insights into structure and rank of neural network hessian maps. **Advances in Neural Information Processing Systems (NeurIPS)**, 2021.

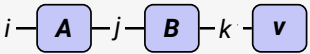
Sidak Pal Singh, Thomas Hofmann, and Bernhard Schölkopf. The hessian perspective into the nature of convolutional neural networks. 2023.

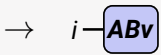
Daniel G. A. Smith and Johnnie Gray. opt_einsum - A python package for optimizing contraction order for einsum-like expressions. **Journal of Open Source Software (JOSS)**, 2018.

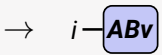
Contraction Path Optimization (e.g. `opt_einsum` [Smith and Gray, 2018])

[Example] Matrix-matrix-vector-product

Consider $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{3.000 \times 3.000}$ and $\boldsymbol{v} \in \mathbb{R}^{3.000}$.



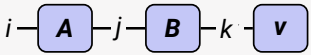


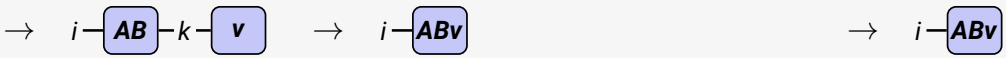


Contraction Path Optimization (e.g. `opt_einsum` [Smith and Gray, 2018])

[Example] Matrix-matrix-vector-product

Consider $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{3.000 \times 3.000}$ and $\boldsymbol{v} \in \mathbb{R}^{3.000}$.





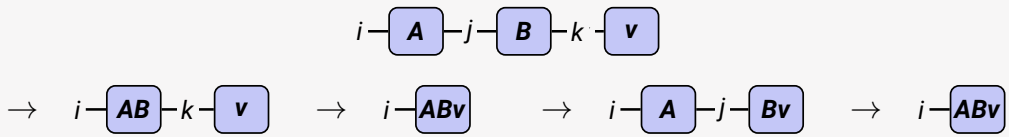
Schedule 1:

```
A @ B @ v
```

Contraction Path Optimization (e.g. `opt_einsum` [Smith and Gray, 2018])

[Example] Matrix-matrix-vector-product

Consider $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{3.000 \times 3.000}$ and $\boldsymbol{v} \in \mathbb{R}^{3.000}$.



Schedule 1:

```
A @ B @ v
```

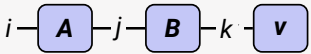
Schedule 2:

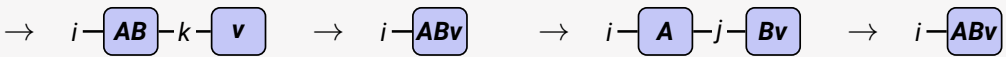
```
A @ (B @ v)
```

Contraction Path Optimization (e.g. `opt_einsum` [Smith and Gray, 2018])

[Example] Matrix-matrix-vector-product

Consider $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{3.000 \times 3.000}$ and $\boldsymbol{v} \in \mathbb{R}^{3.000}$.





Schedule 1: **1.1 s**

```
A @ B @ v
```

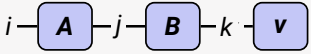
Schedule 2: **2.2 ms**

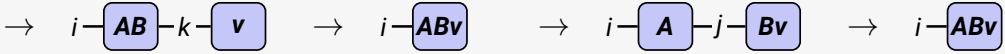
```
A @ (B @ v)
```

Contraction Path Optimization (e.g. `opt_einsum` [Smith and Gray, 2018])

[Example] Matrix-matrix-vector-product

Consider $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{3.000 \times 3.000}$ and $\boldsymbol{v} \in \mathbb{R}^{3.000}$.





Schedule 1: **1.1 s**

```
A @ B @ v
```

Schedule 2: **2.2 ms**

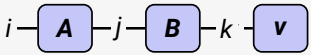
```
A @ (B @ v)
```

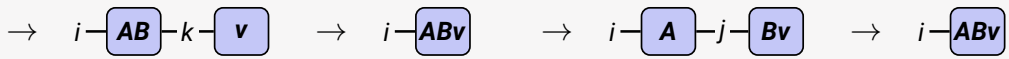
```
einsum("ij,jk,k->i", A, B, v)
```

Contraction Path Optimization (e.g. `opt_einsum` [Smith and Gray, 2018])

[Example] Matrix-matrix-vector-product

Consider $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{3.000 \times 3.000}$ and $\boldsymbol{v} \in \mathbb{R}^{3.000}$.





Schedule 1: **1.1 s**

```
A @ B @ v
```

Schedule 2: **2.2 ms**

```
A @ (B @ v)
```

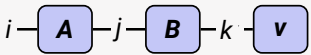
PyTorch: **1.1 s**

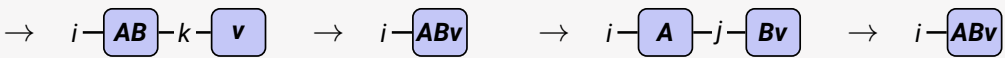
```
einsum("ij,jk,k->i", A, B, v)
```

Contraction Path Optimization (e.g. `opt_einsum` [Smith and Gray, 2018])

[Example] Matrix-matrix-vector-product

Consider $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{3.000 \times 3.000}$ and $\boldsymbol{v} \in \mathbb{R}^{3.000}$.





Schedule 1: **1.1 s**

```
A @ B @ v
```

Schedule 2: **2.2 ms**

```
A @ (B @ v)
```

PyTorch: **1.1 s**

with update + `pip install opt_einsum`, **2.3 ms**

```
einsum("ij,jk,k->i", A, B, v)
```

Order matters; `einsum` automatically finds a 'good' schedule.

Convolution-specific Simplifications

For structured patterns, we can re-wire the TN before evaluation
$\rightarrow$ even better performance

Convolution-specific Simplifications

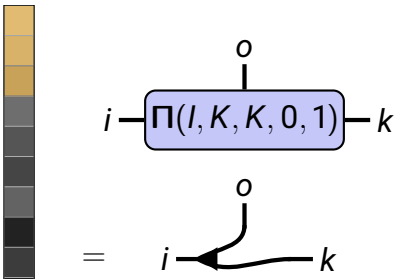
For structured patterns, we can re-wire the TN before evaluation
$\rightarrow$ even better performance

Dense convolution ($K = S$)

Convolution-specific Simplifications

For structured patterns, we can re-wire the TN before evaluation
$\rightarrow$ even better performance

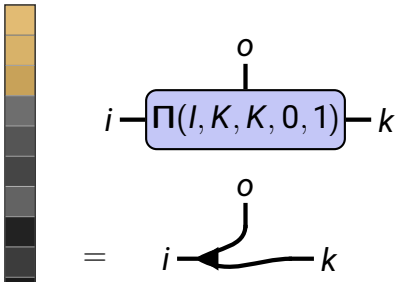
Dense convolution ($K = S$)



Convolution-specific Simplifications

For structured patterns, we can re-wire the TN before evaluation
$\rightarrow$ even better performance

Dense convolution ($K = S$)

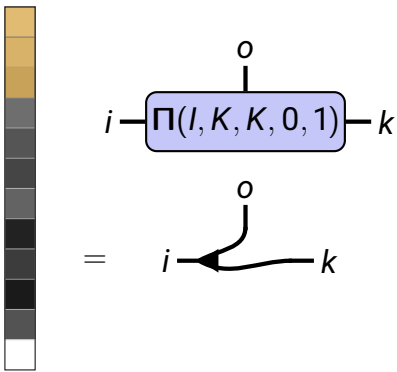


Down-sampling convolution ($S > K$)

Convolution-specific Simplifications

For structured patterns, we can re-wire the TN before evaluation
$\rightarrow$ even better performance

Dense convolution ($K = S$)



Down-sampling convolution ($S > K$)