Title: Deep Learning Convolutions Through the Lens of Tensor Networks

Speakers: Felix Dangel

Series: Machine Learning Initiative

Date: December 01, 2023 - 2:30 PM

URL: https://pirsa.org/23120027

Abstract: Despite their simple intuition, convolutions are more tedious to analyze than dense layers, which complicates the transfer of theoretical and algorithmic ideas. We provide a simplifying perspective onto convolutions through tensor networks (TNs) which allow reasoning about the underlying tensor multiplications by drawing diagrams, and manipulating them to perform function transformations and sub-tensor access. We demonstrate this expressive power by deriving the diagrams of various autodiff operations and popular approximations of second-order information with full hyper-parameter support, batching, channel groups, and generalization to arbitrary convolution dimensions. Further, we provide convolution-specific transformations based on the connectivity pattern which allow to re-wire and simplify diagrams before evaluation. Finally, we probe computational performance, relying on established machinery for efficient TN contraction. Our TN implementation speeds up a recently-proposed KFAC variant up to 4.5x and enables new hardware-efficient tensor dropout for approximate backpropagation.

Zoom link https://pitp.zoom.us/j/99090845943?pwd=NHBNVTNnbDNSOGNSVzNGS21xcllFdz09

Convolutions Are More Tedious than Linear Layers

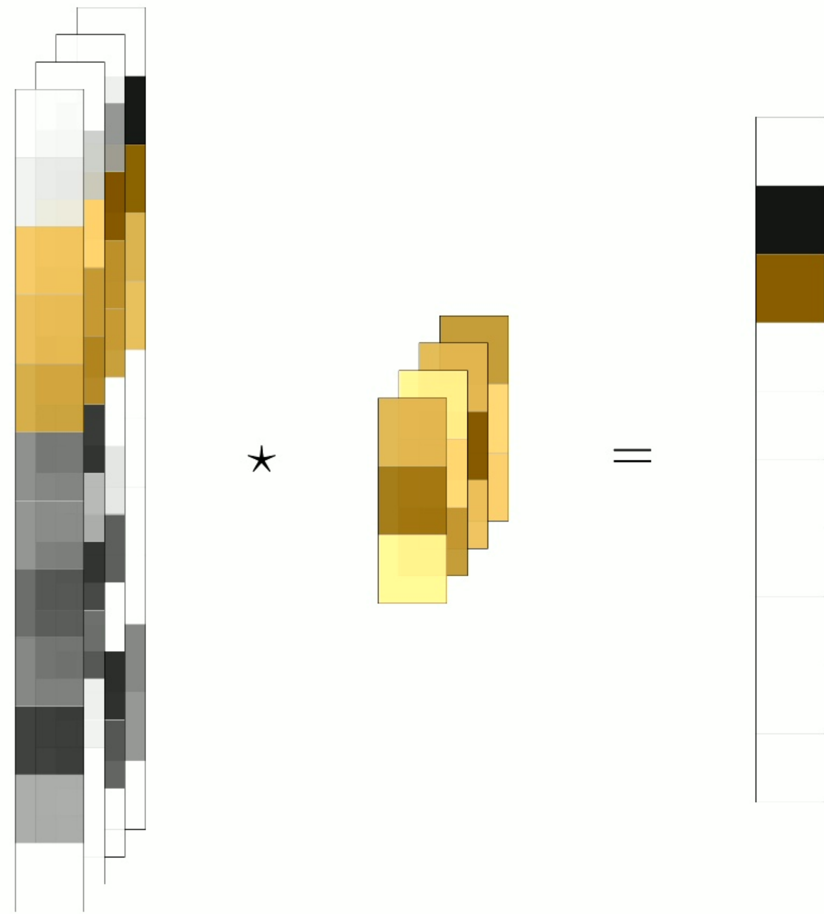
- Personal example: Vector-Jacobian products (VJPs)

```
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)
```

C_{in} is for Input Channels

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

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



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 $S_{\mathbf{A}_1}, \ldots, S_{\mathbf{A}_N}$

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

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 $\boldsymbol{C} = \boldsymbol{AB}$ as tensor multiplication

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

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

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

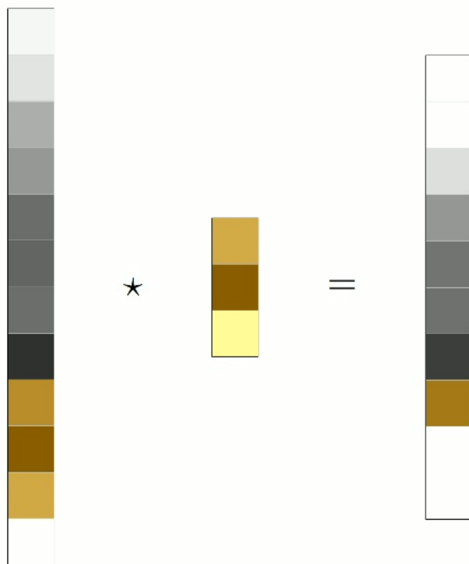
Convolution as Matrix Multiplication

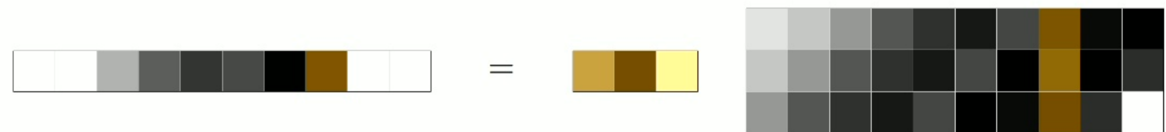
$$\mathbf{X} \in \mathbb{R}^{C_{\text{in}} \times I}$$
$$\mathbf{Y} \in \mathbb{R}^{C_{\text{out}} \times O}$$
$$\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K}$$

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

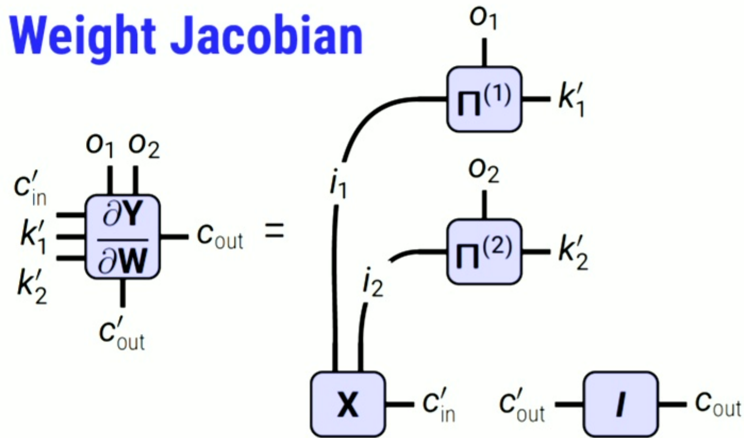
$$[\![\mathbf{X}]\!] \in \mathbb{R}^{C_{\text{in}} K \times O}$$
$$\mathbf{Y} \in \mathbb{R}^{C_{\text{out}} \times O}$$
$$\boldsymbol{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} K}$$

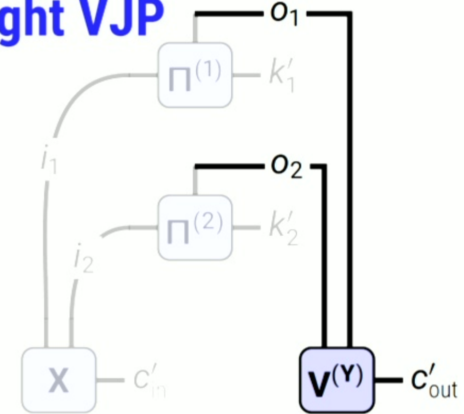
$$\boldsymbol{Y} = \boldsymbol{W}[\![\boldsymbol{X}]\!]$$

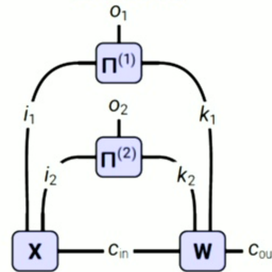


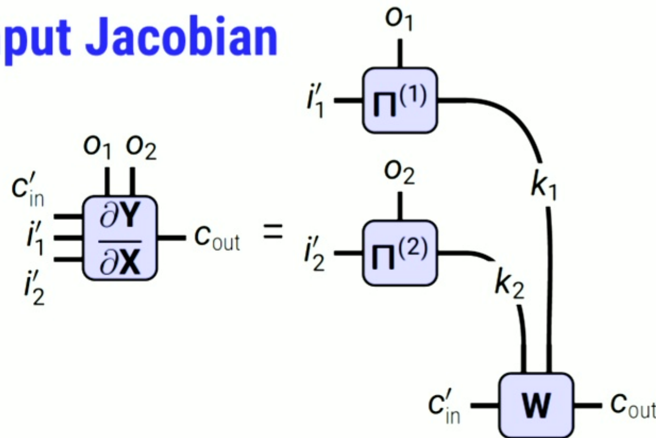


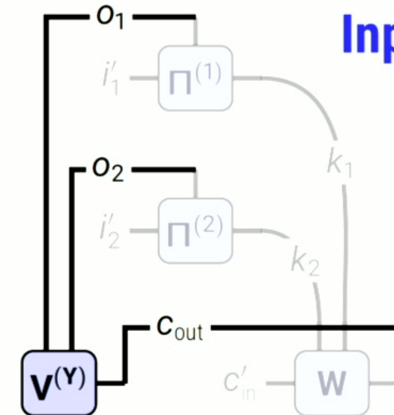
Derivatives & Autodiff Routines











Convolutions Through the Lens of Tensor Networks

Felix Dangel

December 01, 2023

Convolutions Are More Tedious than Linear Layers

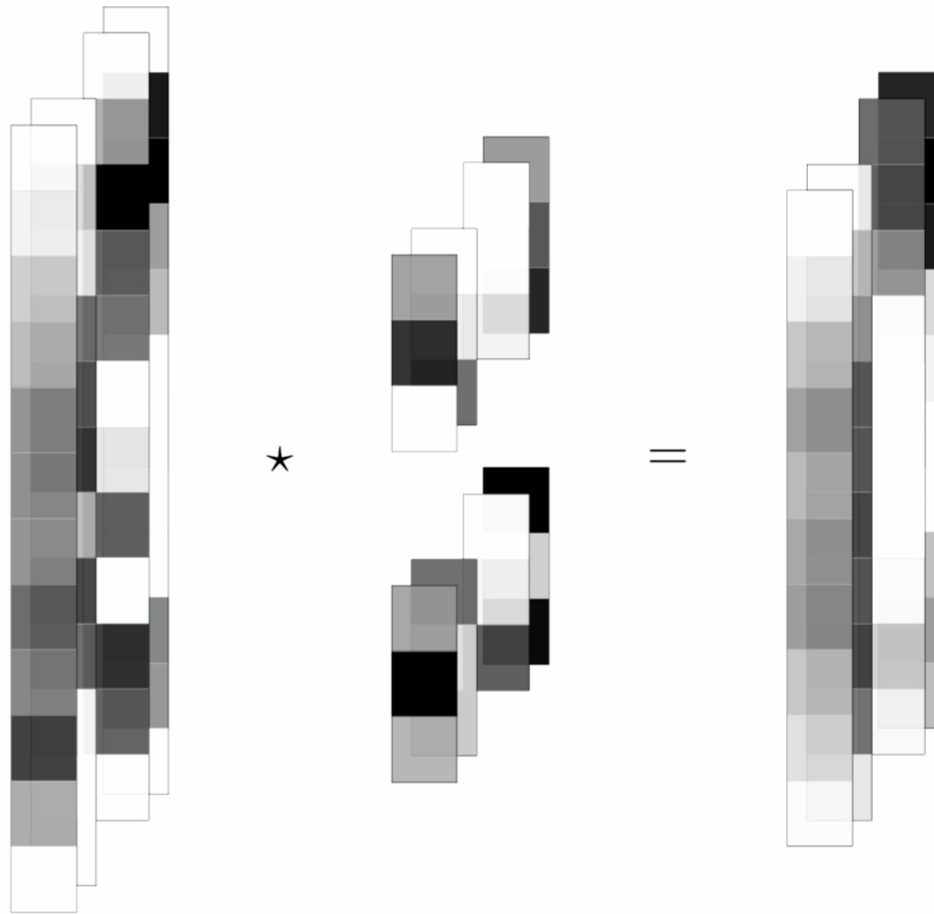
- Personal example: Vector-Jacobian products (VJPs)
- Other algorithmic & theoretical ideas

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

G is for Groups

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

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



Tensor Networks: Graphical Notation for Tensor Multiplications

Operation	Diagram
Scalar	c
Vector	$c - i$
Matrix	$i - C - j$
Tensor	$i - C \equiv j, k$
Matricize	$c - (i,j) \leftarrow i, j$
Flatten	$C \equiv i, j \rightarrow (i,j)$
Trace	$C - i$ (loop)

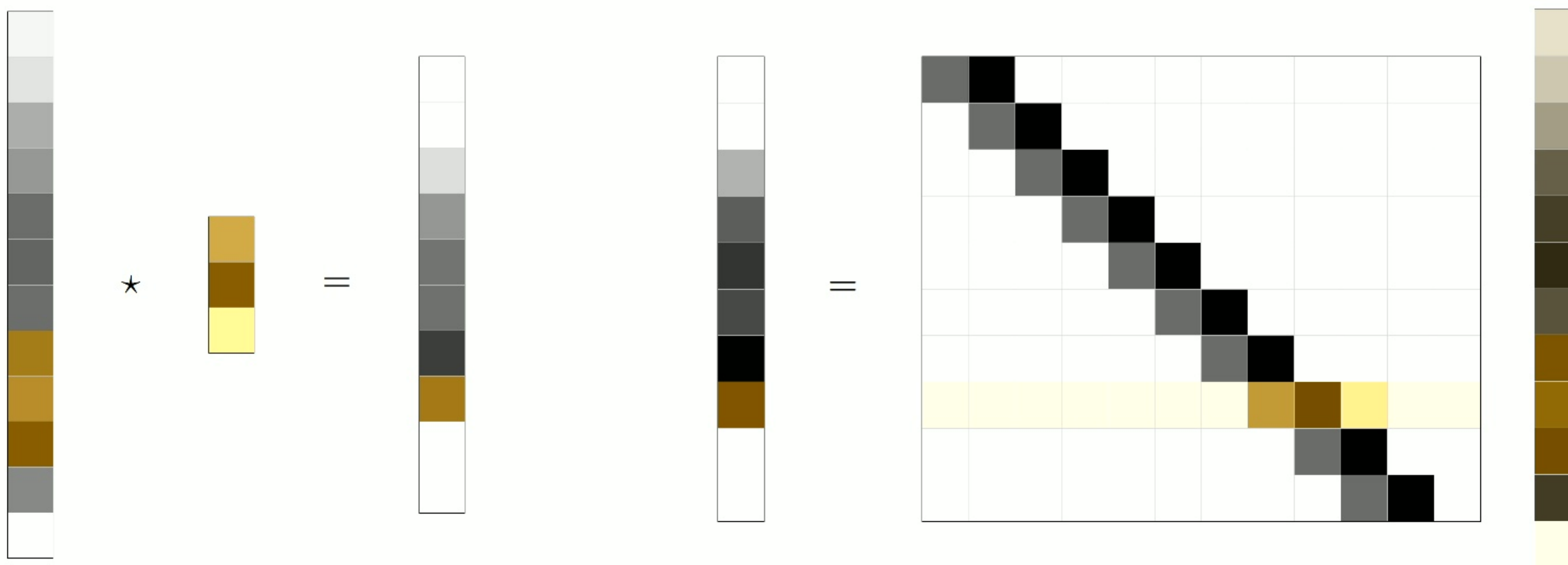
Operation	Diagram
$i - AB - k$	$i - A - j - B - k$

19

Convolution as Structured Matrix Multiplication

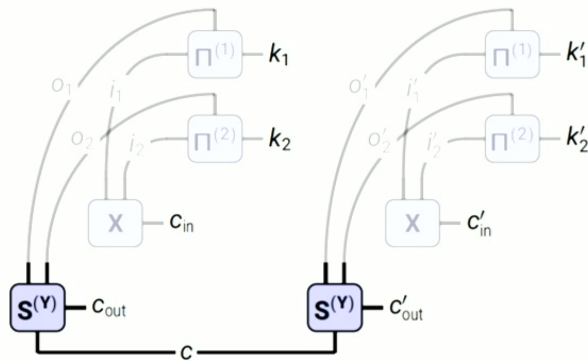
$$\mathbf{X} \in \mathbb{R}^{C_{\text{in}} \times I}$$
$$\mathbf{Y} \in \mathbb{R}^{C_{\text{out}} \times O}$$
$$\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K}$$

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} \qquad \boldsymbol{y} = \boldsymbol{A}(\mathbf{W})\boldsymbol{x}$$

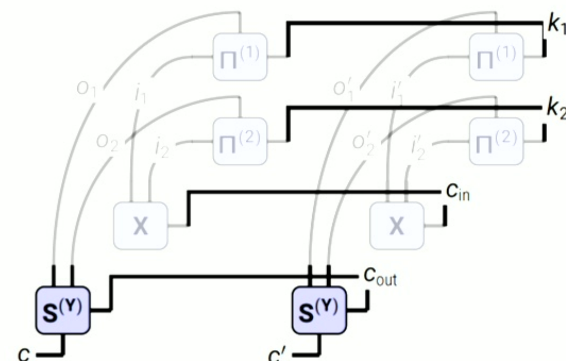


Higher-order Derivatives & Approximations

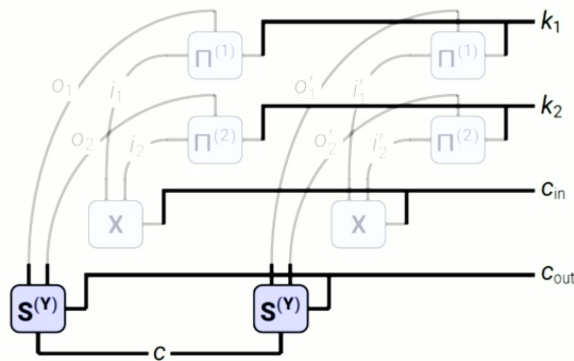
Fisher/GGN block



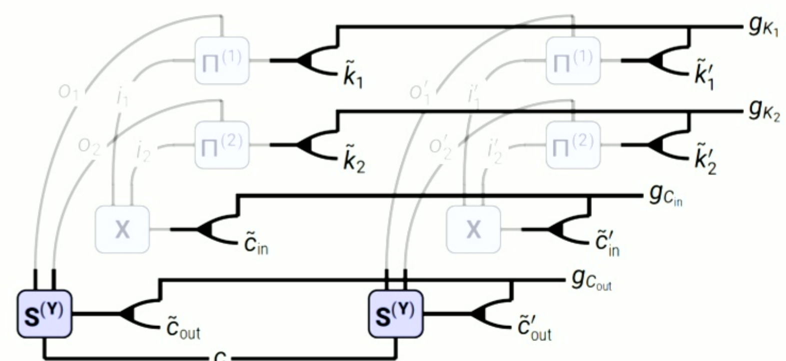
GGN Gram/Empirical NTK matrix



Fisher/GGN diagonal



Fisher/GGN mini-block diagonal



Convolutions ~~Through the Lens of Tensor Networks~~ as `einsum`

Felix Dangel

December 01, 2023

Convolutions Are More Tedious than Linear Layers

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

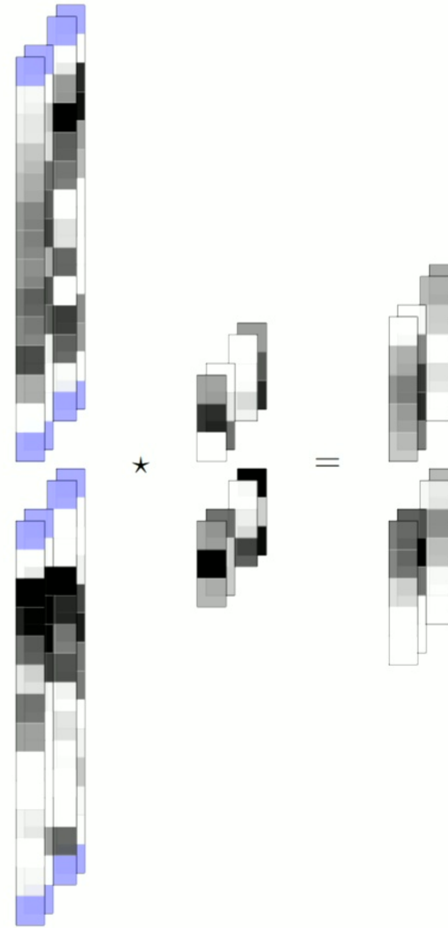
Concept	**for MLPs**	**for CNNs**
Approximate Hessian diagonal	1989	2023
Kronecker-factored curvature (KFAC, KFRA, KFLR)	2015, 2017, 2017	2016, 2020, 2020
Kronecker-factored quasi-Newton methods (KBFGS)	2021	2022
Neural tangent kernel (NTK)	2018	2019
Hessian rank	2021	2023

This talk: A simplifying perspective of convolutions through tensor networks

Putting Everything Together in One Dimension...

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

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



Tensor Networks: Graphical Notation for Tensor Multiplications

Operation	Diagram
Scalar	$\boxed{c}$
Vector	$\boxed{c}-i$
Matrix	$i-\boxed{C}-j$
Tensor	$i-\boxed{C}=^{j}_{k}$
Matricize	$\boxed{c}-(i,j) \prec^{i}_{j}$
Flatten	$\boxed{C}=^{i}_{j} \succ (i,j)$
Trace	$\boxed{C}-i$ (loop)

Operation	Diagram
$i-\boxed{AB}-k$	$i-\boxed{A}-j-\boxed{B}-k$
$i-\boxed{A \odot B}-j$	$i-[\boxed{A}, \boxed{B}]-j$
$(i,k)-\boxed{A \otimes B}-(j,l)$	$(i,k) \prec i-\boxed{A}-j,\ k-\boxed{B}-l \succ (j,l)$
$\boxed{\mathrm{diag}(A)}-i$	$\boxed{A}-i$ (loop)
$i-\boxed{\mathrm{diag}(a)}-i$	$i-\boxed{a}-i$

19

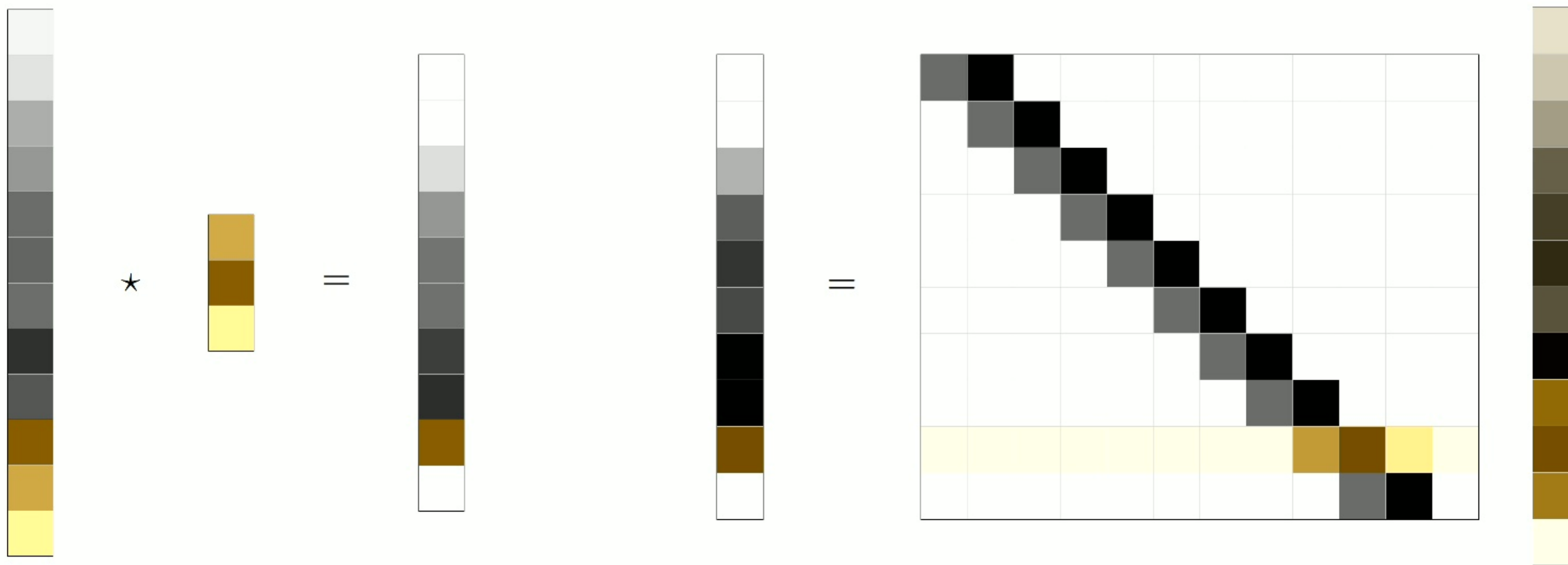
Convolution as Structured Matrix Multiplication

$$\mathbf{X} \in \mathbb{R}^{C_{\text{in}} \times I} \qquad \boldsymbol{x} \in \mathbb{R}^{C_{\text{in}} I}$$

$$\mathbf{Y} \in \mathbb{R}^{C_{\text{out}} \times O} \qquad \boldsymbol{y} \in \mathbb{R}^{C_{\text{out}} O}$$

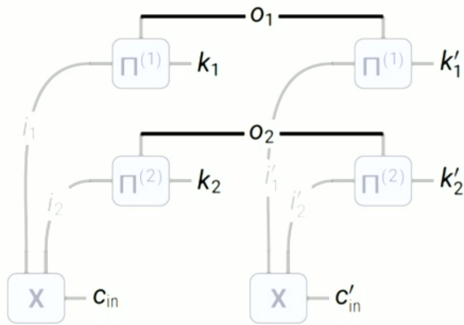
$$\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K} \qquad \boldsymbol{A}(\mathbf{W}) \in \mathbb{R}^{C_{\text{out}} O \times C_{\text{in}} I}$$

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} \qquad \boldsymbol{y} = \boldsymbol{A}(\mathbf{W}) \boldsymbol{x}$$

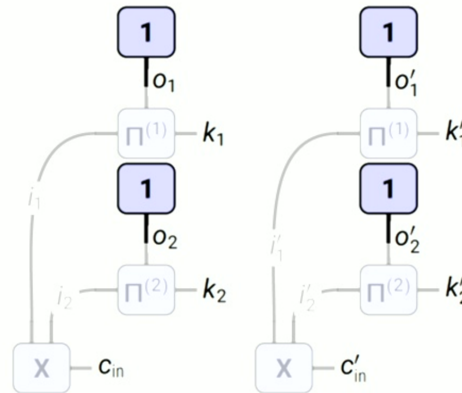


... and More!

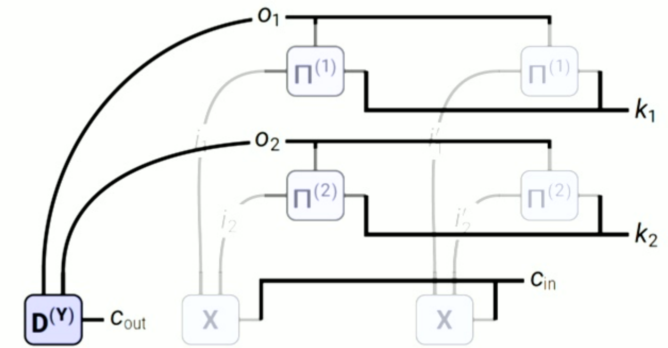
KFC/KFAC-expand



KFAC-reduce



Approximate Hessian diagonal



Check out the table in the paper's appendix!

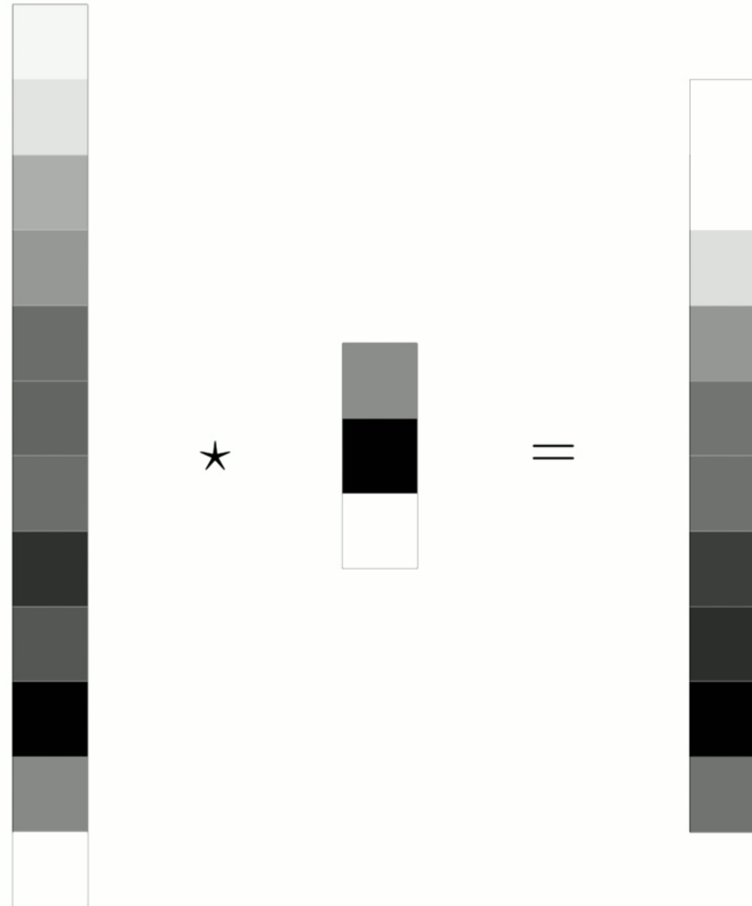
The Broader Picture

1

A Simple Convolution

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

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

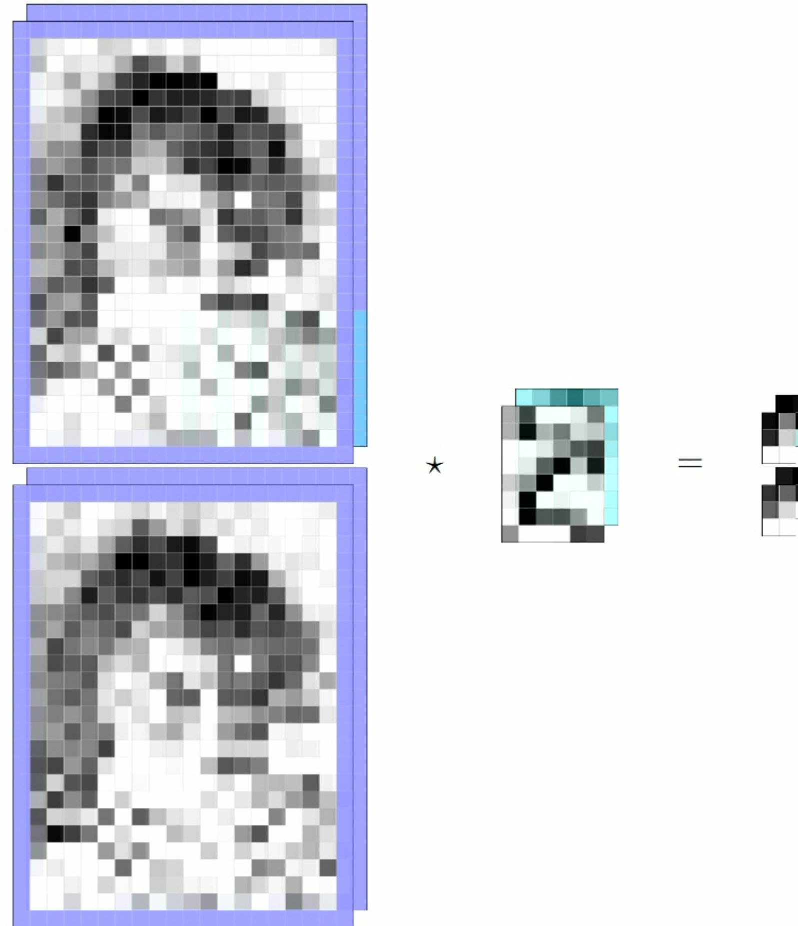


6

. . . and in Two Dimensions

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

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



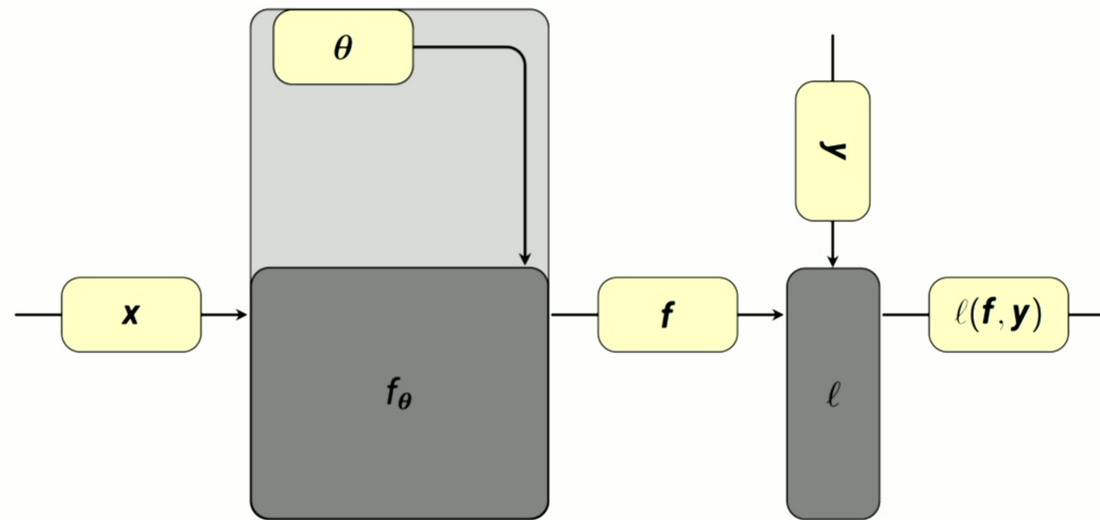
Benefits of TN/`einsum` Abstraction

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

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

Example & Conclusion

Supervised Deep Learning in a Nutshell (Math)



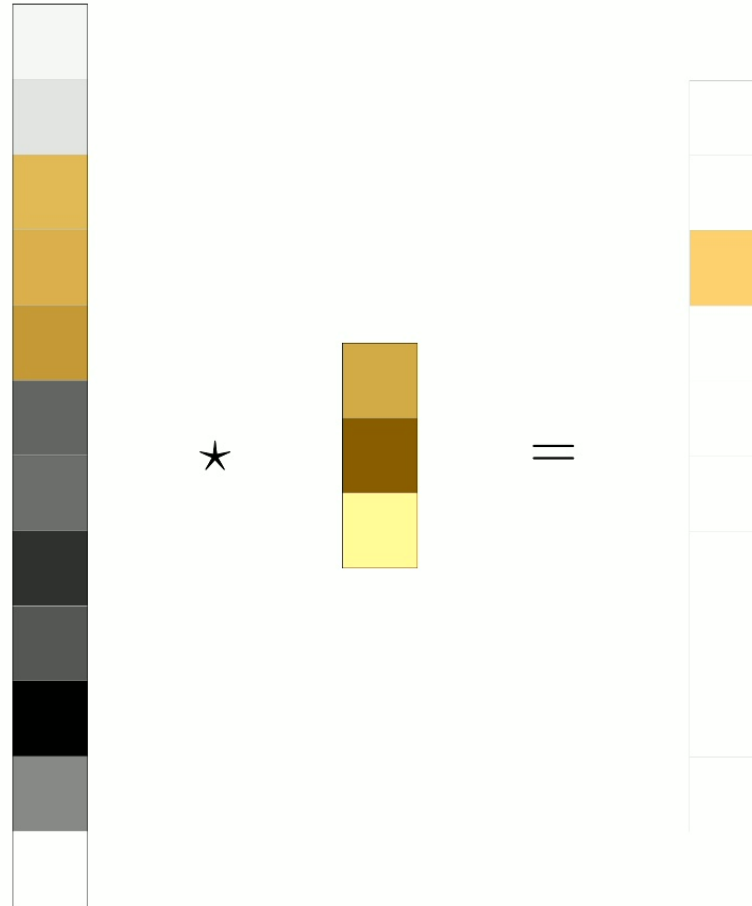
$$\ell(\boldsymbol{f}_{\theta}(\boldsymbol{x}\ \ ), \boldsymbol{y}\ \ )$$

1

A Simple Convolution

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

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

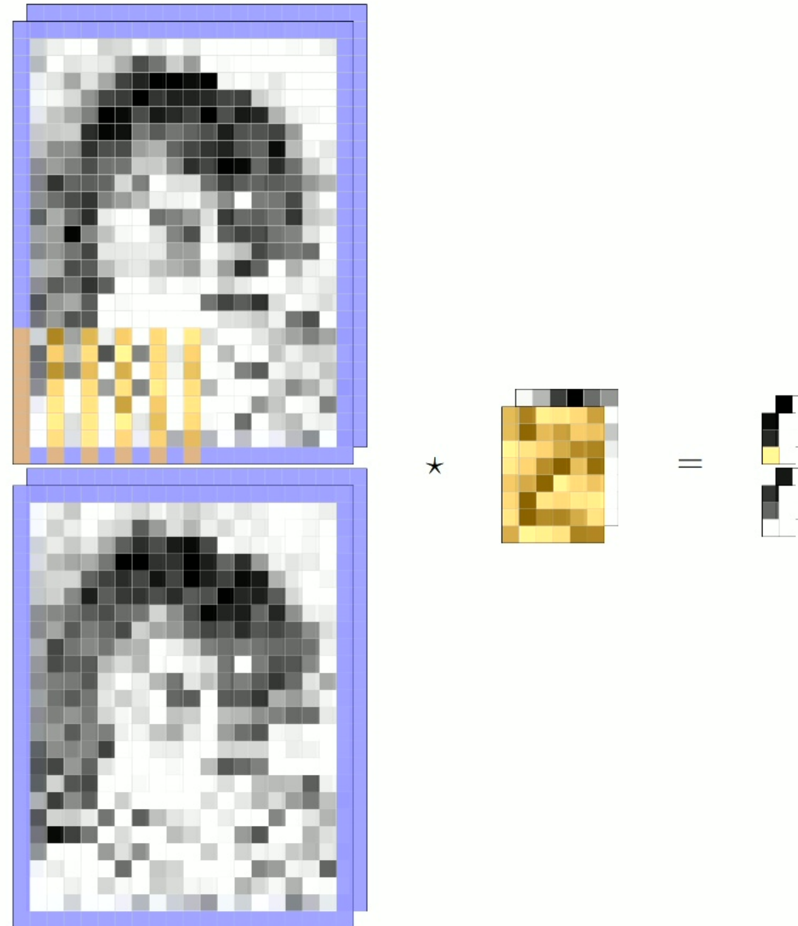


6

. . . and in Two Dimensions

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

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



Improved `einsum` Syntax: `einops` [Rogozhnikov, 2022, ICLR]

Powerful syntax extensions: multi-letter indices, index (un-)grouping, . . .

[Example] Batched matrix-matrix multiplication

Given $\{\boldsymbol{A}_n\}_{n=1}^N$, $\{\boldsymbol{B}_n\}_{n=1}^N$, compute $\{\boldsymbol{C}_n\}_{n=1}^N = \{\boldsymbol{A}_n\boldsymbol{B}_n\}_{n=1}^N$

```
C = einsum(A, B, "batch i j, batch j k -> batch i k")
```

[Example] Improved readability (from an ICLR 2024 submission)

```
1 x = x.unfold(2, kernel_size[0], stride[0])
2 x = x.unfold(3, kernel_size[1], stride[1])
3 x = x.transpose_(1, 2).transpose_(2, 3)
4 return torch.mean(
5     x.reshape((x.size(0), x.size(1), x.size(2), groups, -1, x.size(4), x.size(5))),
6     3,
7 ).view(x.size(0), x.size(1), x.size(2), -1)
```

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

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

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

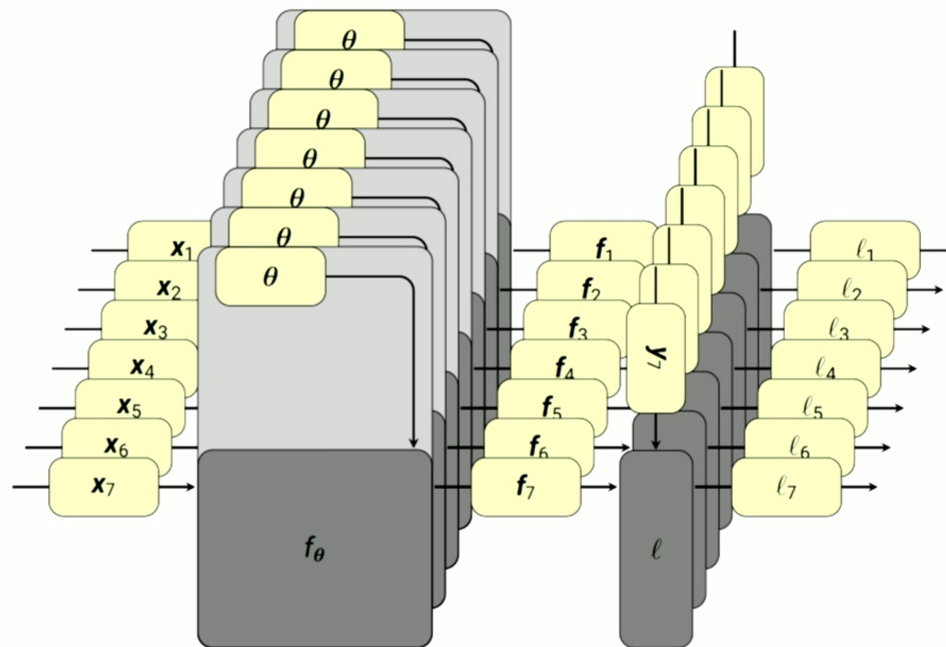
$$= \sum_{c_{\text{in}},k} \sum_{i} X_{c_{\text{in}},i} \Pi_{i,o,k} W_{c_{\text{out}},c_{\text{in}},k}$$

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

State of the art

X

Supervised Deep Learning in a Nutshell (Math)



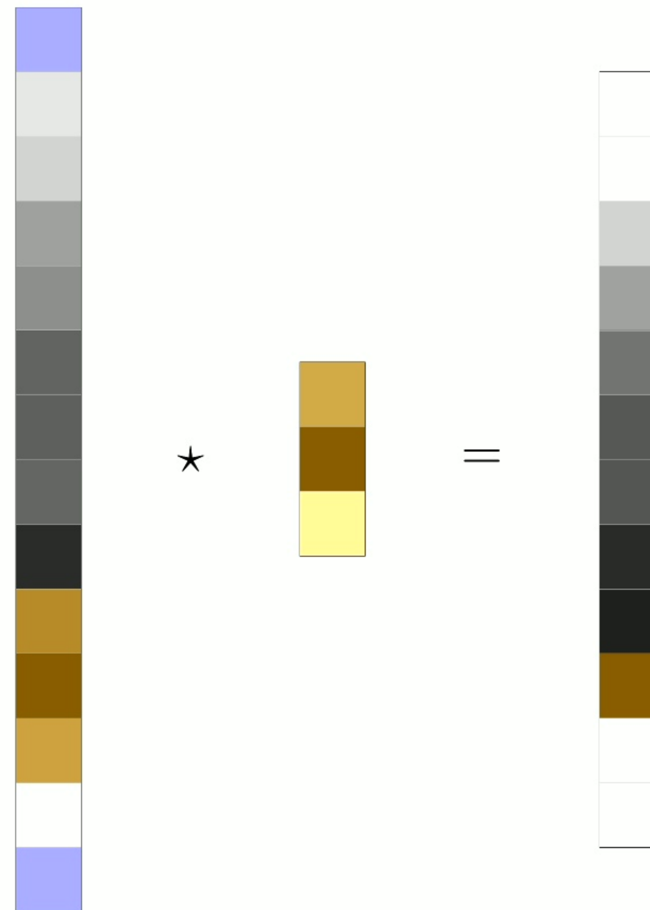
Minimize $\mathcal{L}_{\mathbb{D}}(\boldsymbol{\theta})$ with

$$\mathcal{L}_{\mathbb{D}}(\boldsymbol{\theta}) = \frac{1}{|\mathbb{D}|} \sum_{(\boldsymbol{x}_n, \boldsymbol{y}_n) \in \mathbb{D}} \ell(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_n), \boldsymbol{y}_n)$$

P is for Padding

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

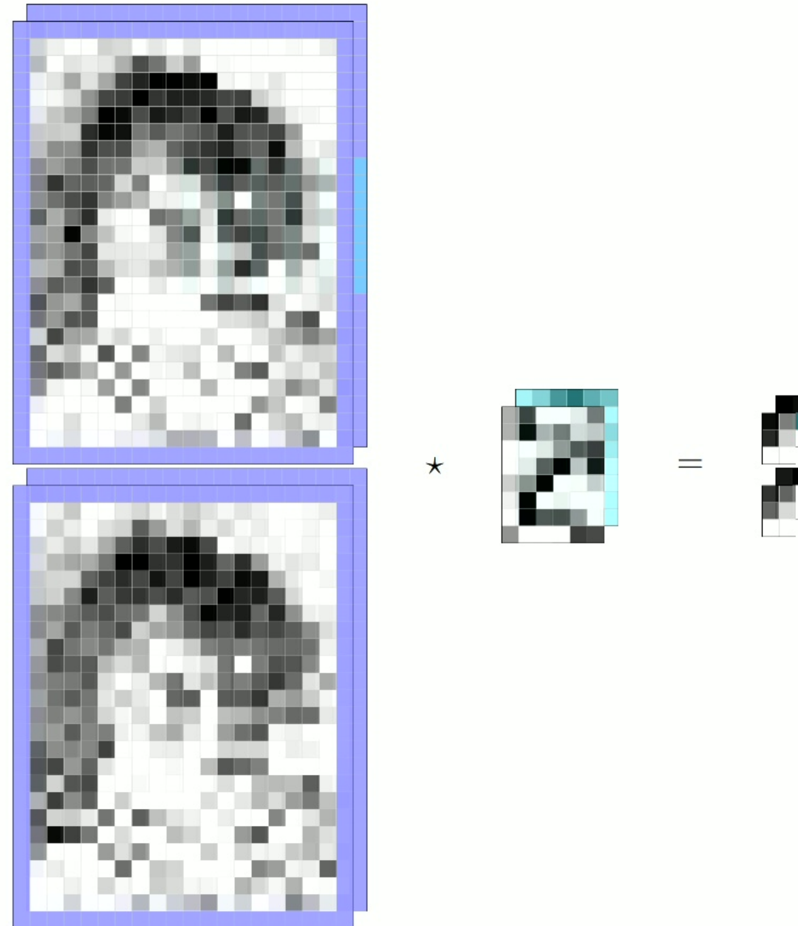
- $\mathbf{X} \in \mathbb{R}^I$
- $\mathbf{W} \in \mathbb{R}^K$
- $\mathbf{Y} \in \mathbb{R}^O$



... and in Two Dimensions

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

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



Improved einsum Syntax: einops [Rogozhnikov, 2022, ICLR]

Powerful syntax extensions: multi-letter indices, index (un-)grouping, . . .

[Example] Batched matrix-matrix multiplication

Given $\{\boldsymbol{A}_n\}_{n=1}^N$, $\{\boldsymbol{B}_n\}_{n=1}^N$, compute $\{\boldsymbol{C}_n\}_{n=1}^N = \{\boldsymbol{A}_n\boldsymbol{B}_n\}_{n=1}^N$

```
C = einsum(A, B, "batch i j, batch j k -> batch i k")
```

[Example] Improved readability (from an ICLR 2024 submission)

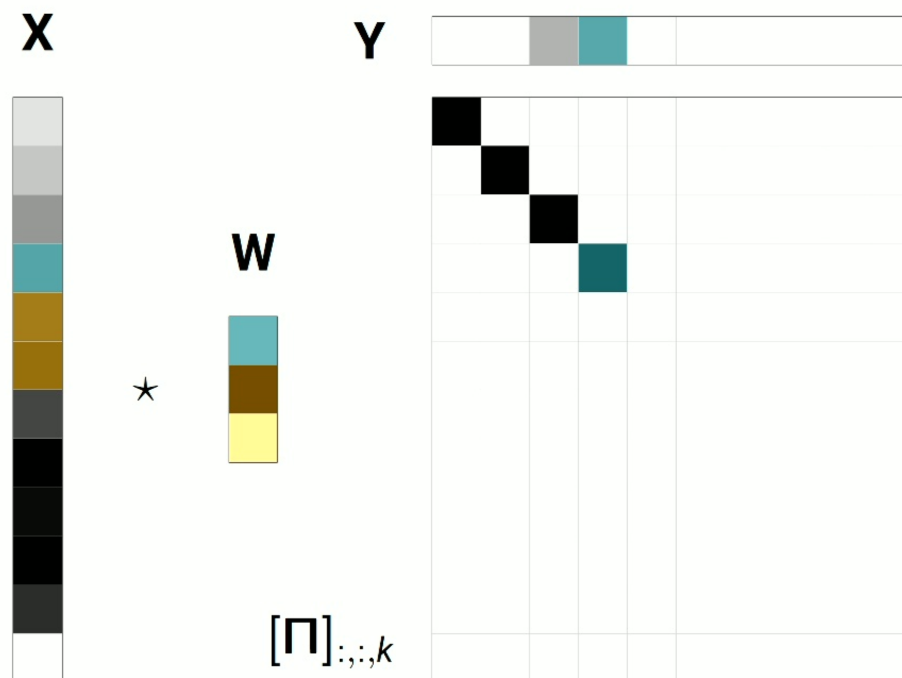
```
1 # average channel groups
2 x = rearrange(x, "b (g c_in) i1 i2 -> b g c_in i1 i2", g=groups)
3 x = reduce(x, "b g c_in i1 i2 -> b c_in i1 i2", "mean")
4
5 x_unfold = F.unfold(x, kernel_size, dilation=dilation, padding=padding, stride=stride)
6 return rearrange(x_unfold, "b c_in_k1_k2 o1_o2 -> b o1_o2 c_in_k1_k2")
```

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

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

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

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



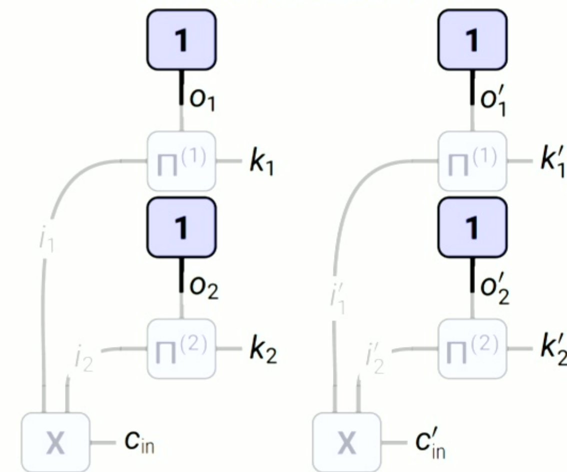
Improving KFAC-reduce [Eschenhagen et al., 2023, NeurIPS]

State of the art

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

Time: **9.87 ms**

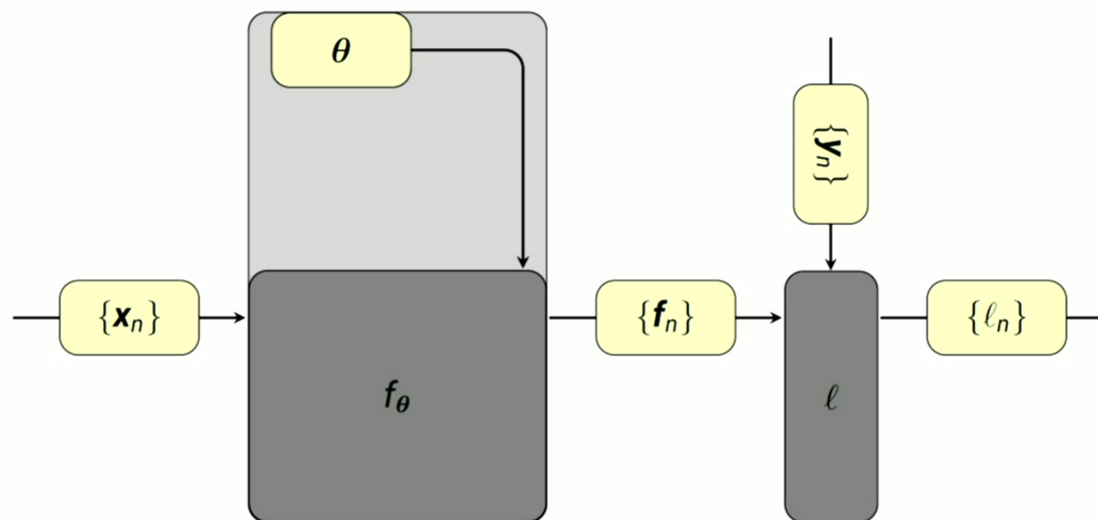
Tensor Network



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

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

Supervised Deep Learning in a Nutshell (Math)



Minimize $\mathcal{L}_{\mathbb{D}}(\boldsymbol{\theta})$ with

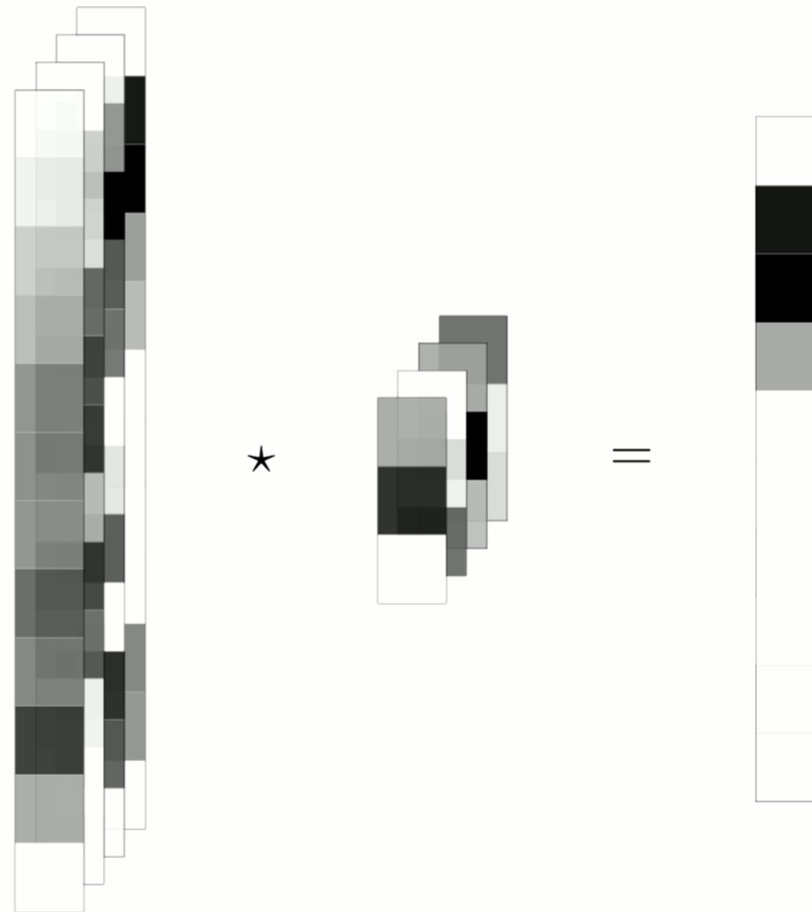
$$\mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathbb{B}|} \sum_{(\boldsymbol{x}_n, \boldsymbol{y}_n) \in \mathbb{B}} \ell(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_n), \boldsymbol{y}_n) \qquad \text{where} \qquad \mathbb{B} \sim \mathbb{D},$$

1

C_{in} is for Input Channels

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

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

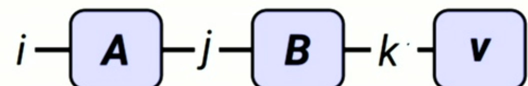


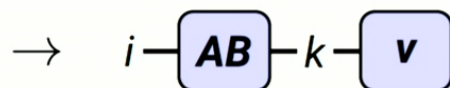
Tensor Multiplication & Tensor Networks

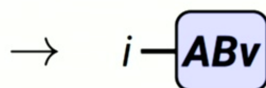
Contraction Path Optimization (e.g. `opt_einsum` [Smith and Gray, 2018])

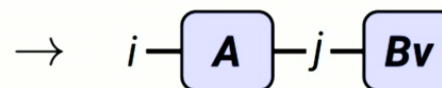
[Example] Matrix-matrix-vector-product

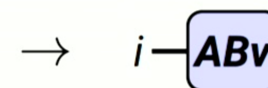
Consider $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{3.000 \times 3.000}$ and $\boldsymbol{v} \in \mathbb{R}^{3.000}$.



→ 

→ 

→ 

→ 

Schedule 1: **1.1 s**

```
A @ B @ v
```

Schedule 2: **2.2 ms**

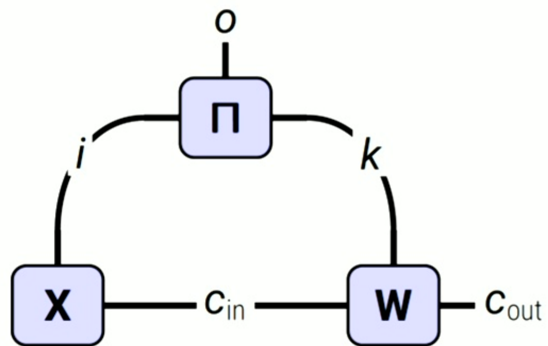
```
A @ (B @ v)
```

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.

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



1d

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

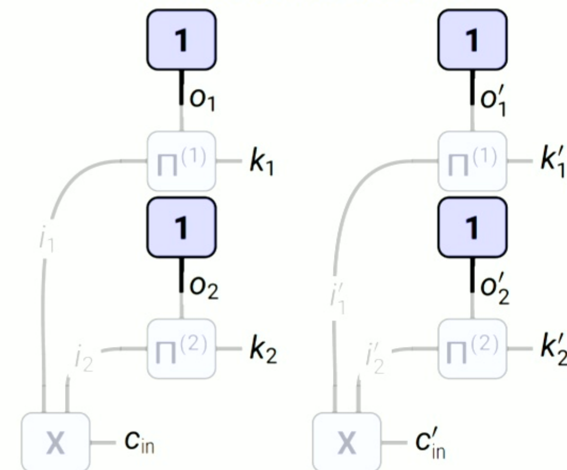
Non-conventional operations can be much cheaper with tensor networks

State of the art

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

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

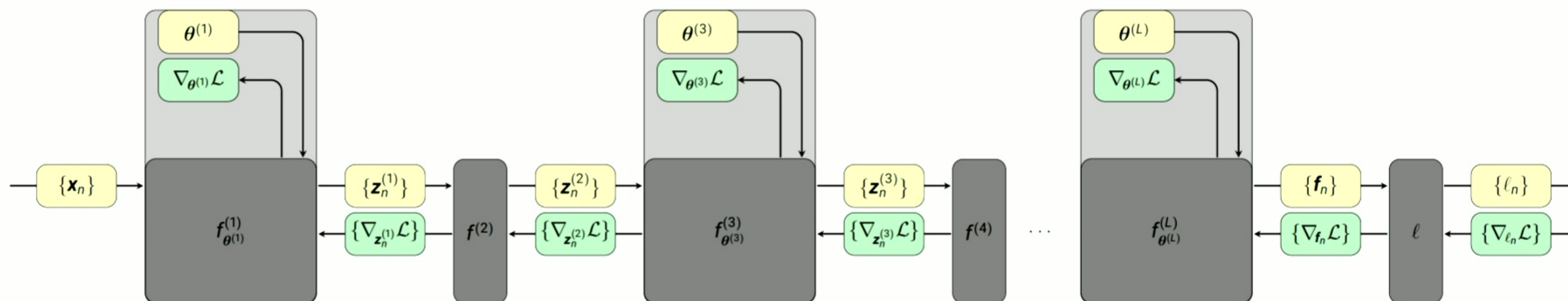
Tensor Network



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)

Supervised Deep Learning in a Nutshell (Math)



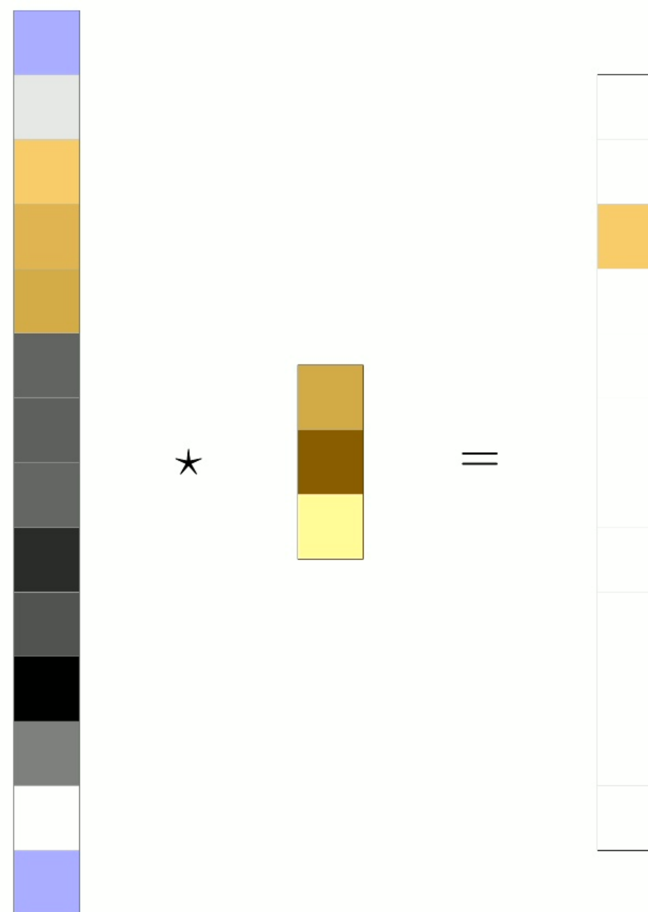
Minimize $\mathcal{L}_{\mathbb{D}}(\boldsymbol{\theta})$ with

$$\mathcal{L}_{\mathbb{B}}(\boldsymbol{\theta}) = \frac{1}{|\mathbb{B}|} \sum_{(\boldsymbol{x}_n, \boldsymbol{y}_n) \in \mathbb{B}} \ell(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}_n), \boldsymbol{y}_n) \qquad \text{where} \qquad \begin{aligned} &\mathbb{B} \sim \mathbb{D}\,, \\ &\boldsymbol{f}_{\boldsymbol{\theta}} = \boldsymbol{f}^{(L)}_{\boldsymbol{\theta}^{(L)}} \circ \boldsymbol{f}^{(L-1)}_{\boldsymbol{\theta}^{(L-1)}} \circ \ldots \circ \boldsymbol{f}^{(1)}_{\boldsymbol{\theta}^{(1)}}\,. \end{aligned}$$

P is for Padding

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

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



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	$\boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{B}$	$C_{(i,k),(j,l)} = A_{i,j} B_{k,l}$

Function Transformations with Tensor Networks

[Example] Batching/vmap-ing: adding legs

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

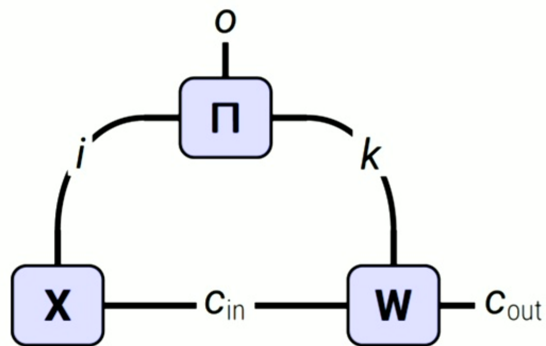
i — A — j — B — k — v → (with shared leg n attached to A, B, v) i — A — j — B — k — v

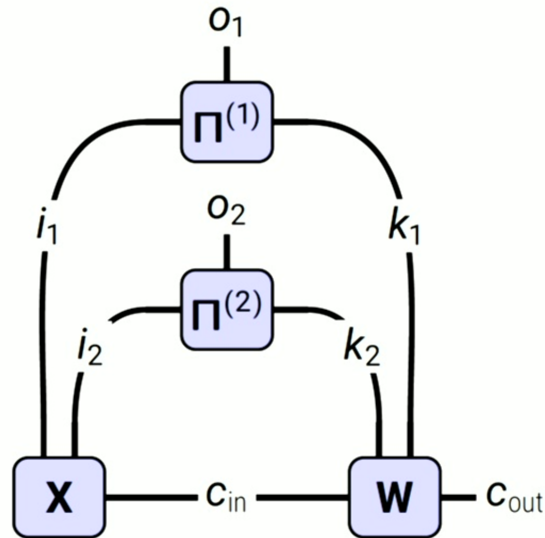
[Example] Differentiation: removing arguments

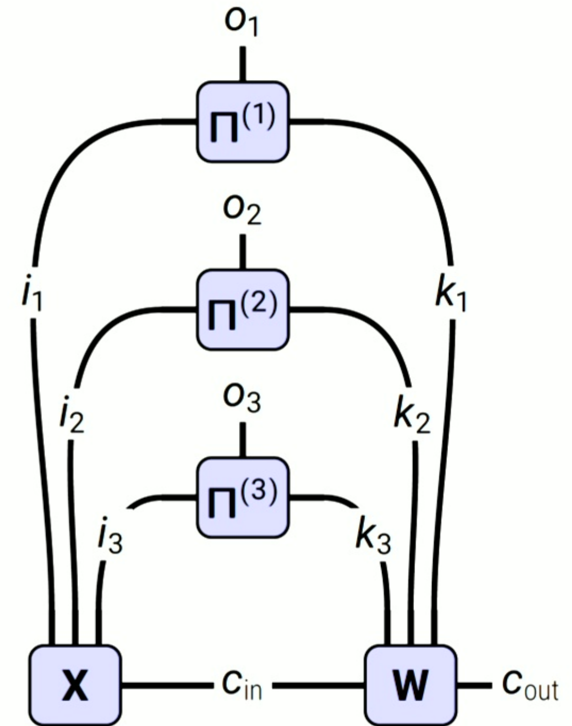
$$i - \frac{\partial(\boldsymbol{ABv})}{\partial \boldsymbol{B}} \begin{matrix} -j' \\ -k' \end{matrix} = \frac{\partial\left(i - \boldsymbol{A} - j - \boldsymbol{B} - k - \boldsymbol{v}\right)}{\partial\left(j' - \boldsymbol{B} - k'\right)} = i - \boldsymbol{A} - j' \qquad k' - \boldsymbol{v}$$

$$i - \frac{\partial(\boldsymbol{ABv})}{\partial \boldsymbol{A}} \begin{matrix} -i' \\ -j' \end{matrix} = \frac{\partial\left(i - \boldsymbol{I} - l - \boldsymbol{A} - j - \boldsymbol{Bv}\right)}{\partial\left(i' - \boldsymbol{A} - j'\right)} = i - \boldsymbol{I} - i' \qquad j' - \boldsymbol{Bv}$$

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







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

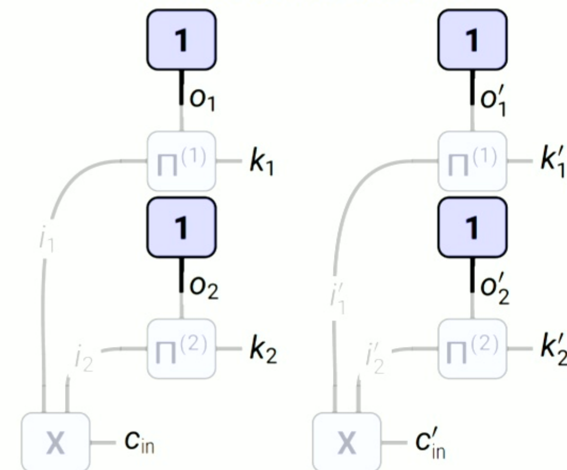
Non-conventional operations can be much cheaper with tensor networks

State of the art

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

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)

Goal: Make Richer Information More Accessible

Cockpit

$\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$ — statistical info → $\{\nabla_{\boldsymbol{\theta}}\ell_n(\boldsymbol{\theta})\}$

Hessian backprop
geometrical info

BackPACK

ViViT

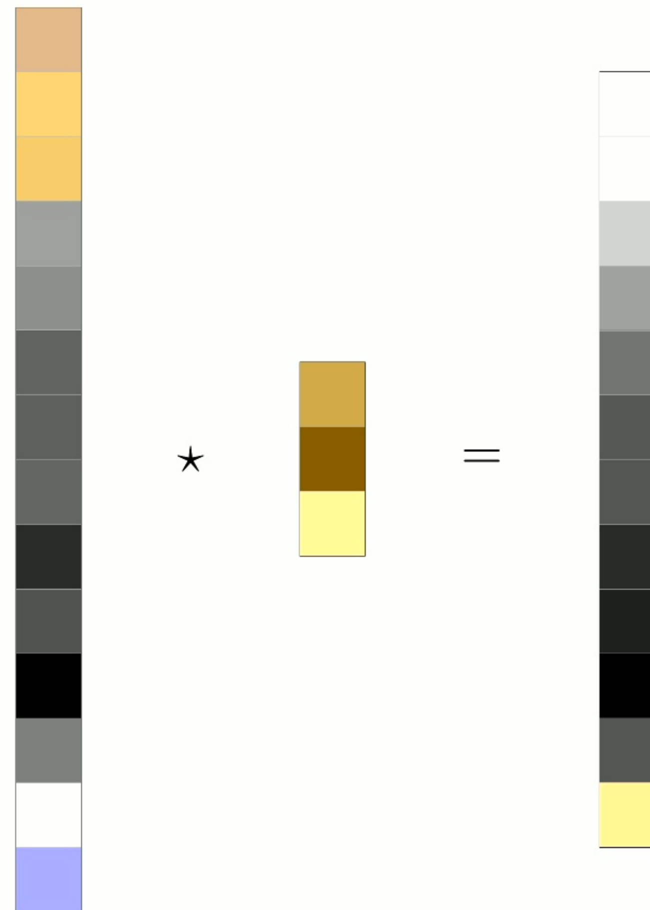
$\nabla^2_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$ → $\{\nabla^2_{\boldsymbol{\theta}}\ell_n(\boldsymbol{\theta})\}$

2

P is for Padding

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

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



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

`einsum` is Probably All You Need

Recommendation: Use `einsum` in your code!

- ☺ Better readability (e.g. `einops` [Rogozhnikov, 2022])
- ☺ Automatic optimization (e.g. `opt_einsum` [Smith and Gray, 2018])

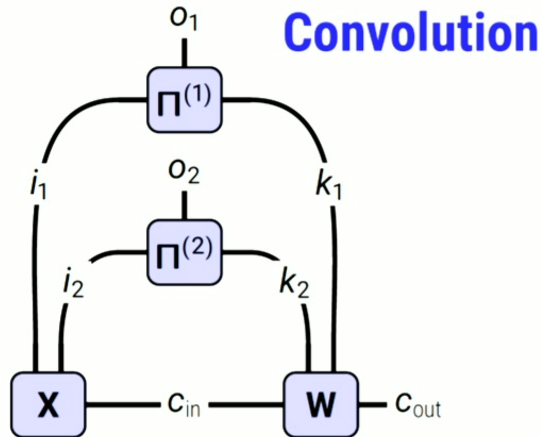
Also (not discussed)

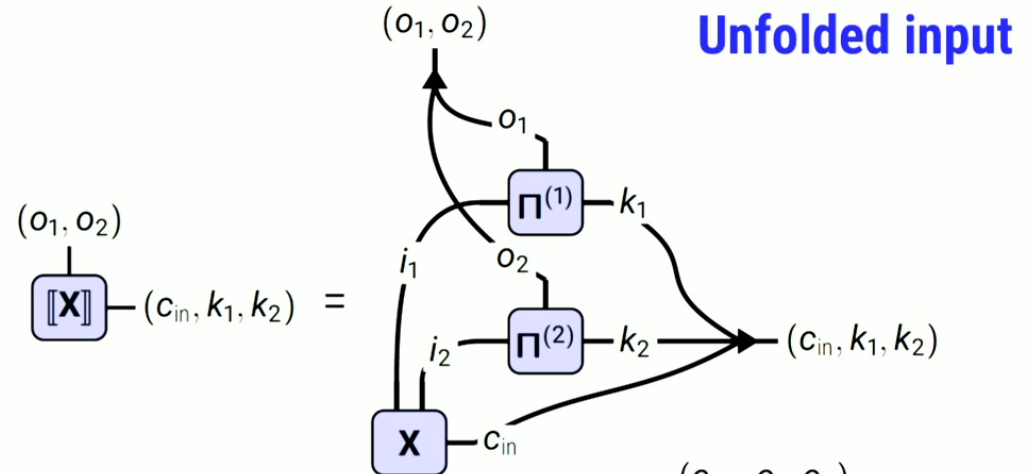
- ☺ Automatic distribution (e.g. `cotengra` [Gray and Kourtis, 2021])
- ☺ Randomized/Approximate evaluation (somebody should do this!)

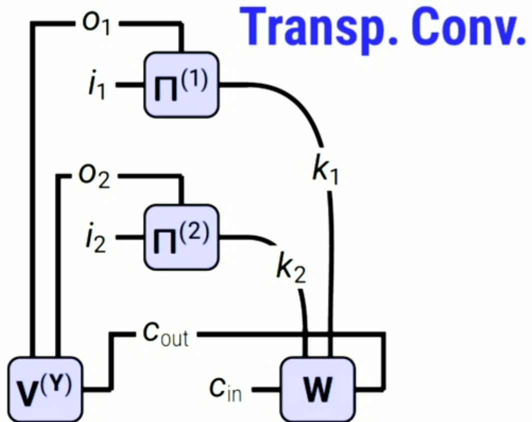
Also (regarding tensor networks)

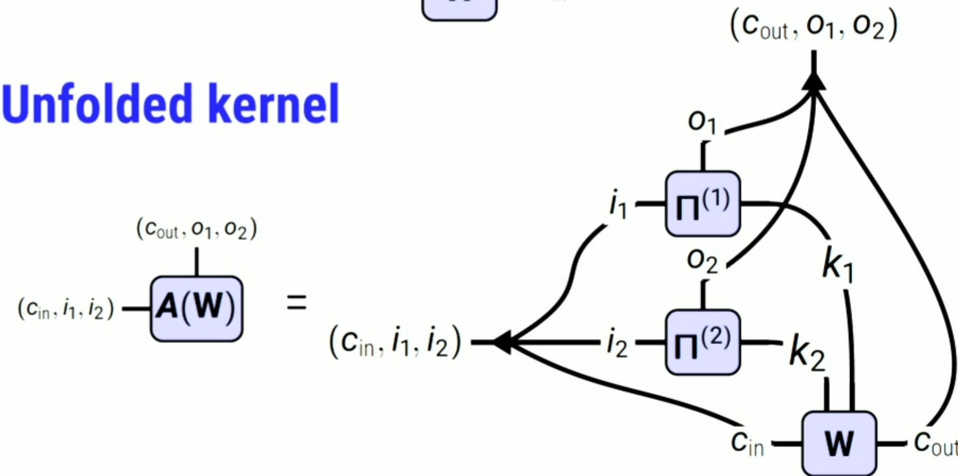
- ☺ Drawing diagrams is more fun
- ☺ Simplifications/Transformations via graphical manipulations

Connection to Matrix-multiplication Perspective









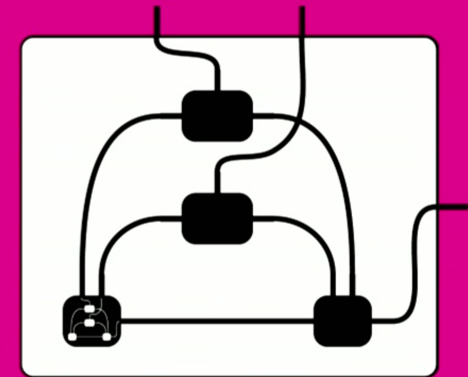
30

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!

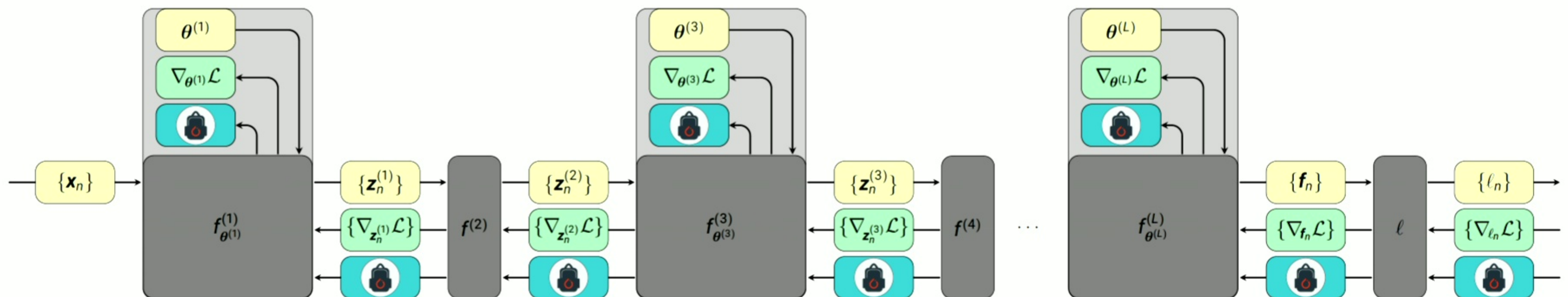
```
from einconv.expressions import kfac_reduce
from torch import einsum

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



`pip install einconv`

Customized Backpropagation

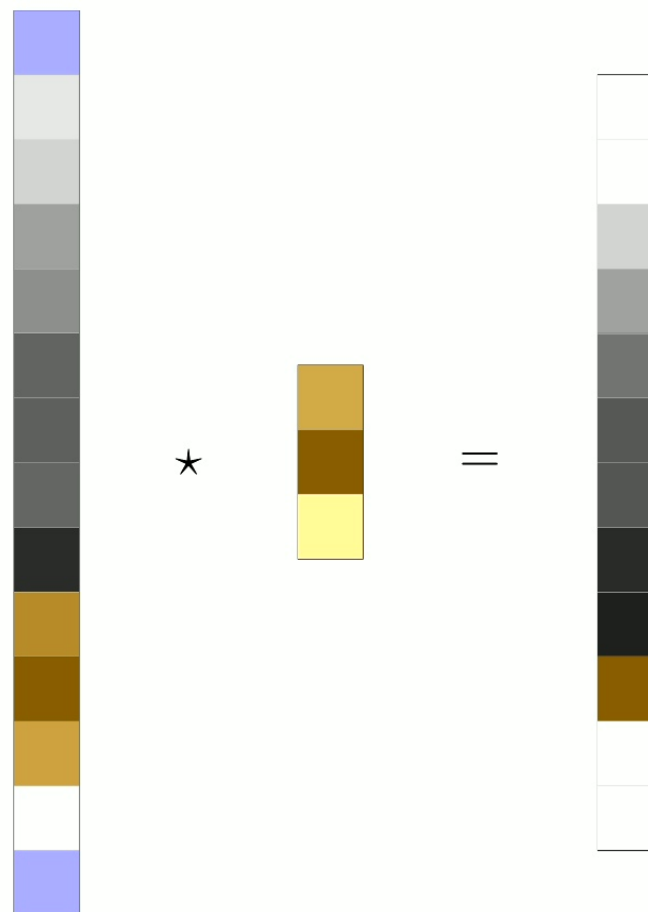
1. What to backpropagate
2. How to backpropagate
3. How to extract target quantity



P is for Padding

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

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



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

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

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

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 $\boldsymbol{C} = \boldsymbol{AB}$ as tensor multiplication

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

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

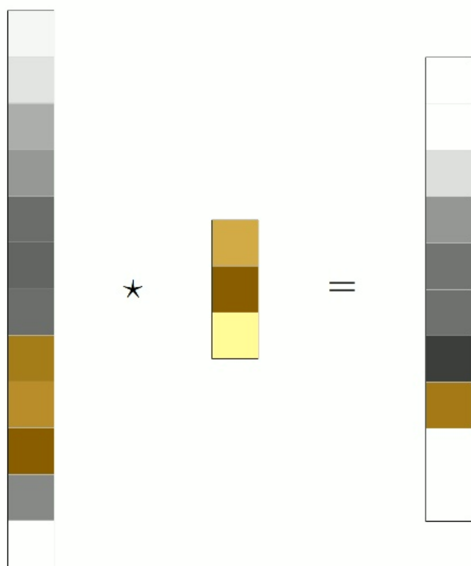
Many libraries provide $*_{(\dots)}(\dots)$ as `einsum`.

Convolution as Matrix Multiplication

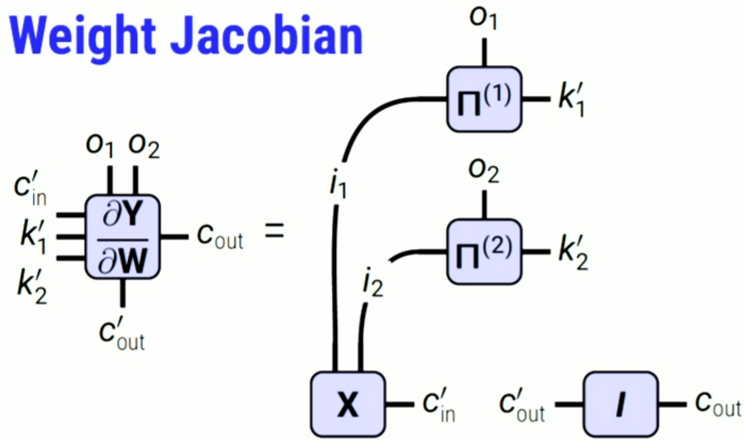
$$\mathbf{X} \in \mathbb{R}^{C_{\text{in}} \times I}$$
$$\mathbf{Y} \in \mathbb{R}^{C_{\text{out}} \times O}$$
$$\mathbf{W} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times K}$$

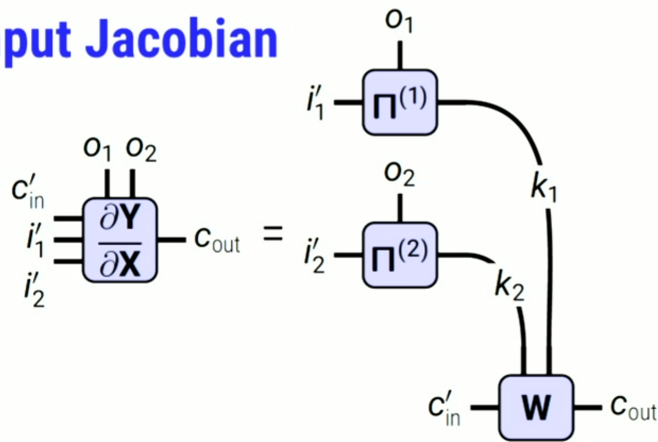
$$\mathbf{Y} = \mathbf{X} \star \mathbf{W}$$

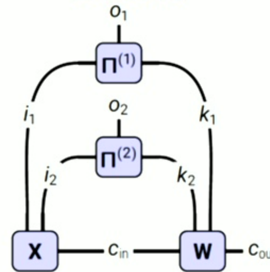
$$\boldsymbol{Y} = \boldsymbol{W}[\![\boldsymbol{X}]\!]$$



Derivatives & Autodiff Routines





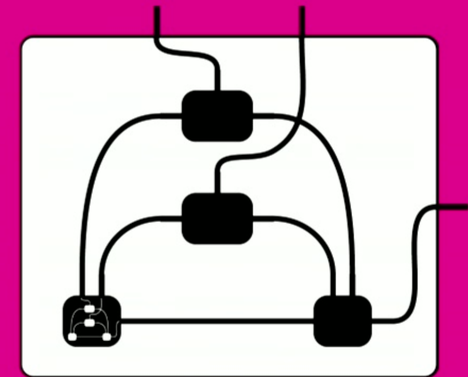


Convolutions and More as `einsum`

- 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?