ML@scale using randomized linear algebra

Felix J. Herrmann

Microsoft Azure/Research, March 18

SLIM
Georgia Institute of Technology

Released to public domain under Creative Commons license type BY (https://creativecommons.org/licenses/by/4.0)
Copyright (c) 2021, Felix J. Herrmann (Georgia Tech)

ML@scale using randomized linear algebra

Mathias Louboutin & Ali Siahkoohi

Microsoft Azure/Research, March 18

SLIM
Georgia Institute of Technology

ML4Seismic

Research presented today is carried out as part of ML4Seismic

- industry partners program for Machine Learning in Seismic
- jointly w/ Dr. Ghassan AlRegib from ECE
- supported by Microsoft Azure & Equinor

Also part of campus-wide initiatives on Carbon Capture Sequestration (CCS)

Today's talk is our response to Philippe Witte's encouragement:

"To think out of the box and leverage that crazy seismic stuff…"

Research goals

Engineer tangible AI-based solutions to combat climate change by

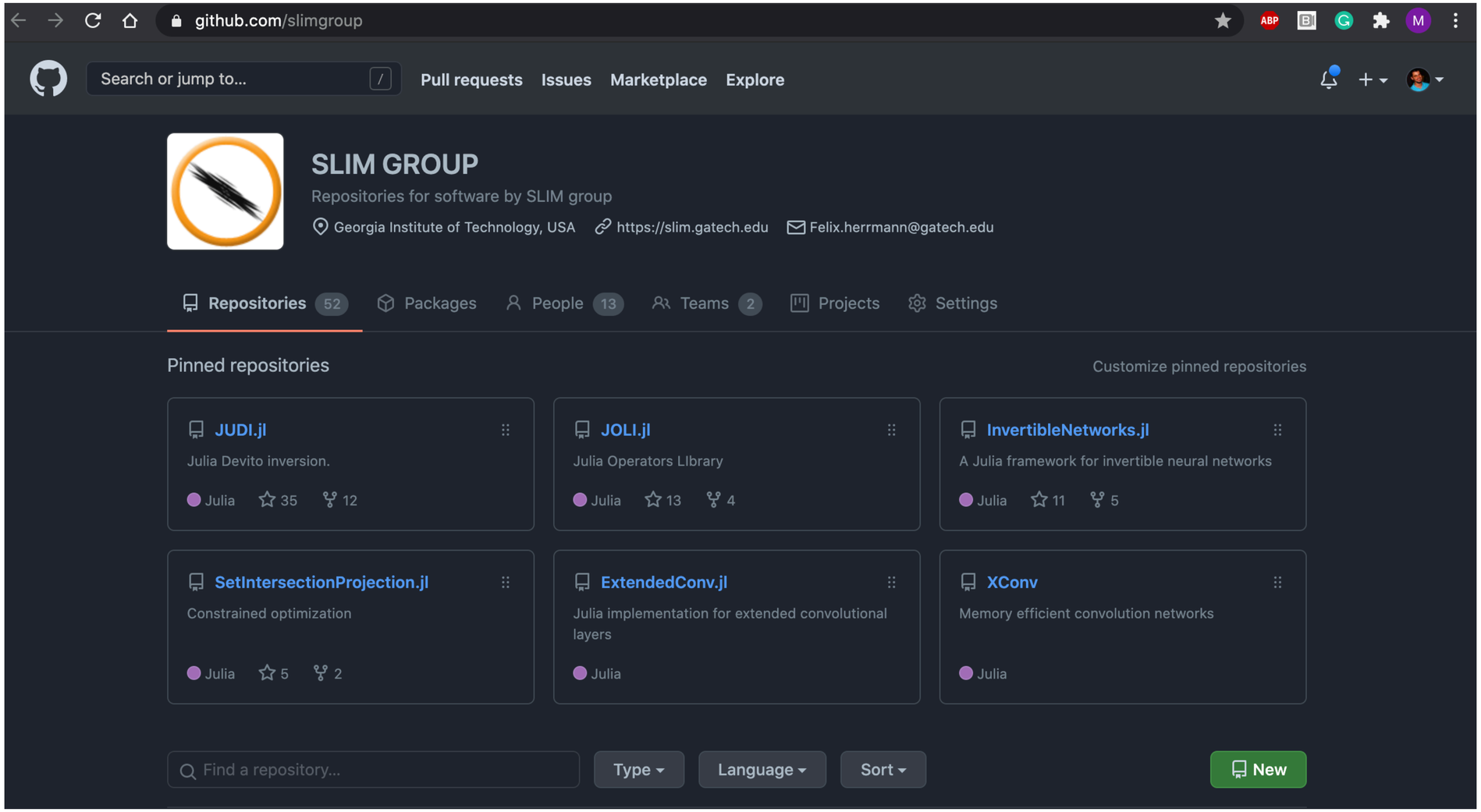
- making training of CNNs & solutions of PDEs computationally more efficient
- maximizing performance of CCS while reducing cost & risk
- accelerating innovations via open source for AI, seismic imaging & monitoring

Create solutions that balance fundamental understanding of

- a domain (e.g. seismic monitoring of CCS or medical ultrasound)
- latest developments in computer science (ML & randomized linear algebra)

ML@Scale: maximize impact of AI via sustainable scale up!

Open source


github.com/slimgroup
Search or jump to...
Pull requests
Issues
Marketplace
Explore
SLIM GROUP
Repositories for software by SLIM group
Georgia Institute of Technology, USA
https://slim.gatech.edu
Felix.herrmann@gatech.edu
Repositories 52
Packages
People 13
Teams 2
Projects
Settings
Pinned repositories
Customize pinned repositories
JUDI.jl
Julia Devito inversion.
Julia 35 12
JOLI.jl
Julia Operators LIbrary
Julia 13 4
InvertibleNetworks.jl
A Julia framework for invertible neural networks
Julia 11 5
SetIntersectionProjection.jl
Constrained optimization
Julia 5 2
ExtendedConv.jl
Julia implementation for extended convolutional layers
Julia
XConv
Memory efficient convolution networks
Julia
Find a repository...
Type
Language
Sort
New

AI challenge

Our successes hinge on deeper NNs & larger training sets
(e.g. video encoding, 3D seismic segmentation, etc.)

Unsustainable demands on energy to train DNNs
(e.g. produces 2 Mt of CO2 to train imagenet & 284 Mt to train transformer)

Impeded rate of innovation
(lack of access to compute & training data & environmental impact)

AI challenge

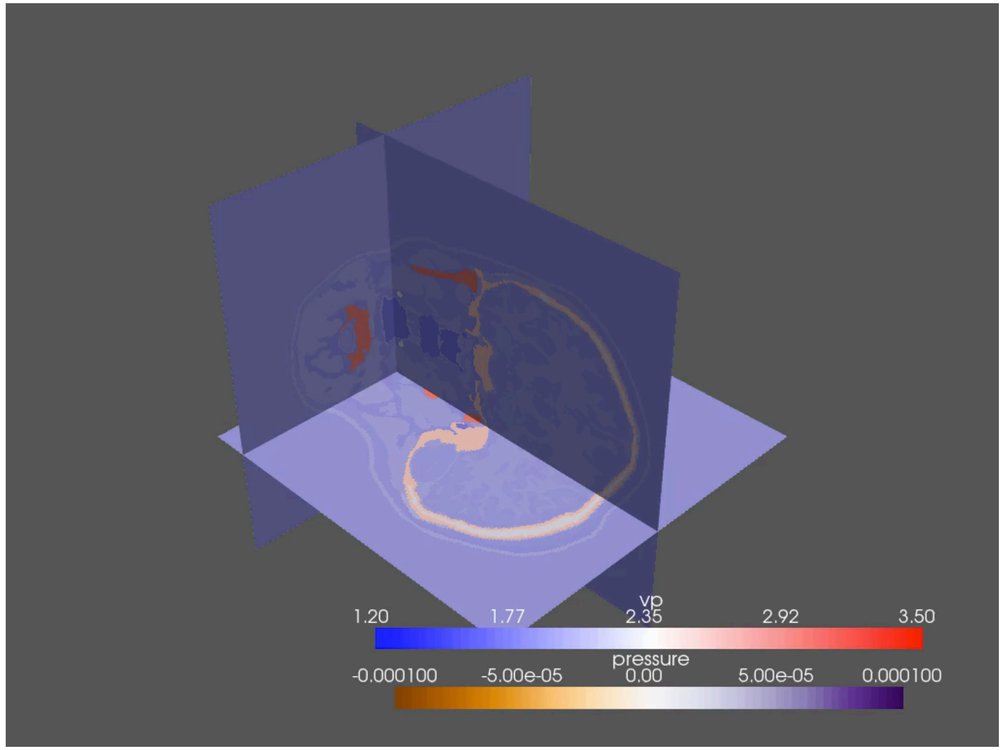
Our successes hinge on deeper & deeper NNs & larger training sets
(e.g. video encoding, 3D seismic segmentation, etc.)

Unsustainable demands on energy to train DNNs
(e.g. produces 2 Mt of CO2 to train imagenet & 284 Mt to train transformer)

Impedes rate of innovation
(lack of access to compute & training data & environmental impact)

Experience

- $10\times$ **cost & environmental imprint reduction for seismic acquisition**
- $7^{+}\times$ **comp. cost reduction wave imaging**
- train problems w/ $\mathcal{O}(10^4)$ time-steps & $\mathcal{O}(10^9)$ voxels during wave imaging
- just-in-time compiler technology
- leverage randomized algorithms
- deal w/ bad local minima



Today's solutions

1. Use randomized trace estimation to reduce the **memory imprint** of training CNNs & of solving PDEs

2. Employ just-in-time compiler technology for **stencil computations** to improve FLOP performance & memory use

3. Extend the search space to enhance training of problems hampered by parasitic **local minima**

Leverage connections between DNNs & PDE-based inversions…

Memory imprint CNNs

```
scripts @ eas-coda-fherr07 [mlouboutin3](master)$ PYTORCH_NO_CUDA_MEMORY_CACHING=1 python3 mem_prof.py 1 &&
PYTORCH_NO_CUDA_MEMORY_CACHING=1 python3 mem_prof.py 2;
1 True
Network Sequential(
  (0): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (1): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (2): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (3): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
Input size torch.Size([128, 16, 256, 256])
GPU usage probe before forward: mem: 26.951%, abs-mem: 1.0595703125 (GiB) # nothing done
GPU usage probe after forward: mem: 27.000%, abs-mem: 1.0615234375 (GiB) # after Y = N(X).mean()
GPU usage probe after backward: mem: 27.000%, abs-mem: 1.0615234375 (GiB) # after Y.backward
Network Sequential(
  (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
Input size torch.Size([128, 16, 256, 256])
GPU usage true before forward: mem: 26.951%, abs-mem: 1.0595703125 (GiB) # nothing done
GPU usage true after forward: mem: 65.154%, abs-mem: 2.5615234375 (GiB) # after Y = N(X).mean()
GPU usage true after backward: mem: 27.000%, abs-mem: 1.0615234375 (GiB) # after Y.backward
```

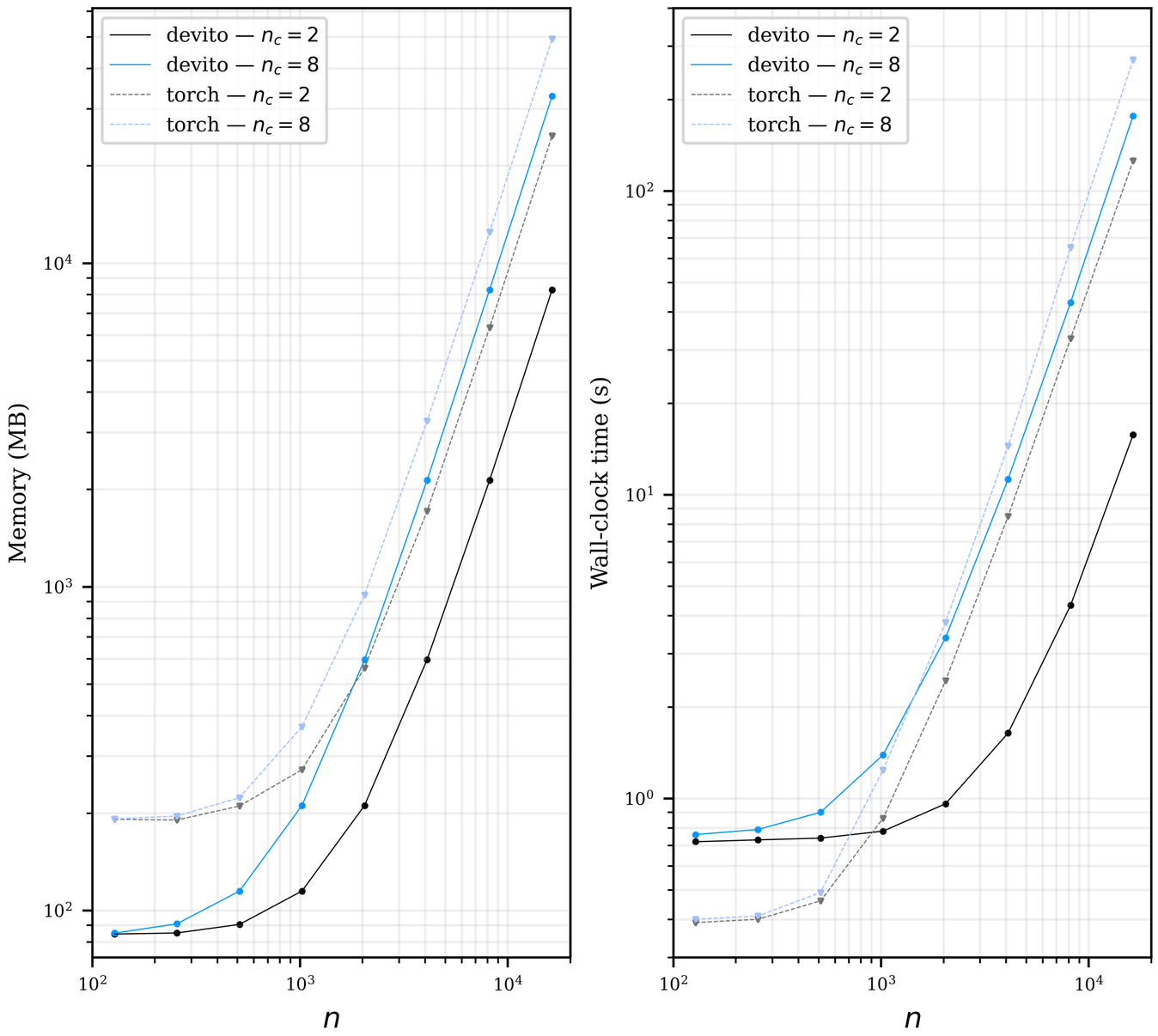
2Mb memory

1.5Gb memory

Orders of magnitude reduction in memory usage…

Stencil-based CNN computations

- $1.5 - 2\times$ **reduction in memory**
- $1.5 - 10\times$ **speed improvement on CPUs**



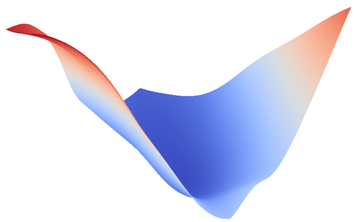
JIT can lead to significant improvements FLOPs & memory use…

Hao Li et al., Visualizing the Loss Landscape of Neural Nets, Advances in Neural Information Processing Systems, 2018.

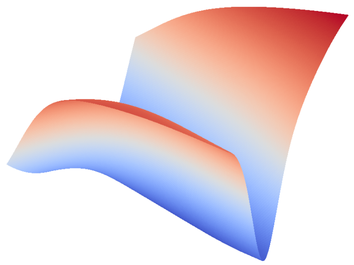
Extended loss function landscape

highly non-convex

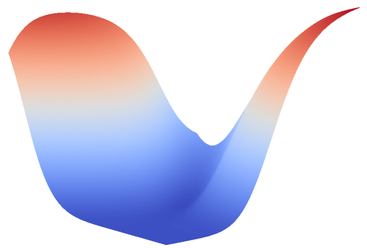
Conventional loss landscape



Conventional loss landscape



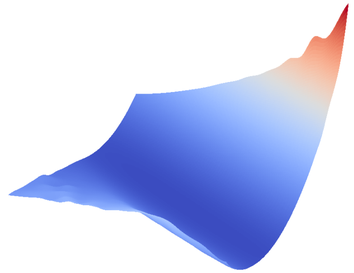
Conventional loss landscape



tentatively quadratic

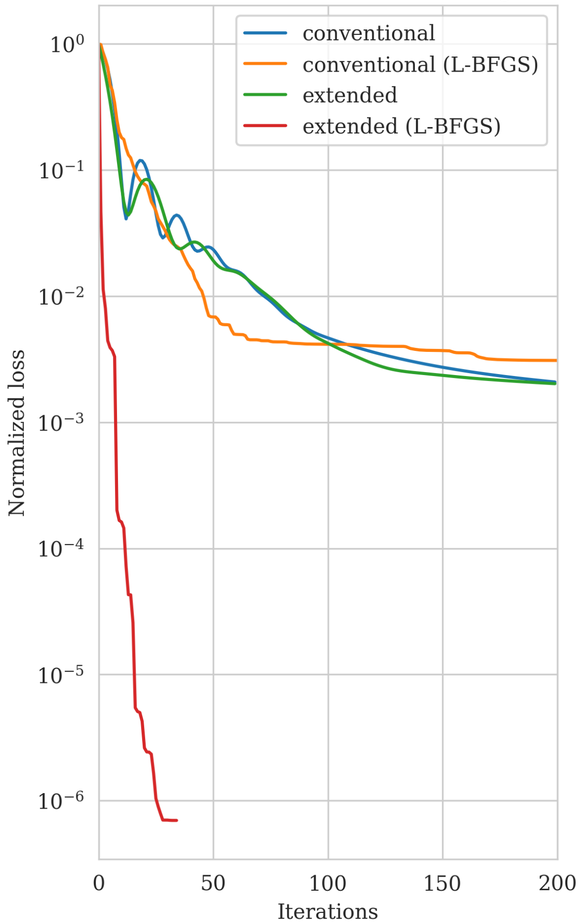
Extended loss landscape

Extended loss landscape



Extended loss landscape

Extended vs conventional convergence



Oktay, Deniz, et al. "Randomized automatic differentiation." *arXiv preprint arXiv:2007.10412* (2020)
Hewett, Russell J., and Thomas J. Grady II. "A Linear Algebraic Approach to Model Parallelism in Deep Learning." arXiv preprint arXiv:2006.03108 (2020)

Challenge – memory imprint

Training of DNNs scales poorly to large problem sizes

- poor memory handling $\Longrightarrow$ large expensive power hungry GPUs
- suboptimal patches $\Longrightarrow$ miss long range correlations
- different architectures (e.g. invertible NNs) $\Longrightarrow$ not mainstream
- complex HPC parallelization
- pruning or **randomized automatic differentiation**

Additional complexities stifle innovations in

- video encoding/labeling/segmentation
- training increasingly deep NNs on large images, video, etc.

Solution: exploit randomized trace estimation during back-propagation…

Oktay, Deniz, et al. "Randomized automatic differentiation." *arXiv preprint arXiv:2007.10412* (2020)

Observations

Many modern DNNs

- rely heavily on CNNs implemented w/ dense linear algebra (BLAS/MKL/CUDNN etc.)
- spend most time in CNNs during training

Training w/ back-propagation complicated by

- need to store the "state variable" at each layer
- dense matrix operations between input & output CNNs for gradients

Question: Are accurate gradients needed during training w/ SGD?

Linear algebra

Singular Value Decomposition:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^\top \quad \text{with} \quad \left[\mathbf{U}, \mathbf{S}, \mathbf{V}\right] = \mathrm{svd}(\mathbf{A})$$

Trace estimation:

$$\mathrm{tr}(\mathbf{A}) = \sum_i \mathbf{A}_{ii} = \sum \overleftarrow{\mathrm{diag}}(\mathbf{A})$$

Do not scale well because they

- need access to the matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ for N very large
- rely on "exact" arithmetic

Halko, Nathan, Per-Gunnar Martinsson, and Joel A. Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions." *SIAM review* 53.2 (2011): 217-288.

Randomized linear algebra

Randomized SVD:

$$\mathbf{A} \approx \mathbf{U}\mathbf{S}\mathbf{V}^\top \quad \text{with} \quad \begin{cases} \left[\mathbf{Q}, \sim\right] & = \mathrm{qr}(\mathbf{A}\mathbf{Z}) \\ \left[\widetilde{\mathbf{U}}, \mathbf{S}, \mathbf{V}\right] & = \mathrm{svd}(\mathbf{Q}^\top \mathbf{A}) \\ \mathbf{U} = \mathbf{Q}\widetilde{\mathbf{U}} & \end{cases}$$

- information is reaped during probing $\mathbf{A}\mathbf{Z}$ w/ $\mathbf{Z} = [\mathbf{z}_1, \cdots, \mathbf{z}_r]$ random
- only need access to action of $\mathbf{A}$ (in parallel)
- approximation w/ accuracy $\propto r \ll N$, # of sketches w/ random vectors $\mathbf{z}_i$

Scales extremely well w/ controllable accuracy…

Hutchinson, Michael F. "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines." *Communications in Statistics-Simulation and Computation* 18.3 (1989): 1059-1076.
Avron, Haim, and Sivan Toledo. "Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix." Journal of the ACM (JACM) 58.2 (2011): 1-34.
Meyer, Raphael A., et al. "Hutch++: Optimal Stochastic Trace Estimation." Symposium on Simplicity in Algorithms (SOSA). Society for Industrial and Applied Mathematics, 2021.

Randomized linear algebra

Randomized Trace Estimation:

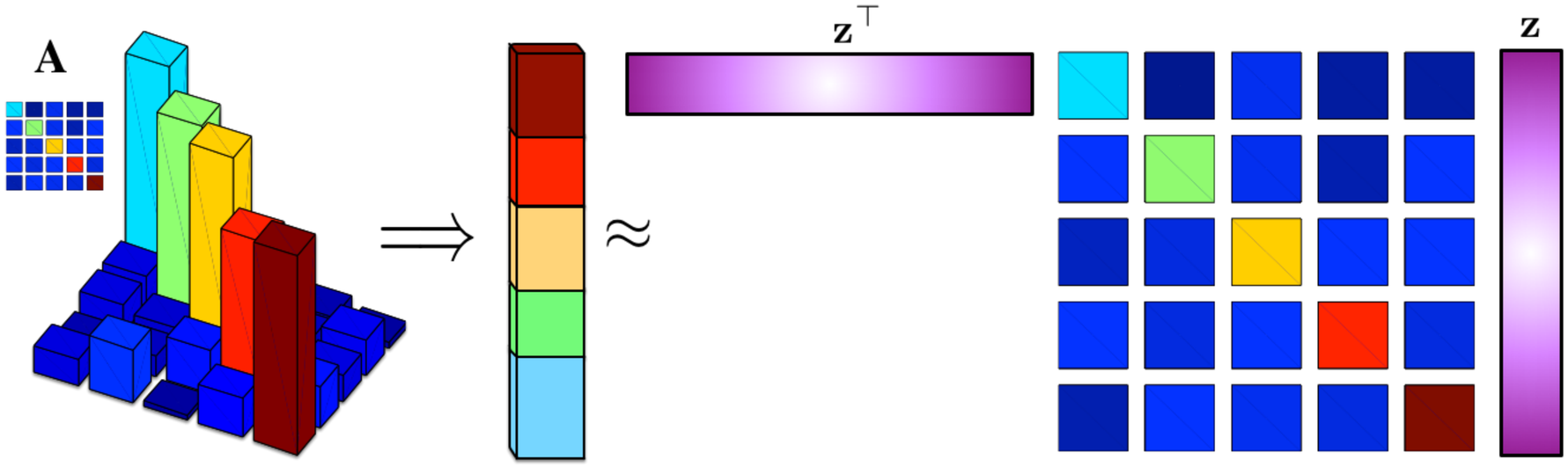
$$\mathrm{tr}(\mathbf{A}) \approx \frac{1}{r}\sum_{j=1}^{r} \mathbf{z}_j^\top \mathbf{A}\mathbf{z}_j = \frac{1}{r}\,\mathrm{tr}(\mathbf{Z}^\top \mathbf{A}\mathbf{Z})$$

- only needs matrix free access to actions of $\mathbf{A}$
- unbiased estimator when $\mathbb{E}(\mathbf{z}\mathbf{z}^\top) = \mathbf{I}$ w/ accuracy $\propto r \ll N$, # of sketches w/ random vectors $\mathbf{z}_j$
- errors studied & understood

Why should we care?

Hutchinson's Trick, http://blog.shakirm.com/2015/09/machine-learning-trick-of-the-day-3-hutchinsons-trick/

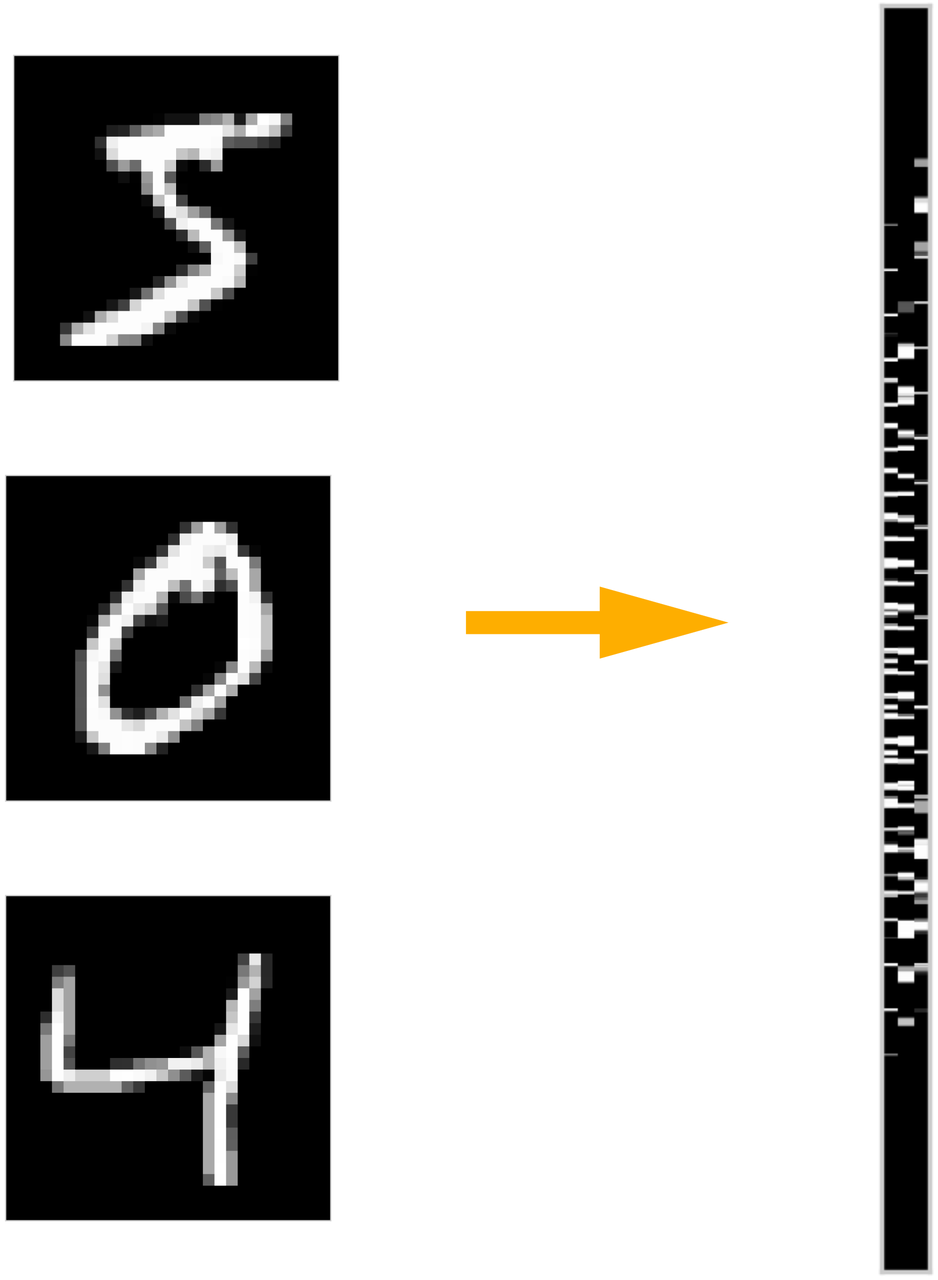
Random Trace Estimation



$$\mathrm{tr}(\mathbf{A}) \approx \frac{1}{r}\sum_{j=1}^{r} \mathbf{z}_j^\top \mathbf{A}\mathbf{z}_j = \frac{1}{r}\,\mathrm{tr}(\mathbf{Z}^\top \mathbf{A}\mathbf{Z})$$

Vectorize

$\mathbb{R}^{n_x \times n_y}$



$\mathbf{X} \in \mathbb{R}^{N \times b}$

b batch size

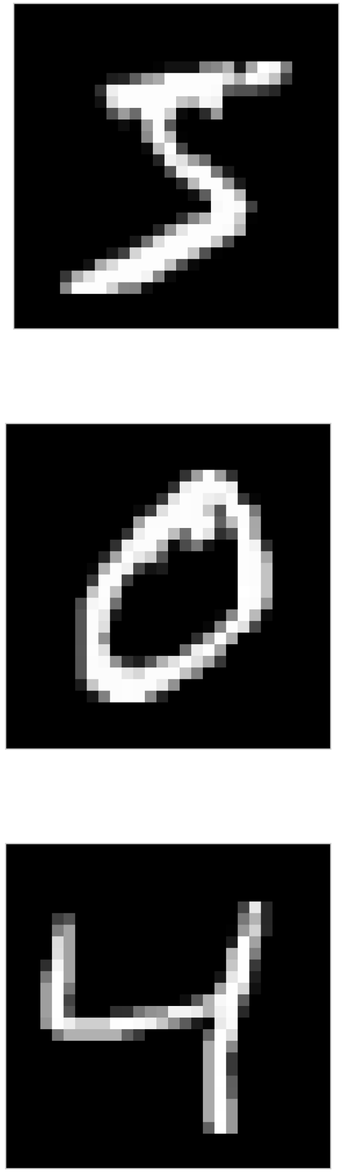
N image size

$N = n_x \times n_y$

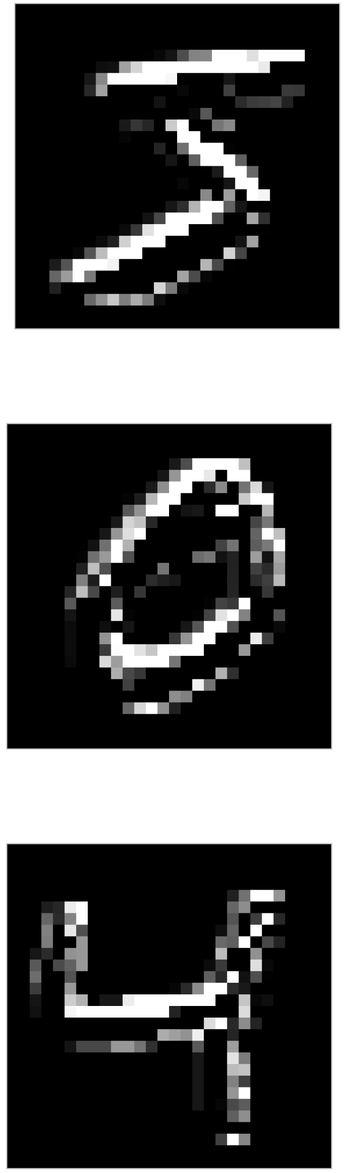
CNN

$\mathbf{w} \in \mathbb{R}^{n_w}$, $n_w = 9$

$\star$



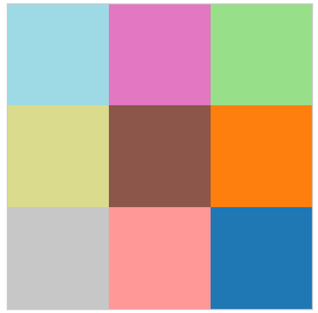
$=$



$\mathrm{conv}(\mathbf{x}; \mathbf{w})$

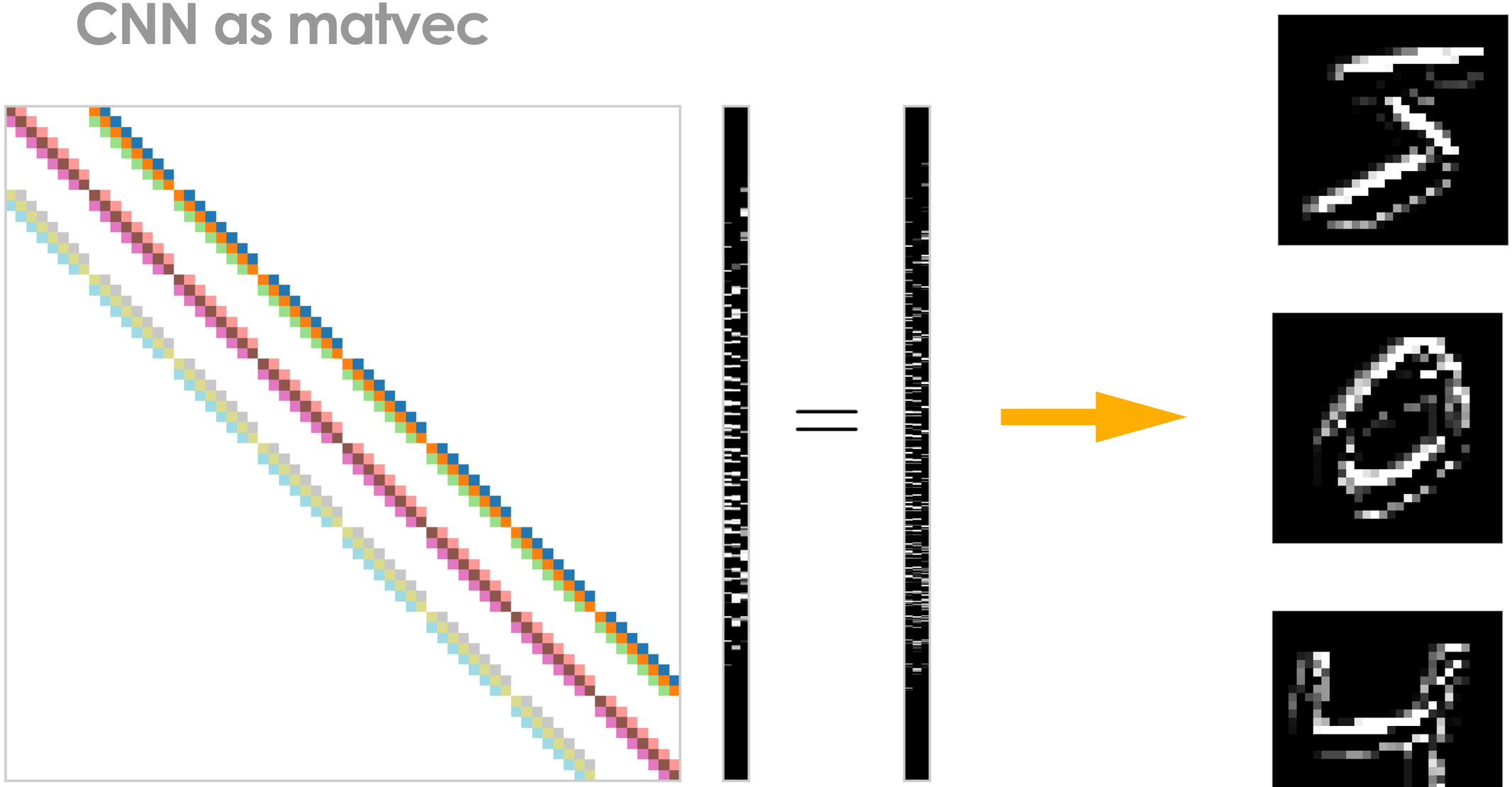
Matricize

$$\mathbf{w} \in \mathbb{R}^{n_w},\ n_w = 9$$



$$\mathbf{W} \in \mathbb{R}^{N \times N}$$

CNN as matvec



$\mathbf{wx}$

Algorithm

Forward pass

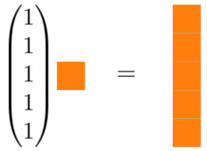
Data: Convolution input $\mathbf{X}$ and weights $\mathbf{w}$
Result: Convolution
begin
  Draw random seed s
  $\overline{\mathbf{X}} = \mathbf{Z}(s)^\top \mathbf{X}$
  $\mathbf{Y} = \mathrm{conv}(\mathbf{X}; \mathbf{w})$
  Store $\overline{\mathbf{X}}, s$
end

Backward pass

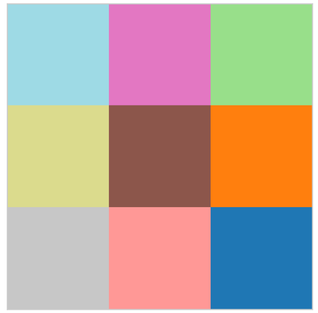
Data: Back-propagated residual $\delta\mathbf{Y}$
Result: Gradient w.r.t to weights
begin
  Load random seed s and probed forward $\overline{\mathbf{X}}$
  $\overline{\mathbf{Y}}^\top = \delta\mathbf{Y}^\top \mathbf{T}_{-k(i)} \mathbf{Z}(s)$
  $\delta\mathbf{w} = \mathrm{tr}(\overline{\mathbf{X}}\,\overline{\mathbf{Y}}^\top)$
end

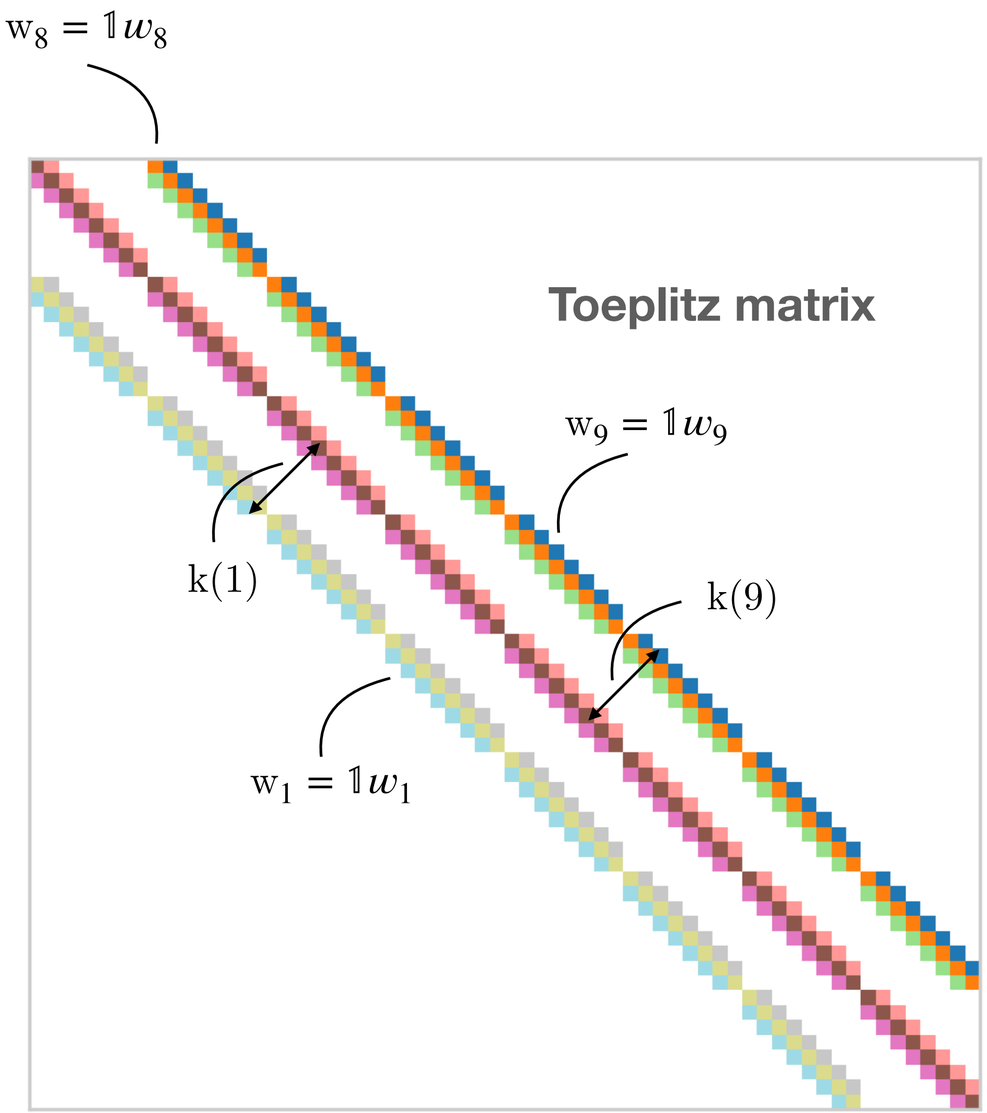
$$\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{N \times b} \Longrightarrow \overline{\mathbf{X}}, \overline{\mathbf{Y}} \in \mathbb{R}^{r \times b},\ r \ll N = n_x \times n_y$$

Matricize



$$\mathbf{w} \in \mathbb{R}^{n_w},\ n_w = 9$$





$$\mathbf{W} = \left(\sum_{i=1}^{n_w} \overrightarrow{\mathrm{diag}}(\mathbf{w}_i) \mathbf{T}_{k(i)} \right)$$

Trace in CNNs?

Gradient w.r.t all weights

$$\frac{\partial}{\partial \mathbf{W}} f(\mathbf{W}\mathbf{X}) = \delta \mathbf{Y}\mathbf{X}^\top$$

"Toeplitz" structure

$$\mathbf{W} = \sum_{i=1}^{n_w} \overrightarrow{\mathrm{diag}}(\mathbf{w}_i) T_{k(i)} = \sum_{i=1}^{n_w} \overrightarrow{\mathrm{diag}}(w_i \mathbf{1}) T_{k(i)}$$

Gradient w.r.t. i^{th} weight conv layer

$$\begin{aligned}
\frac{\partial}{\partial w_i} f(\mathbf{W}\mathbf{X}) &= \mathrm{tr}\left(\left(\frac{\partial f(\mathbf{W}\mathbf{X})}{\partial \mathbf{W}}\right)^\top \frac{\partial \mathbf{W}}{\partial w_i}\right) \\
&= \mathrm{tr}\left(\left(\delta \mathbf{Y}\mathbf{X}^\top\right)^\top \mathbf{T}_{k(i)}^\top\right) \\
&= \mathrm{tr}\left(\mathbf{X}\delta\mathbf{Y}^\top \mathbf{T}_{-k(i)}\right)
\end{aligned}$$

Hutchinson, Michael F. "A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines." *Communications in Statistics-Simulation and Computation* 18.3 (1989): 1059-1076.
Avron, Haim, and Sivan Toledo. "Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix." Journal of the ACM (JACM) 58.2 (2011): 1-34.
Meyer, Raphael A., et al. "Hutch++: Optimal Stochastic Trace Estimation." Symposium on Simplicity in Algorithms (SOSA). Society for Industrial and Applied Mathematics, 2021.

Randomized trace estimation

Based on stochastic approximation of the identity $\mathbf{I}$

$$
\begin{aligned}
\mathrm{tr}(\mathbf{A}) = \mathrm{tr}\left(\mathbf{A}\mathbf{I}\right) &= \mathrm{tr}\left(\mathbf{A}\mathbb{E}\left[\mathbf{z}\mathbf{z}^\top\right]\right) \\
&= \mathbb{E}\left[\mathrm{tr}\left(\mathbf{A}\mathbf{z}\mathbf{z}^\top\right)\right] \\
&= \mathbb{E}\left[\mathbf{z}^\top\mathbf{A}\mathbf{z}\right] \\
&\approx \frac{1}{r}\sum_{i=1}^{r}\left[\mathbf{z}_i^\top\mathbf{A}\mathbf{z}_i\right] \\
&= \frac{1}{r}\,\mathrm{tr}\left(\mathbf{Z}^\top\mathbf{A}\mathbf{Z}\right)
\end{aligned}
$$

http://blog.shakirm.com/2015/09/machine-learning-trick-of-the-day-3-hutchinsons-trick/

Randomized trace estimation

Stochastic approximation of the shift operator $\mathbf{T}_{k(i)}$

$$\begin{aligned}\mathrm{tr}(\mathbf{A}\mathbf{T}_{k(i)}) &= \mathrm{tr}\left(\mathbf{A}\mathbb{E}\left[\mathbf{T}_{-k(i)}\mathbf{z}\mathbf{z}^\top\right]\right) \\ &\approx \frac{1}{r}\sum_{i=1}^{r}\left[\mathbf{z}_i^\top\mathbf{A}\mathbf{T}_{-k(i)}\mathbf{z}_i\right]\end{aligned}$$

So that

$$\begin{aligned}\delta w_i &= \frac{1}{r}\sum_{j=1}^{r}\left(\mathbf{z}_j^\top\mathbf{X}\right)\left(\delta\mathbf{Y}^\top\mathbf{T}_{-k(i)}\mathbf{z}_j\right) \\ &= \frac{1}{r}\,\mathrm{tr}\Big(\underbrace{\left(\mathbf{Z}^\top\mathbf{X}\right)}_{\overline{\mathbf{X}}\in\mathbb{R}^{r\times b}}\underbrace{\left(\delta\mathbf{Y}^\top\mathbf{T}_{-k(i)}\mathbf{Z}\right)}_{\overline{\mathbf{Y}}^\top\in\mathbb{R}^{b\times r}}\Big)\end{aligned}$$

Algorithm

Forward pass

Data: Convolution input $\mathbf{X}$ and weights $\mathbf{w}$
Result: Convolution
begin
  Draw random seed s
  $\overline{\mathbf{X}} = \mathbf{Z}(s)^\top \mathbf{X}$
  $\mathbf{Y} = \mathrm{conv}(\mathbf{X}; \mathbf{w})$
  Store $\overline{\mathbf{X}}, s$
end

Backward pass

Data: Back-propagated residual $\delta\mathbf{Y}$
Result: Gradient w.r.t to weights
begin
  Load random seed s and probed forward $\overline{\mathbf{X}}$
  $\overline{\mathbf{Y}}^\top = \delta\mathbf{Y}^\top \mathbf{T}_{-k(i)} \mathbf{Z}(s)$
  $\delta\mathbf{w} = \mathrm{tr}(\overline{\mathbf{X}}\,\overline{\mathbf{Y}}^\top)$
end

$$\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{N \times b} \Longrightarrow \overline{\mathbf{X}}, \overline{\mathbf{Y}} \in \mathbb{R}^{r \times b},\ r \ll N = n_x \times n_y$$

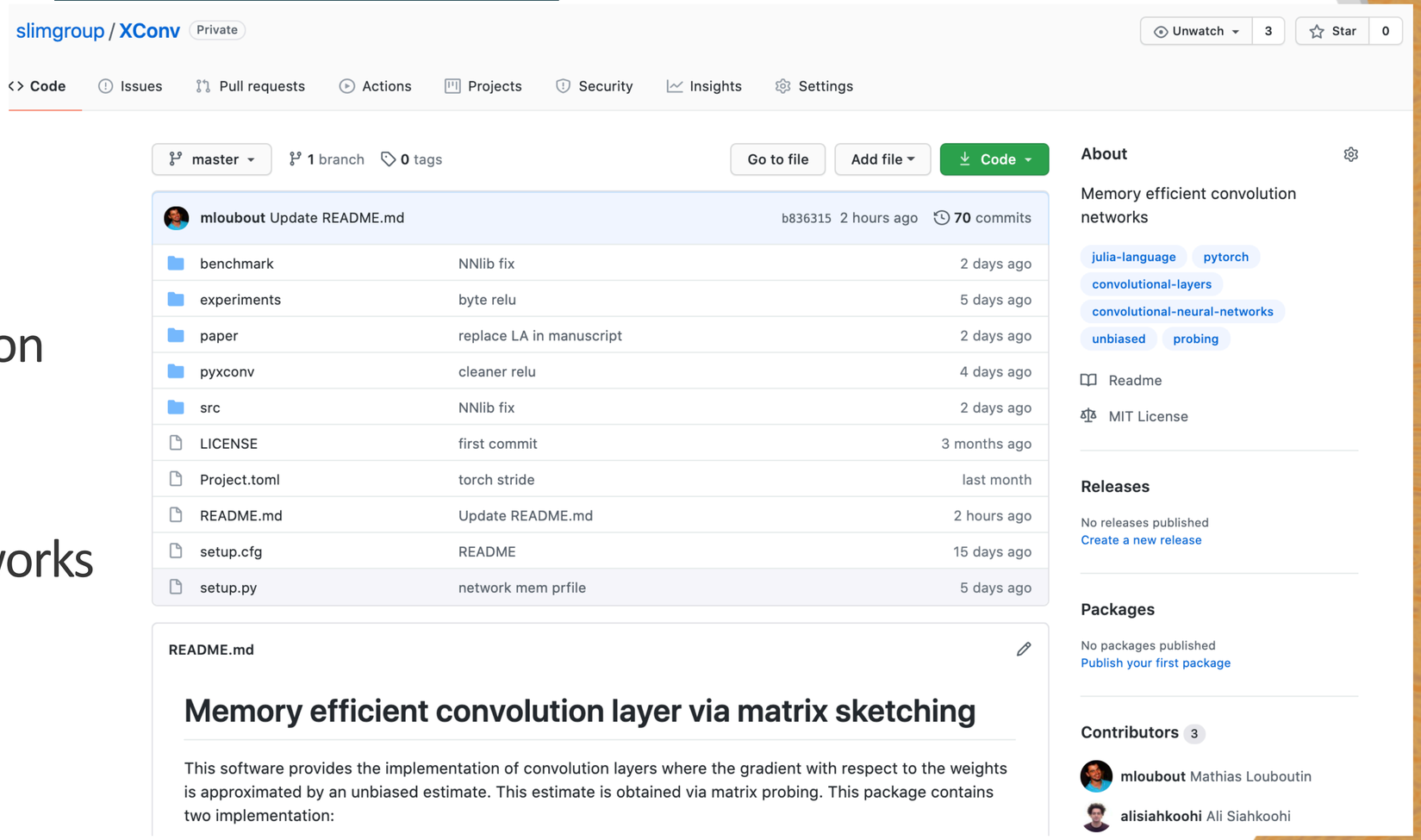
Code

```
# Pytorch
Xconv2D(cc, cc, (3, 3), bias=False, ps=ps, stride=1, padding=1)
torch.nn.Conv2d(cc, cc, (3, 3), bias=False, padding=1, stride=1)

# Converts all pytorch convolutions to XConv in a network
convert_net(model, 'net', mode='conv')

# Julia, Flux.jl overload
XConv.initXConv(ps, "EVGrad")
```

XConv:

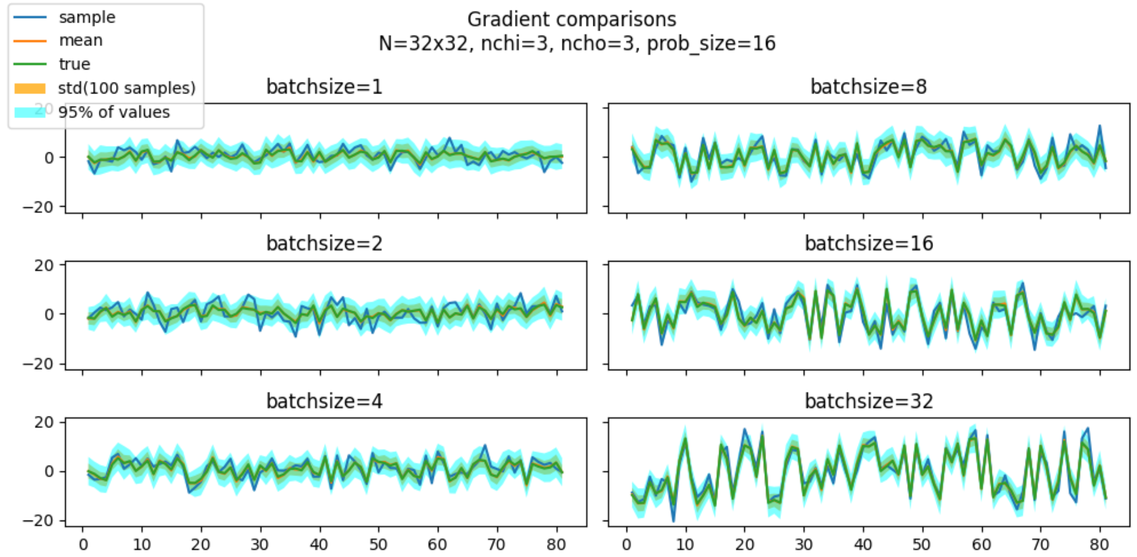
CPU/GPU codes for training w/ randomized trace estimation

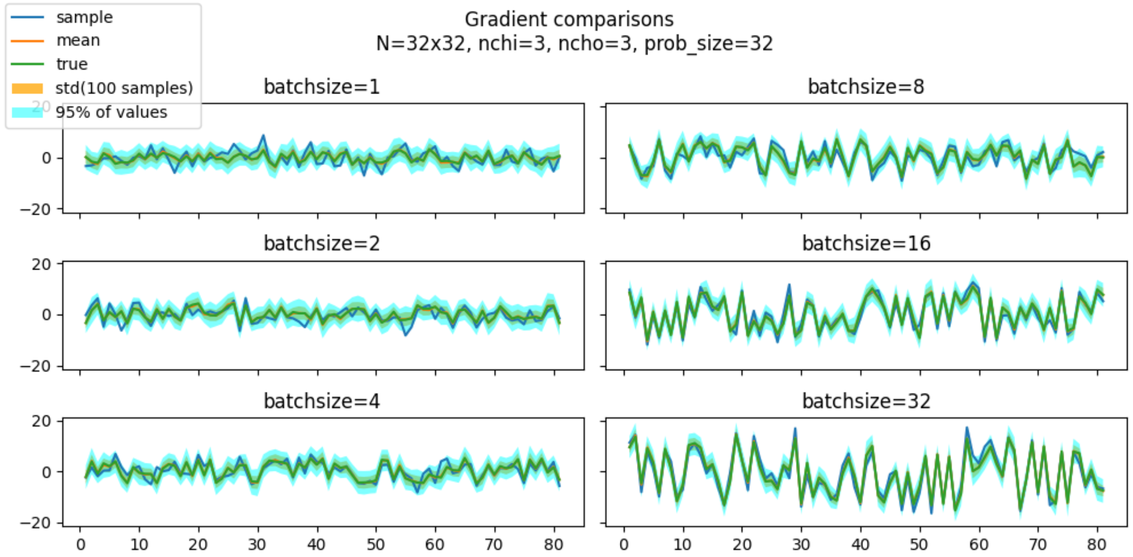
- open-source MIT license
- optimized Python implementation for PyTorch
- Julia implementation for Flux
- easily integrated in existing networks
- https://github.com/slimgroup/XConv



https://github.com/slimgroup/XConv/blob/master/benchmark/var_grad.jl

Accuracy





- theoretical control on the error of random-trace estimation
- for Gaussian probing vectors & $\mathbb{E}(\mathbf{z}\mathbf{z}^\top) = \mathbf{I}$ error decreases w/
  - batch size b
  - number of probing vectors r
- flexibility to strike balance between memory gain, compute & error

https://github.com/slimgroup/XConv/blob/master/benchmark/mem_prof.py

Convolutions only memory

```
scripts @ eas-coda-fherr07 [mlouboutin3](master)$ PYTORCH_NO_CUDA_MEMORY_CACHING=1 python3 mem_prof.py 1 &&
PYTORCH_NO_CUDA_MEMORY_CACHING=1 python3 mem_prof.py 2;
1 True
Network Sequential(
  (0): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (1): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (2): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (3): Xconv2D(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
Input size torch.Size([128, 16, 256, 256])
GPU usage probe before forward: mem: 26.951%, abs-mem: 1.0595703125 (GiB) # nothing done
GPU usage probe after forward: mem: 27.000%, abs-mem: 1.0615234375 (GiB) # after Y = N(X).mean()
GPU usage probe after backward: mem: 27.000%, abs-mem: 1.0615234375 (GiB) # after Y.backward
Network Sequential(
  (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (3): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
)
Input size torch.Size([128, 16, 256, 256])
GPU usage true before forward: mem: 26.951%, abs-mem: 1.0595703125 (GiB) # nothing done
GPU usage true after forward: mem: 65.154%, abs-mem: 2.5615234375 (GiB) # after Y = N(X).mean()
GPU usage true after backward: mem: 27.000%, abs-mem: 1.0615234375 (GiB) # after Y.backward
```

2Mb memory

1.5Gb memory

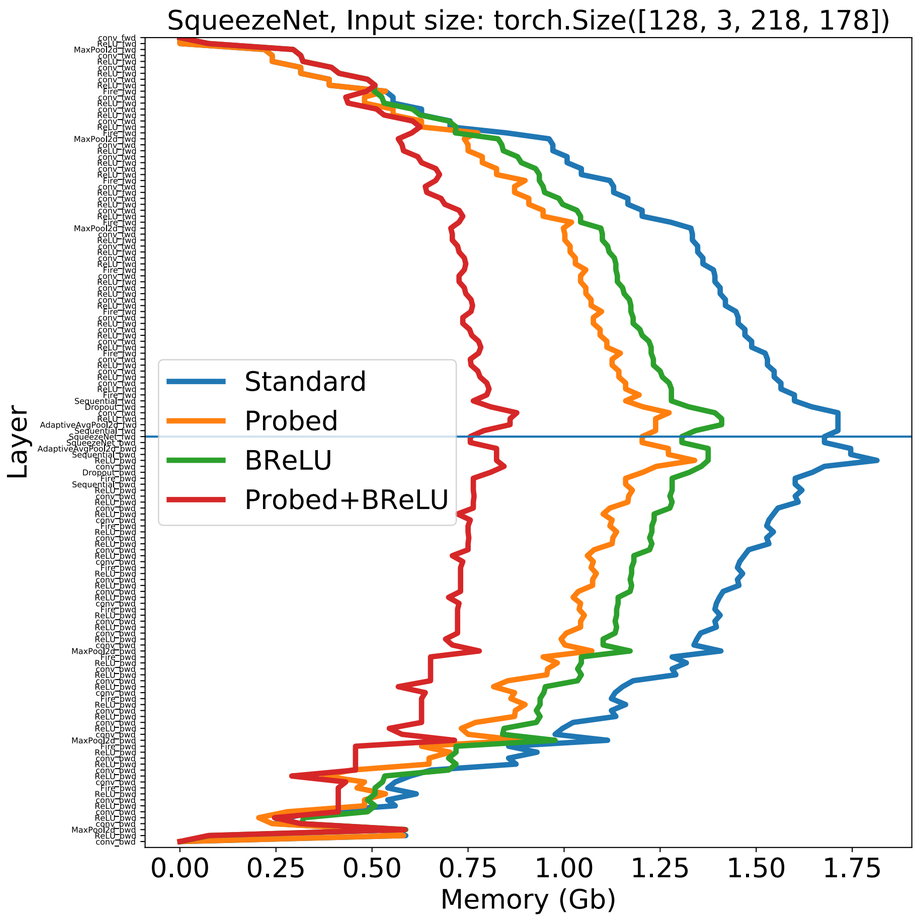
Orders of magnitude ($\mathcal{O}(10^3)$) reduction in memory usage…

Deep Learning Memory Usage and Pytorch Optimization Tricks, Quentin Fevbre. https://github.com/quentinf00/article-memory-log

https://github.com/slimgroup/XConv/blob/master/benchmark/network_mem.py

Network memory – squeezenet

- effective 100% memory improvement
- memory use as good as weakest link
- opportunities for further improvement…



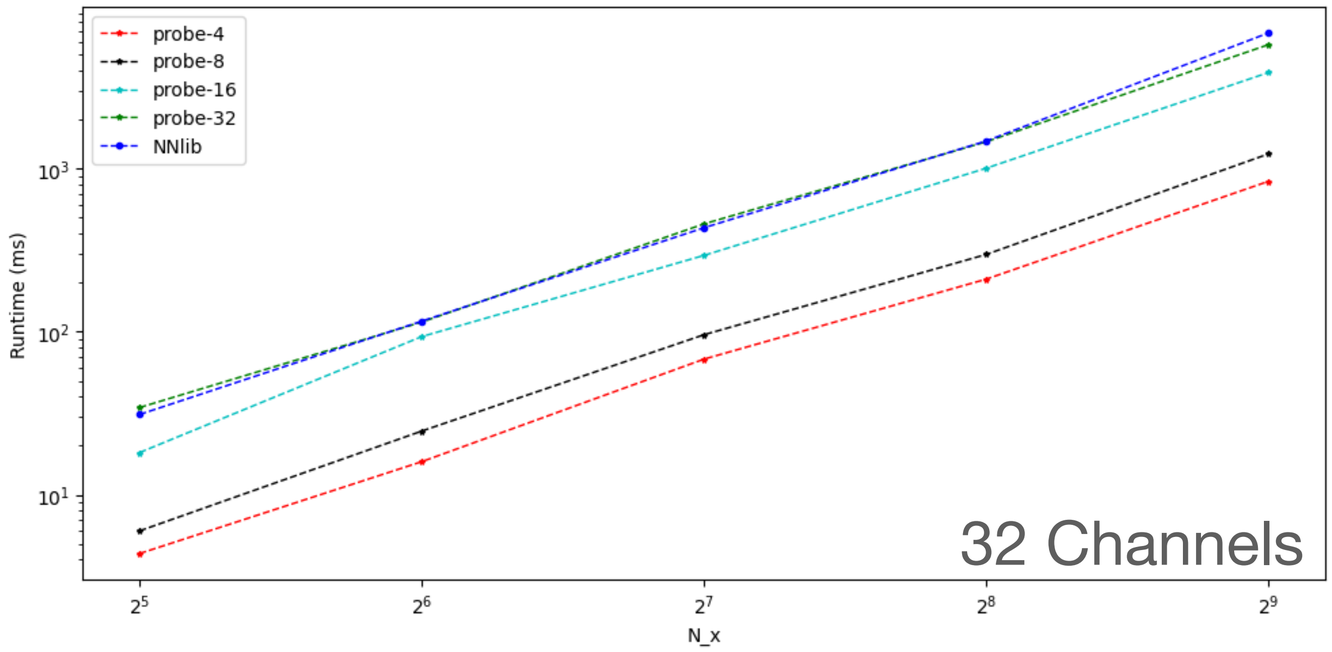
Forward

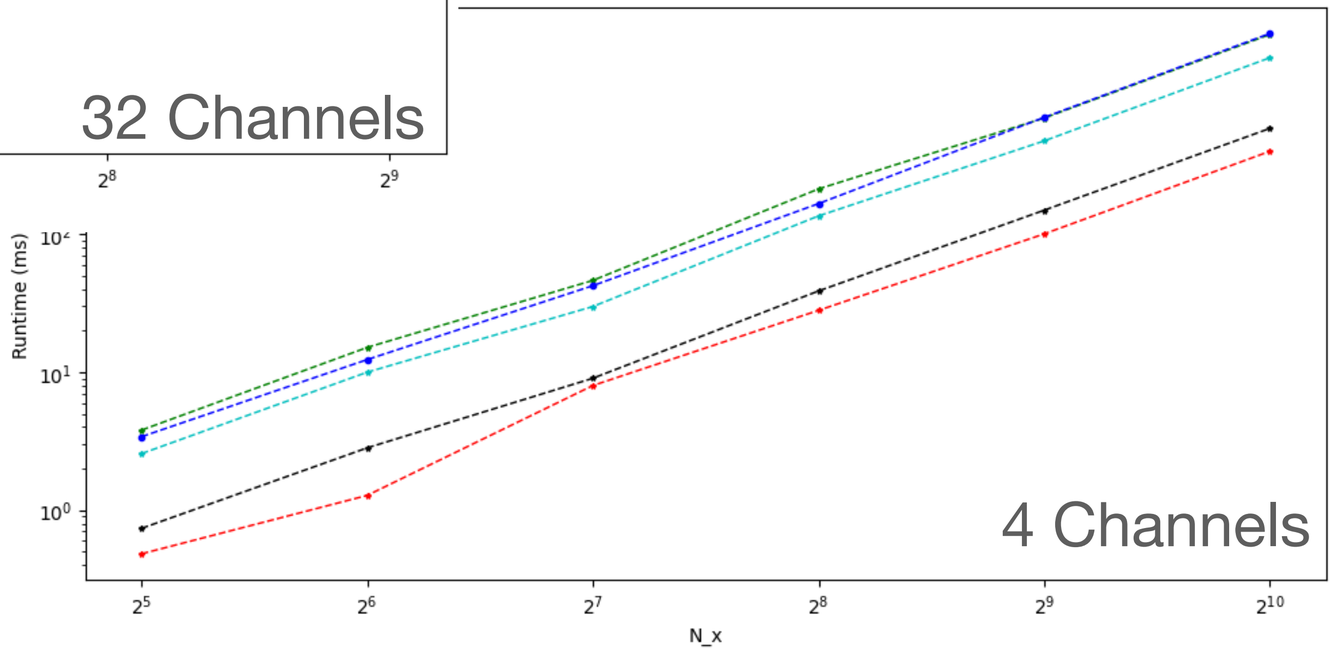
Backward

https://github.com/slimgroup/XConv/blob/master/benchmark/bench_cpu.jl

CPU

Batchsize: 64



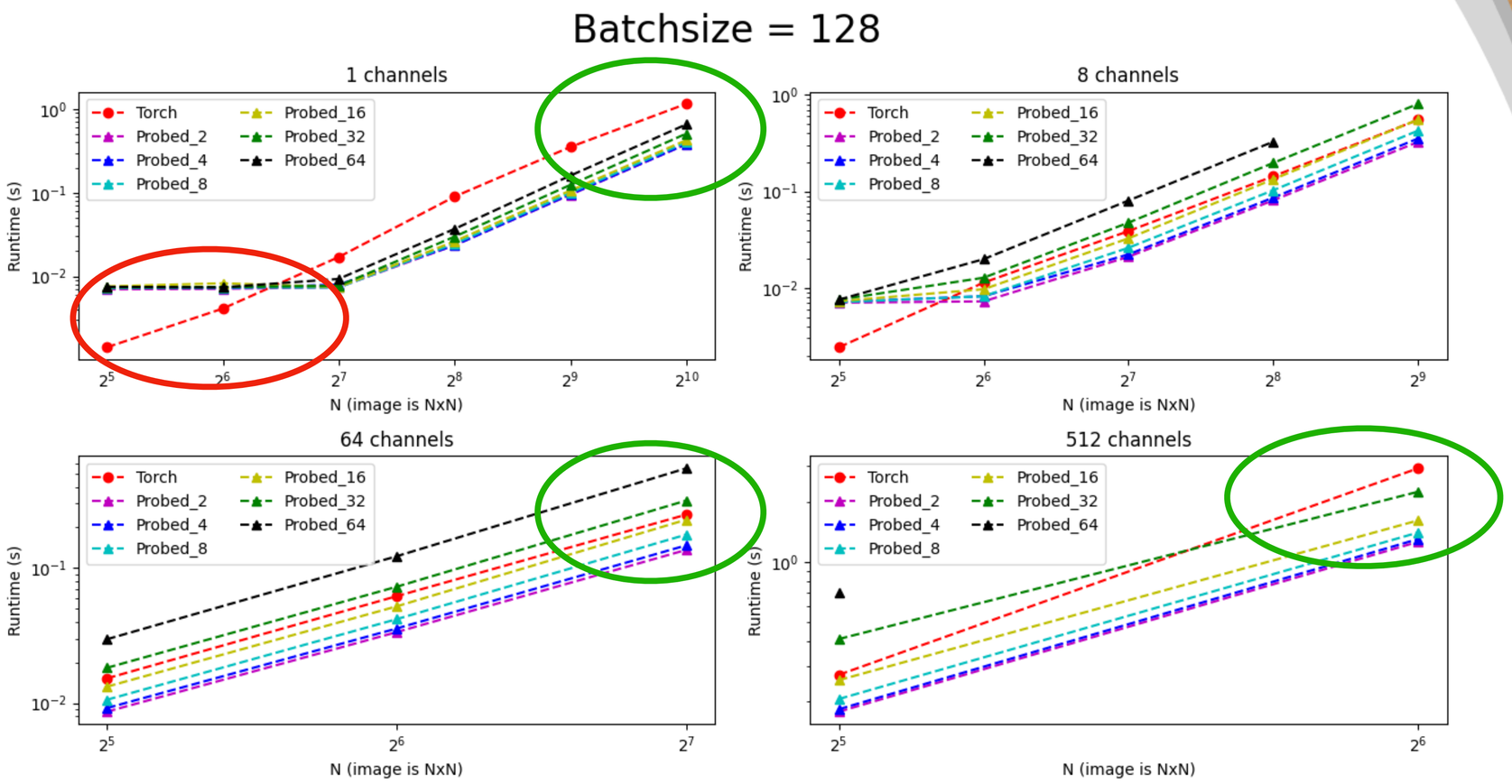


- sizes $n_x \times n_x \times (4 - 32) \times 64$ for $n_x = 2^5 - 2^{10}$
- $2^{10} - 2^{20}$ pixels
- **linear scaling**
- **up to $10\times$ speedup on CPUs in Flux (Julia)**

https://github.com/slimgroup/XConv/blob/master/benchmark/perf_pyxconv.py

GPU

- sizes
  $N \times N \times (1 - 512) \times 128$
  for $N = 2^5 - 2^{10}$
- **linear scaling in PyTorch**
- **up to $3\times$ speedup on GPUs**
- not good for small sizes & small # of channels



GPU

Improved performance

- larger batch sizes
- image sizes

Compared to PyTorch

- **up to 2-3 X faster**
- **faster for large images**

[---------- 2D conv b=128 ----------]

4 threads:	nn.Conv2d	Xconv2D_2	Xconv2D_4	Xconv2D_8	Xconv2D_16	Xconv2D_32	Xconv2D_64
32x32, ci=1	1	7	7	7	8	7	7
64x64, ci=1	4	7	7	7	8	7	7
128x128, ci=1	17	7	8	7	7	8	9
256x256, ci=1	90	24	23	24	26	30	40
512x512, ci=1	400	95	96	100	106	124	160
1024x1024, ci=1	1158	380	380	400	430	500	660
32x32, ci=8	2	7	7	7	7	8	8
64x64, ci=8	11	7	8	8	10	13	20
128x128, ci=8	40	20	20	26	33	47	81
256x256, ci=8	144	81	86	103	134	200	328
512x512, ci=8	600	330	400	430	560	810	
32x32, ci=64	15	9	9	11	13	20	30
64x64, ci=64	62	34	40	40	50	70	122
128x128, ci=64	250	136	147	200	230	300	500
32x32, ci=512	270	180	183	200	300	400	700
64x64, ci=512	2934	1258	1299	1400	1610	2239	

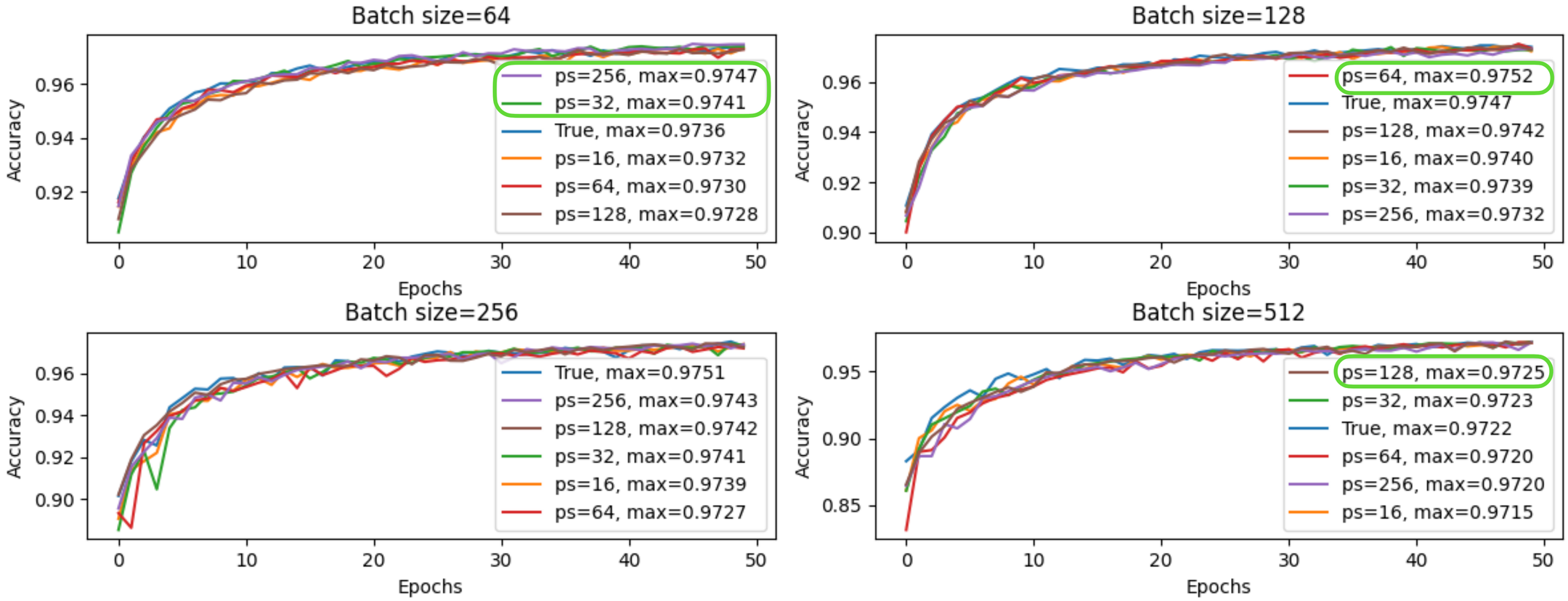
Times are in milliseconds (ms).

[---------- 2D conv b=256 ----------]

4 threads:	nn.Conv2d	Xconv2D_2	Xconv2D_4	Xconv2D_8	Xconv2D_16	Xconv2D_32	Xconv2D_64
32x32, ci=1	3	7	7	7	7	8	8
64x64, ci=1	8	7	7	8	8	7	7
128x128, ci=1	30	11	11	11	12	13	15
256x256, ci=1	145	40	45	45	50	51	59
512x512, ci=1	600	190	200	200	200	200	255
1024x1024, ci=1	2250	760	800	750	780	850	1037
32x32, ci=8	5	7	7	7	8	8	8
64x64, ci=8	22	11	13	10	14	20	25
128x128, ci=8	80	37	38	42	50	70	100
256x256, ci=8	310	159	167	200	200	260	400
32x32, ci=64	30	16	16	17	20	25	40
64x64, ci=64	120	60	63	70	80	100	156
128x128, ci=64	500	300	300	280	330	400	700
32x32, ci=512	1457	616	630	640	690	860	1147

Times are in milliseconds (ms).

Accuracy MNIST training



- Validation classification accuracy vs epochs
- **No loss in classification accuracy**
- **Ability to work w/ larger batch sizes**

Observations

Gradient calculations w/ random trace estimation

- approximate gradient w/ controllable error
- theoretical memory reduction ($\mathcal{O}(n_x \times n_y \times n_c \times b)$ to $\mathcal{O}(r \times b)$) training CNNs
- effective memory improvement of $2\times$ for actual DNN
- computational performance improvement for larger images/channels
- speedups of $2-3\times$ for GPUs and $10\times$ CPUs
- comparable NN performance after training
- option to increase batch sizes (offset inaccuracies gradient)

We leveraged ideas known from randomized linear algebra.

Bottom line

NN training w/ randomized trace estimation

- more efficient use of hardware & less CO2 production
- facilitates ML@scale (e.g. video encoding, 3D seismic segmentation, etc.)
- allows for training next-generation memory-efficient & larger NNs

Use of randomized algorithms

- adaptive accuracy control during training
- use of optical devices (LightOn) for random projections

Technology is as good as the weakest link…

- reliance on dense linear algebra impedes ML@scale
- optimization landscape remains a challenge

Challenge – stencil computations

Forward evaluation DNNs scales poorly to large problem sizes

- suboptimal patches $\Longrightarrow$ miss long range correlations
- strains systems to which NNs are deployed
- performance optimization call for coding by hand

Additional challenges stifle innovations in

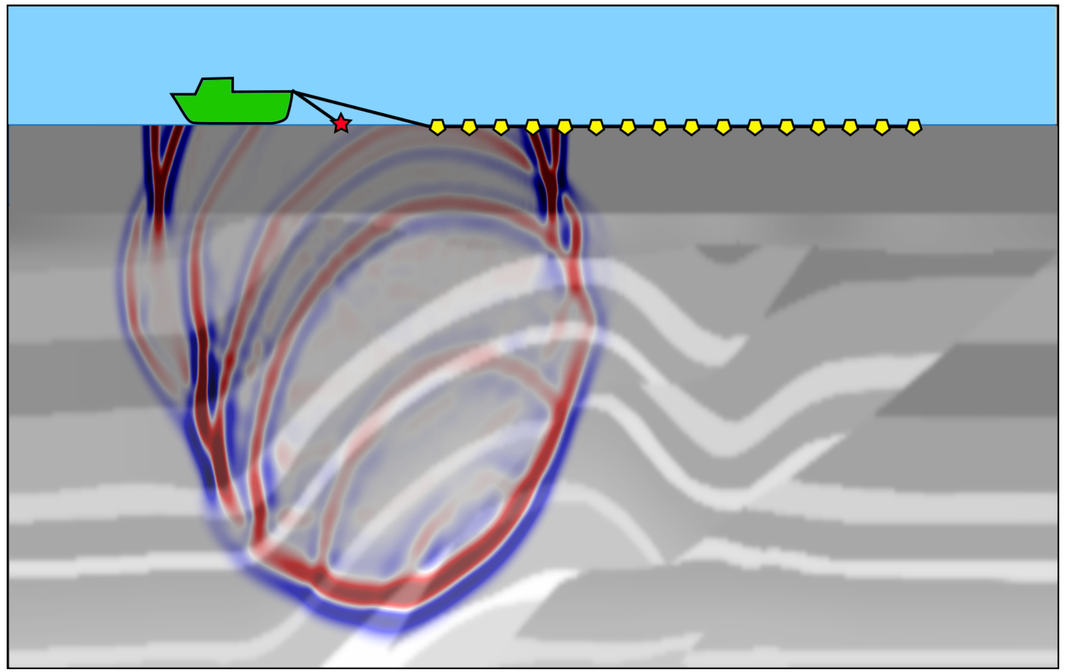
- video encoding/labeling/segmentation
- running trained NNs on different types of hardware

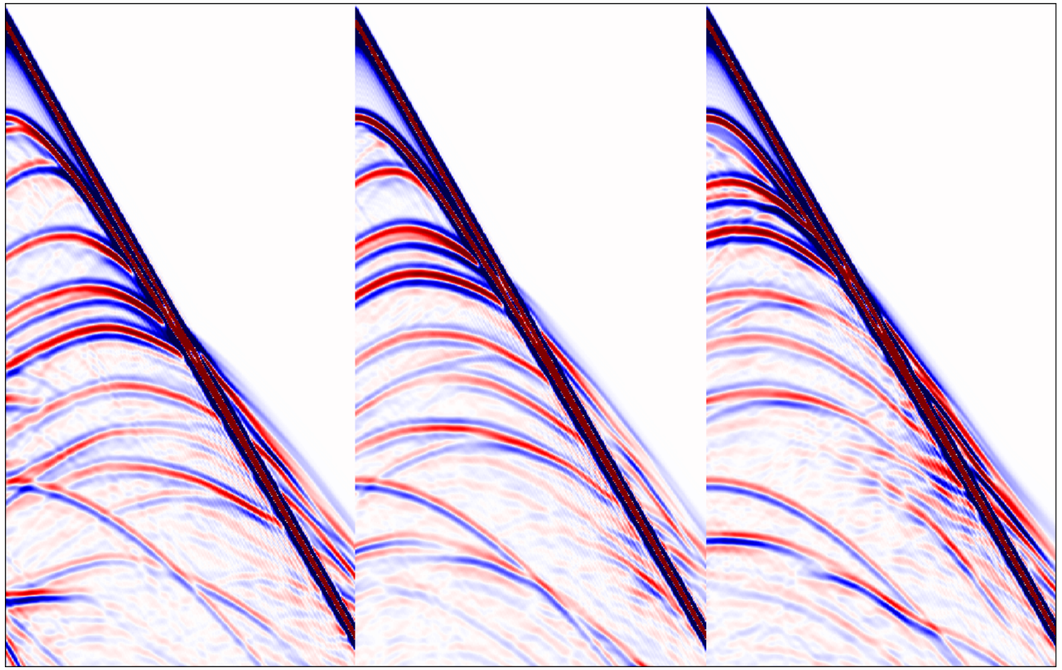
Solution: exploit just-in-time compiler technology for seismic…

Seismic inverse problems

Estimate unknown subsurface properties from seismic (surface) data:

- image geological structures/discontinuities
- estimate physical rock properties (wavespeed, density, etc.)
- turn around times order 1 year

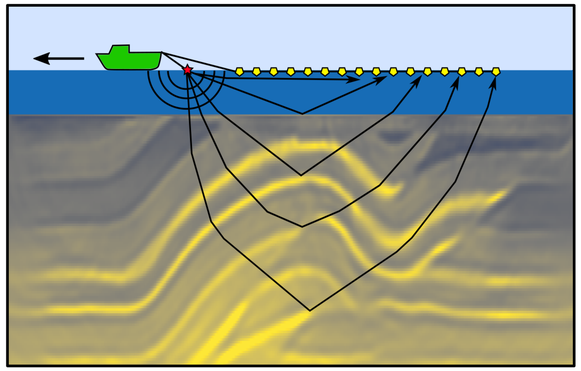




Inverse problem

Parametric nonlinear least-squares:

$$\min_{\mathbf{m}} \frac{1}{2} \sum_{i=1}^{n_s} \|\mathbf{d}_i - \mathbf{F}[\mathbf{m}]\mathbf{q}_i\|_2^2$$



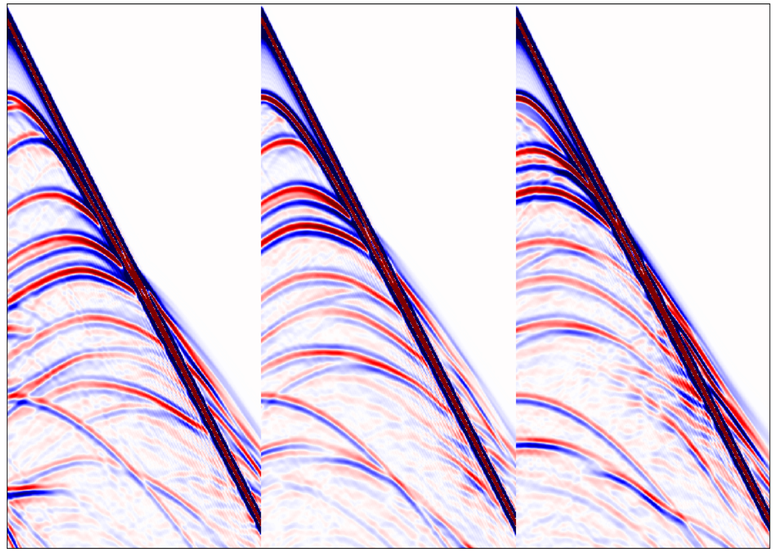
Sensitivities w.r.t. model parameters: $\frac{\partial \mathbf{F}[\mathbf{m}]\mathbf{q}}{\partial \mathbf{m}}$

$$\min_{\mathbf{m}} \Phi(\mathbf{m})$$

$$\mathbf{J} = -\mathbf{P}_r \mathbf{A}[\mathbf{m}]^{-1} \mathrm{diag}\left(\frac{\partial \mathbf{A}[\mathbf{m}]}{\partial \mathbf{m}} \mathbf{A}[\mathbf{m}]^{-1} \mathbf{P}_s^\top \mathbf{q} \right)$$

Gradient of objective function via back-propagation:

$$\mathbf{g} = \sum_{i=1}^{n_s} \mathbf{J}^\top (\mathbf{F}[\mathbf{m}]\mathbf{q}_i - \mathbf{d}_i)$$



Ruthotto, Lars et. al. "Deep neural networks motivated by partial differential equations." *Journal of Mathematical Imaging and Vision* (2019): 1-13.
Peters, Bas. "Deep connections between learning from limited labels & physical parameter estimation--inspiration for regularization." *arXiv preprint arXiv:2003.07908* (2020).

Connections

Time steps discrete scalar wave equation:

$$\mathbf{u}_{\ell+1} = \left(2\mathbf{I} + \Delta t^2 \operatorname{diag}(\mathbf{m})\nabla^2\right)\mathbf{u}_\ell - \mathbf{u}_{\ell-1} + \mathbf{q}_{\ell+1}$$

- medium coefficients $\mathbf{m}$ constant in time & spatially varying
- no action over distance (coefficients enter as pure diagonal)
- relies on multi-dimensional action of ∇^2
- do back-propagation routinely for $\mathcal{O}(10^4)$ time steps on grids of $\mathcal{O}(10^9)$

Raising the level of abstraction

$$m\frac{\partial^2 u}{\partial t^2} + \eta\frac{\partial u}{\partial t} - \Delta u = 0$$

Devito

```
eqn = m * u.dt2 + eta * u.dt - u.laplace
solve(eqn, u.forward)
```

```
void kernel(…) { … }
```

Flexibility in space/time discretization

```
u = TimeFunction(…, space_order=so)
eqn = m * u.dt2 + eta * u.dt - u.laplace
solve(eqn, u.forward)
```

so=4

```
  for (int time = time_m, t0 = (time)%(3), t1 = (time + 1)%(3), t2 = (time +
2)%(3); time <= time_M; time += 1, t0 = (time)%(3), t1 = (time + 1)%(3), t2 =
(time + 2)%(3)) {
    for (int x = x_m; x <= x_M; x += 1) {
      for (int y = y_m; y <= y_M; y += 1) {
        for (int z = z_m; z <= z_M; z += 1) {
          u[t1][x + 4][y + 4][z + 4] = 2*pow(dt,
3)*(-2.08333333333333e-4F*u[t0][x + 2][y + 4][z + 4] +
3.33333333333333e-3F*u[t0][x + 3][y + 4][z + 4] - 2.08333333333333e-4F*u[t0]
[x + 4][y + 2][z + 4] + 3.33333333333333e-3F*u[t0][x + 4][y + 3][z + 4] -
2.08333333333333e-4F*u[t0][x + 4][y + 4][z + 2] + 3.33333333333333e-3F*u[t0]
[x + 4][y + 4][z + 3] - 1.875e-2F*u[t0][x + 4][y + 4][z + 4] +
3.33333333333333e-3F*u[t0][x + 4][y + 4][z + 5] - 2.08333333333333e-4F*u[t0]
[x + 4][y + 4][z + 6] + 3.33333333333333e-3F*u[t0][x + 4][y + 5][z + 4] -
2.08333333333333e-4F*u[t0][x + 4][y + 6][z + 4] + 3.33333333333333e-3F*u[t0]
[x + 5][y + 4][z + 4] - 2.08333333333333e-4F*u[t0][x + 6][y + 4][z + 4])/
(pow(dt, 2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 4][y + 4][z + 4]) +
pow(dt, 2)*damp[x + 1][y + 1][z + 1]*u[t2][x + 4][y + 4][z + 4]/(pow(dt,
2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 4][y + 4][z + 4]) + 4*dt*m[x + 4][y
+ 4][z + 4]*u[t0][x + 4][y + 4][z + 4]/(pow(dt, 2)*damp[x + 1][y + 1][z + 1]
+ 2*dt*m[x + 4][y + 4][z + 4]) - 2*dt*m[x + 4][y + 4][z + 4]*u[t2][x + 4][y +
4][z + 4]/(pow(dt, 2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 4][y + 4][z +
4]);
        }
      }
    }
  }
```

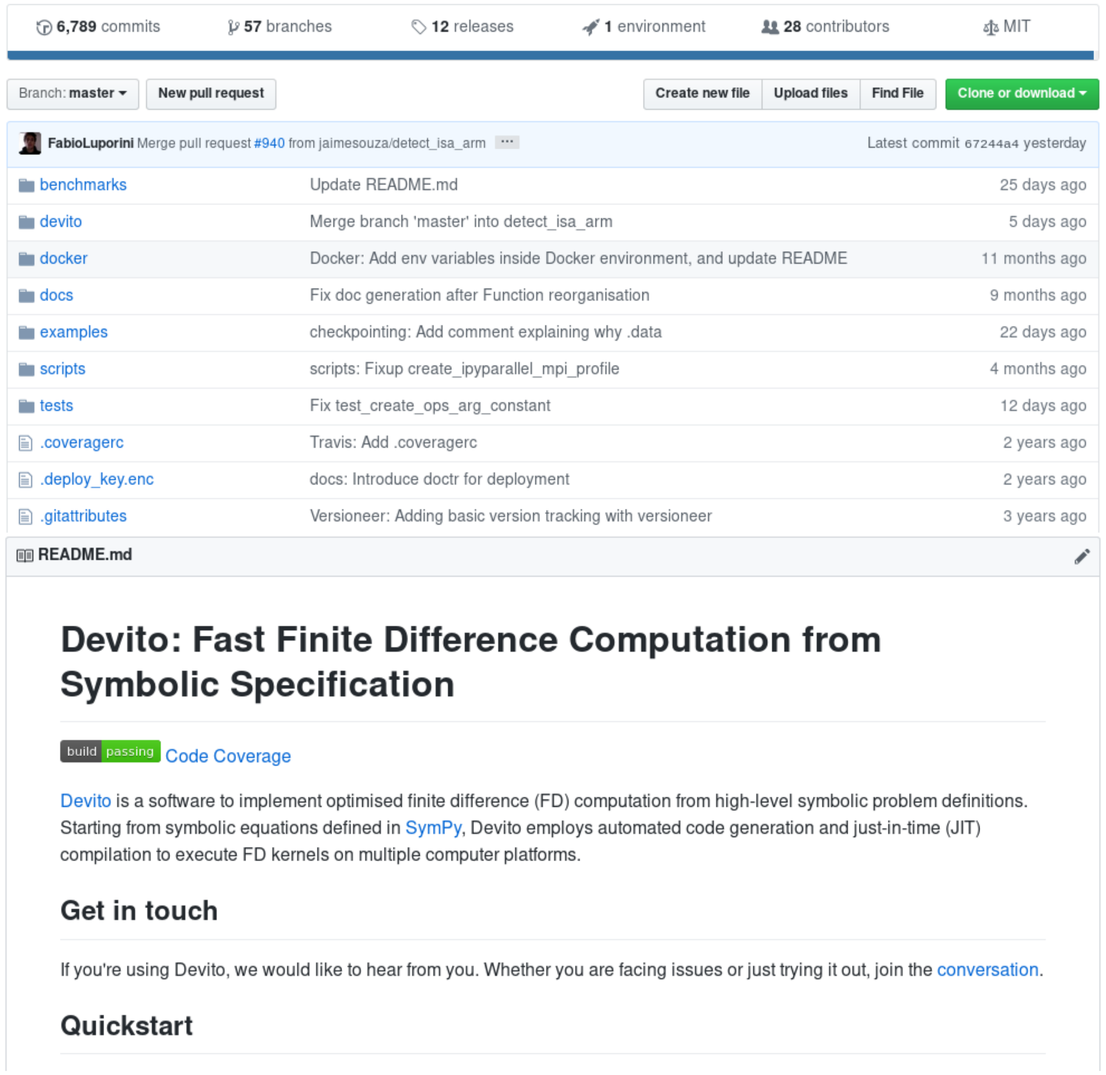
so=12

```
  for (int time = time_m, t0 = (time)%(3), t1 = (time + 1)%(3), t2 = (time +
2)%(3); time <= time_M; time += 1, t0 = (time)%(3), t1 = (time + 1)%(3), t2 =
(time + 2)%(3)) {
    for (int x = x_m; x <= x_M; x += 1) {
      for (int y = y_m; y <= y_M; y += 1) {
        for (int z = z_m; z <= z_M; z += 1) {
          u[t1][x + 12][y + 12][z + 12] = 2*pow(dt,
3)*(-1.5031265031265e-7F*u[t0][x + 6][y + 12][z + 12] +
2.5974025974026e-6F*u[t0][x + 7][y + 12][z + 12] - 2.23214285714286e-5F*u[t0][x
+ 8][y + 12][z + 12] + 1.32275132275132e-4F*u[t0][x + 9][y + 12][z + 12] -
6.69642857142857e-4F*u[t0][x + 10][y + 12][z + 12] + 4.28571428571429e-3F*u[t0]
[x + 11][y + 12][z + 12] - 1.5031265031265e-7F*u[t0][x + 12][y + 6][z + 12] +
2.5974025974026e-6F*u[t0][x + 12][y + 7][z + 12] - 2.23214285714286e-5F*u[t0][x
+ 12][y + 8][z + 12] + 1.32275132275132e-4F*u[t0][x + 12][y + 9][z + 12] -
6.69642857142857e-4F*u[t0][x + 12][y + 10][z + 12] + 4.28571428571429e-3F*u[t0]
[x + 12][y + 11][z + 12] - 1.5031265031265e-7F*u[t0][x + 12][y + 12][z + 6] +
2.5974025974026e-6F*u[t0][x + 12][y + 12][z + 7] - 2.23214285714286e-5F*u[t0][x
+ 12][y + 12][z + 8] + 1.32275132275132e-4F*u[t0][x + 12][y + 12][z + 9] -
6.69642857142857e-4F*u[t0][x + 12][y + 12][z + 10] + 4.28571428571429e-3F*u[t0]
[x + 12][y + 12][z + 11] - 2.23708333333333e-2F*u[t0][x + 12][y + 12][z + 12] +
4.28571428571429e-3F*u[t0][x + 12][y + 12][z + 13] - 6.69642857142857e-4F*u[t0]
[x + 12][y + 12][z + 14] + 1.32275132275132e-4F*u[t0][x + 12][y + 12][z + 15] -
2.23214285714286e-5F*u[t0][x + 12][y + 12][z + 16] + 2.5974025974026e-6F*u[t0]
[x + 12][y + 12][z + 17] - 1.5031265031265e-7F*u[t0][x + 12][y + 12][z + 18] +
4.28571428571429e-3F*u[t0][x + 12][y + 13][z + 12] - 6.69642857142857e-4F*u[t0]
[x + 12][y + 14][z + 12] + 1.32275132275132e-4F*u[t0][x + 12][y + 15][z + 12] -
2.23214285714286e-5F*u[t0][x + 12][y + 16][z + 12] + 2.5974025974026e-6F*u[t0]
[x + 12][y + 17][z + 12] - 1.5031265031265e-7F*u[t0][x + 12][y + 18][z + 12] +
4.28571428571429e-3F*u[t0][x + 13][y + 12][z + 12] - 6.69642857142857e-4F*u[t0]
[x + 14][y + 12][z + 12] + 1.32275132275132e-4F*u[t0][x + 15][y + 12][z + 12] -
2.23214285714286e-5F*u[t0][x + 16][y + 12][z + 12] + 2.5974025974026e-6F*u[t0]
[x + 17][y + 12][z + 12] - 1.5031265031265e-7F*u[t0][x + 18][y + 12][z + 12])/
(pow(dt, 2)*damp[x + 1][y + 1][z + 1] + 2*dt*m[x + 12][y + 12][z + 12]) +
```

Open-source software

Devito:

- Open-source MIT license
- High-level Python interface for discretization of ODEs + PDEs using finite differences
- Automatic performance optimization and JIT code generation
- **https://github.com/opesci/devito**

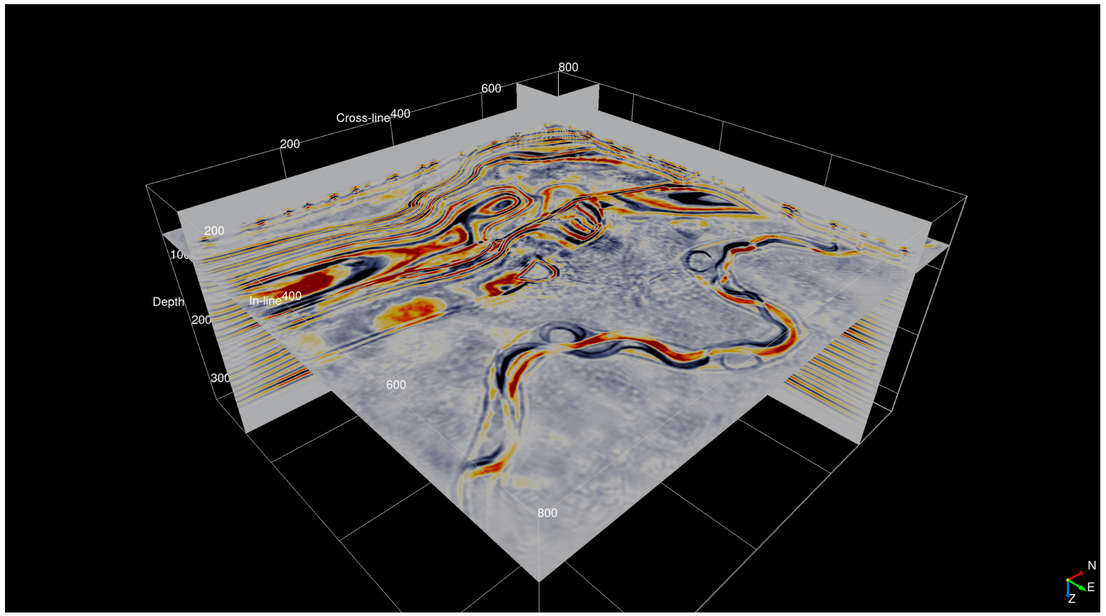


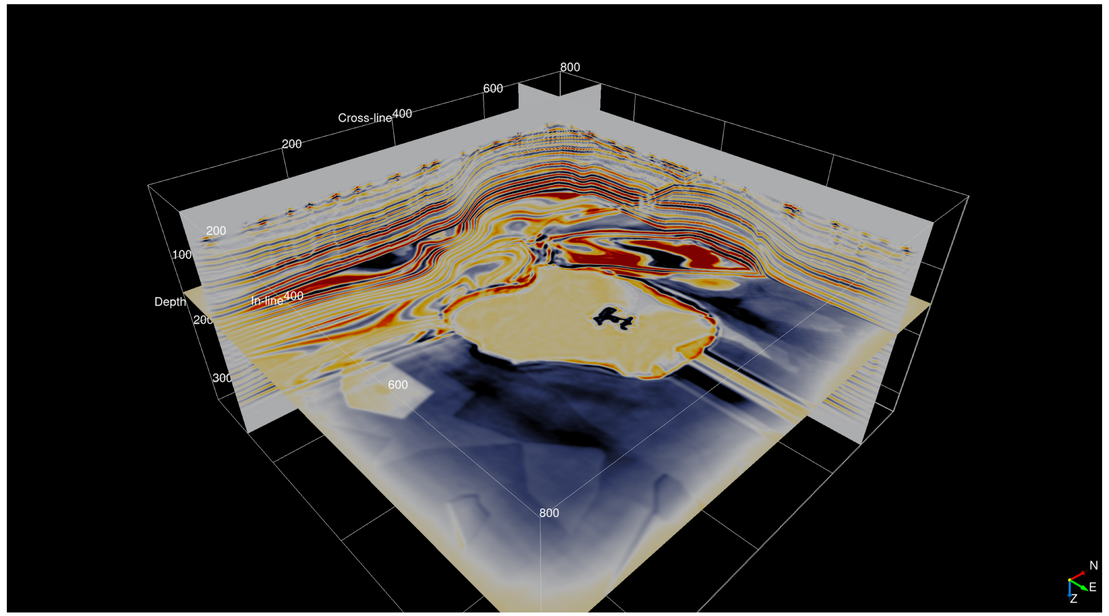
Witte, Philipp A., et al. "An event-driven approach to serverless seismic imaging in the cloud." *IEEE Transactions on Parallel and Distributed Systems* 31.9 (2020): 2032-2049.

Serverless seismic imaging on Azure

“Small” 3D Imaging case study w/ Devito DSL + JUDI

- Data set: 1,500 source locations (~**2.1 TB** data)
- Model: 10 x 10 x 3.325 km (**270 million** unknowns)
- PDE: tilted transversely isotropic (TTI) wave equation, **3,500** time steps
- **Cost: < 10,000$** on 100 E64/E64s instances (2 VMs per gradient with MPI)
- **Peak performance: 140 TFLOPs**
- **cost single image is comparable to training a large NN**





Devito – seismic

Louboutin, Mathias, et al. "Devito (v3. 1.0): an embedded domain-specific language for finite differences and geophysical exploration." *Geoscientific Model Development* 12.3 (2019): 1165-1187.
Lange, Michael, et al. "Devito: Towards a generic finite difference dsl using symbolic python." *2016 6th Workshop on Python for High-Performance and Scientific Computing (PyHPC)*. IEEE, 2016.

SLIM

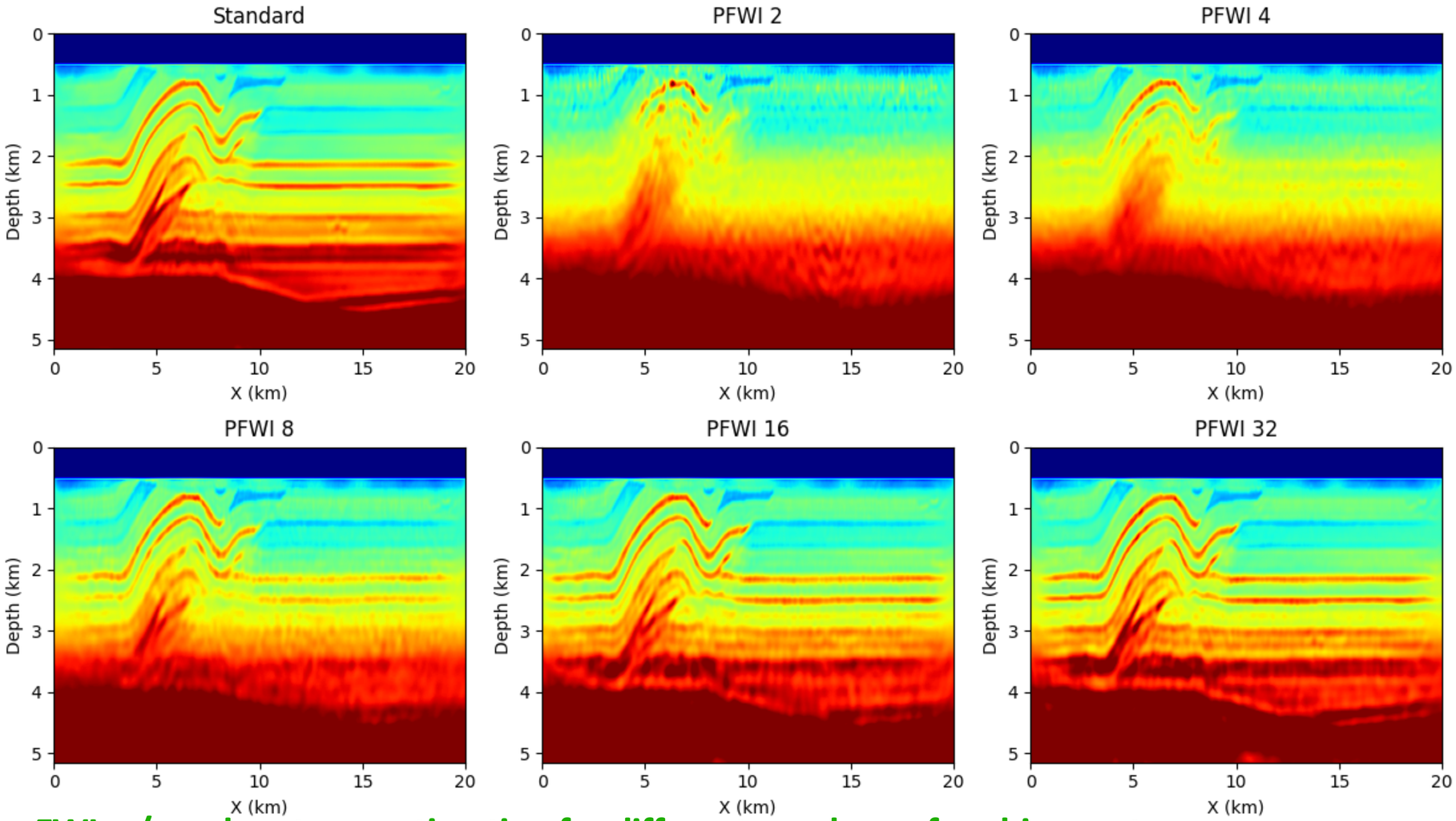
GPU support roadmap

- Support for multiple target languages
  - OpenMP, OpenACC
  - potentially: CUDA, HIP, SYCL, …
- Unreliability of the target languages' software stack
- Multi-GPU support:
  - Make it possible to run different shots on different GPUs
  - Single-node multi-GPU via domain decomposition
  - Multi-node multi-GPU via domain decomposition
- Data movement (optimized)
- Data streaming (optimized)
- Kernel performance (best so far: 27 GPOINTS on iso-acoustic O(2, 8))

Legend:

- **Done**
- **Nearly done**
- **In progress**
- **Potentially later this year**

FWI w/ randomized trace estimation



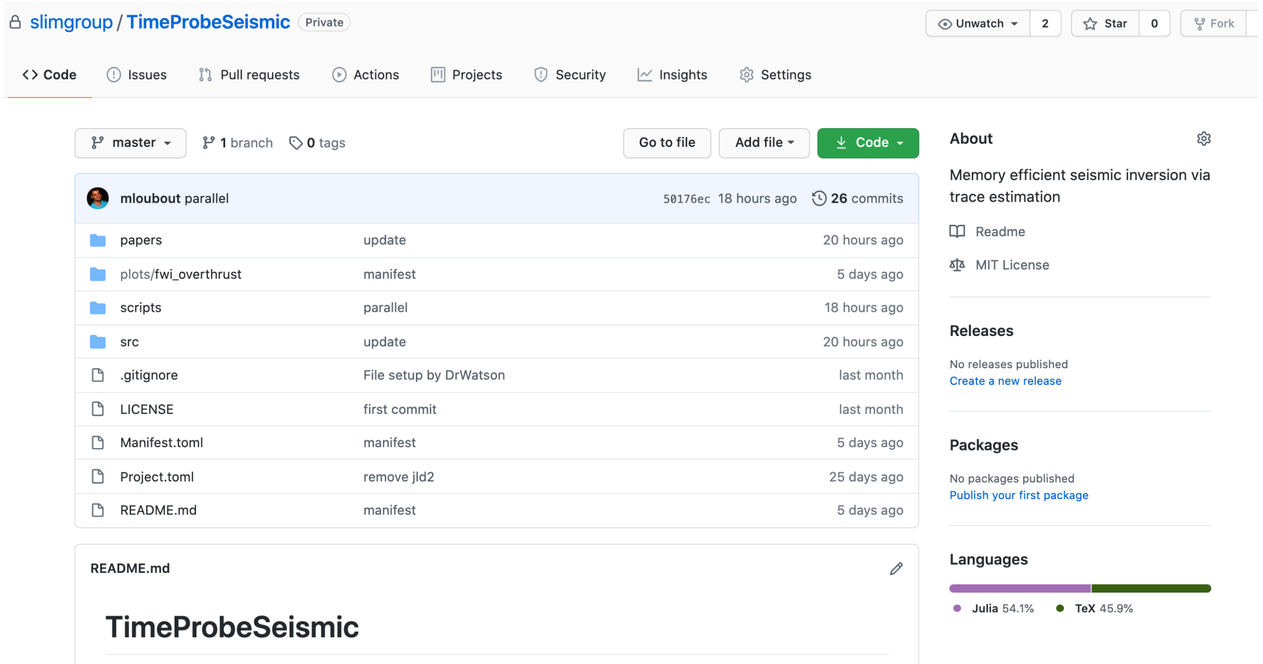
- Validation FWI w/ random trace estimation for different numbers of probing vectors
- $40 - 750 \times$ memory reductions for FWI
- Little loss in inversion accuracy

Open source software

TimeProbeSeismic:

- Open-source MIT license
- Built on top of JUDI
- Leverages Devito
- https://github.com/slimgroup/TimeProbeSeismic

```julia
# Standard JUDI
function objective_function(x)
  model0.m .= x
   f, g = fwi_objective(model0, q[idx], d_obs[idx]; options=opt)
end
options = spg_options(verbose = 3, maxIter = fevals, memory = 3,
iniStep = 1f0)
g_const = 0
sol = spg(x->objective_function(x), vec(m0), ProjBound, options)

# Probing extension
function objective_function(x, ps)
  model0.m .= x
   f, g = fwi_objective(model0, q[idx], d_obs[idx], ps; options=opt)
end
ps = 32
global g_const = 0
sol = spg(x->objective_function(x, ps), vec(m0), ProjBound, options)
```



Ruthotto, Lars et. al. "Deep neural networks motivated by partial differential equations." *Journal of Mathematical Imaging and Vision* (2019): 1-13.
Peters, Bas. "Deep connections between learning from limited labels & physical parameter estimation--inspiration for regularization." *arXiv preprint arXiv:2003.07908* (2020).

Connections

Convolutional residual nets:

$$\mathbf{y}_\ell = \mathbf{y}_{\ell-1} + hf(\mathbf{W}_l \mathbf{y}_{\ell-1})$$

- "operator coefficients" vary in "time" & constant in space
- action of over a distance ("coefficients" enter w/ as convolution operator)
- relies on multi-dimensional convolutions $\mathbf{W}_\ell$
- push for wider & deeper networks

Leverage similarities between wave simulations & CNNs…

Devito – CNN

Use Devito's Domain Specific Language

- implement CNNs
- supports parallellism (OMP+MPI)
- flexibility for different hardware

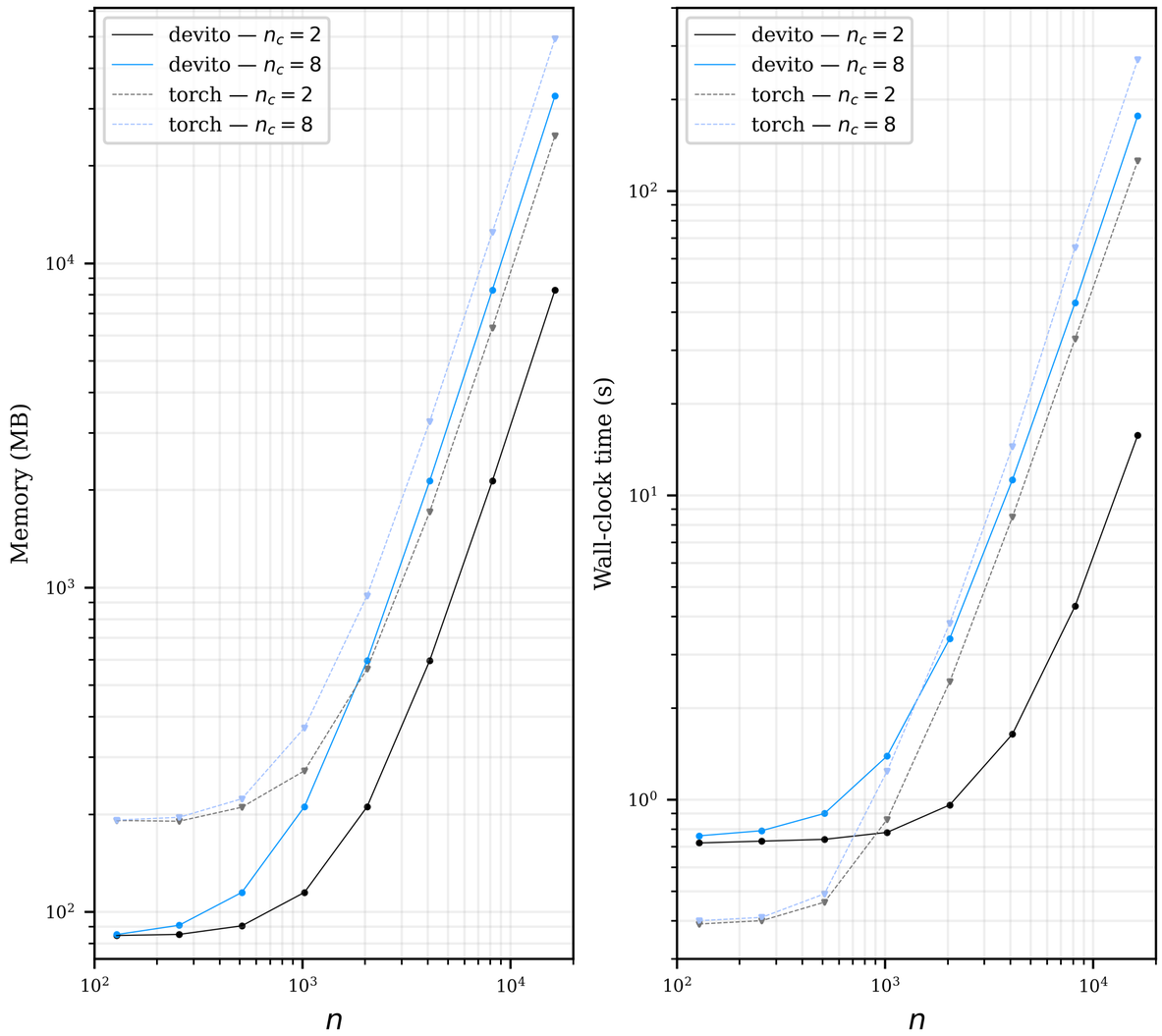
Rapid prototype implementation CNNs

- $1.5 - 2\times$ **reduction in memory**
- $1.5 - 10\times$ **speed improvement on CPUs**

3 X 3 kernel size

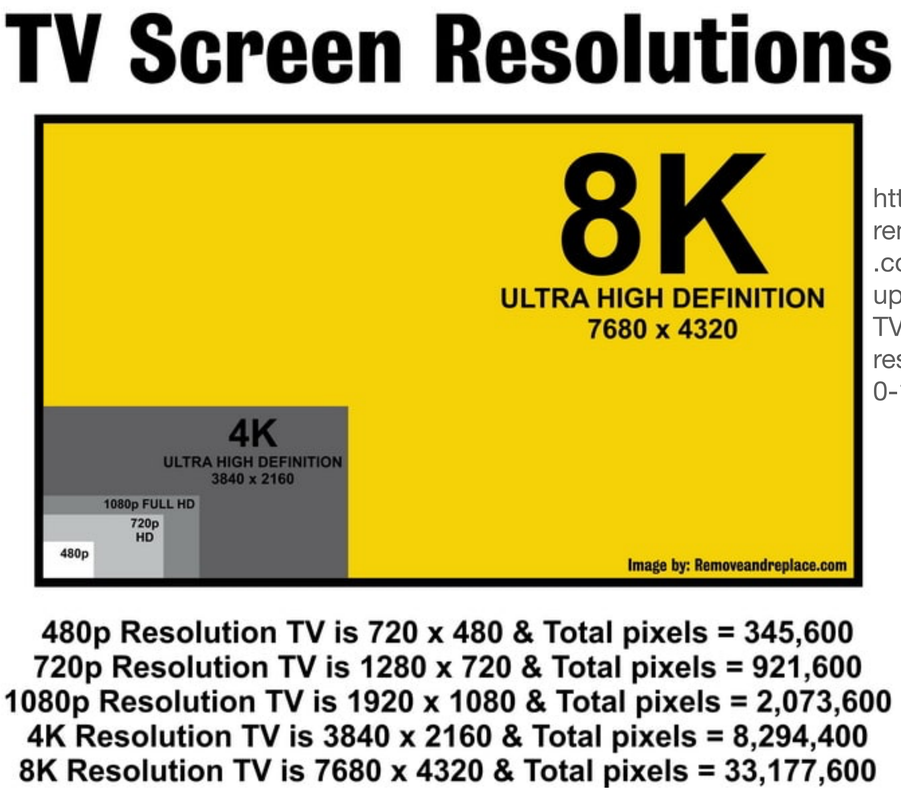
n x n w/ 2 or 8 channels

wall-clock time & peak memory usage



Scaling – on CPU

https://www.pcmag.com/news/what-is-8k-should-you-buy-a-new-tv-or-wait



https://removeandreplace.com/wp-content/uploads/2016/04/TV-screen-resolutions-480-720-1080-4K-8K.jpg

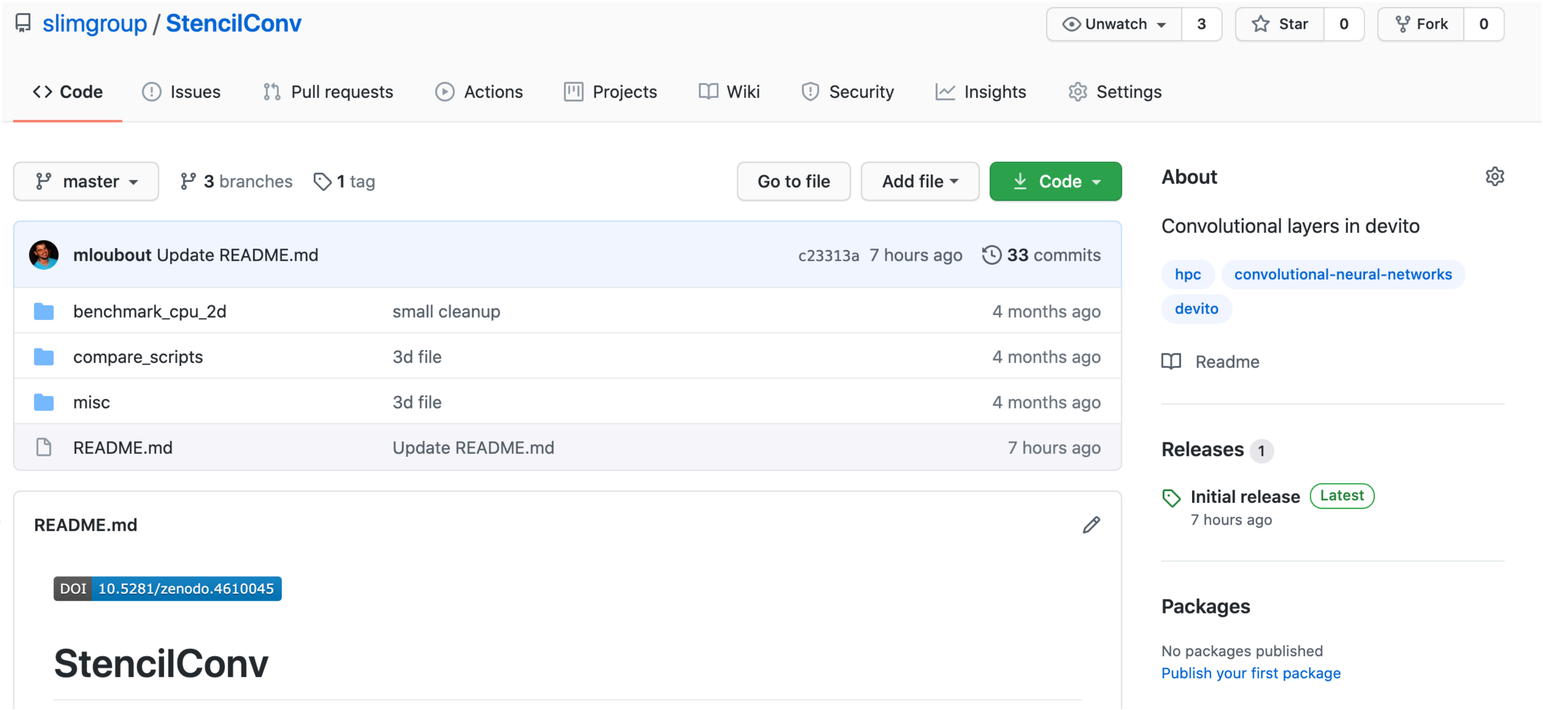
- Validated scaling to size $N \times N \times 4 \times 1$ for $N = 2^{16}$ on Azure's HBv2 instance (AMD EPYC)
- 2^{32} pixels equals $64 \times$ 8K images
- Runtime 54 seconds 120 threads
- 480 Gb memory not enough for PyTorch…

https://github.com/slimgroup/StencilConv

StencilConv

StencilConv:

- Open-source MIT license
- Leverages Devito
- MPI domain decomposition
- Low memory
- https://github.com/slimgroup/StencilConv



Observations

DSLs with proper abstractions

- manage complexity
- increases rate of innovation
- flexibility w.r.t. hardware (Intel, AMD, ARM, NVIDIA, etc.)

Stencil-based computations

- low memory imprint
- allows for ultra large images/volumes

Bottom line

Reliance on dense computations on CPUs & GPUs

- impedes scale up
- patched-based ML leads to suboptimal performance
- unwarranted use of (large) GPUs
- unwanted production of CO2

Automatic code generation w/ just-in-time Devito compiler

- leverage sophisticated parallelization w/o hardcoding
- provides improved flexibility w.r.t. hardware
- improved memory use & comp. efficiency

Challenge – local minima

ML w/ CNNs & FWI both suffer from parasitic local minima

- poor starting model for the velocity in seismic (cycle skips)
- poor initialization for NNs

SGD used as a remedy

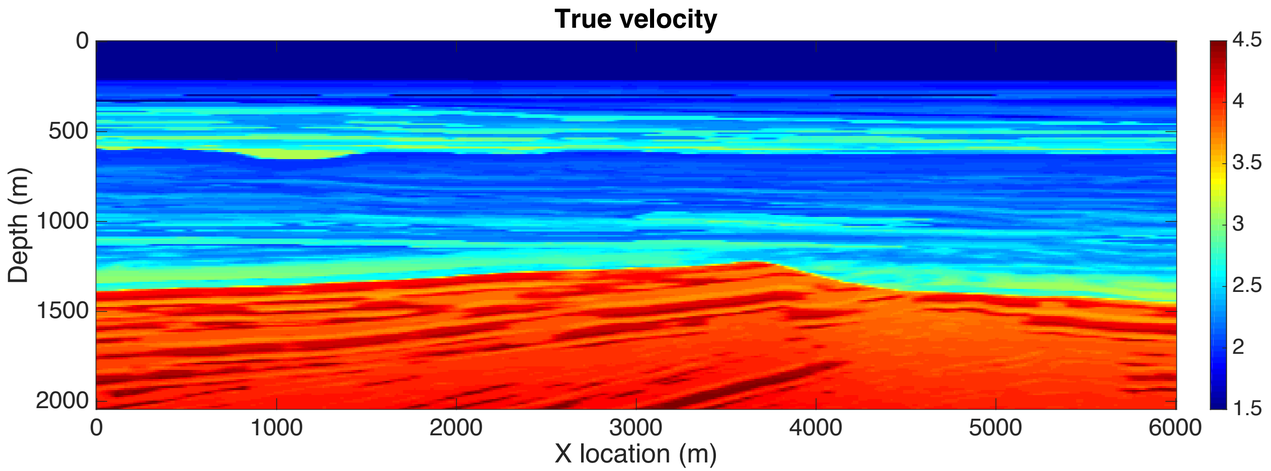
- very limited success for FWI
- reasonable success for ML albeit no guarantees on generalizability
- slow training

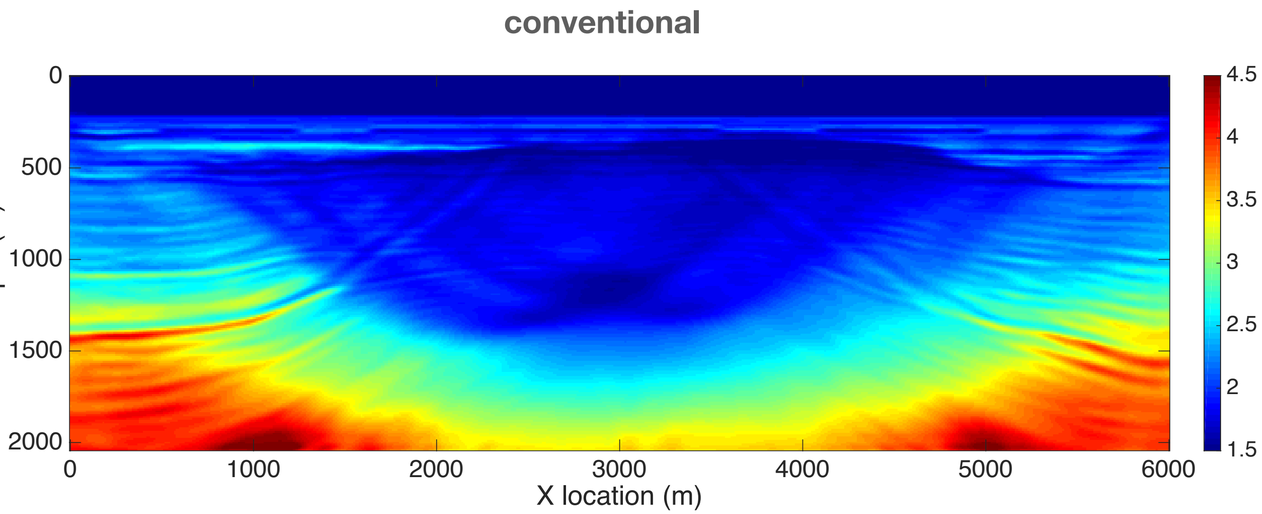
In seismic, more success w/ extensions…

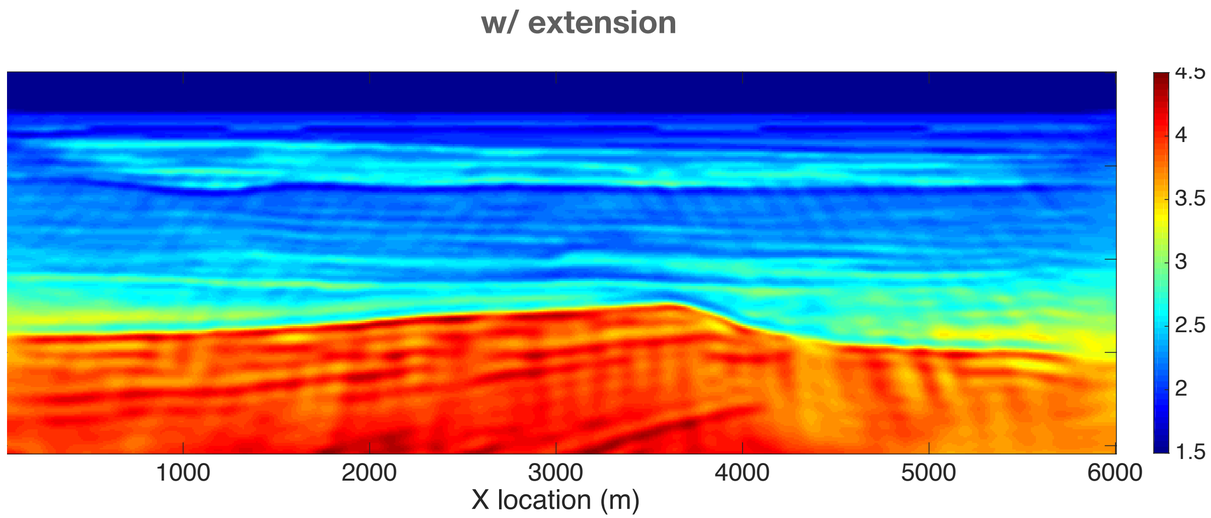
van Leeuwen & FJH "Mitigating local minima in full-waveform inversion by expanding the search space." *GJI* 195.1 (2013): 661-667.
van Leeuwen & FJH "A penalty method for PDE-constrained optimization in inverse problems." Inverse Problems 32.1 (2015): 015007.
Huang,et.al. "Source-independent extended waveform inversion based on space-time source extension: Frequency-domain implementation." Geophysics (2018)

Local minima

- poor starting model $\Longrightarrow$ cycle skipping
- extremely poor inversion results
- **remedies w/ extensions lead to "provable" convexification**







Loss functions

FWI objective:

$$\min_{\mathbf{m}} \frac{1}{2} \sum_{i=1}^{n_s} \|\mathbf{F}[\mathbf{m}]\mathbf{q}^{(i)} - \mathbf{d}^{(i)}\|_2^2$$

- training pairs source & collected data $\{\mathbf{q}^{(i)}, \mathbf{d}^{(i)}\}_{i=1}^{n_s}$
- linear in sources $\mathbf{q}^{(i)}$ & nonlinear in medium $\mathbf{m}$
- fit data at receivers for complete time interval $[0, T]$ w/ $T = n_t \times \Delta t$
- suffers from parasitic local minima (unrealistic images)

Loss functions

ML objective:

$$\min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \frac{1}{2n_d} \sum_{i=1}^{n_d} \ell\left(\mathbf{F}_T[\mathbf{x}^{(i)}; \mathbf{w}],\ \mathbf{y}^{(i)}\right)$$

- training pairs images & labels $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{n_d}$
- nonlinear in images $\mathbf{x}^{(i)}$ and network weights $\mathbf{w}$
- fit labels at "end time" T only
- may suffer from bad minima (do not generalize well)

Entropy-SGD: Biasing Gradient Descent Into Wide Valleys
Pratik Chaudhari Journal of Statistical Mechanics: Theory and Experiment, Volume 2019, December 2019

Entropy–SGD

Algorithm 1: Entropy-SGD algorithm

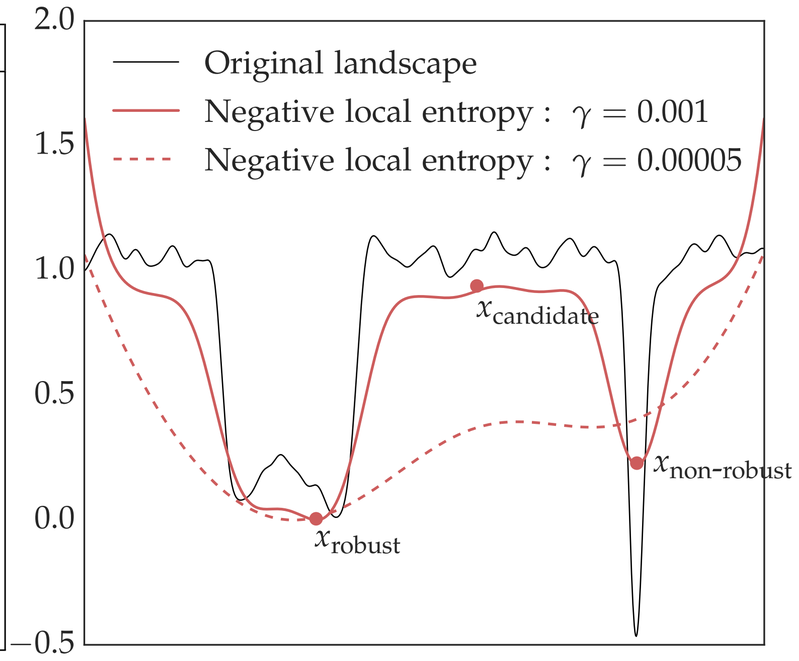
Input : current weights x, Langevin iterations L
Hyper-parameters: scope γ, learning rate η, SGLD step size η'

// SGLD iterations;

$x', \mu \leftarrow x$;
for $\ell \leq L$ **do**
  $\Xi^\ell \leftarrow$ sample mini-batch;
  $dx' \leftarrow \frac{1}{m} \sum_{i=1}^{m} \nabla_{x'} f\left(x'; \xi_{\ell_i}\right) - \gamma\,(x - x')$;
  $x' \leftarrow x' - \eta'\, dx' + \sqrt{\eta'}\, \varepsilon\, \mathrm{N}(0, \mathrm{I})$;
  $\mu \leftarrow (1-\alpha)\mu + \alpha\, x'$;

// Update weights;

$x \leftarrow x - \eta\, \gamma\, (x - \mu)$



- connected to elastic-averaging SGD
- shapes objective into wider valleys
- corresponds to Langevin dynamics known to be slow…

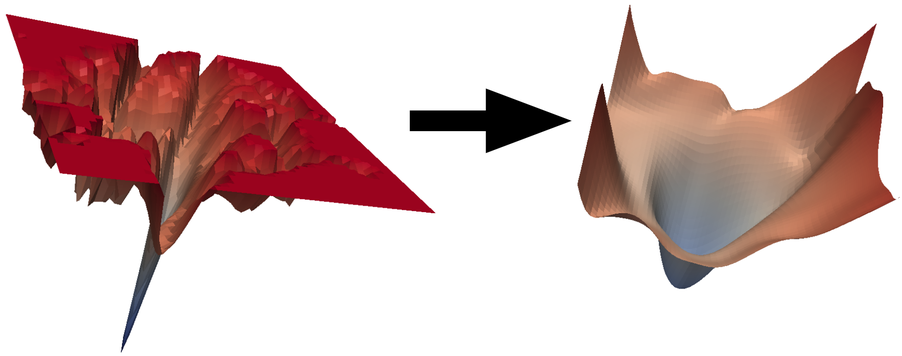
$$\min_{x^1,\ldots,x^p,\tilde{x}} \sum_{i=1}^{p} \mathbb{E}\left[f\left(x^i, \xi^i\right)\right] + \frac{\rho}{2}\left\|x^i - \tilde{x}\right\|^2$$

Foret, Pierre, et al. "Sharpness-Aware Minimization for Efficiently Improving Generalization." *arXiv preprint arXiv:2010.01412* (2020).

Optimize for wide minima

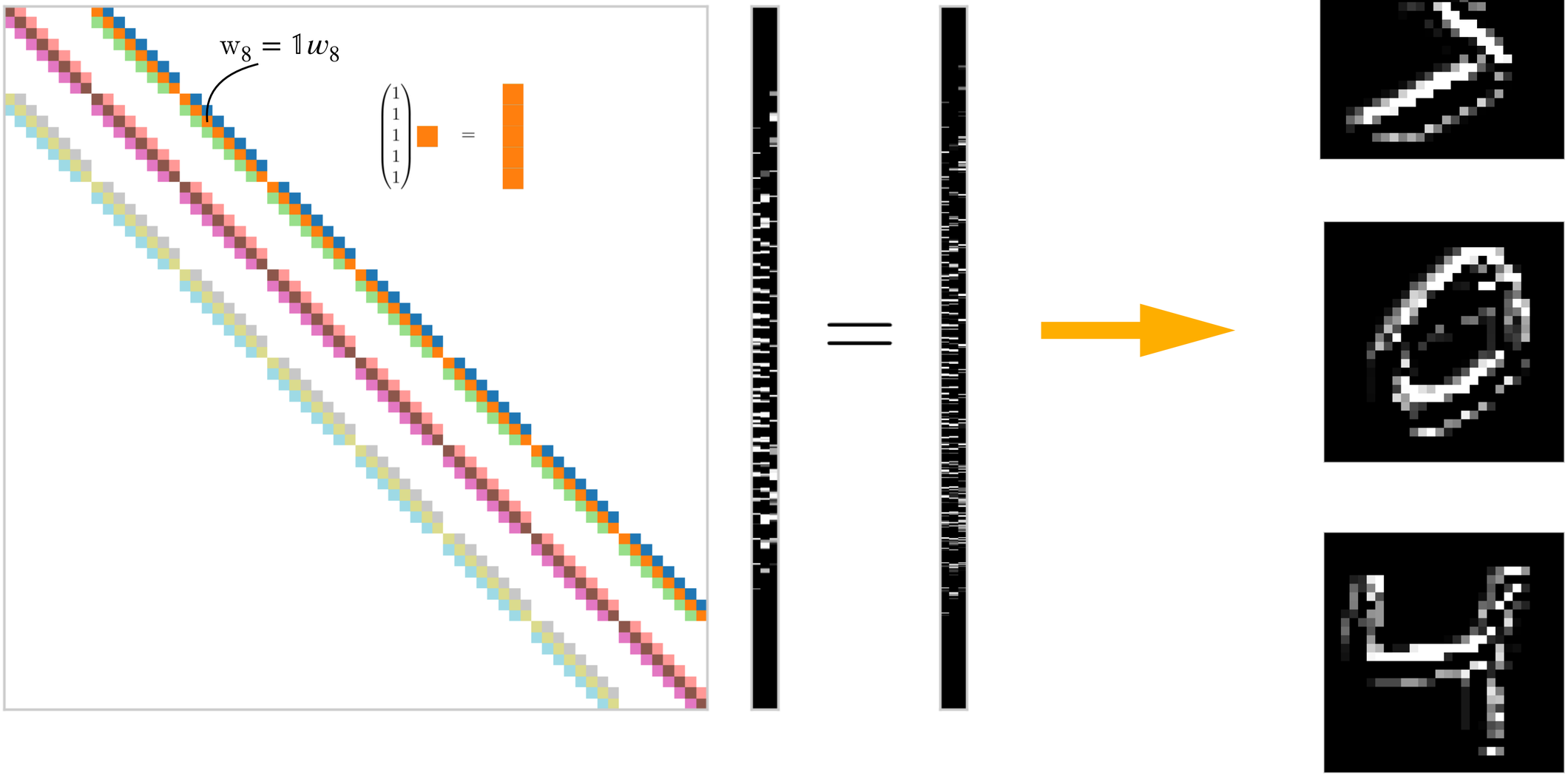
Penalize sharpness local minima:

$$\min_{\mathbf{w}} \mathcal{L}^{SAM}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \quad \text{where} \quad \mathcal{L}^{SAM}(\mathbf{w}) \triangleq \max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} \mathcal{L}(\mathbf{w} + \boldsymbol{\epsilon})$$

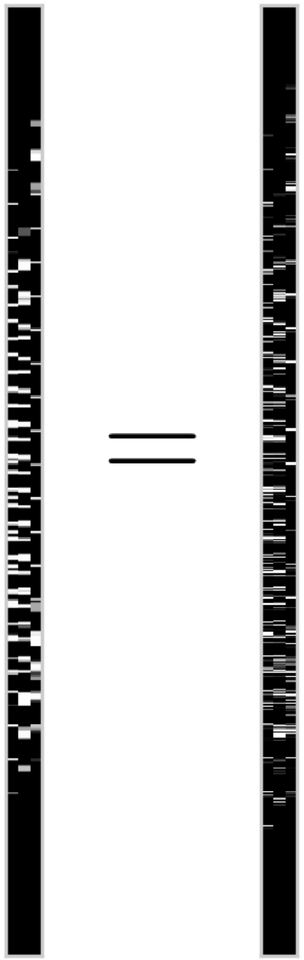


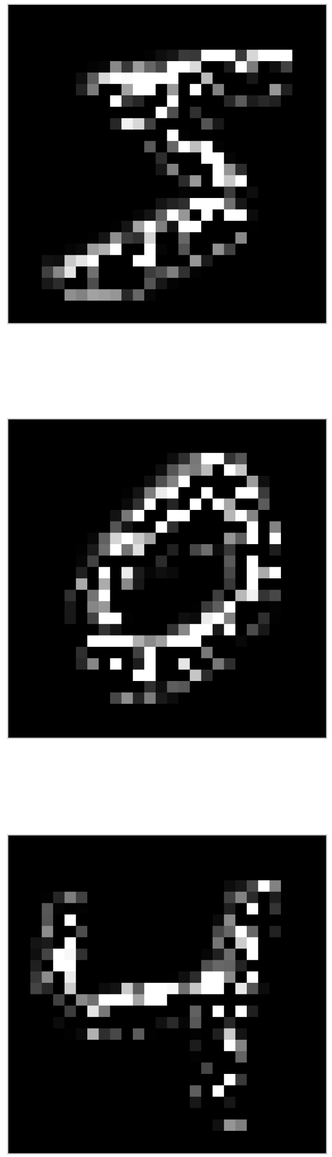
- penalizes bad narrow minima
- extra gradient calculation for all network weights $\mathbf{w}$

CNN as matvec



Generalized convolution – middle ground dense & conv layers





Symes, William W. "Migration velocity analysis and waveform inversion." *Geophysical prospecting* 56.6 (2008): 765-790.

Extended convolutions

Motivated by seismic coefficient-operator extensions

$$\mathrm{diag}(\mathbf{m})\nabla^2 \cdot \quad \Longrightarrow \quad \underline{\mathbf{M}}\nabla^2 \cdot$$

$$\delta\mathbf{m} = \sum_t \mathrm{diag}(\mathbf{u}(t)\mathbf{v}^\top(t)) \in \mathbb{R}^N \quad \Longrightarrow \quad \delta\underline{\mathbf{M}} = \sum_t \mathbf{u}(t)\mathbf{v}^\top(t) \in \mathbb{R}^{N\times N}$$

Extend convolution

$$\mathbf{W}\cdot = \left(\sum_{i=1}^{n_w} \mathrm{diag}(\mathbf{w}_i)\mathbf{T}_{k(i)}\right)\cdot \quad \Longrightarrow \quad \underline{\mathbf{W}}\cdot = \left(\sum_{i=1}^{n_w} \mathrm{diag}(\underline{\mathbf{w}}_i)\mathbf{T}_{k(i)}\right)\cdot$$

$$\mathbf{w}_i = \mathbb{1}w_i \quad \Longrightarrow \quad \underline{\mathbf{w}}_i \in \mathbb{R}^N$$

- non-stationary convolution
- **larger search space**
- **can fit the residuals**

Golub, Gene, and Victor Pereyra. "Separable nonlinear least squares: the variable projection method and its applications." *Inverse problems* (2003).
Aravkin, Aleksandr Y., and Tristan Van Leeuwen. "Estimating nuisance parameters in inverse problems." Inverse Problems 28.11 (2012).
van Leeuwen & FJH "A penalty method for PDE-constrained optimization in inverse problems." Inverse Problems 32.1 (2015).

Variable projection

Minimize

$$\min_{\{w_i,\underline{\mathbf{w}}_i\}_{i=1}^{n_w}} f\left(\underline{\mathbf{W}}\mathbf{X}\right) + \frac{\lambda^2}{2}\|\mathbf{W} - \underline{\mathbf{W}}\|_F^2$$

by taking gradient step on

$$\min_{\{w_i\}_{i=1}^{n_w}} \|\mathbf{W} - \widehat{\underline{\mathbf{W}}}\|_F^2$$

where

$$\{\widehat{\underline{\mathbf{w}}}_i\}_{i=1}^{n_w} \triangleq \arg\min_{\{\underline{\mathbf{w}}_i\}_{i=1}^{n_w}} f\left(\underline{\mathbf{W}}\mathbf{X}\right) + \frac{\lambda^2}{2}\|\mathbf{W} - \underline{\mathbf{W}}\|_F^2$$

yielding

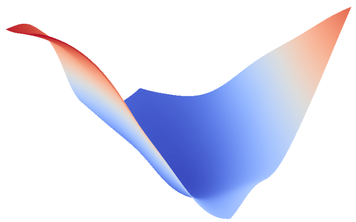
$$\delta\widehat{\underline{w}}_i = N^{-1}\mathbb{1}^\top\widehat{\underline{\mathbf{w}}}_i, \qquad \delta\widehat{\underline{w}}_i \to \delta w_i \quad \text{as} \quad \lambda \to \infty$$

Hao Li et al., Visualizing the Loss Landscape of Neural Nets, Advances in Neural Information Processing Systems, 2018.

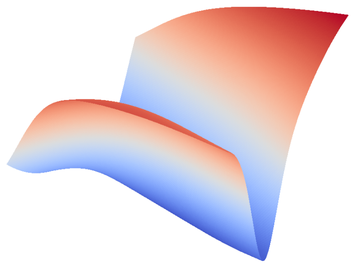
Extended loss function landscape

highly non-convex

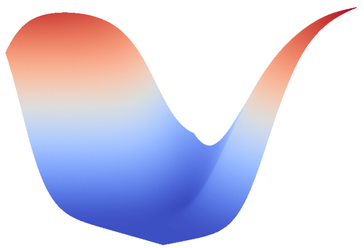
Conventional loss landscape



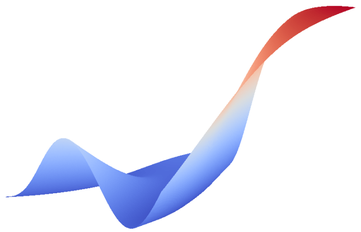
Conventional loss landscape



Conventional loss landscape



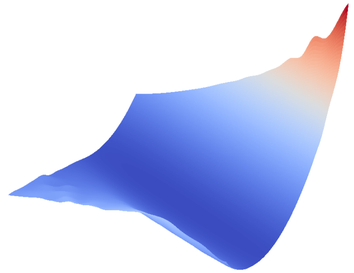
Conventional loss landscape



tentatively quadratic

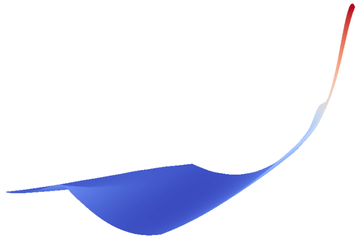
Extended loss landscape

Extended loss landscape

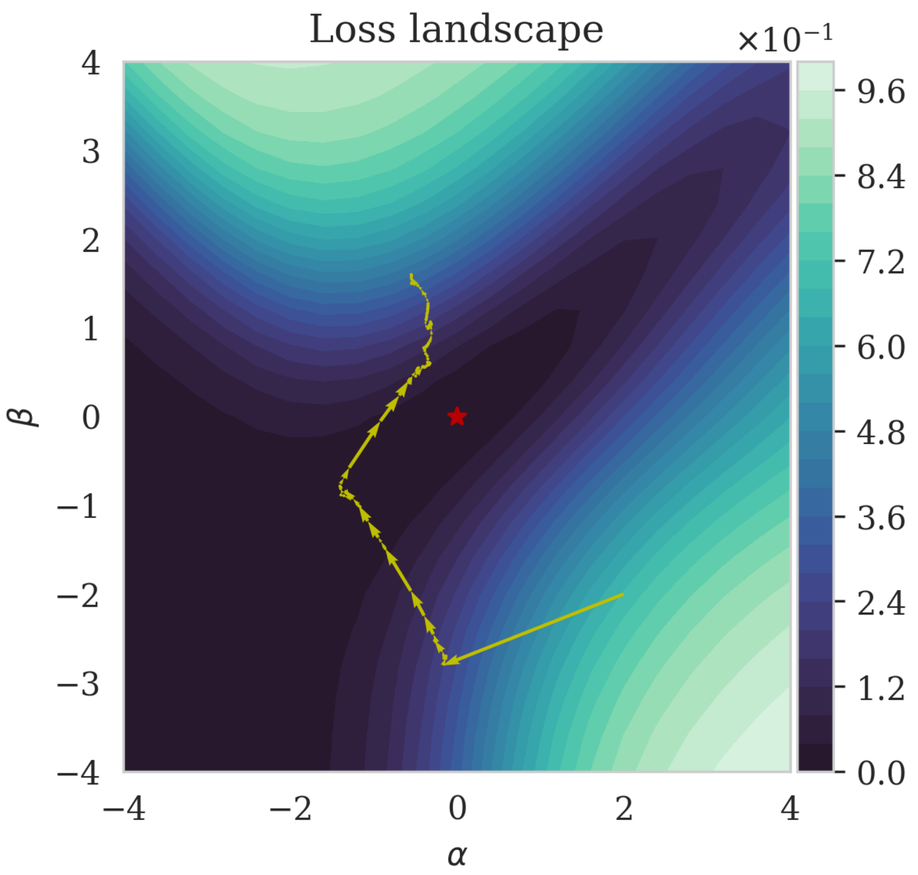


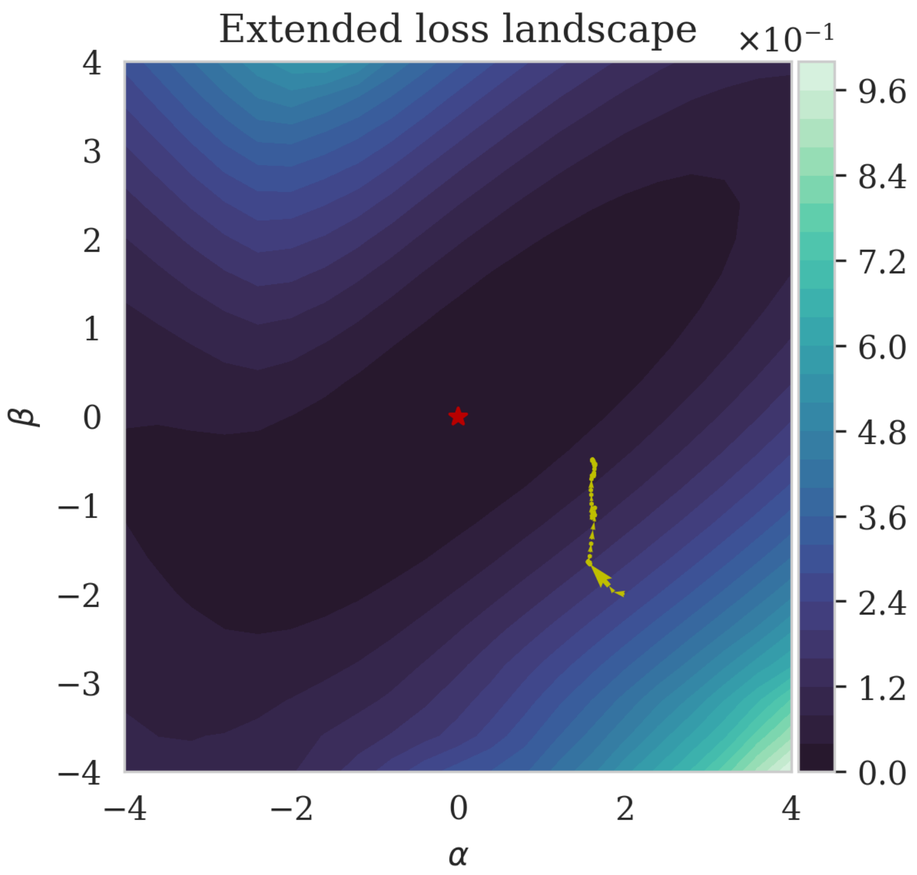
Extended loss landscape

Extended loss landscape

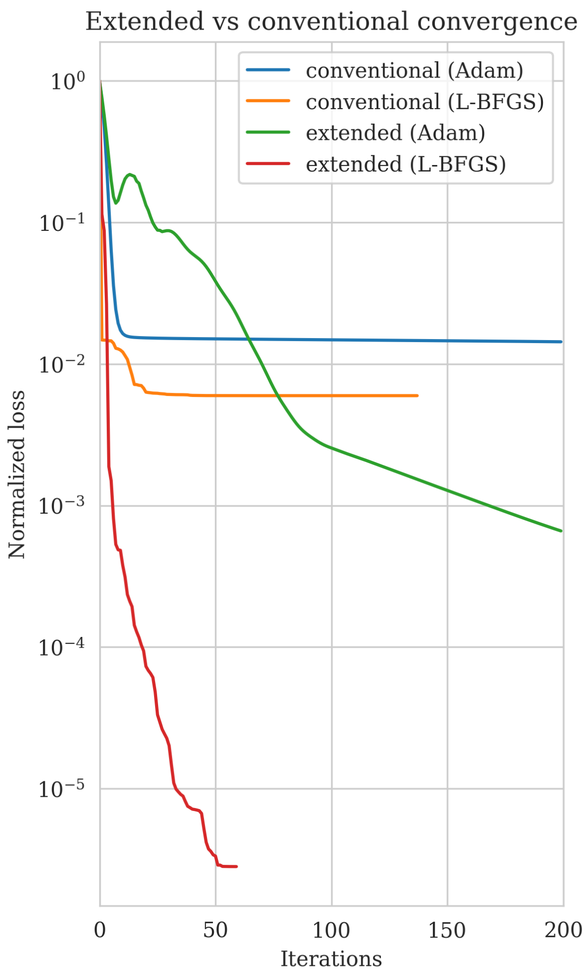


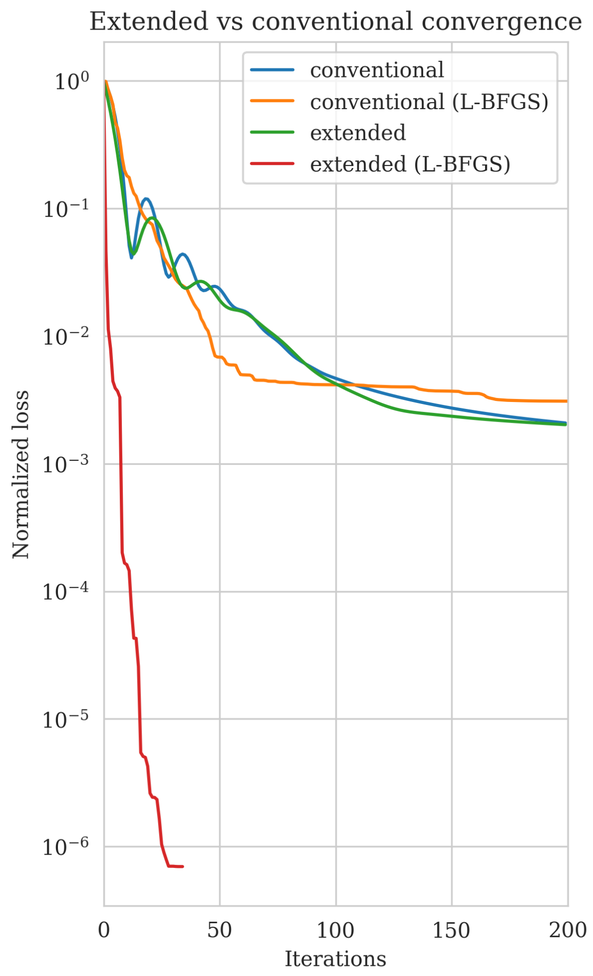
l-BFGS on regular & extended landscape





l-BFGS extended landscape





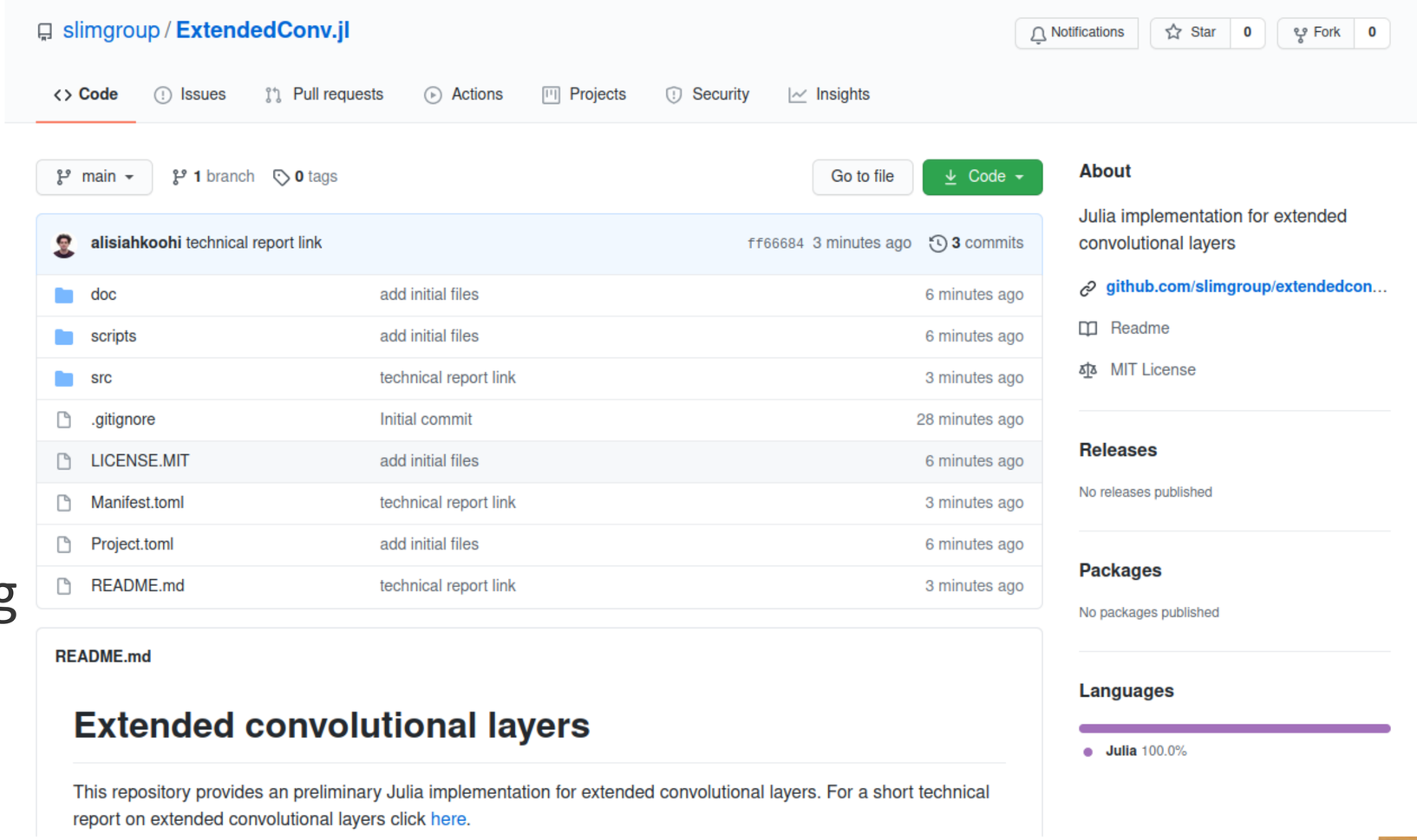
- indications converge faster to solution
- need to offset costs of variable projection

Code

ExtendedConv.jl:

Preliminary CPU codes for training w/ extended convolutional layers

- open-source MIT license
- Julia implementation
- https://github.com/slimgroup/ExtendedConv.jl



Observations

There are indications that the extension

- mollifies the loss function landscape
- leads to faster training possibly w/ fewer data
- possibly yields solutions less prone to bad local minima
- generalize better

When able to offset cost of extension w/ gain in convergence

- more cost effective training
- better performance overall

Bottom line

Training of DNNs relies on

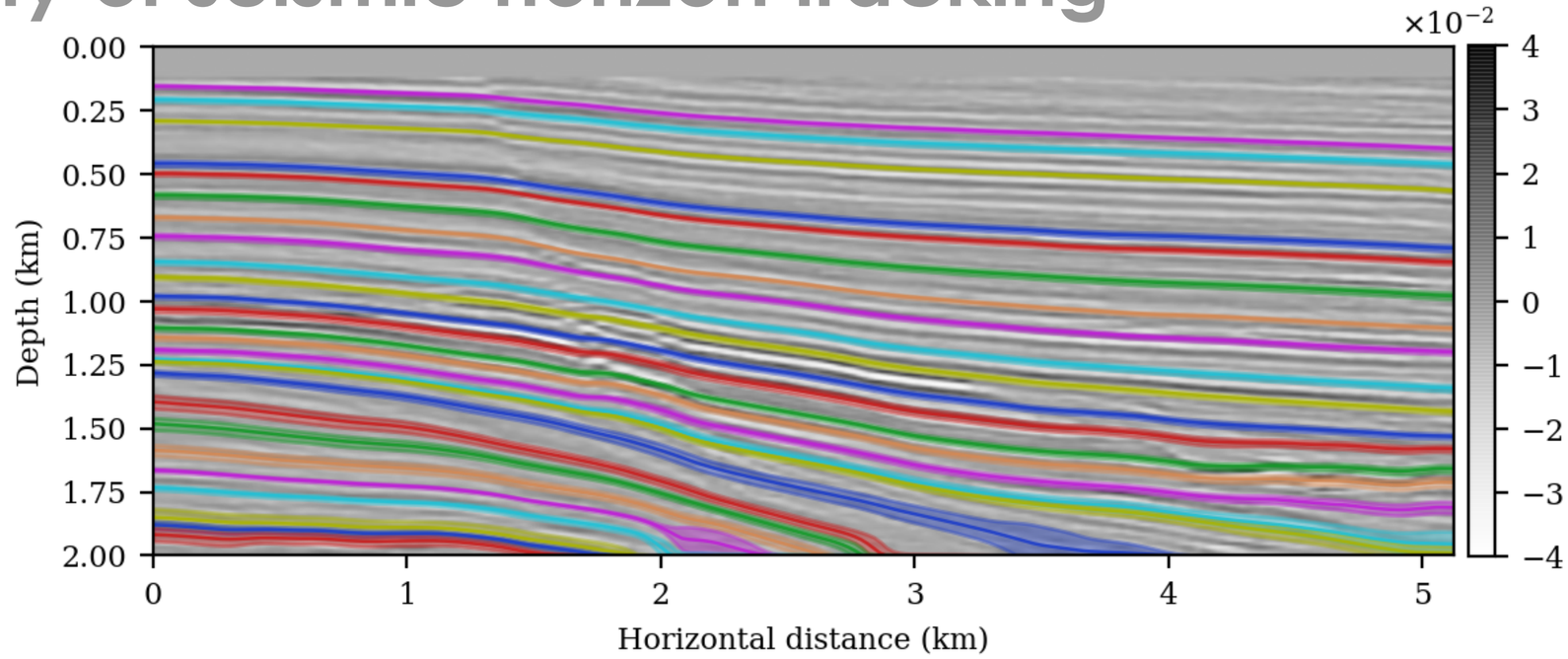
- computationally expensive stochastic optimization
- fundamentally slow “annealing” type of argument

Recent work has shown that fundamental changes in loss function

- may lead to more favorable optimization conditions
- absolute must when scaling to large problem sizes

Siahkoohi, Ali, et. al. "Uncertainty quantification in imaging and automatic horizon tracking—a Bayesian deep-prior based approach.", *2020*.

Uncertainty of seismic horizon tracking



Uncertainty in horizon tracking due uncertainties in imaging and control points

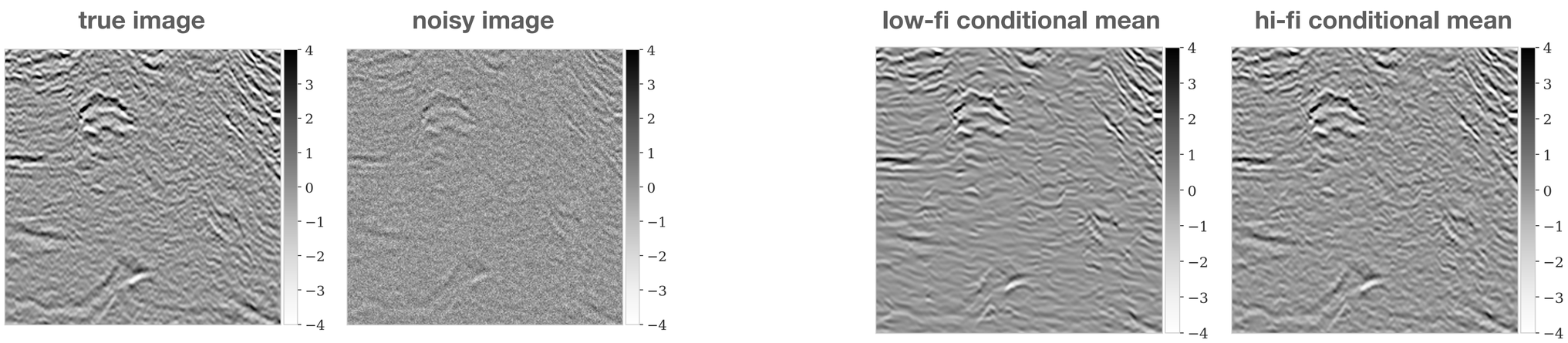
Combination of

- stochastic Langevin dynamics
- deep priors w/ generative DNNs
- exposed complete lack of scaling

First instance of propagating uncertainty to task...

Siahkoohi, Ali, et al. "Preconditioned training of normalizing flows for variational inference in inverse problems." *arXiv preprint arXiv:2101.03709* (2021).

Preconditoned variational inference



Combination of

- normalizing flows implemented via invertible NNs (iNNs)
- data-driven variational inference $\Longrightarrow$ iNN captures low-fidelity prior & posterior
- physics-based variational inference $\Longrightarrow$ high-fi posterior via transfer training iNN

Potential to scale uncertainty quantification...

Siahkoohi, Ali, et al. "Preconditioned training of normalizing flows for variational inference in inverse problems." *arXiv preprint arXiv:2101.03709* (2021).

Data-driven variational inference

$$\arg\max_{\phi} \mathbb{E}_{\mathbf{y},\mathbf{x}\sim\widehat{p}_{y,x}(\mathbf{y},\mathbf{x})} \left[\log p_\phi(\mathbf{y},\mathbf{x})\right]$$

$$= \arg\min_{\phi} \mathbb{E}_{\mathbf{y},\mathbf{x}\sim\widehat{p}_{y,x}(\mathbf{y},\mathbf{x})} \left[\frac{1}{2}\left\|G_\phi(\mathbf{y},\mathbf{x})\right\|^2 - \log\left|\det \nabla_{y,x} G_\phi(\mathbf{y},\mathbf{x})\right|\right]$$

$$\approx \arg\min_{\phi} \frac{1}{n}\sum_{i=1}^{n} \left[\frac{1}{2}\left\|G_\phi(\mathbf{y}_i,\mathbf{x}_i)\right\|^2 - \log\left|\det \nabla_{y,x} G_\phi(\mathbf{y}_i,\mathbf{x}_i)\right|\right]$$

NF, $G_\phi : \mathcal{Y}\times\mathcal{X} \to \mathcal{Z}_y \times \mathcal{Z}_x$

$$G_\phi(\mathbf{y},\mathbf{x}) = \begin{bmatrix} G_{\phi_y}(\mathbf{y}) \\ G_{\phi_x}(\mathbf{y},\mathbf{x}) \end{bmatrix}, \quad \phi = \begin{bmatrix} \phi_y \\ \phi_x \end{bmatrix}$$

conditional sampling: $G_{\phi_x}^{-1}(G_{\phi_y}(\mathbf{y}),\mathbf{z}) \sim p_{\text{post}}(\boldsymbol{x} \mid \boldsymbol{y})$ for $\mathbf{z} \sim \mathrm{N}(\mathbf{0},\boldsymbol{I})$

posterior density estimation: $p_G(\boldsymbol{x} \mid \boldsymbol{y}) = p_z\big(G_{\phi_x}(\boldsymbol{y},\boldsymbol{x})\big)\left|\det \nabla_x G_{\phi_x}(\boldsymbol{y},\boldsymbol{x})\right|.$

Ricardo Baptista, Olivier Zahm, and Youssef Marzouk. *An adaptive transport framework for joint and conditional density estimation*. 2020. arXiv: 2009.10303 [stat.ML].

Jakob Kruse et al. "HINT: Hierarchical Invertible Neural Transport for Density Estimation and Bayesian Inference". In: *arXiv preprint arXiv:1905.10687* (2019).

Combination of

- normalizing flows implemented via invertible NNs (iNNs)
- data-driven variational inference $\Longrightarrow$ iNN captures low-fidelity prior & posterior
- physics-based variational inference $\Longrightarrow$ high-fi posterior via transfer training iNN

Potential to scale uncertainty quantification...

https://github.com/slimgroup/FastApproximateInference.jl https://github.com/slimgroup/InvertibleNetworks.jl

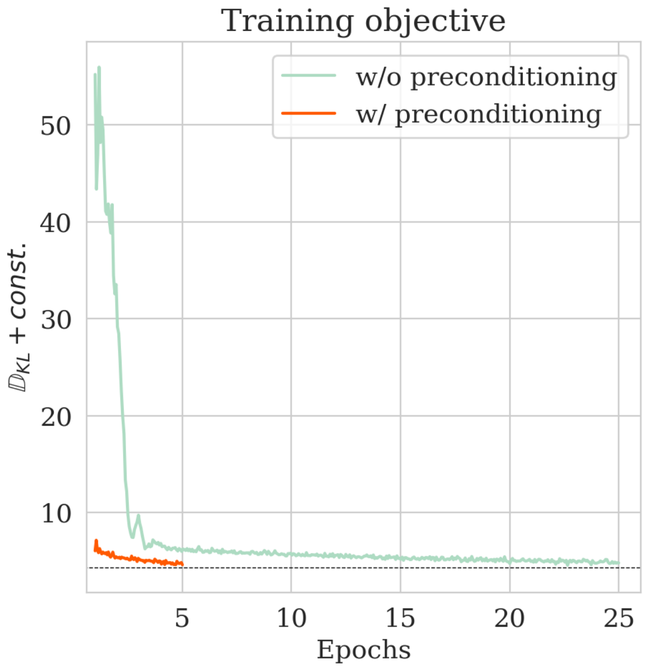
Preconditioned physics-based variational inference

$$\min_{\phi_x} \mathbb{E}_{\mathbf{z}\sim\mathrm{N}(\mathbf{0},\mathbf{I})}\left[\frac{1}{2\sigma^2}\left\|F\big(T_{\phi_x}(\mathbf{z})\big)-\mathbf{y}\right\|_2^2 - \log p_G\big(T_{\phi_x}(\mathbf{z})\big) - \log\left|\det\nabla_z T_{\phi_x}(\mathbf{z})\right|\right]$$

transfer learning:

- corrects for low-fidelity training data, $p_{y,x} \neq p_{y,x}$
- less evaluations of F and $\nabla F^\top$

learned prior density, $p_G(\mathbf{x}) := p_{\mathbf{z}}\big(G_{\phi_x}(\mathbf{y},\mathbf{x})\big)\left|\det\nabla_x G_{\phi_x}(\mathbf{y},\mathbf{x})\right|$

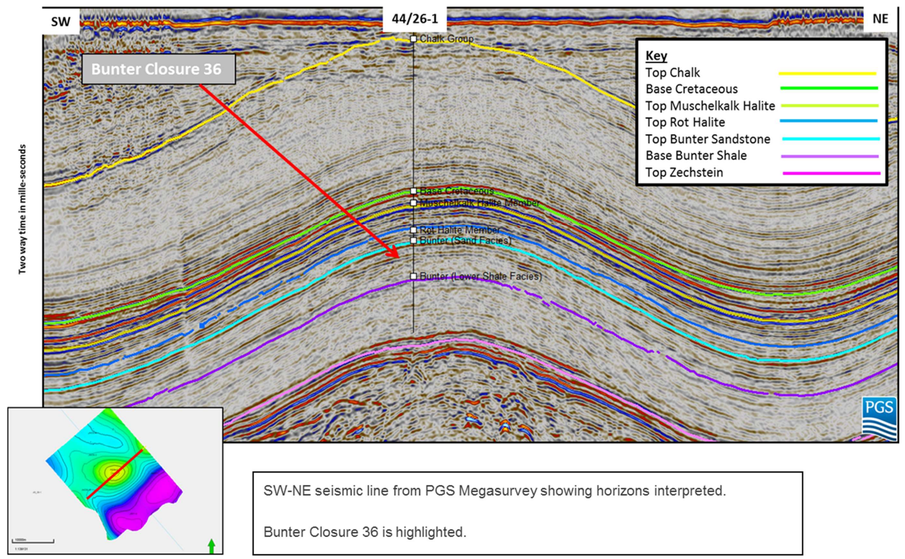


Combination of

- normalizing flows implemented via invertible NNs (iNNs)
- data-driven variational inference $\Longrightarrow$ iNN captures low-fidelity prior & posterior
- physics-based variational inference $\Longrightarrow$ high-fi posterior via transfer training iNN

Potential to scale uncertainty quantification...

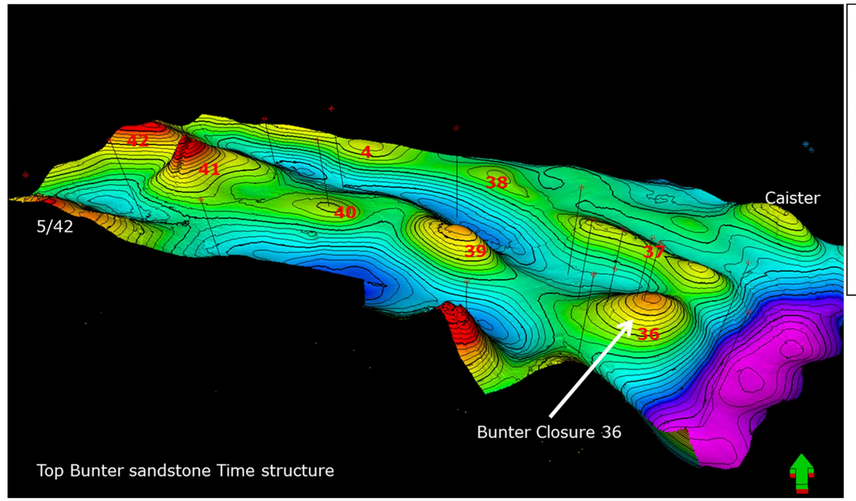
Seismic monitoring CCS – Bunter Sandstone SW North Sea



SW-NE seismic line from PGS Megasurvey showing horizons interpreted.

Bunter Closure 36 is highlighted.

from: Strategic UK CCS Storage Appraisal Project



3D view of Top Bunter Sandstone two-way time structure over the Fairway and Closure 36.

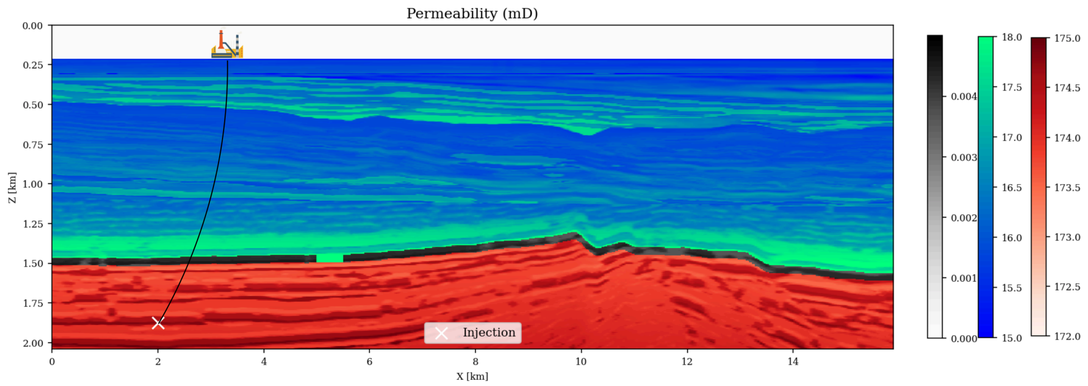
Shows 8 anticlines with the UKSAP numbers plus 5/42 and Caister.

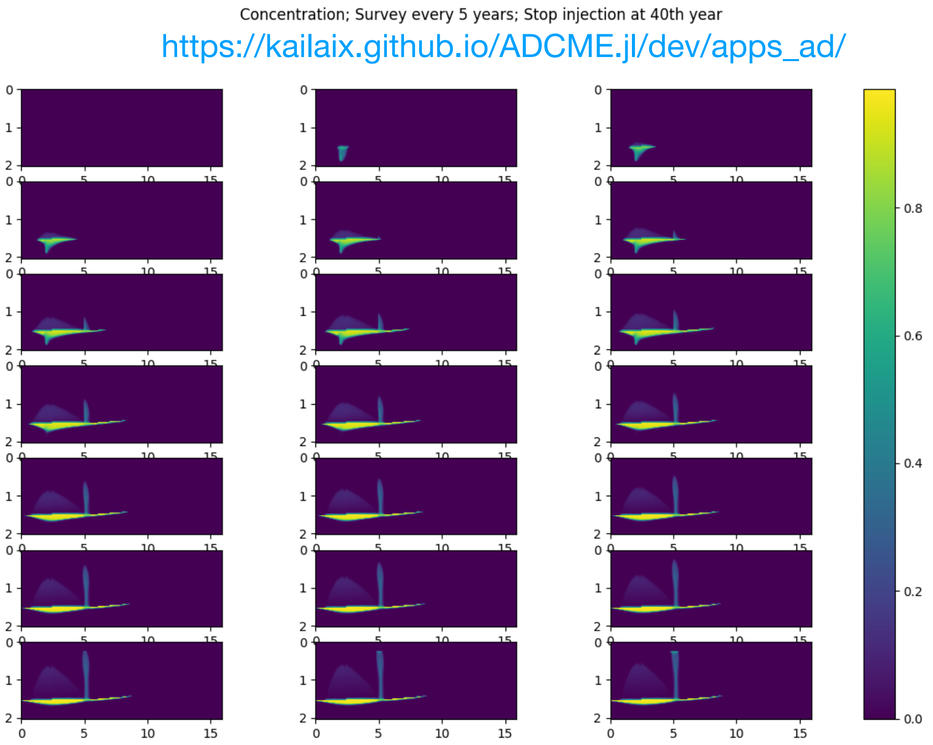
CO2 sequestration into underground marine saline aquifers

- **inject 7Mt/y of CO2 for 40 years ⟺ offset 2.5k big transformer annually**
- monitor CCS w/ time-lapse seismic imaging
- follow the CO2 plume for 100's years

Li, Dongzhuo, et al. "Coupled Time–Lapse Full–Waveform Inversion for Subsurface Flow Problems Using Intrusive Automatic Differentiation." *Water Resources Research* 56.8 (2020): e2019WR027032.
Li, Zongyi, et al. "Fourier neural operator for parametric partial differential equations." arXiv preprint arXiv:2010.08895 (2020).
Kolster, Clea, et al. "The impact of time-varying CO2 injection rate on large scale storage in the UK Bunter Sandstone." International Journal of Greenhouse Gas Control 68 (2018): 77-85.

3D BG Compass model – synthetic proxy





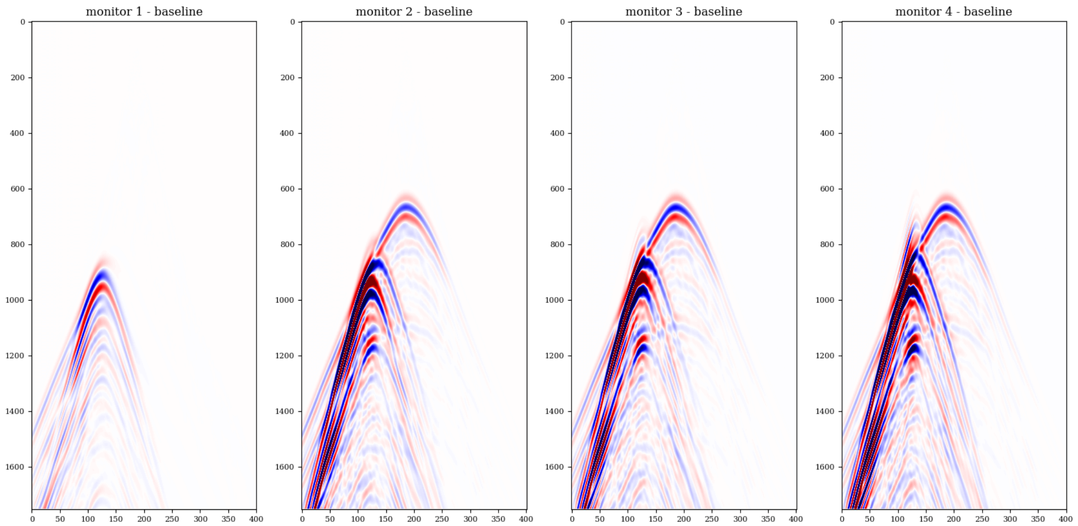
https://kailaix.github.io/ADCME.jl/dev/apps_ad/

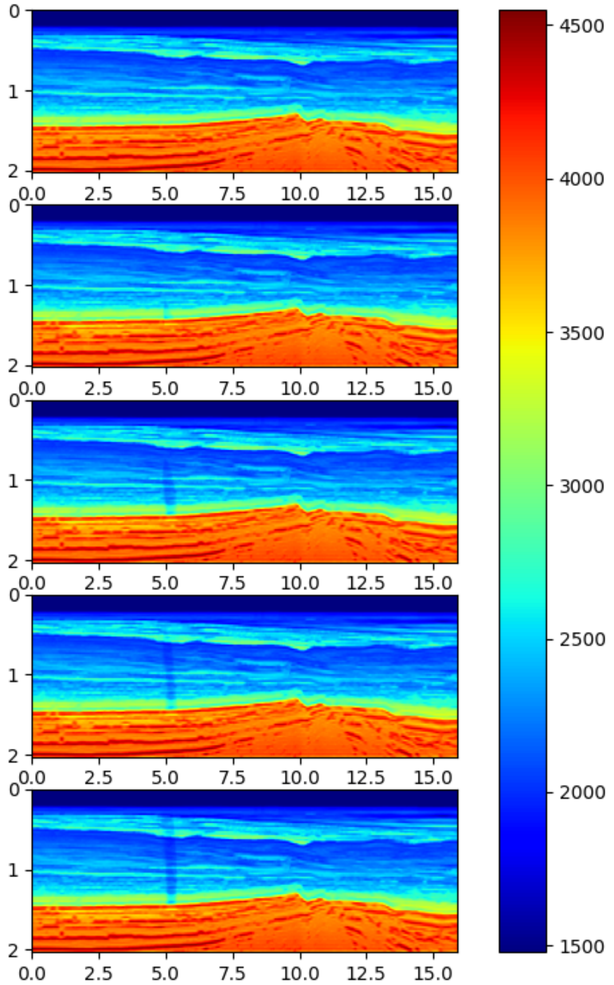
Simulations 2 phase flow

- approximate 2 phase flow PDE w/ Fourier Neural Operators
- monitor w/ time-lapse seismic imaging
- optimize sequestration & lower cost & risk

Li, Dongzhuo, et al. "Coupled Time−Lapse Full−Waveform Inversion for Subsurface Flow Problems Using Intrusive Automatic Differentiation." *Water Resources Research* 56.8 (2020): e2019WR027032.

3D BG Compass model – synthetic proxy

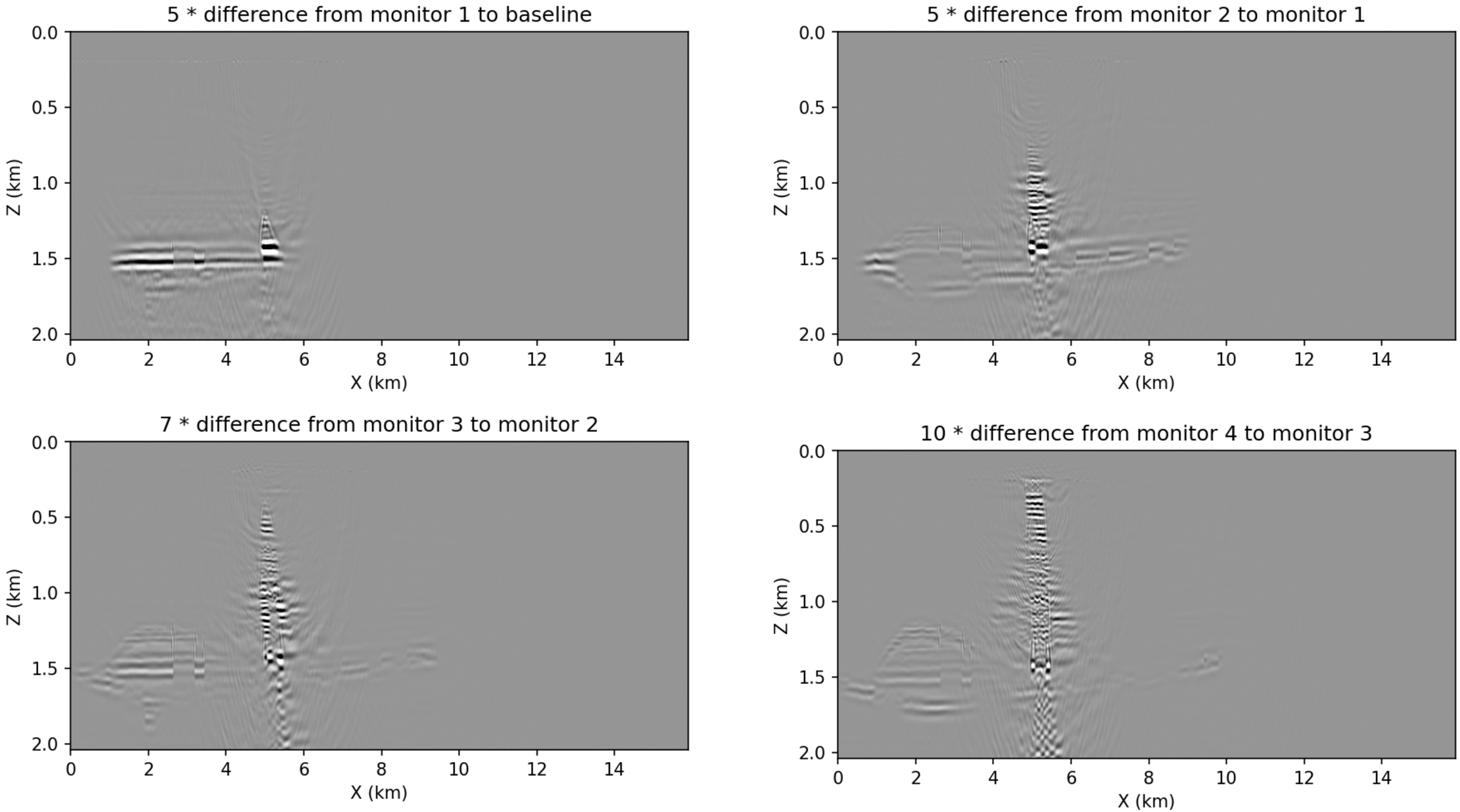




Simulate seismic response every 20 years

https://kailaix.github.io/ADCME.jl/dev/apps_ad/

Ideal difference images



Faculty involved in CCS

Felix J. Herrmann

- GRA Eminent Scholar Chair in Energy
- Director Center for ML for Seismic Industry Partners Program
- Computational imaging

Christopher Jones

- Associate VP Research, 2013-2019
- CO_2 Capture from Air
- CO_2 Conversion to Products

Matthew Realff

- Associate Dir. Strategic Energy Institute
  Direct Air Capture Center (DirACC)
- Multi-scale Design & Optimization
- Life-Cycle Assessment

Conclusions

Research programs @ leading academic institutions

- mixture of tangible & blue sky research
- can produce major cost-saving innovations

Rethink of AI is essential to reduce Carbon footprint

CCS is arguably one of few truly scalable solutions

- major scale up
- cost reduction
- risk mitigation

This is where ML@scale comes in handy… Thank you!