Basic Machine Learning Approaches for the Acceleration of PDE Simulations

H. Ruelmann, S. Turek

TU Dortmund University, Vogelpothsweg 87, 44227 Dortmund,
hannes.ruelmann@math.tu-dortmund.de

Eccomas Thematic Conference
Computational Science and AI in Industry (CSAI)
7th - 9th June 2021, Virtual Conference

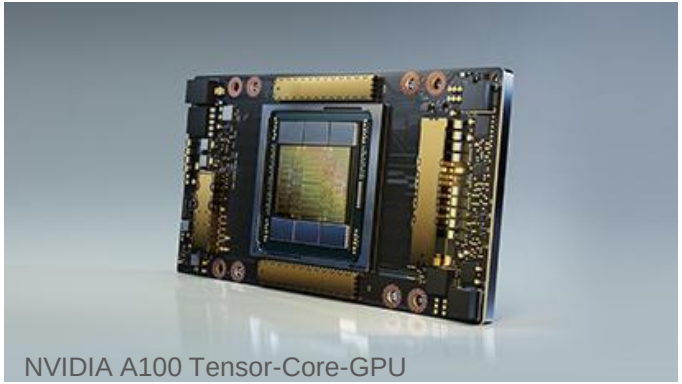
Table of Contents

Neural Networks

- multitude of problems: health, recognition, statistic, logistic, gaming, ...

Neural Networks

- multitude of problems:
  health, recognition, statistic,
  logistic, gaming, ...
- technical revolution
  CPUs, GPUs, TPUs
- good for simulations/
  solving PDEs?


NVIDIA A100 Tensor-Core-GPU


NVIDIA DGX Server (A100)

Pressure-Poisson problem

- Navier-Stokes equations

$$\underbrace{u_t - \nu\Delta u + u \cdot \nabla u}_{S(u)\cdot u} + \nabla p = f$$
$$-\nabla \cdot u = 0$$

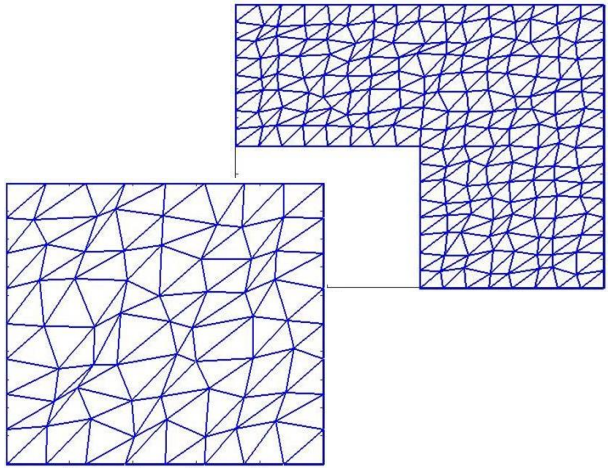
- weak formulation
- FEM discretisation
- time discretisation (theta-scheme)
- decouple u and p: fixpoint solver + Pressure-Poisson problem (+ some magic...)

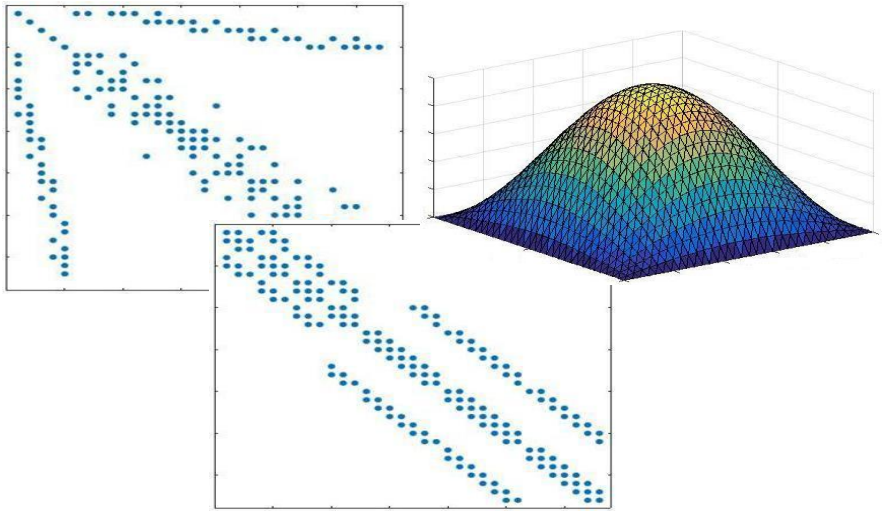
$$P \approx \Delta_h$$

$$S(u) \cdot u = f_{FP} \qquad P \cdot q = \frac{1}{\Delta t} B^T \cdot u$$

Model Problem: Poisson

- find $u : \Omega \to \mathbb{R}$ such that
  $-\Delta u = f \text{ in } \Omega,\ u = 0 \text{ on } \partial\Omega$
- discretize with FEM
  $a_h(u_h, v_h) = b_h(v_h) \quad \forall v_h \in V_h$





calculate system matrix and rhs

$$(A_h)_{ij} = \sum_{m=1}^{N} \int_{K_m} \nabla\phi_j \cdot \nabla\phi_i dx = \sum_{m=1}^{N} A_{ij}^{(m)}$$

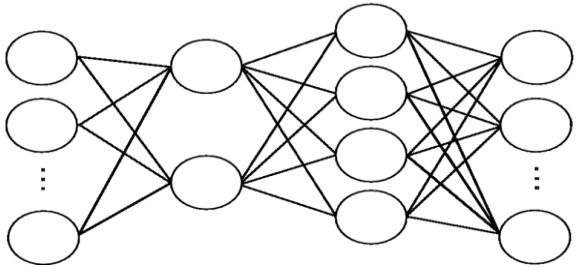
$$(b_h)_i = \int_\Omega f\phi_i dx$$

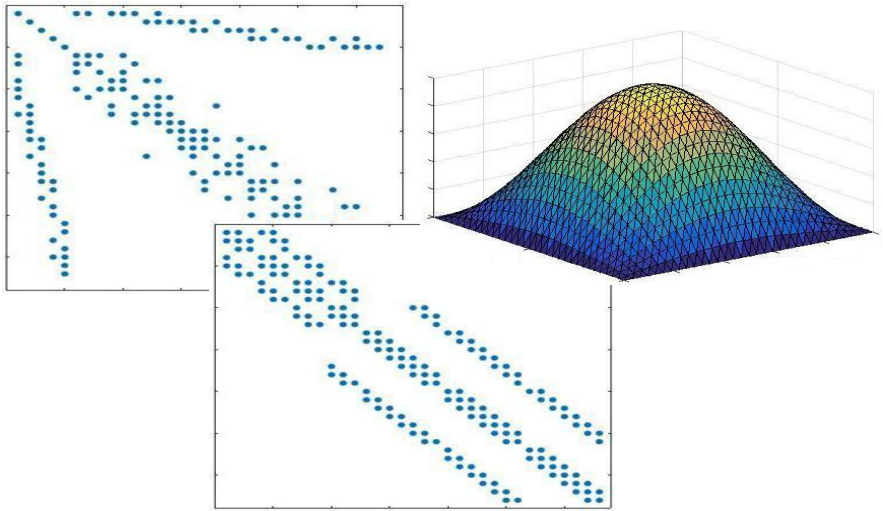
$$\longrightarrow \quad A_h x = b_h$$

Model Problem: Poisson

Richardson Iteration:

$$x^{(k+1)} = x^{(k)} + \omega M(b_h - A_h x^{(k)})$$





calculate system matrix and rhs

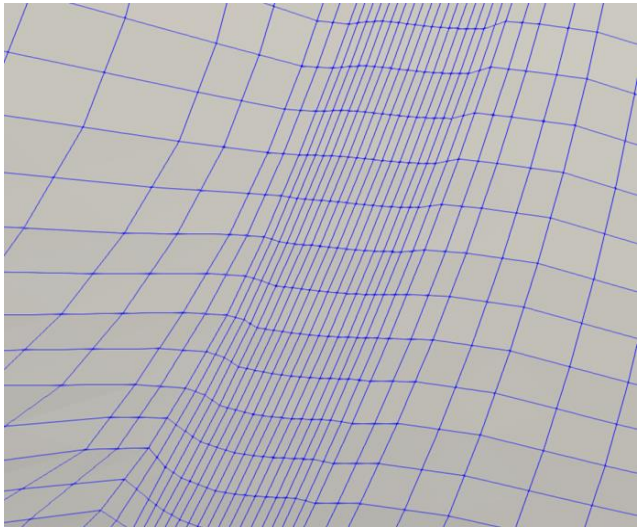
$$(A_h)_{ij} = \sum_{m=1}^{N} \int_{K_m} \nabla\phi_j \cdot \nabla\phi_i dx = \sum_{m=1}^{N} A_{ij}^{(m)}$$

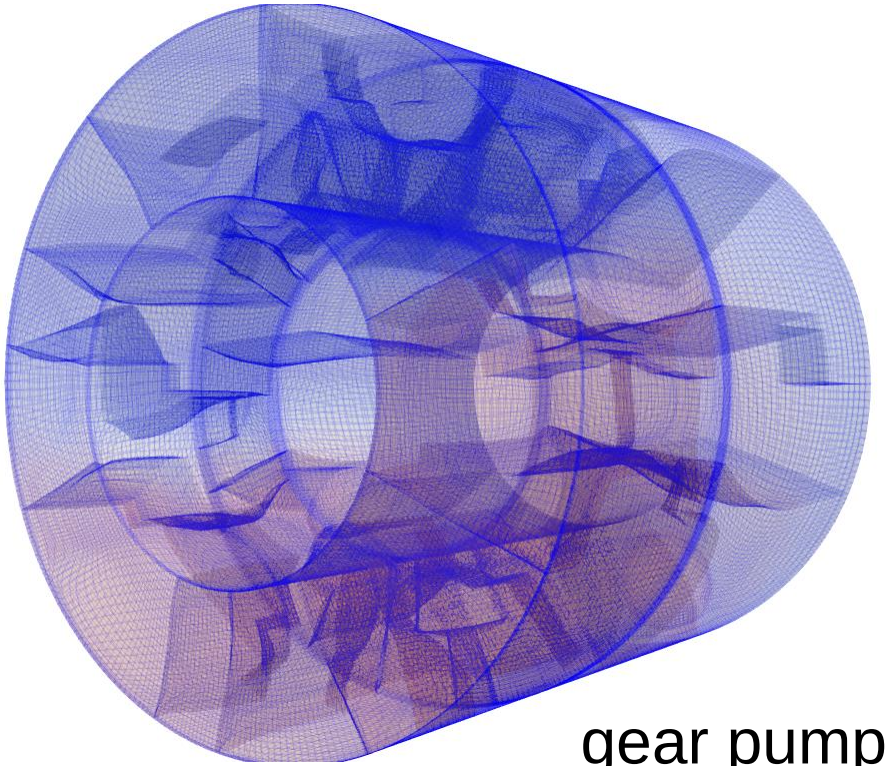
$$(b_h)_i = \int_\Omega f\phi_i dx$$

$$\longrightarrow \quad A_h x = b_h$$

Anisotropies

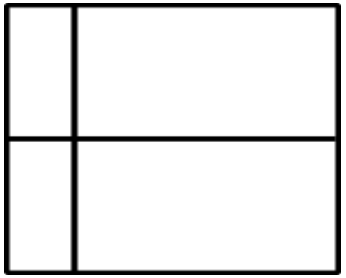
- Real world application
- Complex geometry
- Mesh anisotropies

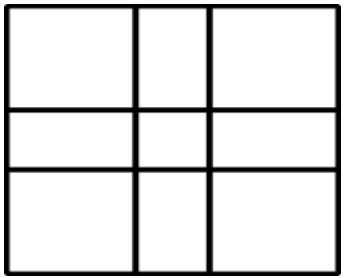




gear pump

Anisotropies





multigrid solver, fixed 8 smoother steps

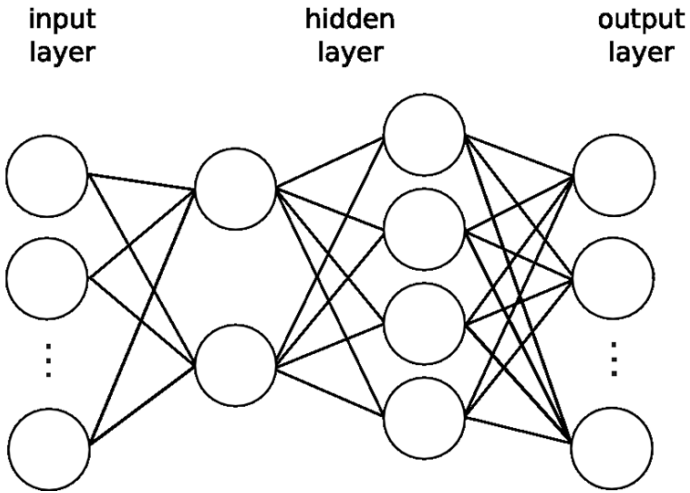
lvl	dofs	Jacobi (0,8)	GS (1.0)	SPAI-1 (1,0)	ILU-0 (0,5)
10	4.198.401	4	4	3	4
9	1.050.625	4	4	3	4
8	263.169	4	4	3	4

lvl	dofs	Jacobi (0,5)	GS (1.0)	SPAI-1 (1,0)	ILU-0 (0,5)	ILU-RCMK
10	2.100.225	109	33	24	26	7
9	525.825	106	32	23	25	7
8	131.841	103	30	22	24	7
7	32.960	98	28	21	23	7
6	8.240	90	25	19	21	6
5	2.060	78	22	17	18	6
4	515	55	16	13	12	6
3	129	35	9	10	8	6

lvl	dofs	Jacobi (0,5)	GS (0,7)	ILU (0,7)	SPAI-1 (1,0)
9	2.362.369	654	370	102	140
8	591.361	619	350	95	130
7	148.225	562	319	85	118
6	37.249	486	289	75	103
5	9.409	377	218	57	80
4	2.401	258	166	34	51
3	625	175	96	20	31

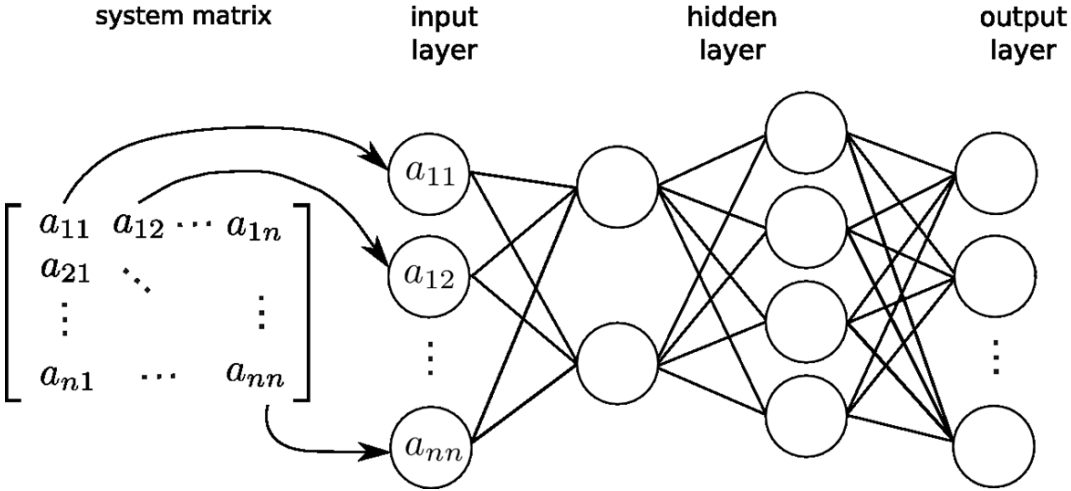
Neural Networks

- How to design NN?
  - → start straightforward and as simple as possible
  - → fully connected feedforward neural network



Neural Networks

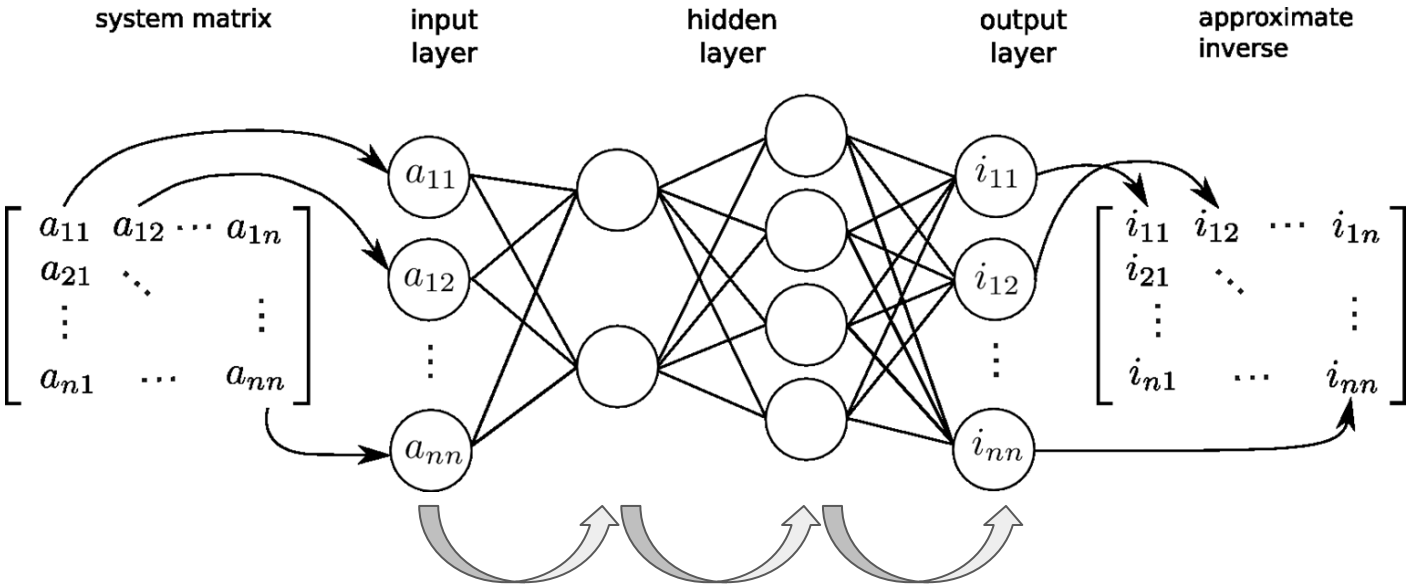
- How to design NN?
  - → start straightforward and as simple as possible
  - → fully connected feedforward neural network



- Input of the NN: matrix entries

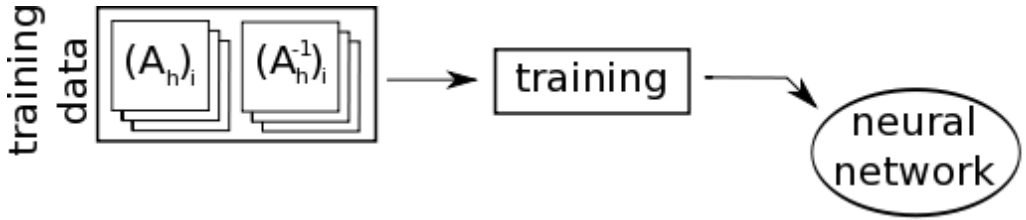
Neural Networks

- How to design NN?
  - → start straightforward and as simple as possible
  - → fully connected feedforward neural network



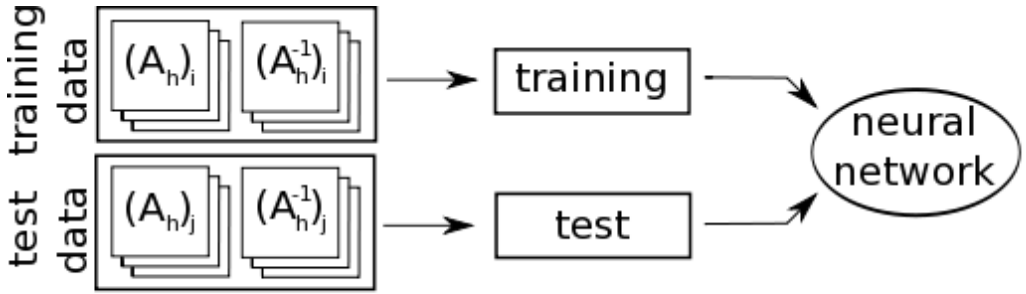
Neural Networks

1. training phase → supervised learning
   → backpropagation



Neural Networks

1. training phase → different dataset
2. test phase → generalization



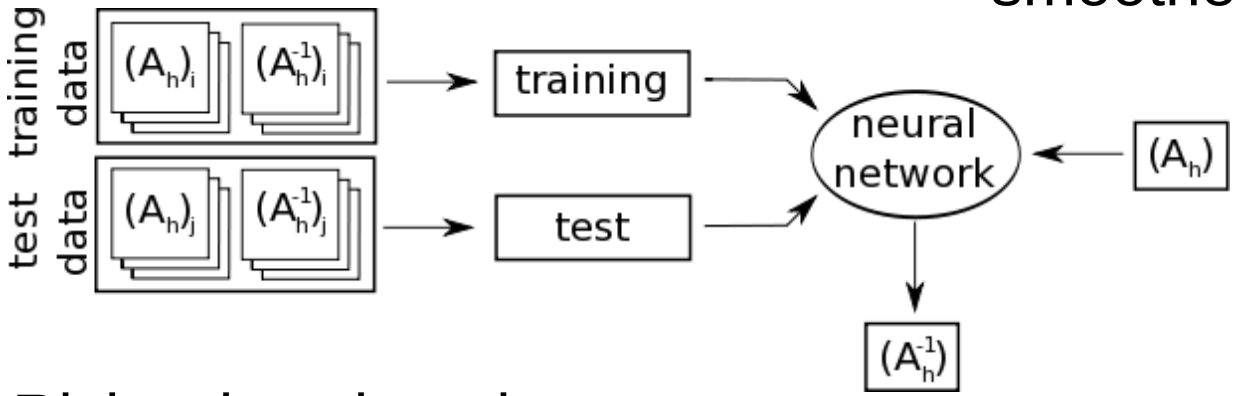
Neural Networks

1. training phase
2. test phase

3. application phase

→ preconditioner

→ smoother



$$A_h x = b_h$$

Richardson iteration:

$$x^{(k+1)} = x^{(k)} + \omega M(b_h - A_h x^{(k)})$$

Numerical experiments

condition number

		# it			it down	κ	
lvl	n	$J_{\omega=0.7}$	GS	NN		before	after
2	9	49	17	8	2.13	5.9	1.6
3	49	273	95	21	4.52	29.8	2.9
4	225	1 323	463	66	7.02	127.3	7.8
5	961	5 879	2 057	39	52.74	516.0	23.4

3 layered NN

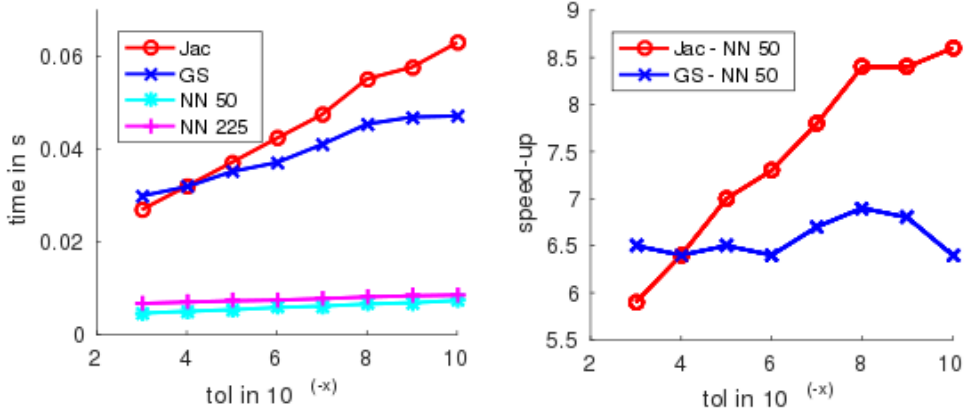
reduction factor (GS vs NN)

- first experiments with NN prototype
- adjust training parameters
- batch vs. online learning

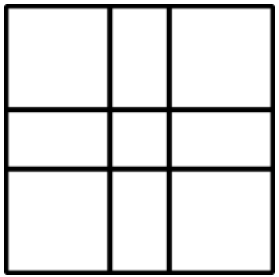
Numerical experiments

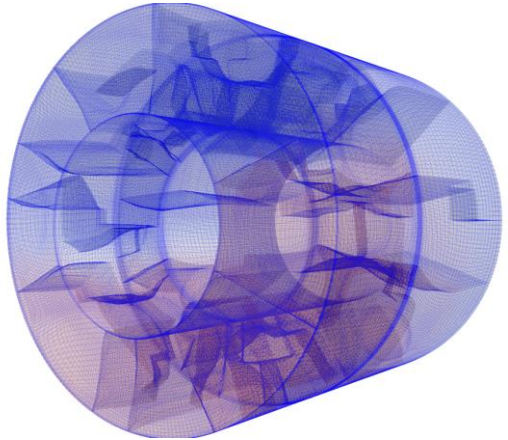
condition number

		# it			it down	κ	
lvl	n	$J_{\omega=0.7}$	GS	NN		before	after
2	9	49	17	8	2.13	5.9	1.6
3	49	273	95	21	4.52	29.8	2.9
4	225	1 323	463	66	7.02	127.3	7.8
5	961	5 879	2 057	39	52.74	516.0	23.4



Numerical experiments





aspect ratios

1:3

dim	Jac (0,7)	GS	NN
25	422	147	26
121	1955	683	39
529	8622	3017	64

1:10

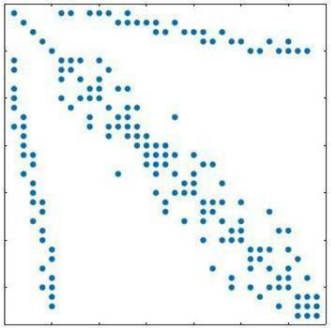
dim	Jac (0,7)	GS	NN
25	1001	385	22
121	5036	1762	32
529	-	7939	37

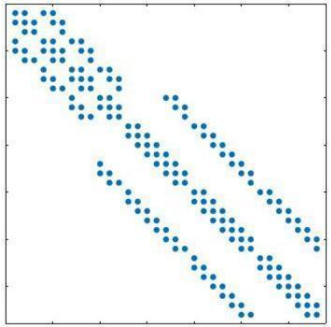
Numerical experiments

$$O(n^2 \cdot M) \quad \longrightarrow \quad O(\bar{n} \cdot M)$$

n	49	225	961	3 969	16 129
full	2 401	50 625	923.521	15 752 961	260 144 641
$\bar{n}$	289	1 457	6 481	27 281	111 889
diag	180	868	3 780	15 748	64 260
%	7.497	1.715	0.409	0.100	0.025

system matrix: dense to sparse





Numerical experiments

$$O(n^2 \cdot M) \quad \longrightarrow \quad O(\bar{n} \cdot M)$$

n	49	225	961	3 969	16 129
full	2 401	50 625	923.521	15 752 961	260 144 641
$\bar{n}$	289	1 457	6 481	27 281	111 889
diag	180	868	3 780	15 748	64 260
%	7.497	1.715	0.409	0.100	0.025

system matrix: dense to sparse

	NN_{sparse} (1 000)			NN_{full}(1 500)			NN_{full} (2 000)		
ω	0.6	0.7	0.8	0.6	0.7	0.8	0.6	0.7	0.8
1	33	27	23	118	110	88	29	24	37
2	37	31	26	110	94	82	31	25	20
3	35	29	25	109	93	81	29	24	22

different damping and different size of training data for 3 matrices

Numerical experiments

$$O(n^2 \cdot M) \quad \rightarrow \quad O(\bar{n} \cdot M)$$

n	49	225	961	3 969	16 129
full	2 401	50 625	923.521	15 752 961	260 144 641
$\bar{n}$	289	1 457	6 481	27 281	111 889
diag	180	868	3 780	15 748	64 260
%	7.497	1.715	0.409	0.100	0.025

system matrix: dense to sparse

	NN_{sparse} (1 000)			NN_{full}(1 500)			NN_{full} (2 000)		
ω	0.6	0.7	0.8	0.6	0.7	0.8	0.6	0.7	0.8
1	33	27	23	118	110	88	29	24	37
2	37	31	26	110	94	82	31	25	20
3	35	29	25	109	93	81	29	24	22

different damping and different size of training data for 3 matrices

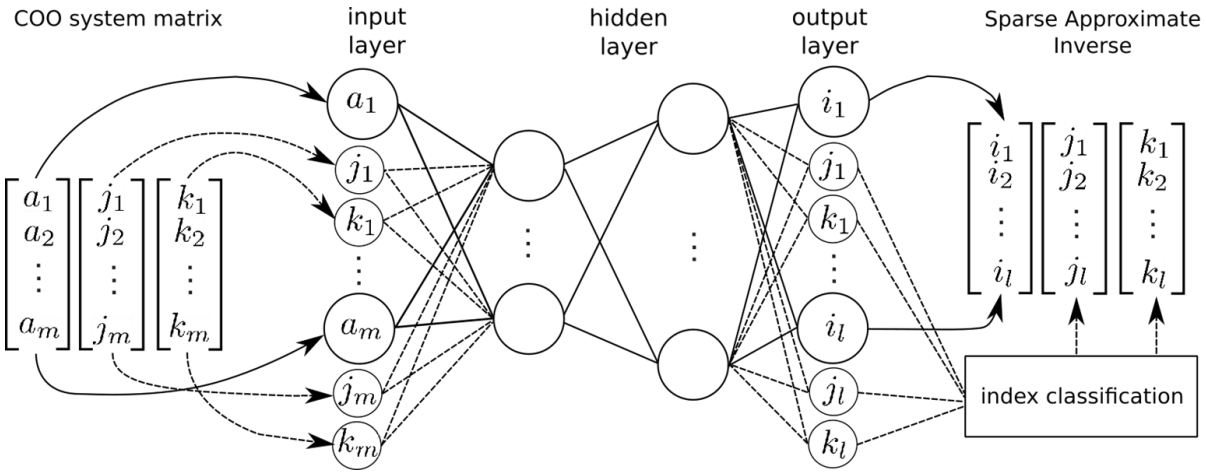
level	non zeros	time (s)
7	148.225	0.0027
8	591.361	0.0059
9	2.362.369	0.0153
10	9.443.329	0.0498

→ Not fully connected

→ Pruning

Extension and alternatives

- sparse matrix formats (more flexibility?)
- one NN vs two NN
- adjust loss function



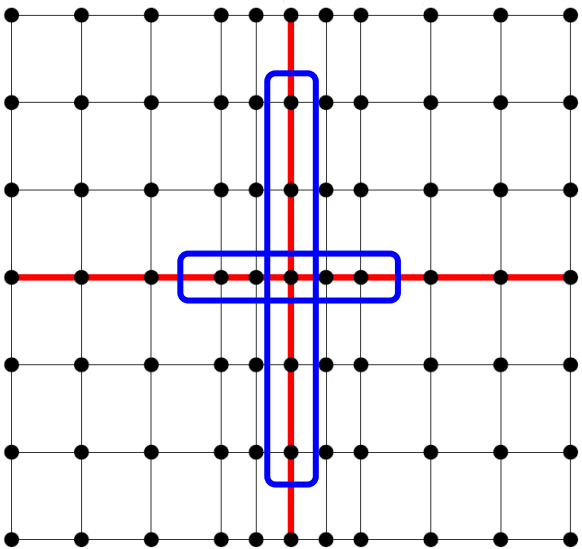
Extension and alternatives

In cooperation with
Enno Tiemann

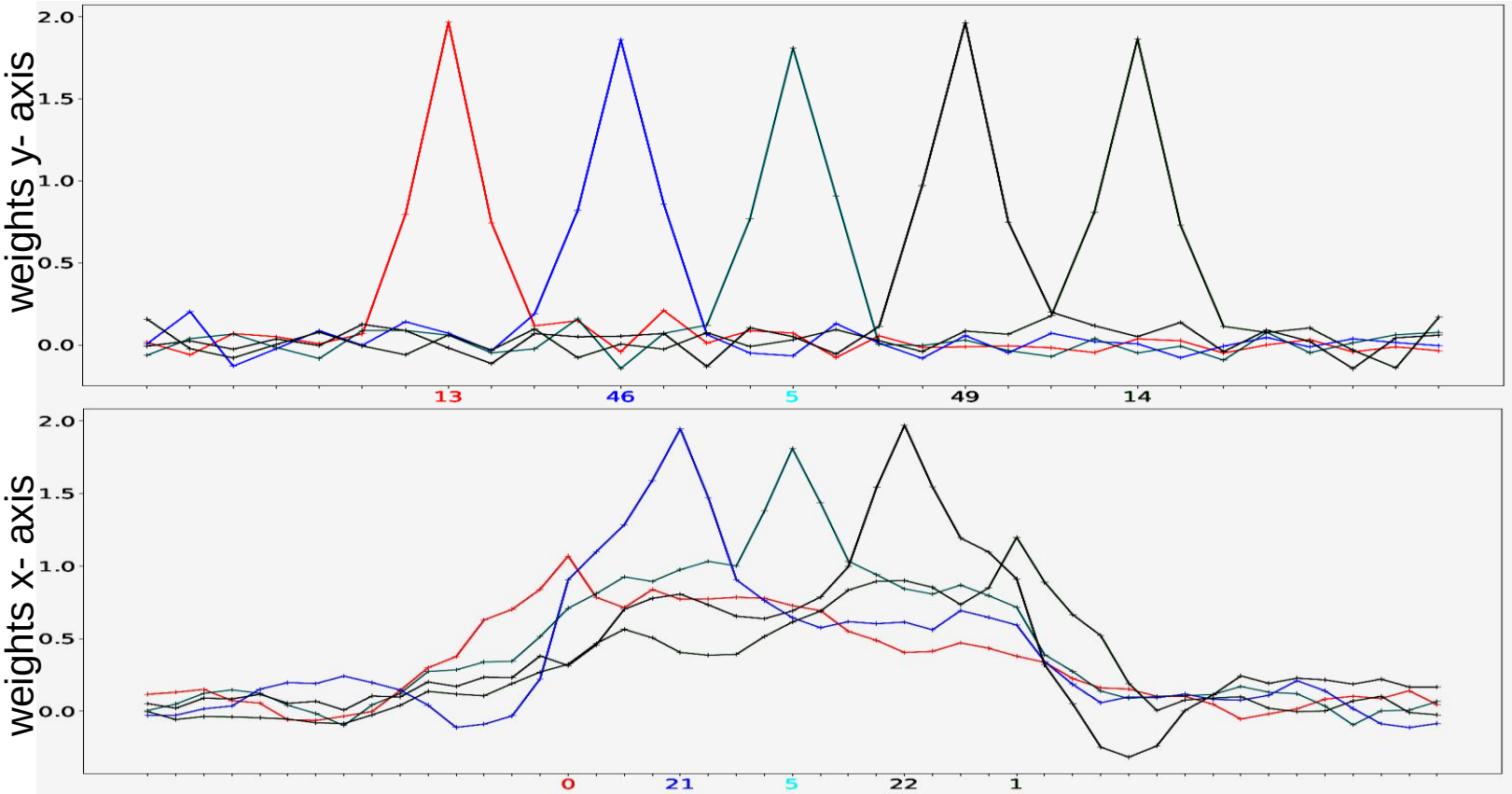
- now: try prolongation and restriction
- different NN and loss function

$$P = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$R = P^T$$



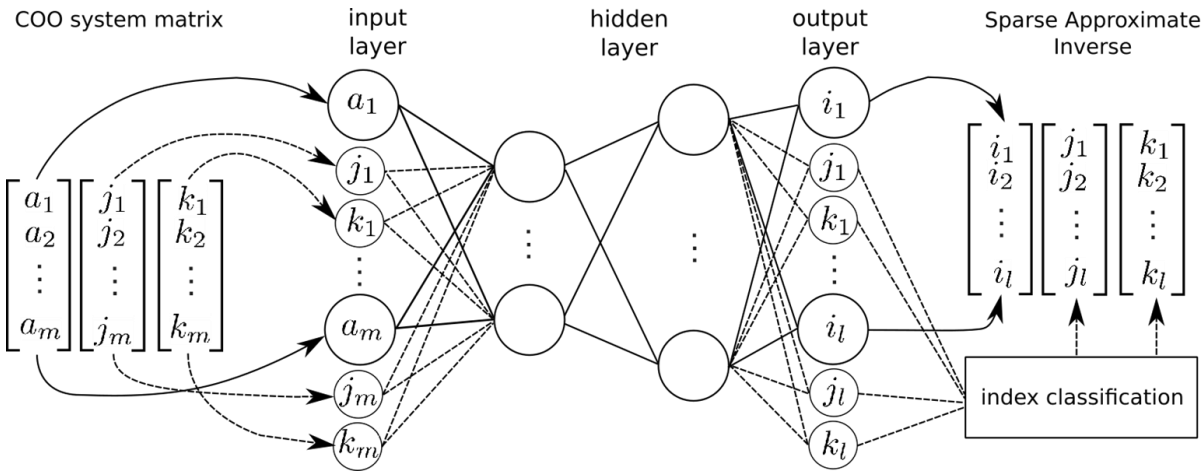
Extension and alternatives



Conclusion and future work

achievements:

- simple learning systems
  - → MLPs
  - → CNNs
  - → U-nets
  - → custom NN

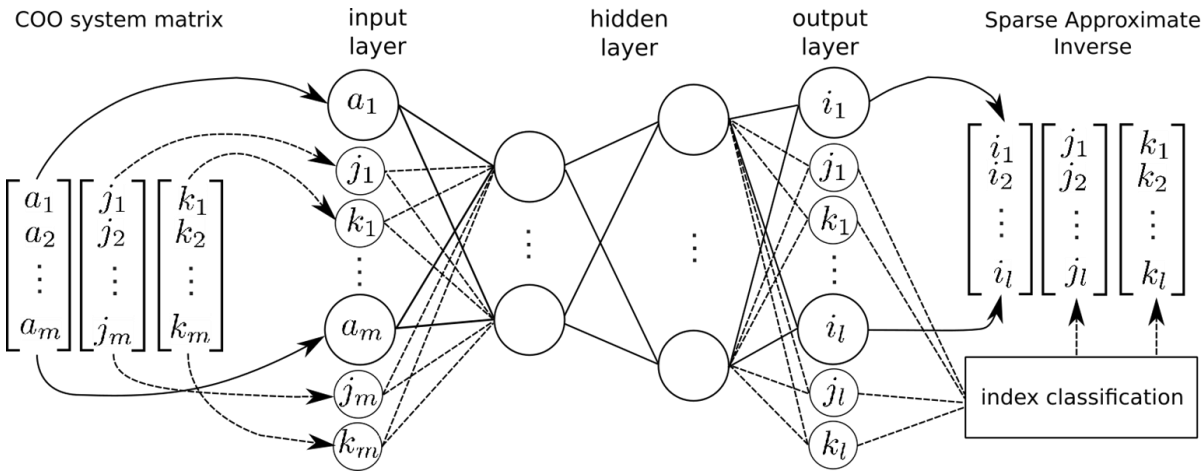


Conclusion and future work

achievements:

- simple learning systems
- support solver

→ preconditioner
→ smoother
→ operators
→ parameters

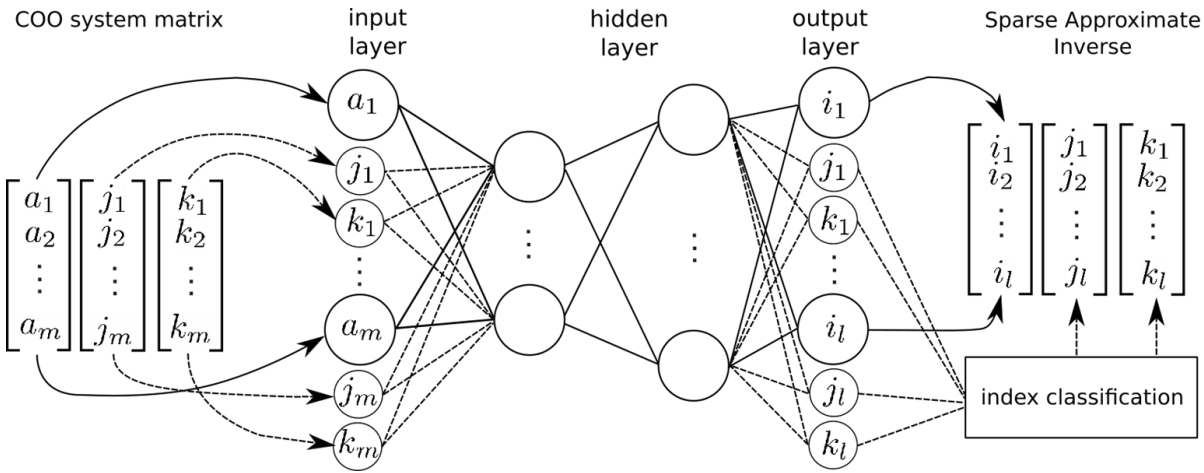


Conclusion and future work

achievements:

- simple learning systems
- support solver
- strong components

→ condition number
→ iteration number
→ time
→ anisotropies



Conclusion and future work

achievements:

- simple learning systems → condition number
- support solver → iteration number
- strong components → time, anisotropies

→ make machine learning methods usable for solving linear systems of equations and thus make modern hardware accessible

Conclusion and future work

achievements:

- simple learning systems → condition number
- support solver → iteration number
- strong components → time, anisotropies

→ make machine learning methods usable for solving linear systems of equations and thus make modern hardware accessible

→ but we need more flexibility
not (yet) a blackbox solver


Thanks for your attention!