Multilevel and Domain Decomposition Strategies for Training Neural Networks

R. Krause[1,2]

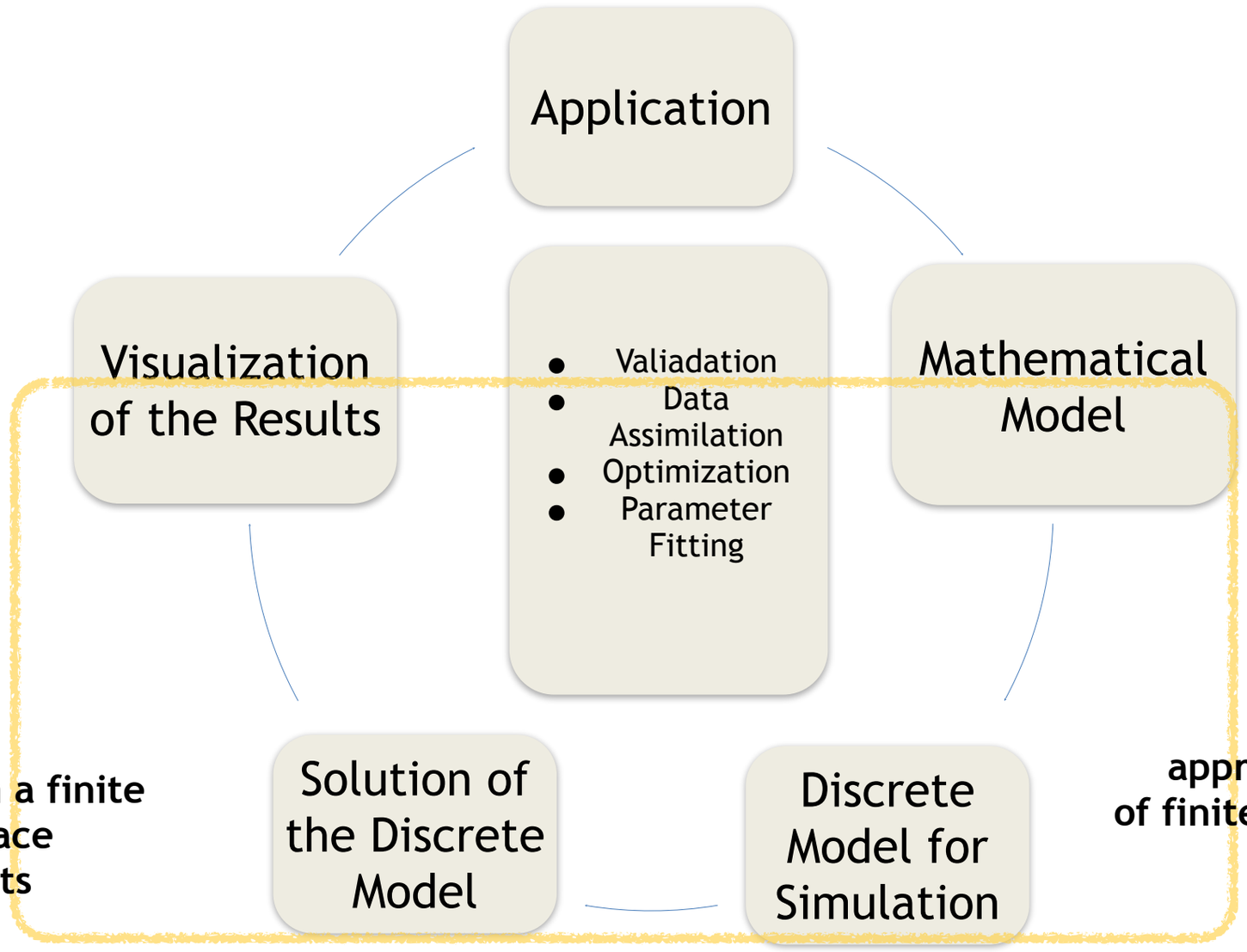
S. Cruz[1,2], A. Kopaničáková[2,3], H. Kothari[1], K. Trotti[1], G. Karniadakis[3]

1 Euler Institute
Università della Svizzera italiana, Lugano

2 Faculty of Mathematics and Informatics
UniDistance Suisse, Brig

3 Brown University
Providence, Rhode Island

Model Based Simulation



physics, biology, chemistry

first principles

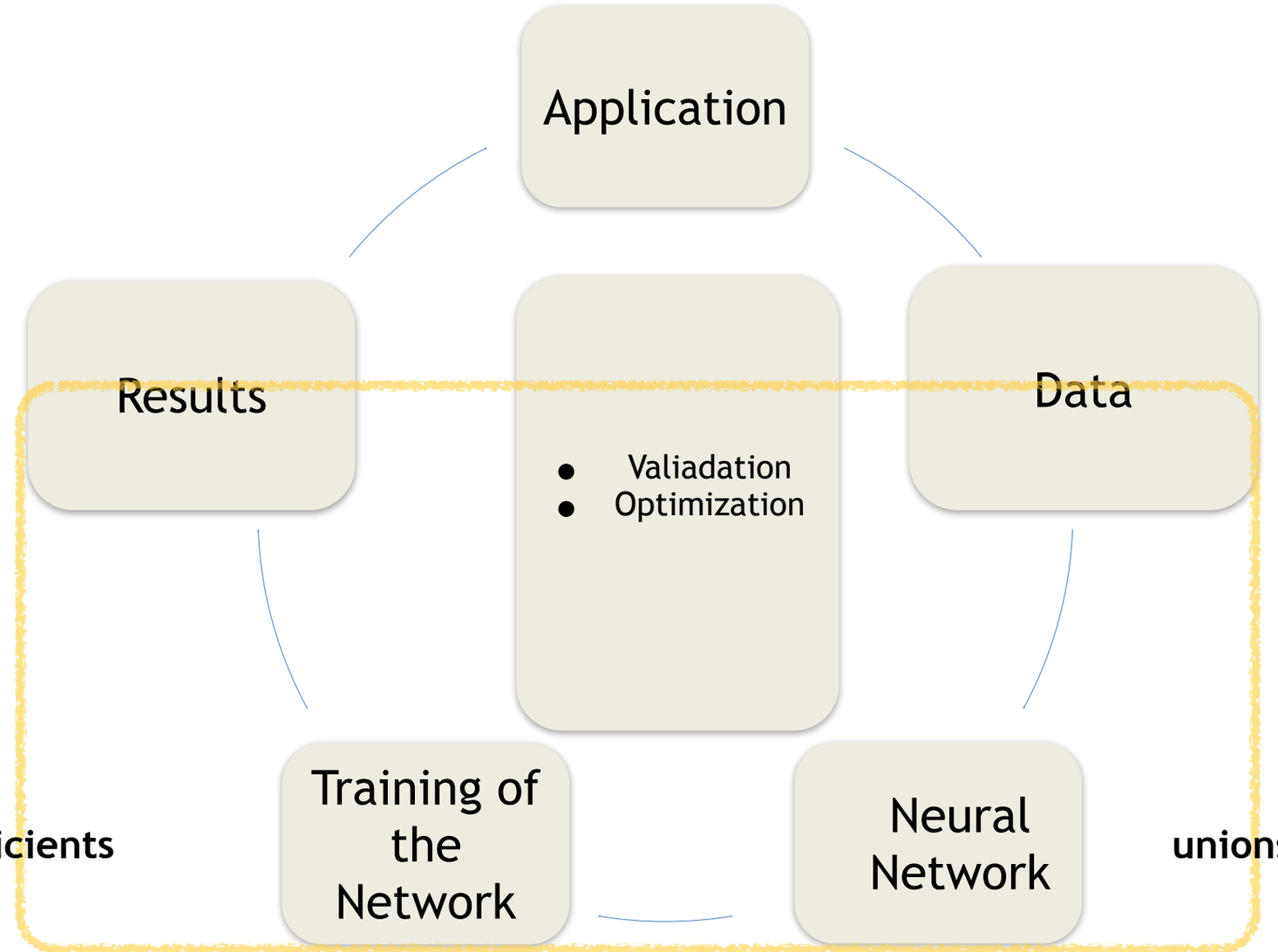
(non-)linear equation in a finite dimensional subspace choose coefficients

approximation: sequence of finite dimensional subspaces prescribe basis

Approximation and Solution separated

Machine Learning

network shall learn physics, biology, chemistry, ...



no physics, no biology, no chemistry

no first principles

choose basis and coefficients

approximation: unions of finite dimensional subspaces

Approximation and Solution done simultaneously

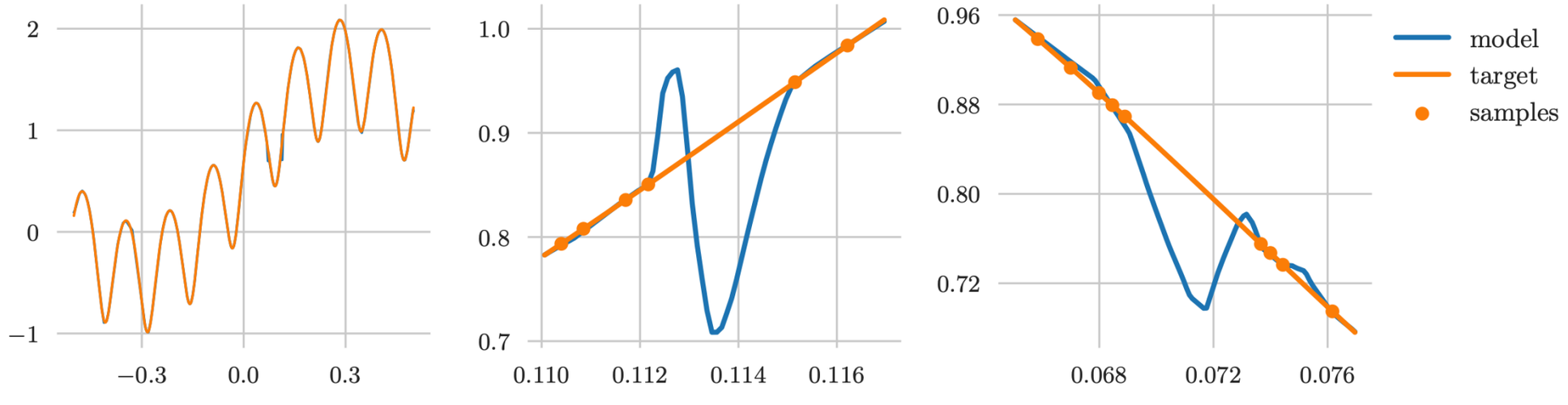
First principles nevertheless needed

LEARNING RELU NETWORKS TO HIGH UNIFORM ACCURACY IS INTRACTABLE

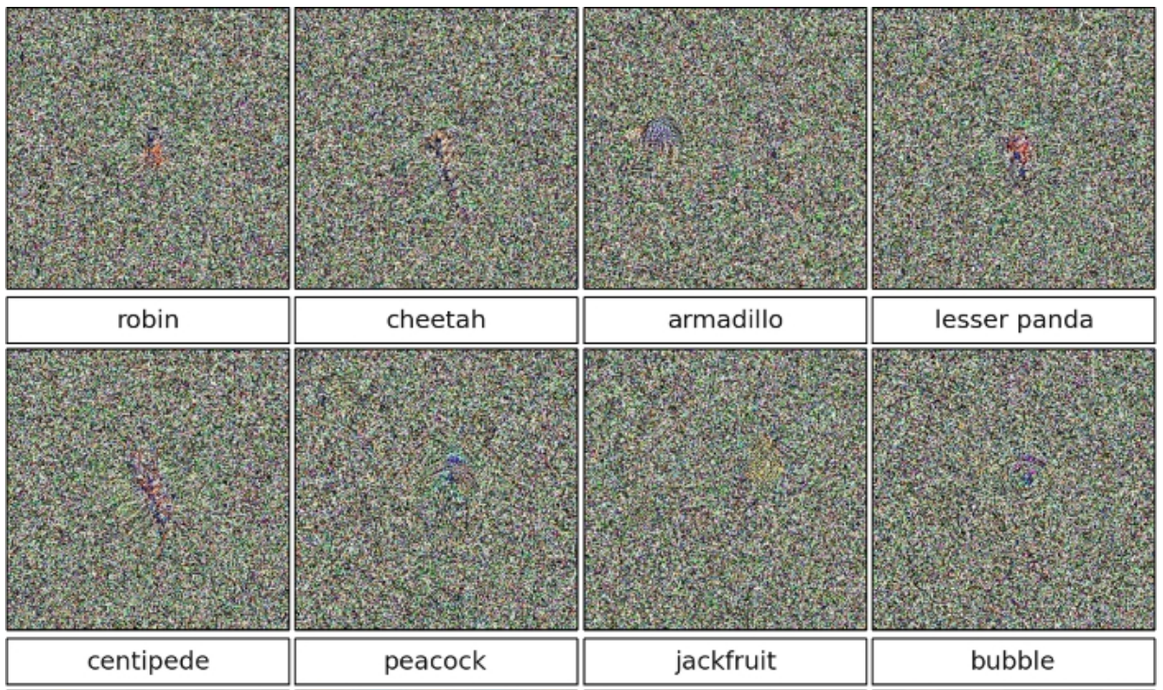
Julius Berner, Philipp Grohs, and Felix Voigtlaender, ICLR 2023

In this paper we precisely quantify the number of training samples needed for any conceivable training algorithm to guarantee a given uniform accuracy on any learning problem formulated over target classes containing (or consisting of) ReLU neural networks of a prescribed architecture.

We prove that, under very general assumptions, the minimal number of training samples for this task scales exponentially both in the depth and the input dimension of the network architecture.

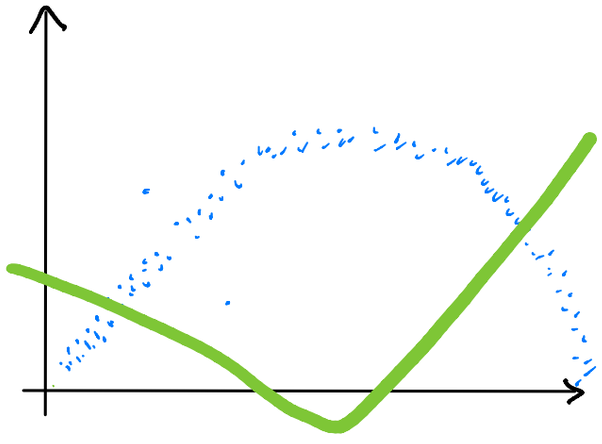


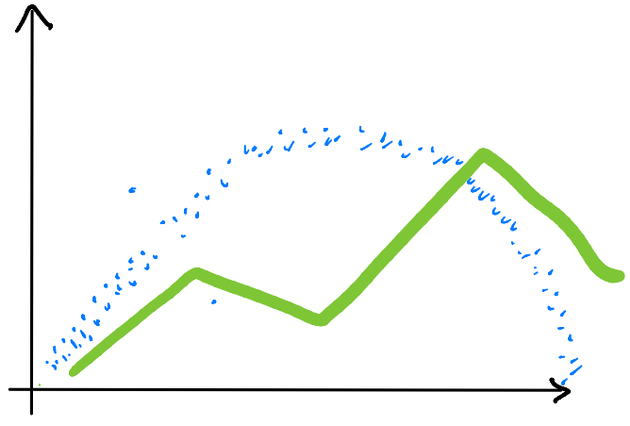
How to Fool a Neural Network

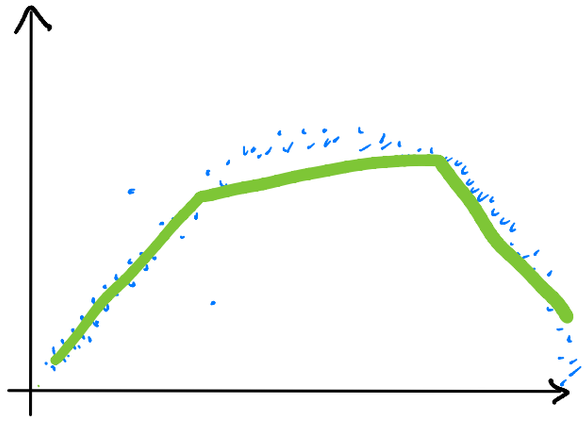


Nguyen A, Yosinski J, Clune J. *Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images*. In Computer Vision and Pattern Recognition (CVPR '15), IEEE, 2015.

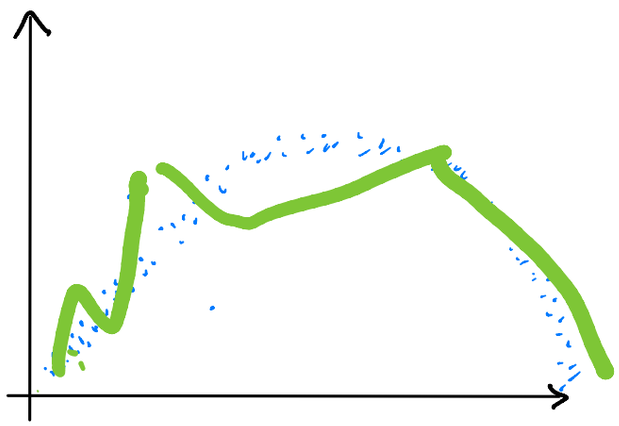
Piecewise Linear Approximation (ReLU) in High Dimensions



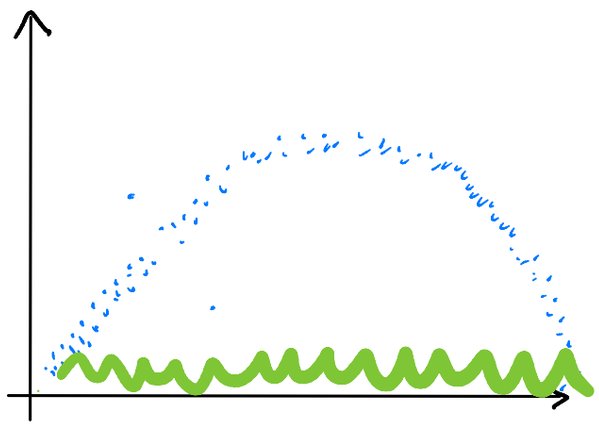




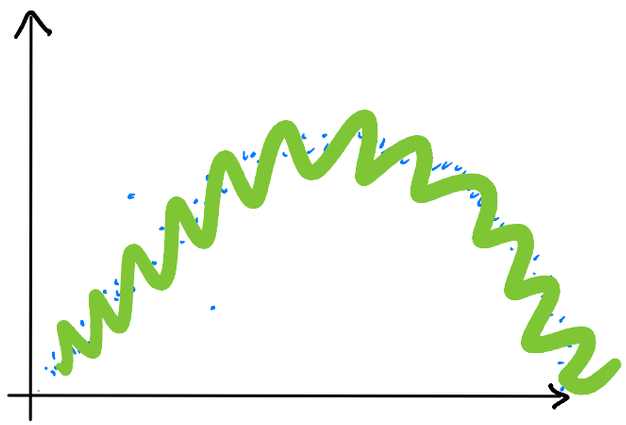
Training of a ReLU network



Bad generalization



Only noise captured



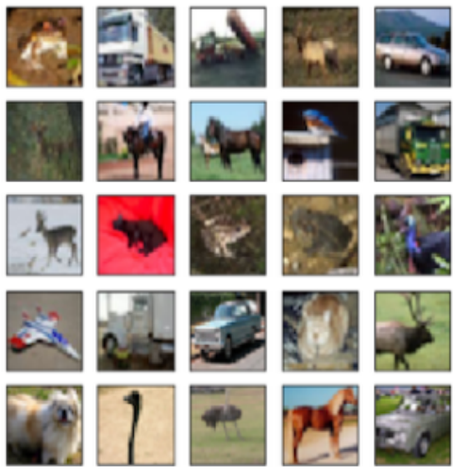
Overfitting

Architecture and number of parameters heavily influence the approximation quality

Training takes time

Fashion

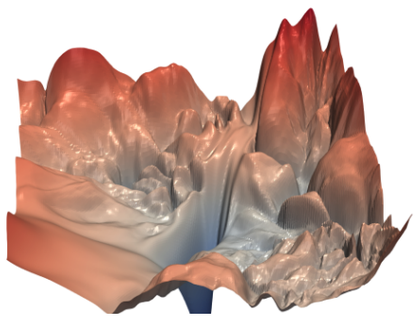
SVNH



CIFAR 10

Loss-functional Non Convex

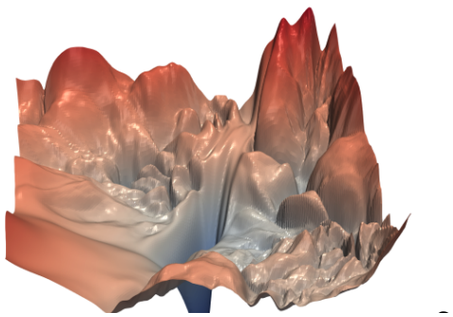
$$\theta := \text{argmin}_{\theta}\ \mathcal{L}(\theta)$$



- Rough energy landscape
- Size and complexitiy of the dataset and size of the network heavily influence the training speed
- Deep networks are highly expressive, i.e. can approximate any (sufficiently smooth) function with arbitrary accuracy
- standard training methods are "slow" methods of classical optimization

Training is difficult

$$\theta := \operatorname{argmin}_\theta \ \mathcal{L}(\theta)$$

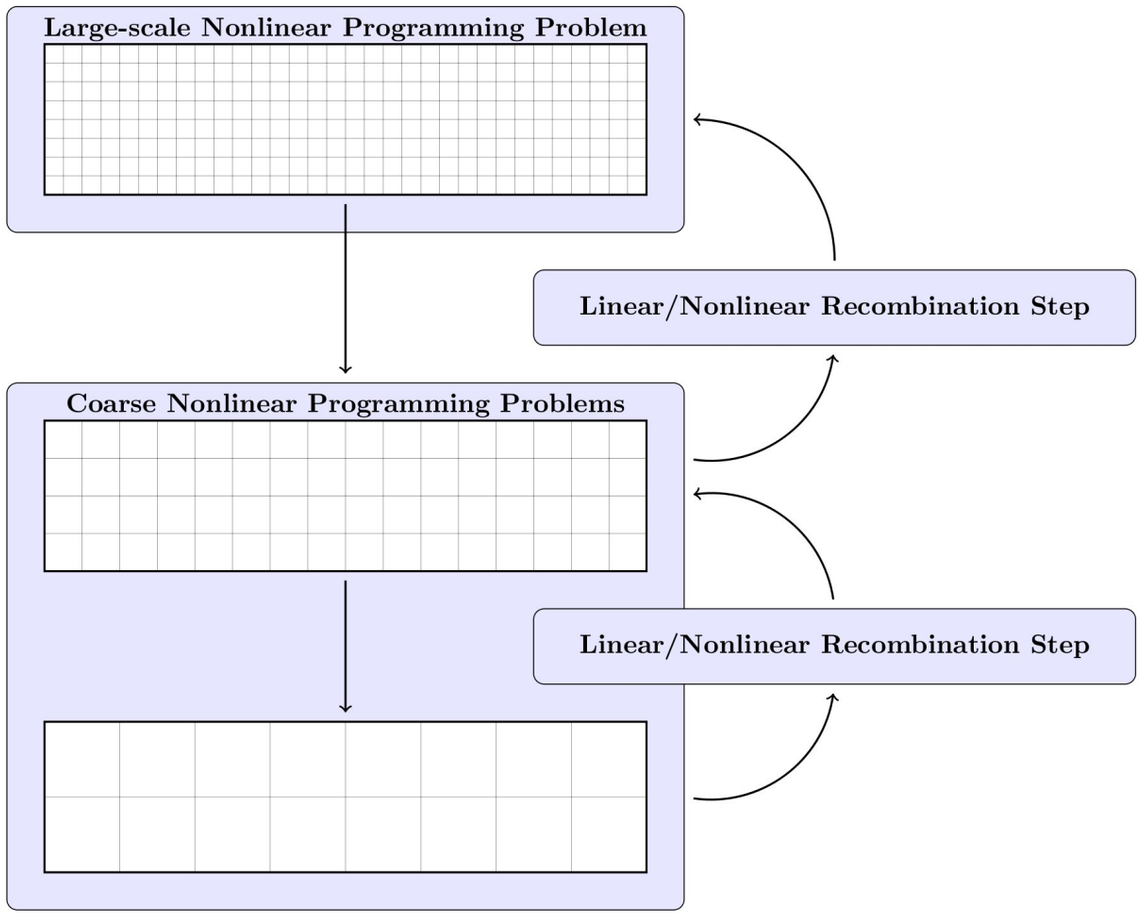


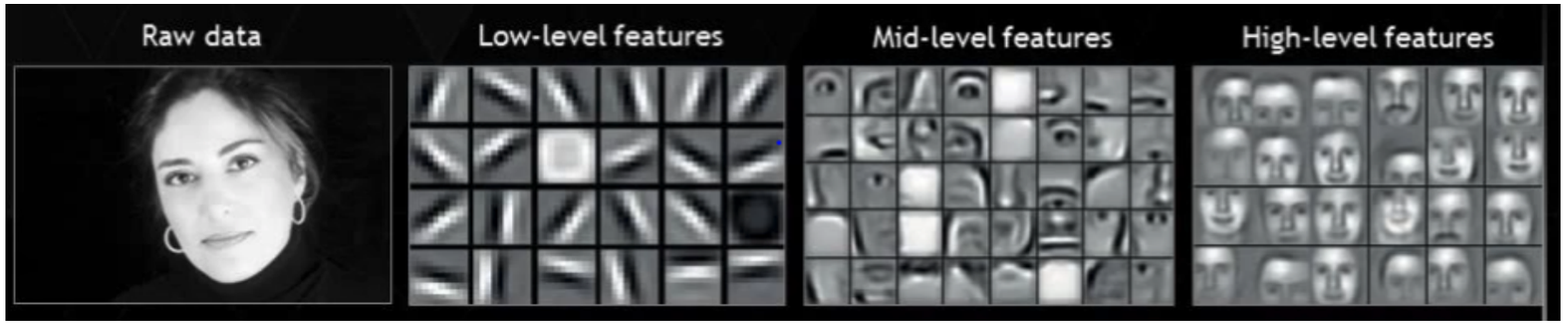
- Approximation error estimates for (deep) networks are available ("expressivity")
- But: merely existence results. We do not know, whether the network we have trained actually is some best approximation
- Size of the network heavily influences the approximation quality
- Standard training tend to ignore high-frequency information in the data ("spectral bias")
- Initialization of the weights is important
- Hyperparameter search costly
- Standard training methods are not parallel
- The mapping from weight space to the approximating function is not necessarily continuous
- Inversely not stable: close networks -as approximating elements in function space- in general do not have close weights [Petersen, Raslan, Voigtländer; 2020]

Improve and Control the Training Process

Multilevel Minimization

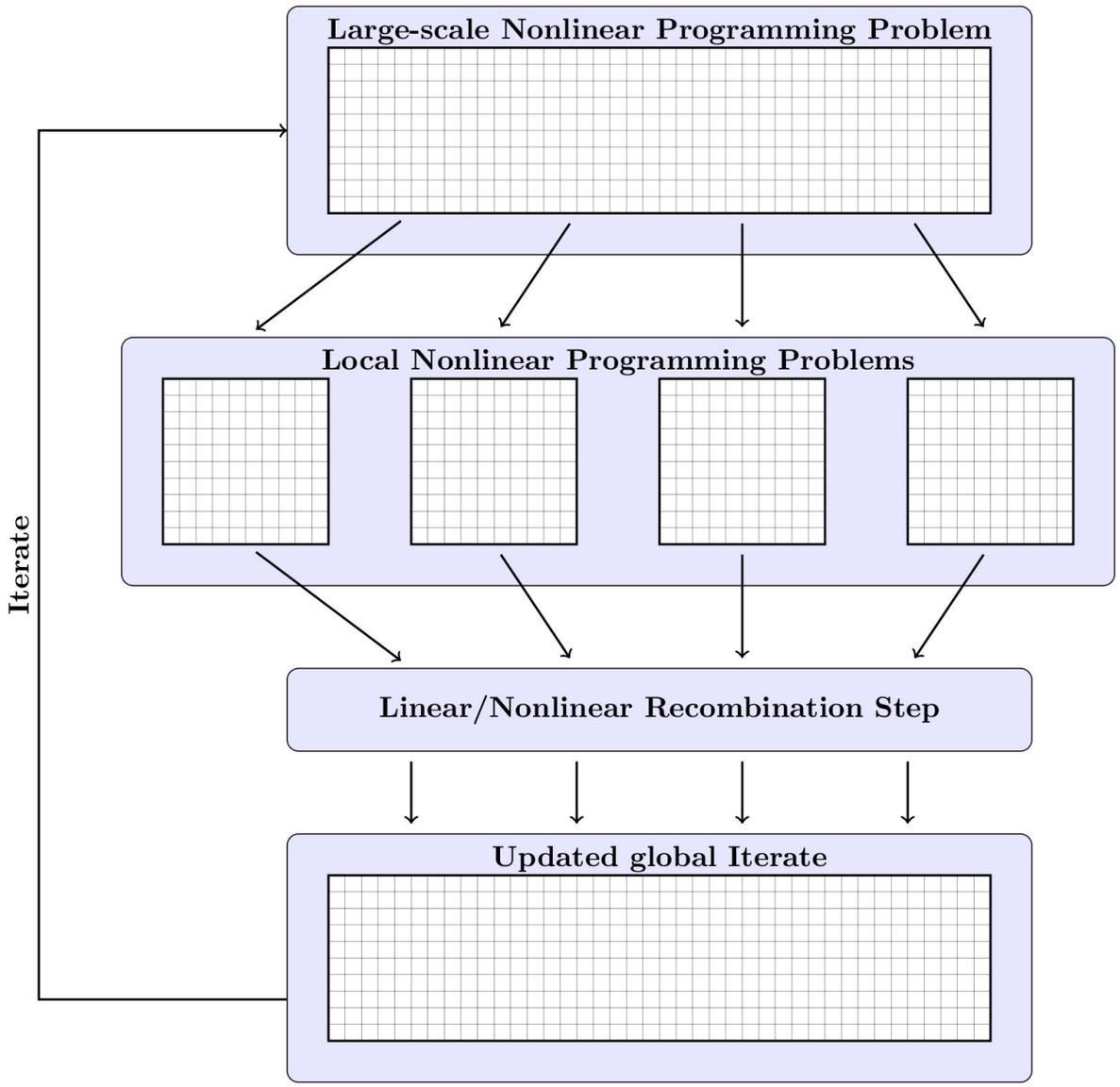
- Hierarchical basis
- Multilevel basis
- Deep neural networks
- Sparse grids
- Wavelets





R. Pieters, Python for Image Understanding, 2015

Parallel Minimization

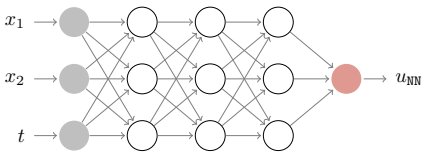


Partial-differential equations and physics-informed neural networks[1]

Given $\Omega \times (0, T]$, find $u : \Omega \times (0, T] \to \mathbb{R}$, such that

$$\begin{aligned} \mathcal{P}(u) &= f(\boldsymbol{x}), && \text{in } \Omega \times (0, T], \\ u &= u_{\mathtt{IC}}, && \text{at } \Omega \times \{0\}, \\ u &= u_{\mathtt{BC}}, && \text{on } \partial\Omega \times (0, T], \end{aligned}$$

where $\mathcal{P}$ denotes a nonlinear operator



Goal: Approximate solution $u(x, t)$ using neural network, i.e., $u(x, t) \approx u_{\mathtt{NN}} = DNN(\boldsymbol{\theta}; x, t)$

[1] Raissi et al., Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational physics, 2019

PINNs

Loss functional

$$\mathcal{L}(\boldsymbol{\theta}) := \underbrace{\frac{1}{|\mathcal{D}_{\mathtt{int}}|} \sum_{(\boldsymbol{x}_j,t_j)\in\mathcal{D}_{\mathtt{int}}} |\mathcal{R}(u_\theta(\boldsymbol{x}_j,t_j))|^2}_{\text{Interior loss}} + \underbrace{\frac{1}{|\mathcal{D}_{\mathtt{BC}}|} \sum_{(\boldsymbol{x}_j,t_j)\in\mathcal{D}_{\mathtt{BC}}} |u_\theta(\boldsymbol{x}_j,t_j) - u_{\mathtt{BC}}|^2}_{\text{Boundary loss}} + \underbrace{\frac{1}{|\mathcal{D}_{\mathtt{IC}}|} \sum_{\boldsymbol{x}_j\in\mathcal{D}_{\mathtt{IC}}} |u_\theta(\boldsymbol{x}_j,0) - u_{\mathtt{IC}}|^2}_{\text{Initial-condition loss}}$$

Error

$$\mathcal{E} := \|u_{\mathtt{NN}} - u\| \leq \underbrace{\|u_{\mathtt{NN}} - u_{\mathtt{opt}}\|}_{\text{Optimization error}} + \underbrace{\|u_{\mathtt{opt}} - u_{\mathtt{h}}\|}_{\text{Network's approximation error}} + \underbrace{\|u_{\mathtt{h}} - u\|}_{\text{Discretization error}}$$

- Discretization error - determined by the number/locations of collocation points
- Network's approximation error - determined by the network architecture
- Optimization error - determined by the choice of optimizer

Nonlinear preconditioning framework[3],[4]

- Consider the framework of nonlinear system of equations

$$F(\boldsymbol{\theta}) := \nabla \mathcal{L}(\boldsymbol{\theta}) = 0$$

- Instead of solving $F(\boldsymbol{\theta}) = 0$, our goal is to construct and solve a nonlinearly preconditioned system of equations

$$\mathcal{H}(\boldsymbol{\theta}) = F(G(\boldsymbol{\theta})) = 0$$

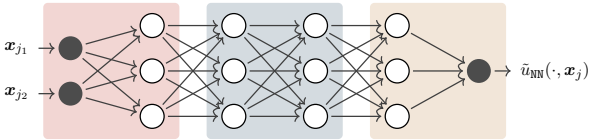
where $G(\boldsymbol{\theta})$ is an outcome of local solution process, i.e.,

  - $G(\boldsymbol{\theta})$ provides an improved initial iterate for the next optimization step
  - $F(G(\boldsymbol{\theta}))$ can be seen as composite multiplicative preconditioner

[3] Brune et al., Composing scalable nonlinear algebraic solvers, SIAM Review, 2015

[4] Dolean et al., Nonlinear Preconditioning: How to Use a Nonlinear Schwarz Method to Precondition Newton's Method, SIAM SISC. 2016

Decomposition of DNN



Example of the horizontal decomposition of network.

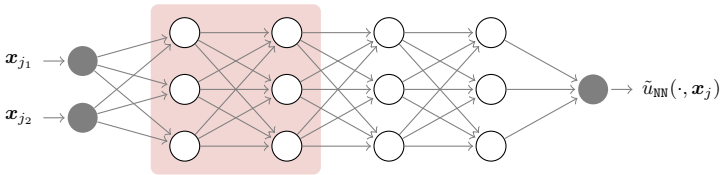
- Decompose the network into S subnetworks
- Transfer operators
  - Restriction operator $\boldsymbol{R}_s : \mathbb{R}^n \to \mathbb{R}^{n_s}$ extracts the parameters associated with subdomain s, i.e.,

$$\boldsymbol{\theta}_s = \boldsymbol{R}_s \boldsymbol{\theta}, \quad \text{for } s = 1, \ldots, S$$

  - Extension operator $\boldsymbol{E}_s : \mathbb{R}^{n_s} \to \mathbb{R}^n$ extends quantities related to subdomain s to the whole DNN, i.e.,

$$\boldsymbol{\theta} = \sum_{s=1}^{S} \boldsymbol{E}_s \boldsymbol{\theta}_s$$

Local solves



- Let $G_s : \mathbb{R}^n \to \mathbb{R}^{n_s}$ be a local solution operator for $1 \leq s \leq S$, such that

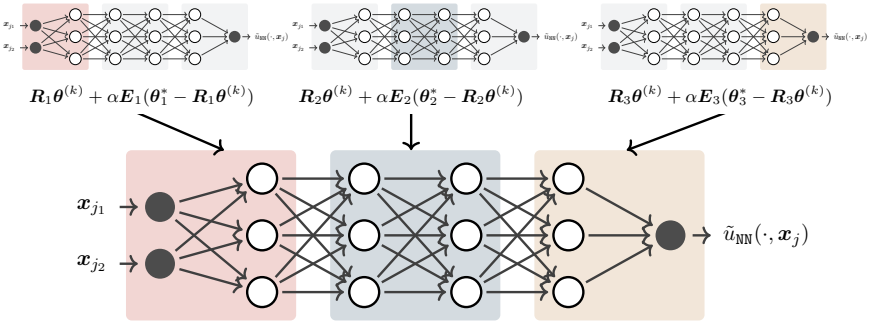
$$F\left(\sum_{s=1}^{S} \boldsymbol{E}_s G_s(\underbrace{\boldsymbol{R}_s \boldsymbol{\theta}}_{:=\boldsymbol{\theta}_s})\right) = 0,$$

- This corresponds to minimizing $\mathcal{L}$ wrt. $\boldsymbol{\theta}_s$, thus

$$\boldsymbol{\theta}_s^* = \mathrm{argmin}_{\boldsymbol{\theta}_s} \mathcal{L}(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_s, \ldots, \boldsymbol{\theta}_S),$$

where $\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_s, \ldots, \boldsymbol{\theta}_S]^\top$, while parameters of all other subdomains are kept fixed

Training of subnetworks



A sketch of local-to-global updates utilized by the ASPQN

- update global parameters: $G(\boldsymbol{\theta}^{(k)}) = \boldsymbol{\theta}^{(k)} + \alpha^{(k)} \sum_{s=1}^{N_{sd}} \boldsymbol{E}_s(\boldsymbol{\theta}_s^* - \boldsymbol{R}_s\boldsymbol{\theta}^{(k)})$
  where $\boldsymbol{\theta}_s^*$ represents a solution of the local minimization problem obtained in additive or multiplicative manner, associated with s-th subnetwork, while $\alpha^{(k)}$ denotes a step-size.

Right preconditioned L-BFGS

- Update from the subnetworks

$$\boldsymbol{\theta}^{(k+1/2)} = G(\boldsymbol{\theta}^{(k)}) = \boldsymbol{\theta}^{(k)} + \alpha^{(k)} \sum_{s=1}^{N_{sd}} \boldsymbol{E}_s(\boldsymbol{\theta}_s^* - \boldsymbol{R}_s\boldsymbol{\theta}^{(k)})$$

- Given memory of m secant pairs $\{\boldsymbol{s}^{(i)}, \boldsymbol{y}^{(i)}\}_{i=k-m}^{k-1}$, L-BFGS Hessian approximation is given as

$$\boldsymbol{B}^{(k+1)} = \boldsymbol{B}^{(0)} - \begin{bmatrix}\boldsymbol{B}^{(0)}\boldsymbol{S}^{(k)} & \boldsymbol{Y}^{(k)}\end{bmatrix} \begin{bmatrix}(\boldsymbol{S}^{(k)})^\top\boldsymbol{B}^{(0)}\boldsymbol{S}^{(k)} & \boldsymbol{L}^{(k)} \\ (\boldsymbol{L}^{(k)})^\top & -\boldsymbol{D}^{(k)}\end{bmatrix}^{-1} \begin{bmatrix}(\boldsymbol{S}^{(k)})^\top\boldsymbol{B}^{(0)} \\ (\boldsymbol{Y}^{(k)})^\top\end{bmatrix},$$

where $(\boldsymbol{S}^{(k)})^\top\boldsymbol{Y}^{(k)} = \boldsymbol{L}^{(k)} + \boldsymbol{D}^{(k)} + \boldsymbol{U}^{(k)}$ and $\boldsymbol{B}^{(0)} = \gamma\boldsymbol{I}$, with $\gamma = \frac{\langle \boldsymbol{y}^{(k)}, \boldsymbol{y}^{(k)}\rangle}{\langle \boldsymbol{y}^{(k)}, \boldsymbol{s}^{(k)}\rangle}$

- Matrices $\boldsymbol{S}^{(k)}, \boldsymbol{Y}^{(k)} \in \mathbb{R}^{m\times n}$ contain corrections and gradient displacements obtained as

$$\boldsymbol{s}^{(k)} = \boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k+1/2)}$$
$$\boldsymbol{y}^{(k)} = \nabla\mathcal{L}(\boldsymbol{\theta}^{(k+1)}) - \nabla\mathcal{L}(\boldsymbol{\theta}^{(k+1/2)})$$

Pseudo-algorithm

1. For a given $\boldsymbol{\theta}^{(k)}$, perform local step:
   - training on subnetworks in an *additive* manner (parallel)

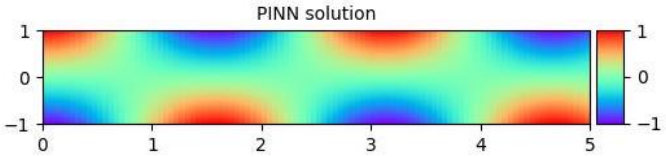
$$\text{Find } \boldsymbol{\theta}_s^* = \text{argmin}_{\boldsymbol{\theta}_s} \mathcal{L}(\boldsymbol{\theta}_1^{(k)}, \ldots, \boldsymbol{\theta}_s, \ldots, \boldsymbol{\theta}_S^{(k)})$$

   - training on subnetworks in a *multiplicative* manner (sequential)

$$\text{Find } \boldsymbol{\theta}_s^* = \text{argmin}_{\boldsymbol{\theta}_s} \mathcal{L}(\boldsymbol{\theta}_1^*, \ldots, \boldsymbol{\theta}_{s-1}^*, \boldsymbol{\theta}_s, \boldsymbol{\theta}_{s+1}^{(k)}, \ldots, \boldsymbol{\theta}_S^{(k)})$$

2. Synchronization step $\quad \boldsymbol{\theta}^{(k+1/2)} \leftarrow\!\shortmid \boldsymbol{\theta}^{(k)} + \sum_{s=1}^{S} \boldsymbol{E}_s(\boldsymbol{R}_s\boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}_s^*)$
3. Preconditioned quasi-Newton step $\quad \boldsymbol{p}^{(k+1/2)} \leftarrow\!\shortmid -(\boldsymbol{B}^{(k+1)})^{-1}\nabla\mathcal{L}(\boldsymbol{\theta}^{(k+1/2)})$
4. Momentum update $\quad \boldsymbol{v}^{(k+1/2)} \leftarrow\!\shortmid (1-\mu)\boldsymbol{v}^{(k-1/2)} + \mu\boldsymbol{p}^{(k+1/2)}$
5. Global update using momentum step $\quad \boldsymbol{\theta}^{(k+1)} \leftarrow\!\shortmid \boldsymbol{\theta}^{(k+1/2)} + \alpha^{(k+1/2)}\boldsymbol{v}^{(k+1/2)}$
6. Update $\boldsymbol{S}^{(k)}$ with $\quad \boldsymbol{s}^{(k)} \leftarrow\!\shortmid \boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k+1/2)}$
7. Update $\boldsymbol{Y}^{(k)}$ with $\quad \boldsymbol{y}^{(k)} \leftarrow\!\shortmid \nabla\mathcal{L}(\boldsymbol{\theta}^{(k+1)}) - \nabla\mathcal{L}(\boldsymbol{\theta}^{(k+1/2)})$

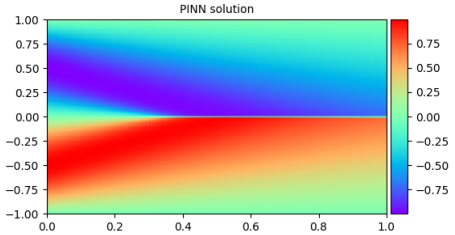
Klein-Gordon Equation (Nonlinear second-order hyperbolic PDE)



$$
\begin{aligned}
\frac{\partial^2 u}{\partial t^2} + \alpha \nabla^2 u + \beta u + \gamma u^2 &= f(t,x), && \forall\ (t,x) \in (0,12] \times (-1,1),\\
u &= x, && \forall\ (t,x) \in \{0\} \times [-1,1],\\
\frac{\partial u}{\partial t} &= 0, && \forall\ (t,x) \in \{0\} \times [-1,1],\\
u &= -\cos(t), && \forall\ (t,x) \in (0,12] \times \{-1\},\\
u &= \cos(t), && \forall\ (t,x) \in (0,12] \times \{1\},
\end{aligned}
$$

We employ $\alpha = -1, \beta = 0, \gamma = 1$ and $f(t,x) := -x\cos(t) + x^2\cos^2(t)$.

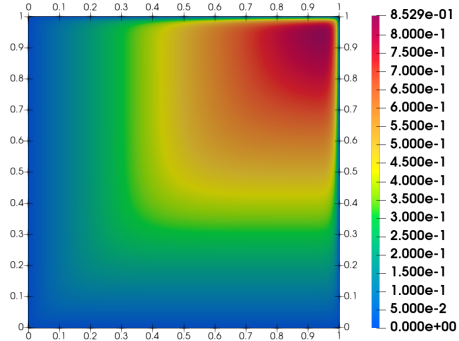
Burgers' equation



$$
\begin{aligned}
\frac{\partial u}{\partial t} + u\nabla u - \nu\nabla^2 u = 0, \qquad & \forall\ (t,x) \in (0,1] \times (-1,1), \\
u = -\sin(\pi x), \qquad & \forall\ (t,x) \in \{0\} \times [-1,1], \\
u = 0, \qquad & \forall\ (t,x) \in (0,1] \times \{1\}, \\
u = 0, \qquad & \forall\ (t,x) \in (0,1] \times \{-1\},
\end{aligned}
$$

where $\nu = 0.01/\pi$.

Diffusion-Transport equation



$$-\nabla \cdot \mu \nabla u + \boldsymbol{b} \cdot \nabla u = f, \quad \forall\ (x_1, x_2) \in (0,1) \times (0,1),$$
$$u = 0, \quad \text{on } \partial\Omega,$$

where $\boldsymbol{b} = (1,1)^\top$, $f = 1$ and $\mu = 10^{-2}$.

Network architecture and optimizer setup

Klein-Gordon:

- 6 hidden layers, 50 neurons each
- $10,000$ collocation points

Burgers':

- 8 hidden layers, 20 neurons each
- $10,000$ collocation points

Transport-Diffusion:

- 10 hidden layers, 50 neurons each
- $10,000$ collocation points

State-of-the-art optimizers:

- Adam with fixed learning rate
- L-BFGS with momentum, line-search and memory size, $m = 3$

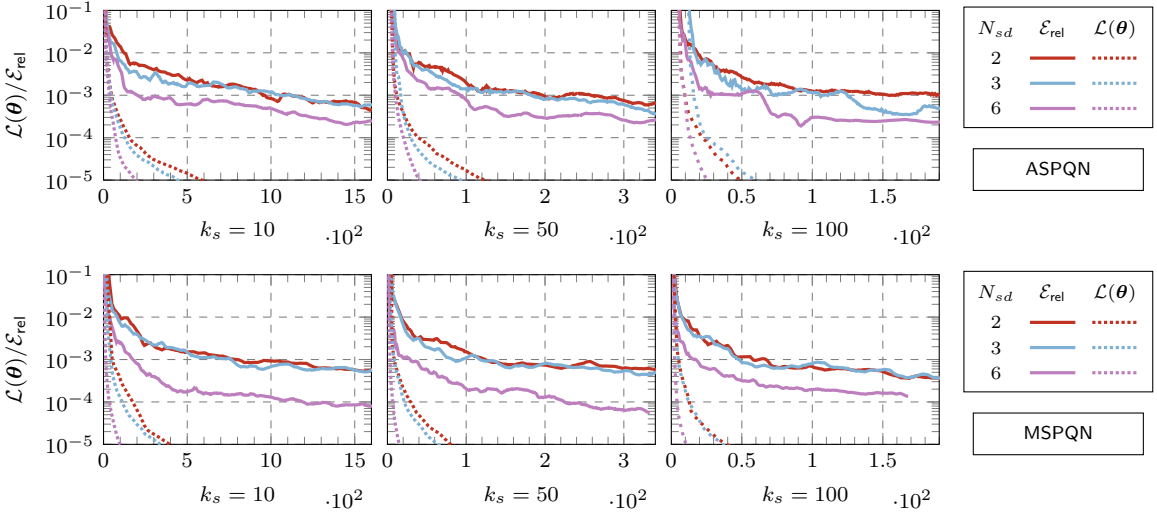
Right preconditioned L-BFGS setup:

- L-BFGS as a local optimizer with varying number of steps
- Global step performed by preconditioned L-BFGS with line-search
- Varying number of subdomains and memory size, $m = 3$

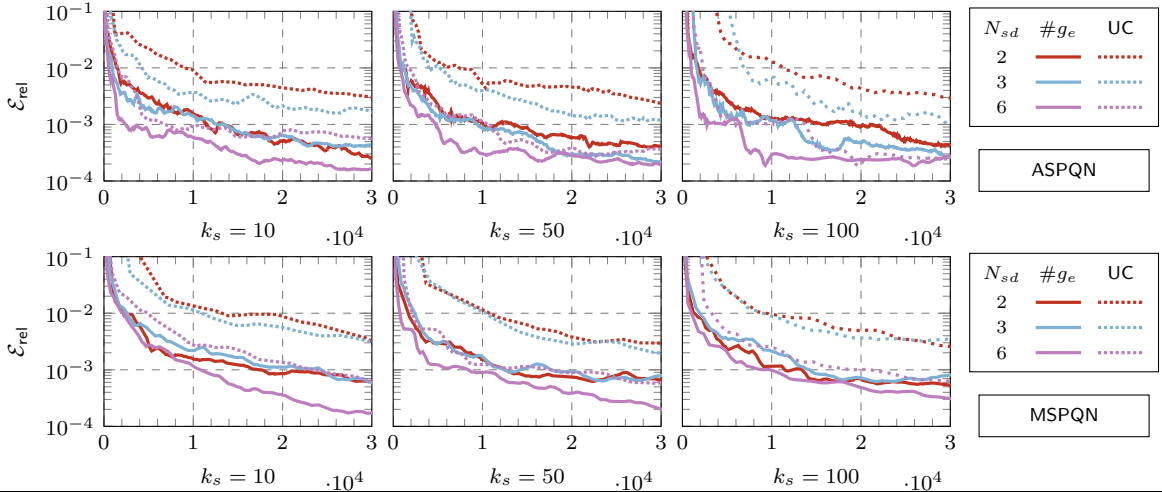
Implementation

- PyTorch library
- Nvidia's NCCL backend

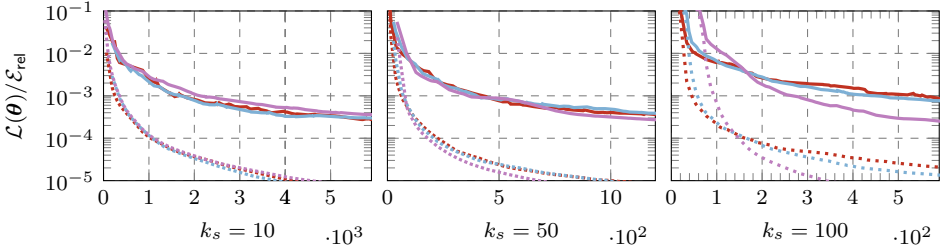
Klein-Gordon: ASPQN v/s MSPQN (# iterations)

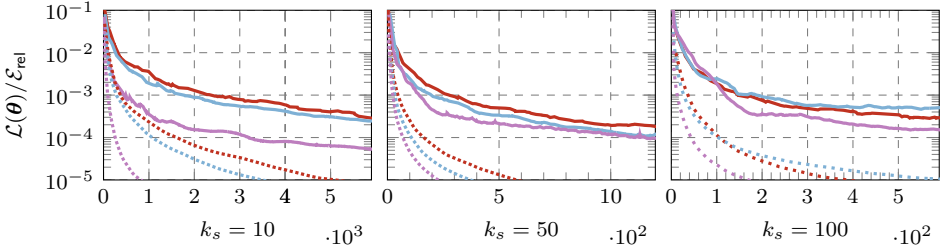


Klein-Gordon: ASPQN v/s MSPQN (Grad evals ($\# g_e$) and Update Cost)

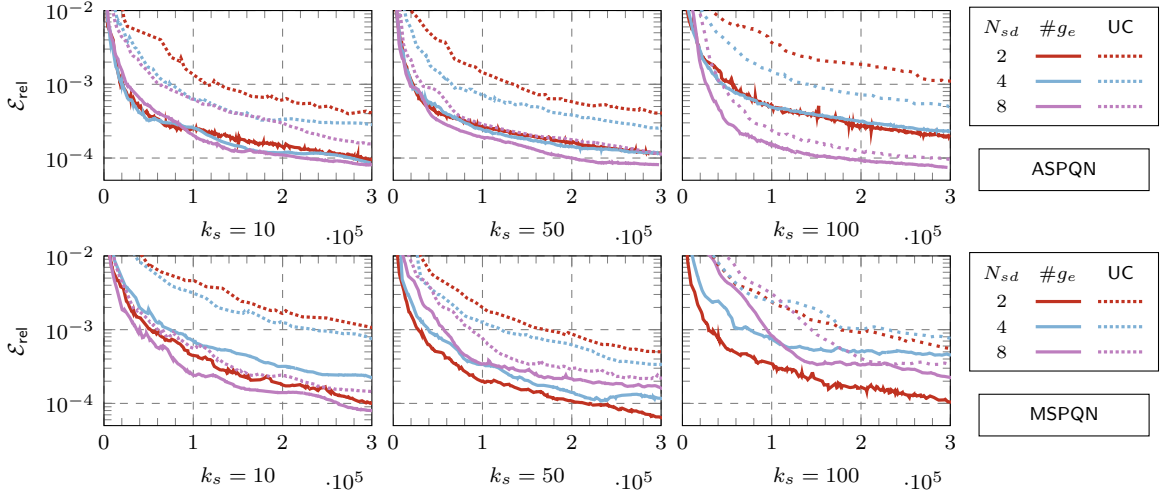


Burgers': ASPQN v/s MSPQN (# iterations)

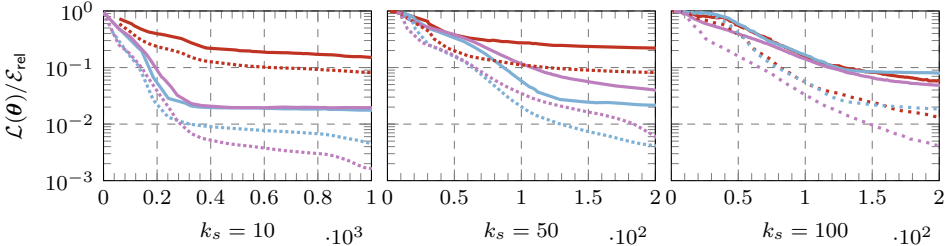


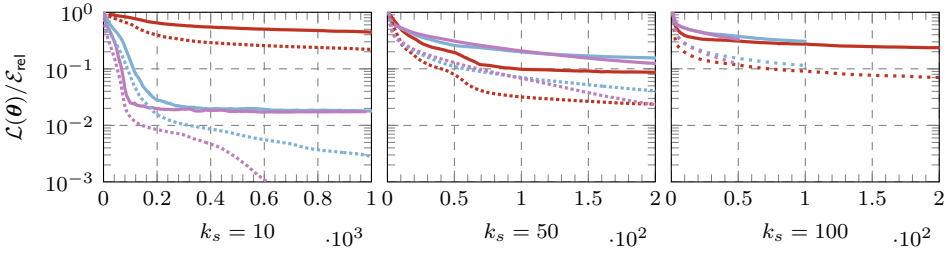


Burgers': ASPQN v/s MSPQN (Grad evals (# g_e) and Update Cost)

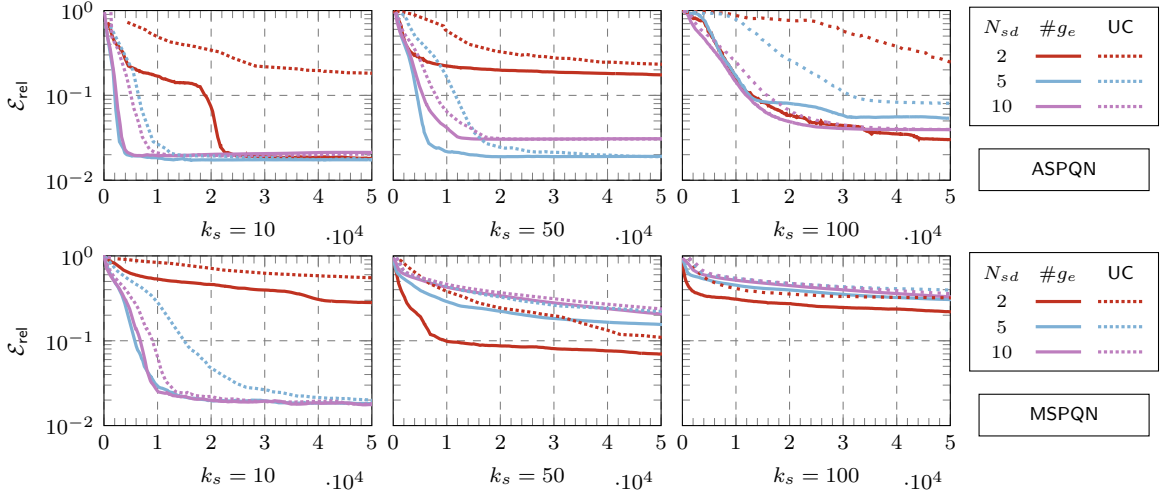


Transport: ASPQN v/s MSPQN (#iterations)

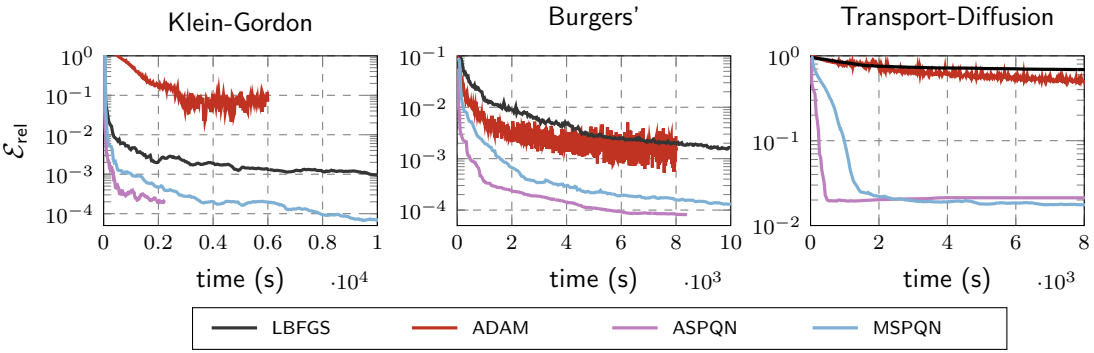




Transport: ASPQN v/s MSPQN (Grad evals (# g_e) and Update Cost)



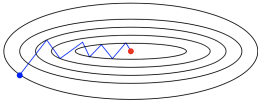
Comparing computational time



Example	$\mathcal{E}_{\text{rel}}$ (L-BFGS)	Time to solution (mins)				
		L-BFGS	Adam	ASPQN	(# nodes)	MSPQN
Burgers'	4.6×10^{-4}	558.5	–	14.4	(8)	40.7
Klein-Gordon	6.1×10^{-4}	236.5	–	6.8	(6)	26.9

Line-search framework

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \mathbf{s}_k$$

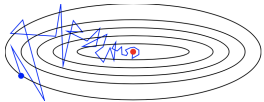


Batch methods:

- Construct $\mathbf{s}_k$ using all samples, e.g.

$$\mathbf{s}_k = -\frac{1}{p}\sum_{j=1}^{p} \nabla \ell(f_m(\boldsymbol{x}_j, \boldsymbol{\theta}), \boldsymbol{c}_j)$$

- Iteration cost is linear in p
- Convergence with constant α_k or α_k which is adaptively chosen via line-search



Stochastic methods:

- Construct $\mathbf{s}_k$ using a sample, e.g.

$$\mathbf{s}_k = -\nabla \ell_{j_k}(f_m(\boldsymbol{x}_j, \boldsymbol{\theta}), \boldsymbol{c}_j),$$

  where j_k from $\{1, \ldots, p\}$
- With prob$(j_k = j)$, the SG is an unbiased estimate of gradient, i.e., $\mathbb{E}[\nabla \ell_{j_k}(\boldsymbol{\theta})] = \nabla \mathcal{L}(\boldsymbol{\theta})$
- Iteration cost is independent of p
- Convergence requires $\alpha_k \to 0$

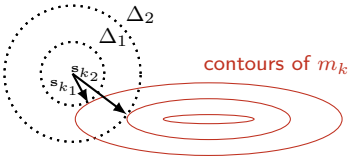
[2] Schmidt, CPSC 540: Machine learning. Lecture notes on stochastic gradient, 2018.

Batch trust-region (TR) framework [Conn et al., '00,...]

1. Generate the model $m_k(\mathbf{s}_k) := \mathcal{L}_k + \langle \boldsymbol{g}_k, \mathbf{s}_k \rangle + \frac{1}{2}\langle \mathbf{s}_k, \boldsymbol{B}_k \mathbf{s}_k \rangle$

2. Solve TR subproblem

$$\min_{\mathbf{s}_k \in \mathbb{R}^n} m_k(\mathbf{s}_k)$$
$$\text{subject to } \|\mathbf{s}_k\|_p \leq \Delta_k$$



3. Acceptance: $\rho = \frac{\mathcal{L}(\boldsymbol{\theta}_k + \mathbf{s}_k) - \mathcal{L}(\boldsymbol{\theta}_k)}{m_k(\mathbf{s}_k)} \geq \eta$ then $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{s}_k$,
otherwise $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k$, $\eta \in (0, 1)$

4. Update of the trust-region radius: Δ_k by means of ρ

No user-specified learning rate needed!

Stochastic trust-region (TR) framework

1. Generate the model
   $m_k(\mathbf{s}_k) := \mathcal{L}_k + \langle \boldsymbol{g}_k, \mathbf{s}_k \rangle + \frac{1}{2}\langle \mathbf{s}_k, \boldsymbol{B}_k \mathbf{s}_k \rangle$
2. Solve TR subproblem

$$\min_{\mathbf{s}_k \in \mathbb{R}^n} m_k(\mathbf{s}_k)$$
$$\text{subject to } \|\mathbf{s}_k\|_p \leq \Delta_k$$

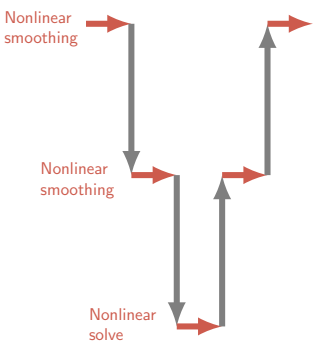
3. Acceptance: $\rho = \frac{\mathcal{L}(\boldsymbol{\theta}_k + \mathbf{s}_k) - \mathcal{L}(\boldsymbol{\theta}_k)}{m_k(\mathbf{s}_k)} \geq \eta$
   then $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \mathbf{s}_k$,

   otherwise $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k$, $\eta \in (0, 1)$
4. Update of the trust-region radius: Δ_k by means of ρ

- Evaluate $\mathcal{L}$ and g_k exactly, while sub-sample $\boldsymbol{B}_k$ [Xu et al., '20, '21 . . .]
- Evaluate $\mathcal{L}$ exactly, but use sub-sampled information for evaluation of g_k and $\boldsymbol{B}_k$ [Gratton et al., '18, Erway et al., '20,. . .]
- Use subsampled information to evaluate $\mathcal{L}, g_k$ and $\boldsymbol{B}_k$ [Bellavia et al., '18, Blanchet et al., '19, Chen et al., '15, Mohr et al., '19,. . .]
  - Decrease in mini-batch loss does not imply decrease in global loss
    $\implies$ to preserve global convergence $\mathcal{L}, g_k$ have to be estimated with increasing accuracy, e.g., by enlarging the sample sizes

Nonlinear multilevel minimization techniques

Nonlinear multilevel minimization methods



- Commonly used in numerics to solve PDEs as they are of optimal complexity and scalability
- An extension to linear multigrid [Briggs et al., '00; Hackbusch '13;...]

- Solving system of eqs. arising from discretization of PDEs:
  - FAS[Brandt '77], NLMG [Hackbusch '85; Reusken et al., '87, '88], MNM[Yavneh, Dardyk '06],...

- Nonlinear optimization perspective:
  - Line-search: MG/OPT [Nash '00], LSMM [Wen, Goldfarb '08], MMOP[Borzi, Schultz '09], CSML[Frandi, Papini '14], SDM[Ta, Xu '98 '02; Chen et al., '19], NeMO[He et al., '22],...
  - Trust-region: RMTR [Gratton et al., '06 '08 '10; Gross et al., '09], AMRTR[Ulbrich, Ziems '11 '17], PODRMTR[Kragel '05],...
  - Cubic/higher order regularization: MARq [Calandra et al., '19]

Nonlinear multilevel minimization framework

1. Construction of a hierarchy of auxiliary "low-cost" objective functions $\{\mathcal{L}^l\}_{l=1}^L$, where $\mathcal{L}^l$ is computationally cheaper to minimize than $\mathcal{L}^{l+1}$

2. Transfer operators $\{\mathbf{I}_l^{l+1}\}_{l=1}^{L-1}$, $\{\mathbf{R}_{l+1}^l\}_{l=1}^{L-1}$, $\{\mathbf{P}_{l+1}^l\}_{l=1}^{L-1}$
   - Actual form of the transfer operators depends on the structure and the dimension of $\{\mathcal{L}^l\}_{l=1}^L$
   - We consider three types:
     - Prolongation operator $\mathbf{I}_l^{l+1} : \mathbb{R}^{n^l} \to \mathbb{R}^{n^{l+1}}$ for transferring primal variables
     - Restriction operator $\mathbf{R}_{l+1}^l : \mathbb{R}^{n^{l+1}} \to \mathbb{R}^{n^l}$, where $\mathbf{R}_{l+1}^l := (\mathbf{I}_l^{l+1})^T$, for transferring dual variables
     - Projection operator $\mathbf{P}_{l+1}^l : \mathbb{R}^{n^{l+1}} \to \mathbb{R}^{n^l}$ for transferring primal variables

3. Convergence control
   - Constructing the coarse level models, which are at least first-order consistent with fine-level model
   - Employing globalization strategy to ensure a convergence

Construction of coarse-level models

1st-order additive approach[Brandt '77; Nash '00;...]

$$h^l(\boldsymbol{\theta}^l) := \underbrace{\mathcal{L}^l(\boldsymbol{\theta}^l)}_{\text{low-cost model}} + \underbrace{\langle \delta \mathbf{g}^l, \boldsymbol{s}^l \rangle}_{\text{correction term}}$$

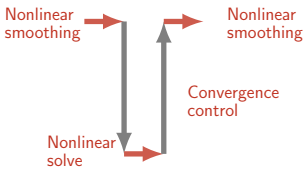
where

$$\delta \mathbf{g}^l := \begin{cases} \boldsymbol{R}_{l+1}^l \nabla h^{l+1}(\boldsymbol{\theta}_{\mu_1}^{l+1}) - \nabla \mathcal{L}^l(\boldsymbol{\theta}_0^l), & \text{if } l < L, \\ \mathbf{0}, & \text{otherwise,} \end{cases}$$

- First coarse-level correction $\mathbf{s}^l$ goes in the direction of the restricted fine-level gradient
- If $\mathbf{s}^l$ is descent direction on level $l \implies$ descent direction on level $l+1$

$$\langle \nabla h^l(\boldsymbol{\theta}_0^l), \mathbf{s}^l \rangle = \langle \boldsymbol{R}_{l+1}^l \nabla h^{l+1}(\boldsymbol{\theta}_{\mu_1}^{l+1}), \mathbf{s}^l \rangle = \langle \nabla h^{l+1}(\boldsymbol{\theta}_{\mu_1}^{l+1}), \mathbf{I}_l^{l+1} \mathbf{s}^l \rangle$$

Convergence control



Line-search

- Employ globally convergent optimizer (LS method) on each level
- Find α^{l+1}, such that

$$h^{l+1}(\boldsymbol{\theta}_{\mu_1}^{l+1} + \alpha^{l+1}\mathbf{I}_l^{l+1}\mathbf{s}_*^l) < h^{l+1}(\boldsymbol{\theta}_{\mu_1}^{l+1})$$

Trust-region

- Employ globally convergent optimizer (TR method) on each level
- Preserve fine-level TR constraint, i.e., $\|\mathbf{I}_l^{l+1}\mathbf{s}_*^l\|_p \leq \Delta_{\mu_1}^{l+1}$
- Accept $\mathbf{I}_l^{l+1}\mathbf{s}_*^l$ only if $\rho_{\mu_1+1}^{l+1} \geq \eta$, where $\eta > 0$ and

$$\rho_{\mu_1+1}^{l+1} = \frac{h^{l+1}(\boldsymbol{\theta}_{\mu_1}^{l+1}) - h^{l+1}(\boldsymbol{\theta}_{\mu_1}^{l+1} + \mathbf{I}_l^{l+1}\mathbf{s}_*^l)}{h^l(\mathbf{P}_{l+1}^l\boldsymbol{\theta}_{\mu_1}^{l+1}) - h^l(\boldsymbol{\theta}_*^l)}$$

$$= \frac{\text{fine-level decrease}}{\text{coarse-level decrease}}$$

Construction of multilevel hierarchy and transfer operators by exploring finite-sum structure of the loss function

Multilevel variance reduction (MLVR) method

Low-cost models:

- Construction of multilevel hierarchy by coarsening in number of samples

$$\Downarrow$$

hierarchy of datasets:

$$|\mathcal{D}^1| \leq \cdots \leq |\mathcal{D}^L| := |\mathcal{D}|$$

$$\boxed{\mathcal{D}^3 := \mathcal{D}} \qquad \mathcal{L} := \frac{1}{|\mathcal{D}|} \sum_{j=1}^{|\mathcal{D}|} \ell\big(f_m(\boldsymbol{x}_j, \boldsymbol{\theta}), \boldsymbol{c}_j\big)$$

$$\boxed{\mathcal{D}^2 \subset \mathcal{D}} \qquad \mathcal{L}^2 := \frac{1}{|\mathcal{D}^2|} \sum_{j=1}^{|\mathcal{D}^2|} \ell\big(f_m(\boldsymbol{x}_j, \boldsymbol{\theta}), \boldsymbol{c}_j\big)$$

$$\boxed{\mathcal{D}^1 \subset \mathcal{D}} \qquad \mathcal{L}^1 := \frac{1}{|\mathcal{D}^1|} \sum_{j=1}^{|\mathcal{D}^1|} \ell\big(f_m(\boldsymbol{x}_j, \boldsymbol{\theta}), \boldsymbol{c}_j\big)$$

Transfer operators:

- Parameter space remains same

$$\Downarrow$$

Transfer operators are identity!

- Different choice of $\{\mathcal{D}^l\}_{l=1}^L$, coarse-level models and level optimizers give rise to novel as well as existing algorithms, e.g., SSN[Bollapragada et al., '18], SVRG[Johnson et al, '13], SPIDER[Fang et al., '18], ...

  $\Longrightarrow$ allows for theoretical analysis of multiple methods using one algorithmic framework

Example: Subsampled Newton as a special case of two level MLVR method

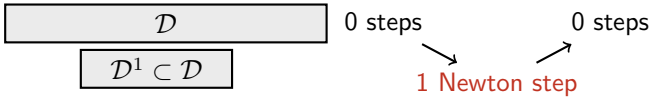
Construction of coarse-level models (1st-order additive approach) - two level settings:

$$h^1(\boldsymbol{\theta}) := \mathcal{L}^1(\boldsymbol{\theta}) + \langle \underbrace{R\nabla\mathcal{L}(\tilde{\boldsymbol{\theta}}) - \nabla\mathcal{L}^1(P\tilde{\boldsymbol{\theta}})}_{\delta \mathbf{g}^1}, \boldsymbol{s}^1 \rangle,$$

$$:= \frac{1}{|\mathcal{D}^1|}\sum_{j\in\mathcal{D}^1} \ell\big(f_m(\boldsymbol{x}_j, \boldsymbol{\theta}), \boldsymbol{c}_j\big) + \Big\langle \underbrace{\frac{1}{|\mathcal{D}|}\sum_{j\in\mathcal{D}} \nabla\ell\big(f_m(\boldsymbol{x}_j, \tilde{\boldsymbol{\theta}}), \boldsymbol{c}_j\big) - \frac{1}{|\mathcal{D}^1|}\sum_{j\in\mathcal{D}^1} \nabla\ell\big(f_m(\boldsymbol{x}_j, \tilde{\boldsymbol{\theta}}), \boldsymbol{c}_j\big)}_{\delta \mathbf{g}^1}, \boldsymbol{s}^1 \Big\rangle$$

- $\tilde{\boldsymbol{\theta}} := \boldsymbol{\theta}^2_{\mu_1} := \boldsymbol{\theta}^1_0$ is the initial guess on coarse level $(l = 1)$, i.e. projected iterate from the fine level

Example: Subsampled Newton as a special case of two level MLVR method



$$h^1(\boldsymbol{\theta}) := \frac{1}{|\mathcal{D}^1|}\sum_{j\in\mathcal{D}^1}\ell\big(f_m(\boldsymbol{x}_j,\boldsymbol{\theta}),\boldsymbol{c}_j\big) + \left\langle \frac{1}{|\mathcal{D}|}\sum_{j\in\mathcal{D}}\nabla\ell\big(f_m(\boldsymbol{x}_j,\tilde{\boldsymbol{\theta}}),\boldsymbol{c}_j\big) - \frac{1}{|\mathcal{D}^1|}\sum_{j\in\mathcal{D}^1}\nabla\ell\big(f_m(\boldsymbol{x}_j,\tilde{\boldsymbol{\theta}}),\boldsymbol{c}_j\big), \boldsymbol{s}^1\right\rangle$$

The V-cycle of MLVR method then produces the following update rule

$$\tilde{\boldsymbol{\theta}} \leftarrow \tilde{\boldsymbol{\theta}} + I(-\alpha\big(\nabla^2 h^1(P\tilde{\boldsymbol{\theta}})\big)^{-1}\nabla h^1(P\tilde{\boldsymbol{\theta}}))$$

$$\tilde{\boldsymbol{\theta}} \leftarrow \tilde{\boldsymbol{\theta}} - \alpha\underbrace{\left(\frac{1}{|\mathcal{D}^1|}\sum_{j\in\mathcal{D}^1}\nabla^2\ell\big(f_m(\boldsymbol{x}_j,\tilde{\boldsymbol{\theta}}),\boldsymbol{c}_j\big)\right)}_{\text{Subsampled Hessian}}^{-1}\underbrace{\frac{1}{|\mathcal{D}|}\sum_{j\in\mathcal{D}}\nabla\ell\big(f_m(\boldsymbol{x}_j,\tilde{\boldsymbol{\theta}}),\boldsymbol{c}_j\big)}_{\text{Full gradient}},$$

MLVR - traditional MG configuration and line-search globalization strategy

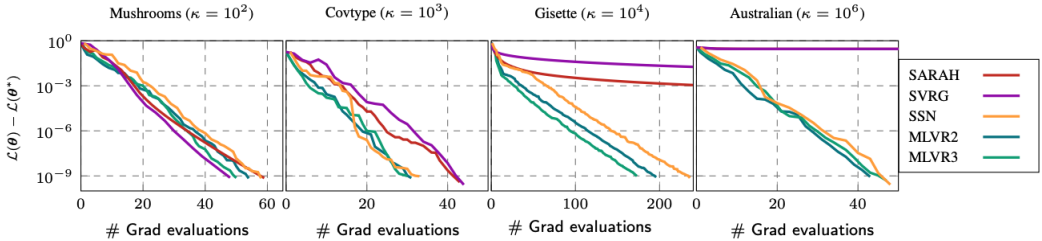
μ_1 smoothing steps — $\mathcal{D}$

μ_1 smoothing steps — $\mathcal{D}^2 \subset \mathcal{D}$

1 Newton step — $\mathcal{D}^1 \subset \mathcal{D}^2$

ℓ_2-regularized logistic loss:

$$\mathcal{L}(\boldsymbol{\theta}) := \frac{1}{p}\sum_{j=1}^{p} \log\left(1 + e^{-\boldsymbol{c}_j(\boldsymbol{\theta}^T \boldsymbol{x}_j)}\right) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$$



Training error, $\mathcal{L}(\boldsymbol{\theta}) - \mathcal{L}(\boldsymbol{\theta}^*)$, with respect to effective gradient evaluations for SVRG, SARAH, sub-sampled Newton (SSN), two and three level variants of MLVR method (MLVR2, MLVR3).

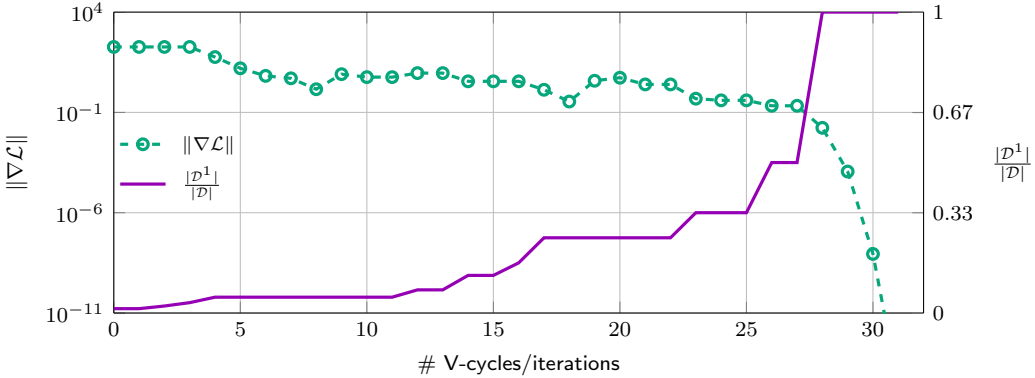
ASTROM - Adaptive Sub-sampled Trust-RegiOn Method

- Employ trust-region globalization strategy
- Recall, quality of the coarse-level correction $\mathbf{s}_*^l$ is measured by means of $\rho_{\mu_1+1}^{l+1}$, where

$$\rho_{\mu_1+1}^{l+1} = \frac{h^{l+1}(\tilde{\boldsymbol{\theta}}) - h^{l+1}(\tilde{\boldsymbol{\theta}} + \mathbf{s}_*^l)}{h^l(\tilde{\boldsymbol{\theta}}) - h^l(\tilde{\boldsymbol{\theta}} + \mathbf{s}_*^l)} = \frac{\text{fine-level decrease}}{\text{coarse-level decrease}}$$

- Utilize $\rho_{\mu_1+1}^{l+1}$ to dynamically adapt coarse-level approximation, i.e. $\mathcal{L}^l$
  - If $\rho_{\mu_1+1}^{l+1} > \eta_1$, accept coarse level correction, keep the dataset $\mathcal{D}^l$ as is
  - If $\rho_{\mu_1+1}^{l+1} \leq \eta_1$, reject coarse level correction, increase number of samples in the dataset $\mathcal{D}^l$
- As the iterative process progresses, ASTROM becomes batch TR

ASTROM (2 levels) - typical convergence behavior



Convergence history of ASTROM method for the logistic regression example with the australian dataset. An initial number of samples on coarse level equals to 2% of the whole dataset $\mathcal{D}$.

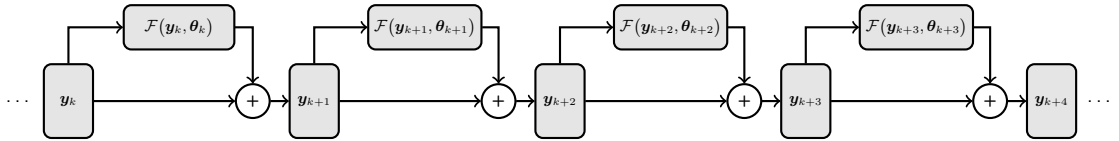
ASTROM (2 levels) vs. SSN vs. Batch TR (Gissette dataset)

Method	**TR**	**ASTROM**				**SSN (lr)**			
$\lvert\mathcal{D}^1\rvert/\lvert\mathcal{D}\rvert$	**100%**	**1%**	**10%**	**25%**	**50%**	**1% (0.075)**	**10% (0.1)**	**25% (0.5)**	**50% (0.75)**
# iterations	16	31	16	14	13	190	143	23	21
#$\nabla\mathcal{L}$ evals	16	32	24	24	29	190	143	23	21
#$\nabla^2\mathcal{L}$ evals	16	5.9	10.1	12.4	14.8	1.9	14.3	5.75	11.5
# CG iters.	148	47.2	108.3	125.2	134.9	$1,900$	$1,430$	230	210

Convergence history of ASTROM, TR, SSN methods for logistic regression example with Gissette dataset.

Construction of multilevel hierarchy and transfer operators by exploring structure of the DNN architecture

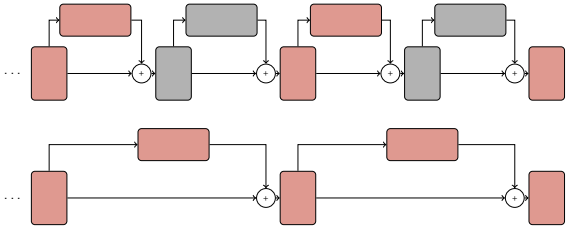
ResNets and multilevel methods [He et al., '15; He et al., '16]



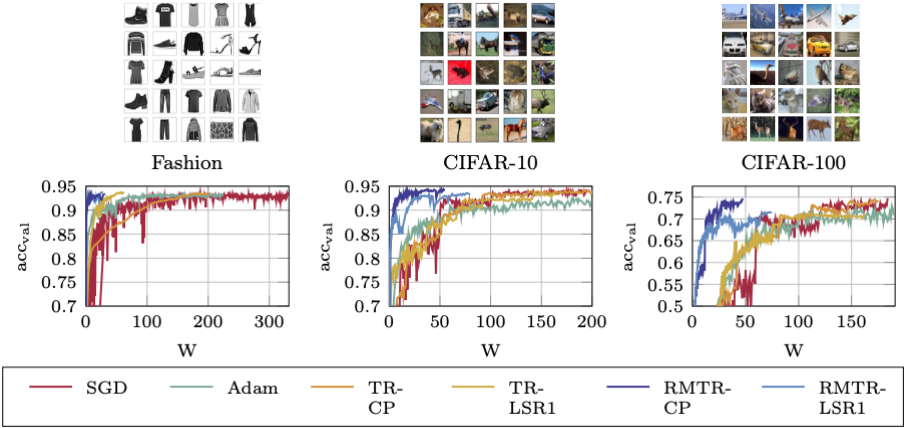
- Forward propagation:

$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + \mathcal{F}(\boldsymbol{y}_k, \boldsymbol{\theta}_k), \qquad k \in \{0, \dots, K-1\},$$
$$\boldsymbol{y}_0 \quad = \boldsymbol{Q}\boldsymbol{x},$$

- Example of resid. block:
  $\mathcal{F}(\boldsymbol{y}_k, \boldsymbol{\theta}_k) := \sigma(\boldsymbol{W}_k \boldsymbol{y}_k + \boldsymbol{b}_k)$

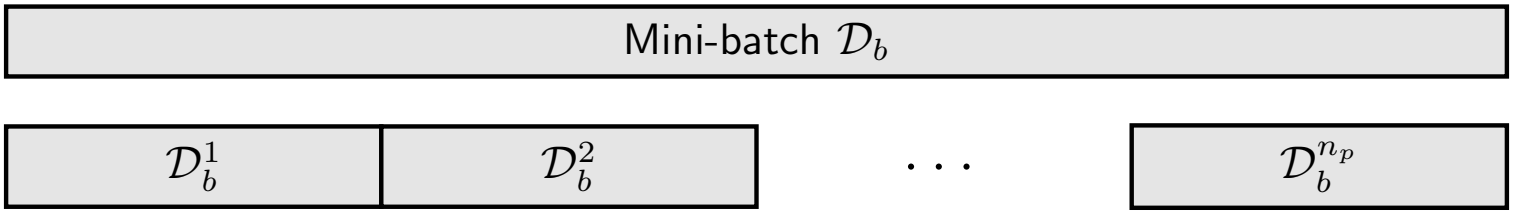


Numerical Results - Convolutional ResNets [3]



[3] Kopaničáková A., Krause R., Globally Convergent Multilevel Training of Deep Residual Networks, SIAM Journal on Scientific Computing, 2022.

Distributed learning by parallelizing in samples

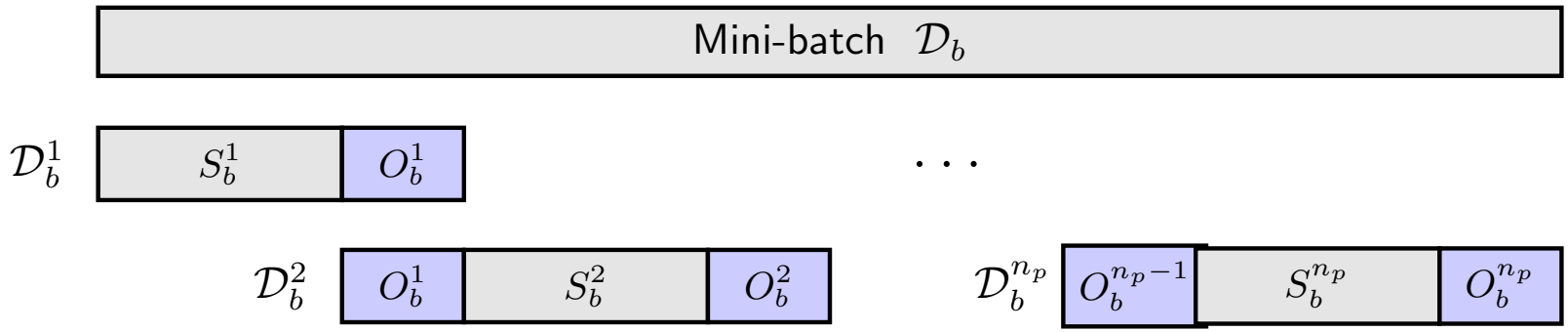


- Explore the fact that the arising minimization problems have finite sum form, i.e.,

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{j\in\mathcal{D}} \ell_j(\boldsymbol{\theta})$$

- Evaluation of objective and derivatives can be performed independently for subsets of the dataset
  - Larger mini-batches allows to explore more resources, but hinders the generalization properties of the SGD[Zinkevich et al., '10, Shallue et al., '19]
  - A synchronization step is necessary after each gradient evaluation
  - Hyper-parameter search required for learning rate

Additive preconditioned trust-region method (APTS)[Gross, Krause '09]



- Subdomains are created by decomposing the current mini-batch into n_p smaller chunks/sub-domains

  $\Longrightarrow$ suitable for any type of DNN architecture

- Non-overlapping decomposition, i.e.,

  $\mathcal{D}_b := \cup_{p=1}^{n_p} S_b^p$, where $S_b^p \subset \mathcal{D}_b$, $S_b^i \cap S_b^j = \emptyset$, for $i \neq j$

- Overlap can be constructed by repeated sampling

APTS method

Trust-region based convergence control:

- The mini-batch correction $\mathbf{s}_b$ is obtained by summing across all sub-domains, i.e.,

$$\mathbf{s}_b = \sum_{p=1}^{n_p} \mathbf{s}_b^p$$

- The number of subdomain steps and the size of the sub-domain corrections is controlled such that $\|\mathbf{s}_b\| \leq \Delta_b$

- The mini-batch correction $\mathbf{s}_b$ has to provide a decrease in the mini-batch objective function, i.e., acceptance only if $\rho_b > \eta$, where

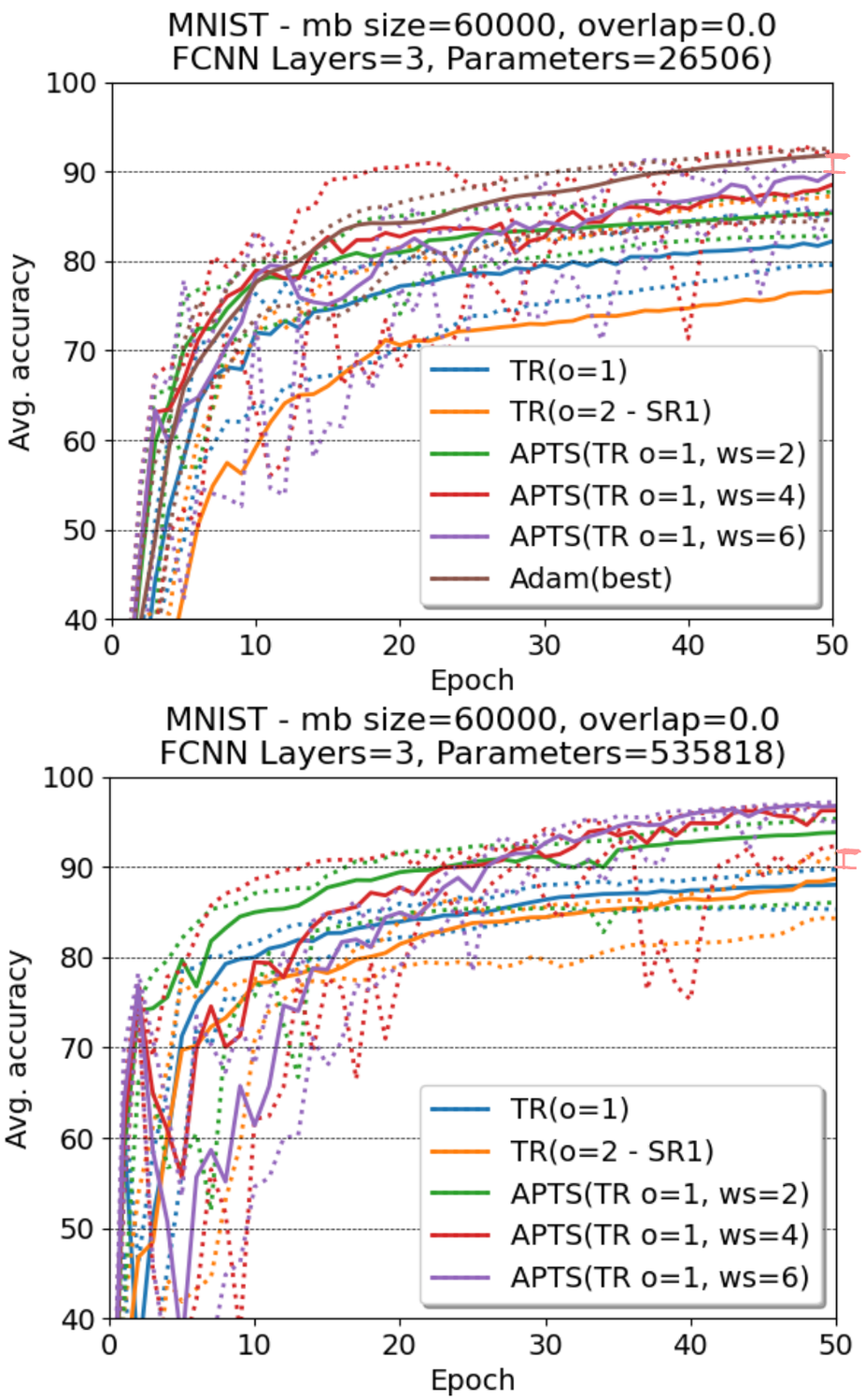
$$\rho_b = \frac{\mathcal{L}(\boldsymbol{\theta}_0, \mathcal{D}_b) - \mathcal{L}(\boldsymbol{\theta}_0 + \mathbf{s}_b, \mathcal{D}_b)}{\sum_{p=1}^{n_p} \left(h(\boldsymbol{\theta}_0, \mathcal{D}_b^p) - h(\boldsymbol{\theta}_0 + \mathbf{s}_b^p, \mathcal{D}_b^p) \right)} = \frac{\text{mini-batch decrease}}{\text{sum of sub-domain descreases}}$$

where h are sub-domain objective functions constructed using the first-order consistency approach

Università della Svizzera italiana
Euler institute

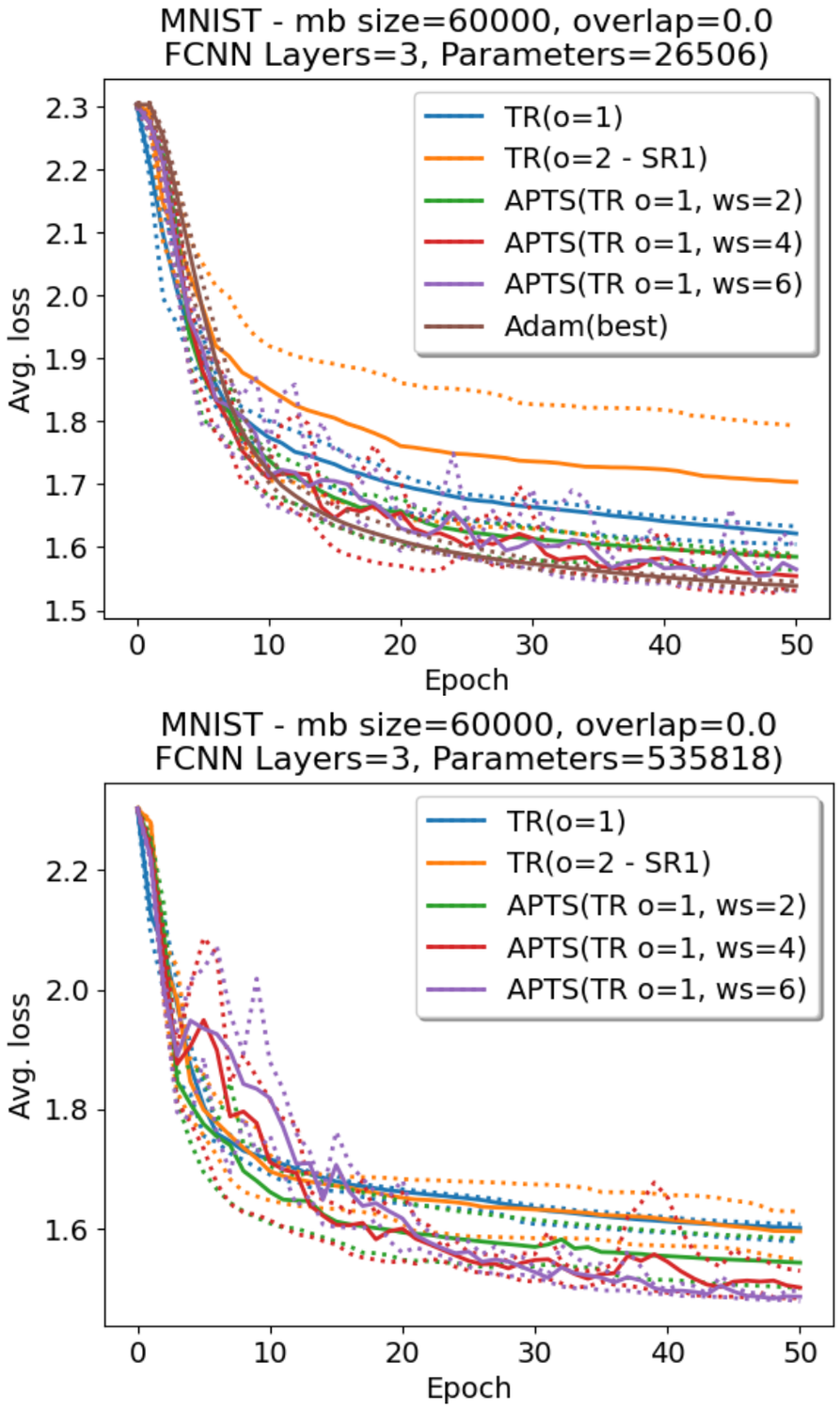
APTS in Weights

MNIST full dataset, 10 runs.
Varying network size



Small

Large

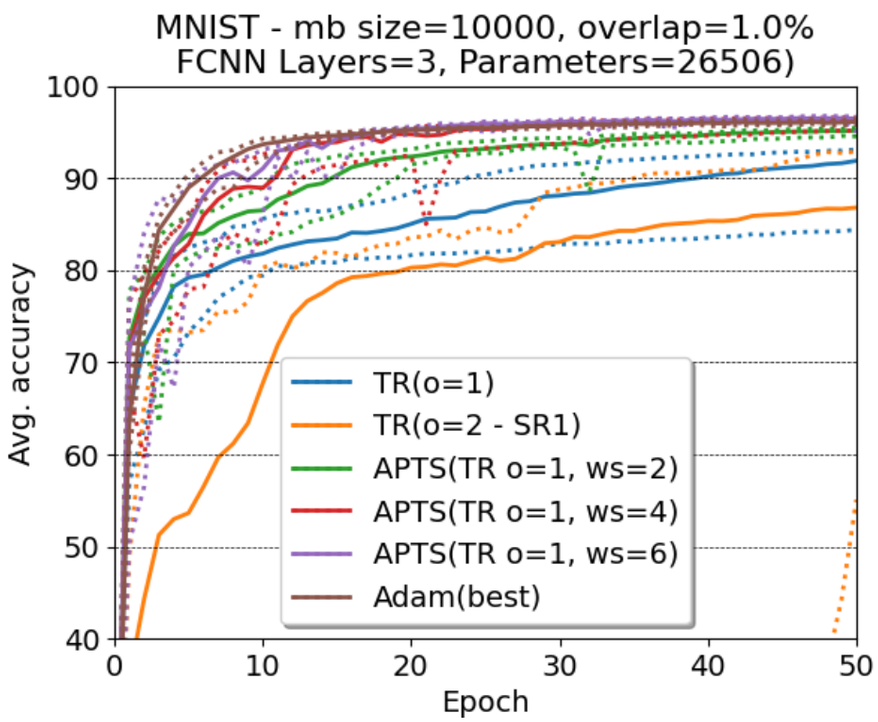


- Dotted lines are min/max accuracy or loss
- Order (o) of Taylor expansion
- World size (ws): number of processes/nodes/submodel
- Overlap (ol): ol% between all mini-batches

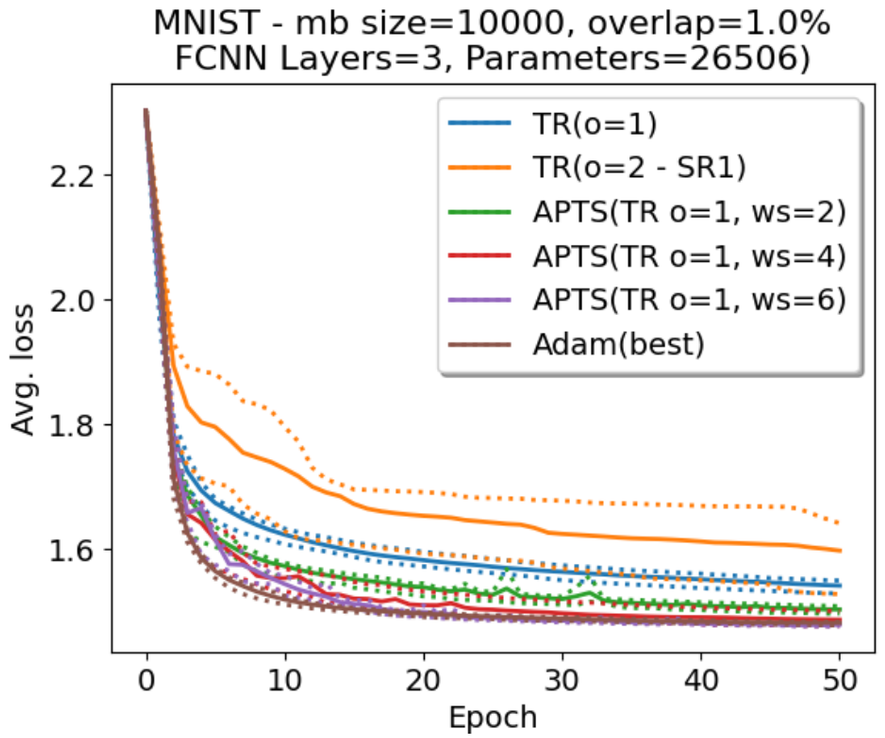
Università della Svizzera italiana
Euler institute

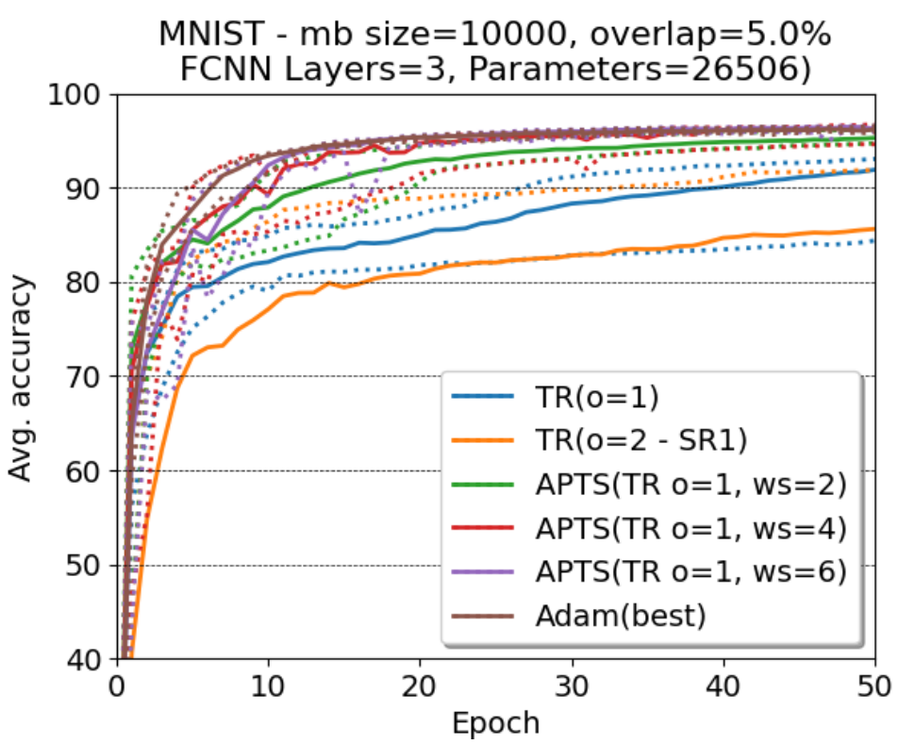
APTS in Weights

MNIST mbs=10K, 10 runs.
Small network, varying overlap

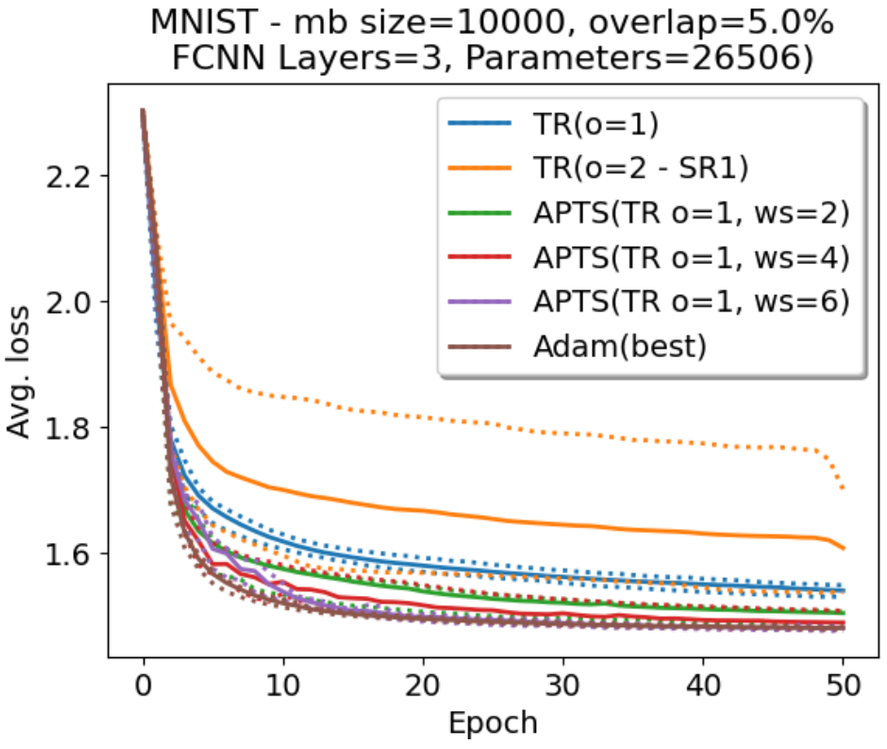


OL 1%





OL 5%

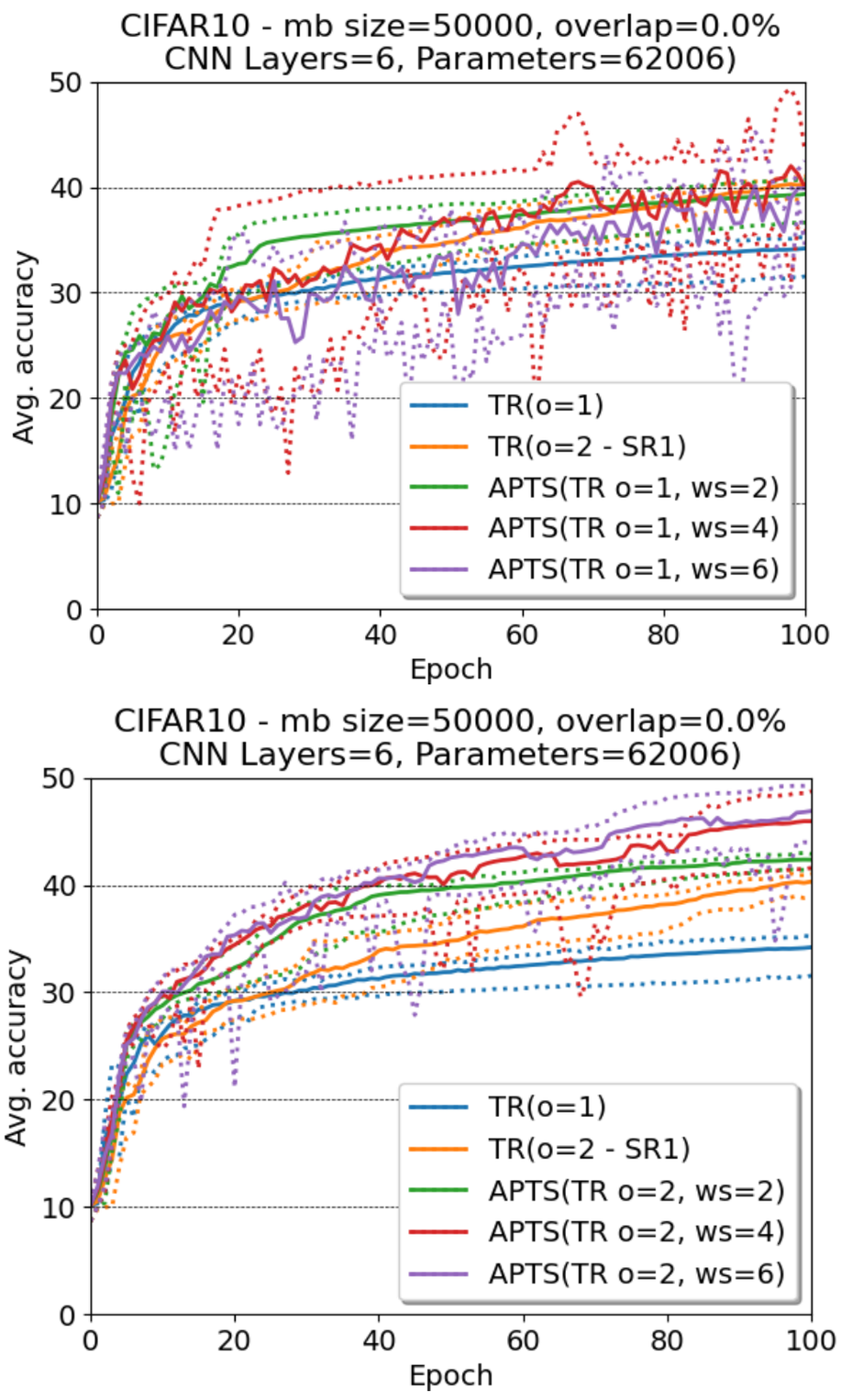


- Dotted lines are min/max accuracy or loss
- Order (o) of Taylor expansion
- World size (ws): number of processes/nodes/submodel
- Overlap (ol): ol% between all mini-batches

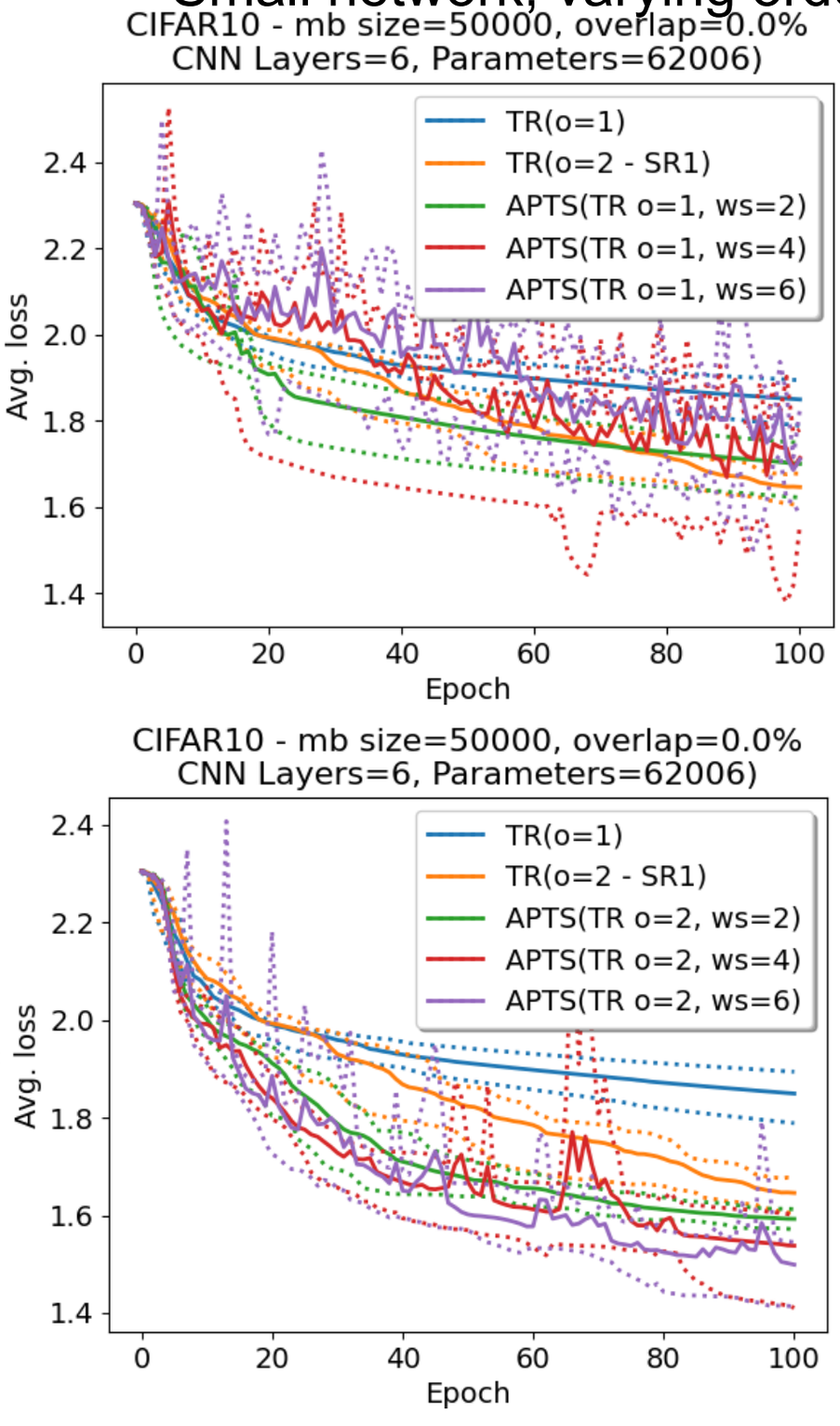
Università della Svizzera italiana
Euler institute

APTS in Weights

CIFAR full dataset, 10 runs.
Small network, varying order



Order 1



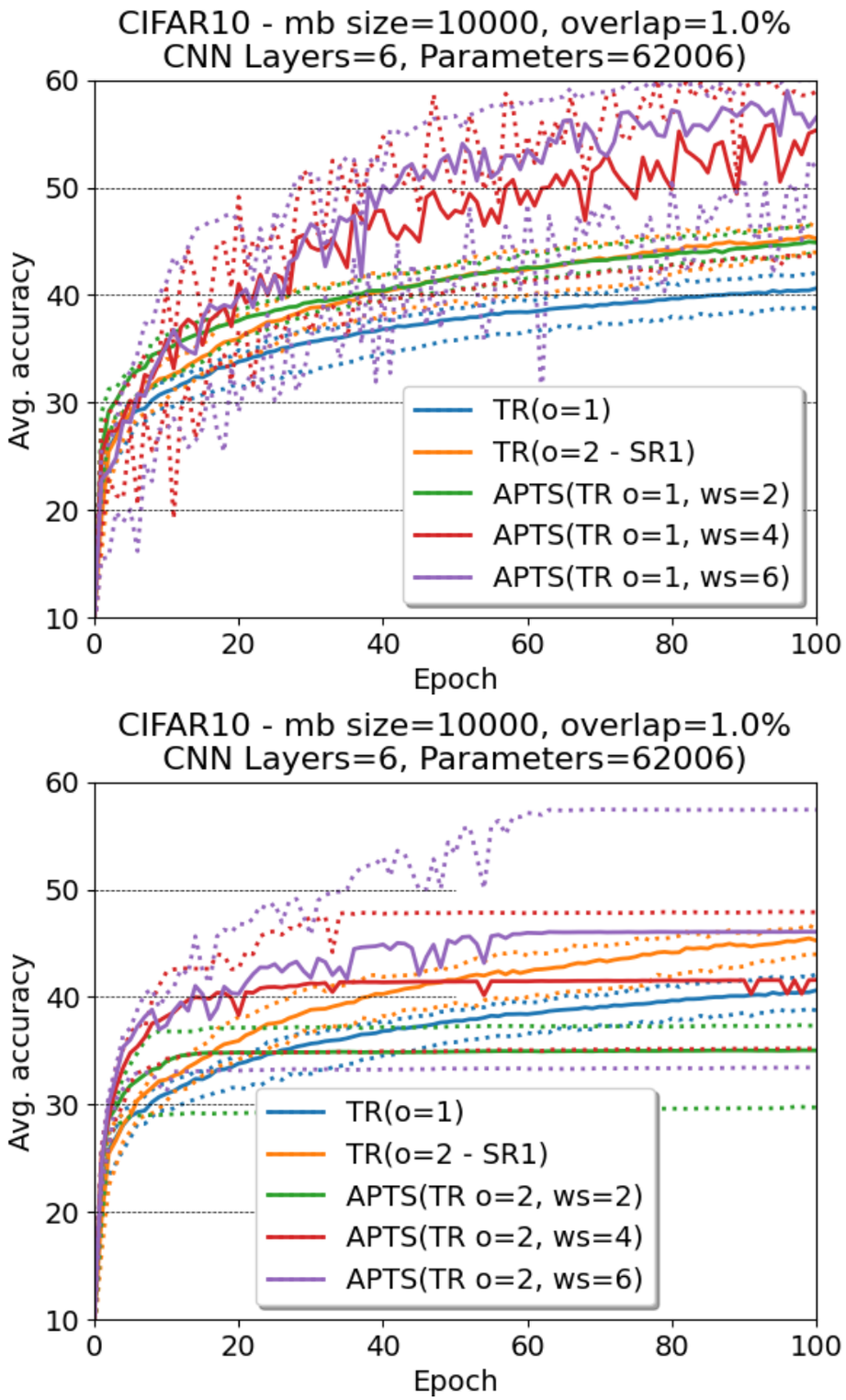
Order 2

- Dotted lines are min/max accuracy or loss
- Order (o) of Taylor expansion
- World size (ws): number of processes/nodes/submodel
- Overlap (ol): ol% between all mini-batches

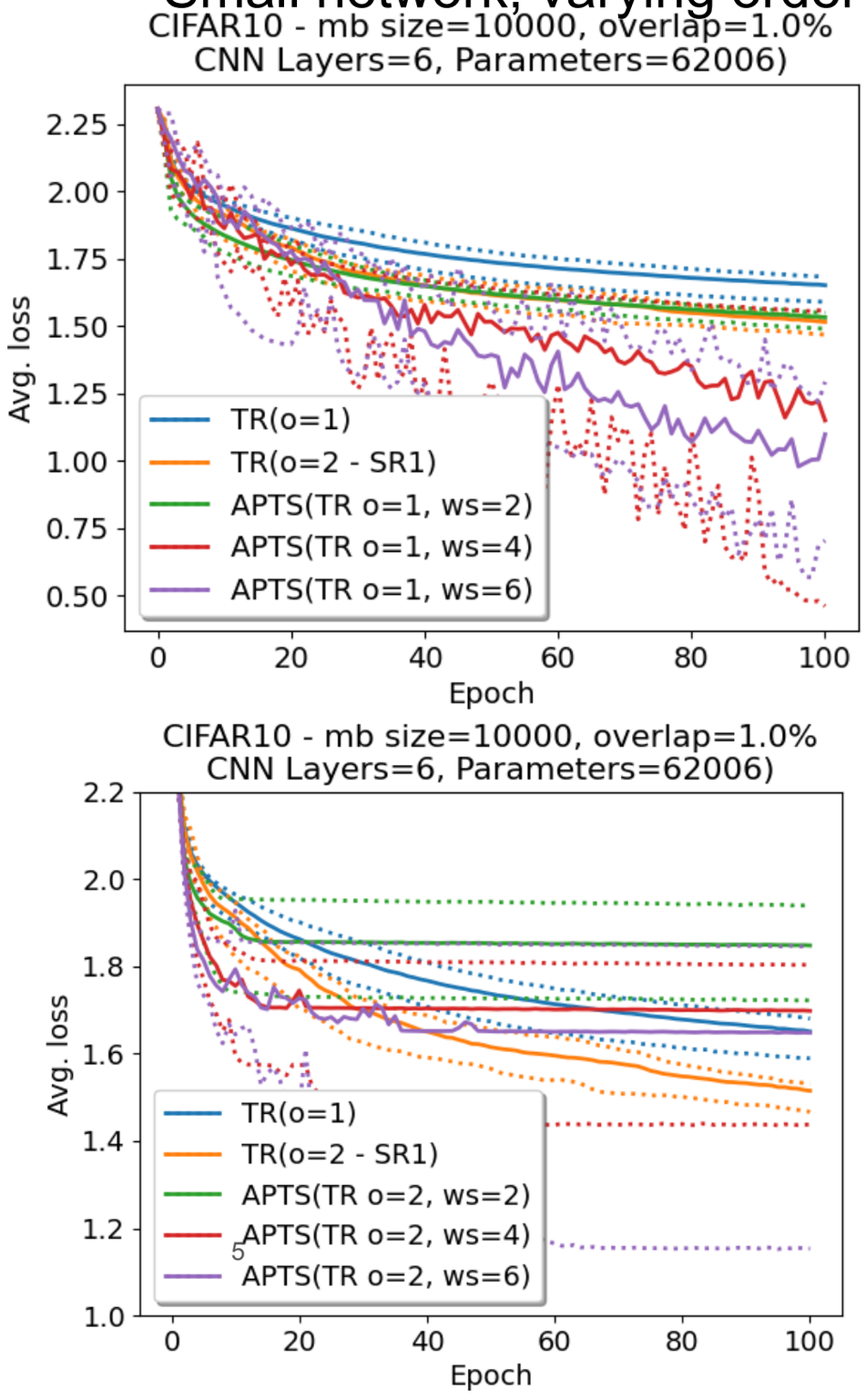
Università della Svizzera italiana
Euler institute

APTS in Weights

CIFAR mbs=10K, 10 runs.
Small network, varying order



Order 1



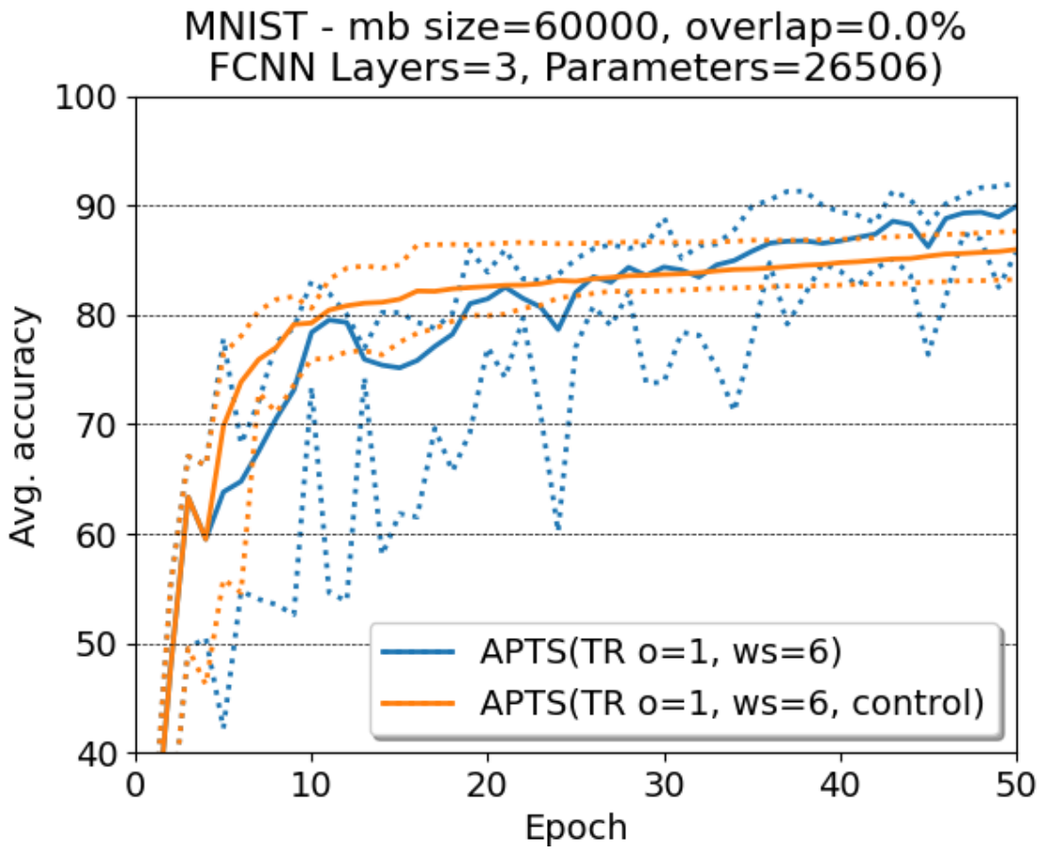
Order 2

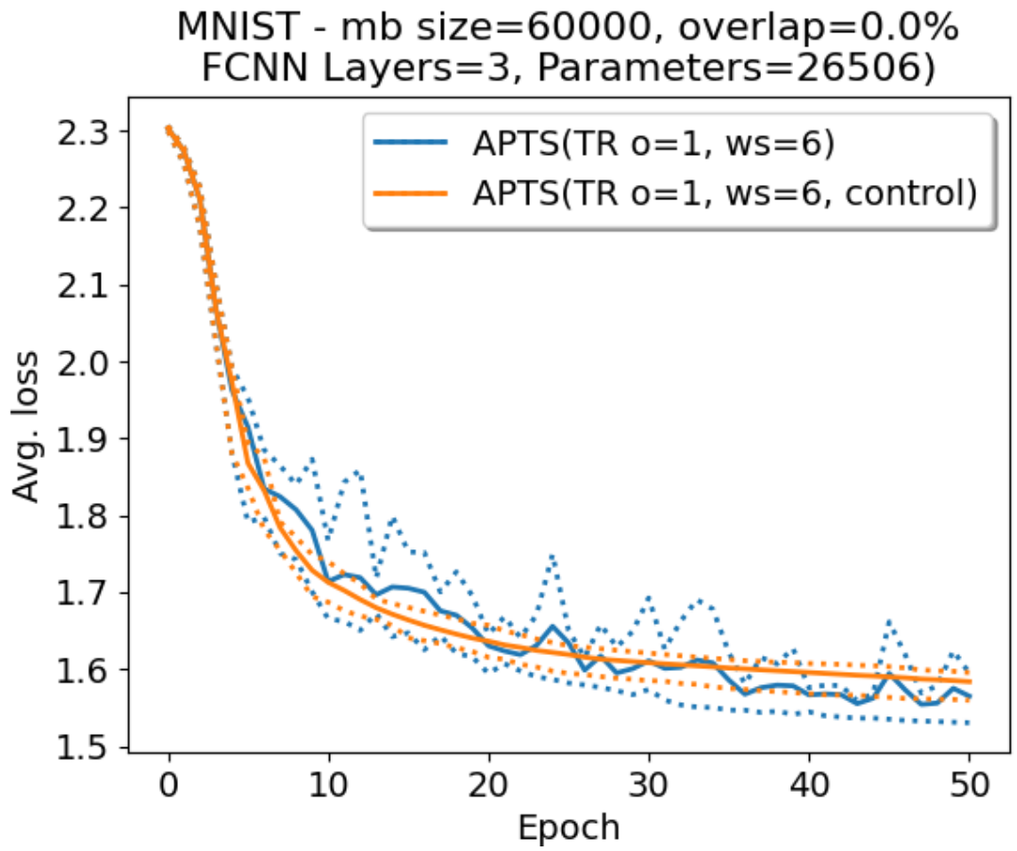
- Dotted lines are min/max accuracy or loss
- Order (o) of Taylor expansion
- World size (ws): number of processes/nodes/submodel
- Overlap (ol): ol% between all mini-batches

Università della Svizzera italiana
Euler institute

APTS in Weights

Influence of global convergence control



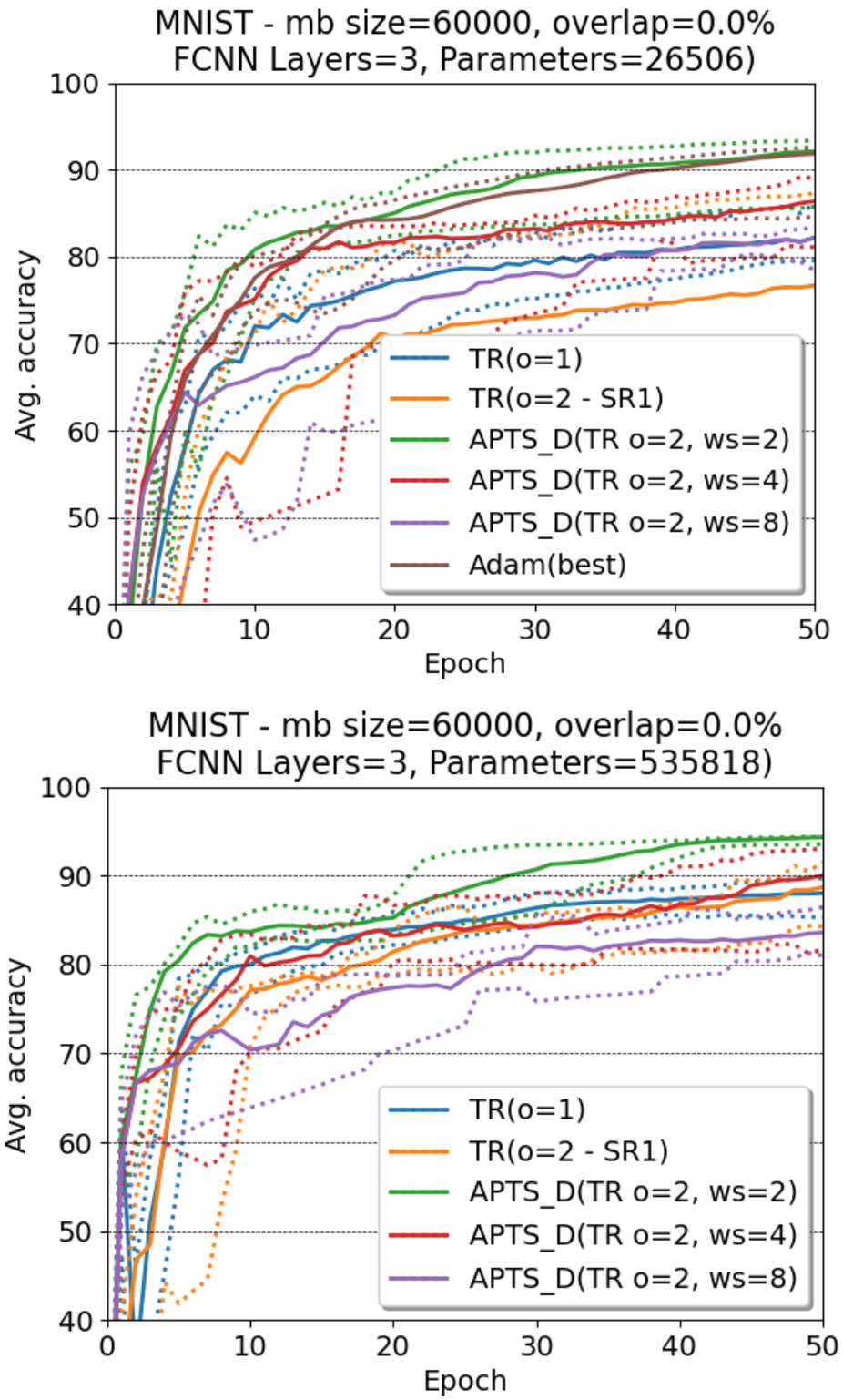


- Dotted lines are min/max accuracy or loss
- Order (o) of Taylor expansion
- World size (ws): number of processes/nodes/submodel
- Overlap (ol): ol% between all mini-batches

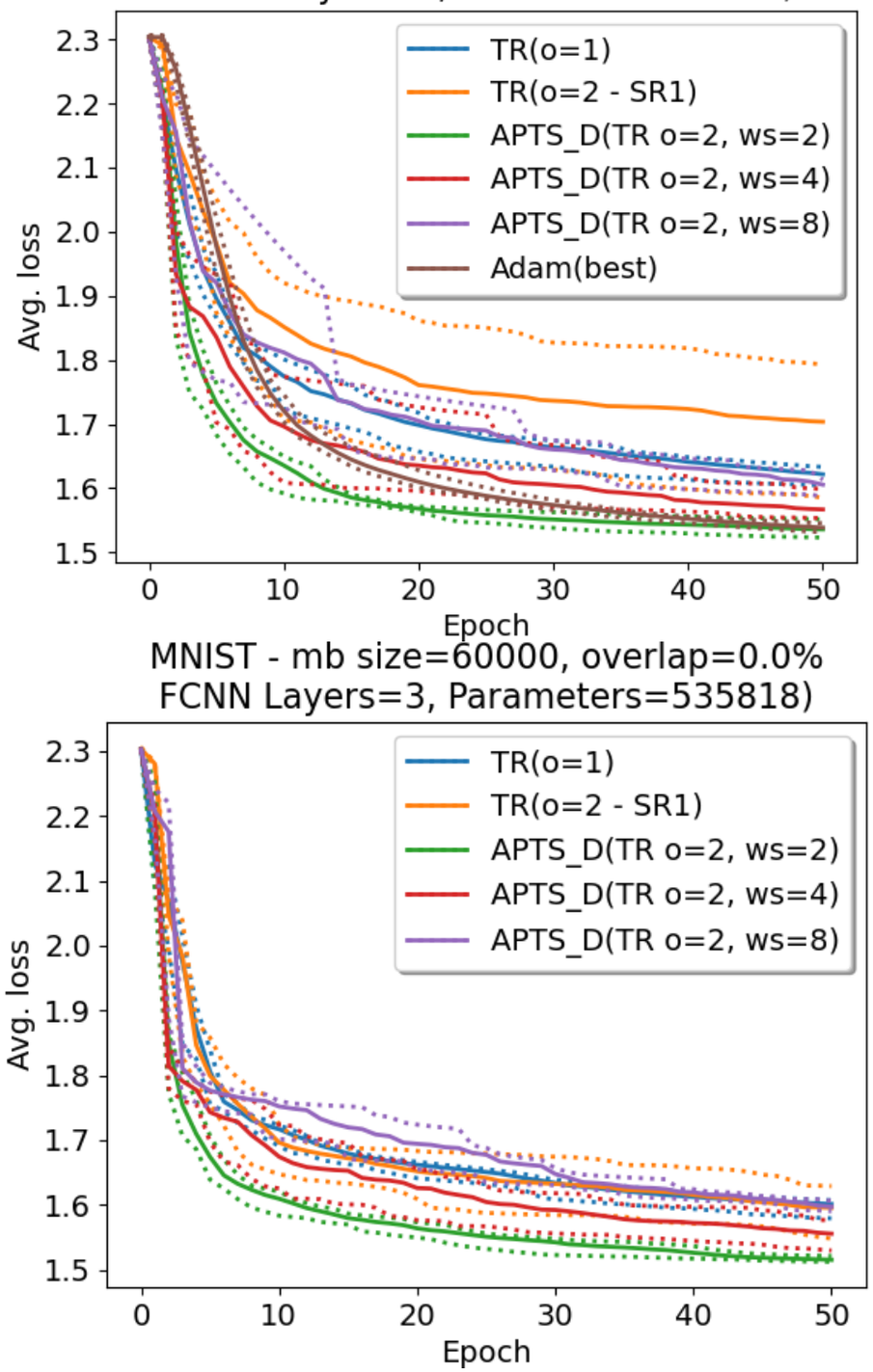
Università della Svizzera italiana
Euler institute

APTS in Data

MNIST full dataset, 10 runs.
Varying network size



Small



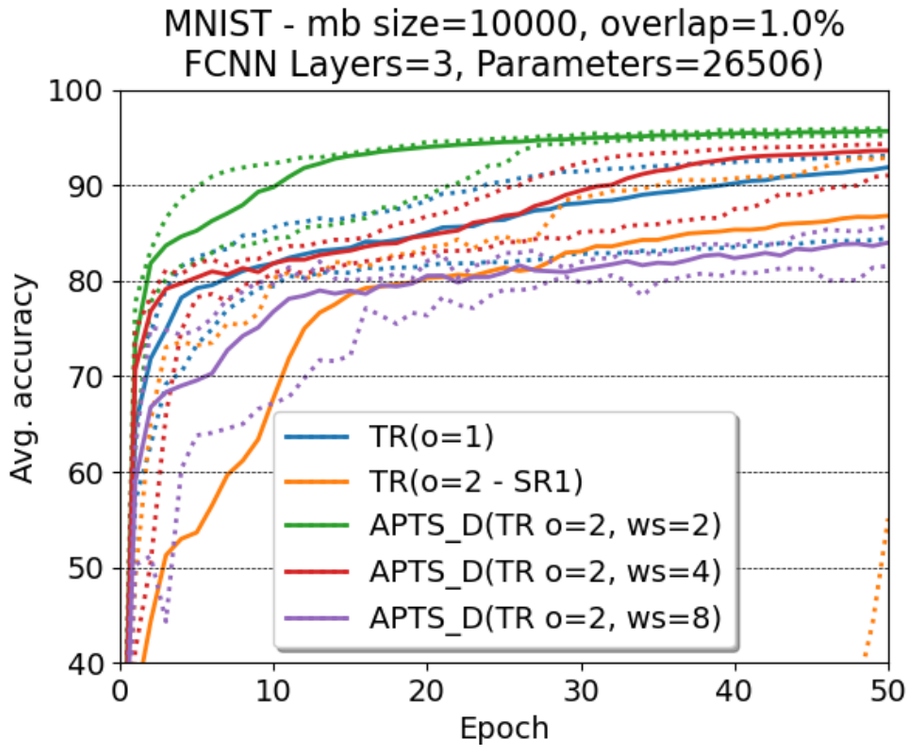
Large

- Dotted lines are min/max accuracy or loss
- Order (o) of Taylor expansion
- World size (ws): number of processes/nodes/submodel
- Overlap (ol): ol% between all mini-batches

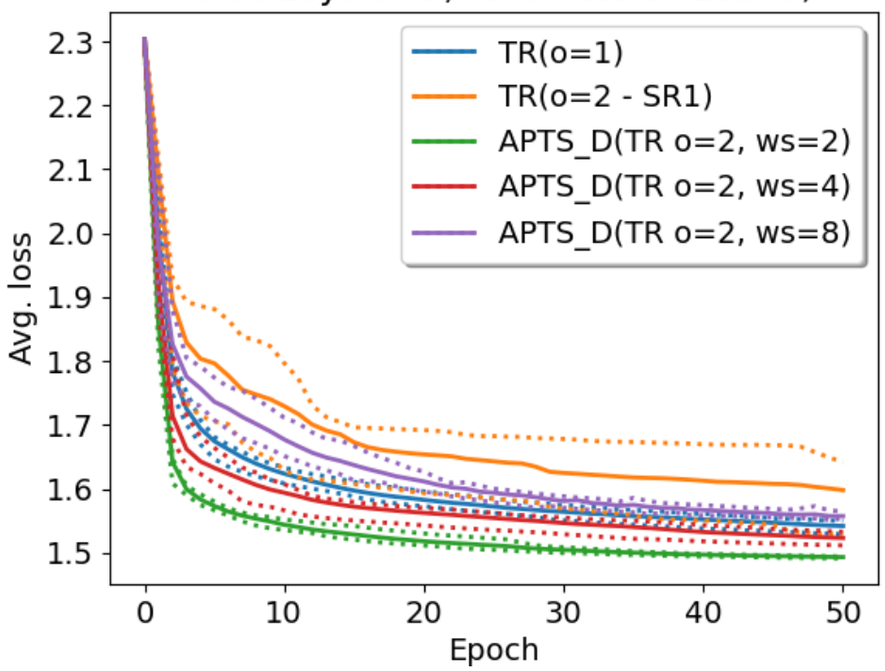
Università della Svizzera italiana
Euler institute

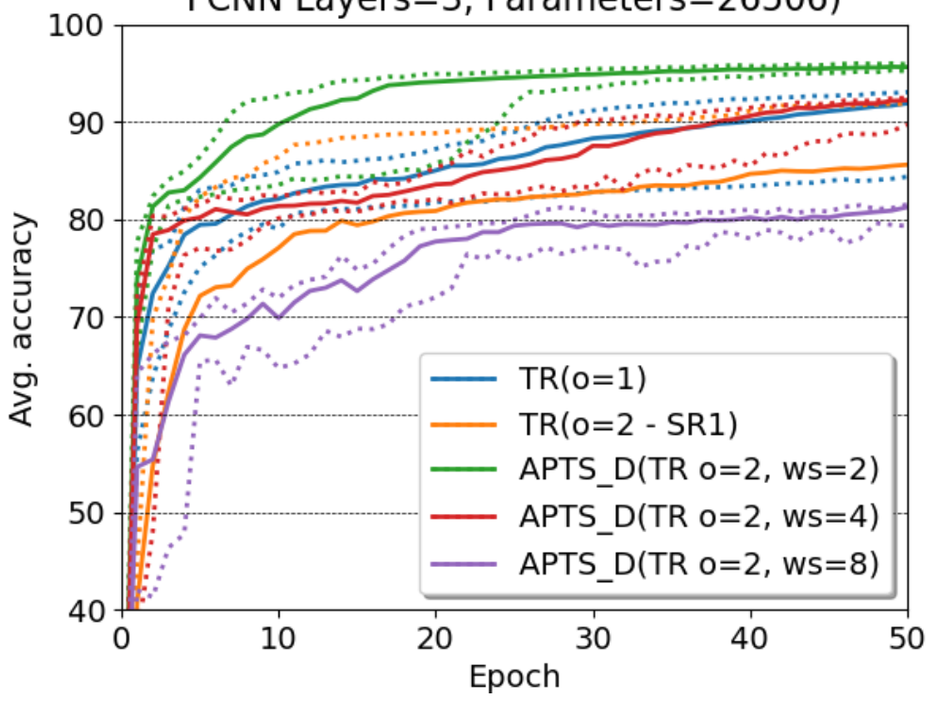
APTS in Data

MNIST mbs=10K, 10 runs.
Small network, varying overlap

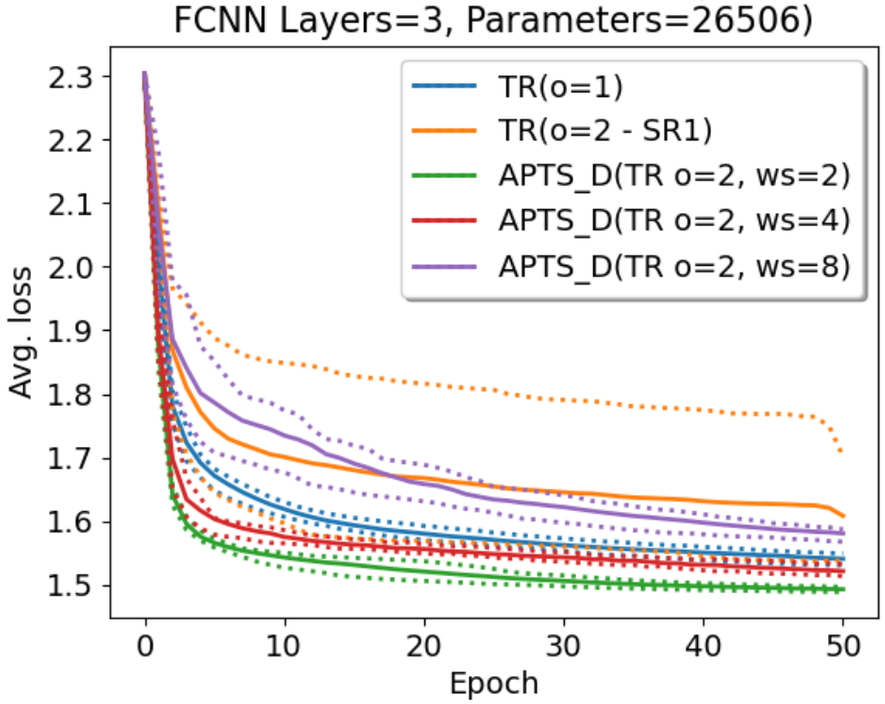


OL 1%





OL 5%



- Dotted lines are min/max accuracy or loss
- Order (o) of Taylor expansion
- World size (ws): number of processes/nodes/submodel
- Overlap (ol): ol% between all mini-batches