A multilevel training method for ANNs Application to PDEs solution

E. Riccietti (IRIT-INP, Toulouse)

Joint work with: H. Calandra (TOTAL)
S. Gratton (IRIT-INP, Toulouse)
X. Vasseur (ISAE-SUPAERO, Toulouse)

MATHIAS 2019

Serris, France - October 14-17, 2019

Context: Numerical solution of PDEs

$$D(z,u(z)) = g_1(z),\ z \in \Omega \subseteq \mathbb{R}^m;$$
$$u(z) = g_2(z),\ z \in \partial\Omega.$$

Classical approaches

- Finite differences methods

Alternative approach by Artificial Neural Networks

- Natural approach for nonlinear equations,
- Provides analytical expression of the solution,
- Provides approximation of the solution in all the points of the domain,
- Allows to alleviate the effect of the curse of dimensionality

Hot topic

Many recent papers on the use of Artificial Neural Networks to deal with Partial Differential Equations, both direct and inverse problems:

- Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data (2018)
- The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems (2017)
- A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients (2018).
- Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations (2018).
- Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations (2018).
- Solving stochastic differential equations and Kolmogorov equations by means of deep learning (2018).
- Deep Neural Networks motivated by Partial Differential Equations (2018).

Drawbacks

- The approximation of highly oscillatory solutions may require a large number of neurons.
- Gradient training methods depend on free parameters, they may be slow and better suited for convex problems

Drawbacks

- The approximation of highly oscillatory solutions may require a large number of neurons.
- Gradient training methods depend on free parameters, they may be slow and better suited for convex problems

New trend in machine learning

- Second order methods

Optimization Methods for Large-Scale Machine Learning, L. Bottou, F. E. Curtis, J. Nocedal (2018)

Second-Order Optimization for Non-Convex Machine Learning: An Empirical Study, P. Xu, F. Roosta-Khorasani, M.W. Mahoney (2018)

Drawbacks

- The approximation of highly oscillatory solutions may require a large number of neurons.
- Gradient training methods depend on free parameters, they may be slow and better suited for convex problems

New trend in machine learning

- Second order methods

Optimization Methods for Large-Scale Machine Learning, L. Bottou, F. E. Curtis, J. Nocedal (2018)

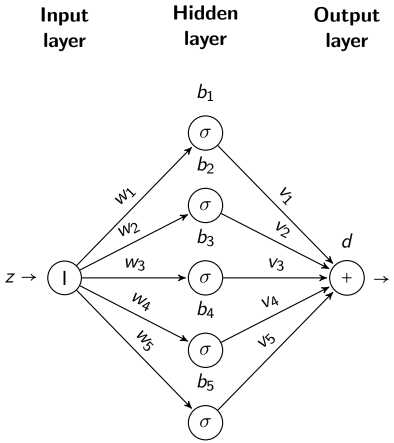
Second-Order Optimization for Non-Convex Machine Learning: An Empirical Study, P. Xu, F. Roosta-Khorasani, M.W. Mahoney (2018)

Our approach

- Use of a ANN to approximate PDEs solution trained by a Multilevel Levenberg-Marquardt method

Our approach: Artificial neural networks (1D case)

$$D(z, u(z)) = g(z),\ z \in (a, b) \quad u(a) = A,\ u(b) = B$$



$\rightarrow$ $\hat{u}(p, z) = \sum_{i=1}^{r} v_i \sigma(w_i z + b_i) + d$

$p = [v, w, b, d]^T$
$\hat{u}(p, z) \sim u(z)$

Activation funct. σ,
sigmoid: $\sigma(z) = \frac{e^z - 1}{e^z + 1}$,
tanh: $\sigma(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$,
logit: $\sigma(z) = \frac{e^z}{e^z + 1}$, soft-plus: $\sigma(z) = \log(e^z + 1)$.

Our approach: training problem

Training problem:

$$\min_{p} \mathcal{L}(\hat{u}(p,z), p; z), \qquad z \in \mathcal{T}$$

$$\hat{u}(p,z) = \sum_{i=1}^{r} v_i \sigma(w_i z + b_i) + d$$

where $\mathcal{L}$ is the loss function, $\mathcal{T}$ training set.

Our approach: training problem

Training problem:

$$\min_p \mathcal{L}(\hat{u}(p,z),p;z), \qquad z \in \mathcal{T}$$

$$\hat{u}(p,z) = \sum_{i=1}^{r} v_i \sigma(w_i z + b_i) + d$$

where $\mathcal{L}$ is the loss function, $\mathcal{T}$ training set.

We select a training set $\mathcal{T}$ s.t. $|\mathcal{T}| = t$:

$$z_T = [z_1, \dots, z_t]^T, \quad a \le z_1 < \dots < z_t \le b$$

We define

$$\mathcal{L}(\hat{u}(p,z),p;z) = \frac{1}{2t}(\|D(z_T, \hat{u}(p,z_T)) - g(z_T)\|^2 + \lambda_p(\|\hat{u}(p,a) - A\|^2 + \|\hat{u}(p,b) - B\|^2))$$

for $\hat{u}(p,z_T) \in \mathbb{R}^t$, where $u(a) = A$ and $u(b) = B$ are the boundary conditions.

Nonlinear least-squares problem

Our approach: the training method

We consider large-scale nonlinear least-squares problems:

$$\min_x f(x) = \frac{1}{2}\|F(x)\|^2$$

with $F : \mathbb{R}^n \to \mathbb{R}^m$, $m \geq n$ and $x \in \mathcal{D} \subset \mathbb{R}^n$ and n large.

Our approach: the training method

We consider large-scale nonlinear least-squares problems:

$$\min_x f(x) = \frac{1}{2}\|F(x)\|^2$$

with $F : \mathbb{R}^n \to \mathbb{R}^m$, $m \geq n$ and $x \in \mathcal{D} \subset \mathbb{R}^n$ and n large.

We propose a Multilevel extension of classical Levenberg-Marquardt method.

Classical Levenberg-Marquardt

Iterative method for nonlinear least-squares problems:

$$\min_x f(x) = \frac{1}{2}\|F(x)\|^2.$$

Classical Levenberg-Marquardt optimization method:

$$f(x_k + s) \simeq T_2(x_k, s)$$

with $T_2(x_k, s)$ Taylor model of order 2 with approximated Hessian matrix. At each iteration we compute a step s_k to update the iterate:

$$\min_s m_k(x_k, s) = T_2(x_k, s) + \frac{\lambda_k}{2}\|s\|^2, \qquad \lambda_k > 0.$$

Bottelneck: Subproblem solution

Solving

$$\min_s T_2(x_k, s) + \frac{\lambda_k}{2}\|s\|^2$$

represents greatest cost per iteration, which depends on the size of the problem.

$$\Downarrow$$

S. Gratton, A. Sartenaer, PH. Toint, 'Multilevel trust region method' 2008

→ IDEA: extend multigrid strategies to nonlinear optimization

Hierarchy of problems

- $\{f_l(x_l)\}$, $x_l \in \mathcal{D}_l$
- $|\mathcal{D}_l| < |\mathcal{D}_{l+1}|$
- f_l is cheaper to optimize compared to f_{l+1}

Multilevel setting

- At each level l, $x \in \mathbb{R}^{n_l}$. $l_{\max}$ finest level, 0 coarsest level.

$$\begin{array}{c|c|ccc}
\text{level } l_{\max} & \mathbb{R}^{n} & x^{l_{\max}} & f^{l_{\max}} = f & \mu^{l_{\max}} = f \\
\vdots & & \vdots & \vdots & \vdots \\
\text{level } l+1 & \mathbb{R}^{n_{l+1}} & x^{l+1} & f^{l+1} & \mu^{l+1} \\
 & R^{l+1} \Downarrow \quad \Uparrow P^{l+1} & & & \\
\text{level } l & \mathbb{R}^{n_l} & x^{l} & f^{l} & \mu^{l} \\
\vdots & & \vdots & \vdots & \vdots \\
\text{level } 0 & \mathbb{R}^{n_0} & x^{0} & f^{0} & \mu^{0}
\end{array}$$

- f^l represents f on the coarse spaces (it is e.g. the discretization of f on a coarse space)
- The functions μ^l are modifications of the f^l to ensure inter-level coherence.
- $R^l = \alpha (P^l)^T$, for some $\alpha > 0$.

One level strategy

At level $l = l_{\max}$, let x_k^l be the current approximation. We look for a correction s_k^l to define the new approximation $x_{k+1}^l = x_k^l + s_k^l$.

$$x_k^l$$

One level strategy

At level $l = l_{\max}$, let x_k^l be the current approximation. We look for a correction s_k^l to define the new approximation $x_{k+1}^l = x_k^l + s_k^l$.

$$x_k^l \xrightarrow{\qquad T_2^l \qquad} x_{k+1}^l = x_k^l + s_k^l$$

Multilevel strategy

Two choices:

1. minimize regularized Taylor model, get s_k^l,
2. choose lower level model μ_k^{l-1}:

x_k^l

Multilevel strategy

Two choices:

1. minimize regularized Taylor model, get s_k^l,
2. choose lower level model μ_k^{l-1}:

$$x_k^l \xrightarrow{\quad T_2^l \quad} x_{k+1}^l = x_k^l + s_k^l$$

Multilevel strategy

Two choices:

1. minimize regularized Taylor model, get s_k^l,
2. choose lower level model μ_k^{l-1}:

Multilevel strategy

Two choices:

1. minimize regularized Taylor model, get s_k^l,
2. choose lower level model μ_k^{l-1}:

x_k^l

Multilevel strategy

Two choices:

1. minimize regularized Taylor model, get s_k^l,
2. choose lower level model μ_k^{l-1}:

$$\begin{tikzcd} x_k^l \arrow[d, "R^l"'] \\ R^l x_k^l := x_{0,k}^{l-1} \end{tikzcd}$$

Multilevel strategy

Two choices:

1. minimize regularized Taylor model, get s_k^l,
2. choose lower level model μ_k^{l-1}:

$$\begin{tikzcd} x_k^l \arrow[d, "R^l"'] & \\ R^l x_k^l := x_{0,k}^{l-1} \arrow[r, "\mu_k^{l-1}"] & x_{*,k}^{l-1} \end{tikzcd}$$

Multilevel strategy

Two choices:

1. minimize regularized Taylor model, get s_k^l,
2. choose lower level model μ_k^{l-1}:

$$\begin{tikzcd}
x_k^l \arrow[d, "R^l"'] & x_{k+1}^l = x_k^l + s_k^l \\
R^l x_k^l := x_{0,k}^{l-1} \arrow[r, "\mu_k^{l-1}"] & x_{*,k}^{l-1} \arrow[u, "s_k^l = P^l(x_{*,k}^{l-1} - x_{0,k}^{l-1})"']
\end{tikzcd}$$

Multilevel strategy

Two choices:

1. minimize regularized Taylor model, get s_k^l,
2. choose lower level model μ_k^{l-1}:

$$\begin{tikzcd}
x_k^l \arrow[d, "R^l"'] & & x_{k+1}^l = x_k^l + s_k^l \\
R^l x_k^l := x_{0,k}^{l-1} \arrow[rr, "\mu_k^{l-1}"] & & x_{*,k}^{l-1} \arrow[u, "s_k^l = P^l(x_{*,k}^{l-1} - x_{0,k}^{l-1})"']
\end{tikzcd}$$

- The lower level model is cheaper to optimize.
- The procedure is recursive: more levels can be used.

Coherence between levels

Lower level model:

- Let $x_{0,k}^{l-1} = Rx_k^l$. Model with first order correction:

$$\mu_k^{l-1} = f^{l-1}(x_{0,k}^{l-1} + s^{l-1}) + (R^l \nabla f^l(x_k^l) - \nabla f^{l-1}(x_k^{l-1}))^T s^{l-1}$$

This ensures that

$$\nabla \mu_k^{l-1}(x_{0,k}^{l-1}) = R^l \nabla f^l(x_k^l)$$

$\rightarrow$ first-order behaviours of f^l and μ^{l-1} are coherent in a neighbourhood of the current approximation. If $s^l = P^l s^{l-1}$

$$\nabla f^l(x_k^l)^T s^l = \nabla f^l(x_k^l)^T P^l s^{l-1} = \nabla \mu_k^{l-1}(x_{0,k}^{l-1})^T s^{l-1}.$$

Theoretical results

Global convergence

The sequence of iterates generated by the algorithm converges globally to a first-order stationary point.

Complexity

The method requires at most $O(\epsilon^{-2})$ iterations to achieve an iterate x_k such that $\|\nabla f(x_k)\| \leq \epsilon$.

Theoretical results

Global convergence

The sequence of iterates generated by the algorithm converges globally to a first-order stationary point.

Complexity

The method requires at most $O(\epsilon^{-2})$ iterations to achieve an iterate x_k such that $\|\nabla f(x_k)\| \leq \epsilon$.

Contribution:

Generalized convergence theory from single level optimization to multilevel optimization for LM methods, much simpler proofs than for previously proposed trust-region method.

Exploit multilevel method for training of ANNs

Training problem:

$$\min_p \mathcal{L}(\hat{u}(p,z), p; z), \qquad z \in \mathcal{T}$$

$$\hat{u}(p,z) = \sum_{i=1}^{r} v_i \sigma(w_i z + b_i) + d$$

where $\mathcal{L}$ is the loss function, $\mathcal{T}$ training set.

Exploit multilevel method for training of ANNs

Training problem:

$$\min_{p} \mathcal{L}(\hat{u}(p,z), p; z), \qquad z \in \mathcal{T}$$

$$\hat{u}(p,z) = \sum_{i=1}^{r} v_i \sigma(w_i z + b_i) + d$$

where $\mathcal{L}$ is the loss function, $\mathcal{T}$ training set.

Large-scale problem: can we exploit multilevel methods for the training?

- How to build the coarse problem? The variables to be optimized are the network's weights:
  NO evident geometrical structure to exploit!

Exploit multilevel method for training of ANNs

Training problem:

$$\min_p \mathcal{L}(\hat{u}(p,z), p; z), \qquad z \in \mathcal{T}$$

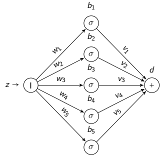
$$\hat{u}(p,z) = \sum_{i=1}^{r} v_i \sigma(w_i z + b_i) + d$$

where $\mathcal{L}$ is the loss function, $\mathcal{T}$ training set.

Large-scale problem: can we exploit multilevel methods for the training?

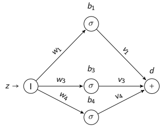
- How to build the coarse problem? The variables to be optimized are the network's weights:
  NO evident geometrical structure to exploit!
- The network possesses a purely algebraic structure: can we exploit it?

Exploit multilevel method for training of ANNs



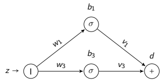
$$\mathcal{F}_1 : \mathbb{R}^{3r_1} \to \mathbb{R}$$
$$\hat{g}(p,z) = \textstyle\sum_{i \in I_1} v_i \sigma(w_i z + b_i) + d$$
$$|I_1| = r_1$$

$R_1 \Downarrow P_1 \Uparrow$



$$\mathcal{F}_2 : \mathbb{R}^{3r_2} \to \mathbb{R}$$
$$\hat{g}(p,z) = \textstyle\sum_{i \in I_2} v_i \sigma(w_i z + b_i) + d$$
$$I_2 \subset I_1,\ |I_2| = r_2 < r_1$$

$R_2 \Downarrow P_2 \Uparrow$



$$\mathcal{F}_3 : \mathbb{R}^{3r_3} \to \mathbb{R}$$
$$\hat{g}(p,z) = \textstyle\sum_{i \in I_3} v_i \sigma(w_i z + b_i) + d$$
$$I_3 \subset I_2,\ |I_3| = r_3 < r_2$$

How do we select the hierarchy of variables?

Algebraic multigrid: C/F splitting

Ruge and Stueben C/F splitting for $Ax = b$

- Two variables i, j are said to be *coupled* if $a_{i,j} \neq 0$.
- We say that a variable i is strongly coupled to another variable j, if $-a_{i,j} \geq \epsilon \max_{a_{i,k}<0} |a_{i,k}|$ for a fixed $0 < \epsilon < 1$, usually $\epsilon = 0.25$.

Prolongation-Restriction operators

$P = [I; \Delta]$, $R = P^T$.

Which matrix should we use?

We use a second-order model:

$$m(x_k, s) = f(x_k) + s^T \nabla f(x_k) + \frac{1}{2} s^T B_k s + \frac{\lambda_k}{2} \|s\|^2$$

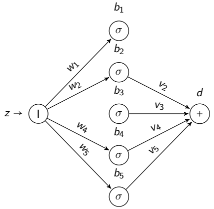
where $B_k = J(x_k)^\top J(x_k)$. At each iteration we have to solve a linear system of the form:

$$(B_k + \lambda_k I)s = -\nabla f(x_k), \quad \lambda_k > 0.$$

As in AMG for linear systems, we use information contained in matrix B_k.

Which matrix should we use?

Remark
Variables are coupled!
$\{w_i, b_i, v_i\}$



We do not use the full matrix B_k and we define A as:

$$B_k = \begin{bmatrix} f_{v,v} & .. & .. \\ .. & f_{w,w} & .. \\ .. & .. & f_{b,b} \end{bmatrix} \rightarrow A = \frac{f_{v,v}}{\|f_{v,v}\|_\infty} + \frac{f_{w,w}}{\|f_{w,w}\|_\infty} + \frac{f_{b,b}}{\|f_{b,b}\|_\infty}$$

We define the coarse/fine splitting based on the auxiliary matrix A.

Numerical tests: Choice of the true solution

$$D(z, u(z)) = g(z),\ z \in \Omega \subset \mathbb{R}^n,\ n = 1, 2$$
$$u(z) = g_2(z)\ z \in \partial\Omega$$

- We choose g to have true solution $u_T(z, \nu)$ depending on ν

Remark

- As ν increases the function becomes more oscillatory and it is harder to approximate.
- The size of the problem increases with the number of nodes.
- $\mathcal{T}$: equispaced points in $(0, 1)$ with $h = \frac{1}{3\nu}$ (Shannon's criterion).

Preliminary results: Poisson's equation 10 runs

1D		$\nu = 20$	$r = 2^9$		$\nu = 25$	$r = 2^{10}$
Solver	iter	RMSE	save	iter	RMSE	save
LM	869	1.e-4		1439	1.e-3	
MLM	507	1.e-4	1.1-2.6-4.3	1325	1.e-3	1.2-1.7-2.8

Table: 1D Poisson's equation, $u_T(z, \nu) = cos(\nu z)$, 10 runs

2D		$\nu = 5$	$r = 2^{10}$		$\nu = 6$	$r = 2^{11}$
Solver	iter	RMSE	save	iter	RMSE	save
LM	633	1.e-3		1213	1.e-3	
MLM	643	1.e-3	1.1-1.5-2.1	1016	1.e-3	1.2-1.9-2.4

Table: 2D Poisson's equation, $u_T(z, \nu) = cos(\nu z)$, 10 runs

save(min,average,max)=ratio between total number of flops required for matrix-vector products

Helmholtz's and nonlinear equations, 10 runs

	$\nu = 5$		$r = 2^{10}$
Solver	iter	RMSE	save
LM	1159	1.e-3	
MLM	1250	1.e-3	1.2-1.9-3.1

Table: Helmholtz's equations. $\Delta u(z) + \nu^2 u(z) = 0$, $u_T(z, \nu) = sin(\nu z) + cos(\nu z)$

	$\nu = 20$		$r = 2^9$	$\nu = 1$		$r = 2^9$
Method	iter	RMSE	save	iter	RMSE	save
LM	950	10^{-5}		270	10^{-3}	
MLM	1444	10^{-5}	0.8-2.9-5.3	320	10^{-3}	1.2-1.7-1.8

Table: Left: $\Delta u + \sin u = g_1$ (1D) $u_T(z, \nu) = 0.1 \cos(\nu z)$. Right: $\Delta u + e^u = g_1$ (2D), $u_T(z, \nu) = \log\left(\frac{\nu}{z_1 + z_2 + 10}\right)$

2D Helmholtz's equation

$$-\Delta u - \left(\frac{2\pi\nu}{c(z)}\right)^2 u = g_1$$

		$\nu = 1$	$r = 2^9$		$\nu = 2$	$r = 2^9$
Method	iter	RMSE	save	iter	RMSE	save
LM	200	10^{-3}		200	10^{-2}	
MLM	200	10^{-3}	1.7-1.8-1.9	200	10^{-2}	1.7-1.8-1.9

		$\nu = 2$	$r = 2^9$		$\nu = 2$	$r = 2^9$
Method	iter	RMSE	save	iter	RMSE	save
LM	200	10^{-2}		200	$5\,10^{-3}$	
MLM	200	10^{-2}	1.7-1.8-1.8	200	$5\,10^{-3}$	1.7-1.8-1.9

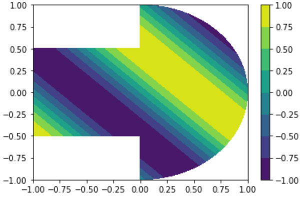
Table: In all the tests $g_1([z_1, z_2]) = (0.25 < z_1 < 0.75)(0.25 < z_2 < 0.75)$, and $c(z)$ has been chosen as: $\bar{c}_1([z_1, z_2]) = 40$ (up, left); $\bar{c}_1([z_1, z_2]) = 20\,(0 \leq z_1 < 0.5) + 40\,(0.5 \leq z_1 \leq 1)$ (up right); $\bar{c}_2([z_1, z_2]) = 20\,(0 \leq z_1 < 0.25) + 40\,(0.25 \leq z_2 \leq 0.5) + 60\,(0.5 \leq z_3 < 0.75) + 80\,(0.75 \leq z_4 \leq 1)$ (bottom, left); $\bar{c}_2([z_1, z_2]) = 0.1\sin(z_1 + z_2)$ (bottom, right).

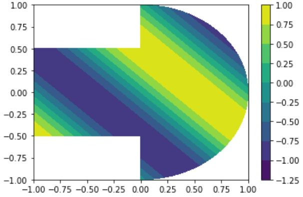
Difficult domain

2D		$\nu = 3$	$r = 2^9$
Solver	`iter`	`RMSE`	`save`
LM	395	3.e-4	
MLM	110	2.e-4	1.3-5.6-10.0

Table: 2D Screened Poisson's equation, $\Delta u - \nu^2 u = -f$, $u_T(x, y, \nu) = sin(\nu(x + y))$, 10 runs

save(min,average,max)=ratio between total number of flops required for matrix-vector products





Future work

- Design a Hessian-free variant of the method for large scale problems. The method needs to compute and store the Hessian matrix (for step computation and to build transfer operators): too expensive for large-scale problems.
- Extend to deep neural networks
- Tests on more physical/industrial/larger problems (problems in seismology)

Thank you for your attention!

For more details:

- H. Calandra, S. Gratton, E. Riccietti X. Vasseur, On the approximation of the solution of partial differential equations by artificial neural networks trained by a multilevel Levenberg-Marquardt method, submitted.
- H. Calandra, S. Gratton, E. Riccietti X. Vasseur, On high-order multilevel optimization strategies , submitted.
- H. Calandra, S. Gratton, E. Riccietti X. Vasseur, On the solution of systems of the form $A^T Ax = A^T b + c$, submitted.

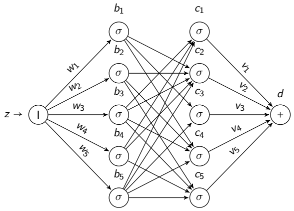
When to use the lower level model?

The lower level model is not always useful, we can use it if

- if $\|\nabla \mu_{q,k}^{l-1}(x_{0,k}^{l-1})\| = \|R^l \nabla f^l(x_k^l)\| \geq \kappa \|\nabla f^l(x_k^l)\|$, $\kappa > 0$,
- if $\|R \nabla f^l(x_k^l)\| > \epsilon^l$

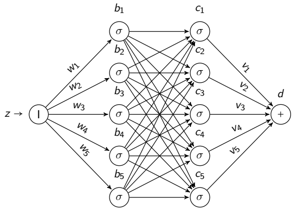
Future work 1: Extend the method to multilayer networks.

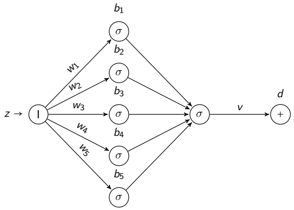
- Extend the method as it is: use a sparse network.



Future work 1: Extend the method to multilayer networks.

- Extend the method as it is: use a sparse network.
- Change strategy to build coarse problems: compress variables in a layer to exploit the structure of the multilayer network.





Coherence between levels, $q = 2$

Lower level model: Let $x_{0,k}^{l-1} = Rx_k^l$. We define $\mu_{2,k}^{l-1}$ as

$$\mu_{2,k}^{l-1}(x_{0,k}^{l-1} + s^{l-1}) = f^{l-1}(x_{0,k}^{l-1} + s^{l-1}) + (R^l \nabla f^l(x_k^l) - \nabla f^{l-1}(x_k^{l-1}))^T s^{l-1}$$
$$+ \frac{1}{2}(s^{l-1})^T((R^l)^T \nabla f^l(x_k^l) P^l - \nabla^2 f^{l-1}(x_k^{l-1}))s^{l-1}$$

Prolongation operator

$$x_i^h = (Px^H)_i = \begin{cases} x_i^H & \text{if } i \in C, \\ \sum_{k \in P_i} \delta_{i,k} x_k^H & \text{if } i \in F, \end{cases}$$

with

$$\delta_{i,k} = \begin{cases} -\alpha_i a_{i,k}/a_{i,i} & \text{if } k \in P_i^-, \\ -\beta_i a_{i,k}/a_{i,i} & \text{if } k \in P_i^+, \end{cases} \qquad \alpha_i = \frac{\sum_{j \in N_i} a_{i,j}^-}{\sum_{k \in P_i} a_{i,k}^-}, \qquad \beta_i = \frac{\sum_{j \in N_i} a_{i,j}^+}{\sum_{k \in P_i} a_{i,k}^+},$$

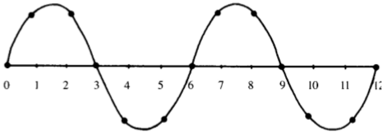
where $a_{i,j}^+ = \max\{a_{i,j}, 0\}$, $a_{i,j}^- = \min\{a_{i,j}, 0\}$, N_i is the set of variables connected to i (i.e. all j such that $a_{i,j} \neq 0$), P_i the set of coarse variables strongly connected to i, which is partitioned in P_i^- (negative couplings) and P_i^+ (positive couplings). The interpolation operator, assuming to have regrouped and ordered the variables to have all those corresponding to indexes in C at the beginning, is then defined as $P = [I; \Delta]$ where I is the identity matrix of size $|C|$ and Δ is the matrix such that $\Delta_{i,j} = \delta_{i,j}$.

Classical multigrid methods

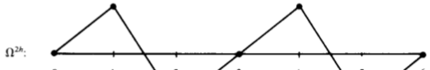
- Consider a linear elliptic PDE: $D(z, u(z)) = f(z)\ z \in \Omega$ + b.c.
- Discretize on grid h. Get a large-scale linear system $A_h x_h = b_h$.

Consider the discretization of the same PDE problem on a coarser grid: $A_H x_H = b_H$, $H > h$.

- Relaxation methods fails to eliminate smooth components of the error efficiently.
- Smooth components projected on a coarser grid appear more oscillatory.



$k = 4$ wave on $n = 12$ grid



Coarse problem construction

Define transfer grid operators: P prolongation and R restriction to project vectors from a grid to another: $x_H = Rx_h$, $x_h = Px_H$, such that $R = \alpha P^T$.

Geometry exploitation

The geometrical structure of the problem is exploited to build R and P.

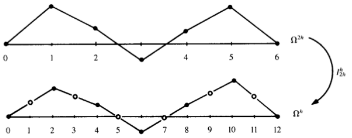


Figure 3.2: *Interpolation of a vector on coarse grid Ω^{2h} to fine grid Ω^h.*

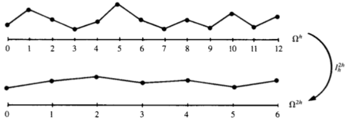


Figure 3.4: *Restriction by full weighting of a fine-grid vector to the coarse grid.*