Artificial Neural Networks : scalable optimization methods for large-scale training problems

E. Riccietti (IRIT)

Journée Deep Learning

Joint work with: H. Calandra (Total)
S. Gratton (IRIT)
X. Vasseur (ISAE-SUPAERO)

IRIT, Toulouse - 26 June 2018

The problem

We consider optimization problems arising in the training of artificial neural networks:

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

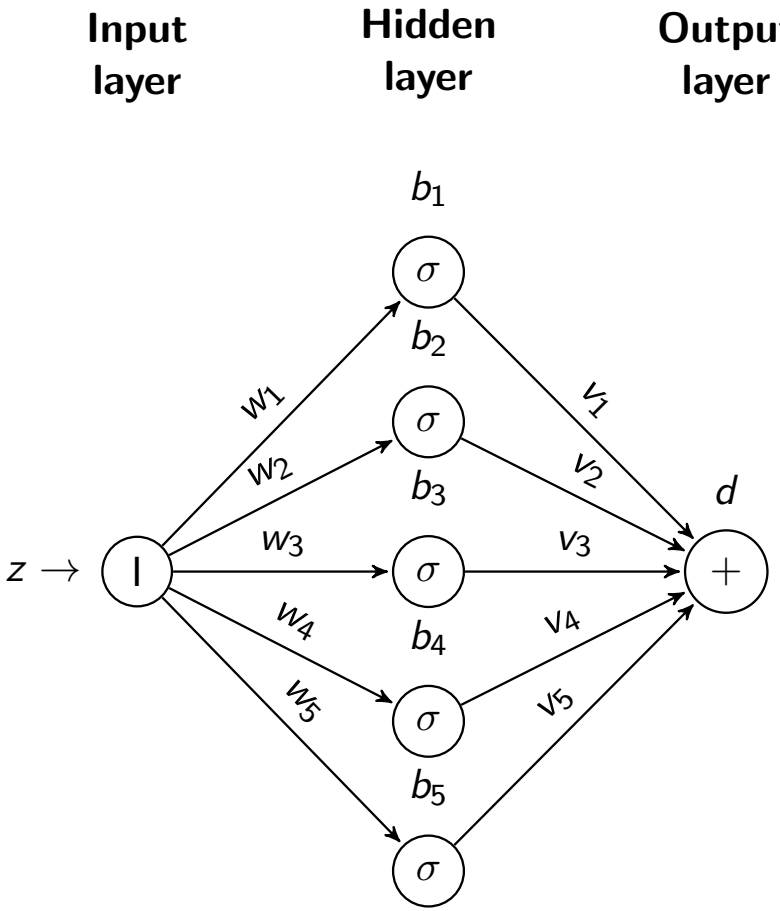
where $\mathcal{L}$ is the loss function, p is the vector of weights and biases of the network, z is the problem's variable and $\mathcal{T}$ is the training set.

Example: predict $y = g(z)$

Given a training set $\{(z_1, y_1), \ldots, (z_t, y_t)\}$ and denoted with $\hat{g}$ the output of the network, we define

- L_1 loss: $\mathcal{L}(p, z) = \frac{1}{t} \sum_{i=1}^{t} |y_i - \hat{g}(z_i, p)|$,
- L_2 loss: $\mathcal{L}(p, z) = \frac{1}{t} \sum_{i=1}^{t} (y_i - \hat{g}(z_i, p))^2$,
- Logistic loss: $\mathcal{L}(p, z) = \frac{1}{t} \sum_{i=1}^{t} \frac{1}{1+e^{y_i - \hat{g}(z_i, p)}}$.

Network architecture



$p = [v, w, b, d]^T$

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

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

Large-scale problems

The optimization problem may be a large-scale problem, for example if g is an oscillatory function. Many nodes may be necessary to have a network able to accurately approximate it.

We look for an efficient scalable optimization method to solve the training problem.

$$\Downarrow$$

Can we exploit the structure of the network?

Idea

We have to solve a large-scale problem

$$\min_p \mathcal{L}(p, z) = \mathcal{F}(\hat{g}(p, z) - y), \qquad z \in \mathcal{T}.$$

Can we exploit the structure of the network to build a hierarchy of problems approximating the original one?

Hierarchy of problems

$\{\mathcal{F}_l(\hat{g}_l(p_l, z) - y)\}$, $p_l \in \mathcal{D}_l$ such that $|\mathcal{D}_l| < |\mathcal{D}_{l+1}|$ and $\mathcal{F}_l$ is cheaper to optimize compared to $\mathcal{F}_{l+1}$.

This is the idea on which classical multigrid methods are based

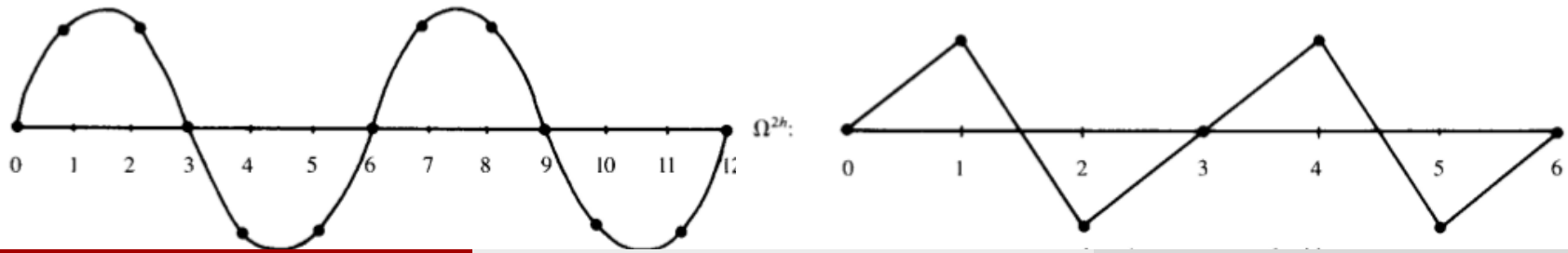
Classical multigrid methods

- 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 $A_h x_h = b_h$.

Multigrid methods

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.



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.

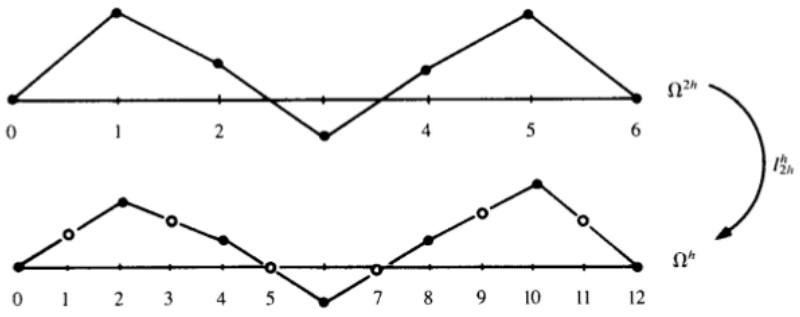


Figure 3.2: *Interpolation of a vector on coarse grid Ω^{2h} to fine grid Ω^h.*

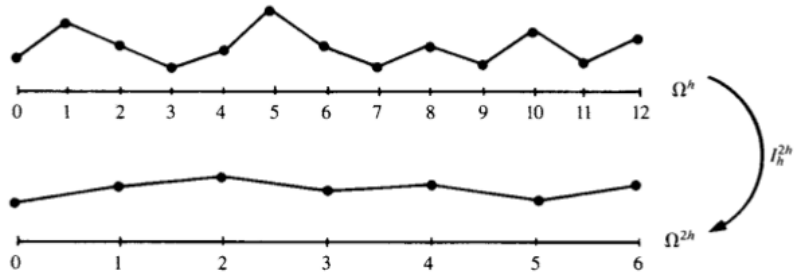


Figure 3.4: *Restriction by full weighting of a fine-grid vector to the coarse grid.*

Optimization methods

We have a nonlinear problem to solve

$$\min_x f(x)$$

Classical iterative optimization methods:

$$f(x_k + s) \simeq T_q(x_k, s) = f(x_k) + s^T \nabla f(x_k) + \frac{1}{2} s^T B_k s + \dots$$

with $T_q(x_k, s)$ Taylor model of order $q \geq 1$, B_k approximation to Hessian matrix. At each iteration we compute a step s_k to update the iterate:

$$\min_s m_k(x_k, s) = T_q(x_k, s) + \frac{\lambda_k}{q+1} \|s\|^{q+1}, \qquad \lambda_k > 0$$

- Least-squares: Levenberg-Marquardt (LM), $q = 1$, $B_k = J(x_k)^T J(x_k)$.
- Adaptive Cubic Regularization method (ARC), $q = 2$, $B_k = \nabla^2 f(x_k)$.

Basic iterative optimization algorithm

Until convergence

- Define the local model m_k of f around x_k, depending on λ_k
- Compute a trial point $x_k + s_k$ that decreases this model
- Compute the predicted reduction $m_k(x_k) - m_k(x_k + s_k)$
- Evaluate change in the objective function $f(x_k) - f(x_k + s_k)$
- If achieved change $\sim$ predicted reduction then
  - Accept trial point as new iterate $x_{k+1} = x_k + s_k$

  else
  - Reject the trial point $x_{k+1} = x_k$
  - Increase λ_k

Subproblem solution

Solving

$$\min_s T_q(x_k, s) + \frac{\lambda_k}{q+1}\|s\|^{q+1}$$

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

$$\Downarrow$$

Recursive multilevel trust region method [Gratton, Sartenaer, Toint, 2008]

Assumption

1. Assume to have at disposal a sequence of approximations $\{f_l\}$ to the objective function f such that f_l is cheaper to optimize than f_{l+1}.
2. Assume to have linear full-rank operators R_l and P_l to move from a level to another, such that $R_l = P_l^T$ (up to a scalar).

Multigrid setting

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

level $l_{\max}$	$\mathbb{R}^n$	$x^{l_{\max}}$	$f^{l_{\max}} = f$	$\mu^{l_{\max}} = f$
$\vdots$		$\vdots$	$\vdots$	$\vdots$
level $l+1$	$\mathbb{R}^{n^{l+1}}$	x^{l+1}	f^{l+1}	μ^{l+1}
	$R^{l+1} \Downarrow \quad \Uparrow P^{l+1}$			
level l	$\mathbb{R}^{n^l}$	x^l	f^l	μ^l
$\vdots$		$\vdots$	$\vdots$	$\vdots$
level 0	$\mathbb{R}^{n^0}$	x^0	f^0	μ^0

- f^l represent 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's to ensure inter-level coherence.

Coherence between levels

Lower level model:

- Model with first order correction:

$$\mu^{l-1} = f^{l-1}(x_0^{l-1} + s^{l-1}) + (R^l \nabla f^l(x_k^l) - \nabla f^{l-1}(x_k^{l-1}))^T s^{l-1}$$

This ensures that

$$\nabla \mu^{l-1}(x_0^{l-1}) = R^l \nabla f^l(x_k^l)$$

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

$$\nabla f^l(x_k^l)^T s^l = \nabla f^l(x_k^l)^T P^l s^{l-1} = \frac{1}{\alpha} \nabla \mu^{l-1}(x_0^{l-1})^T s^{l-1}.$$

Multilevel strategy

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

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

$$
\begin{array}{ccc}
x_k^l & & x_{k+1}^l = x_k^l + s_k^l \\
\Big\downarrow R^l & & \Big\uparrow s_k^l = P^l(x_*^{l-1} - x_0^{l-1}) \\
R^l x_k^l := x_0^{l-1} & \xrightarrow{\mu^{l-1}} & x_*^{l-1}
\end{array}
$$

Recursive multi-scale algorithm

Until convergence

- Choose either a Taylor or a (useful) recursive model
  - Taylor model: compute a Taylor step satisfying a sufficient decrease property
  - Recursive: apply the algorithm recursively
- Evaluate change in the objective function
- If achieved change $\sim$ predicted reduction then
  - Accept trial point as new iterate

  else
  - Reject the trial point
  - Increase λ

The algorithm is proved globally convergent to first order critical points

Our contribution

- In [Gratton, Sartenaer, Toint, 2008] second-order models are considered ($q = 2$). We extend the convergence theory to q-order models, with $q > 2$.
- We specialize recursive algorithm for least-squares problems (Levenberg-Marquardt method).
- We propose a suitable mechanism to construct a hierarchy of problems for neural network training.

How to build the coarse problem?

Remark

The variables to be optimized are the network's weights:

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

NO evident geometrical structure to exploit!

Algebraic multigrid

We can take inspiration from algebraic multigrid techniques.
When solving linear systems $Ax = b$, the structure is discovered through the matrix A. R and P are built just looking at the entries of the matrix.

Which matrix should we use?

Assume to use a second-order model. At each iteration we have to solve a linear system of the form:

$$(B_k + \tilde{\lambda}_k I)s = -\nabla f(x_k)$$

for $\tilde{\lambda}_k > 0$. As in AMG for linear systems, we use information contained in matrix B_k.

Remark

Variables are coupled! $\mathcal{L}(p, z) = \mathcal{F}(\hat{g}(p, z) - y)$ and $\hat{g}(p, z) = \sum_{i=1}^{r} v_i \sigma(w_i z + b_i) \rightarrow p = \{(v_i, w_i, b_i)\}$.

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

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

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

Application: solution of PDEs

Approximate the solution u of a PDE:

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

- We approximate $u \sim \hat{u}(p, z)$ for $p \in \mathbb{R}^n$ and we define

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

  for $z \in \mathcal{T}$ training set.
- We build g to have true solution $u_T(z, \nu)$ depending on ν
- $\mathcal{T}$: equispaced points in $(0, 1)$ with $h = \frac{1}{3\nu}$ (Shannon's criterion).

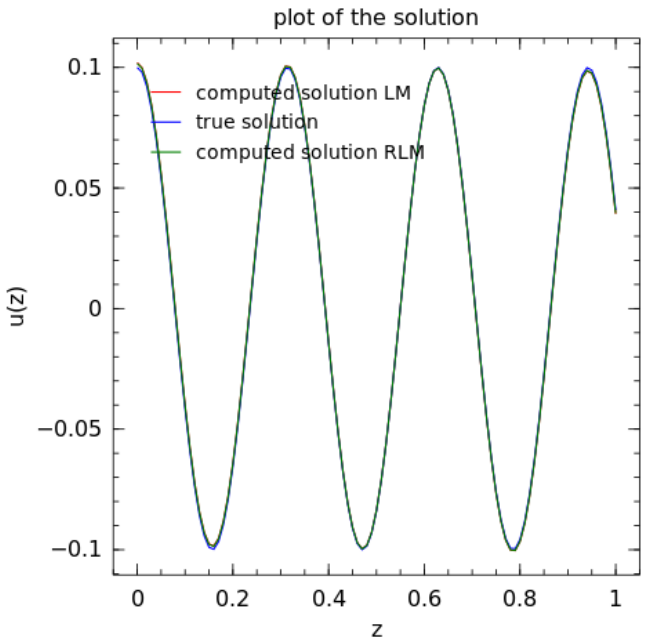
Remark

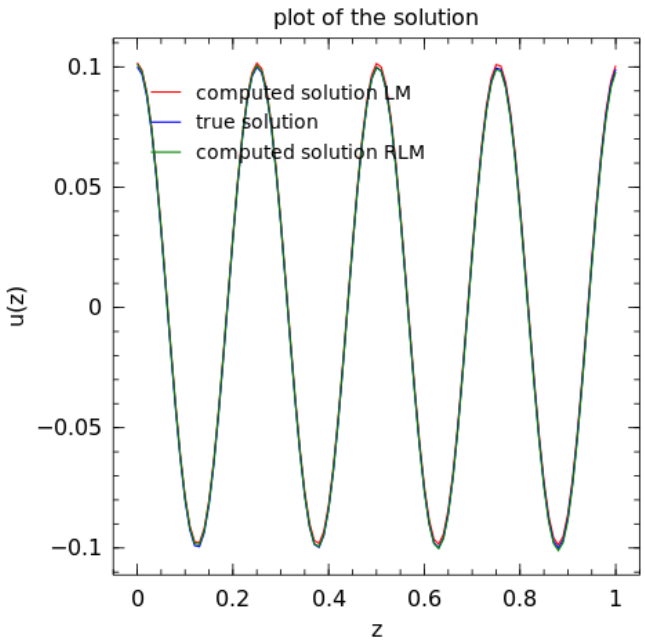
- As ν increases the function becomes more oscillatory and it is harder to approximate.
- The size of the problem increases with the number of nodes.

Poisson's equation, $u_T(z, \nu) = cos(\nu z)$, 5 runs

Problem	$\nu = 20$	$r = 2^9$		$\nu = 25$	$r = 2^{10}$	
Solver	iter	RMSE	save	iter	RMSE	save
LM	282	1.e-3		632	1.e-2-1.e-3	
RLM	193	1.e-3	1.2-1.75	347	1.e-2-1.e-3	1.2-3.15

save=ratio between total number of flops required for matrix-vector products



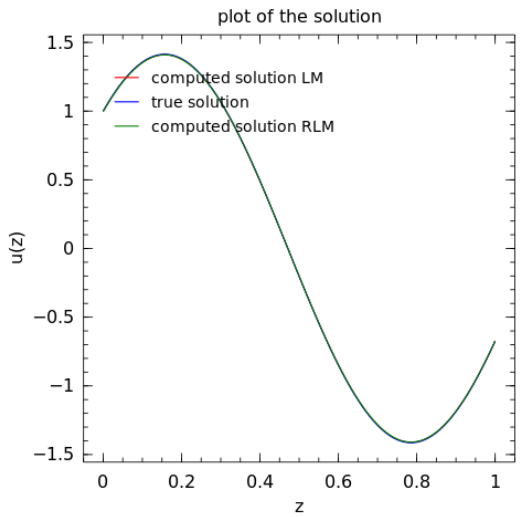


Helmholtz's equation, 5 runs

Equation: $\Delta u(z) + \nu^2 u(z) = 0$, $u_T(z, \nu) = sin(\nu z) + cos(\nu z)$

Problem		$\nu = 5$	$r = 2^{10}$
Solver	iter	RMSE	save
LM	1243	1.e-2-1.e-3	
RLM	1229	1.e-2-1.e-3	1.2-3.1

save=ratio between total number of flops required for matrix-vector products



Thank you for your attention!

Ruge and Stueben AMG

To build the coarse problem, the variables are divided into two sets, set C of coarse variables and set F of fine variables.

Ruge and Stueben C/F splitting

- 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 $\epsilon = 0.25$.
- Each F variable is required to have a minimum number of its strong couplings be represented in C. The C/F splitting is usually made choosing some first variable i to become a coarse variable. Then, all variables strongly coupled to it become F variables. The process is repeated until all variables have been split.