
Autodiff of Neural networks and application to DeepRitz and FEM

B. Després and M. Pintore
(LJLL-SU+MEGAVOLT-INRIA)

What mathematical framework for NN/ML?

Take a fully-connected **feed-forward neural network** function

$$f : \mathbb{R}^{a_0} \rightarrow \mathbb{R}^{a_{\ell+1}}$$

$$f = f_{\ell} \circ S_{\ell} \circ f_{\ell-1} \circ S_{\ell-1} \circ \cdots \circ f_1 \circ S_1 \circ f_0$$

where $x_i \mapsto f_i(x_i)$ is linear with respect to $x_i \in \mathbb{R}^{a_i}$ for all $1 \leq i \leq \ell$

$$f_i(x_i) = W_i x_i + b_i \in \mathbb{R}^{a_{i+1}}$$

and $S_i \in C^0(\mathbb{R})$ is a non linear activation function $0 \leq S'_i(x) \leq 1$.

- Sigmoidal functions $S_i = \sigma$ are smooth activation functions.
- ReLU function $S_i = R$

$$R(x) = \max(0, x)$$

is an extremely popular activation function used in production codes.

- *Max-pooling* used in **convolutional neural networks** is based on the maximum function

$$(a, b, \dots) \mapsto \max(a, b, \dots) \in \mathbb{R}.$$

Chain rule must be in the framework

Training is based on explicit evaluation of the gradient of the cost

$$\theta'(t) = -\nabla J(\theta(t)), \quad \theta = (W_0, b_0, W_1, b_1, \dots), \quad J = \text{Loss}.$$

- ML is universally based on the **chain rule/backpropagation/autodiff**

$$\nabla f(x) = A_\ell(x)B_\ell(x)A_{\ell-1}(x)B_{\ell-1}(x)\dots A_1(x)B_1(x)A_0(x)$$

which is non ambiguous in the space C^1 .

- What if $f \in \text{Lip}(\mathbb{R}^{a_0} : \mathbb{R}^{a_{\ell+1}})$?

Computer science : consider Karpathy's microgpt.py

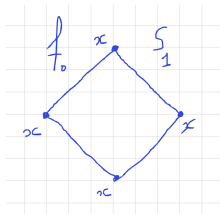
The most atomic way to train and run inference for GPT in Python. @karpathy

```
def log(self): return Value(math.log(self.data), (self,), (1/self.data,))
def exp(self): return Value(math.exp(self.data), (self,), (math.exp(self.data),))
def relu(self): return Value(max(0, self.data), (self,), (float(self.data > 0),));
```

Last line means $R'(0) = 0$, where $R(x) = \max(0, x)$.

Math : Rademacher Th. $\implies \nabla f \in L^\infty(\mathbb{R}^{a_0} : \mathcal{M}_{a_{\ell+1}, a_0}(\mathbb{R}))$.

Chain rule is meaningless in Lip class



Academic one-hidden-layer CNN,
taken from Murat-Trombetti 2003
which was on a subtle issue for the
weak maximum principle in elliptic
equations.

- $f_0 : \mathbb{R} \rightarrow \mathbb{R}^2$ is linear : $f_0(x) = W_0 x$ with $W_0 = (1, 1)$, $\nabla f_0 = W_0 = (1, 1)$.
- $S_1 : \mathbb{R}^2 \rightarrow \mathbb{R}$ is a max-pooling : $S_1(y) = \max(y_1, y_2)$ **where** $y = (y_1, y_2)$, if $y_1 > y_2$ then $\nabla S_1(y) = (1, 0)^t$; $y_1 < y_2$ then $\nabla S_1(y) = (0, 1)^t$;
if $y_1 = y_2$ **then** $\nabla S_1(y)$ **is not defined**.
- Of course $f(x) = x$ and $f'(x) = 1$. The chain rule writes

$$1 = \langle \nabla S_1 \circ f_0(x), (1, 1) \rangle.$$

- Pre-images of **zero measure sets** might be **non zero measure sets** !
Even worse, pre-images by continuous functions of Lebesgue-measurable sets may be **non measurable**.

The class of C_{pw}^1 functions

F. Murat and C. Trombetti. A chain rule formula for the composition of a vector-valued function by a piecewise smooth function. 2003.

Consider a finite decomposition of $\mathbb{R}^a$ in Borel sets or Borel pieces P^α

$$\mathbb{R}^a = \bigcup_{\alpha} P^\alpha, \quad P^\alpha \cap P^\beta = \emptyset \text{ for } \alpha \neq \beta.$$

For the sake of simplicity, replace Borel by piecewise affine.

Definition

We say that a Lipschitz-continuous function $f : \mathbb{R}^a \rightarrow \mathbb{R}^b$ is

$$C_{\text{pw}}^1 = C^1 \text{ piecewise}$$

if there exists

- finite number of Borel pieces P^α ,
- finite number of functions $f^\alpha \in \text{Lip}(\mathbb{R}^a : \mathbb{R}^b) \cap C^1(\mathbb{R}^a : \mathbb{R}^b)$,
- such that

$$f(x) = \sum_{\alpha} \mathbf{1}_{P^\alpha}(x) f^\alpha(x) \quad \forall x \in \mathbb{R}^a.$$

Definition (Associated gradients)

We say that the gradient associated to the representation is

$$\tilde{\nabla} f(x) = \sum_{\alpha} \mathbf{1}_{P^{\alpha}}(x) \nabla f^{\alpha}(x) \in \mathcal{M}_{b,a}(\mathbb{R}) \quad \forall x \in \mathbb{R}^d.$$

In brief it is an associated gradient.

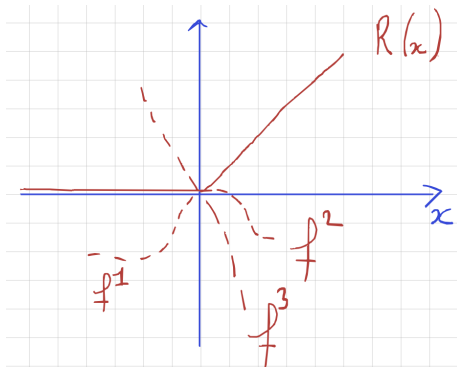
The associated gradient is defined everywhere but is non unique.

It is what is implemented in softwares : Tensorflow, Pytorch, Scikit-learn.

It is extremely close to the notion of intentional derivative computational sciences : **Lee-Yu-Rival-Yang, On Correctness of Automatic Differentiation for Non-Differentiable Functions, 2020).**

Remark

This is an **a priori** definition of a discrete gradient.



$P^1 =]0, \infty[$ and $f^1(x) = x$,
 $P^2 =]-\infty, 0[$ and $f^2(x) = 0$,
 $P^3 = \{0\}$ and f^3 is any C^1 function.

So $\tilde{R}'(x) = 1$ for $x > 0$, $\tilde{R}'(x) = 0$ for $x < 0$ and $\tilde{R}'(0) = \text{any } z \in \mathbb{R}$.

Murat-Trombetti Theorem ($u \in H^1$)

Theorem (D. 2025, with associated gradient notations)

Take $u \in \text{Lip}(\mathbb{R}^a : \mathbb{R}^b)$ Lipschitz-continuous.

Take $v \in \text{Lip}(\mathbb{R}^b : \mathbb{R}^c)$ Lipschitz-continuous and C_{pw}^1 (so with associated gradient).

Then the chain rule identity holds in $L^\infty(\mathbb{R}^a : \mathcal{M}_{c,a}(\mathbb{R}))$

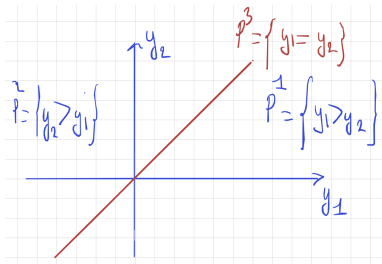
$$\nabla(v \circ u) = \widetilde{\nabla} v \circ u \nabla u$$

where $\widetilde{\nabla} v \circ u(x) = \widetilde{\nabla} v(u(x))$ for all $x \in \mathbb{R}^a$.

Therefore

$$\widetilde{\nabla} v = \nabla v \text{ in } L^\infty(\mathbb{R}^a : \mathcal{M}_{c,a}(\mathbb{R})).$$

The associated gradient is the gradient in the sense of distribution.

Direct proof for $f = S_1 \circ f_0$ 

- In P^1 : Clearly $f^1(y_1, y_2) = y_1$ in P^1 , so $\nabla f^1(y_1, y_2) = (1, 0)$.
- In P^2 : $\nabla f^2(y_1, y_2) = (0, 1)$.
- In P^3 : $y = \max(y, y) = S_1(y, y) = f^3(y, y)$ for all $y \in \mathbb{R}$. Set $\alpha(y) = \partial_{y_1} f^3(y, y)$ and $\beta(y) = \partial_{y_2} f^3(y, y)$.

One has $1 = \alpha(y) + \beta(y)$ for all y , whatever $f^3 \in C^1(\mathbb{R}^2)$.

One obtains the correct chain rule

$$1 = f'(x) = \langle \nabla f^3, W_0 \rangle = \langle (\alpha, \beta), (1, 1) \rangle = \alpha + \beta.$$

Consequence for autodiff

Take a Neural Network function

$$f = f_\ell \circ S_\ell \circ f_{\ell-1} \circ S_{\ell-1} \circ \cdots \circ f_1 \circ S_1 \circ f_0.$$

Assume **all** f_r and S_r are **Lipschitz-continuous** and C_{pw}^1 , with corresponding associated gradients.

Theorem (D. TMLR 2025)

Then f itself has an associated gradient

$$\tilde{\nabla} f = \tilde{A}_\ell(x) \tilde{B}_\ell(x) \tilde{A}_{\ell-1}(x) \tilde{B}_{\ell-1}(x) \cdots \tilde{A}_1(x) \tilde{B}_1(x) \tilde{A}_0(x) \quad \text{for all } x \in \mathbb{R}^a$$

where the right hand side is defined for all $x \in \mathbb{R}^a$, and

$$\tilde{A}_r(x) = \tilde{\nabla} f_r \circ S_r \circ \cdots \circ S_1 \circ f_0(x) \in \mathcal{M}_{a_{r+1}, a_r}(\mathbb{R}) \text{ for all } x \in \mathbb{R}^a,$$

$$\tilde{B}_r(x) = \tilde{\nabla} S_r \circ \cdots \circ S_1 \circ f_0(x) \in \mathcal{M}_{a_r}(\mathbb{R}) \text{ for all } x \in \mathbb{R}^a.$$

The space of functions $\text{Lip}(\mathbb{R}^a : \mathbb{R}^a) \cap C_{pw}^1(\mathbb{R}^a : \mathbb{R}^a)$ is an algebra.

If you want to construct a Banach space on the top of this construction, go to
On the limit set of Neural Network functions trainable by Machine Learning, D., HAL 2026.

Boustany paradox 2024 (also Bolte-Pauwels-...)

- For $x = (1, 2, 3, 4)$ encode the function $t \mapsto f(t) = \max_1(tx) - \max_2(tx)$.

```
def max1(x):
    res = x[0]
    for i in range(1, a):
        if x[i] > res:
            res = x[i]
    return res
```

```
def max2(x):
    return torch.max(x)
```

- Pytorch-autodiff is

t	-1	-0.5	-0.01	0	0.01	0.5	1
$f(t)$	0	0	0	0	0	0	0
$f'(t)$	0	0	0	-1.5	0	0	0

- Solution of the paradox :

- $\max_1$ is represented with 4 pieces and $\widetilde{\nabla} \max_1(0, 0, 0, 0) = (1, 0, 0, 0)$.
- $\max_2$ is represented with 15 pieces and $\widetilde{\nabla} \max_2(0, 0, 0, 0) = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$.
- So $\widetilde{f}'(0) = (1, 2, 3, 4) \cdot (1, 0, 0, 0) - (1, 2, 3, 4) \cdot (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}) = -1.5$.

DeepRitz for elliptic PDEs (with M. P.)

Weinan+Bing, The Deep Ritz method : A deep learning-based numerical algorithm for solving variational problems, 2017.

Solve elliptic problems like the model equation

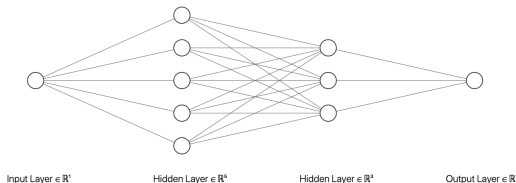
$$\begin{cases} -\Delta u + u = f, & x \in \Omega, \\ \partial_n u = 0, & x \in \partial\Omega, \end{cases}$$

using

$$u = \operatorname{argmin}_{v \in V} \int_{\Omega} \left(\frac{1}{2} |\nabla v(x)|^2 + \frac{1}{2} |v(x)|^2 - f(x)v(x) \right) dx, \quad V = H^1(\Omega).$$

-
- Lagaris+Likas+Fotiadis, Artificial Neural Networks for Solving Ordinary and PDEs, 1997.
 - Raissi+Perdikaris+Karniadakis. Physics-informed neural networks (PINNs), 2019.
 - Kharazmi+Zhang+Karniadakis, VPINNs : Variational Physics-Informed Neural Networks For Solving Partial Differential Equations, 2019,
 - Berrone+Canuto+Pintore. VPINNs quadratures and test functions, 2021.

- Step 1 : choose a NN architecture (hyper-parameters)



from which constructs functions $x \mapsto v_{\theta}^{\text{NN}}(x)$ where we renamed the weights $\theta = W$. It defines the space

$$V^{\text{NN}} = \{v_{\theta}^{\text{NN}} \text{ for all weights } \theta\} \subset C_{\text{pw}}^1.$$

- Step 2 : minimize $u^{\text{NN}} = \underset{v_{\theta}^{\text{NN}} \in V^{\text{NN}}}{\operatorname{argmin}} J(\theta)$

$$J(\theta) = \int_{\Omega} \left(\frac{1}{2} |\nabla v_{\theta}^{\text{NN}}(x)|^2 + \frac{1}{2} |v_{\theta}^{\text{NN}}(x)|^2 - f(x) v_{\theta}^{\text{NN}}(x) \right) dx.$$

- Step 3 : replace $\int h(x) dx$ with $\sum h(x_i) w_i$: **Riemann, Monte Carlo, ...**

A recent universal approximation Theorem

Theorem (Yarotsky 2017)

There exists a ReLU-NN approximating all bounded functions belonging to $W^{n,\infty}([0,1]^d)$ with uniform accuracy ε and at most $O(\varepsilon^{-d/n} \log 1/\varepsilon)$ computational units.

For ReLU networks, one has the embedding

$$\{P^1\text{-based FE spaces}\} \subset V^{\text{NN}}.$$

Approximation of the integral

- Implementation of $\nabla_x v_\theta^{\text{NN}}(x) \leftarrow \widetilde{\nabla}_x v_\theta^{\text{NN}}(x)$ is with autodiff

$$I = \int_{\Omega} \left(\frac{1}{2} |\widetilde{\nabla}_x v_\theta^{\text{NN}}(x)|^2 + \frac{1}{2} |v_\theta^{\text{NN}}(x)|^2 - f(x) v_\theta^{\text{NN}}(x) \right) dx.$$

- Whatever the number of layers,
autodiff is correct as a consequence of the Murat-Trombetti Theorem

$$\widetilde{\nabla} v_\theta^{\text{NN}} = \nabla v_\theta^{\text{NN}} \text{ in } L^\infty.$$

- In 1D, I do oversampling with $x_i = ih$ and $0 < \omega_i = h \ll 1$

$$I_h(\theta) = \sum_i \left(\frac{1}{2} |\widetilde{\nabla}_x v_\theta^{\text{NN}}(x_i)|^2 + \frac{1}{2} |v_\theta^{\text{NN}}(x_i)|^2 - f(x_i) v_\theta^{\text{NN}}(x_i) \right) \omega_i \xrightarrow{h \rightarrow 0} I$$

- Then we do the training (Machine Learning part of the business), also with autodiff

$$\theta' = -\widetilde{\nabla}_\theta I_h(\theta).$$

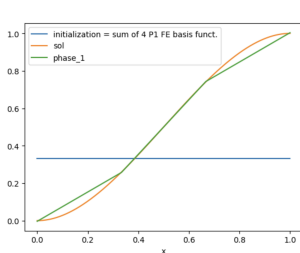
A numerical test (Tensorflow)

$$\begin{cases} -u'' + u = f, & x \in I = (0, 1), \\ u' = 0, & x \in \partial I = \{0, 1\}. \end{cases}$$

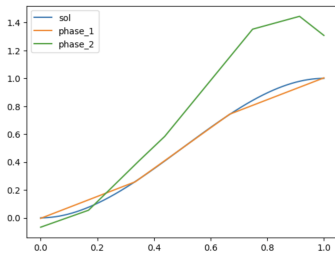
$u(x) = -1/2 + 3x^2 - 2x^3$: 2 hidden layers of 4 neurons with $R=\text{ReLU}$.

Initialization : P^1 basis functions are pre-encoded with hidden layers.

Optimization (gradient descent) performed with ADAM.



Erreur L2 (phase 1)= 0.051075217503822296



Erreur L2 (phase 1, 2)= 0.051075217503822296 0.49719538224957444

Phase I : Optimization of last layer. Phase II : Optimization of all layers.

What is the reason ?

- Hint : for phase II, the loss increases. What the heck ?

```
Epoch 1/4
1/1 ----- 0s 129ms/step - loss: -0.7354 - val_loss: -0.4493
Epoch 2/4
1/1 ----- 0s 13ms/step - loss: -0.4493 - val_loss: -0.2669
Epoch 3/4
1/1 ----- 0s 15ms/step - loss: -0.2669 - val_loss: -0.2751
Epoch 4/4
1/1 ----- 0s 14ms/step - loss: -0.2751 - val_loss: -0.3163
1/1 ----- 0s 7ms/step - loss: -0.3163
Loss= -0.3163410909306603
```

- Explanation : take $u^{\text{NN}}(x) = R(wx + b)$ (that is $\theta = (w, b)$)

$$\frac{\partial}{\partial b} \left| \frac{\partial R(wx + b)}{\partial x} \right|^2 = \frac{\partial}{\partial b} |wH(wx + b)|^2 = 2w^2 H(wx + b) \delta(wx + b).$$

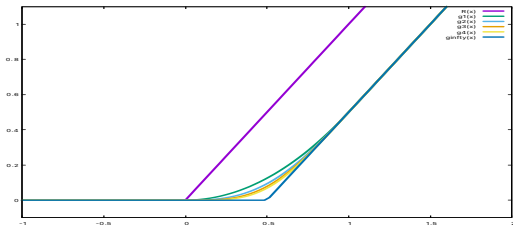
- On computational grounds, $\frac{\partial}{\partial b} \left| \frac{\partial R(wx+b)}{\partial x} \right|^2 = 0$.
- On mathematical grounds, this is **a serious crime**.
- The problem comes from the interaction PDE+training embedded in the differential operators

$$\nabla_{\theta} \nabla_x \text{ or } \tilde{\nabla}_{\theta} \tilde{\nabla}_x.$$

- It raised a mathematical question :
what is the minimal regularity of an activation function for such cases ?

- Define $R_{\text{spl}}^n \in \mathbf{S}_{2n,n} \subset C^n(\mathbb{R})$ such that $\frac{d^n}{dx^n} R_{\text{spl}} \in \text{Lip}(\mathbb{R}) \cap C_{\text{pw}}^1(\mathbb{R})$

$$R_{\text{spl}}^1(x) = \begin{cases} 0 & x \in P^1 = \{x < 0\}, \\ x^2/2 & x \in P^2 = \{0 \leq x \leq 1\}, \\ x - 1/2 & x \in P^3 = \{1 \leq x\}. \end{cases}$$

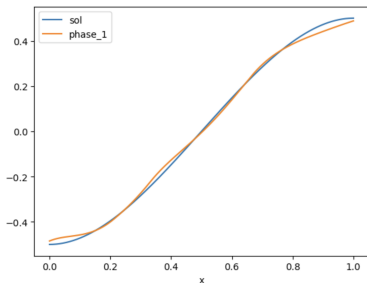


ReLU-splines are like high order Finite Elements.
Error estimates are immediately inherited.

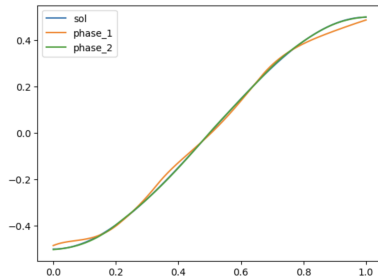
Same numerical test

$$\begin{cases} -u'' + u = f, & x \in I = (0, 1), \\ u' = 0, & x \in \partial I = \{0, 1\}. \end{cases}$$

$u(x) = -1/2 + 3x^2 - 2x^3$, 2 hidden layers of 4 neurons, R_{spl}^1 .



Erreur L2 (phase 1) = 0.04571944033722722



C'est fini

Erreur L2 (phase 1, 2) = 0.04571944033722722 0.004541409064901931

Phase I : Optimization of last layer.

Phase II : Optimization of all layers.

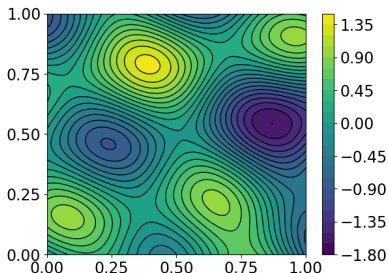
$$2D : \Omega = [0, 1]^2$$

Test problem

$$\begin{cases} -\Delta u + u = f & \text{in } \Omega, \\ \partial_n u = g & \text{on } \partial\Omega. \end{cases}$$

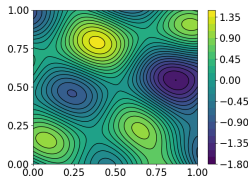
The manufactured exact solution is

$$u(x, y) = \sin(3x(x - y))\cos(4y + x) + \sin(4(x + 2y))\cos(3(y - 2x)).$$

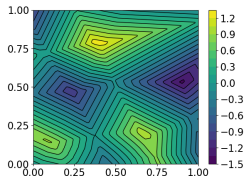


The NN is 5 layers, 20 neurons per layer (probably too much).
Initialization is he-normal initialization popular for ReLU.

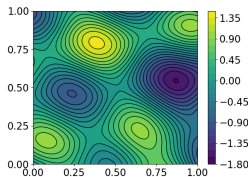
$$J(\theta) = \int_{\Omega} \left(\frac{1}{2} |\nabla u|^2 + \frac{1}{2} |u|^2 - fu \right) dx - \int_{\Gamma} gud\sigma.$$



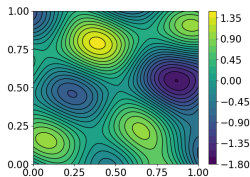
exact



ReLU

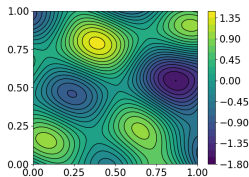


ReLU-Spline O1

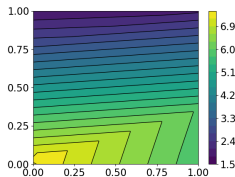


ReLU-Spline O2

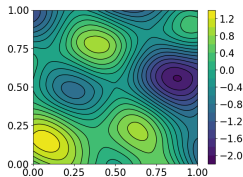
$$J(\theta) = \int_{\Omega} |-\Delta u + u - f|^2 dx + \lambda \int_{\partial\Omega} |\partial_n u - g|^2 d\sigma.$$



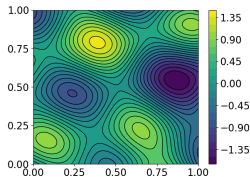
exact



ReLU



ReLU-Spline O1



ReLU-Spline O2

A real engineering problem

Do+Ammar+Gerard Castaing+Madiot (CEA/SERMA) :
Physics Informed Neural Networks for the Mixed Dual Form of the Neutron
Diffusion Equation with Heterogeneous Coefficients, 2025

One desires to solve

$$-\nabla \cdot (D(x) \nabla u)(x)) + u(x) = f(x) \quad + \text{ B.C.}$$

For engineering problems in nuclear reactor simulations, the matrix D is discontinuous (a representative Dauge test case is below).

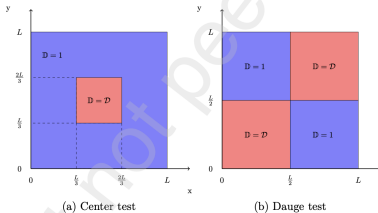


Figure 2: Geometry of the test cases with contrast diffusion coefficient.

Low regularity of the solution.

- A mathematical framework for non smooth autodiff and FEM.

A functional framework for nonsmooth autodiff and application to Neural Networks, D., TMLR 2025.

On the limit set of Neural Network functions trainable by Machine Learning, D., HAL 2026.

- Let n =number of derivatives of your problem

Rule of the thumb : Take $R \in \mathbf{S}_{2n,n}(\mathbb{R})$ such that

$$R \in C^n(\mathbb{R}) \text{ and } \frac{d^n}{dx^n} R \in \text{Lip}(\mathbb{R}) \cap C_{\text{pw}}^1(\mathbb{R}).$$

Plenty of open problems remain.

High-order ReLU-splines for NN approx. of elliptic equations, D.+Moreno Pintore, in prep, 2026.

- See you soon perhaps at **SciML in Paris 19/20/21 May 2027**

