Combining supervised deep learning and scientific computing: some contributions and application to computational fluid dynamics.

29/03/2022

Paul Novello, *CESTA CEA-DAM, INRIA, Ecole Polytechnique*

Ecole Doctorale de Mathématiques Hadamard

Gaël Poëtte, *CESTA CEA-DAM*

David Lugato, *CESTA CEA-DAM*

Pietro Marco Congedo, *INRIA, Ecole Polytechnique*

Context: atmospheric reentry

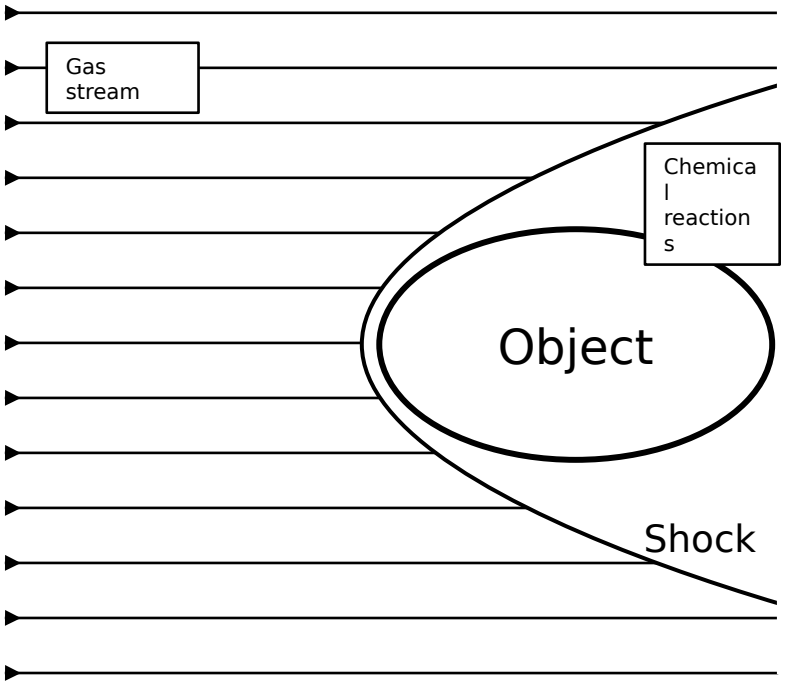
Accurately simulating atmospheric reentry is crucial

Either to ensure that objects, like satellites, are correctly destroyed …

… Or to allow space shuttle passengers to return home safely.

Motivating example: atmospheric reentry with chemical reactions

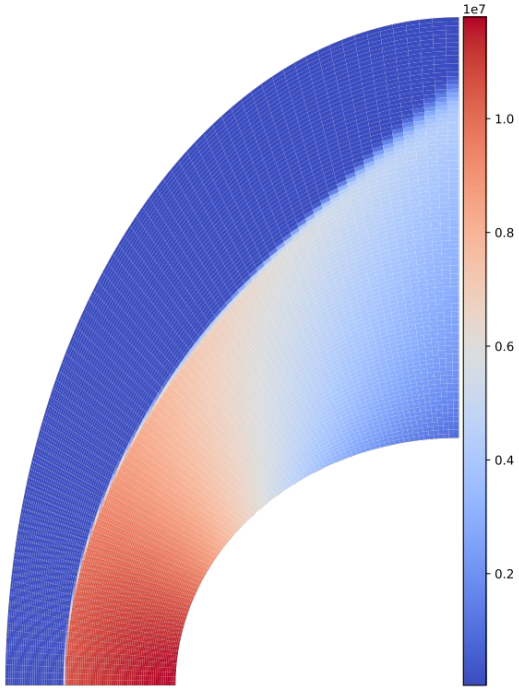
Multi physics, **multi scale** simulations, can be very expensive because of the coupling of different physics



Example: atmospheric reentry coupled with chemical reactions

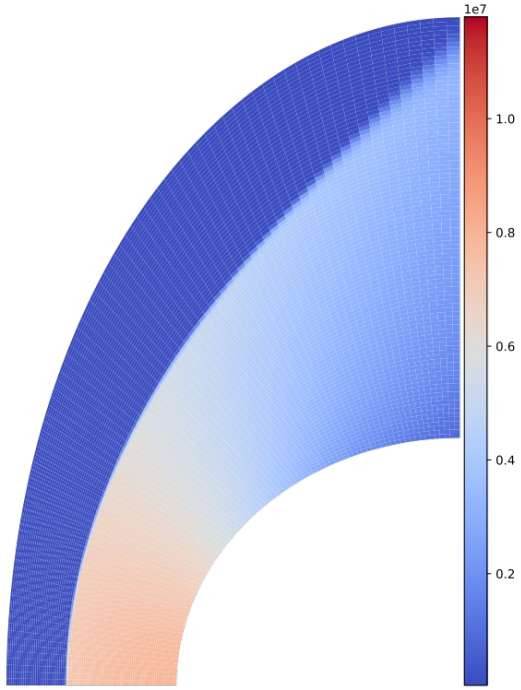
Motivating example: Effect of chemical model on a Toy problem

time **81 s**



Without chemical reactions (Perfect gas)

time **4090 s**



With chemical reactions

Pressure field*

Trade-off between model **accuracy** and **cost-efficiency:**

- We want the code cost effective enough to conduct parametric studies
- But accurate enough to simulate correct physics

* Obtained with a CEA simulation code developed by Simon Peluchon [23]

Goal: Acceleration of the code using Machine Learning

Scientific machine learning:

"Machine learning for advanced scientific computing research" *

Neural nets. as surrogate models for hybridization

Acc.

- More and more **successful,** cf. recent techniques and architectures. [4, 7, 19, 9, 20, 21,...]
- Have theoretical **approximation guarantees -** Hornik et al. 1989, Barron 1994, Zhou et al. 2017 [8, 1, 11].

Perf.

- At inference, the complexity of neural networks is:
  - **Independent** from the size of the training database.
  - **Linear** w.r.t input/output dimension.
- Their implementation boils down to a succession of **matrix-vector** products.

* Baker et al. 2019 [22]

Neural Networks: definition

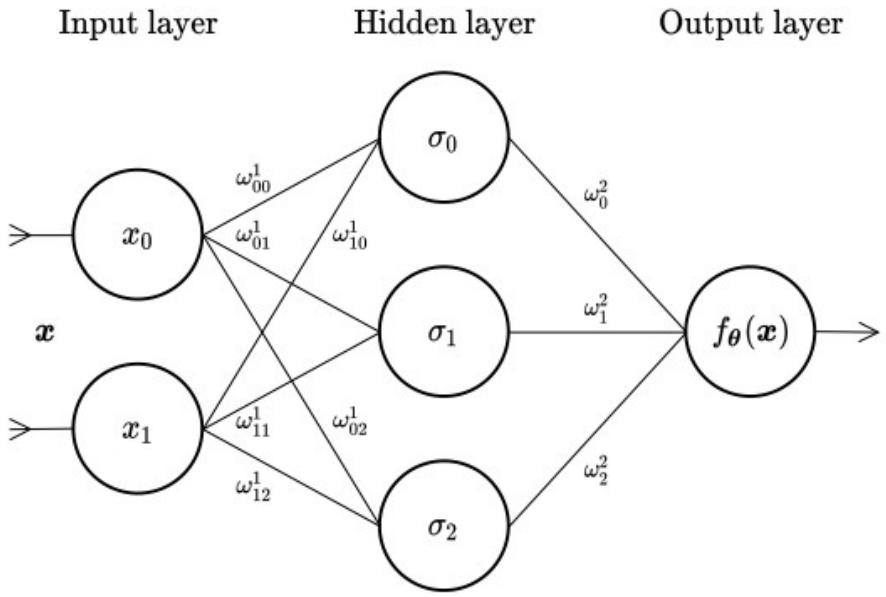
Definition of a fully connected neural network $f_\theta : \mathbb{R}^{n_i} \to \mathbb{R}^{n_o}$

- d the depth of the network
- n_k the number of neurons in the network k-th layer, with and $n_{d+1} = n_o$, $n_0 = n_i$
- σ_k the activation function of the k-th layer (usually non linear)
- $\mathbf{W}^k = \{\omega_{ij}^k\}, (i,j) \in \{0, n_{k+1} - 1\} \times \{0, n_k - 1\}$ the weights matrix between the (k-1)-th and k-th layer
- $\mathbf{b}^k$ the bias vector of the k-th layer

Let $f^k : \mathbb{R}^{n_k} \to \mathbb{R}^{n_{k+1}}$ such that $f_i^k(\mathbf{x}) = \sigma_k(\mathbf{W}^k\mathbf{x} + \mathbf{b}^k)$

$$f_\theta(\mathbf{x}) = f^d \circ \cdots \circ f^1(\mathbf{x})$$

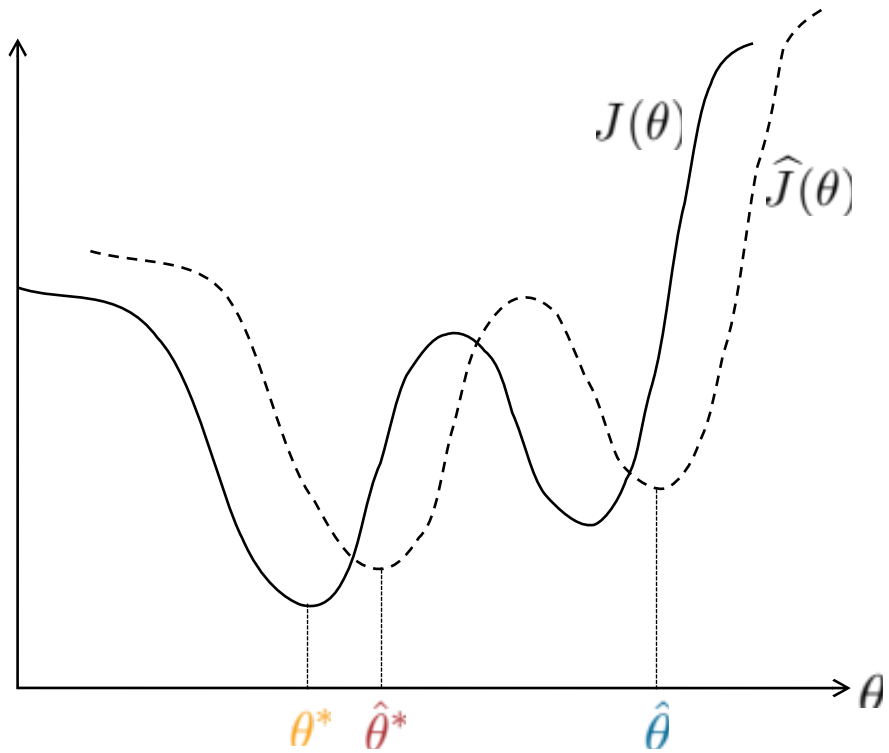
Neural Networks: definition



MLP of depth $d = 1$, with $n_{in} = n_0 = 2$, $n_1 = 3$ and $n_{out} = n_2 = 1$. In this specific case, for readability, we denote by σ_i the i-th component of the output of the hidden layer.

$$\begin{cases} f_\theta(x) & = \omega^2_0\sigma(\omega^1_{00}x_0 + \omega^1_{10}x_1 + b^1_0) + \omega^2_1\sigma(\omega^1_{01}x_0 + \omega^1_{11}x_1 + b^1_1) + \omega^2_2\sigma(\omega^1_{02}x_0 + \omega^1_{12}x_1 + b^1_2) + b^2 \\ & = \sum_{i=0}^{2} \omega^2_i\sigma(\sum_{j=0}^{1} \omega^1_{ji}x_j + b_i) + b^2 \\ \theta & = \{\omega^2_0, \omega^1_{00}, \omega^1_{10}, \omega^2_1, \omega^1_{01}, \omega^1_{11}, \omega^2_2, \omega^1_{02}, \omega^1_{12}, b^1_0, b^1_1, b^1_2, b^2\}. \end{cases}$$

The machine learning task



$$J(\theta) = \int_{\boldsymbol{S}} L(f_\theta(x), f(x)) d\mathbb{P}_x$$

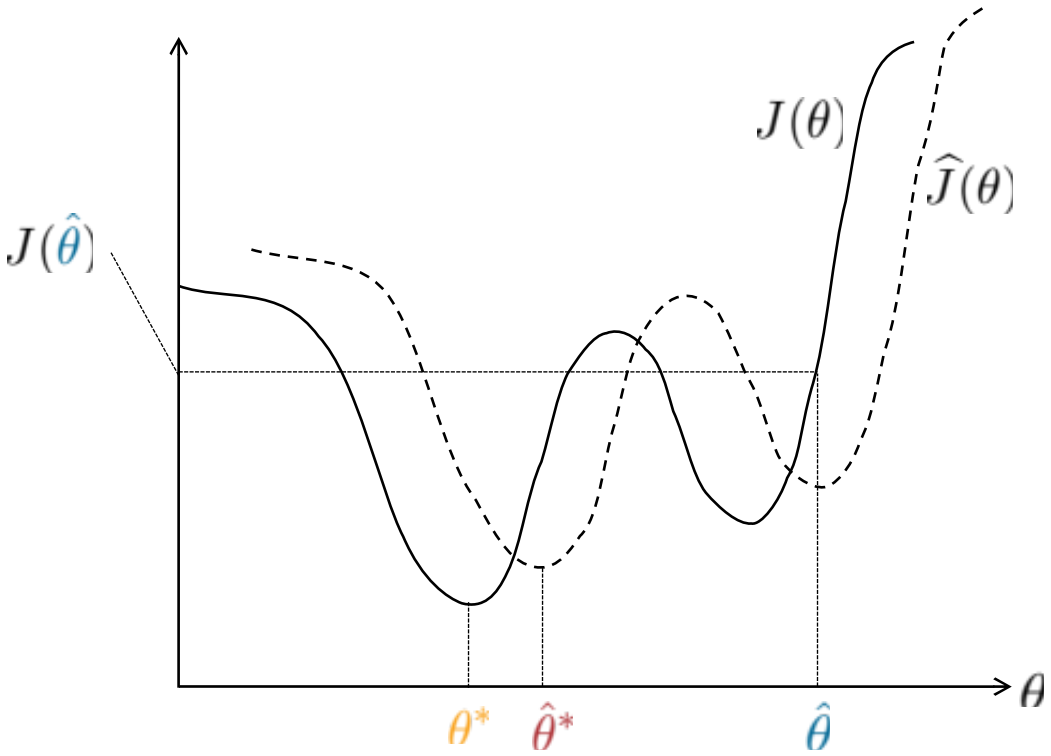
$$\widehat{J}(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(x_i) - f(x_i))$$

θ^*: global min. of $J(\theta)$

$\hat{\theta}^*$: global min. of $\widehat{J}(\theta)$

$\hat{\theta}$: a local min. of $\widehat{J}(\theta)$

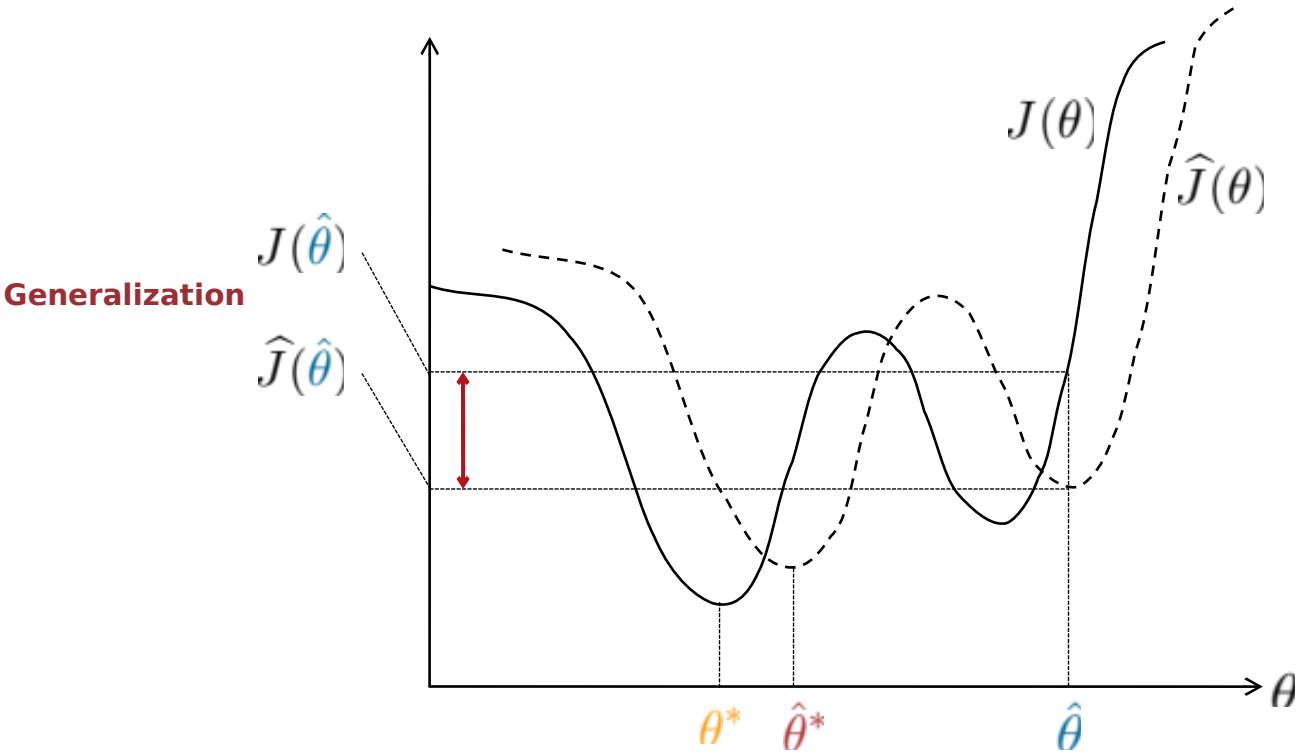
Stakes of Machine Learning: Error decomposition



The final error of f_θ is

$$J(\hat{\theta})$$

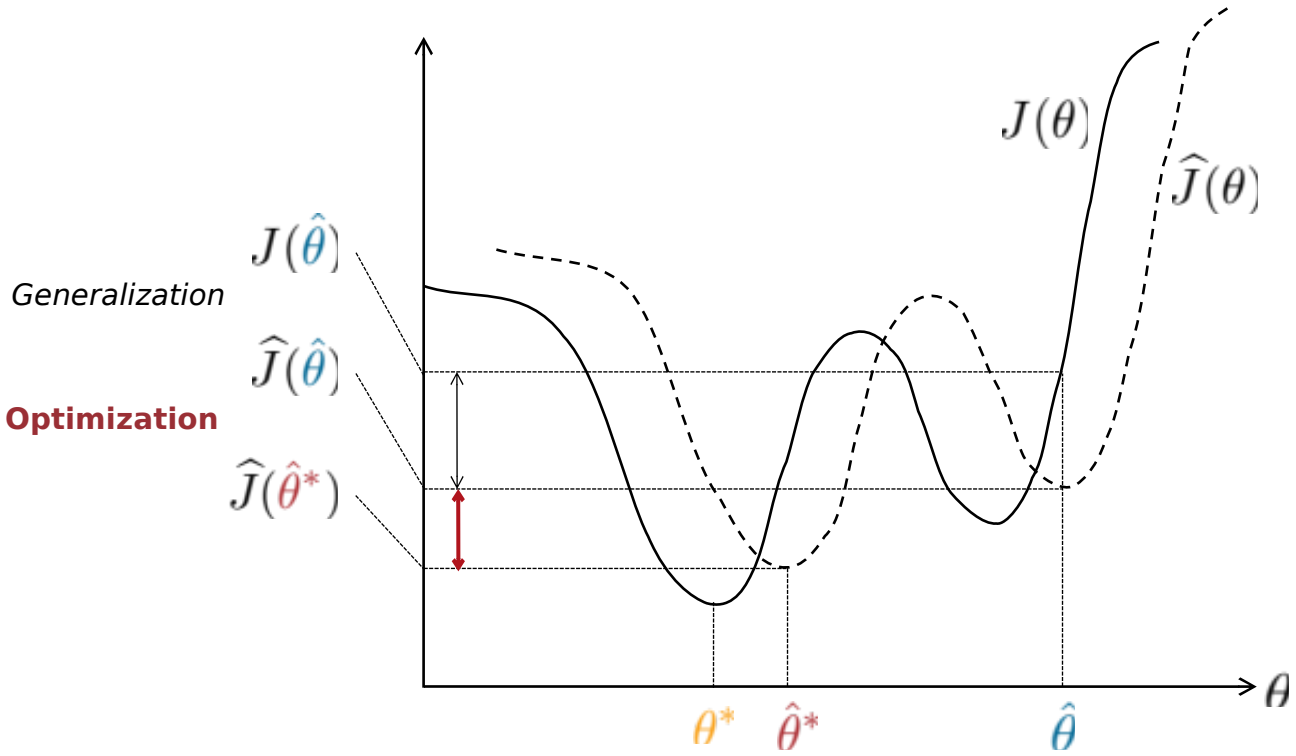
Stakes of Machine Learning: Error decomposition



The final error of f_θ is

$$J(\hat{\theta}) - \widehat{J}(\hat{\theta}) + \underbrace{J(\hat{\theta}) - \widehat{J}(\hat{\theta})}_{\text{Generalization}}$$

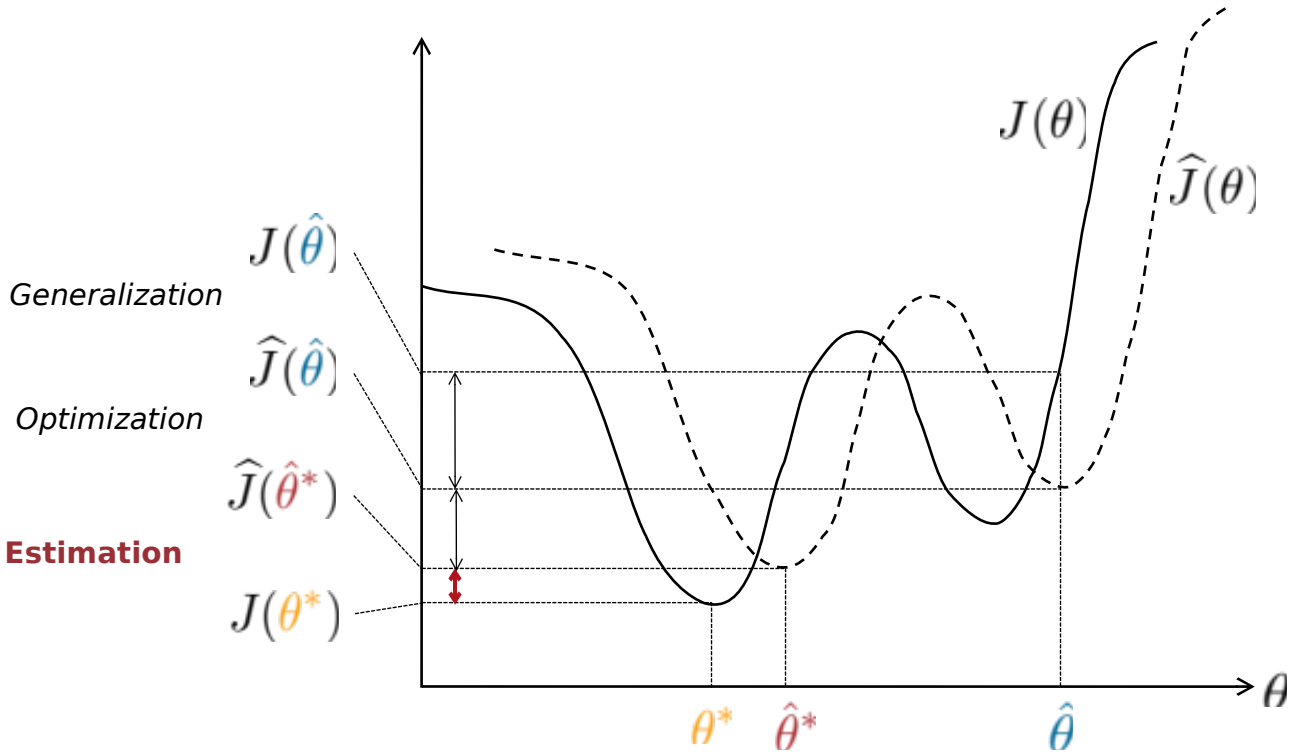
Stakes of Machine Learning: Error decomposition



The final error of f_θ is

$$J(\hat{\theta}) = \widehat{J}(\hat{\theta}^*) + \underbrace{\widehat{J}(\hat{\theta}) - \widehat{J}(\hat{\theta}^*)}_{\textbf{Optimization}} + \underbrace{J(\hat{\theta}) - \widehat{J}(\hat{\theta})}_{\textit{Generalization}}$$

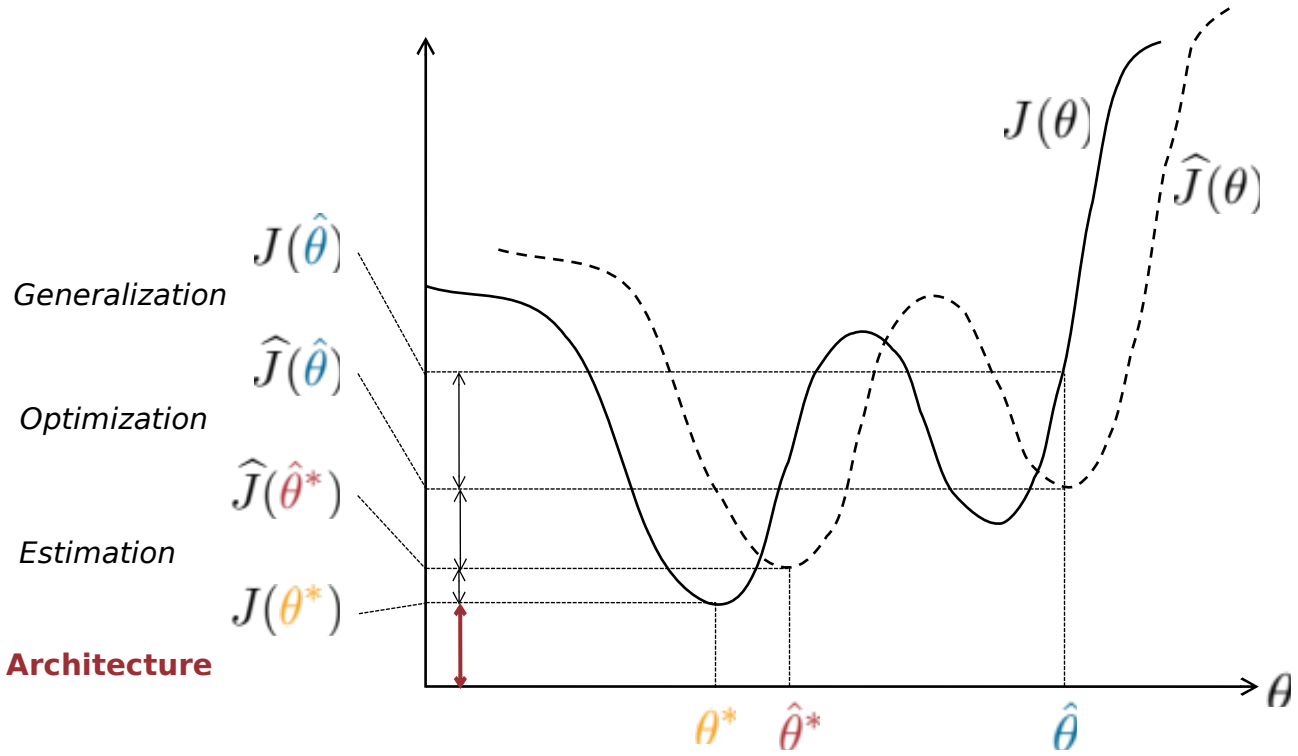
Stakes of Machine Learning: Error decomposition



The final error of f_θ is

$$J(\hat{\theta}) = J(\theta^*) + \underbrace{\widehat{J}(\hat{\theta}^*) - J(\theta^*)}_{\textbf{Estimation}} + \underbrace{\widehat{J}(\hat{\theta}) - \widehat{J}(\hat{\theta}^*)}_{\textit{Optimization}} + \underbrace{J(\hat{\theta}) - \widehat{J}(\hat{\theta})}_{\textit{Generalization}}$$

Stakes of Machine Learning: Error decomposition



The final error of f_θ is

$$J(\hat{\theta}) = \underbrace{J(\theta^*)}_{\text{Architecture}} + \underbrace{\widehat{J}(\hat{\theta}^*) - J(\theta^*)}_{\text{Estimation}} + \underbrace{\widehat{J}(\hat{\theta}) - \widehat{J}(\hat{\theta}^*)}_{\text{Optimization}} + \underbrace{J(\hat{\theta}) - \widehat{J}(\hat{\theta})}_{\text{Generalization}} \quad [3]$$

The Machine Learning methodology

$$J(\hat{\theta}) = \underbrace{\widehat{J}(\hat{\theta}^*) - J(\theta^*)}_{Estimation} + \underbrace{J(\theta^*)}_{Architecture} + \underbrace{\widehat{J}(\hat{\theta}) - \widehat{J}(\hat{\theta}^*)}_{Optimization} + \underbrace{J(\hat{\theta}) - \widehat{J}(\hat{\theta})}_{Generalization}$$

1. Estimation

Construction of the training set

2. Architecture

Hyperparameter optimization

3. Optimization

Training of the neural network

4. Generalization

Integration into a simulation code

Outline of the thesis

1. Estimation

Construction of the training set

- Taylor based Sampling (TBS).
- Variance based Sample Weighting (VBSW).

Leveraging Local Variation in Data: sampling and weighting schemes for supervised deep learning,
Paul Novello, Gael Poette, David Lugato and Pietro Congedo,
Accepted to the Journal of Machine Learning for Modelling and Computing

2. Architecture

Hyperparameter optimization

Goal-Oriented Sensitivity Analysis of Hyperparameters in Deep Learning,
Paul Novello, Gael Poette, David Lugato and Pietro Congedo,
Submitted to the Journal of Scientific Computing

3. Optimization

Training of the neural network

- Construction of a PDE framework for training neural networks.
- PDE based Stochastic Gradient Descent (PDESGD).

An analogy between solving Partial Differential Equations with Monte-Carlo schemes and the Optimisation process in Machine Learning (and few illustrations of its benefits),
Gael Poette, Paul Novello, David Lugato,
Working report.

4. Generalization

Integration into a simulation code

Accelerating hypersonic reentry simulations using deep learning-based hybridization (with guarantees),
Paul Novello, Gael Poette, David Lugato and Pietro Congedo.
To be sent to the Journal of Computational Physics

Outline of the thesis

1. Estimation

Construction of the training set

- Taylor based Sampling (TBS).
- Variance based Sample Weighting (VBSW).

Leveraging Local Variation in Data: sampling and weighting schemes for supervised deep learning,
Paul Novello, Gael Poette, David Lugato and Pietro Congedo,
Accepted to the Journal of Machine Learning for Modelling and Computing

2. Architecture

Hyperparameter optimization

Goal-Oriented Sensitivity Analysis of Hyperparameters in Deep Learning,
Paul Novello, Gael Poette, David Lugato and Pietro Congedo,
Submitted to the Journal of Scientific Computing

3. Optimization

Training of the neural network

- Construction of a PDE framework for training neural networks.
- PDE based Stochastic Gradient Descent (PDESGD).

An analogy between solving Partial Differential Equations with Monte-Carlo schemes and the Optimisation process in Machine Learning (and few illustrations of its benefits),
Gael Poette, Paul Novello, David Lugato,
Working report.

4. Generalization

Integration into a simulation code

Accelerating hypersonic reentry simulations using deep learning-based hybridization (with guarantees),
Paul Novello, Gael Poette, David Lugato and Pietro Congedo.
To be sent to the Journal of Computational Physics

Outline of the thesis

1. Estimation

Construction of the training set

- Taylor based Sampling (TBS).
- Variance based Sample Weighting (VBSW).

Leveraging Local Variation in Data: sampling and weighting schemes for supervised deep learning,
Paul Novello, Gael Poette, David Lugato and Pietro Congedo,
Accepted to the Journal of Machine Learning for Modelling and Computing

2. Architecture

Hyperparameter optimization

Goal-Oriented Sensitivity Analysis of Hyperparameters in Deep Learning,
Paul Novello, Gael Poette, David Lugato and Pietro Congedo,
Submitted to the Journal of Scientific Computing

3. Optimization

Training of the neural network

- Construction of a PDE framework for training neural networks.
- PDE based Stochastic Gradient Descent (PDESGD).

An analogy between solving Partial Differential Equations with Monte-Carlo schemes and the Optimisation process in Machine Learning (and few illustrations of its benefits),
Gael Poette, Paul Novello, David Lugato,
Working report.

4. Generalization

Integration into a simulation code

Accelerating hypersonic reentry simulations using deep learning-based hybridization (with guarantees),
Paul Novello, Gael Poette, David Lugato and Pietro Congedo.
To be sent to the Journal of Computational Physics

The learning task: Second order optimization

$$\nabla_\theta J(\theta) = \int L'(x,\theta)\nabla_\theta f(x,\theta) d\mathbb{P}_x$$

Notations:

$$f_\theta(x) = f(x,\theta)$$
$$L(x,\theta) = L(f(x), f(x,\theta))$$
$$L' : x, y \to \nabla_y L(x,y)$$

Newton algorithm:

We aim at cancelling the gradient with the iterative algorithm defined by

$$\theta^{k+1} = \theta^k - \nabla_\theta^2 J(\theta^k)^{-1} \nabla_\theta J(\theta^k)$$

Advantages:

- Optimal step sizes
- Leverages 2nd order

Drawbacks:

- Computationally expensive
- Goes upward when J is non convex

The learning task: First order optimization

$$\nabla_\theta J(\theta) = \int L'(x,\theta)\nabla_\theta f(x,\theta) d\mathbb{P}_x$$

Notations:

$$f_\theta(x) = f(x,\theta)$$
$$L(x,\theta) = L(f(x), f(x,\theta))$$
$$L' : x, y \to \nabla_y L(x,y)$$

Gradient descent (GD):

We aim at cancelling the gradient with the iterative algorithm defined by

$$\theta^{k+1} = \theta^k - \gamma \nabla_\theta J(\theta^k) \quad \text{where} \quad \gamma > 0$$

Advantages:

- Computationally efficient
- Ensures that the optimization does not go upward when J is non convex

Drawbacks:

- The learning rate may be sub optimal
- Only first order

PDE formulation of Newton algorithm

- The update formula of Newton algorithm can be reformulated as

$$\nabla_\theta J(\theta) + \nabla^2_{\hat{n}} J(\theta)(\theta^* - \theta) = 0$$

where θ is the vector of neural network parameters and θ^* is the updated value that we are looking for. Then, by using the definition of *J*:

$$0 = \int L'(x,\theta)\nabla_\theta f(x,\theta) d\mathbb{P}_x$$
$$+ \int \left[L'(x,\theta)\nabla^2_\theta f(x,\theta) + L''(x,\theta)\nabla_\theta f(x,\theta)\nabla^T_\theta f(x,\theta) \right](\theta^* - \theta) d\mathbb{P}_x.$$

- This equation can be seen as a PDE of unknowns θ^* and $\theta \to f(x,\theta)$ where $f(x,\theta)$ is the neural network.

Goal: solving the PDE by using Kolmogorov's backward equation and simulating an Ito process

Kolmogorov's backward equation:

Let f^0 be in $\mathcal{C}^2$ and let X^t be an Ito process with initial condition $X^0 = \mathbf{s} \in \mathbb{R}^n$. Kolmogorov's theorem states that $f_{\mathbf{s}}(t, \mathbf{s})$ defined as $f_{\mathbf{s}}(t, \mathbf{s}) = \mathbb{E}_{X^t}[f^0(\mathbf{x})]$ is solution of

$$\begin{cases} \partial_t f_s(t,s) + \sum_{i=1}^{n} \partial_{s_i} f_s(t,s)\mu_i(t,s) + \frac{1}{2}\sum_{i,j=1}^{n} \partial^2_{s_i,s_j} f_s(t,s)[\Sigma(t,s)\Sigma^T(t,s)]_{i,j} = 0, \\ f_s(0,s) = f^0(s). \end{cases} \quad (1)$$

Constructing the PDE framework

$$0 = \partial_t f_s(t,s) + \sum_{i=1}^{n} \partial_{s_i} f_s(t,s)\mu_i(t,s) + \frac{1}{2}\sum_{i,j=1}^{n} \partial^2_{s_i,s_j} f_s(t,s)[\Sigma(t,s)\Sigma^T(t,s)]_{i,j}$$

- **Link between the two PDEs:**
  - $\mathbf{s}$ echoes θ
  - $f_{\mathbf{s}}$ echoes f
  - n echoes $Card(\theta$
- **What we have to do to close the gap:**
  - make the PDE scalar
  - make the PDE unstationary
  - deal with the dependancy in x
  - add a closure equation

$$0 = \int L'(x,\theta)\nabla_\theta f(x,\theta) d\mathbb{P}_x$$
$$+ \int \left[L'(x,\theta)\nabla^2_\theta f(x,\theta) + L''(x,\theta)\nabla_\theta f(x,\theta)\nabla^T_\theta f(x,\theta)\right](\theta^* - \theta) d\mathbb{P}_x$$

Making the PDE scalar

Let us introduce an arbitrary vector $\alpha \in \mathbb{R}^n$. The PDE is equivalent tc

$$0 = \int L'(x,\theta)\alpha^T \nabla_\theta f(x,\theta) d\mathbb{P}_x$$
$$+ \int \alpha^T \Big[L'(x,\theta)\nabla^2_\theta f(x,\theta) + L''(x,\theta)\nabla_\theta f(x,\theta)\nabla^T_\theta f(x,\theta) \Big](\theta^* - \theta) d\mathbb{P}_x$$
$$\forall \alpha \in \mathbb{R}^n.$$

And to make it look even more like (1):

$$0 = \sum_{i=1}^n \int \partial_{\theta_i} f_k(x,\theta^t)\alpha_i \left[L'(x,\theta^t) + L''(x,\theta^t) \sum_{j=1}^n \partial_{\theta_j} f_k(x,\theta^t)(\theta^*_j - \theta^t_j) \right] d\mathbb{P}_x$$
$$+ \frac{1}{2} \sum_{i,j=1}^n \int \partial^2_{\theta_i,\theta_j} f_k(x,\theta^t) 2\alpha_i L'(x,\theta^t)(\theta^*_j - \theta^t_j) d\mathbb{P}_x,$$
$$\forall \alpha \in \mathbb{R}^n, k \in \{1,\dots,p\}.$$

Making the PDE scalar

$$0 = \sum_{i=1}^{n} \int \partial_{\theta_i} f_k(x, \theta^t) \alpha_i \left[L'(x, \theta^t) + L''(x, \theta^t) \sum_{j=1}^{n} \partial_{\theta_j} f_k(x, \theta^t)(\theta_j^s - \theta_j^t) \right] d\mathbb{P}_x$$

$$+ \frac{1}{2} \sum_{i,j=1}^{n} \int \partial^2_{\theta_i, \theta_j} f_k(x, \theta^t) 2\alpha_i L'(x, \theta^t)(\theta_j^s - \theta_j^t) d\mathbb{P}_x,$$

$$\forall \alpha \in \mathbb{R}^n, k \in \{1, ..., p\}.$$

$$0 = \partial_t f_s(t,s) + \sum_{i=1}^{n} \partial_{s_i} f_s(t,s) \mu_i(t,s) + \frac{1}{2} \sum_{i,j=1}^{n} \partial^2_{s_i, s_j} f_s(t,s) [\Sigma(t,s)\Sigma^T(t,s)]_{i,j} \quad (1$$

Making the PDE scalar

$$0 = \sum_{i=1}^{n} \int \partial_{\theta_i} f_k(x,\theta^t)\alpha_i \left[L'(x,\theta^t) + L''(x,\theta^t) \sum_{j=1}^{n} \partial_{\theta_j} f_k(x,\theta^t)(\theta_j^s - \theta_j^t) \right] d\mathbb{P}_x$$

$$+ \frac{1}{2} \sum_{i,j=1}^{n} \int \partial^2_{\theta_i,\theta_j} f_k(x,\theta^t) 2\alpha_i L'(x,\theta^t)(\theta_j^s - \theta_j^t) d\mathbb{P}_x,$$

$$\forall \alpha \in \mathbb{R}^n, k \in \{1,....,p\}.$$

$$0 = \partial_t f_s(t,s) + \sum_{i=1}^{n} \partial_{s_i} f_s(t,s)\mu_i(t,s) + \frac{1}{2} \sum_{i,j=1}^{n} \partial^2_{s_i,s_j} f_s(t,s)[\Sigma(t,s)\Sigma^T(t,s)]_{i,j} \quad (1$$

Making the PDE unstationnary

We can add a dependence w.r.t. time. The PDE becomes:

$$
\begin{aligned}
0 = & \partial_t \int f(x,\theta^t) d\mathbb{P}_x \\
& + \sum_{i=1}^{n} \int \partial_{\theta_i} f(x,\theta^t) \alpha_i \left[L'(x,\theta^t) + L''(x,\theta^t) \sum_{j=1}^{n} \partial_{\theta_j} f(x,\theta^t)(\theta_j^* - \theta_j^t) \right] d\mathbb{P}_x \\
& + \frac{1}{2} \sum_{i,j=1}^{n} \int \partial^2_{\theta_i,\theta_j} f(x,\theta^t) 2\alpha_i L'(x,\theta^t)(\theta_j^* - \theta_j^t) d\mathbb{P}_x,
\end{aligned}
$$

$\forall \alpha \in \mathbb{R}^n, k \in \{1, ..., p\}.$

and has to be solved $\forall t \in [0,T]$, $T < \infty$ with initial condition $f^0(\theta) = f(x,\theta^0)$

Making the PDE unstationnary

$$0 = \partial_t \int f(x,\theta^t) d\mathbb{P}_x$$
$$+ \sum_{i=1}^{n} \int \partial_{\theta_i} f(x,\theta^t) \alpha_i \left[L'(x,\theta^t) + L''(x,\theta^t) \sum_{j=1}^{n} \partial_{\theta_j} f(x,\theta^t)(\theta_j^* - \theta_j^t) \right] d\mathbb{P}_x$$
$$+ \frac{1}{2} \sum_{i,j=1}^{n} \int \partial^2_{\theta_i,\theta_j} f(x,\theta^t) 2\alpha_i L'(x,\theta^t)(\theta_j^* - \theta_j^t) d\mathbb{P}_x,$$
$$\forall \alpha \in \mathbb{R}^n, k \in \{1, ..., p\}.$$

$$0 = \partial_t f_s(t,s) + \sum_{i=1}^{n} \partial_{s_i} f_s(t,s)\mu_i(t,s) + \frac{1}{2} \sum_{i,j=1}^{n} \partial^2_{s_i,s_j} f_s(t,s)[\Sigma(t,s)\Sigma^T(t,s)]_{i,j} \quad (1)$$

Closure and dependency on x by assumptions on fluctuations

Let us decompose $f(x, \theta^t)$ into

$$f(x, \theta^t) = \bar{f}(\theta^t) + \hat{f}(x, \theta^t),$$

where

$$\bar{f}(\theta^t) = \int f(x, \theta) d\mathbb{P}_x \quad \text{and} \quad \hat{f}(x, \theta^t) = f(x, \theta^t) - \bar{f}(\theta^t)$$

Assumption: $\hat{f} \equiv 0$

Is this assumption justified ? Cf **[*]**

[*] An analogy between solving Partial Differential Equations with Monte-Carlo schemes and the Optimisation process in Machine Learning (and few illustrations of its benefits),
Gael Poette, Paul Novello, David Lugato, Working report.

Closure and dependency on x by assumptions on fluctuations

Assumption: $\hat{f} \equiv 0$

$$
\begin{cases}
0 = \partial_t \bar{f}(\theta^t) \\
\qquad + \sum_{i=1}^{n} \partial_{\theta_i} \bar{f}(\theta^t) \int \alpha_i \left[L'(x, \theta^t) + L''(x, \theta^t) \sum_{j=1}^{n} \partial_{\theta_j} f(x, \theta^t)(\theta_j^* - \theta_j^t) \right] d\mathbb{P}_x \\
\qquad + \frac{1}{2} \sum_{i,j=1}^{n} \partial^2_{\theta_i, \theta_j} \bar{f}(\theta^t) 2\alpha_i \int L'(x, \theta^t)(\theta_j^* - \theta_j^t) d\mathbb{P}_x, \\
f(x, \theta^0) = f^0(\theta), \quad \theta \in \Theta, \quad t \in [0, T], \quad x \sim d\mathbb{P}_x.
\end{cases}
$$

Closure and dependency on x by assumptions on fluctuations

$$
\begin{cases}
0 = \partial_t \bar{f}(\theta^t) \\
\quad + \sum_{i=1}^{n} \partial_{\theta_i} \bar{f}(\theta^t) \int \alpha_i \left[L'(x,\theta^t) + L''(x,\theta^t) \sum_{j=1}^{n} \partial_{\theta_j} f(x,\theta^t)(\theta_j^* - \theta_j^t) \right] d\mathbb{P}_x \\
\quad + \frac{1}{2} \sum_{i,j=1}^{n} \partial^2_{\theta_i,\theta_j} \bar{f}(\theta^t) 2\alpha_i \int L'(x,\theta^t)(\theta_j^* - \theta_j^t) d\mathbb{P}_x, \\
f(x,\theta^0) = f^0(\theta), \quad \theta \in \Theta, \quad t \in [0,T], \quad x \sim d\mathbb{P}_x.
\end{cases}
$$

$$
\begin{cases}
0 = \partial_t f_s(t,s) + \sum_{i=1}^{n} \partial_{s_i} f_s(t,s) \mu_i(t,s) + \frac{1}{2} \sum_{i,j=1}^{n} \partial^2_{s_i,s_j} f_s(t,s) [\Sigma(t,s)\Sigma^T(t,s)]_{i,j}, \\
f_s(0,s) = f^0(s).
\end{cases}
\tag{1}
$$

PDE framework

- Let us consider
  - The drift term: $\mu(\theta) = \int \alpha\Big(L'(x_i,\theta) + L''(x_i,\theta)\nabla_\theta^T f(x,\theta)(\theta^* - \theta)\Big)d\mathbb{P}_x$,
  - The diffusion coefficients: $\left[\Sigma\Sigma^T\right]_{i,j}(\theta) = 2\int \alpha_i L'(x,\theta)(\theta_j^* - \theta_j)d\mathbb{P}_x$.

If those terms are defined, we can solve the PDE using a discretized Ito process :

$$\theta^{k+1} = \theta^k + \frac{\Delta t}{N}\sum_{i=1}^{N}\left[\alpha L'(x_i,\theta^k) + L''(x_i,\theta^k)\alpha\nabla_\theta^T f(x_i,\theta^k)(\theta^* - \theta^k)\right] + \frac{\sqrt{\Delta t}}{N}\left[\sum_{i=1}^{N}\Sigma(x_i,\theta^k,\alpha,\theta^* - \theta^k)\right]g,$$

with $f(x,\theta)$ the neural network, L the loss function, g a multivariate normal gaussian, $\{x_1,...x_N\}$ the training points, and α, $(\theta^* - \theta_k)$, Σ some parameters to choose when using the PDE framework.

Methodology:

- Choose α, $\theta^* - \theta^k$ and Σ
- Simulate θ^k using

$$\theta^{k+1} = \theta^k + \frac{\Delta t}{N}\sum_{i=1}^{N}\left[\alpha L'(x_i,\theta^k) + L''(x_i,\theta^k)\alpha\nabla_\theta^T f(x_i,\theta^k)(\theta^* - \theta^k)\right] + \frac{\sqrt{\Delta t}}{N}\left[\sum_{i=1}^{N}\Sigma(x_i,\theta^k,\alpha,\theta^* - \theta^k)\right]g,$$

- ... meanwhile monitoring $\hat{J}(\theta^k)$

PDESGD: Improving SGD using the PDE framework

$$\theta^{k+1} = \theta^k - \frac{\Delta t}{N}\sum_{i=1}^{N}\Big[L'(x_i,\theta^k)\nabla_\theta f(x_i,\theta^k) + \frac{\lambda}{N}\nabla_\theta f(x_i,\theta^k)\nabla_\theta^T f(x_i,\theta^k)\sum_{j=1}^{N} L'(x_j,\theta^k)\nabla_\theta f(x_j,\theta^k)\Big] + \sqrt{\frac{\Delta t}{N_b}}\Lambda(\theta^k)g.$$

with **step sizes** Λt defined as:

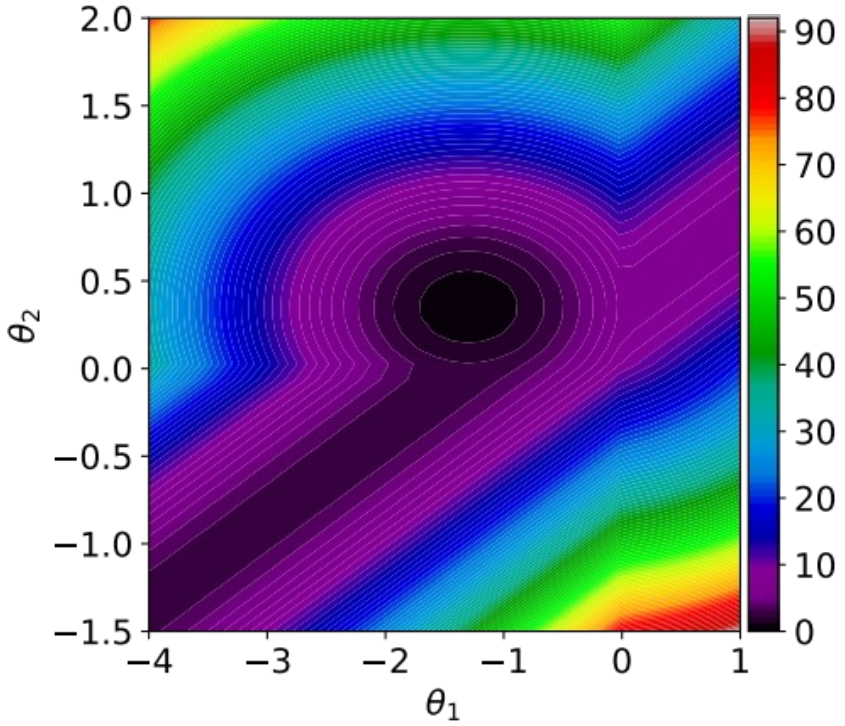
$$\Delta t = \Delta^k t = \min\left(\frac{\epsilon}{\max_{i\in\{1,\dots,n\}}\mu_i(t^k,\theta^k)}, \frac{\epsilon^2}{\max_{i\in\{1,\dots,n\}}[\Sigma(t^k,\theta^k)g]_i^2}\right)$$

to ensure **stability** and **accuracy** in $\mathcal{O}(\epsilon)$

A simple test case

Consider the neural network training problem where:

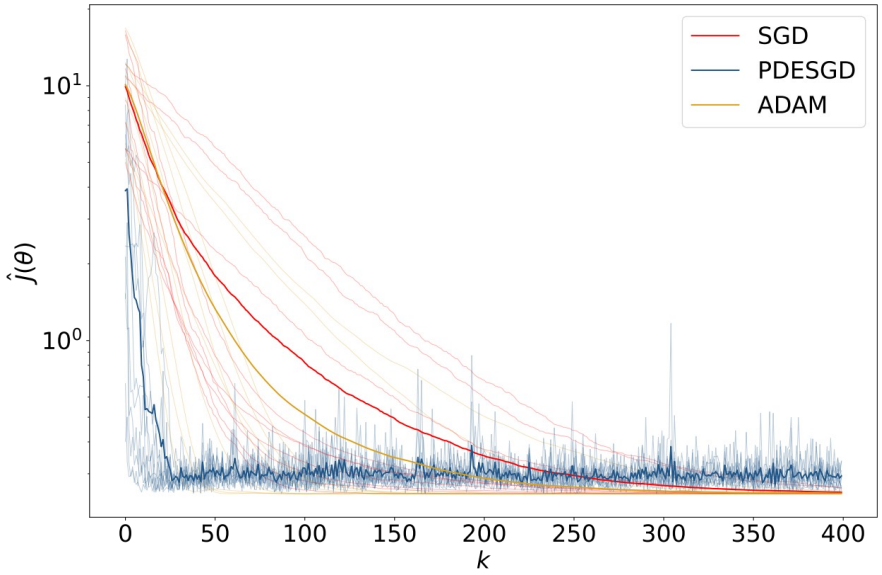
- Depth: 1
- Width: 2
- Activatioon function: ReLU
- Loss function: MSE
- No bias parameters
- Residual connection between input and output
- The function to approximate is $x \mapsto 1$



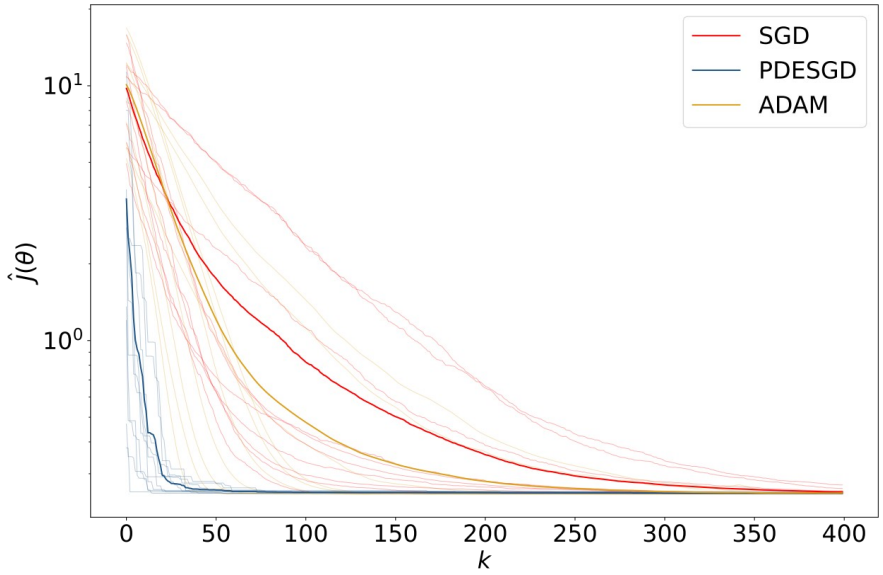
In that case, the training consists in optimizing:

$$\widehat{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\sigma(\theta_1 x_i) - 2\sigma(\theta_2 x_i) + x_i - 1\right)^2$$

Convergence speed in the convex region (empirical)



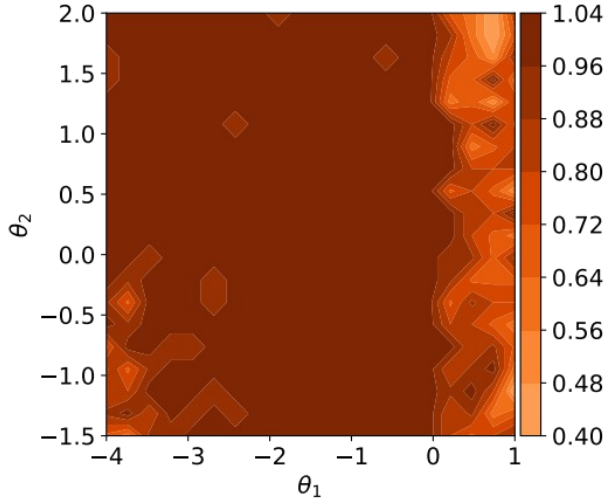
$$k \to \hat{J}(\theta^k)$$



$$k \to \min_{i<k} \hat{J}(\theta^i)$$

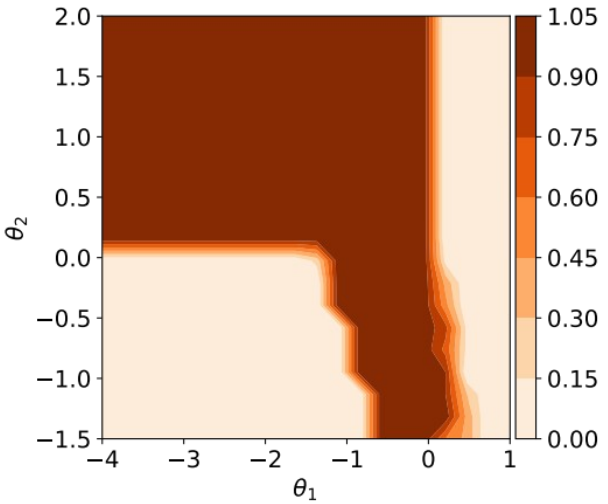
Parameter space stable exploration using Δ^k

PDESGD



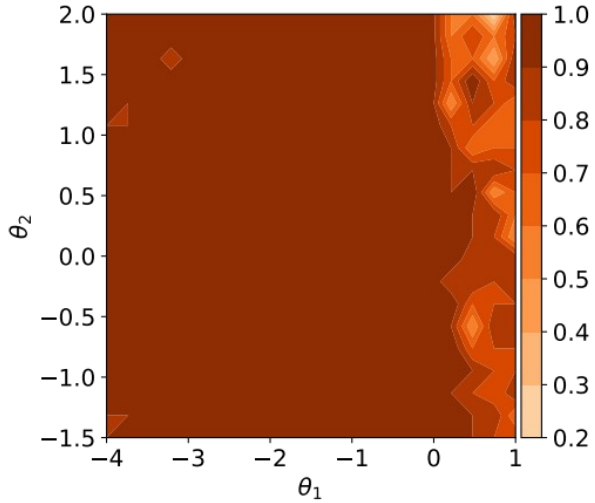
SGD

$\gamma = 0.01$



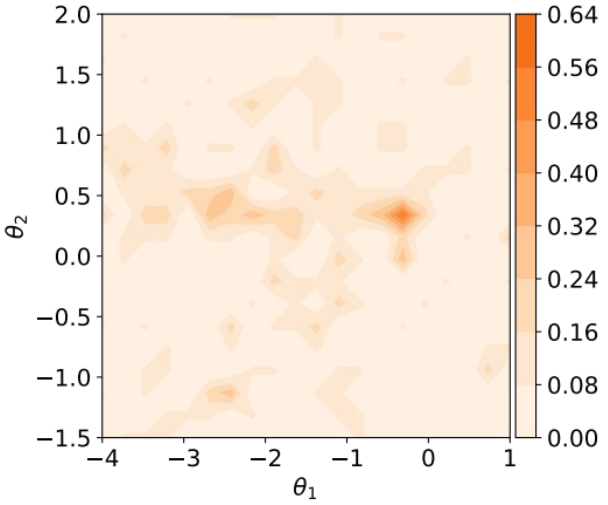
SGD

$\gamma = 0.05$



SGD

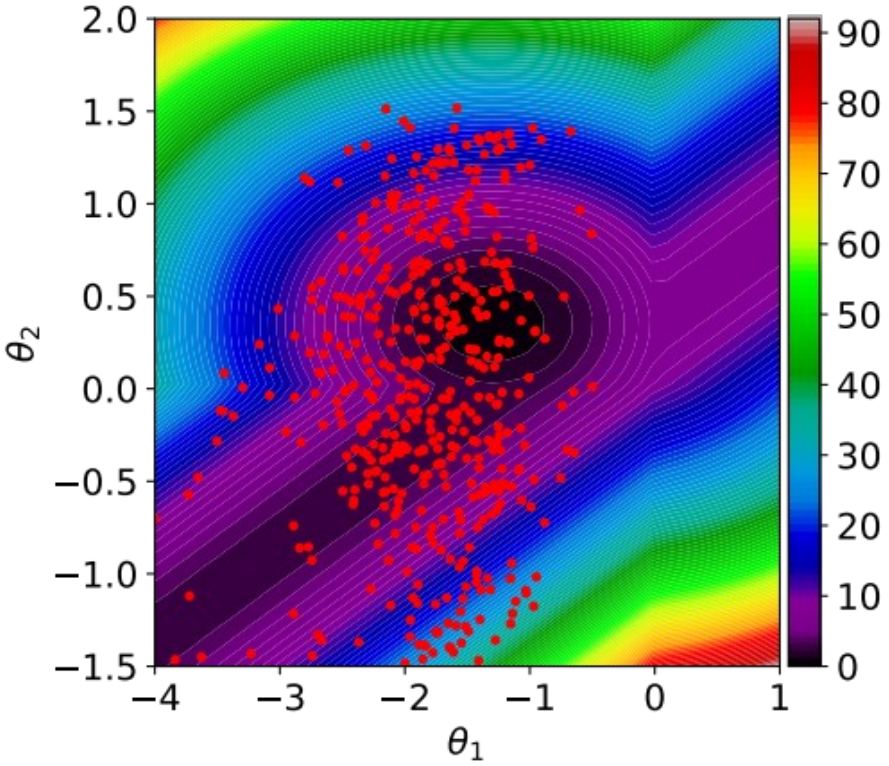
$\gamma = 0.1$



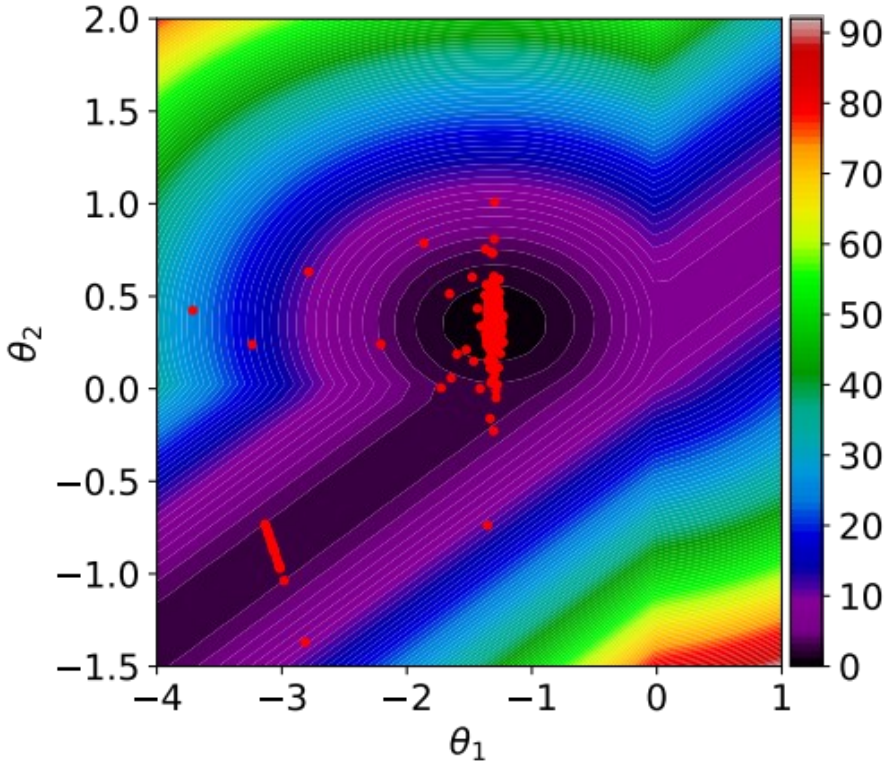
Heat maps of the probability to reach the region of the global minimum from coordinates of the map for PDESGD and SGD with different γ learning rates

Parameter space stable exploration using Δ^k

PDESGD

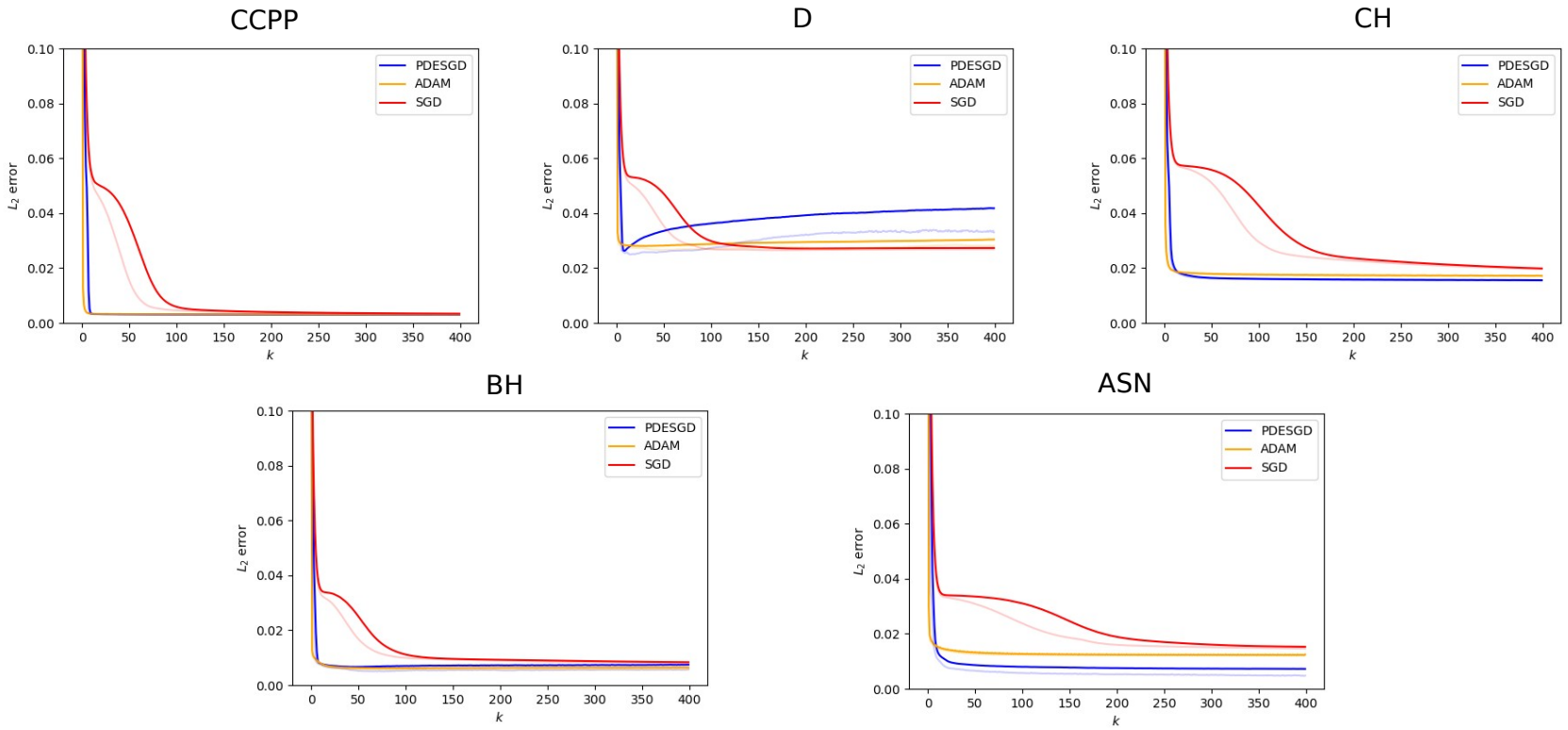


SGD



Example of points visited during the optimization when initialized outside of the convex region

Experiments on UCI data sets



Evolution of validation error on 5 UCI datasets

$\times 10^{-3}$	CH	D	BH	CCPP	ASN
SGD	14.73, 15.29	25.83, 26.99	5.38, 6.77	3.04, 3.06	5.39, 7.51
ADAM	15.02, 16.08	26.59, 28.02	5.58, **6.13**	3.05, 3.07	9.96, 12.01
PDESGD	**14.03**, **14.56**	**24.96**, **26.28**	**4.99**, 6.63	**2.85**, **2.94**	**3.25**, **6.36**

Outline of the thesis

1. Estimation

Construction of the training set

- Taylor based Sampling (TBS).
- Variance based Sample Weighting (VBSW).

Leveraging Local Variation in Data: sampling and weighting schemes for supervised deep learning,
Paul Novello, Gael Poette, David Lugato and Pietro Congedo,
Accepted to the Journal of Machine Learning for Modelling and Computing

2. Architecture

Hyperparameter optimization

Goal-Oriented Sensitivity Analysis of Hyperparameters in Deep Learning,
Paul Novello, Gael Poette, David Lugato and Pietro Congedo,
Submitted to the Journal of Scientific Computing

3. Optimization

Training of the neural network

- Construction of a PDE framework for training neural networks.
- PDE based Stochastic Gradient Descent (PDESGD).

An analogy between solving Partial Differential Equations with Monte-Carlo schemes and the Optimisation process in Machine Learning (and few illustrations of its benefits),
Gael Poette, Paul Novello, David Lugato,
Working report.

4. Generalization

Integration into a simulation code

Accelerating hypersonic reentry simulations using deep learning-based hybridization (with guarantees),
Paul Novello, Gael Poette, David Lugato and Pietro Congedo.
To be sent to the Journal of Computational Physics

Outline of the thesis

1. Estimation

Construction of the training set

- Taylor based Sampling (TBS).
- Variance based Sample Weighting (VBSW).

Leveraging Local Variation in Data: sampling and weighting schemes for supervised deep learning,
Paul Novello, Gael Poette, David Lugato and Pietro Congedo,
Accepted to the Journal of Machine Learning for Modelling and Computing

2. Architecture

Hyperparameter optimization

Goal-Oriented Sensitivity Analysis of Hyperparameters in Deep Learning,
Paul Novello, Gael Poette, David Lugato and Pietro Congedo,
Submitted to the Journal of Scientific Computing

3. Optimization

Training of the neural network

- Construction of a PDE framework for training neural networks.
- PDE based Stochastic Gradient Descent (PDESGD).

An analogy between solving Partial Differential Equations with Monte-Carlo schemes and the Optimisation process in Machine Learning (and few illustrations of its benefits),
Gael Poette, Paul Novello, David Lugato,
Working report.

4. Generalization

Integration into a simulation code

Accelerating hypersonic reentry simulations using deep learning-based hybridization (with guarantees),
Paul Novello, Gael Poette, David Lugato and Pietro Congedo.
To be sent to the Journal of Computational Physics

Motivating example: Approximating the solution of chemical reactions sol

Surrogate model to speed-up simulations

U: Fluid variables

$\mathbf{x}$: Chemical reactions variables

initialize_guess_vector_of_unknowns_on_mesh(U^0,$\mathbf{x}^0$)
while *convergence_criterion_not_satisfied* **do**
  $U^{n+\frac{1}{2}}$ =solve_CFD(U^n,$\mathbf{x}^n$)
  for *each cell* $i \in$ *mesh* **do**
    $(U_i^{n+1}, \mathbf{x}_i^{n+1})$ =solve_chemical_reactions($U_i^{n+\frac{1}{2}}$,$\mathbf{x}_i^n$)
  end
end

Mutation++ - Scoggins et al. 2020 [17]

Sketch of the code

Motivating example: Approximating the solution of chemical reactions sol

Surrogate model to speed-up simulations

U: Fluid variables

$\mathbf{x}$: Chemical reactions variables

initialize_guess_vector_of_unknowns_on_mesh(U^0,$\mathbf{x}^0$)
while *convergence_criterion_not_satisfied* **do**
  $U^{n+\frac{1}{2}}$ =solve_CFD(U^n,$\mathbf{x}^n$)
  for *each cell* $i \in$ *mesh* **do**
    (U_i^{n+1}, $\mathbf{x}_i^{n+1}$) =solve_chemical_reactions($U_i^{n+\frac{1}{2}}$,$\mathbf{x}_i^n$)
  end
end

Mutation++ [17]

Original code

initialize_guess_vector_of_unknowns_on_mesh(U^0,$\mathbf{x}^0$)
while *convergence_criterion_not_satisfied* **do**
  $U^{n+\frac{1}{2}}$ =solve_CFD(U^n,$\mathbf{x}^n$)
  (U_i^{n+1}, $\mathbf{x}_i^{n+1}$) =solve_chemical_reactions_NN($U_i^{n+\frac{1}{2}}$,$\mathbf{x}_i^n$)
end

Neural Network

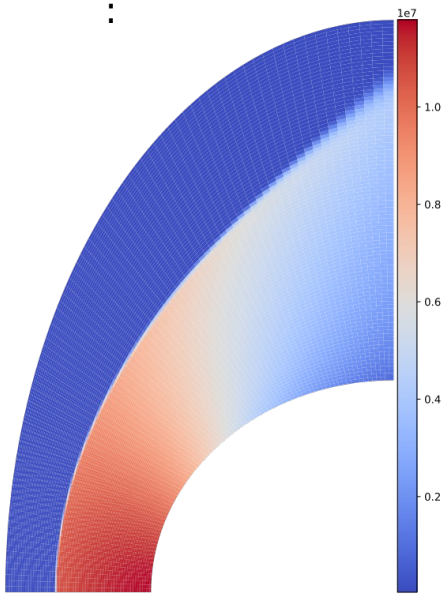
Hybrid code

Advantages

- **Construction of a large training database**
  - Mutation++ is cheap when called as a standalone function
- **Vectorization of the chemical reactions**
  - The neural net can process the whole mesh at once whereas Mutation++ includes a Newton i.e. is can not be vectorized easily

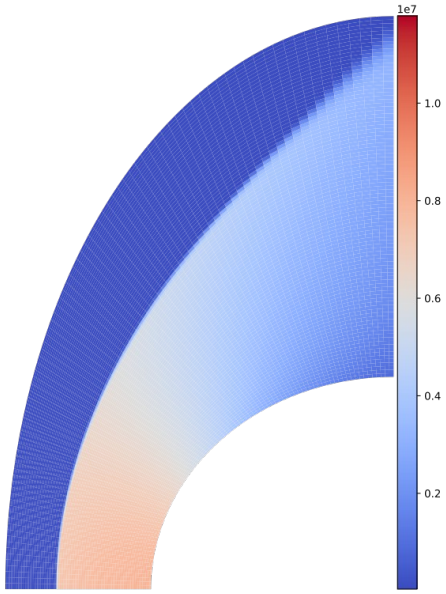
Neural networks based hybrid code: some results [*]

time : **81 s**
error : 7.74×10^{-2}



PG (Perfect Gas)

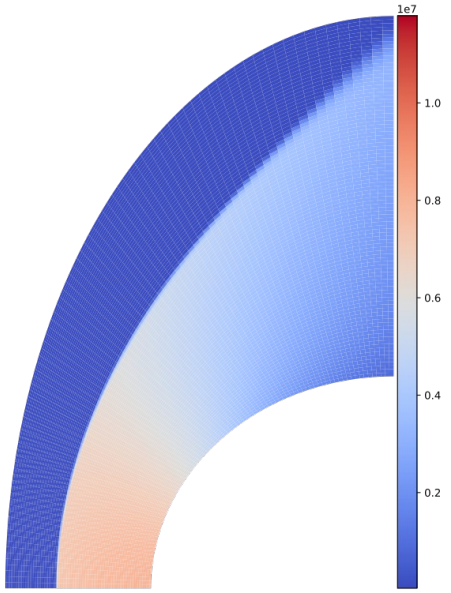
time : **531 s**
error : 3.06×10^{-7}



NN (Neural Network)

Pressure field

time : **4090 s**
error : **ref**



MPP (Mutation ++)

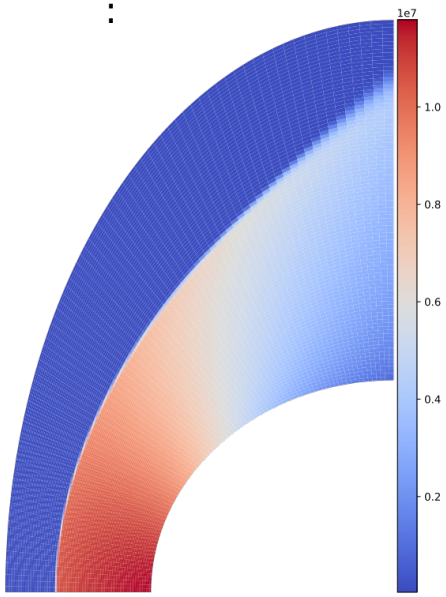
×8

[*] Accelerating hypersonic reentry simulations using deep learning-based hybridization (with guarantees),
Paul Novello, Gael Poette, David Lugato and Pietro Congedo.
In preparation. To be sent to the Journal of Computational Physics

Neural networks based hybrid code: some results

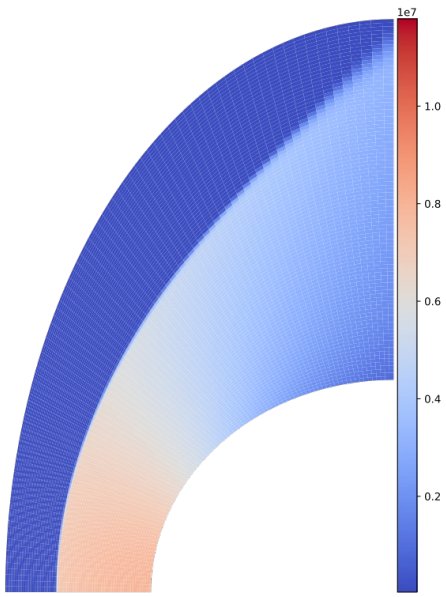
With BO + HSIC ! $\times 18.7$

time : **81 s**
error : 7.74×10^{-2}



PG (Perfect Gas)

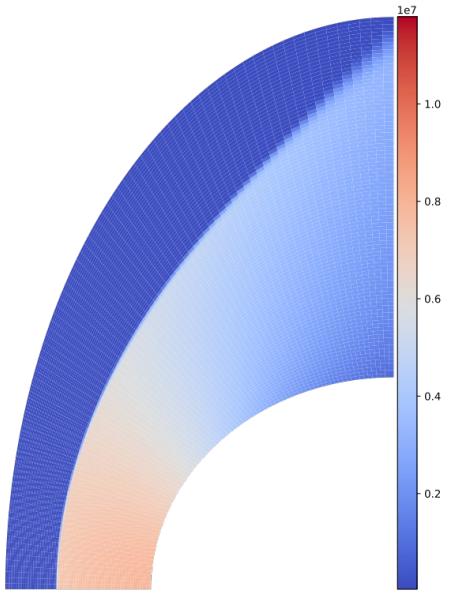
time : **220 s**
error : 3.06×10^{-7}



NN (Neural Network)

Pressure field

time : **4050 s**
error : **ref**



MPP (Mutation ++)

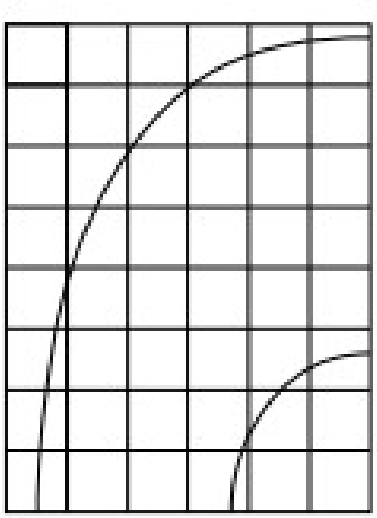
Is this error **acceptable ?** Can we obtain **guarantees ?**

Zero error guarantee: initialization of MPP with NN

initialize_guess_vector_of_unknowns_on_mesh(U^0,$\mathbf{x}^0$)
while *convergence_criterion_not_satisfied* **do**
  $U^{n+\frac{1}{2}}$ =solve_CFD(U^n,$\mathbf{x}^n$)
  for *each cell* $i \in$ *mesh* **do**
    (U_i^{n+1}, $\mathbf{x}_i^{n+1}$) =solve_chemical_equilibrium($U_i^{n+\frac{1}{2}}$,$\mathbf{x}_i^n$)

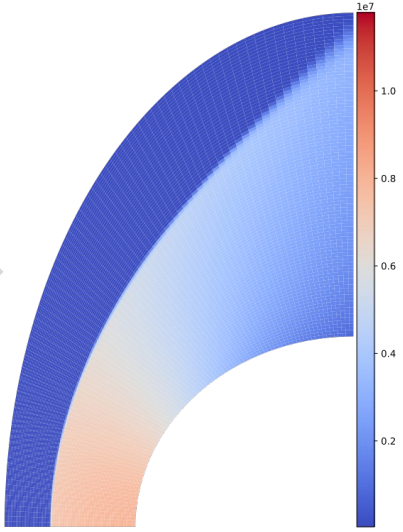
Mutation++

MPP



Uninformative initial guess

conv. in t_1



Prediction

Zero error guarantee: initialization of MPP with NN

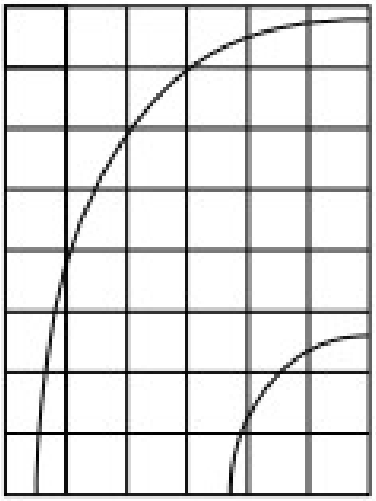
initialize_guess_vector_of_unknowns_on_mesh(U^0,$\mathbf{x}^0$)
while *convergence_criterion_not_satisfied* **do**
  $U^{n+\frac{1}{2}}$ =solve_CFD(U^n,$\mathbf{x}^n$)
  (U_i^{n+1}, $\mathbf{x}_i^{n+1}$) =solve_chemical_reactions_NN($U_i^{n+\frac{1}{2}}$,$\mathbf{x}_i^n$)

Neural net

initialize_guess_vector_of_unknowns_on_mesh($U^{n_{t_2}}$,$\mathbf{x}^{n_{t_2}}$)
while *convergence_criterion_not_satisfied* **do**
  $U^{n+\frac{1}{2}}$ =solve_CFD(U^n,$\mathbf{x}^n$)
  for *each cell* $i \in$ *mesh* **do**
    (U_i^{n+1}, $\mathbf{x}_i^{n+1}$) =solve_chemical_reactions($U_i^{n+\frac{1}{2}}$,$\mathbf{x}_i^n$)

Mutation++

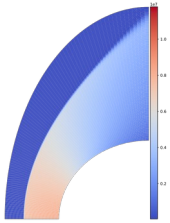
NN+MPP



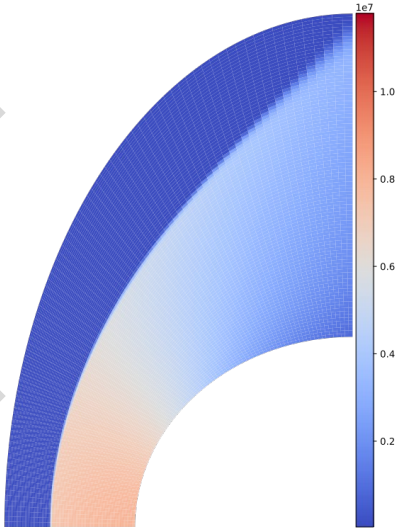
Uninformative initial guess

MPP conv. in t_1

NN conv. in t_2



MPP conv. in t_3



Prediction

$$t_2 + t_3 < t_1$$

NN+MPP: **Exact same** accuracy as **MPP**, **10.6** times faster

Acceptable error: Error study

- Parameter uncertainty error: δ_n
- Discretization error: δ_Δ
- Model error: δ_M

How do they compare to δ_{NN}, the error coming from the approximation of Mutation++ by a neural network?

Error study:

Mesh size: 90×100 (PG/NN/MPP$_{high}$) vs 30×100 (PG/NN/MPP$_{low}$)

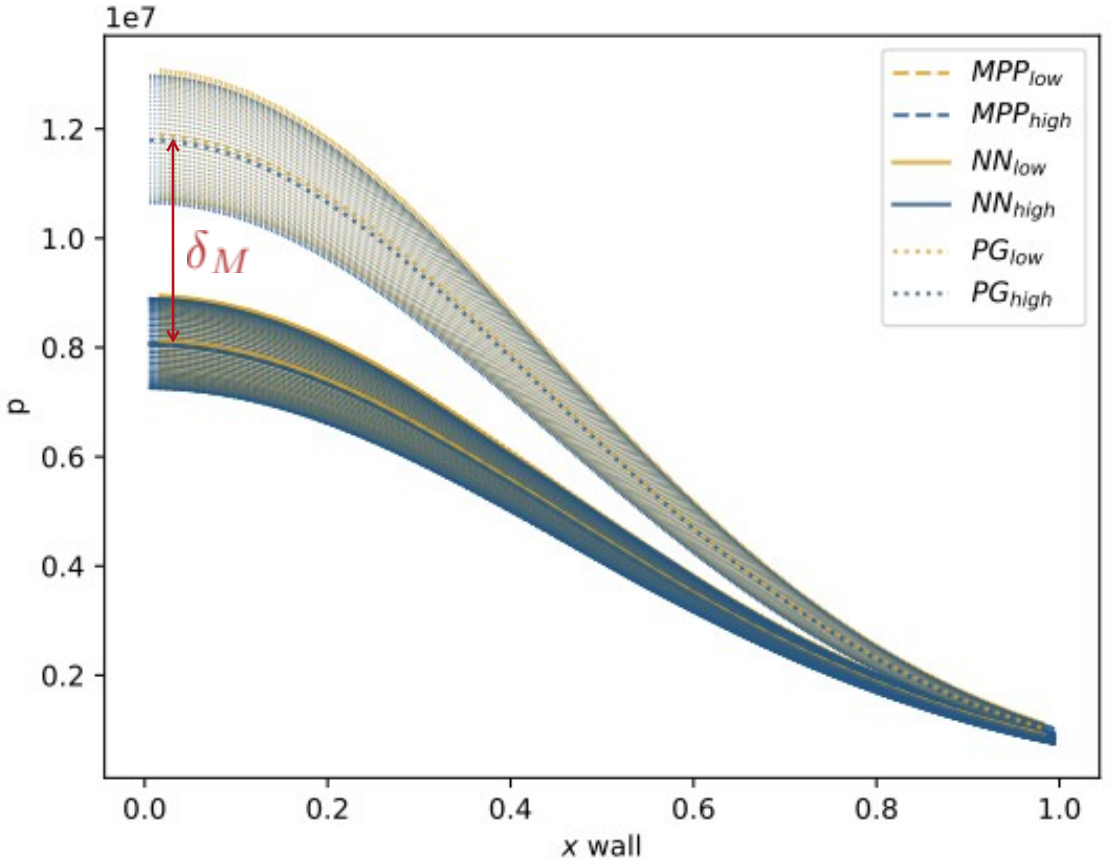
Model: PG vs NN/MPP

Neural network approximation: MPP vs NN

…under the uncertainty of

Upstream speed: $v_0 + \delta_v$ with $\delta_v \sim \mathcal{U}(-0.05v_0, +0.05v_0)$

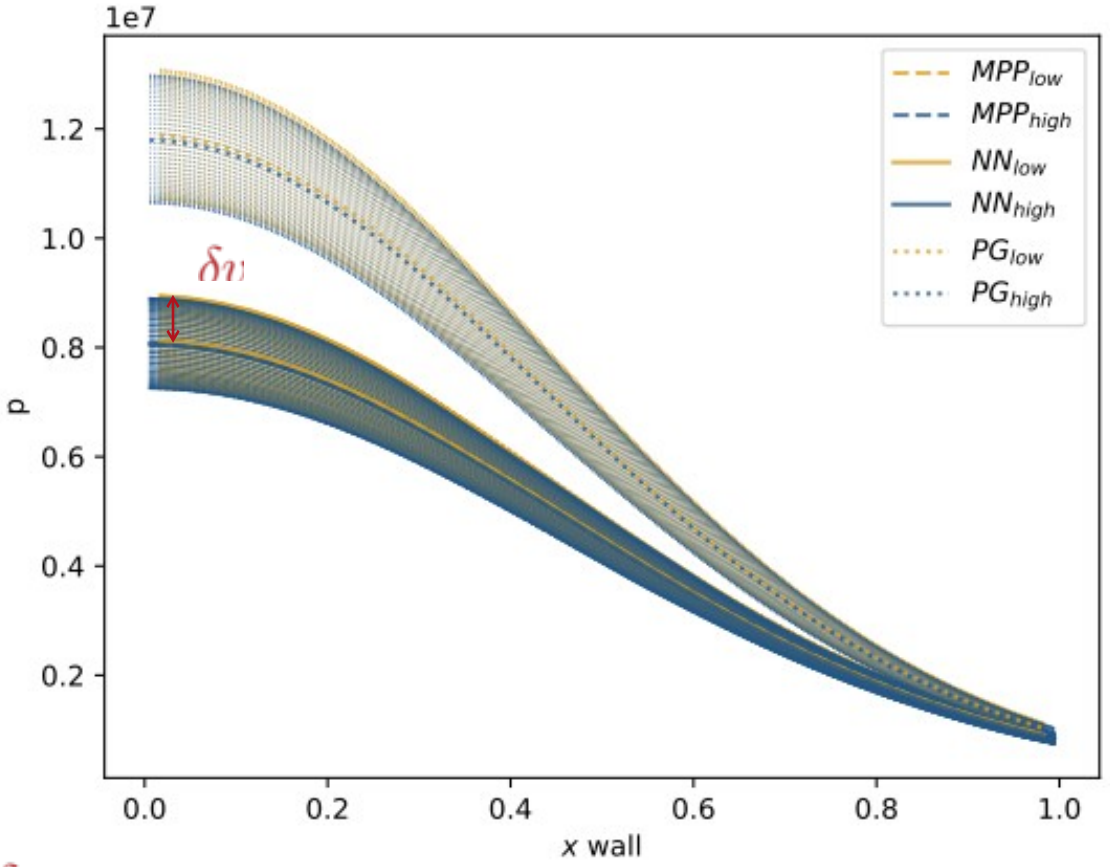
Acceptable error: Error study



Pressure at the wall of the object

- Model: δ_M

Acceptable error: Error study

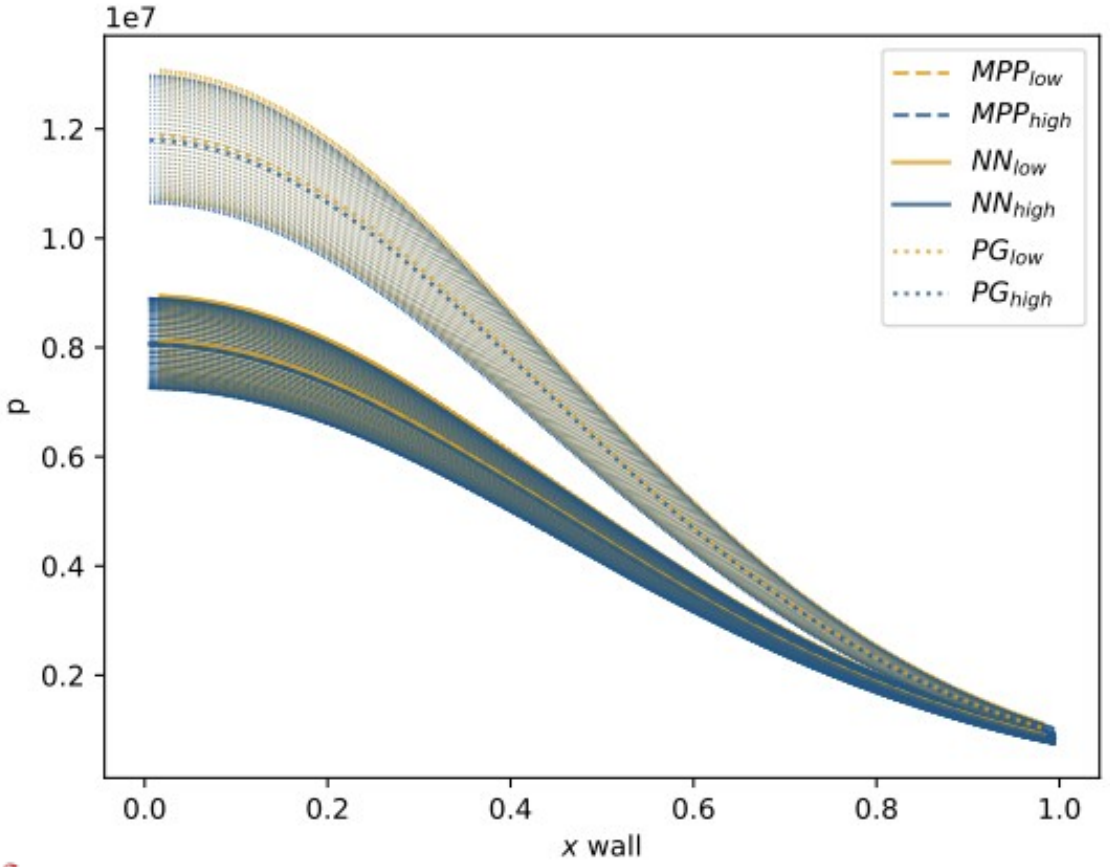


- Model: δ_M
- Uncertainty: δ_U

Pressure at the wall of the object

δ_M

Acceptable error: Error study

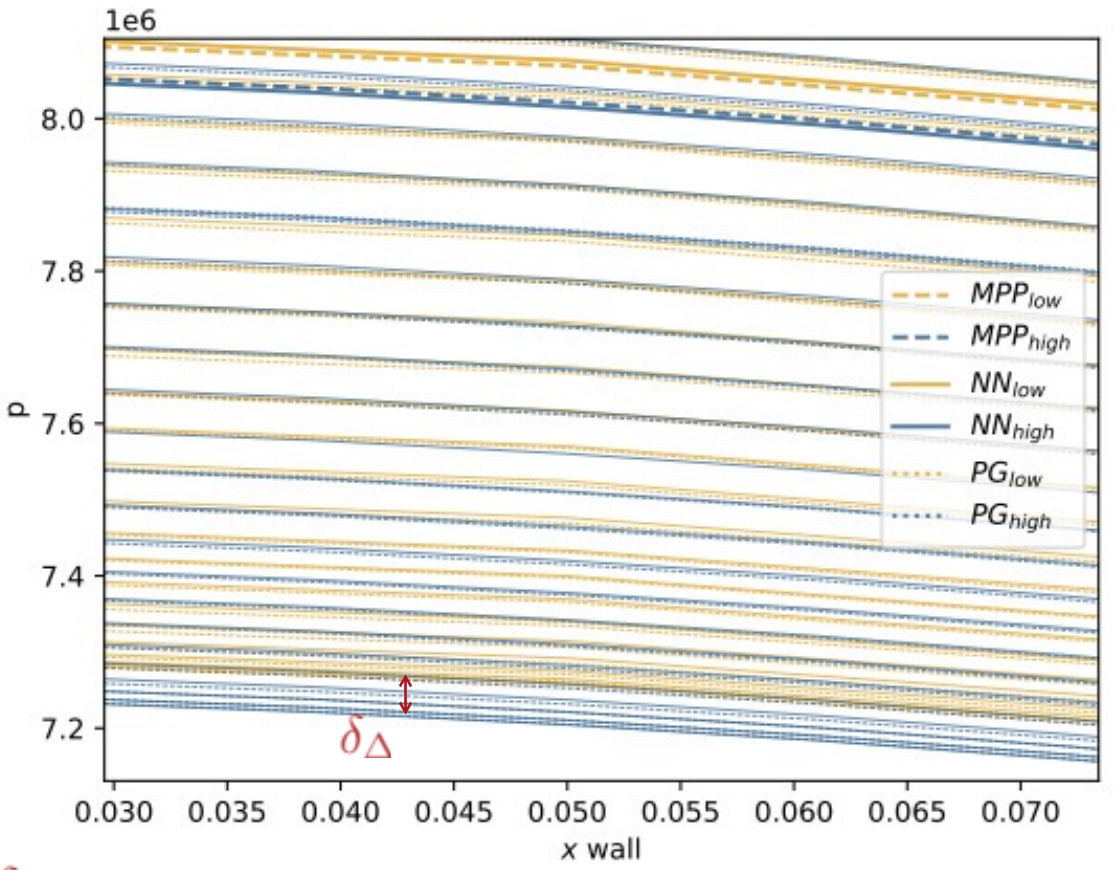


- Model: δ_M
- Uncertainty: δ_U

Pressure at the wall of the object

$$\delta_U < \delta_M$$

Acceptable error: Error study

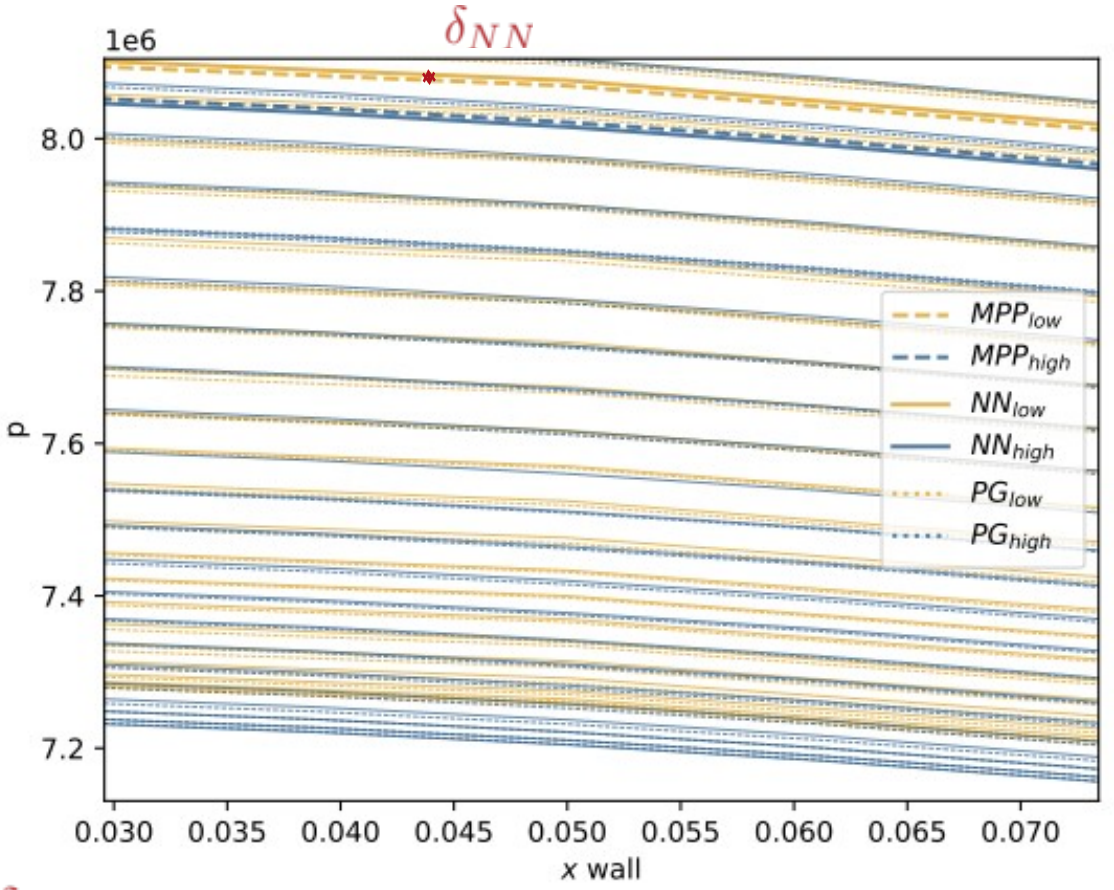


- Model: δ_M
- Uncertainty: δ_U
- Discretization: δ_Δ

Pressure at the wall of the object (zoomed)

$$\delta_U < \delta_M$$

Acceptable error: Error study

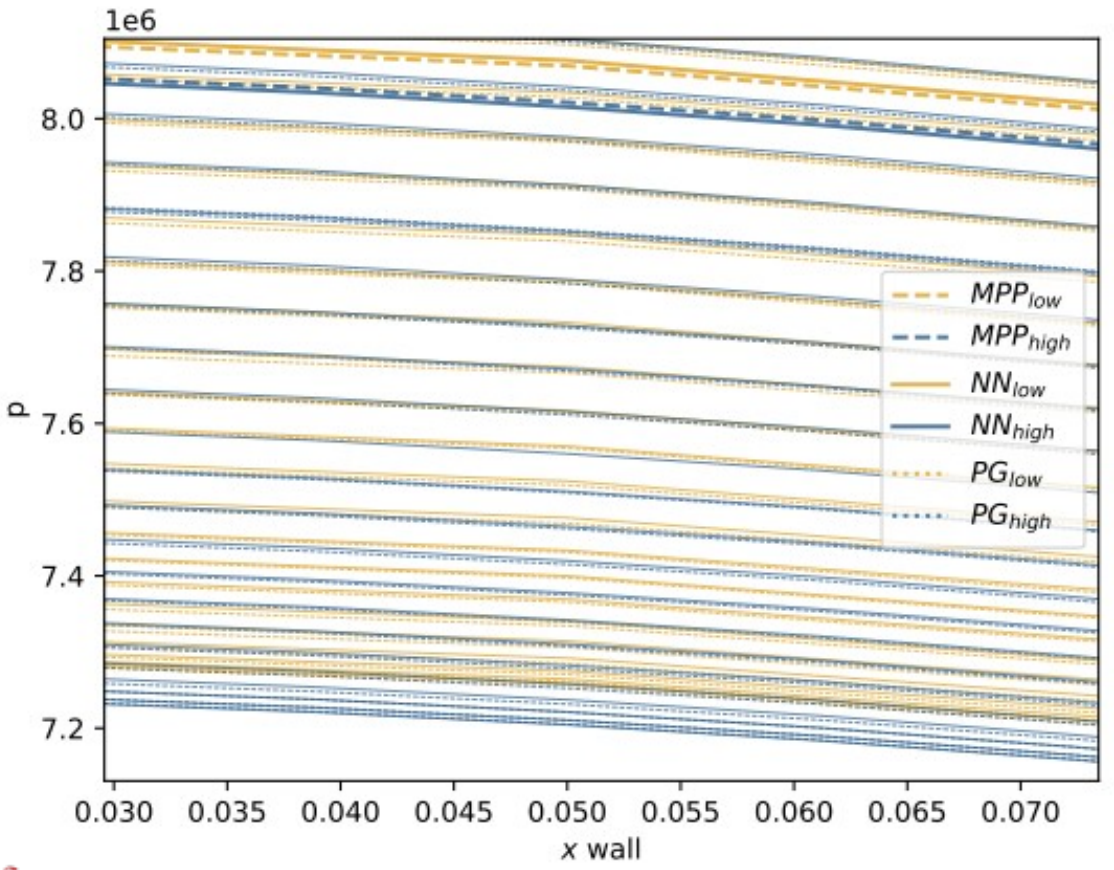


Pressure at the wall of the object (zoomed)

- Model: δ_M
- Uncertainty: δ_U
- Discretization: δ_Δ
- Neural network: δ_{NN}

$$\delta_\Delta < \delta_U < \delta_M$$

Acceptable error: Error study

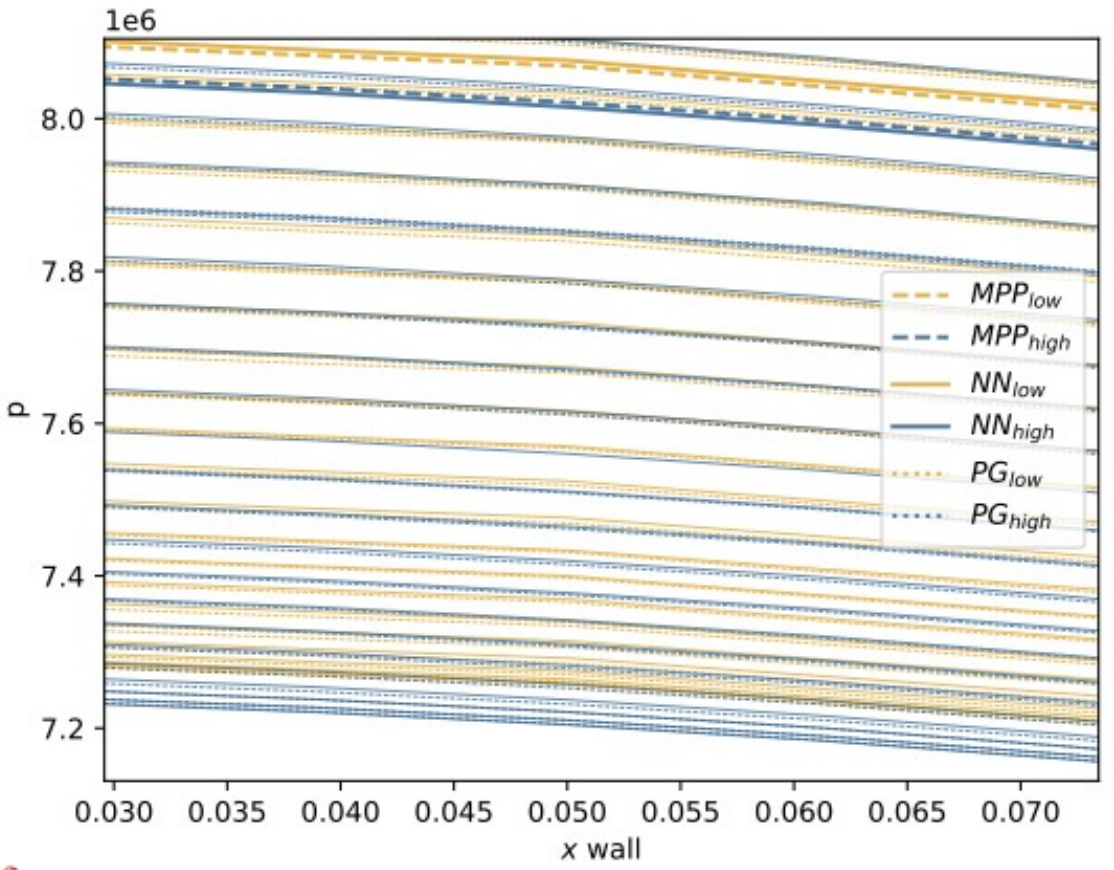


Pressure at the wall of the object (zoomed)

- Model: δ_M
- Uncertainty: δ_{ν}
- Discretization: δ_Δ
- Neural network: δ_{NN}

$$\delta_{NN} < \delta_\Delta < \delta_{\nu} < \delta_M$$

Acceptable error: Error study



Pressure at the wall of the object (zoomed)

- Model: δ_M
- Uncertainty: δ_U
- Discretization: δ_Δ
- Neural network: δ_{NN}

$$\delta_{NN} < \delta_\Delta < \delta_U < \delta_M \Rightarrow \times \mathbf{18.7} \text{ is OK}$$

Acceptable error: Error study

$\delta_{NN} < \delta_{\Delta} < \delta_{N} < \delta_{M} \Rightarrow \times 18.7$ is OK

This study has been conducted with NN+MPP:

- 10 times faster
- Enabled the prediction for some values of δ_{Δ} that led to numerical instabilities, preventing convergence with MPP

Conclusion

1. Estimation	2. Architecture	3. Optimization	4. Generalization
Construction of the training set	Hyperparameter optimization	Training of the neural network	Integration into a simulation code

Contributions [*]

- We constructed a framework for training neural networks based on a PDE formulation of Newton algorithm.
- We derived an optimization algorithm (PDESGD) from this framework, exhibiting interesting properties and good results on simple machine learning test cases.

Perspectives

- Apply PDESGD on large scale deep learning problems, like image and text processing
- Leverage the PDE framework to improve the understanding of the learning process of neural networks.

[*] An analogy between solving Partial Differential Equations with Monte-Carlo schemes and the Optimisation process in Machine Learning (and few illustrations of its benefits),
Gael Poette, Paul Novello, David Lugato, Working report.

Conclusion

1. Estimation	2. Architecture	3. Optimization	4. Generalization
Construction of the training set	Hyperparameter optimization	Training of the neural network	Integration into a simulation code

Contributions [**]

- We approximated the chemical solver of a CFD simulation code with a neural network and obtained a significant speed-up.
- We designed methodologies to obtain accuracy guarantees of the hybrid code.

Perspectives

- Room for computational optimization
- Use this tool to accelerate large scale simulation codes, together with the methodologies to ensure accuracy.

[**] Accelerating hypersonic reentry simulations using deep learning-based hybridization (with guarantees),
Paul Novello, Gael Poette, David Lugato and Pietro Congedo.
In preparation. To be sent to the Journal of Computational Physics

References

1. Barron, Andrew R. “Approximation and Estimation Bounds for Artificial Neural Networks.” *Machine Learning* 14, no. 1 (January 1, 1994): 115–33. https://doi.org/10.1007/BF00993164.
2. Bergstra, James, and Yoshua Bengio. “Random Search for Hyper-Parameter Optimization.” *Journal of Machine Learning Research* 13, no. 10 (2012): 281–305.
3. Bottou, Léon, and Olivier Bousquet. “The Tradeoffs of Large Scale Learning.” In *Advances in Neural Information Processing Systems*, edited by J. Platt, D. Koller, Y. Singer, and S. Roweis, Vol. 20. Curran Associates, Inc., 2008. https://proceedings.neurips.cc/paper/2007/file/0d3180d672e08b4c5312dcdafdf6ef36-Paper.pdf.
4. Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding.” In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–86. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. https://doi.org/10.18653/v1/N19-1423.
5. Eiben, A. E., and M. Schoenauer. “Evolutionary Computing.” *Information Processing Letters* 82, no. 1 (2002): 1–6. https://doi.org/10.1016/S0020-0190(02)00204-1.
6. Gretton, Arthur, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. “Measuring Statistical Dependence with Hilbert-Schmidt Norms.” In *Proceedings of the 16th International Conference on Algorithmic Learning Theory*, 63–77. ALT’05. Berlin, Heidelberg: Springer-Verlag, 2005. https://doi.org/10.1007/11564089_7.
7. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, 770–78.
8. Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. “Multilayer Feedforward Networks Are Universal Approximators.” *Neural Networks* 2, no. 5 (1989): 359–66. https://doi.org/10.1016/0893-6080(89)90020-8.
9. Kingma, Diederik P., and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” In *ICLR (Poster)*, 2015. http://arxiv.org/abs/1412.6980.
10. Li, Lisha, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. “Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization.” *Journal of Machine Learning Research* 18, no. 185 (2018): 1–52.

References

11. Lu, Zhou, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. “The Expressive Power of Neural Networks: A View from the Width.” In *Advances in Neural Information Processing Systems*, edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, 30:6231–39. Curran Associates, Inc., 2017. https://proceedings.neurips.cc/paper/2017/file/32cbf687880eb1674a07bf717761dd3a-Paper.pdf.
12. Prieur, Clémentine, and Stefano Tarantola. “Variance-Based Sensitivity Analysis: Theory and Estimation Algorithms.” In *Handbook of Uncertainty Quantification*, edited by Roger Ghanem, David Higdon, and Houman Owhadi, 1–23. Cham: Springer International Publishing, 2016. https://doi.org/10.1007/978-3-319-11259-6_35-1.
13. Mockus, Jonas. “On Bayesian Methods for Seeking the Extremum.” *Proceedings of the IIP Technical Conference, 1974*.
14. Sobol, Ilya M. “Sensitivity Estimates for Nonlinear Mathematical Models.” *MMCE*, no. 1 (1993): 407–14.
15. Spagnol, Adrien, Rodolphe Le Riche, and Sébastien Da Veiga. “Global Sensitivity Analysis for Optimization with Variable Selection.” *SIAM/ASA J. Uncertain. Quantification* 7 (2018): 417–43.
16. Stanley, Kenneth O., and Risto Miikkulainen. “Evolving Neural Networks through Augmenting Topologies.” *Evol. Comput.* 10, no. 2 (June 2002): 99–127. https://doi.org/10.1162/106365602320169811.
17. Scoggins, James B., Vincent Leroy, Georgios Bellas-Chatzigeorgis, Bruno Dias, and Thierry E. Magin. “Mutation++: MUlticomponent Thermodynamic And Transport Properties for IONized Gases in C++.” *SoftwareX* 12 (juillet 2020): 100575. https://doi.org/10.1016/j.softx.2020.100575.
18. Razavi, Saman, Anthony Jakeman, Andrea Saltelli, Clémentine Prieur, Bertrand Iooss, Emanuele Borgonovo, Elmar Plischke, et al. “The Future of Sensitivity Analysis: An Essential Discipline for Systems Modeling and Policy Support.” *Environmental Modelling & Software* 137 (March 1, 2021): 104954. https://doi.org/10.1016/j.envsoft.2020.104954.
19. Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.” *ArXiv:2003.08934 [Cs]*, August 3, 2020. http://arxiv.org/abs/2003.08934.
20. Zeiler, Matthew D. “ADADELTA: An Adaptive Learning Rate Method.” *ArXiv:1212.5701 [Cs]*, December 22, 2012. http://arxiv.org/abs/1212.5701.
21. Gitman, Igor, Hunter Lang, Pengchuan Zhang, and Lin Xiao. “Understanding the Role of Momentum in Stochastic Gradient Methods.” *ArXiv:1910.13962 [Cs, Math, Stat]*, October 30, 2019. http://arxiv.org/abs/1910.13962.
22. Baker, Alexander, Bremer, Hagberg, Kevrekidis, Najm, Parashar, Patra, Sethian, Wild, Willcox, Lee. “Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence”, 2019
23. Peluchon, “Approximation numérique et modelisation de l'ablation liquid”, Université de Bordeaux, 2017
24. Da Veiga, Gamboa, Iooss, Prieur. “Basics and Trends in Sensitivity Analysis : Theory and Practice in R”, 2021

References

25. Fidkowski, K., and Guodong Chen. “Metric-Based, Goal-Oriented Mesh Adaptation Using Machine Learning.” *J. Comput. Phys.*, 2021. https://doi.org/10.1016/j.jcp.2020.109957.
26. Raissi, Maziar, Paris Perdikaris, and George Em Karniadakis. “Physics Informed Deep Learning (Part I): Data-Driven Solutions of Nonlinear Partial Differential Equations.” *ArXiv Preprint ArXiv:1711.10561*, 2017.
27. Raissi, M., P. Perdikaris, and G. E. Karniadakis. “Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations.” *Journal of Computational Physics* 378 (2019): 686–707. https://doi.org/10.1016/j.jcp.2018.10.045.
28. Gramacy, Robert B. *Surrogates: Gaussian Process Modeling, Design and \ Optimization for the Applied Sciences*. Boca Raton, Florida: Chapman Hall/CRC, 2020.
29. Kluth, G., K. D. Humbird, B. K. Spears, J. L. Peterson, H. A. Scott, M. V. Patel, J. Koning, M. Marinak, L. Divol, and C. V. Young. “Deep Learning for NLTE Spectral Opacities.” *Physics of Plasmas* 27, no. 5 (2020): 052707. https://doi.org/10.1063/5.0006784.
30. Kluth, Gilles, Kelli Humbird, Brian Spears, Howard Scott, Mehul Patel, Luc Peterson, Joe Koning, Marty Marinak, Laurent Divol, and Chris Young. “Deep Learning for Non-Local Thermodynamic Equilibrium in Hydrocodes for ICF.” In *APS Division of Plasma Physics Meeting Abstracts*, 2019:BO5.009. APS Meeting Abstracts, 2019.
31. Fort, Jean-Claude, Thierry Klein, and Nabil Rachdi. “New Sensitivity Analysis Subordinated to a Contrast.” *Communications in Statistics - Theory and Methods* 45, no. 15 (2016): 4349–64. https://doi.org/10.1080/03610926.2014.901369.
32. Da Veiga, Sébastien. “Kernel-Based ANOVA Decomposition and Shapley Effects -- Application to Global Sensitivity Analysis.” *ArXiv:2101.05487 [Math, Stat]*, January 14, 2021. http://arxiv.org/abs/2101.05487.