FROM RESEARCH TO INDUSTRY

INTRODUCTION TO PHYSICS INFORMED NEURAL NETWORKS (PINN)

14 décembre 2022, MS-HPC-IA, Sophia-Antipolis

P. Kestener

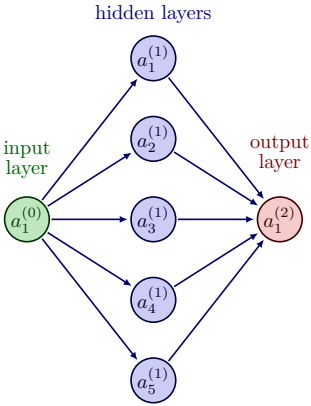
CEA

Résumé

- **Qu'est ce qu'un PINN (Physics-Informed Neural Networks) ?**
  - mini-historique, >= 2018
  - **data-driven** versus **model-driven**
- **Quels types de problèmes ?** : EDP, problème inverse, quantification d'incertitude, ...
- **Quelles applications ? Quelles EDP ?**
- **Quels frameworks logiciels ?**
  - modulus (Nvidia) : surcouche au dessus de pytorch + tiny-cuda-nn (implantation optimisée pour les réseaux *fully-connected*)
  - `https://github.com/maziarraissi/PINNs` (2017), 1ère implantation
  - DeepXDE, NeuralPDE.jl, NeuroDiffEq, SciANN, ...
- **Un exemple rapide, les *operateurs neuronaux*, une mini-biblio**

Short remainder: Universal approximation theorem(s)

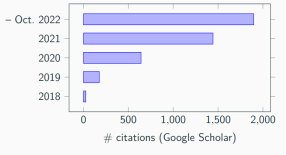
- **Given a function $y = f(x)$ (with some conditions), can we find a neural network such that its output $\widehat{f}(x)$ can approximate $f(x)$ with arbitrary precision ?**
- i.e. find weights w_i, θ_i, α_i which defines $\widehat{f}(x) = \sum_j \alpha_i \sigma(w_i x + \theta_i)$?
- each neural networks node realizes a sigmoïdal operation : $\alpha\sigma(w * x + \theta)$
- **answer is yes.**
  for example we consider $f \in C(I_n)$, $\forall \epsilon$, there exist $G(x) = \sum_i \alpha_i \sigma(w_i x + \theta_i)$, for which $|G(x) - f(x)| < \epsilon, \forall x \in I_n$
  Approximation by superpositions of sigmoïdal function, G. Cybenko, Math. Control Signals Systems, (1989) 2: 303-314.
- but how many nodes in the hidden layer ? Does it work in practice ?
- finding networks parameters $\boldsymbol{\theta} = (w_i, \theta_i, \alpha_i)$ knowing x_i, y_i is called **data-driven supervised learning**, but how ?
  One defines a **loss function**, e.g. $L(\boldsymbol{\theta}) = \sum_i ||y_i - \widehat{y}_i(\boldsymbol{\theta})||_2^2$
  $\Rightarrow$ then use minimization, e.g. **(stochastic) gradient descent** : $\boldsymbol{\theta_{n+1}} = \boldsymbol{\theta_n} - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_n})$
- once neural network is trained, you can use it to **make predictions**: i.e. for each x **compute** $\widehat{y}$ **as an approximation of the** *true* $y = f(x)$



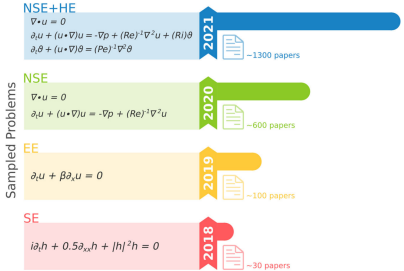
$x = a_1^{(0)}$ and $y = a_1^{(2)} = \widehat{f}(x)$

Early work on using neural networks to solve PDEs

- Early work of Lagaris, etal : Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Networks 9(5), 9871000 (1998).
- Et après plus rien jusque ≈ 2017...
  Raissi etal, https://arxiv.org/pdf/1711.10561.pdf, publié dans JCP début 2019.



Number of articles referencing Raissi or including PINN in title

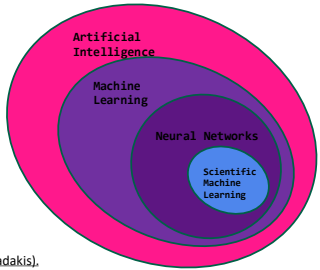


Cuomo etal, Scientific Machine Learning Through Physics-Informed Neural Networks: Where we are and What's Next, Journal of Scientific Computing, vol. 92, (2022)

SciML history

https://www.import.io/post/history-of-deep-learning/

- Artificial intelligence (AI) > Machine Learning (ML) > Deep Learning > Scientific Machine Learning (SciML).
- The expression "Deep Learning" was (probably) first used by Igor Aizenberg and colleagues around 2000.
- 1960s: Shallow Neural Networks.
- 1982: Hopfield Network – A Recurrent NN.
- 1988-89: Learning by backpropagation, Rumelhart, Hinton & Williams; hand-written text, LeCun.
- 1993 NVIDIA was founded; GeForce is the first GPU.
- 1990s Unsupervised Deep Learning.
- 1993: A Recurrent NN with 1,000 layers (Jürgen Schmidhuber)
- 1994: NN for solving PDEs, Dissanayake & Phan-Thien
- 1998: Gradient-based learning, LeCun.
- 1998: ANN for solving ODEs&PDEs, Lagaris, Likas & Fotiadis
- 1990-2000: Supervised Deep Learning.
- 2006: A fast learning algorithm for deep belief nets, Hinton.
- 2006-present: Modern Deep Learning.
- 2009: ImageNet: A large-scale hierarchical image database (Fei-Fei).
- 2010: GPUs are only up to 14 times faster than CPUs (Intel).
- 2010: Tackling the vanishing/exploding gradients: Glorot & Bengio.
- 2011: AlexNet – Convolutional NN (CNN) - Alex Krizhevsky.
- 2014: Generative Adversarial Networks (GANs) – Ian Goodfellow.
- 2015: Batch normalization, Ioffe & Szegedy.
- 2017: PINNs: Physics-Informed Neural Networks (Raissi, Perdikaris, Karniadakis).
- 2019: Scientific Machine Learning (ICERM workshop Jan. 2019; DOE report, Feb 2019).
- 2019: DeepONet – Operator regression (Lu, Jin, Karniadakis).

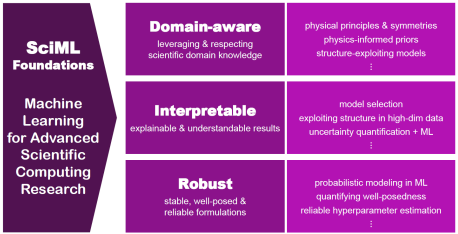


ref: https://raw.githubusercontent.com/lululxvi/tutorials/master/20220722_sciml/sciml_introduction.pdf

SciML history

Basic Research Needs Workshop for Scientific Machine Learning
Core Technologies for Artificial Intelligence, DOE ASCR Report, Feb 2019

❑ Scientific machine learning (**SciML**) is a core component of artificial intelligence (AI) and a computational technology that can be trained, with scientific data, to augment or automate human skills.



❑ SciML must achieve the same level of scientific rigor expected of established methods deployed in science and applied mathematics. Basic requirements include validation and limits on inputs and context implicit in such validations, as well as verification of the basic algorithms to ensure they are capable of delivering known prototypical solutions.

❑ Can SciML achieve Robustness?

- NSF/ICERM Workshop on SciML, January 28-30, 2019 (Organizers: J. Hesthaven & G.E. Karniadakis)
- https://icerm.brown.edu/events/ht19-1-sml/)

ref: https://raw.githubusercontent.com/lululxvi/tutorials/master/20220722_sciml/sciml_introduction.pdf

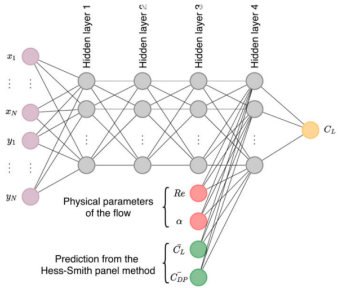
What are PINNs ?

- An alternative / complementary approach to purely data-driven NN training
- How can **physics information** can be introduced ?
  - either in the **loss function** (used to train the NN)
  - either in the **geometry / architecture** of the NN
- y is a vector of observation, $\widehat{y}$ is a vector of predicted values (NN output)
- define a loss function that is a mixture of **data-driven** and **model/physics-driven**

$$L_{total} = \lambda_d L_{data}(y, \widehat{y}) + \lambda_p L_{\text{physics}}(\widehat{y})$$

example of physics-guided NN with inhomogeneous architecture

Lift (portance) prediction



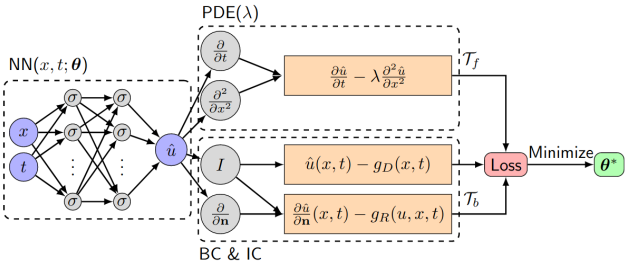
(x_i, y_i) for $i = 1...N$ are points on the airfoil surface.

ref: https://aip.scitation.org/doi/10.1063/5.0038929

What are PINNs ?

Using PINN to solve Partial Differential Equations (physics-informed loss function)

- e.g. diffusion equation : $\frac{\partial u}{\partial t} - \lambda \frac{\partial^2 u}{\partial x^2} = 0$, with initial conditions and border conditions



- **key ingredient** : **use automatic/algorithmic differentiation** to compute $\nabla_{\boldsymbol{\theta}} L$ (gradient descent) which also requires $\frac{\partial \hat{u}}{\partial t}$, ... computed with chain rule (back-propagation)
- observed data not needed, neural networks training can be completely model-driven (PDE-driven); just use random values for x and t, evaluate $\hat{u}$, $L(\hat{u})$ and update parameters (as before when data-driven)

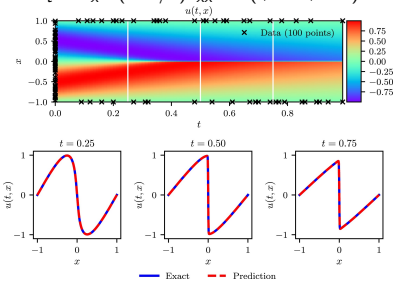
reference: DeepXDE: A deep Learning library for solving differential equations (2019).

What are PINNs ?

Raissi etal, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, JCP, vol 378, 2019. ⇒ **example: viscous Burgers**

solve:

$u_t + uu_x - (0.01/\pi)u_{xx} = 0$ (+ IC + BC)

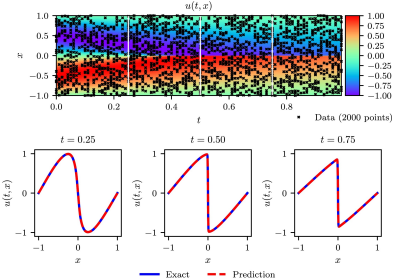


- 9-layers, 20 neurons/layers, 3021 parameters ($\boldsymbol{\theta}$)
- Loss function to optimise :
  $L(\boldsymbol{\theta}) = \frac{1}{N_f}\sum_i |f(t_i,u_i)|^2 + \frac{1}{N_d}\sum_j |u(t_j,x_j) - u_j|^2$

parameter identification:

find (λ_1, λ_2) for $u_t + \lambda_1 uu_x - \lambda_2 u_{xx} = 0$

$(u, \lambda_1, \lambda_2)$ are outputs of NN



Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Identified PDE (clean data)	$u_t + 0.99915uu_x - 0.0031794u_{xx} = 0$
Identified PDE (1% noise)	$u_t + 1.00042uu_x - 0.0032098u_{xx} = 0$

What are PINNs ?

Raissi etal, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, JCP, vol 378, 2019.

- https://github.com/maziarraissi/PINNs
- PDE ?
  - Non-linear Schrödinger eq : $ih_t + 0.5h_{xx} + |h|^2 h = 0$
  - Allen-Cahn (Reaction-Diffusion, biphasic flow): $u_t - 0.0001u_{xx} + 5u^3 - 5u = 0$
  - Navier-Stokes incompressible
  - Korteweg-de Vries / shallow water

PINN theoretical difficulties

- Training relies on optimizing a non-convex problem for several 1000's of parameters
- Training ($\boldsymbol{\theta}$ optimization) does not converge when activation function is not sufficiently smooth [1]
- How to assess if PINN converges to the correct solution of a differential equation ?
- What is the influence of **depth** (number of hidden layers) and **width** (number of neuron per layer) on the quality of training ? Just validation ?
- Long training time. How to chose inputs NN during training ?
- Which types of PDE ? stiff PDE ?
- **Beginning of interpretation ?** initial hidden layers may be in charge of encoding low-frequency components and subsequent hidden layers may be in charge of representing higher-frequency component [2]

[1] Markidis etal, Designing Next-Generation Numerical Methods with Physics-Informed Neural Networks (2022).

[2] Markidis, S.: The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers? Frontiers in Big Data 4 (2021).

PINN - théorie de l'approximation

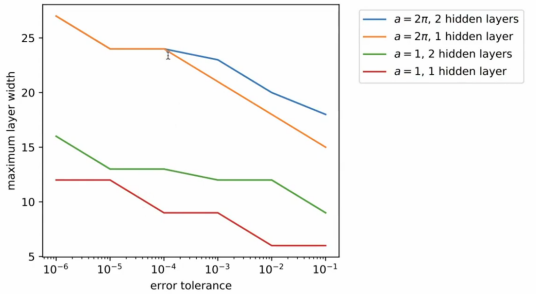
- **théorème d'approximation de fonction par NN** [3]:
  $\forall \epsilon > 0, \forall f \in C^k([0,1]^d)$, il existe un réseau de neurone à fonction d'activation tanh dont la sortie $\widehat{f}$ est telle que $||f - \widehat{f}||_\infty < \epsilon$ et avec $\mathcal{O}(\epsilon^{-d/k})$ neurones.
- Existe-t'il des résultats théoriques concernant des NN qui approximent des solutions d'EDP ? oui mais peu (apparemment)
- **Questions:** pour une EDP donnée (diffusion) + BC + IC
  - **existence**: existe-t'il un PINN tel que $\lambda_R|\mathscr{R}\widehat{u}| + \lambda_B|\mathscr{B}\widehat{u}|$ petit et si oui pour quelle taille de réseau ? rappel : en pratique on minimise $\frac{\lambda_R}{N}\sum_{n=1}^{N}||\mathscr{R}\widehat{u}(x_n)|| + \frac{\lambda_B}{M}\sum_{m=1}^{M}||\mathscr{B}\widehat{u}(x'_m)||$
  - **stabilité:** si l'entrainement du PINN converge, est-ce que $|u - \widehat{u}|_{L^2}$ petit également ?
  - **généralisation:** une fois le réseau entrainé, est-ce qu'on peut garantir que l'erreur de généralisation $(\forall x)$ sera petite ?
- **Réponses:** On the approximation of functions by tanh neural networks, De Ryck et al, Neural Networks, Vol. 143, Nov. 2021

[3] Error bounds for approximations with deep ReLU networks Neural Networks, 94 (2017), pp. 103-114, https://doi.org/10.1016/j.neunet.2021.08.015

PINN - théorie de l'approximation

- **Quelle taille de réseau pour approcher e.g.** $f_a : [0,1] \mapsto [-1,1] : x \mapsto \sin(ax), a > 0$ **?**

- $\|f_a - \widehat{f}^{N,s}\|_\infty \leq \epsilon$



ref: On the approximation of functions by tanh neural networks, De Ryck et al, Neural Networks, Vol. 143, Nov. 2021

PINN - théorie de l'approximation

- Error estimates for physics informed neural networks approximating the Navier-Stokes equations, https://arxiv.org/abs/2203.09346

PINNs for Navier-Stokes equations

Bounds in $W^{k,\infty}$-norm $\Rightarrow$ existence of NNs with small PINN error

Case study: Navier-Stokes equations in d space dimensions (periodic BC)

$$\begin{cases} u_t + u \cdot \nabla u + \nabla p - \nu \Delta u = 0 & \text{in } D \times [0,T], \\ \operatorname{div}(u) = 0 & \text{in } D \times [0,T], \\ u(t=0) - u_0 = 0 & \text{in } D. \end{cases} \quad \Rightarrow \quad \text{residuals} \begin{cases} \mathcal{R}_{\text{PDE}} \\ \mathcal{R}_{\text{div}} \\ \mathcal{R}_{\text{t}} \end{cases}$$

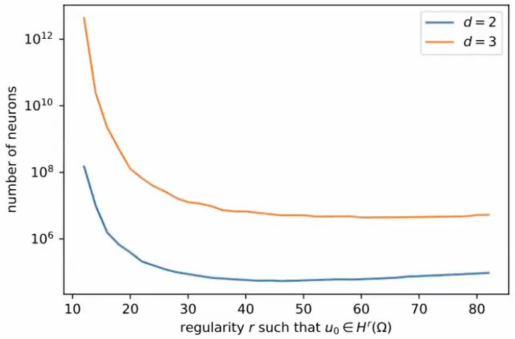
Theorem [DR, Jagtap, and Mishra, 2022]

Let $u_0 \in H^r$ with $r > \frac{d}{2} + 2k$ and $\operatorname{div}(u_0) = 0$. For every $N \in \mathbb{N}$ there exists a tanh neural network $\widehat{u}$ with 2 hidden layers of width N^{d+1} s.t.

$$\|\mathcal{R}_{\text{PDE}}[\widehat{u}]\|_{L^2} + \|\mathcal{R}_{\text{div}}[\widehat{u}]\|_{L^2} + \|\mathcal{R}_{\text{t}}[\widehat{u}]\|_{L^2} \leq C(\ln(cN))^2 N^{-k+2}.$$

PINN - théorie de l'approximation

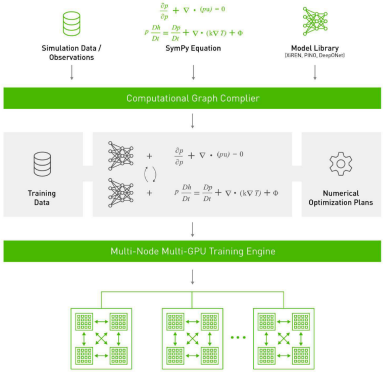
- Quelle taille de réseau pour approcher une solution de Navier-Stokes avec une erreur de 1% ?
- Ca fait beaucoup de neurones; probablement que le résultat pourra être amélioré.



ref: Error estimates for physics informed neural networks approximating the Navier-Stokes equations, https://arxiv.org/abs/2203.09346

Nvidia modulus

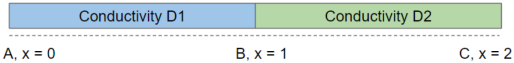
- modulus documentation : https://docs.nvidia.com/deeplearning/modulus/index.html
- modulus source code : https://gitlab.com/nvidia/modulus/modulus
- modulus examples : https://gitlab.com/nvidia/modulus/examples
- les sources de modulus sont librement disponibles mais faut juste s'enregistrer sur le site d'nvidia avant: https://developer.nvidia.com/modulus
- c'est encore un peu expérimental, un peu difficile à installer; mais une image docker est disponible pour tester rapidement



les sources des TP sont là : https://github.com/openhackathons-org/gpubootcamp/tree/master/hpc_ai/PINN

Un exemple simple : équation de la chaleur avec modulus

- Equation de la chaleur dans une barre



$$\frac{\mathrm{d}}{\mathrm{d}x}\left(D_1 \frac{\mathrm{d}U_1}{\mathrm{d}x}\right) = 0, \qquad \text{when } 0 < x < 1$$

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(D_2 \frac{\mathrm{d}U_2}{\mathrm{d}x}\right) = 0, \qquad \text{when } 1 < x < 2$$

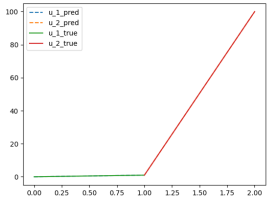
- continuité du flux thermique en $x = 1$, continuité de la température en $x = 1$
- Conditions de bords de Dirichlet, $T(x = 0) = T_0$ et $T(x = 2) = T_2$

Un exemple simple : équation de la chaleur avec modulus

```python
from sympy import Symbol, Eq, Function, Number
from modulus.eq.pde import PDE

class Diffusion(PDE):
    name = "Diffusion"

    def __init__(self, T="T", D="D", Q=0, dim=3, time=True):
        # set params
        self.T = T
        self.dim = dim
        self.time = time

        # coordinates
        x, y, z = Symbol("x"), Symbol("y"), Symbol("z")

        # time
        t = Symbol("t")

        # make NN input variables
        input_variables = {"x": x, "y": y, "z": z, "t": t}
        if self.dim == 1:
            input_variables.pop("y")
            input_variables.pop("z")
        elif self.dim == 2:
            input_variables.pop("z")
        if not self.time:
            input_variables.pop("t")

        # Temperature
        assert type(T) == str, "T needs to be string"
        T = Function(T)(*input_variables)

        # Diffusivity
        if type(D) is str:
            D = Function(D)(*input_variables)
        elif type(D) in [float, int]:
            D = Number(D)

        # Source
        if type(Q) is str:
            Q = Function(Q)(*input_variables)
        elif type(Q) in [float, int]:
            Q = Number(Q)

        # set equations
        self.equations = {}
        self.equations["diffusion_" + self.T] = (
            T.diff(t)
            - (D * T.diff(x)).diff(x)
            - (D * T.diff(y)).diff(y)
            - (D * T.diff(z)).diff(z)
            - Q
        )
```

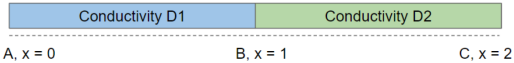
```python
@modulus.main(config_path="conf", config_name="config")
def run(cfg: ModulusConfig) -> None:

    # make list of nodes to unroll graph on
    diff_u1 = Diffusion(T="u_1", D=D1, dim=1, time=False)
    diff_u2 = Diffusion(T="u_2", D=D2, dim=1, time=False)
    diff_in = DiffusionInterface("u_1", "u_2", D1, D2, dim=1, time=False)

    diff_net_u_1 = instantiate_arch(
        input_keys=[Key("x")],
        output_keys=[Key("u_1")],
        cfg=cfg.arch.fully_connected,
    )
    diff_net_u_2 = instantiate_arch(
        input_keys=[Key("x")],
        output_keys=[Key("u_2")],
        cfg=cfg.arch.fully_connected,
    )

    nodes = (
        diff_u1.make_nodes()
        + diff_u2.make_nodes()
        + diff_in.make_nodes()
        + [diff_net_u_1.make_node(name="u1_network", jit=cfg.jit)]
        + [diff_net_u_2.make_node(name="u2_network", jit=cfg.jit)]
    )

    # make domain add constraints to the solver
    domain = Domain()

    # sympy variables
    x = Symbol("x")

    # right hand side (x = 2) Pt c
    rhs = PointwiseBoundaryConstraint(
        nodes=nodes,
        geometry=L2,
        outvar={"u_2": Tc},
        batch_size=cfg.batch_size.rhs,
        criteria=Eq(x, 2),
    )
    domain.add_constraint(rhs, "right_hand_side")

    # ...
    # interior 1
    interior_u1 = PointwiseInteriorConstraint(
        nodes=nodes,
        geometry=L1,
        outvar={"diffusion_u_1": 0},
        bounds={x: (0, 1)},
        batch_size=cfg.batch_size.interior_u1,
    )
    domain.add_constraint(interior_u1, "interior_u1")
    # ...
        # make solver
    slv = Solver(cfg, domain)

    # start solver
    slv.solve()
```

```yaml
# INPUT CONFIG
defaults :
  - modulus_default
  - arch:
      - fully_connected
  - scheduler: tf_exponential_lr
  - optimizer: adam
  - loss: sum
  - _self_

arch:
    fully_connected:
        layer_size: 256

save_filetypes : "vtk,npz"

scheduler:
  decay_rate: 0.95
  decay_steps: 100

optimizer:
  lr : 1e-4

training:
  rec_results_freq: 1000
  max_steps : 5000

batch_size:
  rhs: 2
  lhs: 2
  interface: 2
  interior_u1: 200
  interior_u2: 200
```

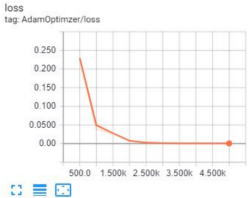


Un exemple simple : équation de la chaleur avec modulus

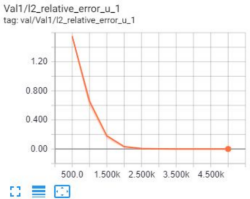
- Equation de la chaleur dans une barre

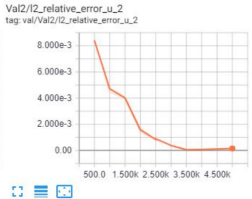


- monitoring des itérations de l'entrainement avec TensorBoard
- erreur de validation: calculée avec une solution analytique (quand elle est connue), ou avec des data (lues depuis un fichier numpy ou VTK) simulées ou expérimentales

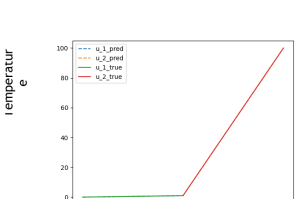


Training error





Validation error



Un exemple simple : équation de la chaleur avec modulus

- Et si on met un coefficient de diffusion qui dépend de x ? e.g. linéaire en x
- L'entrainement ne converge plus !
- Mais on ré-entraîne le réseau en mettant des poids sur les points d'entrainement en utilisant la SDF (Signed Distance Function), ça remarche !
- Si on connait des invariants du problème, il vaut mieux les mettre en contraintes de la fonction de coût, ça améliore l'efficacité de l'entrainement, et améliore en général la qualité de la solution (constaté empiriquement).
- recettes de cuisine *magiques* :

  https://docs.nvidia.com/deeplearning/modulus/user_guide/theory/recommended_practices.html

Nvidia modulus

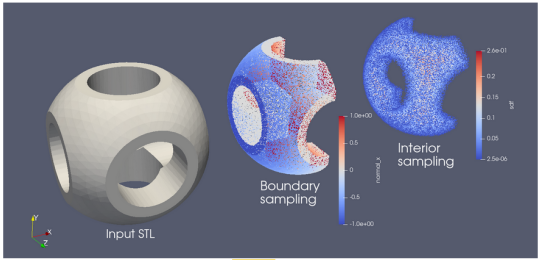


Fig. 40 *Tesselated Geometry sampling using Modulus*

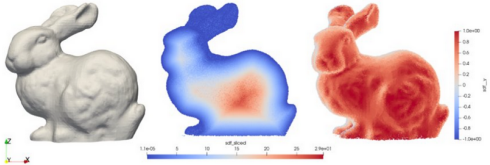
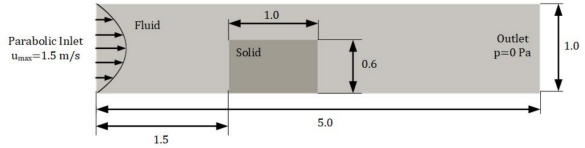


Fig. 41 *Tesselated Geometry sampling using Modulus: Stanford bunny*

CHALLENGE: FLOW OVER 2D CHIP

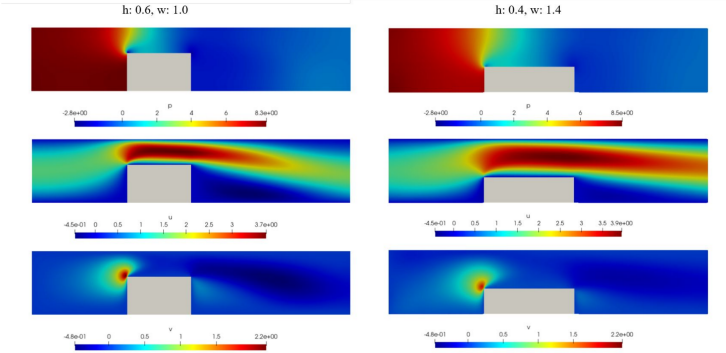


- Solve the flow over 2D chip for the given boundary conditions. The challenge problem has 3 parts:

1. Solve the fluid flow for the given boundary conditions and geometry
2. Solve the fluid flow for the parameterized Chip geometry
3. Solve the inverse problem where, given a flow field, use it to invert out the viscosity of the flow

Nvidia modulus

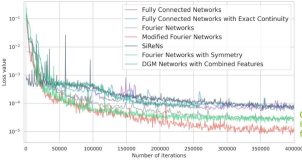
CHALLENGE: FLOW OVER 2D CHIP



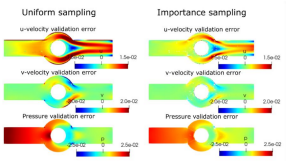
MODULUS FEATURES AND ADVANCEMENTS

Several neural network architectures:
- Fully connected Network
- Fourier Feature Network
- Sinusoidal Representation Network (SiReN)
- Modified Fourier Network
- Deep Galerkin Method Network
- Modified Highway Network
- Multiplicative Filter Networks

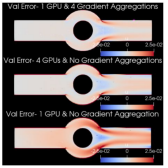
Other Features include:
- Global and local learning rate annealing
- Global adaptive activation functions
- Halton sequences for low-discrepancy point cloud creation
- Gradient Accumulation
- Time-stepping schemes for transient problems
- Temporal loss weighting and time marching for the continuous time approach
- Importance sampling
- Homoscedastic task uncertainty quantification for loss weighting



Comparisons of various networks in Modulus applied to solve the flow over a heatsink

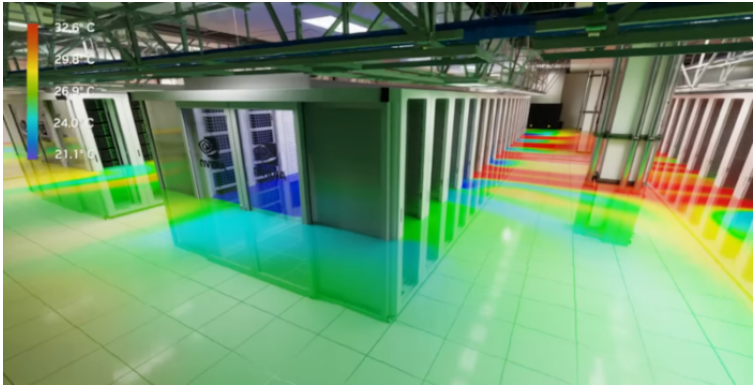


Importance sampling



Handling larger batch sizes using Multi-GPU and/or Gradient Aggregation

Nvidia modulus: digital twin of a datacenter

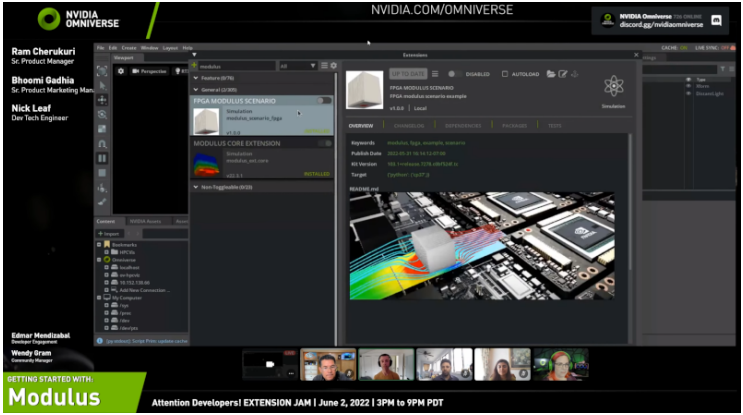


ref: https://blogs.nvidia.com/blog/2022/11/14/omniverse-digital-twin-data-center/ (air flow + heat transfert)

https://developer.nvidia.com/blog/building-scientifically-accurate-digital-twins-using-modulus-with-omniverse-and-ai/

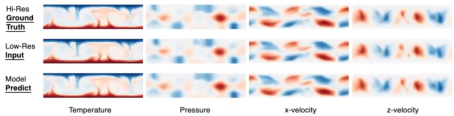
Nvidia modulus / Omniverse

- real-time rendering of a parametrized modulus inference,
- engineering design optimization studies

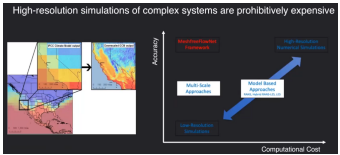


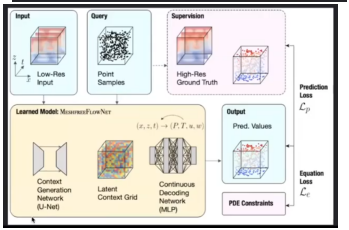
Exemples d'applications des PINNs

- météorologie : https://arxiv.org/pdf/2005.01463.pdf



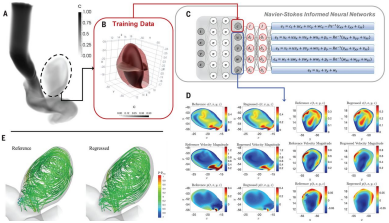
- hémodynamique
- article de revue :
  https://link.springer.com/article/10.1007/s10915-022-01939-z





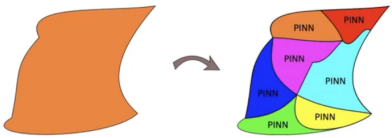
Exemples d'applications des PINNs

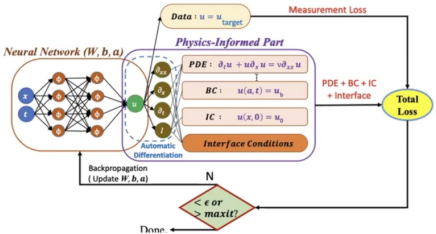
- météorologie : https://arxiv.org/pdf/2005.01463.pdf
- hémodynamique
- article de revue :
  https://link.springer.com/article/10.1007/s10915-022-01939-z



https://doi.org/10.1126/science.aaw4741 (2020)

Lois de conservations et décomposition de domaine



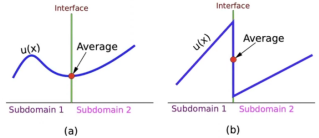


cPINNs[3] : **Interface conditions in the "q^{th}" subdomain.**

$$\mathrm{MSE}_{\mathrm{flux}} = \sum_{\forall q^+} \left(\frac{1}{N_{I_q}} \sum_{i=1}^{N_{I_q}} \left| f_q(u(\mathbf{x}_{I_q}^{(i)})) \cdot \mathbf{n} - f_{q^+}(u(\mathbf{x}_{I_q}^{(i)}) \cdot \mathbf{n}) \right|^2 \right)$$

where f is the flux and $q = 1, 2, \cdots, N_{sd}$.

$$\mathrm{MSE}_{u_{avg}} = \sum_{\forall q^+} \left(\frac{1}{N_{I_q}} \sum_{i=1}^{N_{I_q}} \left| u_{\bar{\Theta}_q}(\mathbf{x}_{I_q}^{(i)}) - \left\{ \left\{ u_{\bar{\Theta}_q}(\mathbf{x}_{I_q}^{(i)}) \right\} \right\} \right|^2 \right)$$



reference : Jagtap A.D., Kharazmi E., Karniadakis G.E., CMAME 365 (2020) 113028.

Pour aller plus loin

- gPINN (gradient enhanced PINN) [4]: on modifie la fonction de coût, les résidus de l'EDP doivent être nuls, et les **gradients des résidus** aussi !
  - EDP: $f(x,t,\partial u/\partial t,...)=0$
  - $$\mathscr{L}_{loss} = \lambda_{data}\sum_i |u(x_i,t_i)-\hat{u}(x_i,t_i)|^2 + \lambda_f \sum_j |f(x_j,t_j,...)|^2 + \lambda_g \sum_j |\frac{\partial f}{\partial t}(x_j,t_j,...)|^2 + \dots$$
  - implanté dans DeepXDE : https://github.com/lululxvi/deepxde
- Residual adaptive refinement : adaptively add more points with large PDE residual (same ref)
- **Neural Operators**, e.g. *DeepONet*, *Low-rank Neural Operator* (LNO), *Graph Neural Operator* (GNO), *Fourier Neural Operators* (FNO) and *Physics Informed Neural Operators* (PINO)
- Neural Tangents handle infinite-width NN, utilise des processus Gaussian
- Solver multi-grille hydride: PINN sur les grandes échelle + MG sur les échelles fines

[4] Yu etal, Comp. Meth. App. Mech and Eng., vol 393, 2022. https://doi.org/10.1016/j.cma.2022.114823

Universal Approximation theorem for Operators / DeepOnet

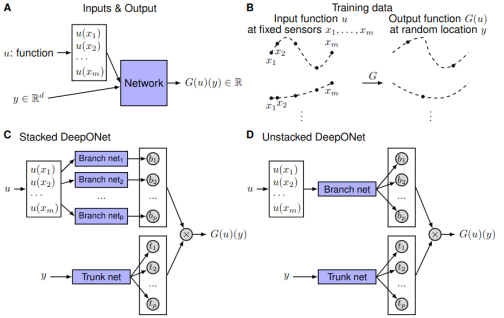
- u est une fonction de la variable x
- $G(u)$ est une fonction de la variable y, image de u par l'opérateur G (à approcher)
- u est supposée *connue / échantilonée* aux points x_i pour $i = 1,..N$
- DeepONet : on cherche à évaluer $G(u)$ au point y sous la forme $G(u)(y) \simeq \sum_{k=1}^{p} b_k t_k + b_0$

ref: https://arxiv.org/pdf/1910.03193.pdf

Theorem 1 (Universal Approximation Theorem for Operator). *Suppose that σ is a continuous non-polynomial function, X is a Banach Space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1,\dots,n$, $k = 1,\dots,p$, $j = 1,\dots,m$, such that*

$$\left| G(u)(y) - \underbrace{\sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon \qquad (1)$$

holds for all $u \in V$ and $y \in K_2$.



Learning Green's function operator

- What is a Green's function ? Let a linear PDE be

$$\mathscr{F}_L(u) = f, \quad \mathbf{x} \in \Omega$$
$$\mathscr{B}_L(u) = g, \quad \mathbf{x} \in \partial\Omega$$

- **Green's function** $\mathscr{G}(\mathbf{x},\mathbf{y})$ is an operator, defined as a particular solution such that

$$\mathscr{F}_L(\mathscr{G}(\mathbf{x},\mathbf{y})) = \delta(\mathbf{x}-\mathbf{y})), \qquad \mathbf{x} \in \Omega$$
$$\mathscr{B}_L(\mathscr{G}(\mathbf{x},\mathbf{y})) = 0, \qquad \mathbf{x} \in \partial\Omega$$

- linear PDE ⇒ superposition theorem, the general solution is

$$u(\mathbf{x}) = \int_\Omega \mathscr{G}(\mathbf{x},\mathbf{y}) f(\mathbf{y})\, d\mathbf{y} + u_{\text{homo}}(\mathbf{x})$$

- Green's function can be seen as the operator that maps the *source* $f(\mathbf{x})$ to the solution $u(\mathbf{x})$
- example: Green's function of Laplacian is $\frac{-1}{4\pi r}$ which is why we can write Coulomb's law for electrical potential : $\phi(\mathbf{r}') = \frac{1}{4\pi\epsilon_0}\int \frac{\rho(\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|} d\mathbf{r}'$
- it is possible to design a NN to learn Green's function of a PDE ⇒ DeepGreen (and extend it to non-linear opertor/PDE through transformations / autoencoder networks) [a]

[a] Ref: DeepGreen: deep learning of Greens functions for nonlinear boundary value problems https://www.nature.com/articles/s41598-021-00773-x, and code https://github.com/sheadan/DeepGreen

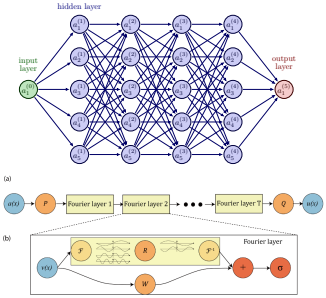
Fourier Neural Operator (FNO) / Physics Informed Neural Operator (PINO) [5]

- équation de Burgers: $\partial_t u + \partial_x(\frac{u^2}{2}) = 0$ et la condition initiale $u(x, t=0) = u_0(x)$ avec $u_0 \in L^2_{per}([0,1], \mathbb{R})$
- Généralisation: et si on entrainait un NN à approximer le mapping $G^\dagger : u_0 \mapsto u(x,T)$ par $G^\dagger_{\boldsymbol{\theta}} : u_0 \mapsto \widehat{u}(x,T)$
- les noeuds du réseau sont plus compliqués qu'avant (!) :

$$v_{l+1}(x) = \sigma(W_l + K_l)v_l(x)$$

où les W_l sont des opérateurs linéaires *pointwise* et les K_l sont des convolutions, i.e. $K_l = \mathscr{F}^{-1} R_l \mathscr{F}$ (convolution faite dans Fourier); mais **pas de différentiation automatique, les dérivées sont faites dans Fourier**

- **FNO** : on entraine avec des couples (u_0, u) connus, obtenu par une méthode tierce (différences finies, ...)
- **PINO** : on entraine directement le réseau avec les contraintes que $\widehat{u}(x,T)$ doit être solution de l'EDP
- modulus ou https://github.com/shawnrosofsky/PINO_Applications



- input layer : $u(x,0)$
- output layer : $\widehat{u}(x,T)$, solution de l'EDP
- phase d'apprentissage, on génère des fonctions conditions initiales $u_0 \sim \mu$ avec $\mu \sim \mathcal{N}(0, 625(-\Delta + 25I)^2)$ (réalisation d'un champ Gaussien aléatoire, i.e. un bruit blanc intégré dans Fourier)

[5] https://arxiv.org/pdf/2010.08895.pdf et https://arxiv.org/abs/2111.03794, Physics-Informed Neural Operator for Learning Partial Differential Equations

Fourier Neural Operator (FNO) / Physics Informed Neural Operator (PINO) [5]

- équation de Burgers: $\partial_t u + \partial_x(\frac{u^2}{2}) = 0$ et la condition initiale $u(x, t=0) = u_0(x)$ avec $u_0 \in L^2_{per}([0,1], \mathbb{R})$
- Généralisation: et si on entrainait un NN à approximer le mapping $G^\dagger : u_0 \mapsto u(x,T)$ par $G^\dagger_{\boldsymbol{\theta}} : u_0 \mapsto \hat{u}(x,T)$
- les noeuds du réseau sont plus compliqués qu'avant (!) :

$$v_{l+1}(x) = \sigma(W_l + K_l)v_l(x)$$

où les W_l sont des opérateurs linéaires *pointwise* et les K_l sont des convolutions, i.e. $K_l = \mathscr{F}^{-1} R_l \mathscr{F}$ (convolution faite dans Fourier); mais **pas de différentiation automatique, les dérivées sont faites dans Fourier**

- **FNO** : on entraine avec des couples (u_0, u) connus, obtenu par une méthode tierce (différences finies, ...)
- **PINO** : on entraine directement le réseau avec les contraintes que $\hat{u}(x,T)$ doit être solution de l'EDP
- modulus ou https://github.com/shawnrosofsky/PINO_Applications

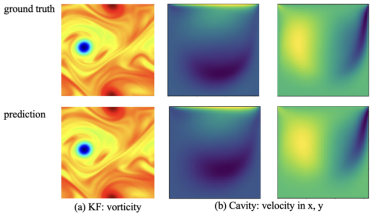
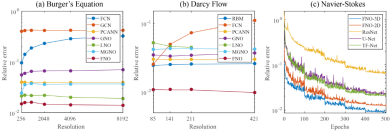


Figure 4: PINO on Kolmogorov flow (left) and Lid-cavity flow (right)



Left: benchmarks on Burgers equation; **Mid:** benchmarks on Darcy Flow for different resolutions; **Right:** the learning curves on Navier-Stokes $\nu = 1e{-}3$ with different benchmarks. Train and test on the same resolution.
For acronyms, see Section 5; details in Tables 1, 3, 4.

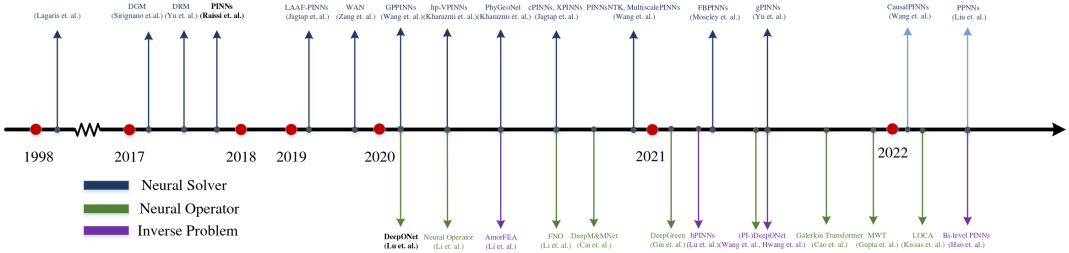
[5] https://arxiv.org/pdf/2010.08895.pdf et https://arxiv.org/abs/2111.03794, Physics-Informed Neural Operator for Learning Partial Differential Equations

Side note on Gaussian Random Fields

- Definition: $\mathbf{x} \mapsto G(\mathbf{x})$, with $\mathbf{x} \in \mathbb{R}^d$ is a Gaussian Random Field (GRF), iff
  - $G(\mathbf{x})$ is a Gaussian random variable, defined by its mean $\mu(\mathbf{x})$ and covariance $c(\mathbf{x_1}, \mathbf{x_2})$
  - it is stationnary (translation invariant), if $\mu(\mathbf{x}) = \mu$ and $c(\mathbf{x_1}, \mathbf{x_2}) = c(\mathbf{x_1} - \mathbf{x_2})$
- in practice stationnary GRF can be generated using FFT, given a known correlation function c, and a white noise $W(\mathbf{x})$, then $G(\mathbf{x}) = \mathscr{F}^{-1}(\sqrt{|\mathscr{F}(c)|}\mathscr{F}(W))$

PINN / Neural operator mini-biblio

- l'article qui présente `modulus` : https://arxiv.org/pdf/2012.07938.pdf
- un article de revue récent:
  Scientific Machine Learning Through Physics-Informed Neural Networks: Where we are and Whats Next, Cuomo et al, Journal of Scientific Computing vol. 92, (2022)
- un autre https://arxiv.org/pdf/2211.08064.pdf, Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications



PINN and conservation laws

- Compressible fluid dynamics :
  $\partial U_t + \nabla.f(U) = 0$, where
  $U = [\rho, \rho v_x, \rho v_y, \rho v_z, \rho e]$
- mini biblio
  - Physics-informed neural networks for high-speed flows, Z. Mao et al, Computer Methods in Applied Mechanics and Engineering, vol 360 (2020); https://doi.org/10.1016/j.cma.2019.112789
  - Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, A.D. Jagtap et al, Computer Methods in Applied Mechanics and Engineering, vol 365 (2020); https://doi.org/10.1016/j.cma.2020.113028
  - Thermodynamically consistent physics-informed neural networks for hyperbolic systems, R. G. Patel et al; https://arxiv.org/pdf/2012.05343.pdf