Machine learning for inverse problems

Mark Asch

4th January, 2022

UPJV & ADSIL

Introduction

Direct and inverse problems

$$y = f(x; \theta)$$

Direct given θ, compute y *(easy)*

Inverse given y, compute θ (hard)

where

- f is an operator/equation/system
- x is the independent variable
- θ is the parameter/feature
- y is the measurement/dependent variable

Direct and inverse problems

$$y = f(x; \theta)$$

Direct given θ, compute y *(easy)*

Inverse given y, compute θ (hard)

where

- f is an operator/equation/system
- x is the independent variable
- θ is the parameter/feature
- y is the measurement/dependent variable

Direct and inverse problems

$$y = f(x; \theta)$$

Direct given θ, compute y *(easy)*

Inverse given y, compute θ (hard)

where

- f is an operator/equation/system
- x is the independent variable
- θ is the parameter/feature
- y is the measurement/dependent variable

Direct and inverse problems

$$y = f(x; \theta)$$

Direct given θ, compute y *(easy)*

Inverse given y, compute θ (hard)

where

- f is an operator/equation/system
- x is the independent variable
- θ is the parameter/feature
- y is the measurement/dependent variable

Uncertainty is everywhere

- the observations are uncertain,

$$y = f(x;\theta) + \xi,$$

  where ξ is a random variable, or more generaly a stochastic process...

- the model f is uncertain:
  - unknown unknowns,
  - uncertain material properties
  - uncertain geometry, boundary conditions, input signals, etc.

Uncertainty is everywhere

- the observations are uncertain,

$$y = f(x;\theta) + \xi,$$

  where ξ is a random variable, or more generaly a stochastic process...
- the model f is uncertain:
    - unknown unknowns,
    - uncertain material properties
    - uncertain geometry, boundary conditions, input signals, etc.

Uncertainty is everywhere

- the observations are uncertain,

$$y = f(x;\theta) + \xi,$$

  where ξ is a random variable, or more generaly a stochastic process...
- the model f is uncertain:
  - unknown unknowns,
  - uncertain material properties
  - uncertain geometry, boundary conditions, input signals, etc.

Uncertainty is everywhere

- the observations are uncertain,

$$y = f(x;\theta) + \xi,$$

  where ξ is a random variable, or more generaly a stochastic process...
- the model f is uncertain:
  - unknown unknowns,
  - uncertain material properties
  - uncertain geometry, boundary conditions, input signals, etc.

Uncertainty is everywhere

- the observations are uncertain,

$$y = f(x;\theta) + \xi,$$

  where ξ is a random variable, or more generaly a stochastic process...
- the model f is uncertain:
    - unknown unknowns,
    - uncertain material properties
    - uncertain geometry, boundary conditions, input signals, etc.

Bayesian inversion

Bayes' theorem:

$$P(\theta \mid y) = \frac{P(y \mid \theta)P(\theta)}{P(y)}$$

- solves the inverse problem, $\theta" = " f^{-1}(y)$
- and provides complete uncertainty quantification!

(Too) high cost

For reasonable accuracy of the posterior, we need a good exploration of the prior and likelihood, which implies a large number of simulations and/or measurements for the evaluation of a very high-dimensional integral...

Bayesian inversion

Bayes' theorem:

$$P(\theta \mid y) = \frac{P(y \mid \theta)P(\theta)}{P(y)}$$

- solves the inverse problem, θ " $=$ " $f^{-1}(y)$
- and provides complete uncertainty quantification!

(Too) high cost

For reasonable accuracy of the posterior, we need a good exploration of the prior and likelihood, which implies a large number of simulations and/or measurements for the evaluation of a very high-dimensional integral...

Bayesian inversion

Bayes' theorem:

$$P(\theta \mid y) = \frac{P(y \mid \theta)P(\theta)}{P(y)}$$

- solves the inverse problem, θ " $=$ " $f^{-1}(y)$
- and provides complete uncertainty quantification!

(Too) high cost

For reasonable accuracy of the posterior, we need a good exploration of the prior and likelihood, which implies a large number of simulations and/or measurements for the evaluation of a very high-dimensional integral...

Bayesian inversion

Bayes' theorem:

$$P(\theta \mid y) = \frac{P(y \mid \theta)P(\theta)}{P(y)}$$

- solves the inverse problem, θ " $=$ " $f^{-1}(y)$
- and provides complete uncertainty quantification!

(Too) high cost

For reasonable accuracy of the posterior, we need a good exploration of the prior and likelihood, which implies a large number of simulations and/or measurements for the evaluation of a very high-dimensional integral...

Classical inversion

- use the model to generate measurements, $\hat{y} = f(x, \hat{\theta})$
- define a suitably regularized cost function, $F(\theta) = g\left(\|\hat{y} - y\|\right)$ with a function-space norm
- minimize the cost function

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, F(\hat{\theta}),$$

  subject to (PDE) constraint.

Difficulties

ill-posed problem with local mimima, requires computation of a gradient, needs regularization, does not deal well with noise and uncertainty

Classical inversion

- use the model to generate measurements, $\hat{y} = f(x, \hat{\theta})$
- define a suitably regularized cost function, $F(\theta) = g\left(\|\hat{y} - y\|\right)$ with a function-space norm
- minimize the cost function

$$\theta^* = \operatorname*{argmin}_{\theta} F(\hat{\theta}),$$

subject to (PDE) constraint.

Difficulties

ill-posed problem with local mimima, requires computation of a gradient, needs regularization, does not deal well with noise and uncertainty

Classical inversion

- use the model to generate measurements, $\hat{y} = f(x, \hat{\theta})$
- define a suitably regularized cost function, $F(\theta) = g\left(\|\hat{y} - y\|\right)$ with a function-space norm
- minimize the cost function

$$\theta^* = \operatorname*{argmin}_{\theta} F(\hat{\theta}),$$

subject to (PDE) constraint.

Difficulties

ill-posed problem with local mimima, requires computation of a gradient, needs regularization, does not deal well with noise and uncertainty

Classical inversion

- use the model to generate measurements, $\hat{y} = f(x, \hat{\theta})$
- define a suitably regularized cost function,
  $F(\theta) = g\left(\|\hat{y} - y\|\right)$ with a function-space norm
- minimize the cost function

$$\theta^* = \operatorname*{argmin}_{\theta} F(\hat{\theta}),$$

subject to (PDE) constraint.

Difficulties

ill-posed problem with local mimima, requires computation of a gradient, needs regularization, does not deal well with noise and uncertainty

Solutions exist

Bayesian

MCMC methods, Bayesian optimization

Classical

adjoint-state methods, quasi-Newton, regularization techniques

- But even with these, the inverse problem is hard.

Solutions exist

Bayesian

MCMC methods, Bayesian optimization

Classical

adjoint-state methods, quasi-Newton, regularization techniques

- But even with these, the inverse problem is hard.

Solutions exist

Bayesian

MCMC methods, Bayesian optimization

Classical

adjoint-state methods, quasi-Newton, regularization techniques

- But even with these, the inverse problem is hard.

Machine Learning

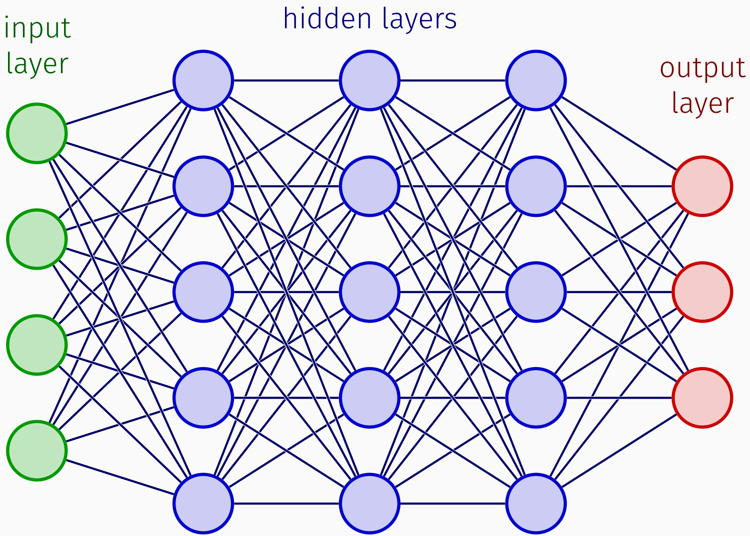
2 key properties

- Universal approximation
- Automatic differentiation

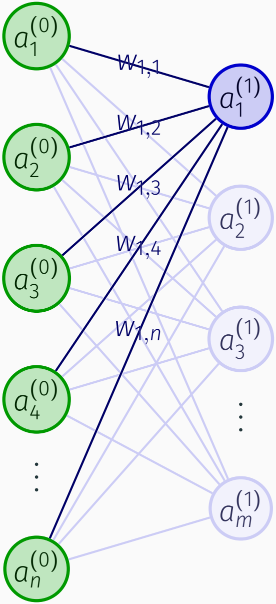
2 key properties

- Universal approximation
- Automatic differentiation

FCNN - architecture



NN - neuron activation



$$= \sigma\left(w_{1,0}a_0^{(0)} + w_{1,1}a_1^{(0)} + \ldots + w_{1,n}a_n^{(0)} + b_1^{(0)}\right)$$

$$= \sigma\left(\sum_{i=1}^{n} w_{1,i}a_i^{(0)} + b_1^{(0)}\right)$$

$$\begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,0} & \ldots & w_{1,n} \\ w_{2,0} & \ldots & w_{2,n} \\ \vdots & \ddots & \vdots \\ w_{m,0} & \ldots & w_{m,n} \end{pmatrix}\begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{pmatrix}\right]$$

$$\mathbf{a}^{(1)} = \sigma\left(\mathbf{W}^{(0)}\mathbf{a}^{(0)} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{y} = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right), \quad \text{for a single hidden layer.}$$

Universal approximation: functions

Theorem (Cybenko 1989)

If σ is any continuous sigmoidal function, then finite sums $G(x) = \sum_{j=1}^{N} \alpha_j \sigma\left(y_j \cdot x + \theta_j\right)$ are dense in $C(I_d)$.

Theorem (Pinkus 1999)

Let $\mathrm{m}_i \in \mathbb{Z}^d$, $i = 1, \ldots, s$, and set $m = \max_i \left|\mathrm{m}^i\right|$. Suppose that $\sigma \in C^m(\mathbb{R})$, not polynomial. Then the space of single hidden layer neural nets,

$$\mathcal{M}(\sigma) = \operatorname{span}\left\{\sigma(\mathrm{w} \cdot \mathrm{x} + b) \colon \mathrm{w} \in \mathbb{R}^d,\, b \in \mathbb{R}\right\},$$

is dense in $C^{\mathrm{m}^1,\ldots,\mathrm{m}^s}(\mathbb{R}^d) \doteq \cap_{i=1}^{s} C^{\mathrm{m}^i}(\mathbb{R}^d)$.

Universal approximation: functions

Theorem (Cybenko 1989)

If σ is any continuous sigmoidal function, then finite sums $G(x) = \sum_{j=1}^{N} \alpha_j \sigma\left(y_j \cdot x + \theta_j\right)$ are dense in $C(I_d)$.

Theorem (Pinkus 1999)

Let $\mathbf{m}_i \in \mathbb{Z}^d$, $i = 1, \ldots, s$, and set $m = \max_i \left|\mathbf{m}^i\right|$. Suppose that $\sigma \in C^m(\mathbb{R})$, not polynomial. Then the space of single hidden layer neural nets,

$$\mathcal{M}(\sigma) = \operatorname{span}\left\{\sigma(\mathbf{w} \cdot \mathbf{x} + b) \colon \mathbf{w} \in \mathbb{R}^d,\ b \in \mathbb{R}\right\},$$

is dense in $C^{\mathbf{m}^1, \ldots, \mathbf{m}^s}(\mathbb{R}^d) \doteq \cap_{i=1}^{s} C^{\mathbf{m}^i}(\mathbb{R}^d)$.

Universal approximation: operators

Theorem (Chen, Chen 1995)

Suppose σ is continuous, non-polynomial, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are compact sets, V is compact in $C(K_1)$, G is continuous operator from V into $C(K_2)$. Then, for any $\epsilon > 0$, there exist positive integers m, n, p, constants c_k^i, $\xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, such that

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

for all $u \in V$, $y \in K_2$.

Automatic differentiation

- Training/learning = finding the coefficients, $w_{i,j}$, that minimize the training loss.
- This minimization is done by a stochastic gradient method.
- The gradient is computed by AD, where the ouput is differentiated with respect to the weights, based on Leibniz's rule.

Fact (Darve, 2021)

Reverse-mode automatic differentiation is mathematically equivalent to the adjoint-state method, and the gradients obtained are the same.

Automatic differentiation

- Training/learning = finding the coefficients, $w_{i,j}$, that minimize the training loss.
- This minimization is done by a stochastic gradient method.
- The gradient is computed by AD, where the ouput is differentiated with respect to the weights, based on Leibniz's rule.

Fact (Darve, 2021)

Reverse-mode automatic differentiation is mathematically equivalent to the adjoint-state method, and the gradients obtained are the same.

Automatic differentiation

- Training/learning = finding the coefficients, $w_{i,j}$, that minimize the training loss.
- This minimization is done by a stochastic gradient method.
- The gradient is computed by AD, where the ouput is differentiated with respect to the weights, based on Leibniz's rule.

Fact (Darve, 2021)

Reverse-mode automatic differentiation is mathematically equivalent to the adjoint-state method, and the gradients obtained are the same.

Automatic differentiation

- Training/learning = finding the coefficients, $w_{i,j}$, that minimize the training loss.
- This minimization is done by a stochastic gradient method.
- The gradient is computed by AD, where the ouput is differentiated with respect to the weights, based on Leibniz's rule.

Fact (Darve, 2021)

Reverse-mode automatic differentiation is mathematically equivalent to the adjoint-state method, and the gradients obtained are the same.

4+1 approaches

- surrogate models
- physics constrained neural networks
- operator learning
- automatic differentiation for gradient computations only
- combinations of the above

4+1 approaches

- surrogate models (data-driven)
- physics constrained neural networks
- operator learning
- automatic differentiation for gradient computations only
- combinations of the above

4+1 approaches

- surrogate models (data-driven)
- physics constrained neural networks

- operator learning
- automatic differentiation for gradient computations only
- combinations of the above

4+1 approaches

- surrogate models (data-driven)
- physics constrained neural networks (data + physics-driven)
- operator learning
- automatic differentiation for gradient computations only
- combinations of the above

4+1 approaches

- surrogate models (data-driven)
- physics constrained neural networks (data + physics-driven)
- operator learning
- automatic differentiation for gradient computations only
- combinations of the above

4+1 approaches

- surrogate models (data-driven)
- physics constrained neural networks (data + physics-driven)
- operator learning (data-driven + system identification)
- automatic differentiation for gradient computations only
- combinations of the above

4+1 approaches

- surrogate models (data-driven)
- physics constrained neural networks (data + physics-driven)
- operator learning (data-driven + system identification)
- automatic differentiation for gradient computations only
- combinations of the above

4+1 approaches

- surrogate models (data-driven)
- physics constrained neural networks (data + physics-driven)
- operator learning (data-driven + system identification)
- automatic differentiation for gradient computations only
- combinations of the above

Software

- More and more software is becoming available...
- "Commercial":
  - MODULUS[1] by NVIDIA
  - DEEPXDE by Karniadakis (Brown, U. Penn.)[2]
- Academic:
  - PINN and it's numerous extensions/improvements (behind DEEPXDE and MODULUS)
  - DeepONet (behind DEEPXDE)
  - Fourier Neural Operators (FNO)
  - ADCME framework (AD)
  - many, many others...

[1] https://developer.nvidia.com/modulus
[2] https://github.com/lululxvi/deepxde

Machine Learning

Surrogate Models (SUMO)

Elementary, data-driven models

Definition

Surrogate models, also known as response surfaces, black-box models, metamodels, or emulators, are simplified approximations of more complex, higher order models. These models are used to map input-data to output-data, when the actual relationship between the two is unknown or computationally too expensive to evaluate.

- ML and regression techniques commonly used:
    - random forest,
    - SVM,
    - BNs and NNs.

Elementary, data-driven models

Definition

Surrogate models, also known as response surfaces, black-box models, metamodels, or emulators, are simplified approximations of more complex, higher order models. These models are used to map input-data to output-data, when the actual relationship between the two is unknown or computationally too expensive to evaluate.

- ML and regression techniques commonly used:
    - random forest,
    - SVM,
    - BNs and NNs.

Elementary, data-driven models

Definition

Surrogate models, also known as response surfaces, black-box models, metamodels, or emulators, are simplified approximations of more complex, higher order models. These models are used to map input-data to output-data, when the actual relationship between the two is unknown or computationally too expensive to evaluate.

- ML and regression techniques commonly used:
  - random forest,
  - SVM,
  - BNs and NNs.

Elementary, data-driven models

Definition

Surrogate models, also known as response surfaces, black-box models, metamodels, or emulators, are simplified approximations of more complex, higher order models. These models are used to map input-data to output-data, when the actual relationship between the two is unknown or computationally too expensive to evaluate.

- ML and regression techniques commonly used:
    - random forest,
    - SVM,
    - BNs and NNs.

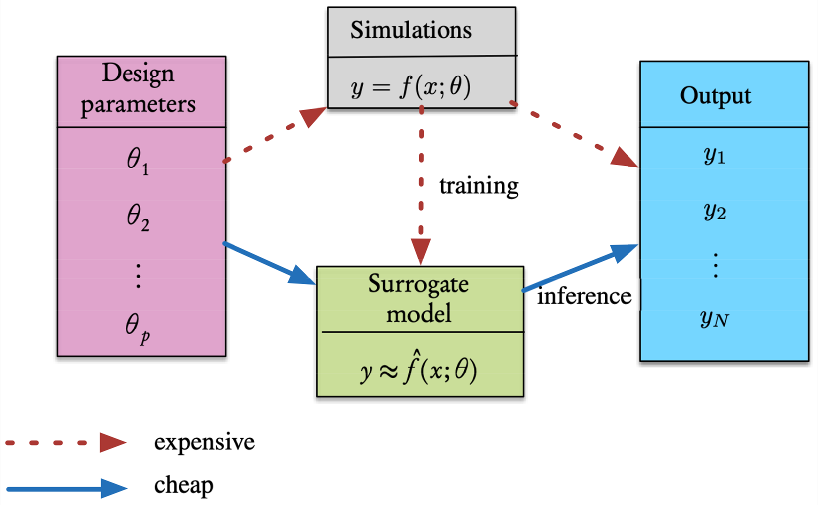
Elementary, data-driven models

Definition

Surrogate models, also known as response surfaces, black-box models, metamodels, or emulators, are simplified approximations of more complex, higher order models. These models are used to map input-data to output-data, when the actual relationship between the two is unknown or computationally too expensive to evaluate.

- ML and regression techniques commonly used:
    - random forest,
    - SVM,
    - BNs and NNs.

SUMO flowchart



Machine Learning

PINN et cie.

2 approaches

- learn the solution
- learn the operator

2 approaches

- learn the solution
- learn the operator

Learn the Solution: PINN

IDEA:

replace traditional numerical discretization methods—FDM, FEM—by a neural network that *learns* an approximate solution.

HOW?

constrain the NN to minimize an *augmented loss* that includes the PDE, boundary and initial conditions, in addition to the usual loss function over the NN parameters (weights and biases).

Learn the Solution: PINN

IDEA:

replace traditional numerical discretization methods—FDM, FEM—by a neural network that *learns* an approximate solution.

HOW?

constrain the NN to minimize an *augmented loss* that includes the PDE, boundary and initial conditions, in addition to the usual loss function over the NN parameters (weights and biases).

PINN: formulation

Let $F = 0$ be the PDE, $B = 0$ the boundary conditions,
then the PINN loss is

$$\mathcal{L}(\theta; \mathcal{T}) = w_f \mathcal{L}_f(\theta; \mathcal{T}_f) + w_b \mathcal{L}_b(\theta, \mathcal{T}_b),$$

PINN: formulation

Let $F = 0$ be the PDE, $B = 0$ the boundary conditions, $I = 0$ the inversion conditions, then the PINN loss is

$$\mathcal{L}(\theta, \lambda; \mathcal{T}) = w_f \mathcal{L}_f(\theta, \lambda; \mathcal{T}_f) + w_b \mathcal{L}_b(\theta\lambda; \mathcal{T}_b) + w_i \mathcal{L}_i(\theta, \lambda; \mathcal{T}_i)$$

- where

$$\mathcal{L}_f(\theta; \mathcal{T}_f) = \|F(\hat{u}, x, \lambda)\|_2^2$$
$$\mathcal{L}_b(\theta; \mathcal{T}_b) = \|B(\hat{u}, x)\|_2^2$$
$$\mathcal{L}_i(\theta, \lambda, \mathcal{T}_i) = \frac{1}{|\mathcal{T}_i|} \sum_{x \in \mathcal{T}_i} \|I(\hat{u}, x)\|_2^2$$

  and x are the training points, $\hat{u}$ the approximate solution, λ the inversion coefficients

- solution, $\{\theta^*, \lambda^*\} = \operatorname{argmin}_{\theta,\lambda} \mathcal{L}(\theta, \lambda; \mathcal{T})$

PINN: error analysis

- error analysis can been derived[3][4], in terms of
  - optimization error $e_o = \|\hat{u}_{\mathcal{T}} - u_{\mathcal{T}}\|$
  - generalization error $e_g = \|u_{\mathcal{T}} - u_{\mathcal{F}}\|$
  - approximation error $e_a = \|u_{\mathcal{F}} - u\|$
- then

$$e \doteq \|\hat{u}_{\mathcal{T}} - u\| \leq e_o + e_g + e_a$$

[3] Lu, Karniadakis, SIAM Review, 2021.
[4] Mishra, Molinaro; arXiv:2006.16144v2.

PINN: error analysis

- error analysis can been derived[3,4], in terms of
    - optimization error $e_o = \|\hat{u}_{\mathcal{T}} - u_{\mathcal{T}}\|$
    - generalization error $e_g = \|u_{\mathcal{T}} - u_{\mathcal{F}}\|$
    - approximation error $e_a = \|u_{\mathcal{F}} - u\|$
- then

$$e \doteq \|\hat{u}_{\mathcal{T}} - u\| \le e_o + e_g + e_a$$

[3] Lu, Karniadakis, SIAM Review, 2021.
[4] Mishra, Molinaro; arXiv:2006.16144v2.

PINN: error analysis

- error analysis can been derived[3,4], in terms of
  - optimization error $e_o = \|\hat{u}_{\mathcal{T}} - u_{\mathcal{T}}\|$
  - generalization error $e_g = \|u_{\mathcal{T}} - u_{\mathcal{F}}\|$
  - approximation error $e_a = \|u_{\mathcal{F}} - u\|$
- then

$$e \doteq \|\hat{u}_{\mathcal{T}} - u\| \leq e_o + e_g + e_a$$

[3] Lu, Karniadakis, SIAM Review, 2021.
[4] Mishra, Molinaro; arXiv:2006.16144v2.

PINN: error analysis

- error analysis can been derived[3,4], in terms of
  - optimization error $e_o = \|\hat{u}_{\mathcal{T}} - u_{\mathcal{T}}\|$
  - generalization error $e_g = \|u_{\mathcal{T}} - u_{\mathcal{F}}\|$
  - approximation error $e_a = \|u_{\mathcal{F}} - u\|$
- then

$$e \doteq \|\hat{u}_{\mathcal{T}} - u\| \leq e_o + e_g + e_a$$

[3] Lu, Karniadakis, SIAM Review, 2021.
[4] Mishra, Molinaro; arXiv:2006.16144v2.

PINN: error analysis

- error analysis can been derived[3,4], in terms of
  - optimization error $e_o = \|\hat{u}_{\mathcal{T}} - u_{\mathcal{T}}\|$
  - generalization error $e_g = \|u_{\mathcal{T}} - u_{\mathcal{F}}\|$
  - approximation error $e_a = \|u_{\mathcal{F}} - u\|$
- then

$$e \doteq \|\hat{u}_{\mathcal{T}} - u\| \leq e_o + e_g + e_a$$

[3] Lu, Karniadakis, SIAM Review, 2021.
[4] Mishra, Molinaro; arXiv:2006.16144v2.

Example: the heat equation

IBVP for heat equation

Compute $u(\mathsf{x},t)\colon \Omega \times [0,T] \to \mathbb{R}$ such that

$$
\begin{aligned}
\frac{\partial u(\mathsf{x},t)}{\partial t} - \nabla\cdot(\lambda(x)\nabla u(\mathsf{x},t)) = f(\mathsf{x},t) \quad &\text{in } \times (0,\mathrm{T}), \qquad (1)\\
u(\mathsf{x},t) = g_D(\mathsf{x},t) \quad &\text{on } \partial_{\mathrm{D}} \times (0,\mathrm{T}),\\
-\lambda(x)\nabla u(\mathsf{x},t)\cdot\mathsf{n} = g_R(\mathsf{x},t) \quad &\text{on } \partial_{\mathrm{R}} \times (0,\mathrm{T}),\\
u(\mathsf{x},0) = u_0(\mathsf{x}) \quad &\text{for } \mathrm{x} \in .
\end{aligned}
$$

Note that $\lambda(x)$ is, in general, a tensor (matrix) with elements λ_{ij}.

- Direct problem: given λ, compute u.
- Inverse problem: given u, compute λ.

Example: the heat equation

IBVP for heat equation

Compute $u(\mathbf{x}, t) \colon \Omega \times [0, T] \to \mathbb{R}$ such that

$$\frac{\partial u(\mathbf{x}, t)}{\partial t} - \nabla \cdot (\lambda(x) \nabla u(\mathbf{x}, t)) = f(\mathbf{x}, t) \quad \text{in } \times (0, \mathrm{T}), \tag{1}$$

$$u(\mathbf{x}, t) = g_D(\mathbf{x}, t) \quad \text{on } \partial_{\mathrm{D}} \times (0, \mathrm{T}),$$

$$-\lambda(x) \nabla u(\mathbf{x}, t) \cdot \mathbf{n} = g_R(\mathbf{x}, t) \quad \text{on } \partial_{\mathrm{R}} \times (0, \mathrm{T}),$$

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad \text{for } \mathrm{x} \in .$$

Note that $\lambda(x)$ is, in general, a tensor (matrix) with elements λ_{ij}.

- Direct problem: given λ, compute u.
- Inverse problem: given u, compute λ.

Example: the heat equation

IBVP for heat equation

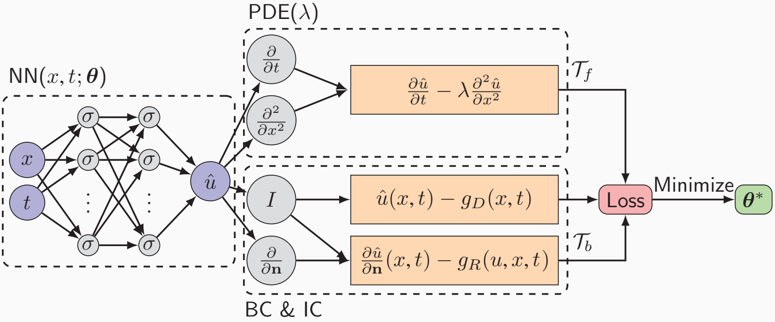
Compute $u(\mathbf{x}, t) \colon \Omega \times [0, T] \to \mathbb{R}$ such that

$$
\begin{aligned}
\frac{\partial u(\mathbf{x}, t)}{\partial t} - \nabla \cdot (\lambda(x) \nabla u(\mathbf{x}, t)) = f(\mathbf{x}, t) \quad & \text{in } \times (0, \mathrm{T}), \\
u(\mathbf{x}, t) = g_D(\mathbf{x}, t) \quad & \text{on } \partial_{\mathrm{D}} \times (0, \mathrm{T}), \\
-\lambda(x) \nabla u(\mathbf{x}, t) \cdot \mathbf{n} = g_R(\mathbf{x}, t) \quad & \text{on } \partial_{\mathrm{R}} \times (0, \mathrm{T}), \\
u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad & \text{for } \mathrm{x} \in .
\end{aligned}
\tag{1}
$$

Note that $\lambda(x)$ is, in general, a tensor (matrix) with elements λ_{ij}.

- Direct problem: given λ, compute u.
- Inverse problem: given u, compute λ.

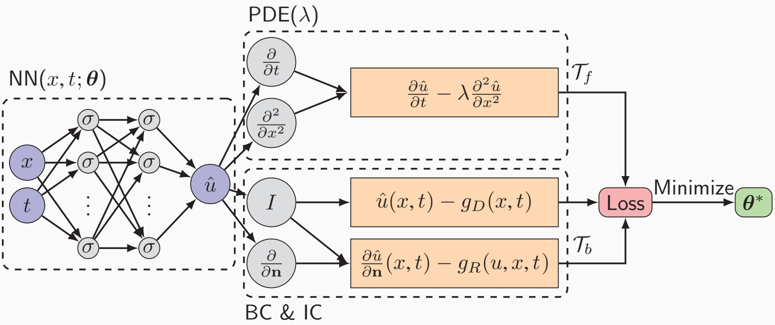
PINN for the heat equation



[Credit: Lu, Karniadakis, SIAM Review, 2021]

- use FCNN to approximate *u* at the selected points *x*, with training data at residual points $\mathcal{T}_f$ and $\mathcal{T}_b$
- use AD to compute derivatives for the PDE and the boundary/initial conditions
- minimize the augmented, weighted loss function

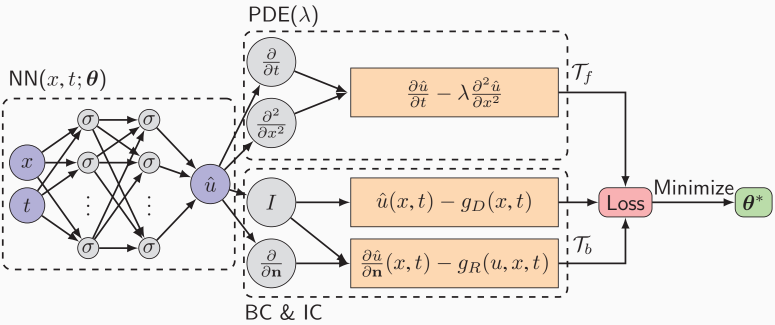
PINN for the heat equation



[Credit: Lu, Karniadakis, SIAM Review, 2021]

- use FCNN to approximate u at the selected points x, with training data at residual points $\mathcal{T}_f$ and $\mathcal{T}_b$
- use AD to compute derivatives for the PDE and the boundary/initial conditions
- minimize the augmented, weighted loss function

PINN for the heat equation



[Credit: Lu, Karniadakis, SIAM Review, 2021]

- use FCNN to approximate u at the selected points x, with training data at residual points $\mathcal{T}_f$ and $\mathcal{T}_b$
- use AD to compute derivatives for the PDE and the boundary/initial conditions
- minimize the augmented, weighted loss function

PINN for inverse problems

- NO modification of the NN
- just augment the parameter vector in the loss function to include the sought-for coefficients, λ, by including a supplementary loss, $\mathcal{L}_i(\theta, \lambda, \mathcal{T}_i)$
- that's it, folks...

PINN for inverse problems

- NO modification of the NN
- just augment the parameter vector in the loss function to include the sought-for coefficients, λ, by including a supplementary loss, $\mathcal{L}_i(\theta, \lambda, \mathcal{T}_i)$
- that's it, folks...

PINN for inverse problems

- NO modification of the NN
- just augment the parameter vector in the loss function to include the sought-for coefficients, λ, by including a supplementary loss, $\mathcal{L}_i(\theta, \lambda, \mathcal{T}_i)$
- that's it, folks...

PINN pros

- mesh-free (only requires residual points where the solution is sought)
- strong (differential) form avoids discretization, stability, numerical integration errors
- leverages AD that is much better than other differentiation methods, especially in higher dimensions
- can deal with noisy/uncertain data
- can use mini-batch techniques for better convergence, especially in inverse problems
- can achieve incredible speed-ups once trained, for subsequent evaluations - order 10^3 to 10^4

PINN pros

- mesh-free (only requires residual points where the solution is sought)
- strong (differential) form avoids discretization, stability, numerical integration errors
- leverages AD that is much better than other differentiation methods, especially in higher dimensions
- can deal with noisy/uncertain data
- can use mini-batch techniques for better convergence, especially in inverse problems
- can achieve incredible speed-ups once trained, for subsequent evaluations - order 10^3 to 10^4

PINN pros

- mesh-free (only requires residual points where the solution is sought)
- strong (differential) form avoids discretization, stability, numerical integration errors
- leverages AD that is much better than other differentiation methods, especially in higher dimensions
- can deal with noisy/uncertain data
- can use mini-batch techniques for better convergence, especially in inverse problems
- can achieve incredible speed-ups once trained, for subsequent evaluations - order 10^3 to 10^4

PINN pros

- mesh-free (only requires residual points where the solution is sought)
- strong (differential) form avoids discretization, stability, numerical integration errors
- leverages AD that is much better than other differentiation methods, especially in higher dimensions
- can deal with noisy/uncertain data
- can use mini-batch techniques for better convergence, especially in inverse problems
- can achieve incredible speed-ups once trained, for subsequent evaluations - order 10^3 to 10^4

PINN pros

- mesh-free (only requires residual points where the solution is sought)
- strong (differential) form avoids discretization, stability, numerical integration errors
- leverages AD that is much better than other differentiation methods, especially in higher dimensions
- can deal with noisy/uncertain data
- can use mini-batch techniques for better convergence, especially in inverse problems
- can achieve incredible speed-ups once trained, for subsequent evaluations - order 10^3 to 10^4

PINN pros

- mesh-free (only requires residual points where the solution is sought)
- strong (differential) form avoids discretization, stability, numerical integration errors
- leverages AD that is much better than other differentiation methods, especially in higher dimensions
- can deal with noisy/uncertain data
- can use mini-batch techniques for better convergence, especially in inverse problems
- can achieve incredible speed-ups once trained, for subsequent evaluations - order 10^3 to 10^4

PINN cons

- terrible optimization problem... though new solutions appear, almost daily—see arXiv
- network architecture/size is very problem dependent
- convergence sensitive to NN initialization, requiring some sort of CV or at least a batch of random repeats
- requires hyperparameter tuning: size, learning rate, number of residual points (no free lunch...)

PINN cons

- terrible optimization problem... though new solutions appear, almost daily—see arXiv
- network architecture/size is very problem dependent
- convergence sensitive to NN initialization, requiring some sort of CV or at least a batch of random repeats
- requires hyperparameter tuning: size, learning rate, number of residual points (no free lunch...)

PINN cons

- terrible optimization problem... though new solutions appear, almost daily—see arXiv
- network architecture/size is very problem dependent
- convergence sensitive to NN initialization, requiring some sort of CV or at least a batch of random repeats
- requires hyperparameter tuning: size, learning rate, number of residual points (no free lunch...)

PINN cons

- terrible optimization problem... though new solutions appear, almost daily—see arXiv
- network architecture/size is very problem dependent
- convergence sensitive to NN initialization, requiring some sort of CV or at least a batch of random repeats
- requires hyperparameter tuning: size, learning rate, number of residual points (no free lunch...)

PINN: conclusions

- recommended for simple PDEs, in geometrically simple domains
- useful for initial, feasibility studies, especially for inverse problems
- must perform extensive hyperparameter tuning

PINN: conclusions

- recommended for simple PDEs, in geometrically simple domains
- useful for initial, feasibility studies, especially for inverse problems
- must perform extensive hyperparameter tuning

PINN: conclusions

- recommended for simple PDEs, in geometrically simple domains
- useful for initial, feasibility studies, especially for inverse problems
- must perform extensive hyperparameter tuning

Learn the Operator: operator nets

- Use the Universal Operator Approximation Theorem...

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

  where G is the solution operator, u is an input function, x_i are "sensor" points, y are random points where we evaluate the output function $G(u)$.

- 2 main contenders:
  - DeepONet
  - Fourier Neural Operators (FNO)
    - – a special case of DeepONet

Learn the Operator: operator nets

- Use the Universal Operator Approximation Theorem...

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

  where G is the solution operator, u is an input function, x_i are "sensor" points, y are random points where we evaluate the output function $G(u)$.
- 2 main contenders:
  - DeepONet
  - Fourier Neural Operators (FNO)
    - – a special case of DeepONet

Learn the Operator: operator nets

- Use the Universal Operator Approximation Theorem…

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

  where G is the solution operator, u is an input function, x_i are "sensor" points, y are random points where we evaluate the output function $G(u)$.

- 2 main contenders:
  - DeepONet
  - Fourier Neural Operators (FNO)
    - – a special case of DeepONet

Learn the Operator: operator nets

- Use the Universal Operator Approximation Theorem...

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

  where G is the solution operator, u is an input function, x_i are "sensor" points, y are random points where we evaluate the output function $G(u)$.
- 2 main contenders:
  - DeepONet
  - Fourier Neural Operators (FNO)
    - - a special case of DeepONet

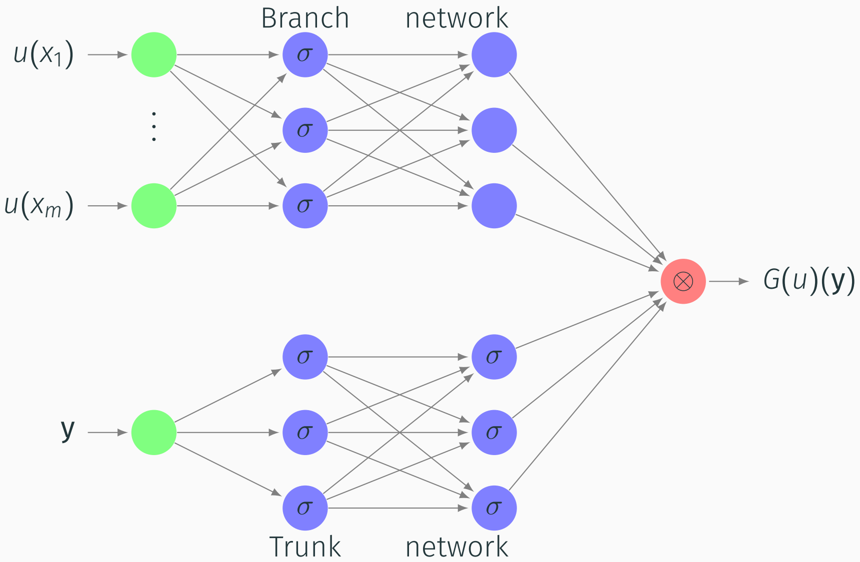
Learn the Operator: operator nets

- Use the Universal Operator Approximation Theorem...

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

  where G is the solution operator, u is an input function, x_i are "sensor" points, y are random points where we evaluate the output function $G(u)$.
- 2 main contenders:
  - DeepONet
  - Fourier Neural Operators (FNO)
    - - a special case of DeepONet

DeepONet architecture



Loss function

- Branch (FCNN, ResNET, CNN, etc.) and trunk networks (FCNN) are merged by an inner product.
- Prediction of a function u evaluated at points $\mathbf{y}$ is then given by

$$G_\theta(u)(y) = \sum_{k=1}^{q} \underbrace{b_k(u(x))}_{\text{branch}} \underbrace{t_k(y)}_{\text{trunk}} + b_0$$

- Training weights and biases, θ, computed by minimizing the loss (mini-batch by Adam, single-batch by L-BFGS)

$$\mathcal{L}_o(\theta) = \frac{1}{NP} \sum_{i=1}^{N} \sum_{j=1}^{P} \left| G_\theta(u^{(i)})(y_j^{(i)}) - G(u^{(i)})(y_j^{(i)}) \right|^2$$

Loss function

- Branch (FCNN, ResNET, CNN, etc.) and trunk networks (FCNN) are merged by an inner product.
- Prediction of a function u evaluated at points $\mathbf{y}$ is then given by

$$G_\theta(u)(y) = \sum_{k=1}^{q} \underbrace{b_k(u(x))}_{\text{branch}} \underbrace{t_k(y)}_{\text{trunk}} + b_0$$

- Training weights and biases, θ, computed by minimizing the loss (mini-batch by Adam, single-batch by L-BFGS)

$$\mathcal{L}_o(\theta) = \frac{1}{NP} \sum_{i=1}^{N} \sum_{j=1}^{P} \left| G_\theta(u^{(i)})(y_j^{(i)}) - G(u^{(i)})(y_j^{(i)}) \right|^2$$

Loss function

- Branch (FCNN, ResNET, CNN, etc.) and trunk networks (FCNN) are merged by an inner product.
- Prediction of a function u evaluated at points $\mathbf{y}$ is then given by

$$G_\theta(u)(y) = \sum_{k=1}^{q} \underbrace{b_k(u(x))}_{\text{branch}} \underbrace{t_k(y)}_{\text{trunk}} + b_0$$

- Training weights and biases, θ, computed by minimizing the loss (mini-batch by Adam, single-batch by L-BFGS)

$$\mathcal{L}_o(\theta) = \frac{1}{NP} \sum_{i=1}^{N} \sum_{j=1}^{P} \left| G_\theta(u^{(i)})(y_j^{(i)}) - G(u^{(i)})(y_j^{(i)}) \right|^2$$

Operator nets: pros and cons

- Pros:
  - relatively fast training (compared to PINN)
  - can overcome the curse of dimensionality (in some cases…)
  - suitable for multiscale and multiphysics problems
- Cons:
  - no guarantee that physics is respected
  - require large training sets of paired input-output observations (expensive!)

DeepONet + PINN = PI-DeepONet

- We can combine the two, to get the best of both worlds

$$\mathcal{L}(\theta) = w_f \mathcal{L}_f(G_\theta(u)(y)) + w_b \mathcal{L}_b(G_\theta(u)(y)) + w_o \mathcal{L}_o(G_\theta(u)(y))$$

- Results:[5]
  - no need for paired input-ouput observations, just samples of the input function and BC/IC (self-supervised learning)
  - respects the physics
  - improved predictive accuracy
  - ideal for parametric PDE studies—optimization, parameter estimation, screening, etc.

[5] Wang, Wang, Bhouri, Perdikaris. arXiv:2103.10974v1, arXiv:2106.05384, arXiv:2110.01654, arXiv:2110.13297

- We can combine the two, to get the best of both worlds

$$\mathcal{L}(\theta) = w_f \mathcal{L}_f(G_\theta(u)(y)) + w_b \mathcal{L}_b(G_\theta(u)(y)) + w_o \mathcal{L}_o(G_\theta(u)(y))$$

- Results:[5]
    - no need for paired input-ouput observations, just samples of the input function and BC/IC (self-supervised learning)
    - respects the physics
    - improved predictive accuracy
    - ideal for parametric PDE studies—optimization, parameter estimation, screening, etc.

[5] Wang, Wang, Bhouri, Perdikaris. arXiv:2103.10974v1, arXiv:2106.05384, arXiv:2110.01654, arXiv:2110.13297

DeepONet + PINN = PI-DeepONet

- We can combine the two, to get the best of both worlds

$$\mathcal{L}(\theta) = w_f \mathcal{L}_f(G_\theta(u)(y)) + w_b \mathcal{L}_b(G_\theta(u)(y)) + w_o \mathcal{L}_o(G_\theta(u)(y))$$

- Results:[5]
    - no need for paired input-ouput observations, just samples of the input function and BC/IC (self-supervised learning)
    - respects the physics
    - improved predictive accuracy
    - ideal for parametric PDE studies—optimization, parameter estimation, screening, etc.

[5] Wang, Wang, Bhouri, Perdikaris. arXiv:2103.10974v1, arXiv:2106.05384, arXiv:2110.01654, arXiv:2110.13297

DeepONet + PINN = PI-DeepONet

- We can combine the two, to get the best of both worlds

$$\mathcal{L}(\theta) = w_f \mathcal{L}_f(G_\theta(u)(y)) + w_b \mathcal{L}_b(G_\theta(u)(y)) + w_o \mathcal{L}_o(G_\theta(u)(y))$$

- Results:[5]
  - no need for paired input-ouput observations, just samples of the input function and BC/IC (self-supervised learning)
  - respects the physics
  - improved predictive accuracy
  - ideal for parametric PDE studies—optimization, parameter estimation, screening, etc.

[5]Wang, Wang, Bhouri, Perdikaris. arXiv:2103.10974v1, arXiv:2106.05384, arXiv:2110.01654, arXiv:2110.13297

DeepONet + PINN = PI-DeepONet

- We can combine the two, to get the best of both worlds

$$\mathcal{L}(\theta) = w_f \mathcal{L}_f(G_\theta(u)(y)) + w_b \mathcal{L}_b(G_\theta(u)(y)) + w_o \mathcal{L}_o(G_\theta(u)(y))$$

- Results:[5]
  - no need for paired input-ouput observations, just samples of the input function and BC/IC (self-supervised learning)
  - respects the physics
  - improved predictive accuracy
  - ideal for parametric PDE studies—optimization, parameter estimation, screening, etc.

[5] Wang, Wang, Bhouri, Perdikaris. arXiv:2103.10974v1, arXiv:2106.05384, arXiv:2110.01654, arXiv:2110.13297

DeepONet + PINN = PI-DeepONet

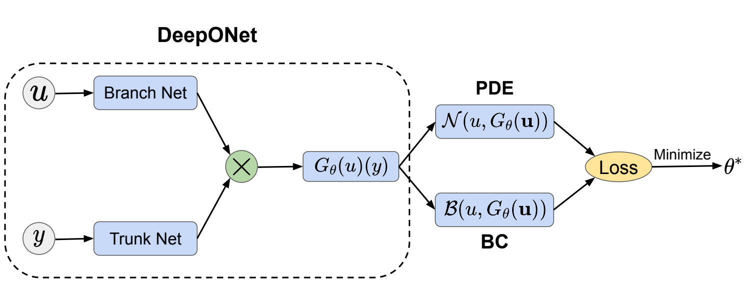
- We can combine the two, to get the best of both worlds

$$\mathcal{L}(\theta) = w_f \mathcal{L}_f(G_\theta(u)(y)) + w_b \mathcal{L}_b(G_\theta(u)(y)) + w_o \mathcal{L}_o(G_\theta(u)(y))$$

- Results:[5]
  - no need for paired input-ouput observations, just samples of the input function and BC/IC (self-supervised learning)
  - respects the physics
  - improved predictive accuracy
  - ideal for parametric PDE studies—optimization, parameter estimation, screening, etc.

[5] Wang, Wang, Bhouri, Perdikaris. arXiv:2103.10974v1, arXiv:2106.05384, arXiv:2110.01654, arXiv:2110.13297

PI-DeepONet



[Credit: Wang, Wang, Perdikaris; arXiv, 2021]

Machine Learning

Automatic Differentiation (AD)

AD

- Used in all (D)NN algorithms to compute the network parameters/coefficients $\boldsymbol{\theta} = \{w, b\}$
- Compute $\boldsymbol{\theta}$ by minimizing a loss function $L(\boldsymbol{\theta})$, based on training pairs, using a stochastic gradient descent method.
- Gradient computed by backpropagation = reverse-mode AD.
- Observation: reverse-mode AD is equivalent to the adjoint-state method, a well-known approach for solving PDE-constrained inverse problems.
- Proposition: use AD to solve inverse probelms
- Pros: AD is robust, scalable, accurate, flexible and efficient (can be parallelized over GPUs).
- Cons: need to incorporate/constrain the respect of the physics...

AD

- Used in all (D)NN algorithms to compute the network parameters/coefficients $\boldsymbol{\theta} = \{w, b\}$
- Compute $\boldsymbol{\theta}$ by minimizing a loss function $L(\boldsymbol{\theta})$, based on training pairs, using a stochastic gradient descent method.
- Gradient computed by backpropagation = reverse-mode AD.
- Observation: reverse-mode AD is equivalent to the adjoint-state method, a well-known approach for solving PDE-constrained inverse problems.
- Proposition: use AD to solve inverse probelms
- Pros: AD is robust, scalable, accurate, flexible and efficient (can be parallelized over GPUs).
- Cons: need to incorporate/constrain the respect of the physics...

AD

- Used in all (D)NN algorithms to compute the network parameters/coefficients $\boldsymbol{\theta} = \{w, b\}$
- Compute $\boldsymbol{\theta}$ by minimizing a loss function $L(\boldsymbol{\theta})$, based on training pairs, using a stochastic gradient descent method.
- Gradient computed by backpropagation = reverse-mode AD.
- Observation: reverse-mode AD is equivalent to the adjoint-state method, a well-known approach for solving PDE-constrained inverse problems.
- Proposition: use AD to solve inverse probelms
- Pros: AD is robust, scalable, accurate, flexible and efficient (can be parallelized over GPUs).
- Cons: need to incorporate/constrain the respect of the physics...

AD

- Used in all (D)NN algorithms to compute the network parameters/coefficients $\boldsymbol{\theta} = \{w, b\}$
- Compute $\boldsymbol{\theta}$ by minimizing a loss function $L(\boldsymbol{\theta})$, based on training pairs, using a stochastic gradient descent method.
- Gradient computed by backpropagation = reverse-mode AD.
- Observation: reverse-mode AD is equivalent to the adjoint-state method, a well-known approach for solving PDE-constrained inverse problems.
- Proposition: use AD to solve inverse probelms
- Pros: AD is robust, scalable, accurate, flexible and efficient (can be parallelized over GPUs).
- Cons: need to incorporate/constrain the respect of the physics...

AD

- Used in all (D)NN algorithms to compute the network parameters/coefficients $\boldsymbol{\theta} = \{w, b\}$
- Compute $\boldsymbol{\theta}$ by minimizing a loss function $L(\boldsymbol{\theta})$, based on training pairs, using a stochastic gradient descent method.
- Gradient computed by backpropagation = reverse-mode AD.
- Observation: reverse-mode AD is equivalent to the adjoint-state method, a well-known approach for solving PDE-constrained inverse problems.
- Proposition: use AD to solve inverse probelms
- Pros: AD is robust, scalable, accurate, flexible and efficient (can be parallelized over GPUs).
- Cons: need to incorporate/constrain the respect of the physics...

AD

- Used in all (D)NN algorithms to compute the network parameters/coefficients $\boldsymbol{\theta} = \{w, b\}$
- Compute $\boldsymbol{\theta}$ by minimizing a loss function $L(\boldsymbol{\theta})$, based on training pairs, using a stochastic gradient descent method.
- Gradient computed by backpropagation = reverse-mode AD.
- Observation: reverse-mode AD is equivalent to the adjoint-state method, a well-known approach for solving PDE-constrained inverse problems.
- Proposition: use AD to solve inverse probelms
- Pros: AD is robust, scalable, accurate, flexible and efficient (can be parallelized over GPUs).
- Cons: need to incorporate/constrain the respect of the physics...

AD

- Used in all (D)NN algorithms to compute the network parameters/coefficients $\boldsymbol{\theta} = \{w, b\}$
- Compute $\boldsymbol{\theta}$ by minimizing a loss function $L(\boldsymbol{\theta})$, based on training pairs, using a stochastic gradient descent method.
- Gradient computed by backpropagation = reverse-mode AD.
- Observation: reverse-mode AD is equivalent to the adjoint-state method, a well-known approach for solving PDE-constrained inverse problems.
- Proposition: use AD to solve inverse probelms
- Pros: AD is robust, scalable, accurate, flexible and efficient (can be parallelized over GPUs).
- Cons: need to incorporate/constrain the respect of the physics...

AD: Physics constrained learning

Problem formulation:

- Given observations/measurements $\mathbf{u}^{\mathrm{obs}} = \{u(x_i)\}_{i\in\mathcal{I}}$ of u at the locations $\mathbf{x} = \{x_i\}$
- Estimate the parameters $\boldsymbol{\theta}$ by minimizing a loss function

$$L(\boldsymbol{\theta}) = \left\| u(\mathbf{x}) - \mathbf{u}^{\mathrm{obs}} \right\|_2^2$$

  subject to $F(u; \theta) = 0$.
- If $\theta = \theta(x)$, then it can be modeled by a NN...

Idea

To incorporate the physics constraint in the minimization problem we use the implicit function theorem and the chain-rule to calculate the gradient of the loss function with respect to all the parameters—inversion and NN

AD: Physics constrained learning

Problem formulation:

- Given observations/measurements $\mathbf{u}^{\mathrm{obs}} = \{u(x_i)\}_{i \in \mathcal{I}}$ of u at the locations $\mathbf{x} = \{x_i\}$
- Estimate the parameters $\boldsymbol{\theta}$ by minimizing a loss function

$$L(\boldsymbol{\theta}) = \left\| u(\mathbf{x}) - \mathbf{u}^{\mathrm{obs}} \right\|_2^2$$

  subject to $F(u; \theta) = 0$.
- If $\theta = \theta(x)$, then it can be modeled by a NN...

Idea

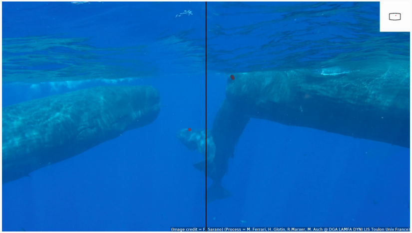
To incorporate the physics constraint in the minimization problem we use the implicit function theorem and the chain-rule to calculate the gradient of the loss function with respect to all the parameters—inversion and NN

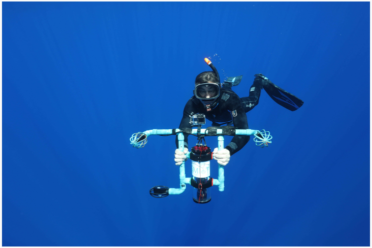
Application

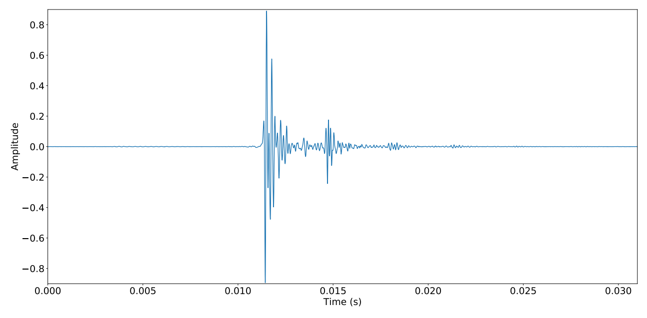
Application

The physical problem

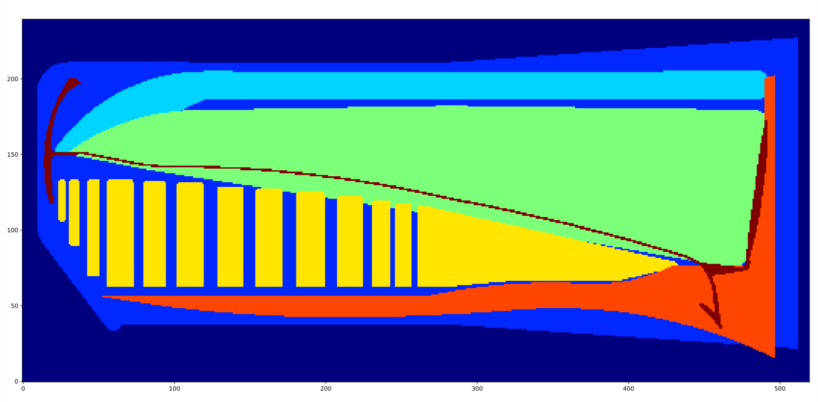
ADSIL: Sperm whale monitoring



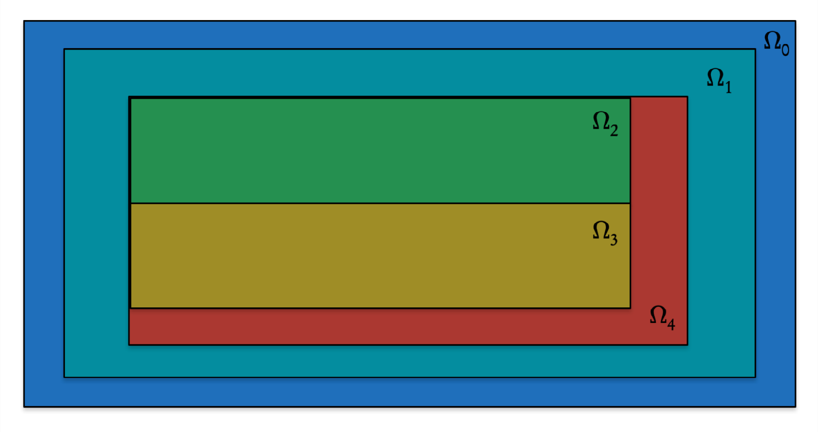




Sperm whale model



Sperm whale model



Inverse problem

For the elasto-acoustic wave propagation model

$$L(u, m) = f, \quad \text{in } \Omega$$

- Given: recorded signals, $u^{\text{obs}}(x, t)$ at points $\{x_i\} \in \Omega_0$
- Estimate:
  - material properties, $m_i(x)$, $i = 1, 2, 3, 4$ in each region Ω_i
  - location and form of the initial source pulse $f(x, t)$.

Inverse problem

For the elasto-acoustic wave propagation model

$$L(u, m) = f, \quad \text{in } \Omega$$

- Given: recorded signals, $u^{\mathrm{obs}}(x, t)$ at points $\{x_i\} \in \Omega_0$
- Estimate:
    - material properties, $m_i(x)$, $i = 1, 2, 3, 4$ in each region Ω_i
    - location and form of the initial source pulse $f(x, t)$.

Inverse problem

For the elasto-acoustic wave propagation model

$$L(u, m) = f, \quad \text{in } \Omega$$

- Given: recorded signals, $u^{\text{obs}}(x, t)$ at points $\{x_i\} \in \Omega_0$
- Estimate:
    - material properties, $m_i(x)$, $i = 1, 2, 3, 4$ in each region Ω_i
    - location and form of the initial source pulse $f(x, t)$.

Inverse problem

For the elasto-acoustic wave propagation model

$$L(u, m) = f, \quad \text{in } \Omega$$

- Given: recorded signals, $u^{\text{obs}}(x, t)$ at points $\{x_i\} \in \Omega_0$
- Estimate:
  - material properties, $m_i(x)$, $i = 1, 2, 3, 4$ in each region Ω_i
  - location and form of the initial source pulse $f(x, t)$.

Inverse problem

For the elasto-acoustic wave propagation model

$$L(u, m) = f, \quad \text{in } \Omega$$

- Given: recorded signals, $u^{\mathrm{obs}}(x, t)$ at points $\{x_i\} \in \Omega_0$
- Estimate:
  - material properties, $m_i(x)$, $i = 1, 2, 3, 4$ in each region Ω_i
  - location and form of the initial source pulse $f(x, t)$.

Application

Classical inversion by adjoint method

Adjoint

- formulation – see Ferrari, 2020.
- implementation of inversion – TBC.

Application

Machine learning inversions

Prospective

- start with AD, using NN for the pulse shape
- continue with PI-DeepONet, for example.

Thank you!

Reserve

Reserve

DeepONet

deepOnet formulation

- Parametric, linear/nonlinear operator plus IBC (IBVP)

$$\mathcal{O}(u, s) = 0,$$
$$\mathcal{B}(u, s) = 0,$$

- where
    - $u \in \mathcal{U}$ is the input function (parameters),
    - $s \in \mathcal{S}$ is the hidden, solution function
- If $\exists!$ solution $s = s(u) \in \mathcal{S}$ to the IBVP, then we can define the solution operator $G\colon \mathcal{U} \mapsto \mathcal{S}$ by

$$G(u) = s(u).$$

deepOnet formulation II

- Approximate the solution map G by a DeepONet G_θ

$$G_\theta(u)(y) = \sum_{k=1}^{q} \underbrace{b_k(u(x))}_{\text{branch}} \underbrace{t_k(y)}_{\text{trunk}} + b_0$$

  where θ represents all the trainable weights and biases, computed by minimizing the loss at a set of P random output points $\{y_j\}_{j=1}^{P}$

$$\mathcal{L}(u,\theta) = \frac{1}{P}\sum_{j=1}^{P} \left|G_\theta(u)(y_j) - s(y_j)\right|^2,$$

  and $s(y_j)$ is the PDE solution evaluated at P locations in the domain of $G(u)$

deepOnet formulation III

- To obtain a vector output, a stacked version is defined by repeated sampling over $i = 1, \ldots, N$, giving the overall operator loss

$$\mathcal{L}_O(\theta) = \frac{1}{NP} \sum_{i=1}^{N} \sum_{j=1}^{P} \left| G_\theta(u^{(i)})(y_j^{(i)}) - s^{(i)}(y_j^{(i)}) \right|^2$$

PI-deepOnet

- Train by minimizing the composite loss

$$\mathcal{L}(\theta) = \mathcal{L}_o(\theta) + \mathcal{L}_\phi(\theta),$$

  where
  - the operator loss is as above for deepOnet, or using the IBC

$$\mathcal{L}_o(\theta) = \frac{1}{NP} \sum_{i=1}^{N} \sum_{j=1}^{P} \left| \mathcal{B}\left(u^{(i)}(x_j^{(i)}), G_\theta(u^{(i)})(y_j^{(i)}) \right) \right|^2$$

  - the physics loss is computed using the operator network approximate solution

$$\mathcal{L}_\phi(\theta) = \frac{1}{NQ} \sum_{i=1}^{N} \sum_{j=1}^{Q} \left| \mathcal{O}\left(u^{(i)}(x_j^{(i)}), G_\theta(u^{(i)})(y_j^{(i)}) \right) \right|^2$$

- This is self-supervised, and does not require paired input-output observations!

Reserve

Inverse problem and AD

AD: Inverse problem formulation

- Given a physical relation

$$F(u; \boldsymbol{\theta}) = 0 \tag{2}$$

  represented by an IBVP, or other functional relationship, with
  - u the physical quantity
  - $\boldsymbol{\theta}$ the (material/medium) properties/parameters
- Inverse Problem is defined as:
  - Given observations/measurements of u at the locations $\mathsf{x} = \{x_i\}$

$$\mathsf{u}^{\mathrm{obs}} = \{u(x_i)\}_{i \in \mathcal{I}}$$

  - Estimate the parameters $\boldsymbol{\theta}$ by minimizing a loss/objective/cost function

$$L(\boldsymbol{\theta}) = \left\| u(\mathsf{x}) - \mathsf{u}^{\mathrm{obs}} \right\|_2^2$$

    subject to (2).

AD: Physics constrained learning

- If $\boldsymbol{\theta} = \boldsymbol{\theta}(x)$, model it by a NN
- Express numerical scheme for approximating the PDE (2) as a computational graph $G(\boldsymbol{\theta})$
- Use reverse-mode AD (aka. backpropagation) to compute the gradient of L with respect to $\boldsymbol{\theta}$ and the NN coefficients (weights and biases)
- Minimize by a suitable gradient algorithm
    - Adam, SGD (1st order)
    - L-BFGS (quasi-Newton)
    - trust-region (2nd order)

AD: Computing the gradient

- Optimization problem: $\min_\theta L(u)$ subject to $F(\theta, u) = 0$.
- Suppose we have a computational graph for $u = G(\theta)$.
- Then $\tilde{L}(\theta) = L\left(G(\theta)\right)$ and by the IFT we can compute the gradient with respect to θ,
    - first of F,

$$\frac{\partial F}{\partial \theta} + \frac{\partial F}{\partial u}\frac{\partial G}{\partial \theta} = 0, \quad \Rightarrow \quad \frac{\partial G}{\partial \theta} = -\left[\frac{\partial F}{\partial u}\right]^{-1}\frac{\partial F}{\partial \theta}$$

    - then of $\tilde{L}$, by the chain rule,

$$\frac{\partial \tilde{L}}{\partial \theta} = \frac{\partial L}{\partial u}\frac{\partial G}{\partial \theta} = -\frac{\partial L}{\partial u}\left[\frac{\partial F}{\partial u}\right]^{-1}\frac{\partial F}{\partial \theta}$$

- The first derivative is obtained directly from the loss function, the second and third by reverse-mode AD

Reserve

Wave propagation model

Acoustic-elastic wave equation

- In the fluid regions, Ω_f, we will solve the acoustic wave equation system,

$$\begin{aligned} \rho\frac{\partial \mathbf{v}}{\partial t} &= -\nabla p \quad \text{in } \Omega_\mathrm{f} \times [0, T], \\ \frac{\partial p}{\partial t} &= -\rho c^2 \nabla \cdot \mathbf{v} \quad \text{in } \Omega_\mathrm{f} \times [0, T]. \end{aligned} \tag{3}$$

- In the solid regions, Ω_s, we will solve the elastic wave equation system,

$$\begin{aligned} \rho\frac{\partial v_i}{\partial t} &= \sum_{j=1}^{3} \frac{\partial \sigma_{ij}}{\partial x_j} \quad \text{in } \Omega_\mathrm{s} \times [0, T], \\ \frac{\partial \sigma_{ij}}{\partial t} &= \frac{1}{2} \sum_{k=1}^{3} \sum_{l=1}^{3} c_{ijkl} \left(\frac{\partial v_k}{\partial x_l} + \frac{\partial v_l}{\partial x_k} \right) \quad \text{in } \Omega_\mathrm{s} \times [0, T]. \end{aligned} \tag{4}$$

Wave equation: source and IBC

- The acoustic source will be simulated as a forcing term, $f(x,t)$, on the right-hand side of the pressure equation (3), or (4), depending on whether it is located in the fluid or solid regions, respectively.
- To complete this system, we add the following boundary conditions:
    - On the exterior, fluid boundary, an absorbing boundary condition on p.
    - On the interior boundaries, between different materials, interface conditions that are described below.
- Finally, the initial conditions are set equal to zero for p, v and σ since a forcing function is used.

Adjoint-state system

- Cost function to be minimized,

$$J(C,\rho) = \frac{1}{2}\int_0^T \int_\Omega \|\boldsymbol{u} - \boldsymbol{u}^{obs}\|_2^2 \,\mathrm{d}V\,\mathrm{d}t$$

where

$$C = \begin{pmatrix} 2\mu+\lambda & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & 2\mu+\lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & 2\mu+\lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & 2\mu & 0 & 0 \\ 0 & 0 & 0 & 0 & 2\mu & 0 \\ 0 & 0 & 0 & 0 & 0 & 2\mu \end{pmatrix},$$

- Gradient of J can be expressed in terms of λ and μ as

$$\nabla_\lambda J(\lambda) = \int_0^T \int_\Omega < 4\lambda \sum_{i=1}^{3}\sum_{j=3}^{3} \frac{\partial v_j}{\partial x_j}\eta_i \boldsymbol{v} | R^* \boldsymbol{\eta} > dVdt,$$

$$\nabla_\mu J(\mu) = \int_0^T \int_\Omega < 2\mu I_6 D_2 \boldsymbol{v} | R^* \boldsymbol{\eta} > dVdt.$$