Some topics on machine learning for scientific computing

Surrogate modeling, optimization and system identification

Kjetil Olsen Lye

Geilo 2026

Surrogate modeling and optimization

SINTEF

Goal of this part

Find

$$\tilde{\mathbf{y}} = \arg\min_{\mathbf{y}\in[0,1]^d} J(\mathbf{y}),$$

where $J : [0,1]^d \to \mathbb{R}$ is assumed expensive to compute.

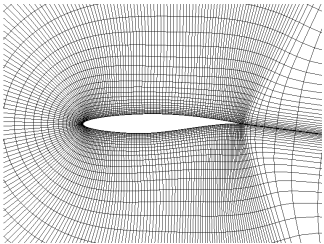
A motivating example: Airfoil design

Find smallest drag-to-lift ratio

A motivating example: Airfoil design

Find smallest drag-to-lift ratio

- $\mathbf{y} = (y_1, \ldots, y_d) \in [0, 1]^d$ airfoil design
- $C_D(\mathbf{y})$ =Drag coefficient
- $C_L(\mathbf{y})$ =Lift coefficient
- $J(\mathbf{y}) = C_D(\mathbf{y})/C_L(\mathbf{y})$



Challenges

$$\min J(\mathbf{y}) \qquad \mathbf{y} \in [0,1]^d.$$

Algorithm (gradient descent):

$$\mathbf{y}_k = \mathbf{y}_{k-1} - \alpha \nabla_{\mathbf{y}} J(\mathbf{y}_{k-1})$$

Challenges

$$\min J(\mathbf{y}) \qquad \mathbf{y} \in [0,1]^d.$$

Algorithm (gradient descent):

$$\mathbf{y}_k = \mathbf{y}_{k-1} - \alpha \nabla_{\mathbf{y}} J(\mathbf{y}_{k-1})$$

- J expensive to evaluate
- $\nabla_{\mathbf{y}} J(\mathbf{y}) = ??$:
  - difficult to obtain
  - expensive to compute

SINTEF

Goal of this talk

Replace J by a cheap surrogate $\mathcal{D}$ and solve

$$\min \mathcal{D}(\mathbf{y}) \qquad \mathbf{y} \in [0,1]^d.$$

SINTEF

Goal of this talk

Replace J by a cheap surrogate $\mathcal{D}$ and solve

$$\min \mathcal{D}(\mathbf{y}) \qquad \mathbf{y} \in [0,1]^d.$$

Can this be done in a ”stable” manner?

SINTEF

Deep Neural Networks: An informal definition

A deep neural network of depth K is a function $\mathcal{D} : \mathbb{R}^n \to \mathbb{R}^m$ where

$$\mathcal{D} = \sigma_K \circ T_K \circ \sigma_{K-1} \circ T_{K-1} \circ \cdots \circ \sigma_1 \circ T_1$$

where

- $\{T_i\}_i$ are affine transformations
- $\{\sigma_i\}_i$ are specified (non-linear) functions.

SINTEF

Universal approximation property

Theorem (Continuous version, Cybenko, Hornik)

Any continuous function can be approximated arbitrarily well (in the sup*-norm) by a neural network with depth at most* 2*.*

SINTEF

Universal approximation property

Theorem (Continuous version, Cybenko, Hornik)

Any continuous function can be approximated arbitrarily well (in the sup*-norm) by a neural network with depth at most* 2*.*

Theorem (L^1 version, Zhou Lu)

Any L^1 function can be approximated arbitrarily well (in the L^1-norm) by a neural network of bounded width.

SINTEF

Universal approximation property

Theorem (Continuous version, Cybenko, Hornik)

Any continuous function can be approximated arbitrarily well (in the sup*-norm) by a neural network with depth at most* 2*.*

Theorem (L^1 version, Zhou Lu)

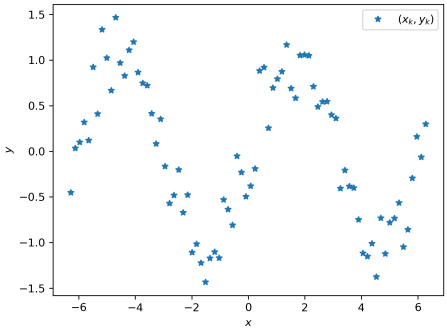
Any L^1 function can be approximated arbitrarily well (in the L^1-norm) by a neural network of bounded width.

Also results for available for solutions of common PDEs/SDEs.

DNN: Coefficients and biases

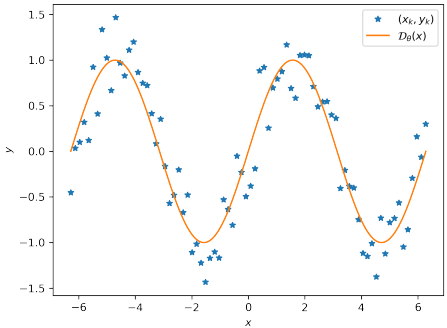
$$\mathcal{D}_\theta = \sigma_K \circ T_{K,\theta} \circ \sigma_{K-1} \circ T_{K-1,\theta} \circ \cdots \circ \sigma_1 \circ T_{1,\theta}$$

DNN: Regression from data



- Observations $\{x_k, y_k\}_{k=1}^{M} \subset \mathbb{R}^n \times \mathbb{R}^m$,

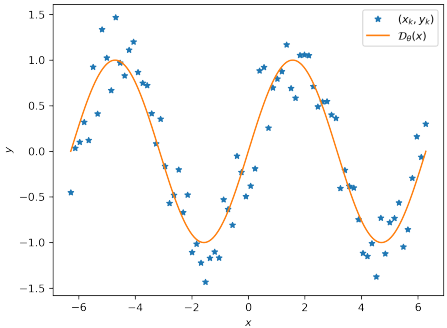
DNN: Regression from data



- Observations $\{x_k, y_k\}_{k=1}^{M} \subset \mathbb{R}^n \times \mathbb{R}^m$,
- Find $\mathcal{D}_\theta$ such that

$$\theta = \arg\min_\theta \left(\frac{1}{M} \sum_{k=1}^{M} |y_k - \mathcal{D}_\theta(x_k)|^p \right)^{1/p}.$$

DNN: Regression from data

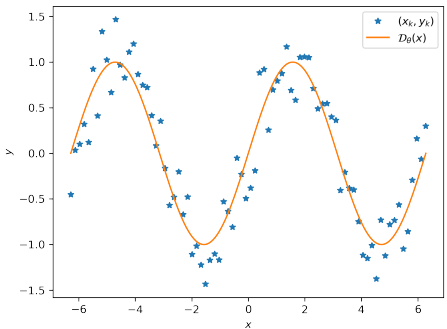


- Observations $\{x_k, y_k\}_{k=1}^M \subset \mathbb{R}^n \times \mathbb{R}^m$,
- Find $\mathcal{D}_\theta$ such that

$$\theta = \arg\min_\theta \left(\frac{1}{M} \sum_{k=1}^{M} |y_k - \mathcal{D}_\theta(x_k)|^p \right)^{1/p}.$$

- Usually solved using stochastic gradient descent.
- We say we *learn* $\mathcal{D}_\theta$ from $\{x_k, y_k\}_{k=1}^M$

DNN: Regression from data



- Observations $\{x_k, y_k\}_{k=1}^M \subset \mathbb{R}^n \times \mathbb{R}^m$,
- Find $\mathcal{D}_\theta$ such that

$$\theta = \arg\min_\theta \left(\frac{1}{M} \sum_{k=1}^{M} |y_k - \mathcal{D}_\theta(x_k)|^p \right)^{1/p}.$$

- Usually solved using stochastic gradient descent.
- We say we *learn* $\mathcal{D}_\theta$ from $\{x_k, y_k\}_{k=1}^M$

Relation to previous talk:

$$\frac{1}{M} \sum_{k=1}^{M} |y_k - \mathcal{D}_\theta(x_k)|^p \approx \mathbb{E}(|y - \mathcal{D}_\theta(x)|^p).$$

Learning functionals/observables

Recall:

$$\min_{\mathbf{y}} J(\mathbf{y})$$

- J is often a composition:

$$J(\mathbf{y}) = \mathcal{J}(u(\mathbf{y}, \cdot)),$$

- where $u(\mathbf{y}, \cdot) : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^N$ is a solution to a PDE.

SINTEF

Learning functionals/observables

Recall:

$$\min_{\mathbf{y}} J(\mathbf{y})$$

- J is often a composition:

$$J(\mathbf{y}) = \mathcal{J}(u(\mathbf{y}, \cdot)),$$

- where $u(\mathbf{y}, \cdot) : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^N$ is a solution to a PDE.
- u expensive ($\mathcal{J}$ is often cheap).

Learning functionals/observables

Recall:

$$\min_{\mathbf{y}} J(\mathbf{y})$$

- J is often a composition:

$$J(\mathbf{y}) = \mathcal{J}(u(\mathbf{y}, \cdot)),$$

- where $u(\mathbf{y}, \cdot) : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^N$ is a solution to a PDE.
- u expensive ($\mathcal{J}$ is often cheap).
- J might admit a simpler approximation

SINTEF

Learning functionals/observables

- Draw M random points $\mathbf{y}_1, \ldots, \mathbf{y}_M$

SINTEF

Learning functionals/observables

- Draw M random points $\mathbf{y}_1, \ldots, \mathbf{y}_M$
- Compute $J(\mathbf{y}_i)$ for $i = 1, \ldots, M$

SINTEF

Learning functionals/observables

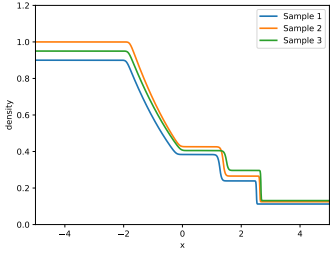
- Draw M random points $\mathbf{y}_1, \ldots, \mathbf{y}_M$
- Compute $J(\mathbf{y}_i)$ for $i = 1, \ldots, M$
- Learn $\mathcal{D}_\theta$ from $\{\mathbf{y}_i, J(\mathbf{y}_i)\}_{i=1}^M$.

First example: Functionals of compressible Euler

With the Sod shock tube problem

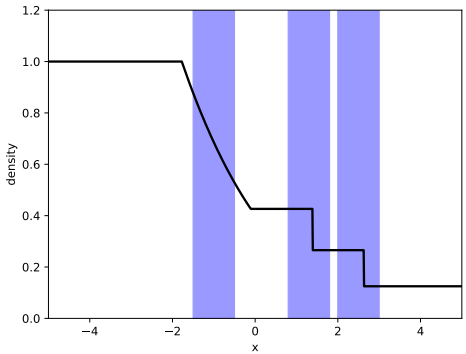
Compressible Euler equations:

$$\rho_t + \nabla \cdot (\rho \vec{v}) = 0$$
$$(\rho \vec{v})_t + \nabla \cdot (\rho \vec{v} \otimes \vec{v} + pI) = 0$$
$$E_t + \nabla \cdot ((E + p)\vec{v}) = 0$$

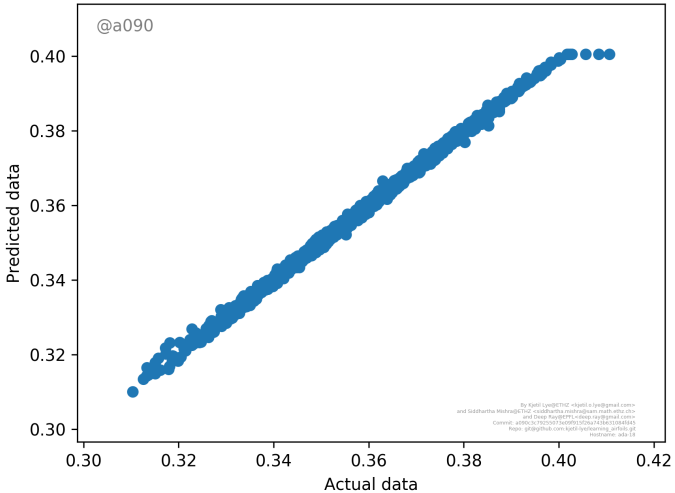


First example: Functionals of compressible Euler

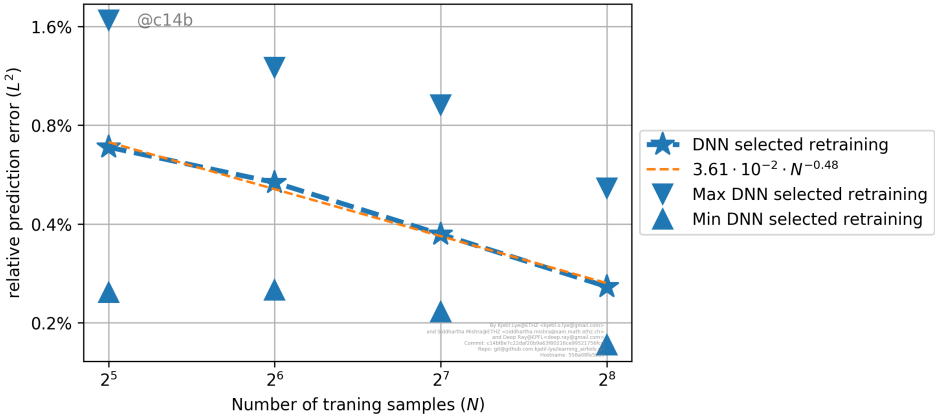
$$J_q(y) = \frac{1}{|I_q|} \int_{I_q} \rho(y; x, T)\, dx \qquad q = 1, 2, 3.$$



Functionals of compressible Euler: Scatter plot



Functionals of compressible Euler: How many samples?



Hyperparameters

- Choice of optimizer:
  - Stochastic Gradient Descent
  - Adam
  - Many more

SINTEF

Hyperparameters

- Choice of optimizer:
  - Stochastic Gradient Descent
  - Adam
  - Many more
- Activation function
  - ReLU
  - Sigmoid
  - Leaky ReLU
  - …

Hyperparameters

- Choice of optimizer:
  - Stochastic Gradient Descent
  - Adam
  - Many more
- Activation function
  - ReLU
  - Sigmoid
  - Leaky ReLU
  - …
- Regularization
  - L^1/L^2
  - Size of regularization

Hyperparameters

- Choice of optimizer:
  - Stochastic Gradient Descent
  - Adam
  - Many more
- Activation function
  - ReLU
  - Sigmoid
  - Leaky ReLU
  - …
- Regularization
  - L^1/L^2
  - Size of regularization
- Loss function
  - mean absolute error
  - mean squared error

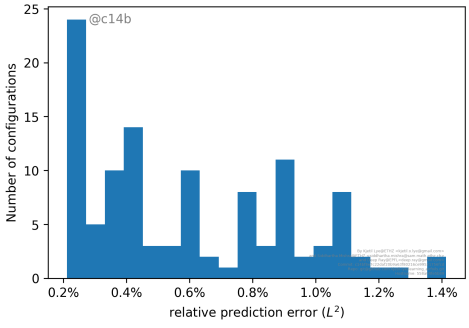
Hyperparameters

- Choice of optimizer:
  - Stochastic Gradient Descent
  - Adam
  - Many more
- Activation function
  - ReLU
  - Sigmoid
  - Leaky ReLU
  - …
- Regularization
  - L^1/L^2
  - Size of regularization
- Loss function
  - mean absolute error
  - mean squared error
- Network architecture
  - Width
  - Depth

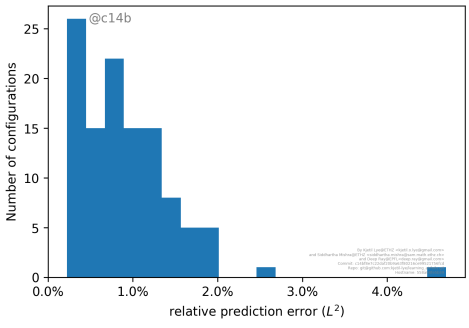
Hyperparameters

- Choice of optimizer:
  - Stochastic Gradient Descent
  - Adam
  - Many more
- Activation function
  - ReLU
  - Sigmoid
  - Leaky ReLU
  - …
- Regularization
  - L^1/L^2
  - Size of regularization
- Loss function
  - mean absolute error
  - mean squared error
- Network architecture
  - Width
  - Depth
- Retraining
  - How many?
  - How to select the best?
- Step size
- Stopping criterion

Functionals of compressible Euler

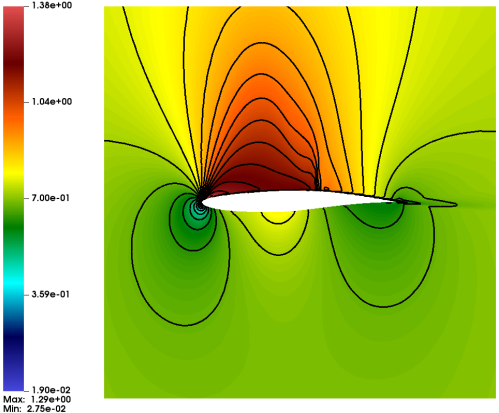


$q = 1$



$q = 2$

Second example: Airfoils

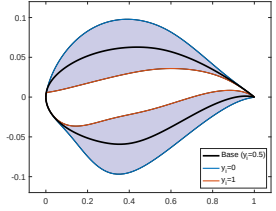


Two functionals: Drag and lift coefficients

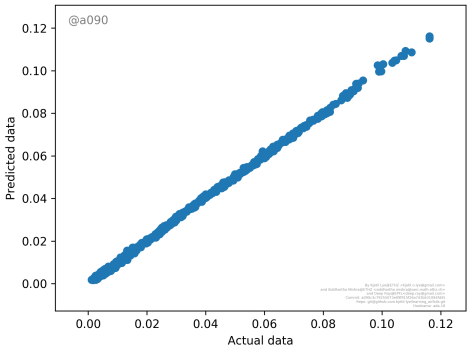
$$C_L(\mathbf{y}) = \frac{1}{K^\infty(\mathbf{y})} \int_S p(\mathbf{y}) n(\mathbf{y}) \cdot \hat{x} ds,$$

$$C_D(\mathbf{y}) = \frac{1}{K^\infty(\mathbf{y})} \int_S p(\mathbf{y}) n(\mathbf{y}) \cdot \hat{y} ds,$$

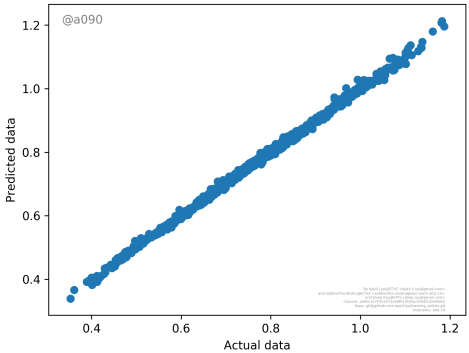
$\mathbf{y}$ is the design of airfoil.



Airfoils: Scatter plot

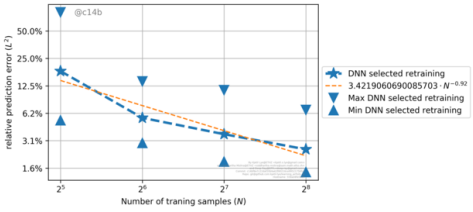


Drag

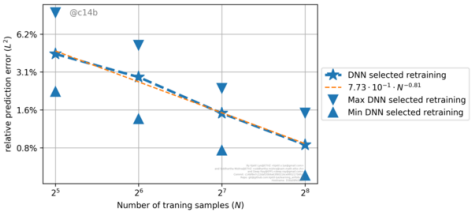


Lift

Airfoils: Number of samples



Drag



Lift

SINTEF

A measure of the error

Theorem (Lye, Mishra, Molinaro; 2020)

$$\mathbb{E}(\|J - \mathcal{D}_\theta\|_{L^2([0,1]^d)}) \sim \bar{\mathcal{E}}_T + \bar{\mathcal{E}}_{TV} + C\frac{(\mathrm{std}(J) + \mathrm{std}(\mathcal{D}_\theta))}{\sqrt{M}}$$

- $\bar{\mathcal{E}}_T$ training error
- $\bar{\mathcal{E}}_{TV}$ validation gap

SINTEF

A measure of the error

Theorem (Lye, Mishra, Molinaro; 2020)

$$\mathbb{E}(\|J - \mathcal{D}_\theta\|_{L^2([0,1]^d)}) \sim \bar{\mathcal{E}}_T + \bar{\mathcal{E}}_{TV} + C\frac{(\mathrm{std}(J) + \mathrm{std}(\mathcal{D}_\theta))}{\sqrt{M}}$$

- $\bar{\mathcal{E}}_T$ training error
- $\bar{\mathcal{E}}_{TV}$ validation gap

A Monte Carlo estimate.

MLMC for machine learning

- Learn base level $\ell = 0$:
  - Draw M_0 samples $\mathbf{y}_1, \ldots, \mathbf{y}_{M_0}$
  - Evolve $J^{\Delta_0}(\mathbf{y}_i)$ for $i = 1, \ldots, M_0$
  - Learn $\mathcal{D}^0$ from $\{\mathbf{y}_i, J^{\Delta_0}(\mathbf{y}_i)\}_i$

MLMC for machine learning

- Learn base level $\ell = 0$:
  - Draw M_0 samples $\mathbf{y}_1, \ldots, \mathbf{y}_{M_0}$
  - Evolve $J^{\Delta_0}(\mathbf{y}_i)$ for $i = 1, \ldots, M_0$
  - Learn $\mathcal{D}^0$ from $\{\mathbf{y}_i, J^{\Delta_0}(\mathbf{y}_i)\}_i$

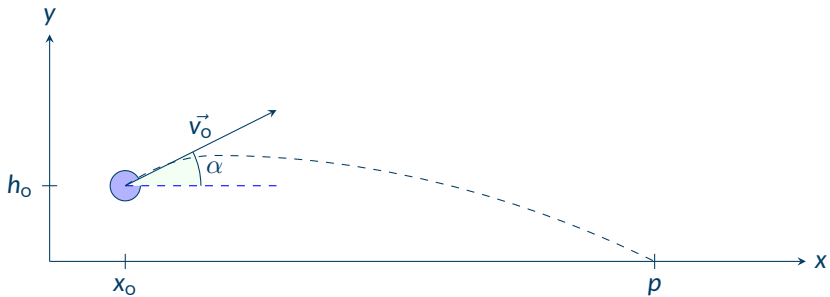
- For each level $\ell = 1, \ldots, L$:
  - Draw M_ℓ samples $\mathbf{y}_1, \ldots, \mathbf{y}_{M_\ell}$
  - Evolve $J^{\Delta_\ell}(\mathbf{y}_i)$ and $J^{\Delta_{\ell-1}}$for $i = 1, \ldots, M_0$
  - Learn $\mathcal{D}^\ell$ from $\{\mathbf{y}_i, J^{\Delta_\ell}(\mathbf{y}_i) - J^{\Delta_{\ell-1}}(\mathbf{y}_i)\}_i$

  Set

$$\mathcal{D} = \sum_{\ell=0}^{L} \mathcal{D}^\ell.$$

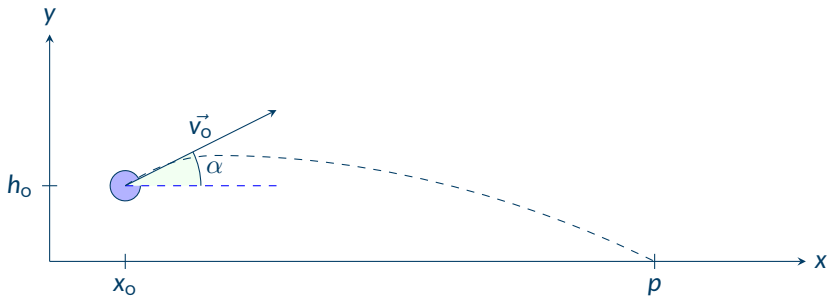
MLMC for machine learning: Numerical experiment

Projectile motion:



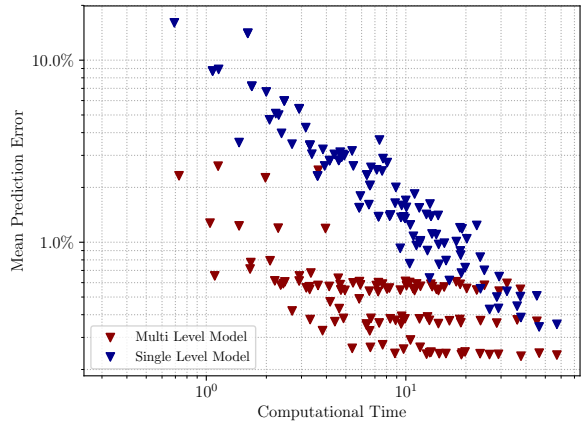
MLMC for machine learning: Numerical experiment

Projectile motion:

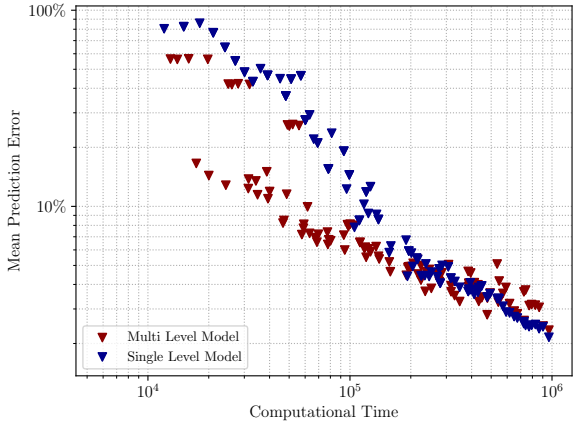


$$p(\|\vec{v}_o\|, \alpha).$$

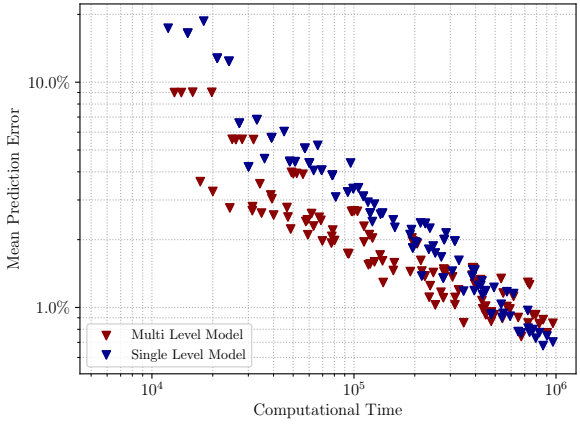
Projectile motion: results



Airfoil: Drag



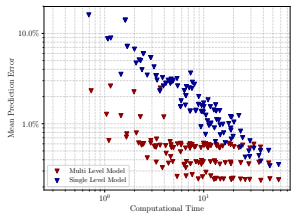
Airfoil: Lift

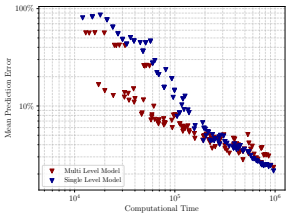


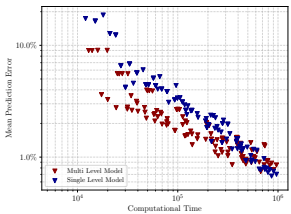
Multilevel surrogate learning offers reduced computational time

Theorem (Lye, Mishra, Molinaro; 2020)

$$\mathbb{E}(\|J - \mathcal{D}_\theta\|_{L^2([0,1]^d)}) \sim \bar{\mathcal{E}}_T + \bar{\mathcal{E}}_{TV} + C\frac{(\mathrm{std}(J) + \mathrm{std}(\mathcal{D}_\theta))}{\sqrt{M}}$$







Optimization using surrogates

The simple approach

$$\tilde{\mathbf{y}} = \arg\min_{\mathbf{y} \in [0,1]^d} J(\mathbf{y})$$

The simple approach

$$\tilde{\mathbf{y}} = \arg\min_{\mathbf{y}\in[0,1]^d} J(\mathbf{y})$$

- Draw M samples $y_1, y_2, \ldots, y_M$
- Evolve $J(\mathbf{y}_i)$ for $i = 1, \ldots, M$
- Learn $\mathcal{D}$ from $\{\mathbf{y}_i, J(\mathbf{y}_i)\}$

The simple approach

$$\tilde{\mathbf{y}} = \arg\min_{\mathbf{y}\in[0,1]^d} J(\mathbf{y})$$

- Draw M samples $y_1, y_2, \ldots, y_M$
- Evolve $J(\mathbf{y}_i)$ for $i = 1, \ldots, M$
- Learn $\mathcal{D}$ from $\{\mathbf{y}_i, J(\mathbf{y}_i)\}$
- Solve

$$\tilde{\mathbf{y}} = \arg\min_{\mathbf{y}\in[0,1]^d} \mathcal{D}(\mathbf{y})$$

The simple approach

$$\tilde{\mathbf{y}} = \arg\min_{\mathbf{y}\in[0,1]^d} J(\mathbf{y})$$

- Draw M samples $y_1, y_2, \ldots, y_M$
- Evolve $J(\mathbf{y}_i)$ for $i = 1, \ldots, M$
- Learn $\mathcal{D}$ from $\{\mathbf{y}_i, J(\mathbf{y}_i)\}$
- Solve

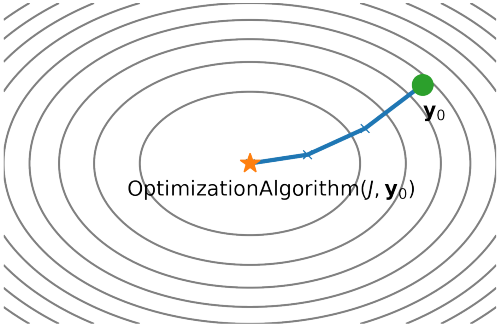
$$\tilde{\mathbf{y}} = \arg\min_{\mathbf{y}\in[0,1]^d} \mathcal{D}(\mathbf{y})$$

Problem: Error of $\mathcal{D}$ is large near extrema!

Iterative surrogate model optimization (ISMO)

Observation: Most optimization algorithms take a *starting value* $\mathbf{y}_0$:

$$\text{OptimizationAlgorithm}(J, \mathbf{y}_0)$$



Iterative surrogate model optimization (ISMO)

Observation: Most optimization algorithms take a *starting value* $\mathbf{y}_0$:

$$\text{OptimizationAlgorithm}(J, \mathbf{y}_0)$$

Bootstrap:

- Draw M samples $\mathbf{y}_1, \ldots, \mathbf{y}_M$
- Evolve $J(\mathbf{y}_i)$ for $i = 1, \ldots, M$
- Learn $\mathcal{D}(\mathbf{y})$ from $\{\mathbf{y}_i, J(\mathbf{y}_i)\}$

1. For iteration $q = 1, \ldots$

Iterative surrogate model optimization (ISMO)

Observation: Most optimization algorithms take a *starting value* $\mathbf{y}_0$:

$$\text{OptimizationAlgorithm}(J, \mathbf{y}_0)$$

Bootstrap:

- Draw M samples $\mathbf{y}_1, \ldots, \mathbf{y}_M$
- Evolve $J(\mathbf{y}_i)$ for $i = 1, \ldots, M$
- Learn $\mathcal{D}(\mathbf{y})$ from $\{\mathbf{y}_i, J(\mathbf{y}_i)\}$

1. For iteration $q = 1, \ldots$
   1.1 Draw M samples $\tilde{\mathbf{y}}_{qM}, \ldots, \tilde{\mathbf{y}}_{(q+1)M}$

Iterative surrogate model optimization (ISMO)

Observation: Most optimization algorithms take a *starting value* $\mathbf{y}_0$:

$$\text{OptimizationAlgorithm}(J, \mathbf{y}_0)$$

Bootstrap:

- Draw M samples $\mathbf{y}_1, \ldots, \mathbf{y}_M$
- Evolve $J(\mathbf{y}_i)$ for $i = 1, \ldots, M$
- Learn $\mathcal{D}(\mathbf{y})$ from $\{\mathbf{y}_i, J(\mathbf{y}_i)\}$

1. For iteration $q = 1, \ldots$
   1.1 Draw M samples $\tilde{\mathbf{y}}_{qM}, \ldots, \tilde{\mathbf{y}}_{(q+1)M}$
   1.2 For $n = qM, \ldots, (q+1)M$: $\mathbf{y}_n = \text{OptimizationAlgorithm}(\mathcal{D}, \tilde{\mathbf{y}}_n)$

Iterative surrogate model optimization (ISMO)

Observation: Most optimization algorithms take a *starting value* $\mathbf{y}_0$:

$$\text{OptimizationAlgorithm}(J, \mathbf{y}_0)$$

Bootstrap:

- Draw M samples $\mathbf{y}_1, \ldots, \mathbf{y}_M$
- Evolve $J(\mathbf{y}_i)$ for $i = 1, \ldots, M$
- Learn $\mathcal{D}(\mathbf{y})$ from $\{\mathbf{y}_i, J(\mathbf{y}_i)\}$

1. For iteration $q = 1, \ldots$
   1.1 Draw M samples $\tilde{\mathbf{y}}_{qM}, \ldots, \tilde{\mathbf{y}}_{(q+1)M}$
   1.2 For $n = qM, \ldots, (q+1)M$: $\mathbf{y}_n = \text{OptimizationAlgorithm}(\mathcal{D}, \tilde{\mathbf{y}}_n)$
   1.3 Learn $\mathcal{D}$ from $\{(\mathbf{y}_1, J(\mathbf{y}_1)), \ldots, (\mathbf{y}_{(q+1)M}, J(\mathbf{y}_{(q+1)M}))\}$

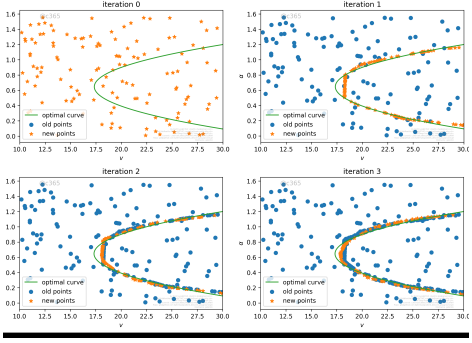
SINTEF

Projectile motion: Evolution of samples

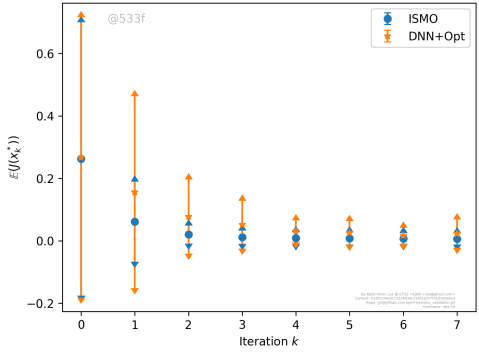
$$J(v_0, \alpha) = \frac{1}{2}\left(p(v_0, \alpha) - 15\right)^2 .$$

Projectile motion: Evolution of samples

$$J(v_0, \alpha) = \frac{1}{2}\left(p(v_0, \alpha) - 15\right)^2 .$$



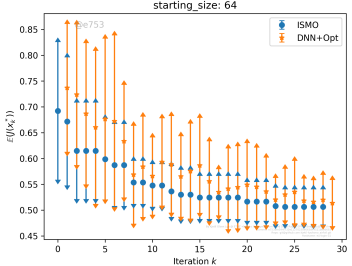
Projectile motion: Objective



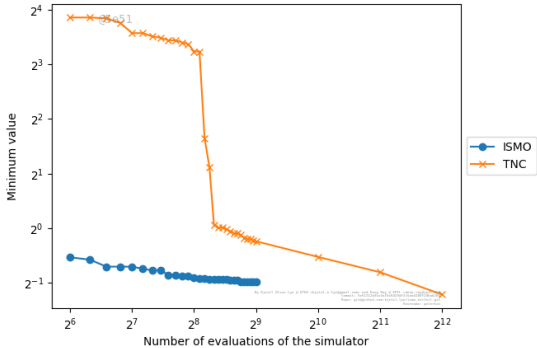
Airfoils: Objective

$$J(\mathbf{y}) = \frac{C_D(\mathbf{y})}{C_D^{ref}} + P \max\left(0, 0.999 - \frac{C_L(\mathbf{y})}{C_L^{ref}}\right)$$

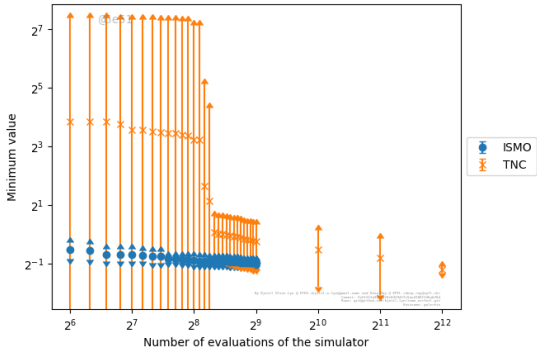
type: objective, script: airfoils_mc, generator: monte-carlo, batch_size_factor: 0.25, starting_size: 64



Comparison with traditional optimization

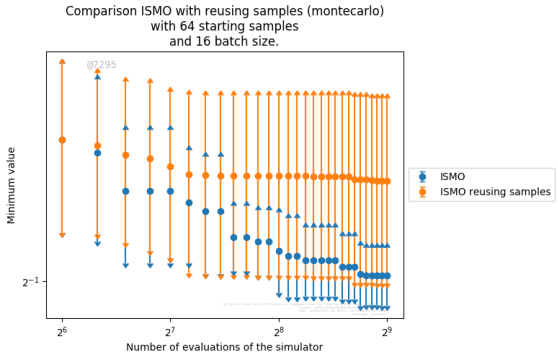


Comparison with traditional optimization



Why not reuse optimal values?

Often reach saddle points



SINTEF

Optimization under uncertainty

- $J(\omega; \mathbf{y})$ where $\omega \in \Omega$

SINTEF

Optimization under uncertainty

- $J(\omega; \mathbf{y})$ where $\omega \in \Omega$
- Look at *robust optimization*

$$\min_{\mathbf{y}} \left[\mathbb{E}(J(\cdot, \mathbf{y})) + \gamma \mathrm{Var}(J(\cdot, \mathbf{y})) \right]$$

SINTEF

Optimization under uncertainty

- $J(\omega; \mathbf{y})$ where $\omega \in \Omega$
- Look at *robust optimization*

$$\min_{\mathbf{y}} \left[\mathbb{E}(J(\cdot, \mathbf{y})) + \gamma \mathrm{Var}(J(\cdot, \mathbf{y})) \right]$$

- Key idea: Apply the ISMO algorithm to the function

$$\mathbf{y} \mapsto \mathbb{E}(J(\cdot, \mathbf{y})) + \gamma \mathrm{Var}(J(\cdot, \mathbf{y})).$$

Optimization under uncertainty

- $J(\omega; \mathbf{y})$ where $\omega \in \Omega$
- Look at *robust optimization*

$$\min_{\mathbf{y}} \left[\mathbb{E}(J(\cdot, \mathbf{y})) + \gamma \mathrm{Var}(J(\cdot, \mathbf{y})) \right]$$

- Key idea: Apply the ISMO algorithm to the function

$$\mathbf{y} \mapsto \mathbb{E}(J(\cdot, \mathbf{y})) + \gamma \mathrm{Var}(J(\cdot, \mathbf{y})).$$

Approximate $\mathbb{E}(J(\cdot, \mathbf{y})) + \gamma \mathrm{Var}(J(\cdot, \mathbf{y}))$ through (ML,quasi-) Monte Carlo.

Projectile motion with uncertain parameters

Design parameters ($\mathbf{y}$)

- Angle, $\alpha \in [0, 5\pi/12]$
- Initial speed $\|v_0\| \in [20, 40]$

Projectile motion with uncertain parameters

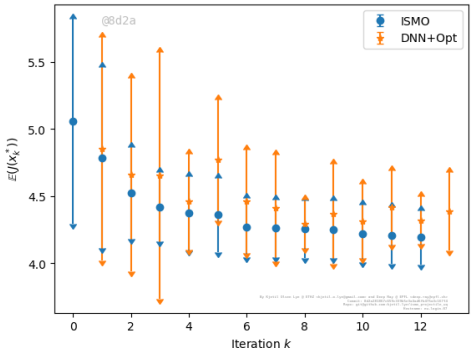
Design parameters ($\mathbf{y}$)

- Angle, $\alpha \in [0, 5\pi/12]$
- Initial speed $\|v_0\| \in [20, 40]$

Uncertain parameters (ω):

- Initial height: $h_0 \sim \mathcal{U}[0, 1]$.
- Air resistance $F_D = \frac{1}{2}\rho v^2 C_D \pi r^2$, where
  - $C_D \sim \mathcal{U}[0.09, 0.11]$ (Drag coefficient)
  - $r \sim \mathcal{U}[0.22, 0.235]$ (Radius of ball)
  - $\rho \sim \mathcal{U}[1.1, 1.4]$ (Density of air)
- Mass $m \sim \mathcal{U}[0.142, 0.149]$

Projectile motion with uncertain parameters



Learning PDEs from data: system identification

SINTEF

Rough idea of this part

Suppose we have M independent timeseries of spatial data of $N_T + 1$ snapshots in time

$$U_m^i \in \mathbb{R} \qquad \text{for } n = 0, \ldots, N_T, \text{ and } m = 1, \ldots, M,$$

can we find the PDE

$$\begin{cases} u_t + \mathcal{Q}(u) = 0 \\ u(x, 0) = U_m^0 \end{cases}$$

that "produced" those samples?

Rough idea of this part

Suppose we have M independent timeseries of spatial data of $N_T + 1$ snapshots in time

$$U_m^i \in \mathbb{R} \qquad \text{for } n = 0, \ldots, N_T, \text{ and } m = 1, \ldots, M,$$

can we find the PDE

$$\begin{cases} u_t + \mathcal{Q}(u) = 0 \\ u(x, 0) = U_m^0 \end{cases}$$

that "produced" those samples?

SINTEF

Universal approximation property (UAP)

Theorem (Continuous version, Cybenko, Hornik)

Any continuous function can be approximated arbitrarily well (in the sup*-norm) by a neural network with depth at most* 2*.*

Universal approximation property (UAP)

Theorem (Continuous version, Cybenko, Hornik)

Any continuous function can be approximated arbitrarily well (in the sup*-norm) by a neural network with depth at most* 2*.*

Theorem (L^1 version, Zhou Lu)

Any L^1 function can be approximated arbitrarily well (in the L^1-norm) by a neural network of bounded width.

SINTEF

Universal approximation property (UAP)

Theorem (Continuous version, Cybenko, Hornik)

Any continuous function can be approximated arbitrarily well (in the sup*-norm) by a neural network with depth at most* 2*.*

Theorem (L^1 version, Zhou Lu)

Any L^1 function can be approximated arbitrarily well (in the L^1-norm) by a neural network of bounded width.

Also results available for solutions of common PDEs/SDEs.

SINTEF

Universal approximation property (UAP)

Great, but how can we use these results?

Theorem (Continuous version, Cybenko, Hornik)

Any continuous function can be approximated arbitrarily well (in the sup*-norm) by a neural network with depth at most* 2*.*

Theorem (L^1 version, Zhou Lu)

Any L^1 function can be approximated arbitrarily well (in the L^1-norm) by a neural network of bounded width.

Also results available for solutions of common PDEs/SDEs.

SINTEF

Key uses of UAP in learning differential equations

- Learn the solution operators (Deep-O-nets, Fourier neural operators, ...):
  $u_o \mapsto u(t) \in \mathcal{F}(D, \mathbb{R})$

SINTEF

Key uses of UAP in learning differential equations

- Learn the solution operators (Deep-O-nets, Fourier neural operators, ...): $u_o \mapsto u(t) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to solution map: $\vec{y} \mapsto u(\vec{y}, t, \cdot) \in \mathcal{F}(D, \mathbb{R})$

SINTEF

Key uses of UAP in learning differential equations

- Learn the solution operators (Deep-O-nets, Fourier neural operators, ...): $u_0 \mapsto u(t) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to solution map: $\vec{y} \mapsto u(\vec{y}, t, \cdot) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to observable map: $\vec{y} \mapsto G(u(\vec{y}, \cdot)) \in \mathbb{R}$

SINTEF

Key uses of UAP in learning differential equations

- Learn the solution operators (Deep-O-nets, Fourier neural operators, ...): $u_0 \mapsto u(t) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to solution map: $\vec{y} \mapsto u(\vec{y}, t, \cdot) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to observable map: $\vec{y} \mapsto G(u(\vec{y}, \cdot)) \in \mathbb{R}$

SINTEF

Key uses of UAP in learning differential equations

- Learn the solution operators (Deep-O-nets, Fourier neural operators, ...): $u_0 \mapsto u(t) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to solution map: $\vec{y} \mapsto u(\vec{y}, t, \cdot) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to observable map: $\vec{y} \mapsto G(u(\vec{y}, \cdot)) \in \mathbb{R}$

Issue with the above: Requires large amounts of data

SINTEF

Key uses of UAP in learning differential equations

- Learn the solution operators (Deep-O-nets, Fourier neural operators, ...): $u_0 \mapsto u(t) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to solution map: $\vec{y} \mapsto u(\vec{y}, t, \cdot) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to observable map: $\vec{y} \mapsto G(u(\vec{y}, \cdot)) \in \mathbb{R}$

Issue with the above: Requires large amounts of data

Physics Informed Neural Networks (PINNs):

- Find $\mathcal{D}_\theta : \mathbb{R}^n \times [0, T) \to \mathbb{R}$ such that

$$\theta = \arg\min \left\{ \| \frac{\partial}{\partial t} \mathcal{D}_\theta + \mathcal{Q}(\mathcal{D}_\theta) \| \right\}$$

Key uses of UAP in learning differential equations

- Learn the solution operators (Deep-O-nets, Fourier neural operators, ...): $u_0 \mapsto u(t) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to solution map: $\vec{y} \mapsto u(\vec{y}, t, \cdot) \in \mathcal{F}(D, \mathbb{R})$
- Learn the parameter to observable map: $\vec{y} \mapsto G(u(\vec{y}, \cdot)) \in \mathbb{R}$

Issue with the above: Requires large amounts of data

Physics Informed Neural Networks (PINNs):

- Find $\mathcal{D}_\theta : \mathbb{R}^n \times [0, T) \to \mathbb{R}$ such that

$$\theta = \arg\min \left\{ \| \frac{\partial}{\partial t} \mathcal{D}_\theta + \mathcal{Q}(\mathcal{D}_\theta) \| \right\}$$

Issues with PINNs

- Dealing with non-smooth solutions
- We already know how to solve PDEs.

Learning differential equations

SINTEF

Fixing notation

We will consider a uniform discretization of $[0, 1]$ as

$$0 = x_1 < \ldots < x_M = 1$$

with $x_{i+1} - x_i = \Delta x$ for $i = 1, \ldots, N_x$. And for some fixed $T > 0$, we discretize $[0, T)$ as

$$0 = t_0 < \ldots < t_{N_T} = T.$$

For a function $u : [0, 1] \times [0, T) \to \mathbb{R}$ we will let

$$U^{n,i} \approx u(x_i, t_n) \qquad n = 0, \ldots, N_T, i = 1, \ldots, N_x.$$

We will let $\mathbf{U}^n := (U^{n,1}, \ldots, U^{n,N_x})$.

Discretization of smooth solutions

Consider the heat equation

$$u_t - u_{xx} = 0.$$

Discretizing in space and time yields

$$\frac{U^{n+1,i} - U^{n,i}}{\Delta t} - \frac{U^{n,i+1} - 2U^{n,i} + U^{n,i-1}}{\Delta x^2} = 0.$$

Discretization of smooth solutions

Consider the heat equation

$$u_t - u_{xx} = 0.$$

Discretizing in space and time yields

$$\frac{U^{n+1,i} - U^{n,i}}{\Delta t} - \frac{U^{n,i+1} - 2U^{n,i} + U^{n,i-1}}{\Delta x^2} = 0.$$

That is

$$\frac{1}{\Delta t}\left(\mathbf{U}^{n+1} - \mathbf{U}^{n}\right) = K_{\Delta x}(\mathbf{U}^{n})$$

where $K_{\Delta x}$ is a discrete convolution with kernel width 3.

Finite differences can be represented as convolutional neural networks

Learning linear PDEs

Suppose we are given measurements uniform in space and time

$$\mathbf{U}_m^n \qquad \text{for samples } m = 1, \dots, M, \qquad \text{and times } n = 1, \dots, N_T.$$

We let $\mathcal{C}_\theta : \mathbb{R}^{N_x} \to \mathbb{R}^{N_x}$ be a convolutional neural network (CNN) and solve

$$\theta = \arg\min \left\{ \sum_{m=1}^{M} \sum_{n=1}^{N_T} \left\| \frac{1}{\Delta t} \left(\mathbf{U}^n - \mathbf{U}_m^{n-1} \right) - \underbrace{\mathcal{C}_\theta(\mathbf{U}_m^{n-1})}_{\approx u_{xx}} \right\|^p \right\}.$$

What about non-linear PDEs?

Prototype PDE:[1]

$$u_t + f(u)_x = 0.$$

Main idea: Approximate f by a DNN $\mathcal{D}_\theta$, then convolve to get

$$f(u)_x \approx \frac{\partial}{\partial x}\mathcal{D}_\theta(\mathbf{U}) \approx \mathcal{C}_\theta(\mathcal{D}_\theta(\mathbf{U})).$$

That is, we optimize

$$\theta = \arg\min \left\{ \sum_{n=1}^{N_T} \sum_{m=1}^{M} \left\| \frac{1}{\Delta t}\left(\mathbf{U}^n - \mathbf{U}_m^{n-1}\right) - \underbrace{\mathcal{C}_\theta \circ \mathcal{D}_\theta(\mathbf{U}_m^{n-1})}_{\approx f(u)_x} \right\|^p \right\}.$$

[1] Ignoring shocks for now

The baseline algorithm: Accounting for source terms

We will consider two DNNs $\mathcal{D}^1_\theta, \mathcal{D}^2_\theta$ and one CNN $\mathcal{C}_\theta$ and solve

$$\theta = \arg\min \left\{ \sum_{n=1}^{N_T} \sum_{m=1}^{M} \left\| \frac{1}{\Delta t} \left(\mathbf{U}^n - \mathbf{U}_m^{n-1} \right) - \mathcal{D}^2_\theta \circ \mathcal{C}_\theta \circ \mathcal{D}^1_\theta (\mathbf{U}_m^{n-1}) \right\|^p \right\}.$$

The baseline algorithm: Accounting for source terms

We will consider two DNNs $\mathcal{D}^1_\theta, \mathcal{D}^2_\theta$ and one CNN $\mathcal{C}_\theta$ and solve

$$\theta = \arg\min \left\{ \sum_{n=1}^{N_T} \sum_{m=1}^{M} \left\| \frac{1}{\Delta t} \left(\mathbf{U}^n - \mathbf{U}_m^{n-1} \right) - \mathcal{D}^2_\theta \circ \mathcal{C}_\theta \circ \mathcal{D}^1_\theta (\mathbf{U}_m^{n-1}) \right\|^p \right\}.$$

Issues:

- Difficult to separate parts of the PDE (eg. dissipation)
- Difficult to enforce assumptions (though not impossible)

The pseudo-Hamiltonian formulation

Consider a class of PDEs expressible as

$$u_t = S\frac{\delta \mathcal{H}}{\delta u}[u] - R\frac{\delta \mathcal{V}}{\delta u}[u] + f(u, x, t)$$

where

$$\mathcal{H}[u] = \int_{[0,1]} H(x, u, u_x)\, dx \qquad \text{and} \qquad \mathcal{V}[u] = \int_{[0,1]} V(x, u, u_x)\, dx.$$

and

- f represents source terms
- S is a skew-symmetric operator
- R a positive semi-definite operator

The pseudo-Hamiltonian formulation: KdV

:

$$u_t = S\frac{\delta \mathcal{H}}{\delta u}[u] - R\frac{\delta \mathcal{V}}{\delta u}[u] + f(u, x, t)$$

KdV equation

$$u_t + \underbrace{\eta u u_x - \gamma^2 u_{xxx}}_{=S\frac{\delta \mathcal{H}}{\delta u}[u]} - \overbrace{\nu u_{xx}}^{R\frac{\delta \mathcal{V}}{\delta u}[u]} = 0. \tag{1}$$

- $R = I$
- $S = \frac{\partial}{\partial x}$
- $\mathcal{H} = -\int_\Omega \left(\frac{\eta}{6}u^3 + \frac{\gamma^2}{2}u_x^2\right) dx$
- $\mathcal{V} = \frac{\nu}{2}\int_\Omega u_x^2\, dx$.

Pseudo-Hamiltonian neural networks for PDEs [Eidnes and Lye, 2024]

Consider the class of PDEs expressible as

$$u_t = S\frac{\delta \mathcal{H}}{\delta u}[u] - R\frac{\delta \mathcal{V}}{\delta u}[u] + f(u, x, t)$$

We will discretize as

$$\frac{\mathbf{U}^{n+1} - \mathbf{U}^n}{\Delta t} = \left(\hat{S}_\theta \nabla \hat{\mathcal{H}}_\theta(\mathbf{U}^n) - \hat{R}_\theta \nabla \hat{\mathcal{V}}_\theta(\mathbf{U}^n) + \hat{f}_\theta(\mathbf{U}^n, x, t)\right),$$

$$\theta = \arg\min \left\{ \sum_{m=1}^{M} \sum_{n=1}^{N_T} \left\| \frac{\mathbf{U}_m^{n+1} - \mathbf{U}_m^n}{\Delta t} - \left(\hat{S}_\theta \nabla \hat{\mathcal{H}}_\theta(\mathbf{U}_m^n) - \hat{R}_\theta \nabla \hat{\mathcal{V}}_\theta(\mathbf{U}_m^n) + \hat{f}_\theta(\mathbf{U}_m^n, x, t)\right) \right\| \right\}$$

Theoretical foundation for both baseline and PHNN

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{u}(t) &= \mathbf{g}(\mathbf{u}(t), t) \\ \mathbf{u}(0) &= \mathbf{u}_0, \end{cases} \tag{2}$$

Theorem (Eidnes and Lye, 2024)

Assume that $u : [0, T) \to \mathbb{R}^{N_x}$ *solves* (2) *and that* $\tilde{\mathbf{U}}^1, \ldots, \tilde{\mathbf{U}}^N \in \mathbb{R}^{N_x}$ *obey*

$$\frac{\tilde{\mathbf{U}}^{n+1} - \mathbf{U}^n}{\Delta t} = \tilde{g}(\mathbf{U}^n) \qquad \text{for } n = 0, \ldots, N_T.$$

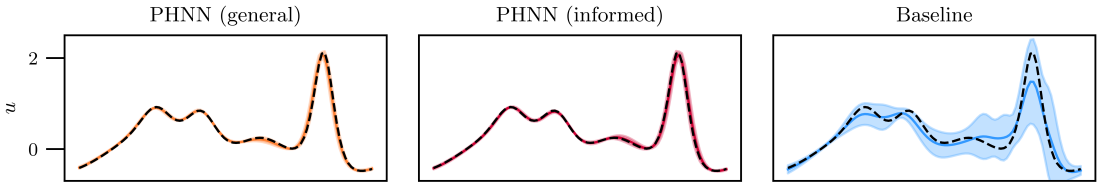
Then,

$$\left(\Delta t \sum_{n=1}^{N_T - 1} \left(g(\mathbf{u}(t_n), t_n) - \tilde{g}(\mathbf{U}^n, t_n) \right)^p \right)^{1/p} \leq \frac{1}{\Delta t} \left(\sum_{n=1}^{N_T} \Delta t \left| \mathbf{U}^n - \tilde{\mathbf{U}}^n \right|^p \right)^{1/p} + C_g \Delta t.$$

Numerical experiments

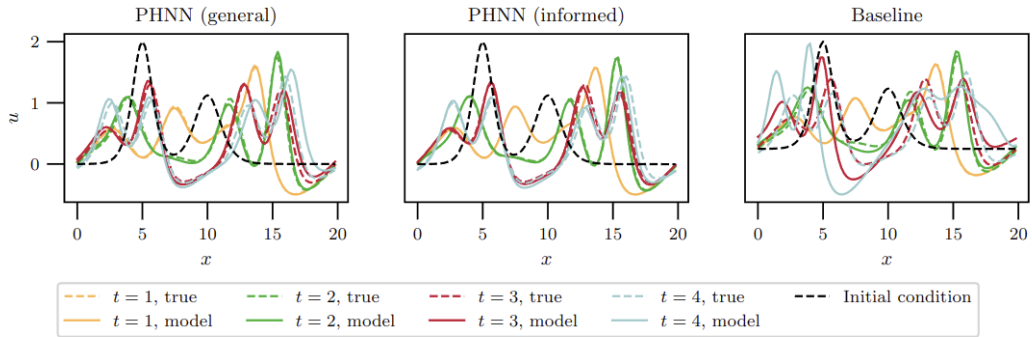
The forced KdV equation

$$u_t + \eta u u_x - \gamma^2 u_{xxx} = \frac{3}{5} \sin\left(\frac{4\pi}{P} x - t\right).$$



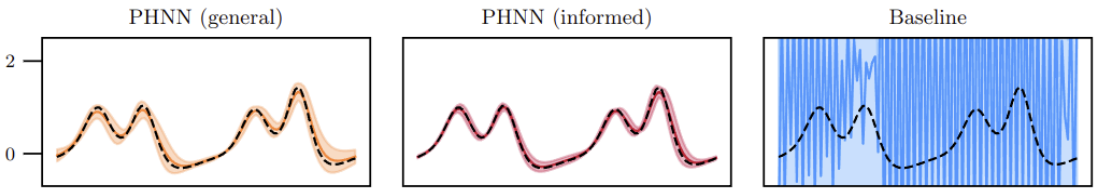
The forced KdV equation: spatial source term

$$u_t + \eta u u_x - \gamma^2 u_{xxx} = \frac{3}{5} \sin\left(\frac{4\pi}{P} x\right).$$



The forced KdV equation: spatial source term

$$u_t + \eta u u_x - \gamma^2 u_{xxx} = \frac{3}{5} \sin\left(\frac{4\pi}{P} x\right).$$



SINTEF

Limitations of PHNN and the baseline model

- Smoothness assumptions (can be relaxed)
- Requires measurements of the actual solution state u
- In practice one often has a better knowledge of the PDEs governing the system than we assumed
- Requires quite a large amount of data
- Requires measurements of the solution variable: Not always feasible in practice

Learning only parts of the equation

Balance laws with unknown source term

$$\begin{cases} u_t + f(u)_x = S(u) \\ u(x, 0) = u_0(x) \end{cases}$$

Balance laws with unknown source term

$$\begin{cases} u_t + f(u)_x = S(u) \\ u(x,0) = u_0(x) \end{cases}$$

We will discretize as

$$\frac{U^{n+1,i} - U^{n,i}}{\Delta t} + \frac{1}{\Delta x}\left(F(U^{n,i}, U^{n,i+1}) - F(U^{n,i-1}, U^{n,i})\right) - S_\theta(U^{n,i}, x_i, t_n) = 0.$$

Balance laws with unknown source term

$$\begin{cases} u_t + f(u)_x = S(u) \\ u(x,0) = u_0(x) \end{cases}$$

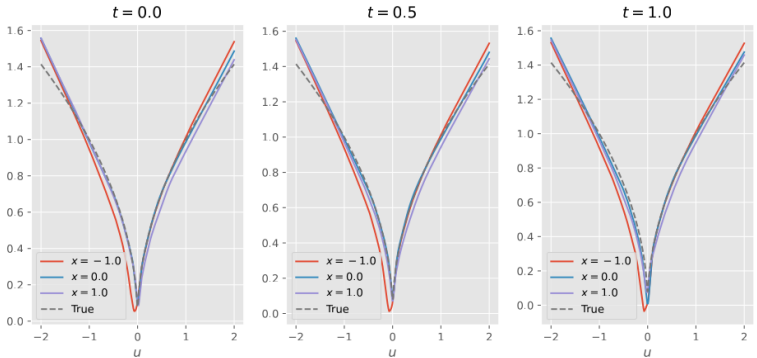
We will discretize as

$$\frac{U^{n+1,i} - U^{n,i}}{\Delta t} + \frac{1}{\Delta x}\left(F(U^{n,i}, U^{n,i+1}) - F(U^{n,i-1}, U^{n,i})\right) - S_\theta(U^{n,i}, x_i, t_n) = 0.$$

$$\mathcal{L}(\theta) = \sum_{n=1}^{N_T}\sum_{m=1}^{M}\left\| \frac{1}{\Delta t}\left(\mathbf{U}^n - \mathbf{U}_m^{n-1}\right) - \frac{1}{\Delta x}\left(F(\mathbf{U}^{n-1,i}, \mathbf{U}^{n-1,i+1}) - F(\mathbf{U}^{n-1,i-1}, \mathbf{U}^{n-1,i})\right) - S_\theta(U^{n-1,i}, x_i, t_{n-1})\right\|^p.$$

Example: Burgers' equation[2]

$$\begin{cases} u_t + (\frac{1}{2}u^2)_x = \sqrt{|u|} \\ u(x, 0) = -\sin(\pi x) \end{cases}$$



[2]Done by Bendik S. Waade, previous master student

The shallow water equations: Bottom topography[3]

$$\begin{bmatrix} h \\ hv \end{bmatrix}_t + \begin{bmatrix} hv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_x = \begin{bmatrix} 0 \\ -ghB_x \end{bmatrix}$$

[3]Done by Bendik S. Waade, previous master student

The shallow water equations: Bottom topography[3]

$$\begin{bmatrix} h \\ hv \end{bmatrix}_t + \begin{bmatrix} hv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_x = \begin{bmatrix} 0 \\ -ghB_x \end{bmatrix}$$

Assume we know the **initial data**, and for $t > 0$ have observations in **velocity** v **only** for snapshots

$$\{v_{\text{obs}}(x_i, t^n)\}_{i \in I, n=1,\ldots,N_T}.$$

[3] Done by Bendik S. Waade, previous master student

The shallow water equations: Bottom topography[3]

$$\begin{bmatrix} h \\ hv \end{bmatrix}_t + \begin{bmatrix} hv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_x = \begin{bmatrix} 0 \\ -ghB_x \end{bmatrix}$$

Assume we know the **initial data**, and for $t > 0$ have observations in **velocity** v **only** for snapshots

$$\{v_{\text{obs}}(x_i, t^n)\}_{i \in I, n=1,\ldots,N_T}.$$

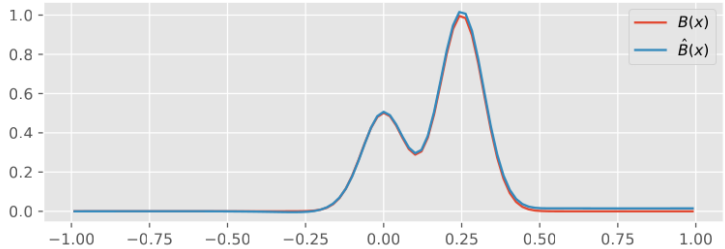
Idea: Do the full simulation with B given as a DNN, and let the loss function be

$$\mathcal{L}(\theta) = \sum_{i \in I} \sum_{n=1}^{N_T} \| v_{\text{obs}}(x_i, t^n) - v^{i,n}_{\text{simulated},\theta} \|.$$

[3] Done by Bendik S. Waade, previous master student

The shallow water equations: Bottom topography[4]

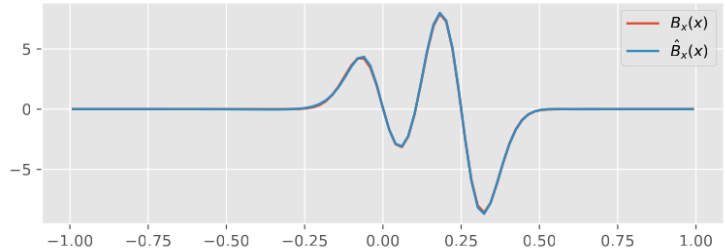
$$\begin{bmatrix} h \\ hv \end{bmatrix}_t + \begin{bmatrix} hv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_x = \begin{bmatrix} 0 \\ -ghB_x \end{bmatrix}$$



[4]Done by Bendik S. Waade, previous master student

The shallow water equations: Bottom topography[4]

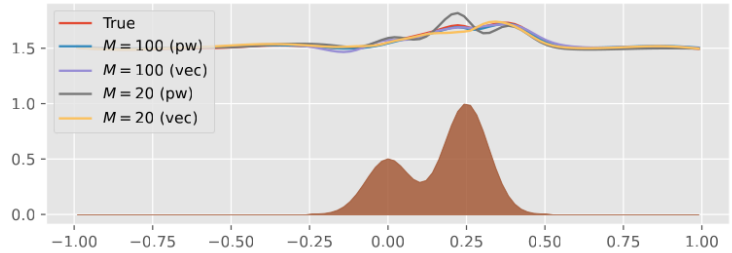
$$\begin{bmatrix} h \\ hv \end{bmatrix}_t + \begin{bmatrix} hv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_x = \begin{bmatrix} 0 \\ -ghB_x \end{bmatrix}$$



[4]Done by Bendik S. Waade, previous master student

The shallow water equations: Bottom topography[4]

$$\begin{bmatrix} h \\ hv \end{bmatrix}_t + \begin{bmatrix} hv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix}_x = \begin{bmatrix} 0 \\ -ghB_x \end{bmatrix}$$



[4]Done by Bendik S. Waade, previous master student

Conclusions and future work

- The PHNN and baseline framework.
  - Both perform well for simple cases
  - The PHNN performs better than the baseline approach
  - Have more fine-grained control with the PHNN approach
- Learning isolated part of a PDE.
  - works well, even with slightly incomplete data

Future work:

- more complicated PDEs/simulations
- **Limited data** with incomplete, noisy measurements.
  - will require a Bayesian approach

`https://github.com/SINTEF/pseudo-hamiltonian-neural-networks`

Conclusions and future work

- The PHNN and baseline framework.
  - Both perform well for simple cases
  - The PHNN performs better than the baseline approach
  - Have more fine-grained control with the PHNN approach
- Learning isolated part of a PDE.
  - works well, even with slightly incomplete data

Future work:

- more complicated PDEs/simulations
- **Limited data** with incomplete, noisy measurements.
  - will require a Bayesian approach

`https://github.com/SINTEF/pseudo-hamiltonian-neural-networks`

Thank you.

Kjetil Olsen Lye
`kjetil.olsen.lye@sintef.no`

$$\frac{\delta \mathcal{H}}{\delta u}[u] = \frac{\partial H}{\partial u} - \frac{d}{dx}\frac{\partial H}{\partial u_x},$$

$$u_t + \eta u u_x - \nu u_{xx} - \gamma^2 u_{xxx} = 0. \tag{3}$$

R the identity operator I, $S = \frac{\partial}{\partial x}$ and $f(u, x, t) = 0$, and

$$\mathcal{H} = -\int_\Omega \left(\frac{\eta}{6} u^3 + \frac{\gamma^2}{2} u_x^2 \right) dx \tag{4}$$

and

$$\mathcal{V} = \frac{\nu}{2} \int_\Omega u_x^2 \, dx. \tag{5}$$

We see this connection by deriving the variational derivatives

$$\frac{\delta \mathcal{H}}{\delta u}[u] = -(\frac{\eta}{2} u^2 - \gamma^2 u_{xx}) \tag{6}$$

and

$$\frac{\delta \mathcal{V}}{\delta u}[u] = -\nu u_{xx}. \tag{7}$$