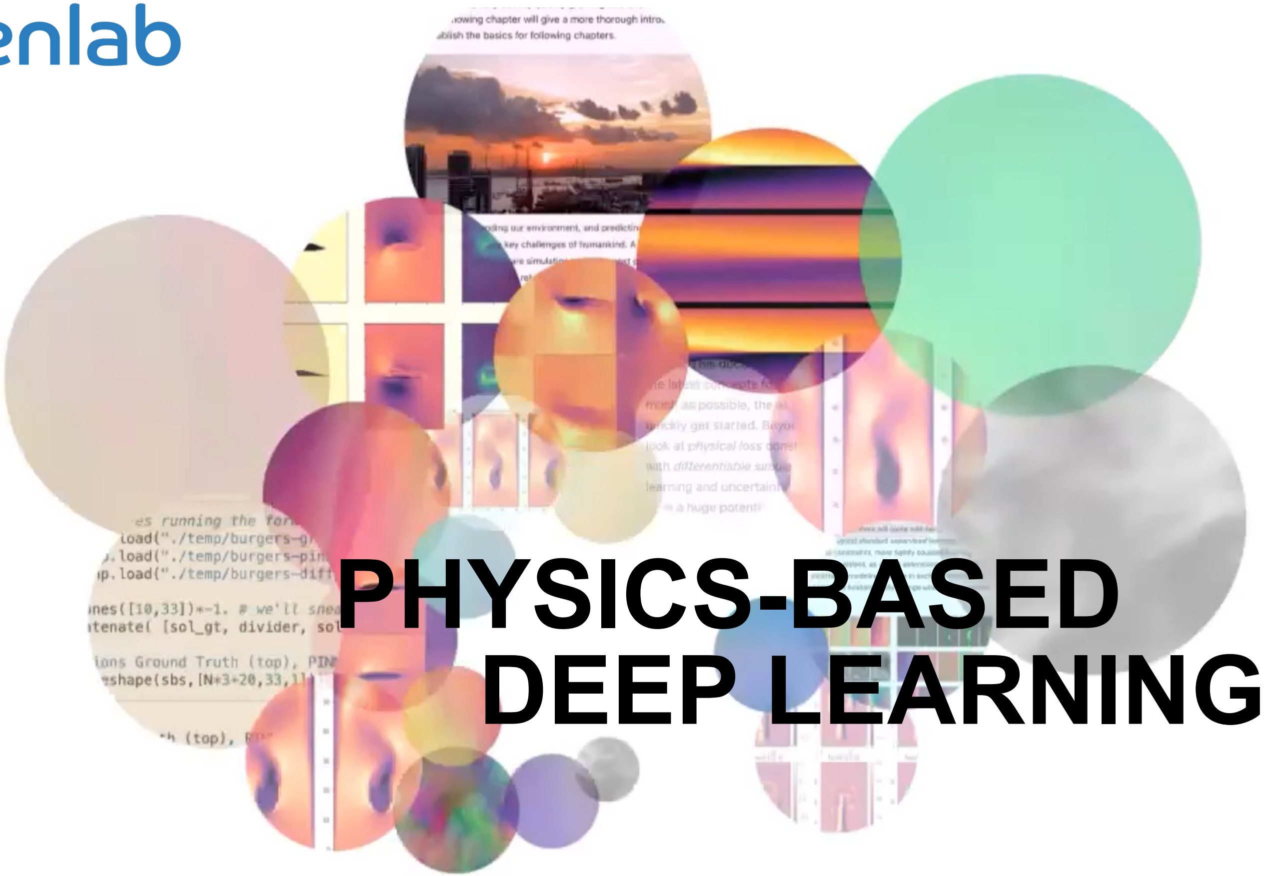
CERN openlab

CERN



PHYSICS-BASED DEEP LEARNING

Peter Kicsiny | CERN Openlab Summer Student Lectures | 15th July 2024

Scope

What is this lecture about?

- Introduction to some of the main concepts in the field
- Walkthrough of the concepts using toy examples
- Some application highlights

What is not covered in this lecture?

- Introduction of numerical simulations / differential equations
- Introduction to machine learning

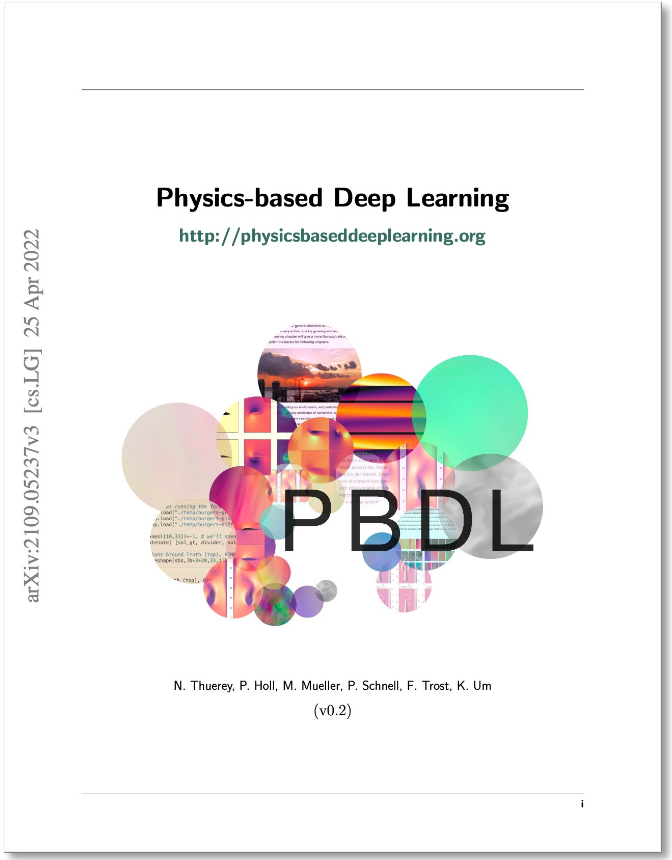
Literature

The lecture will cover:

1. Surrogate models
2. Physics informed neural networks
3. Differentiable physics

Content based on:
Physics-based Deep Learning
by
N. Thuerey, P . Holl, M. Mueller, P. Schnell, F. Trost, K. Um

+ supplemented by real physics examples


arXiv:2109.05237v3 [cs.LG] 25 Apr 2022

Physics-based Deep Learning

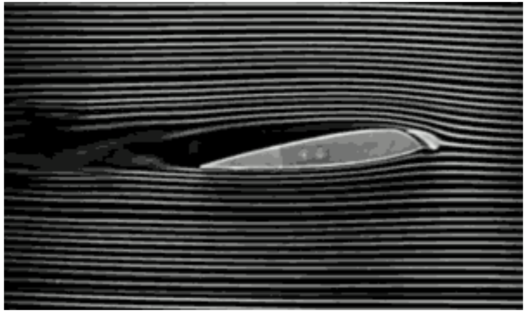
http://physicsbaseddeeplearning.org

N. Thuerey, P. Holl, M. Mueller, P. Schnell, F. Trost, K. Um

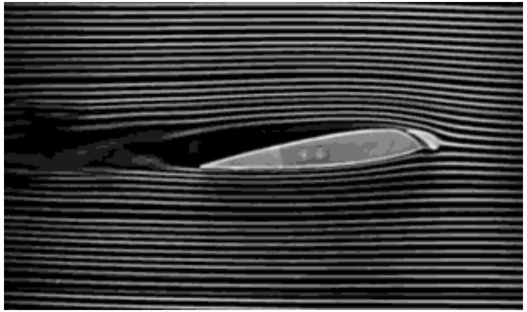
(v0.2)

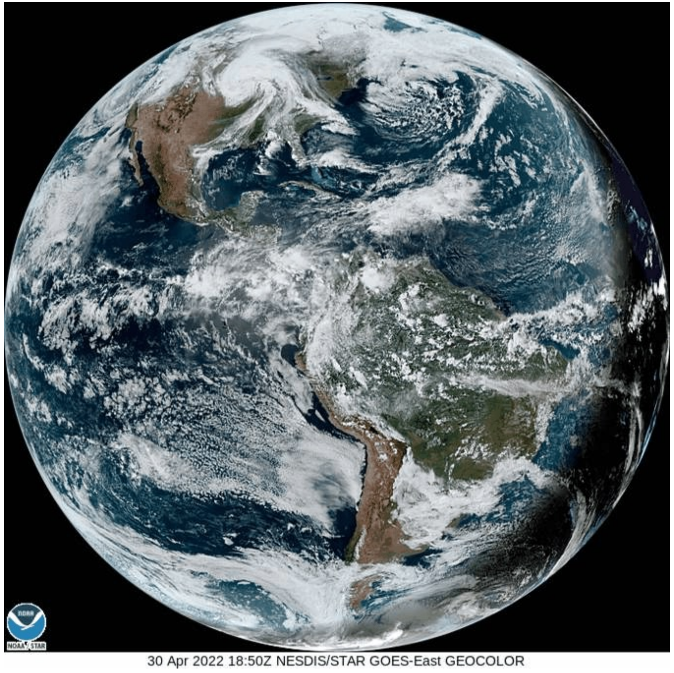
i

Applications of physics simulations

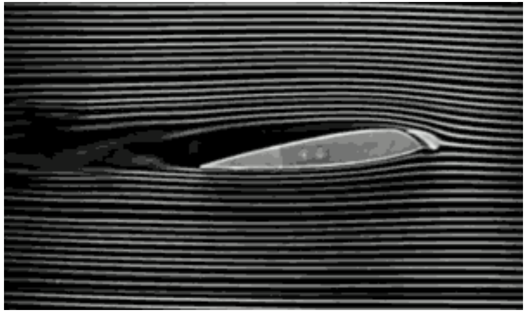


Applications of physics simulations

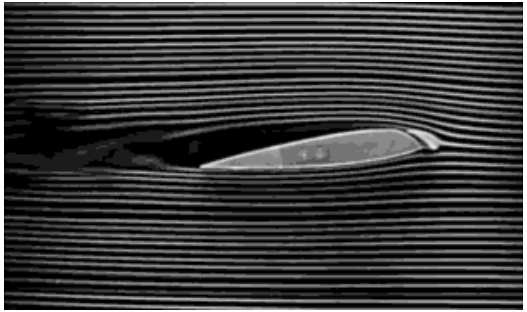


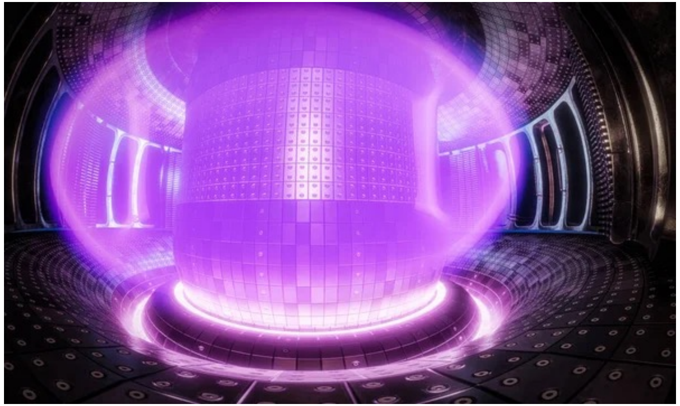


Applications of physics simulations



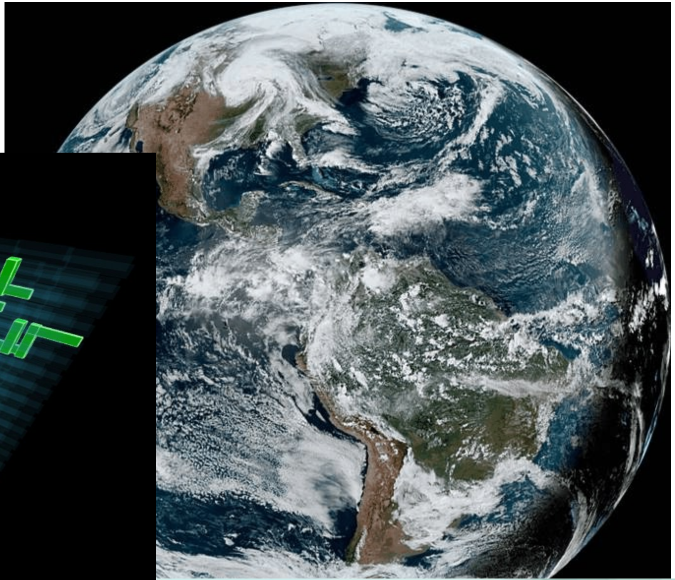
Applications of physics simulations

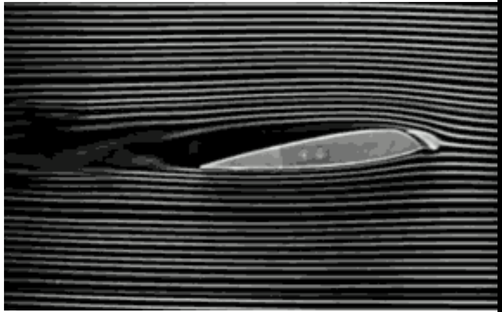


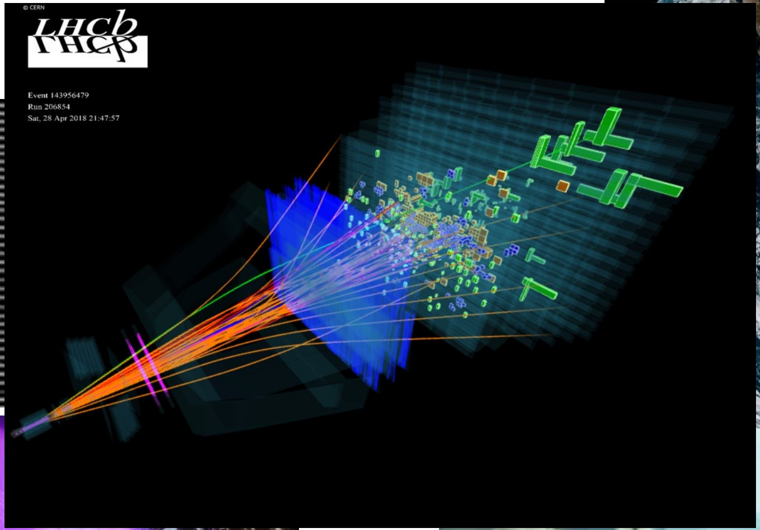


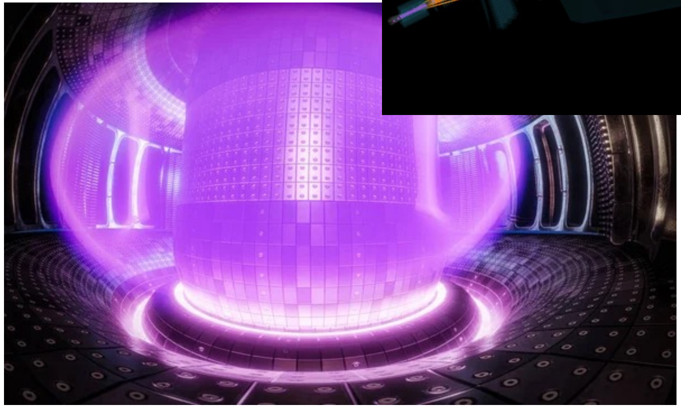
Motivation

Applications of physics simulations

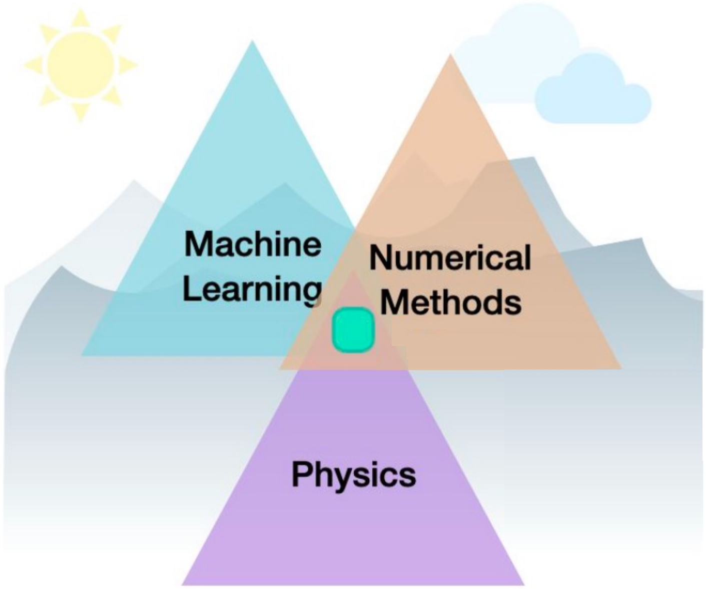






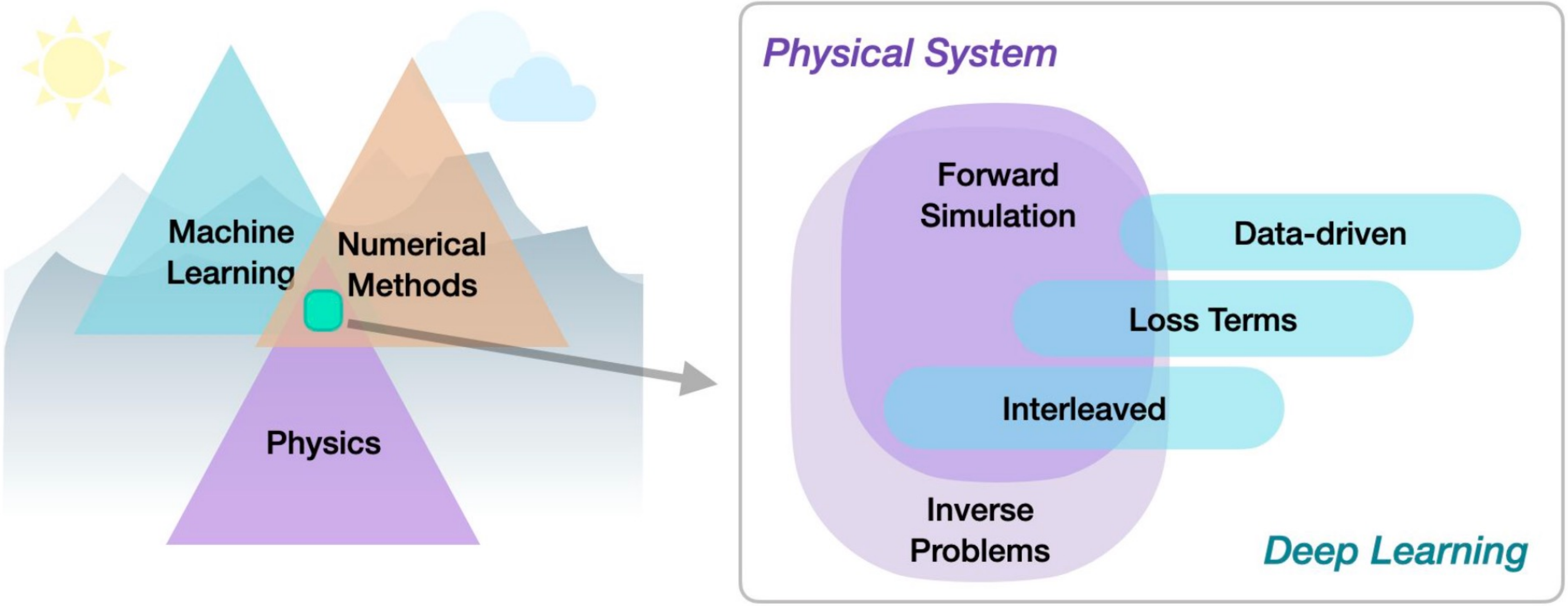


ML & numerical methods: new possibilities



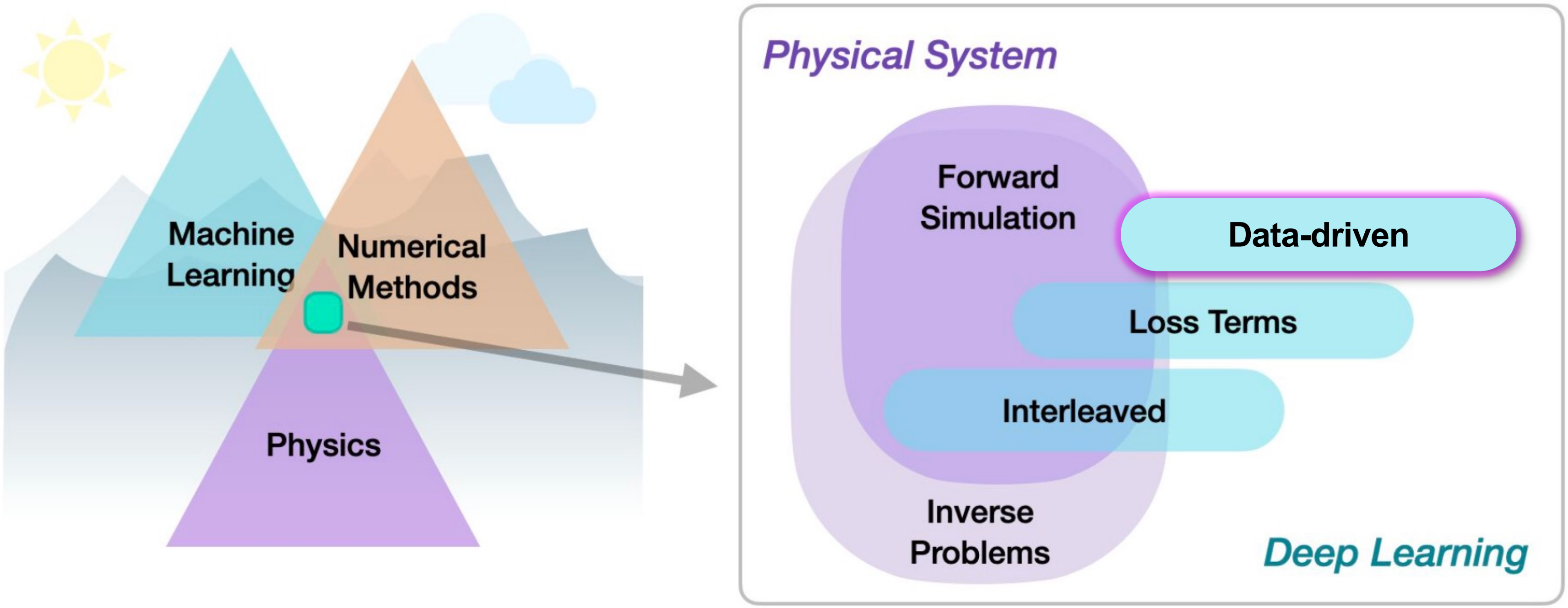
Goal: make classical simulations faster & more accurate

ML & numerical methods: new possibilities



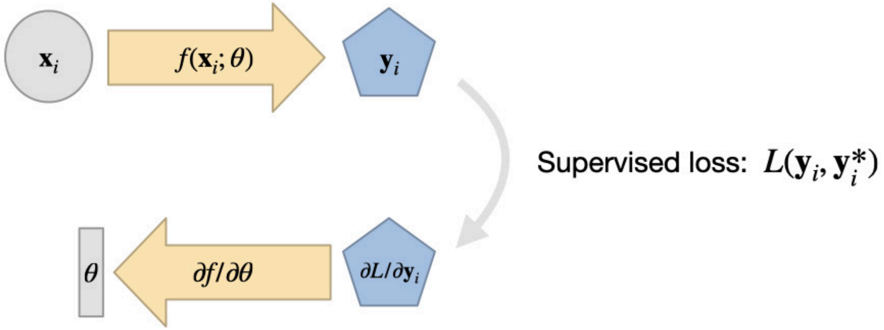
Goal: make classical simulations faster & more accurate

ML & numerical methods: new possibilities



Surrogate models: concept

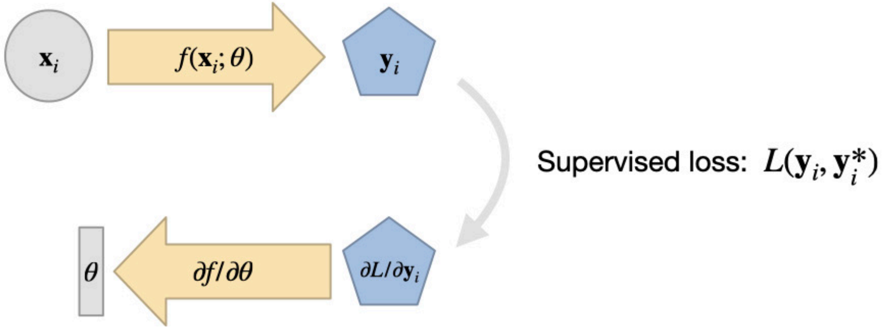
- Supervised learning



Surrogate models: concept

- Supervised learning

- Given arbitrary unknown function describing a physical system: $f^*(x) = y^*$

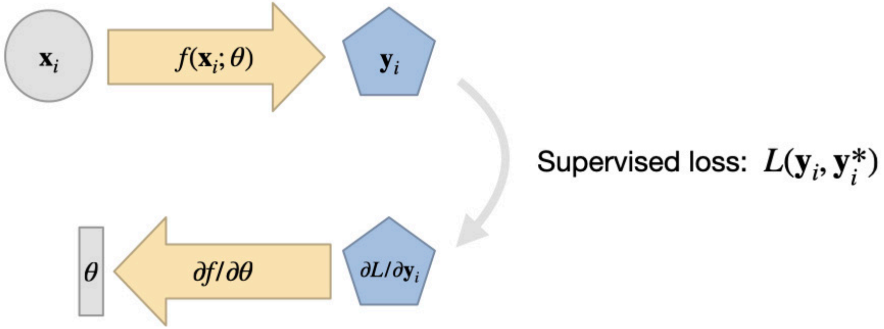


Surrogate models: concept

- Supervised learning

- Given arbitrary unknown function describing a physical system: $f^*(x) = y^*$

- Data: measured/simulated samples
  $[x_0, y_0^*], ...[x_n, y_n^*]$



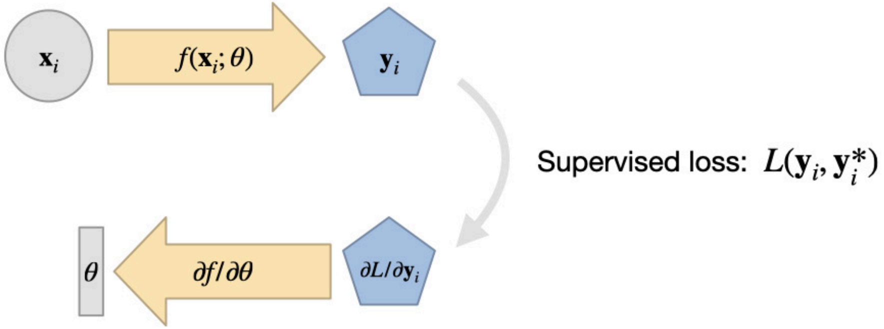
Surrogate models: concept

- Supervised learning

- Given arbitrary unknown function describing a physical system: $f^*(x) = y^*$

- Data: measured/simulated samples
  $[x_0, y_0^*], ...[x_n, y_n^*]$

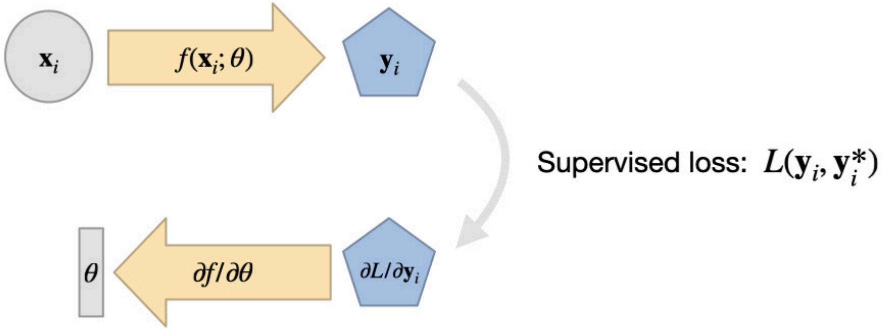
- Goal: approximate f^* with a neural network (NN) denoted by f, trained on this data



Surrogate models: concept

- Supervised learning
- Given arbitrary unknown function describing a physical system: $f^*(x) = y^*$
- Data: measured/simulated samples

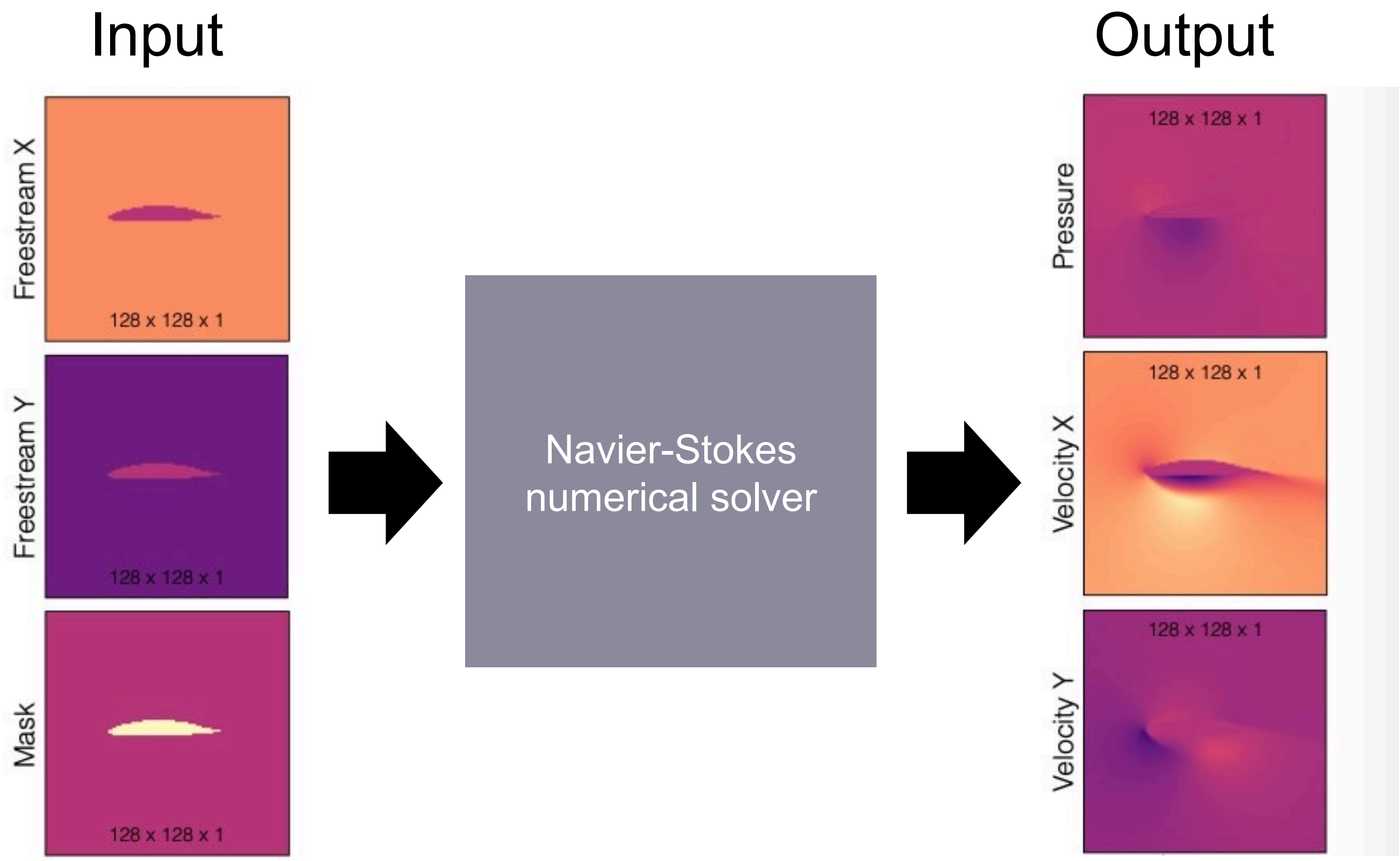
  $[x_0, y_0^*], ...[x_n, y_n^*]$
- Goal: approximate f^* with a neural network (NN) denoted by f, trained on this data
- Evaluate loss function & optimize weights of NN via backpropagation



$$L = \sum_i (f(x_i; \theta) - y_i^*)^2$$

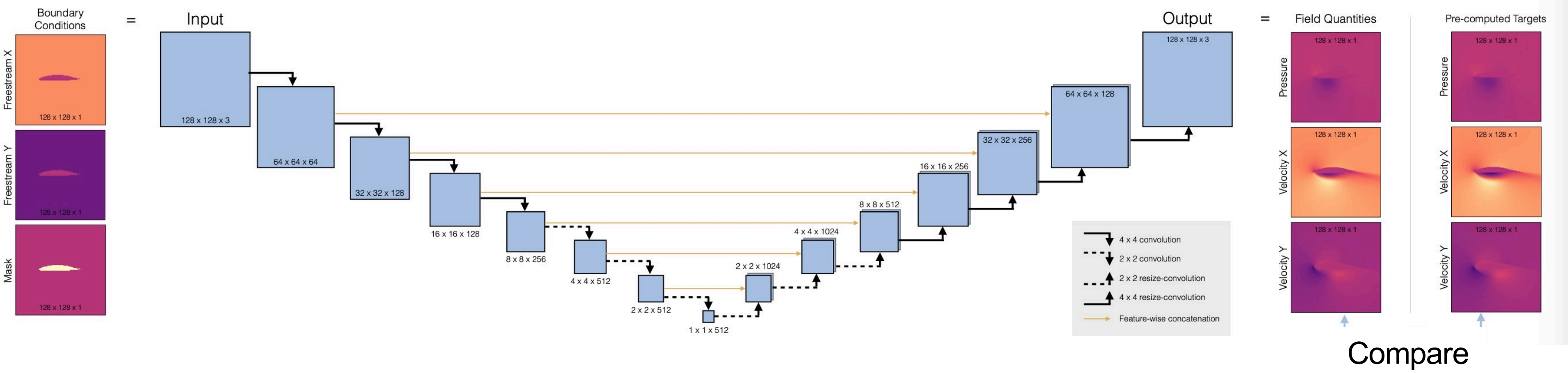
$$\underset{\theta}{\operatorname{argmin}} \sum_i (f(x_i; \theta) - y_i^*)^2$$

Surrogate models: example



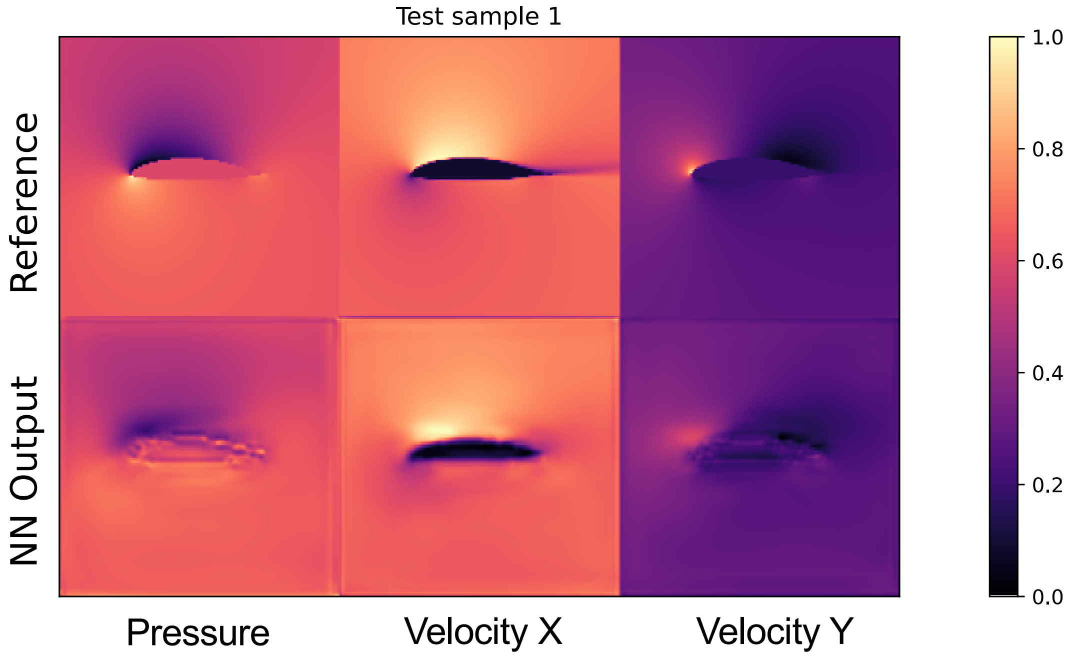
Surrogate models: example

$$x_i = [f_x, f_y, \mathrm{mask}]_i \qquad f(x_i;\theta) \qquad y_i^* = [p, u_x, u_y]_i$$



$$\underset{\theta}{\mathrm{argmin}} \sum_i (f(x_i;\theta) - y_i^*)^2$$

Surrogate models: example



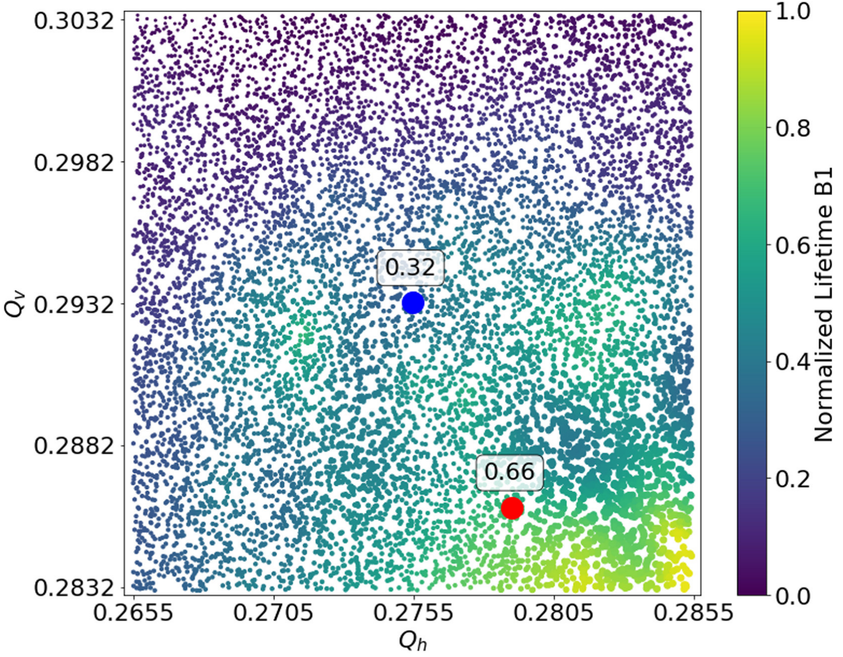
- Reconstructing pressure seems to be the most challenging
- Error on few % level

Surrogate models: best practices

- Supervised training setup is a good first approach in many situations
- Always start with a 1-sample overfitting test
- Check how many trainable parameters your network has
- Slowly increase the amount of training data (and potentially network parameters and depth)
- Adjust hyperparameters (especially the learning rate)
- For any structured data, like spatial functions, or data of any physical field, convolutional NNs are preferable to fully connected NNs

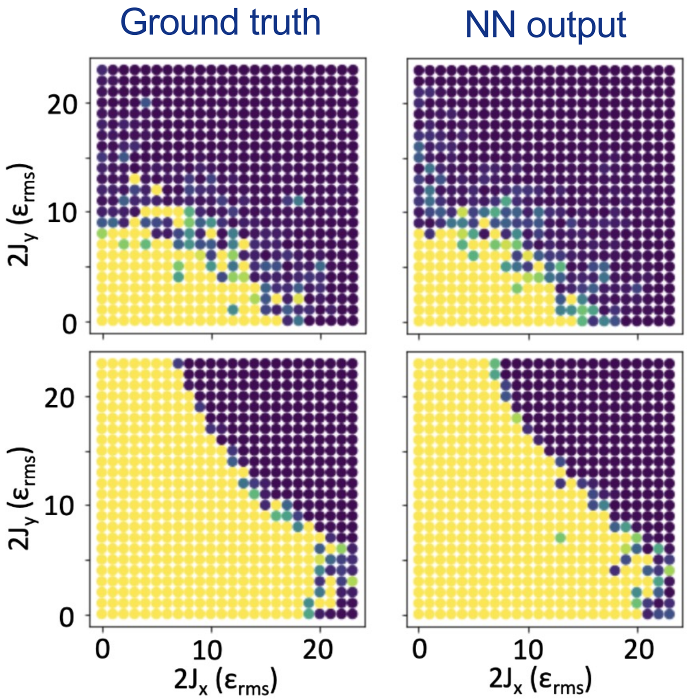
Surrogate models: CERN applications

- Surrogate modeling of LHC beam lifetime
  - Machine learning for beam dynamics studies at the CERN Large Hadron Collider, P. Arpaia et al. [10.1016/j.nima.2020.164652]
  - Data: lifetime **measurements** from real-life LHC parameter scans



Surrogate models: CERN applications

- Surrogate modeling of LHC dynamic aperture
  - Modeling Particle Stability Plots for Accelerator Optimization Using Adaptive Sampling, M. Schenk et al. [[10.18429/JACoW-IPAC2021-TUPAB216]](10.18429/JACoW-IPAC2021-TUPAB216)
  - Data: **numerical simulations** of different LHC configurations



Colors: how many turns the particles survive

Surrogate models: pros & cons

- Fast training & constant time evaluation compared to numerical solvers or measurements

- Lots of data needed

Surrogate models: pros & cons

- Fast training & constant time evaluation compared to numerical solvers or measurements
- Simple network architecture

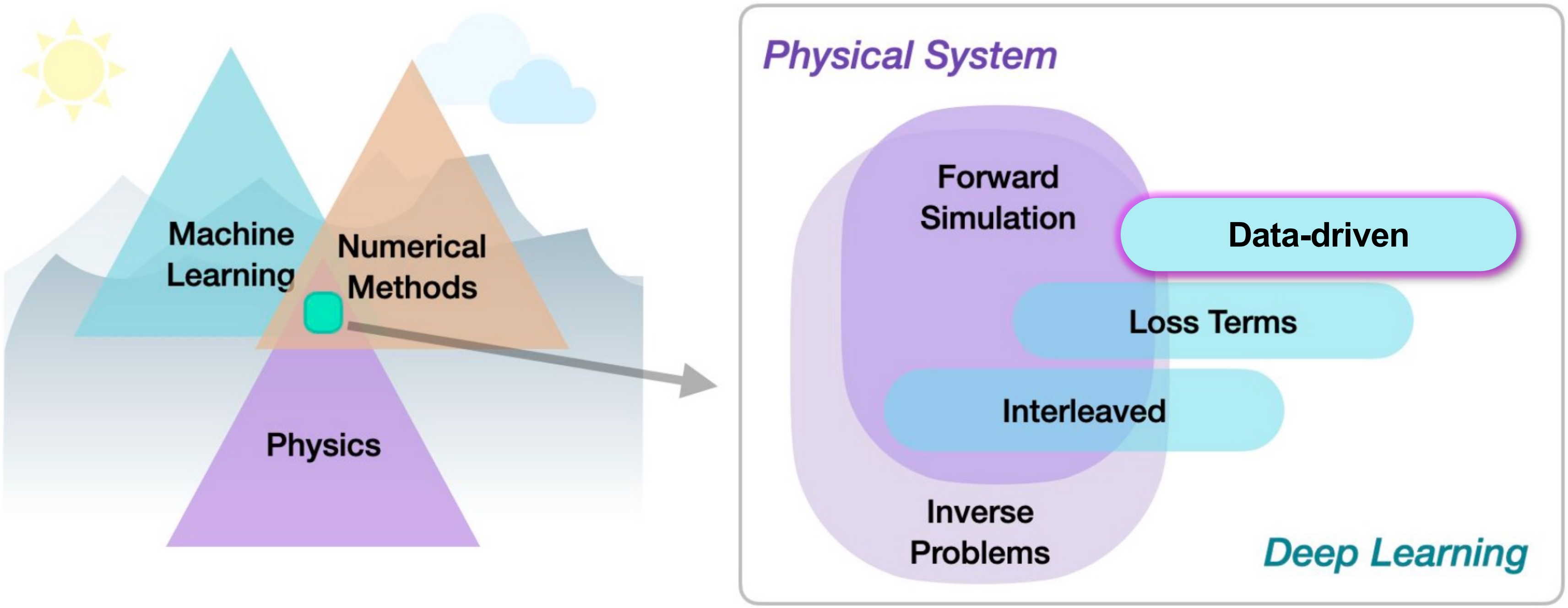
- Lots of data needed
- Sub-optimal performance, accuracy and generalization

Surrogate models: pros & cons

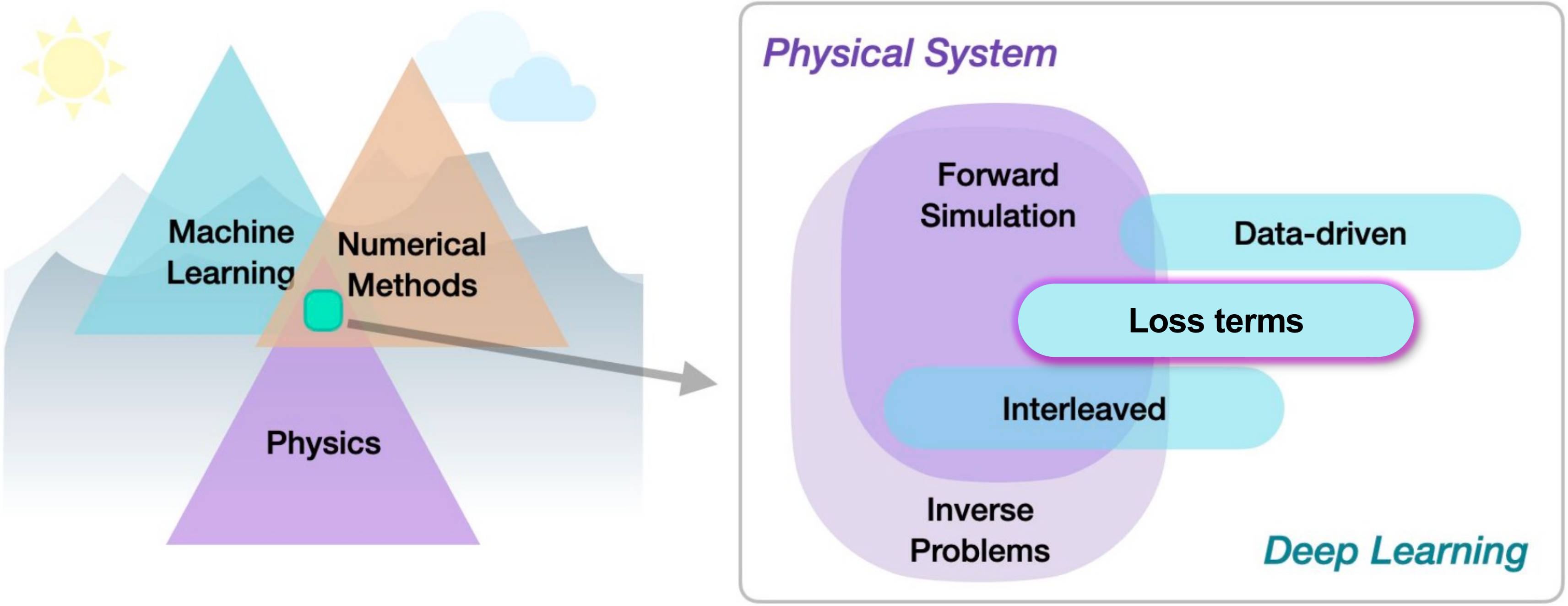
- Fast training & constant time evaluation compared to numerical solvers or measurements
- Simple network architecture
- Simple concept

- Lots of data needed
- Sub-optimal performance, accuracy and generalization
- Purely data driven. Interactions with external “processes” (such as embedding into a solver for refining results) are difficult

ML & numerical methods: new possibilities



ML & numerical methods: new possibilities



Physical losses: PDEs

- Partial differential equation (PDE)
- Time evolution of a physical field e.g. velocity expressed with spatial derivatives

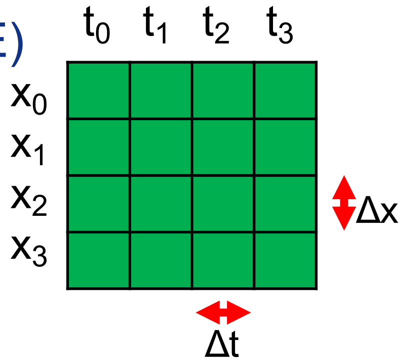
$$\mathbf{u}_t = \mathcal{F}(\mathbf{u}_x, \mathbf{u}_{xx}, \dots \mathbf{u}_{xx\dots x}) \qquad \mathbf{u} = \mathbf{u}(\mathbf{x}, t) \qquad \mathbf{x}, \mathbf{u} \in \mathbb{R}^N$$

N dimensional space

- Initial condition: $\mathbf{u}(\mathbf{x}, t=0) = \mathbf{u}^0(\mathbf{x})$

$$t \in \mathbb{R}^+$$

- Boundary condition: $\mathbf{u}(\mathbf{x} = \partial\mathbf{x}, t) = 0$
- With this (usually) a unique solution for **u** exists (=well-posed PDE)
- Space-time domain is discretized into a **computational grid**



Physical losses: Burger's equation

- Partial differential equation (PDE)

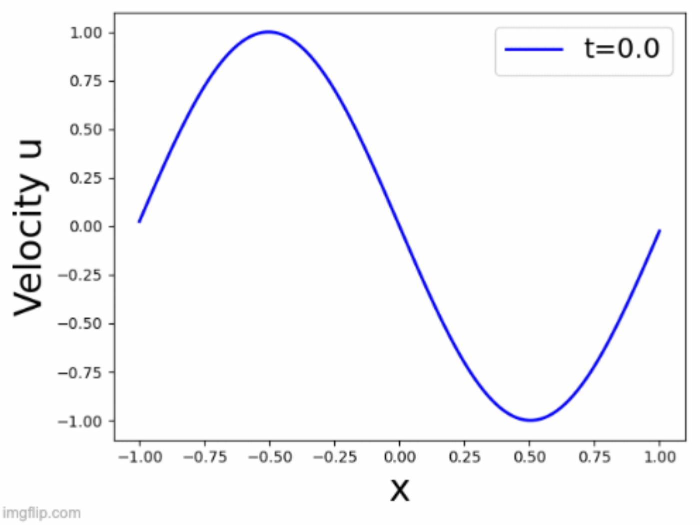
$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \qquad \begin{matrix} x, u \in \mathbb{R} \\ t \in \mathbb{R}^+ \end{matrix}$$

Initial condition	**Boundary condition**
$u(x, t=0) = -\sin(\pi x)$	$u(x = \partial x, t) = 0$

Physical losses: Burger's equation

- Partial differential equation (PDE)

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \qquad \begin{matrix} x, u \in \mathbb{R} \\ t \in \mathbb{R}^+ \end{matrix}$$

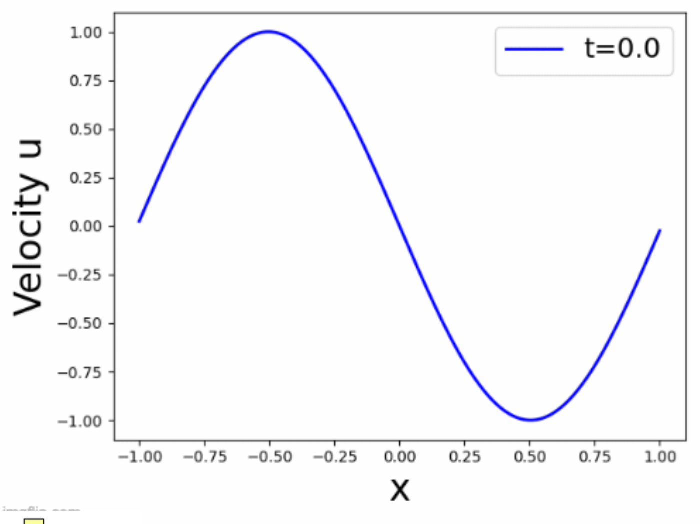


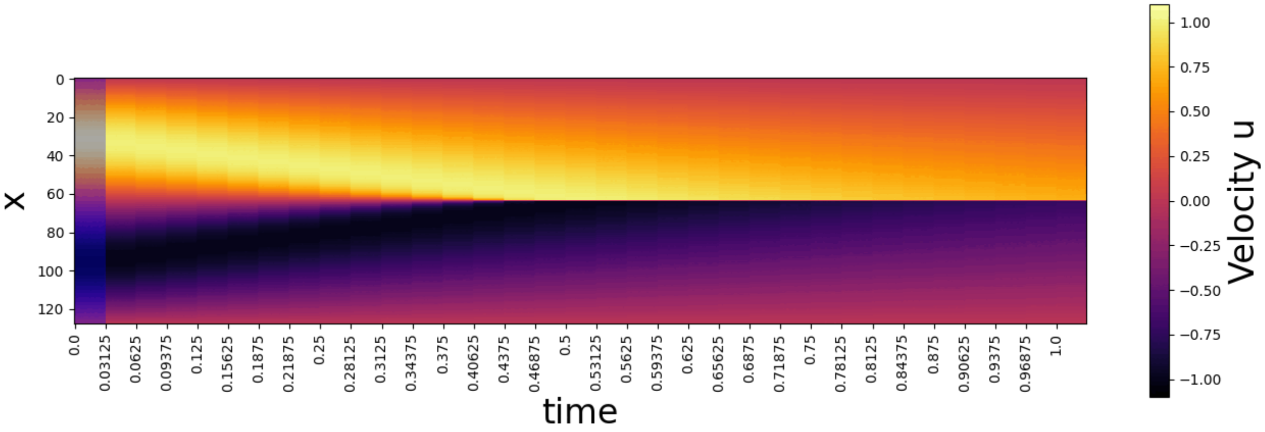
Initial condition	Boundary condition
$u(x, t=0) = -\sin(\pi x)$	$u(x = \partial x, t) = 0$

Physical losses: Burger's equation

- Partial differential equation (PDE)

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \qquad \begin{matrix} x, u \in \mathbb{R} \\ t \in \mathbb{R}^+ \end{matrix}$$





Physical losses: residual loss

Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

- Approximate velocity field ***u*** with a NN ***f***

$$f(X_i, \theta) \approx u(X_i) \qquad X_i = [x_i, t_i]$$

Physical losses: residual loss

- Approximate velocity field ***u*** with a NN ***f***

$$f(X_i, \theta) \approx u(X_i) \qquad X_i = [x_i, t_i]$$

Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

$$0 = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu \frac{\partial^2 u}{\partial x^2}$$

Physical losses: residual loss

- Approximate velocity field ***u*** with a NN ***f***

$$f(X_i, \theta) \approx u(X_i) \qquad X_i = [x_i, t_i]$$

Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

$$0 = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2}$$

Residual loss

$$R = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2}$$

Physical losses: residual loss

- Approximate velocity field ***u*** with a NN ***f***

$$f(X_i, \theta) \approx u(X_i) \qquad X_i = [x_i, t_i]$$

➢ Physics informed neural network (PINN)

$$L(X_i;\theta) = \sum_i \overbrace{\alpha_0 \left(f(X_i;\theta) - y_i^*\right)^2}^{\text{supervised loss term}} + \overbrace{\alpha_1 R(X_i;\theta)}^{\text{residual loss term}}$$

Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

$$0 = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu \frac{\partial^2 u}{\partial x^2}$$

Residual loss

$$R = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu \frac{\partial^2 u}{\partial x^2}$$

replace ***u*** by ***f***

$$R = \frac{\partial f(X_i;\theta)}{\partial t} + f(X_i;\theta)\frac{\partial f(X_i;\theta)}{\partial x} - \nu \frac{\partial^2 f(X_i;\theta)}{\partial x^2}$$

Physical losses: obtaining partial derivatives

sample grid point

$$X_i = [x_i, t_i]$$

$$R = \frac{\partial f(X_i;\theta)}{\partial t} + f(X_i;\theta)\frac{\partial f(X_i;\theta)}{\partial x} - \nu\frac{\partial^2 f(X_i;\theta)}{\partial x^2}$$

$$L(X_i;\theta) = \sum_i \alpha_0 \big(f(X_i;\theta) - y_i^*\big)^2 + \alpha_1 R(X_i;\theta)$$

$$\Downarrow$$

$$\operatorname*{argmin}_\theta L(X_i;\theta)$$

$$\Downarrow$$

$$\operatorname*{argmin}_\theta \sum_i \alpha_0 \big(f(X_i;\theta) - y_i^*\big)^2 + \alpha_1 R(X_i;\theta)$$

Physical losses: obtaining partial derivatives

$$R = \frac{\partial f(X_i;\theta)}{\partial t} + f(X_i;\theta)\frac{\partial f(X_i;\theta)}{\partial x} - \nu\frac{\partial^2 f(X_i;\theta)}{\partial x^2}$$

$$\operatorname*{argmin}_{\theta} \sum_i \alpha_0 \big(f(X_i;\theta) - y_i^*\big)^2 + \alpha_1 R(X_i;\theta)$$

- Derivatives in ***R*** are obtained from the NN via standard backpropagation

Physical losses: obtaining partial derivatives

$$R = \frac{\partial f(X_i;\theta)}{\partial t} + f(X_i;\theta)\frac{\partial f(X_i;\theta)}{\partial x} - \nu\frac{\partial^2 f(X_i;\theta)}{\partial x^2}$$

$$\underset{\theta}{\operatorname{argmin}} \sum_i \alpha_0\big(f(X_i;\theta) - y_i^*\big)^2 + \alpha_1 R(X_i;\theta)$$

- Derivatives in ***R*** are obtained from the NN via standard backpropagation

NN output (***f***) at sample point [x_i, t_i]

```
In [ ]: def R(u, xi, ti):
            u_t = tf.gradients(u, ti)
            u_x = tf.gradients(u, xi)
            u_xx = tf.gradients(u_x, xi)
            return u_t + u*u_x - nu * u_xx
```

Physical losses: obtaining partial derivatives

$$R = \frac{\partial f(X_i;\theta)}{\partial t} + f(X_i;\theta)\frac{\partial f(X_i;\theta)}{\partial x} - \nu \frac{\partial^2 f(X_i;\theta)}{\partial x^2}$$

$$\operatorname*{argmin}_\theta \sum_i \alpha_0 \left(f(X_i;\theta) - y_i^*\right)^2 + \alpha_1 R(X_i;\theta)$$

- Derivatives in ***R*** are obtained from the NN via standard backpropagation

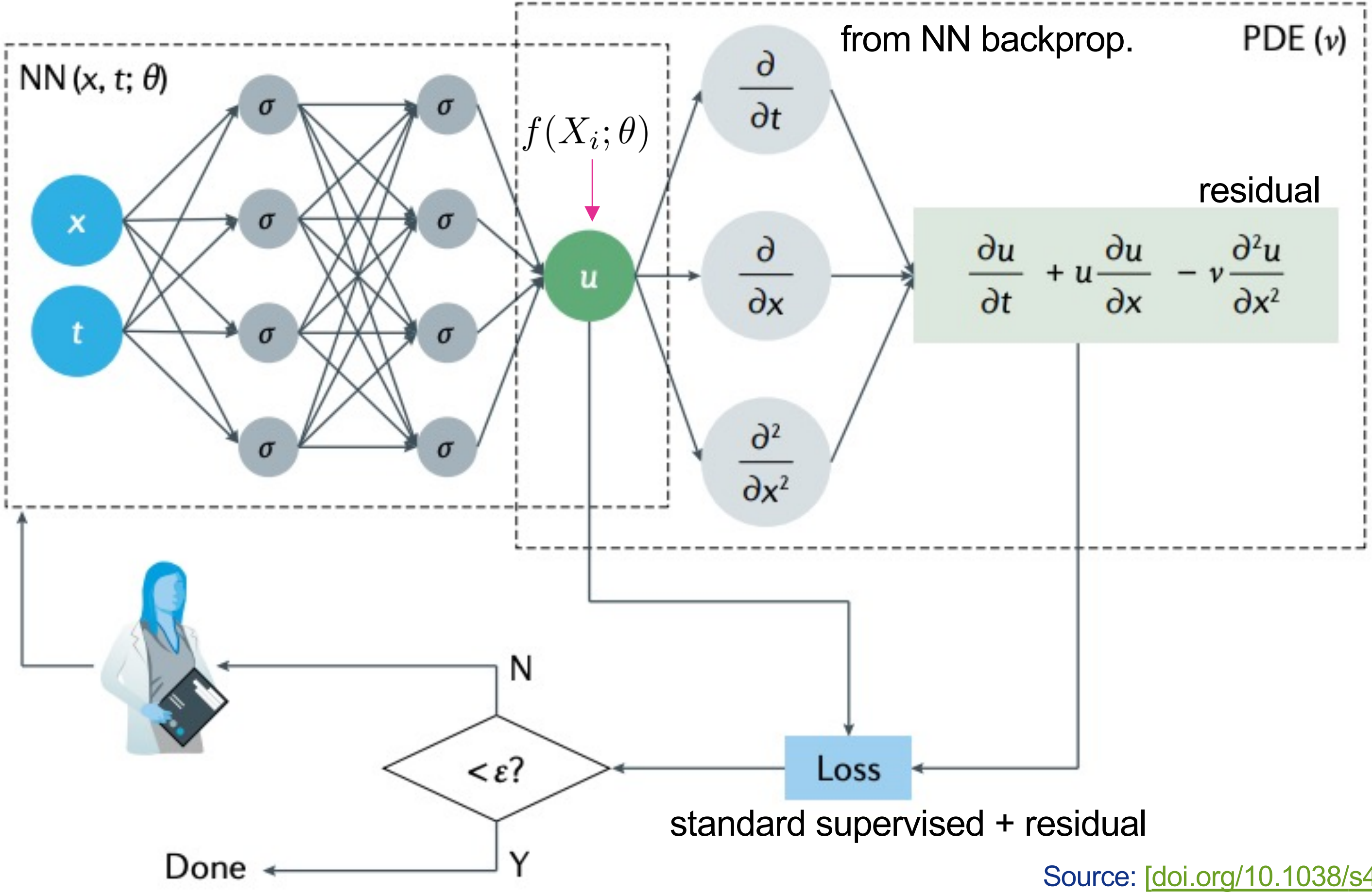
NN output (***f***) at sample point [x_i, t_i]

```
In [ ]: def R(u, xi, ti):
            u_t = tf.gradients(u, ti)
            u_x = tf.gradients(u, xi)
            u_xx = tf.gradients(u_x, xi)
            return u_t + u*u_x - nu * u_xx
```

- When R is minimized: ***u*** (NN output) approximately solves the PDE

$$f(X_i;\theta) \approx u(X_i) \qquad X_i = [x_i, t_i]$$

Physical losses: NN training overview



Source: [doi.org/10.1038/s42254-021-00314-5]

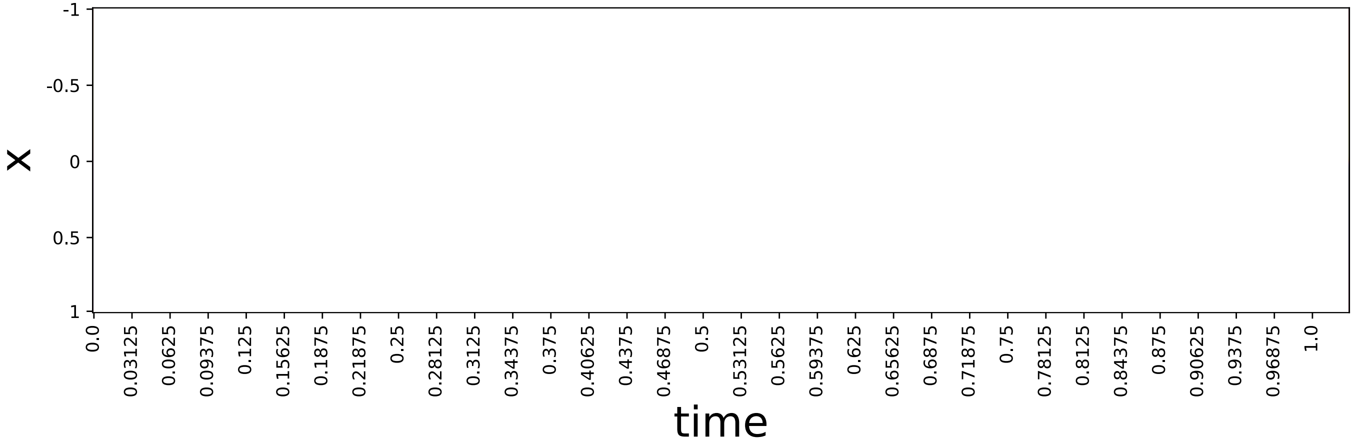
Physical losses: solving Burger's equation

find ***u*** with a PINN

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

1D computational "grid": $x \in [-1, 1]$ 128 steps

Forward simulation in time: $t \in [0, 1]$ 33 steps



Physical losses: solving Burger's equation

- Supervised loss: ground truth data $\boldsymbol{y_i^*}$
  - Reference $\boldsymbol{u}$

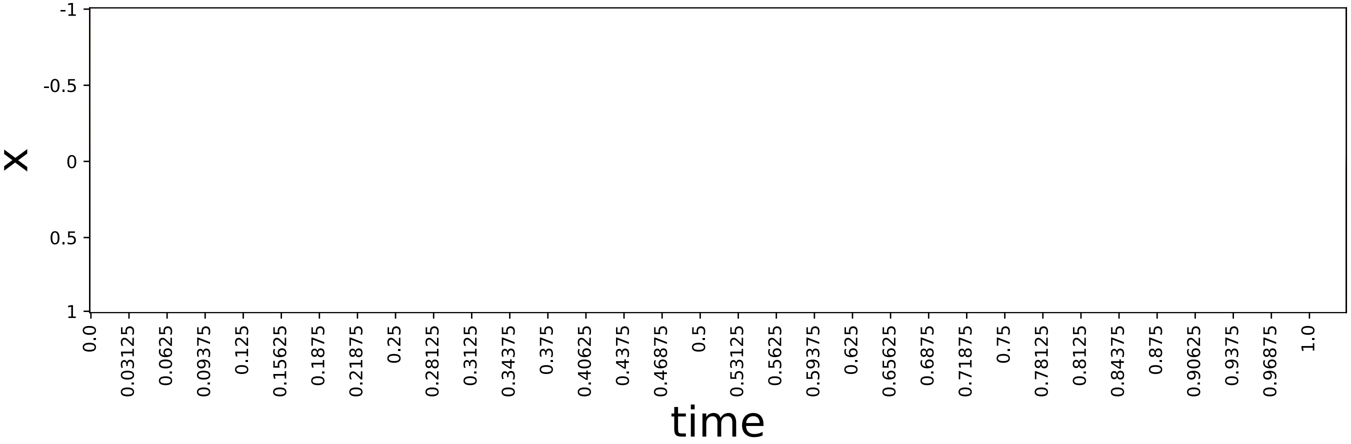
supervised loss term

$$\underset{\theta}{\operatorname{argmin}} \sum_i \boxed{\alpha_0 \big(f(X_i;\theta) - y_i^*\big)^2} + \alpha_1 R(X_i;\theta)$$

$$X_i = [x_i, t_i]$$

1D computational "grid": $x \in [-1, 1]$ 128 steps

Forward simulation in time: $t \in [0, 1]$ 33 steps



Physical losses: solving Burger’s equation

- Supervised loss: ground truth data y_i^*
  - Reference ***u***
  - E.g. we know u(x, t=0.5) (direct constraint)

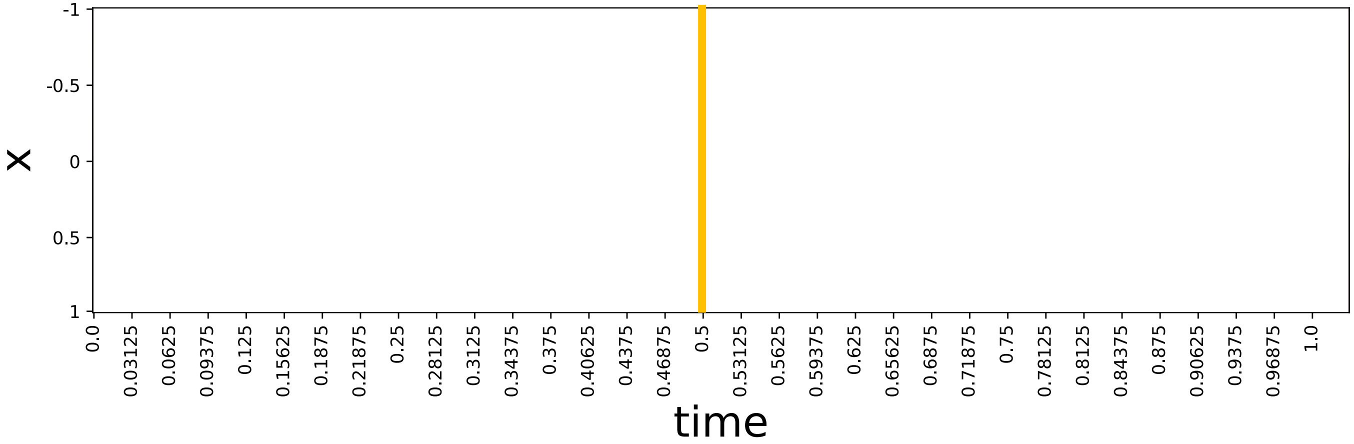
supervised loss term

$$\underset{\theta}{\operatorname{argmin}} \sum_i \boxed{\alpha_0 \left(f(X_i;\theta) - y_i^* \right)^2} + \alpha_1 R(X_i;\theta)$$

$$X_i = [x_i, t_i]$$

$x \in [-1, 1]$ 128 steps

$t \in [0, 1]$ 33 steps



Physical losses: solving Burger's equation

- Supervised loss: ground truth data $\boldsymbol{y_i^*}$
  - Reference ***u***
  - E.g. we know u(x, t=0.5) (direct constraint)
  - We also know u(x=-1;1, t)=0 (boundary conditions of PDE)

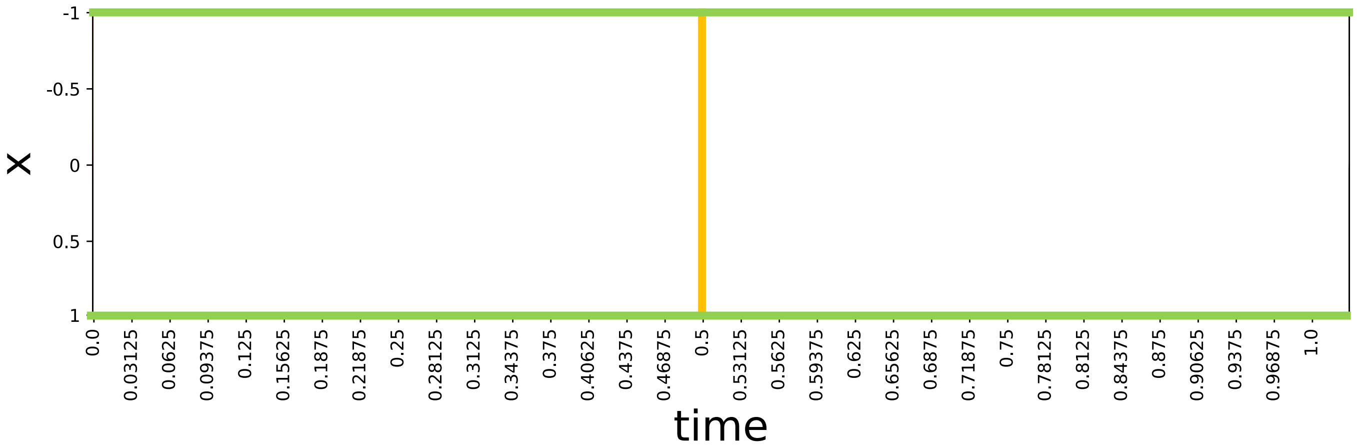
supervised loss term

$$\arg\min_{\theta} \sum_i \boxed{\alpha_0 \big(f(X_i;\theta) - y_i^*\big)^2} + \alpha_1 R(X_i;\theta)$$

$$X_i = [x_i, t_i]$$

$x \in [-1, 1]$ 128 steps

$t \in [0, 1]$ 33 steps



Physical losses: solving Burger's equation

- Supervised loss: ground truth data y_i^*
  - Reference u
  - E.g. we know u(x, t=0.5) (direct constraint)
  - We also know u(x=-1;1, t)=0 (boundary conditions of PDE)

  y_i^*

  - "Pin down" PDE solution at these points

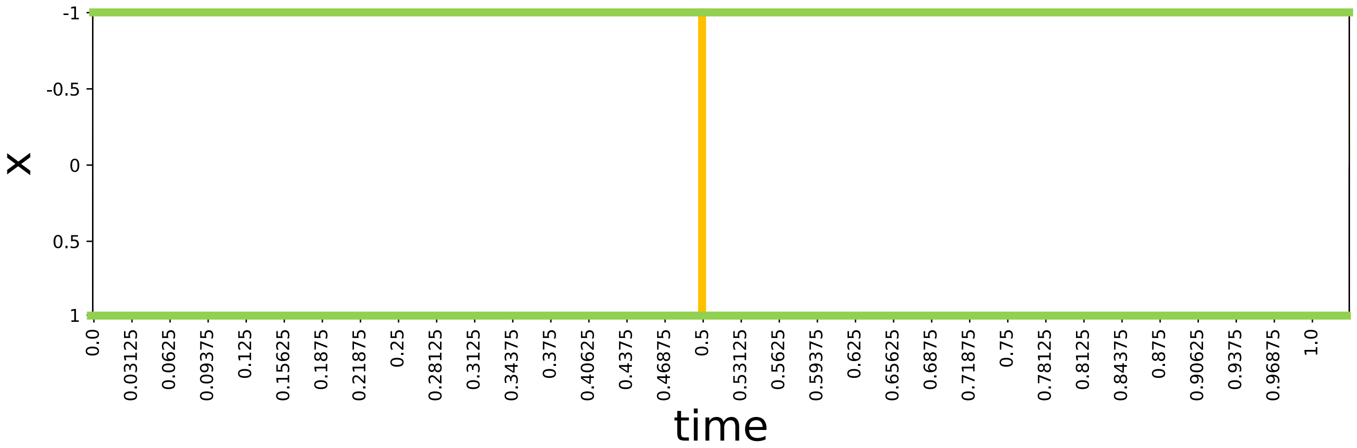
supervised loss term

$$\underset{\theta}{\text{argmin}} \sum_i \boxed{\alpha_0 \big(f(X_i;\theta) - y_i^*\big)^2} + \alpha_1 R(X_i;\theta)$$

$$X_i = [x_i, t_i]$$

$x \in [-1, 1]$ 128 steps

$t \in [0, 1]$ 33 steps



Physical losses: solving Burger's equation

- Residual loss: no ground truth data, but we **sample** the the NN inside the domain

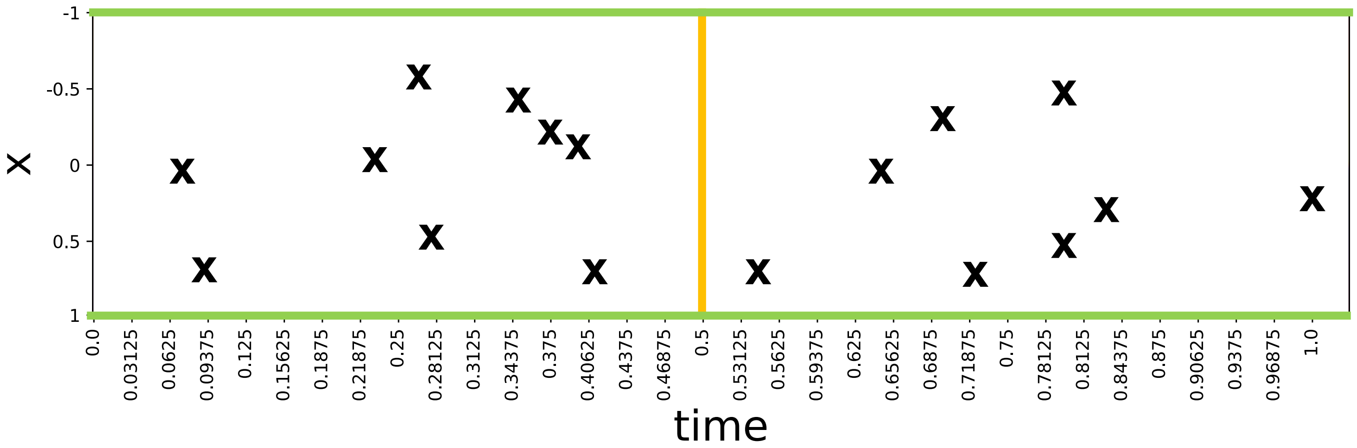
residual loss term

$$\underset{\theta}{\operatorname{argmin}} \sum_i \alpha_0 \big(f(X_i;\theta) - y_i^*\big)^2 + \boxed{\alpha_1 R(X_i;\theta)}$$

$$X_i = [x_i, t_i]$$

$x \in [-1, 1]$ 128 steps

$t \in [0, 1]$ 33 steps



Physical losses: solving Burger's equation

residual loss term

$$\underset{\theta}{\operatorname{argmin}} \sum_i \alpha_0 \left(f(X_i;\theta) - y_i^*\right)^2 + \boxed{\alpha_1 R(X_i;\theta)}$$

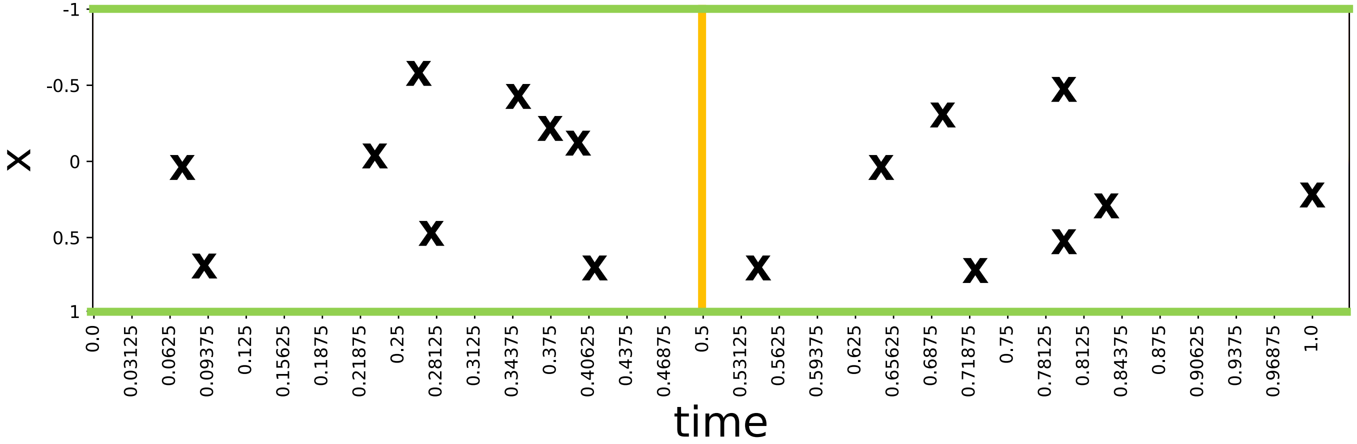
$$X_i = [x_i, t_i]$$

- Residual loss: no ground truth data, but we **sample** the the NN inside the domain
  - i.e.: input random X_i to NN
  - NN estimates u as $f(X_i;\theta)$

$$R = \frac{\partial f(X_i;\theta)}{\partial t} + f(X_i;\theta)\frac{\partial f(X_i;\theta)}{\partial x} - \nu \frac{\partial^2 f(X_i;\theta)}{\partial x^2}$$

$x \in [-1, 1]$ 128 steps

$t \in [0, 1]$ 33 steps



Physical losses: solving Burger's equation

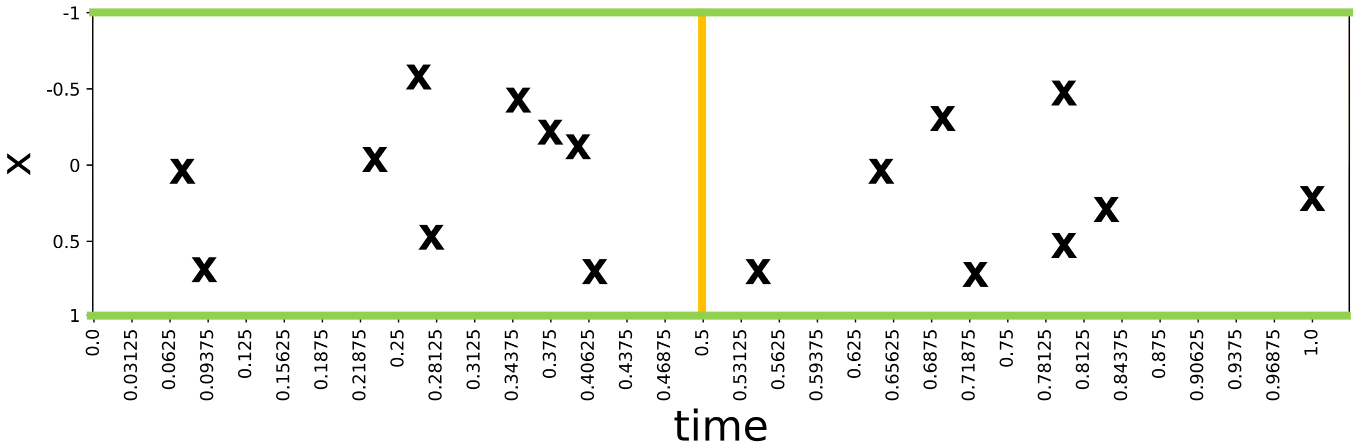
- NN in this example:
  - 8 fully connected layers
  - ~200 trainable parameters

$$\underset{\theta}{\operatorname{argmin}} \sum_i \alpha_0 \left(f(X_i;\theta) - y_i^*\right)^2 + \alpha_1 R(X_i;\theta)$$

$$X_i = [x_i, t_i]$$

$x \in [-1, 1]$ 128 steps

$t \in [0, 1]$ 33 steps



Physical losses: solving Burger's equation

- NN in this example:
  - 8 fully connected layers
  - ~200 trainable parameters

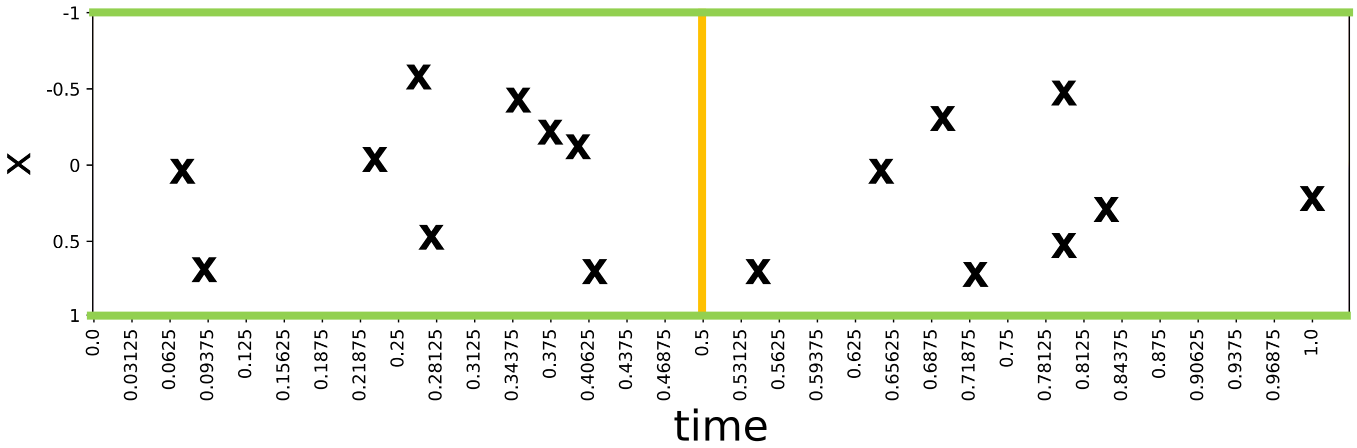
$$\underset{\theta}{\operatorname{argmin}} \sum_i \alpha_0 \left(f(X_i;\theta) - y_i^*\right)^2 + \alpha_1 R(X_i;\theta)$$

$$X_i = [x_i, t_i]$$

- Evaluate NN at constraint points |, | & **x**
- Calculate loss & update weights
- Repeat 10,000 iterations (~1 min)

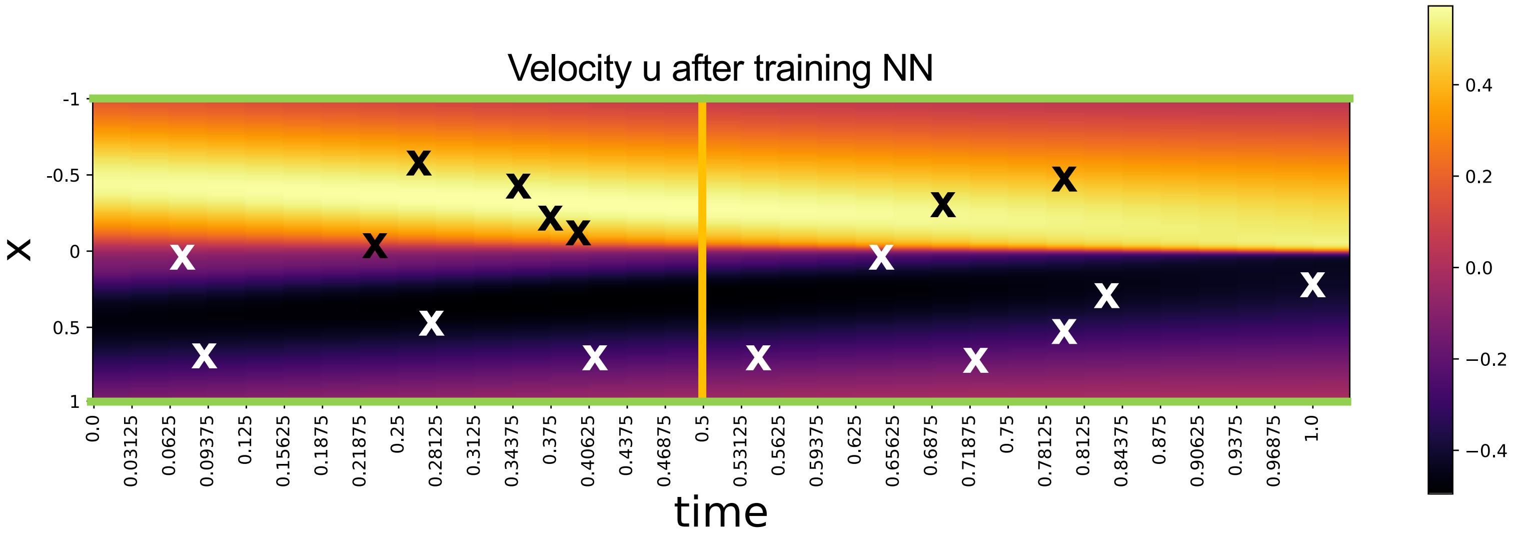
$x \in [-1, 1]$ 128 steps

$t \in [0, 1]$ 33 steps

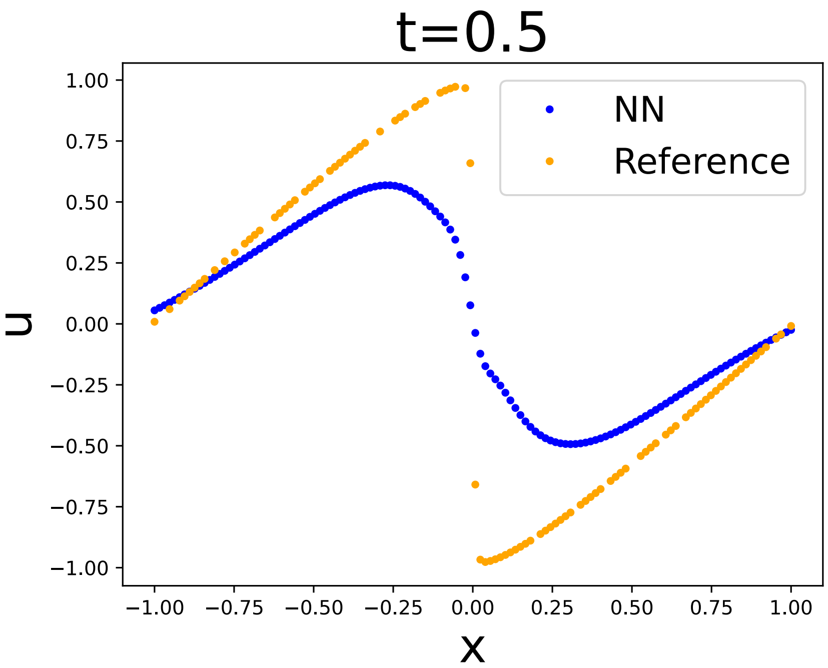


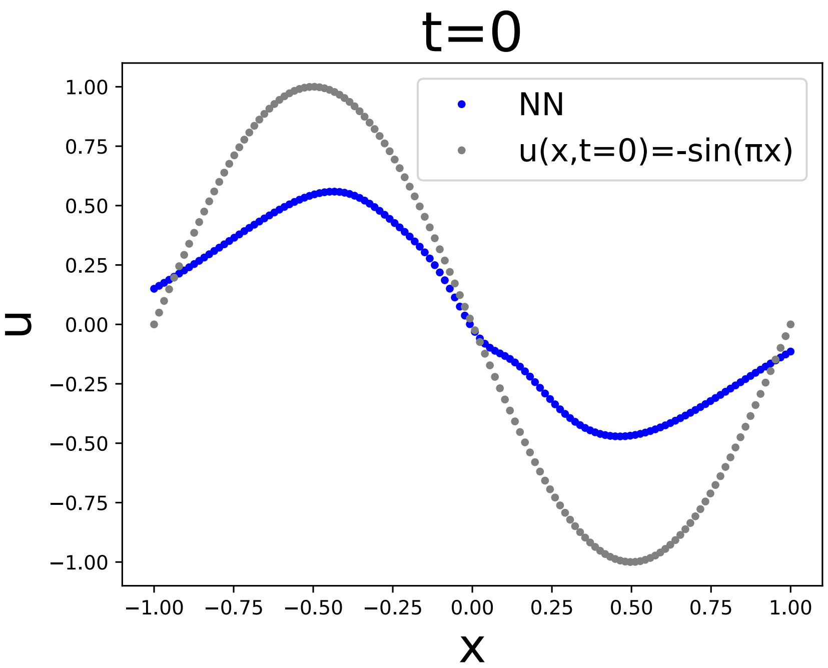
Physical losses: solving Burger's equation

- After training: need to evaluate NN for all grid points!
  - Computationally expensive for large grids
- Why is this possible?
  - NN inherently supports calculation of derivatives



Physical losses: discussion





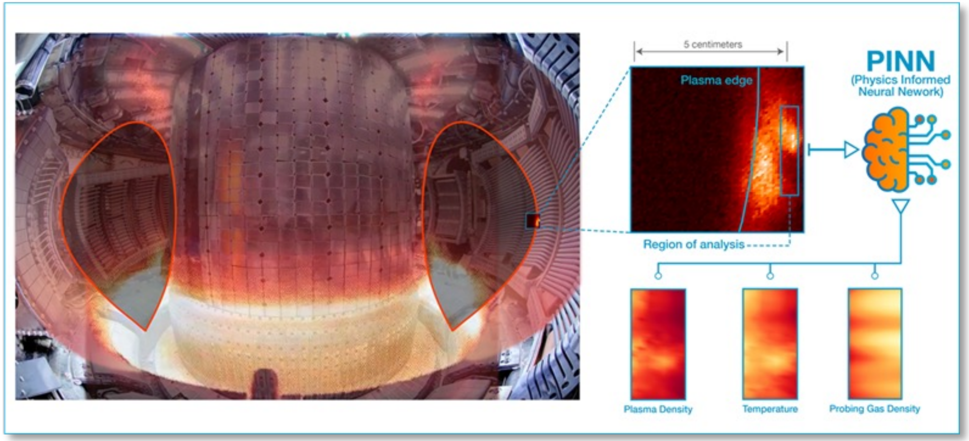
- Boundary conditions u=0 are fulfilled
- Shock at center not well represented

- Initial state of PDE solver not well represented
- More accurate representation requires significantly more iterations even for this simple case

Physical losses: discussion

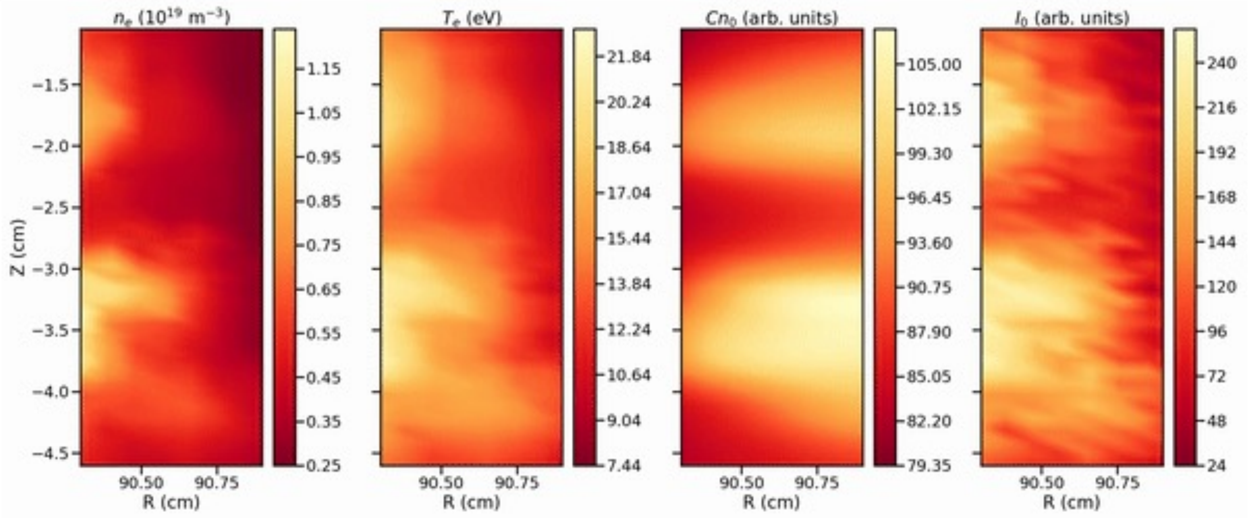
- This is a conceptual starting point, but not very accurate
- Physical loss allows to encode (unique) solutions to PDEs with NNs, which allows to use NNs as universal function approximators
- Not really “machine learning”: we reconstruct a single PDE solution in a known space-time region

PINN application examples



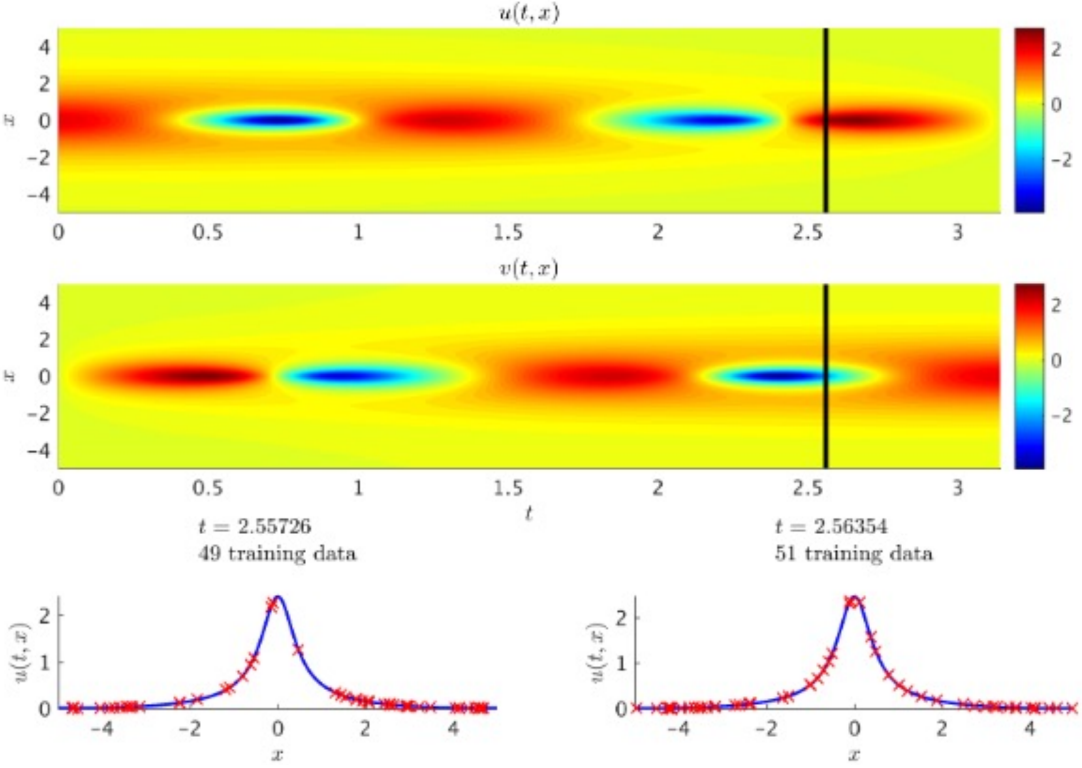
The learned two-dimensional n_e (a), T_e (b), and Cn_0 (c) for plasma discharge 1 120 711 021 along with the experimentally observed 587.6 nm photon emission (d) at $t = 1.312815$ s. The learned measurements are based on the collective predictions within the deep learning framework training against the neutral transport physics and $N_P = 1$ CR theory constraints. Multimedia view: https://doi.org/10.1063/5.0088216.1.

Reconstruction of turbulent fluctuations of plasma in a tokamak (A. Mathews et al., 2022, [doi.org/10.1063/5.0088216])



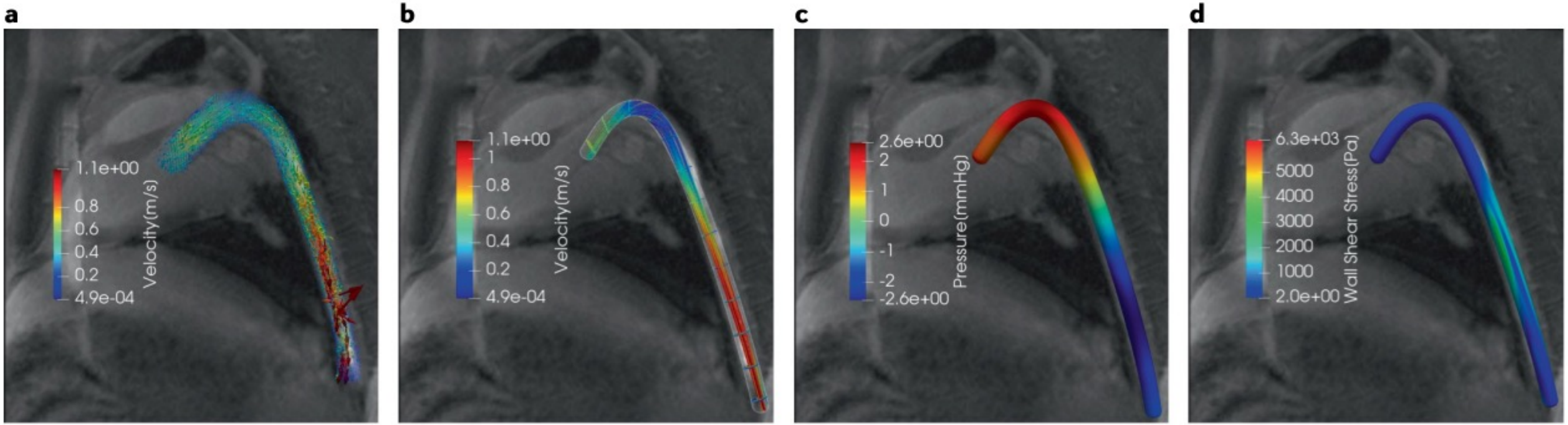
PINN application examples

Reconstructing unknown parameters in Schrödinger equation (M. Raissi et al., 2017, [arxiv.org/abs/1708.00588])



Correct PDE	$ih_t + 0.5h_{xx} + \lvert h\rvert^2 h = 0$
Identified PDE (clean data)	$ih_t + 0.506h_{xx} + 0.995\lvert h\rvert^2 h = 0$
Identified PDE (1% noise)	$ih_t + 0.476h_{xx} + 0.999\lvert h\rvert^2 h = 0$

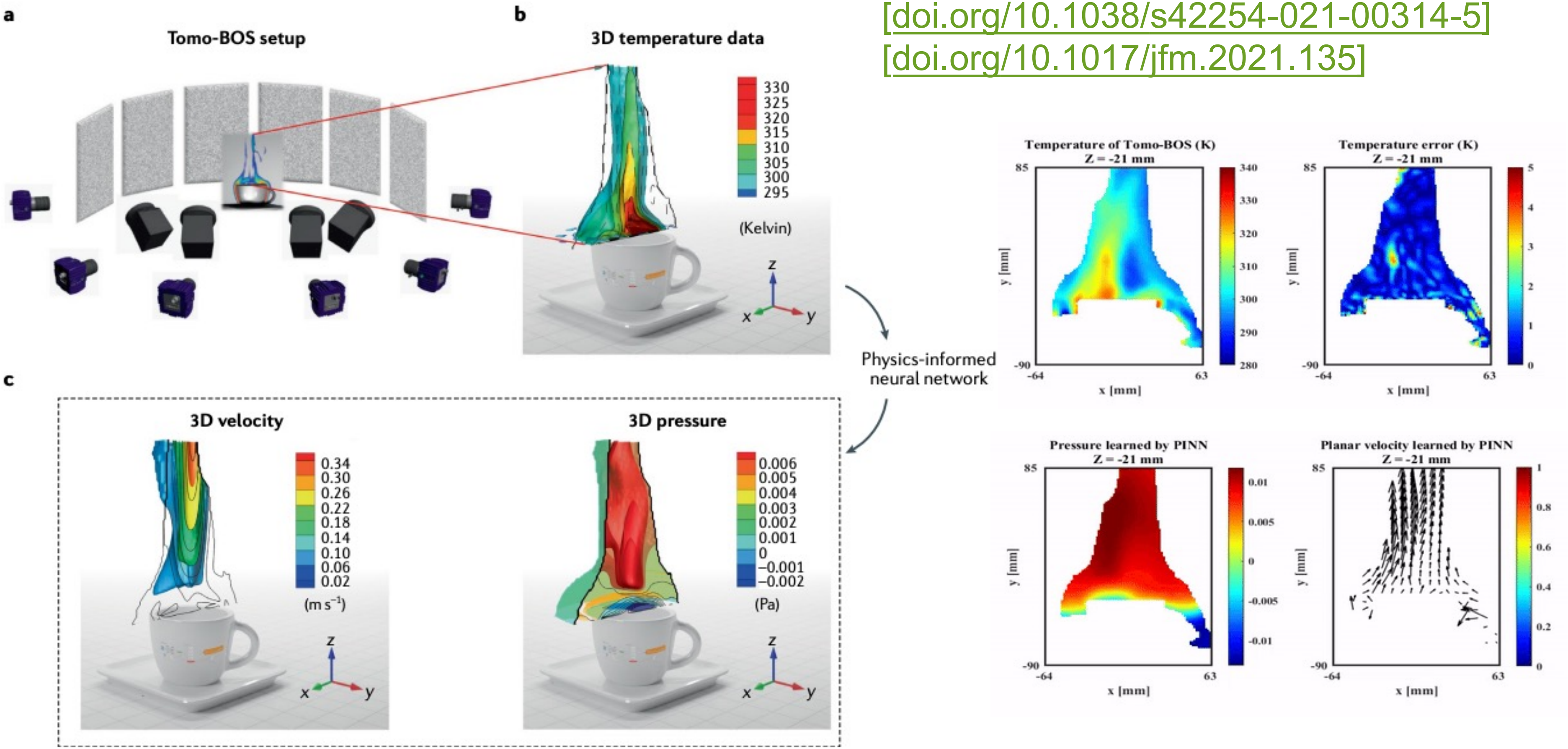
PINN application examples



Reconstruction of blood flow in a blood vessel (E. Hwuang, S. Wang et al. [doi.org/10.1038/s42254-021-00314-5])

PINN application examples

F. Fuest, S. Cai et al.
[doi.org/10.1038/s42254-021-00314-5]
[doi.org/10.1017/jfm.2021.135]



Physical losses: pros & cons

Easy setup with simple NN

Quite slow: need to evaluate NN at every grid point, i.e. “paint the image pixel by pixel”

Physical losses: pros & cons

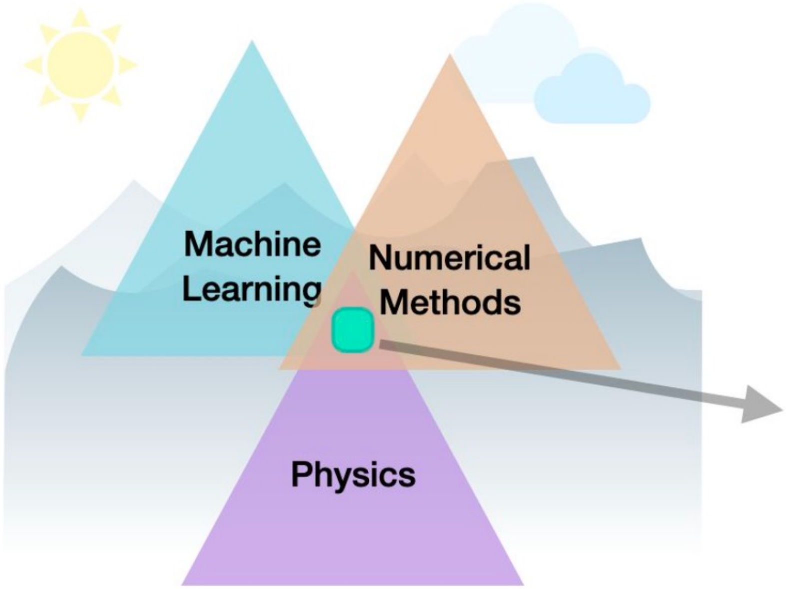
- Easy setup with simple NN
- PDE derivatives in physical loss can be computed with backpropagation

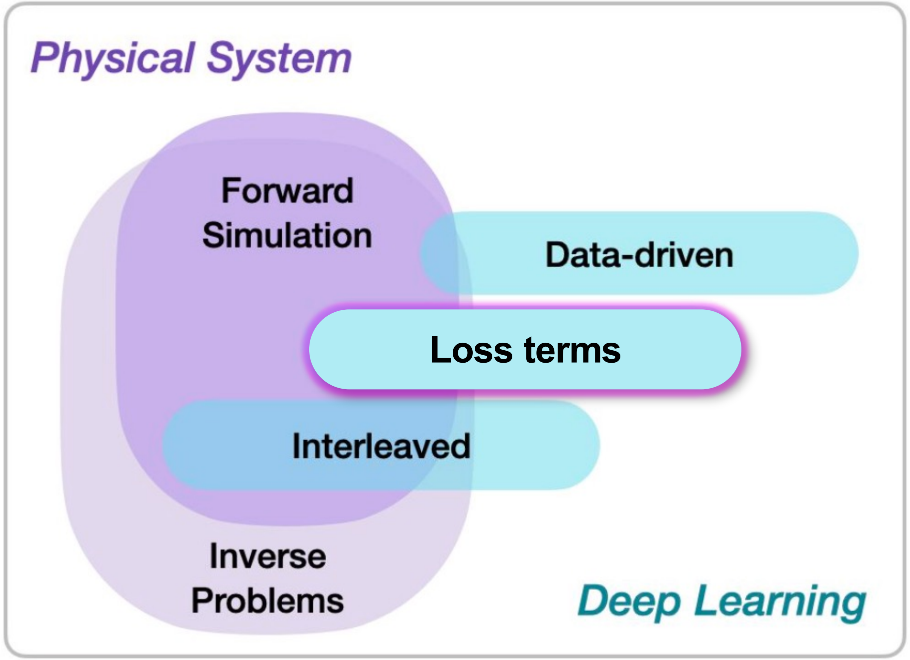
- Quite slow: need to evaluate NN at every grid point, i.e. “paint the image pixel by pixel”
- Accuracy of computed derivatives relies on learned representation

Physical losses: pros & cons

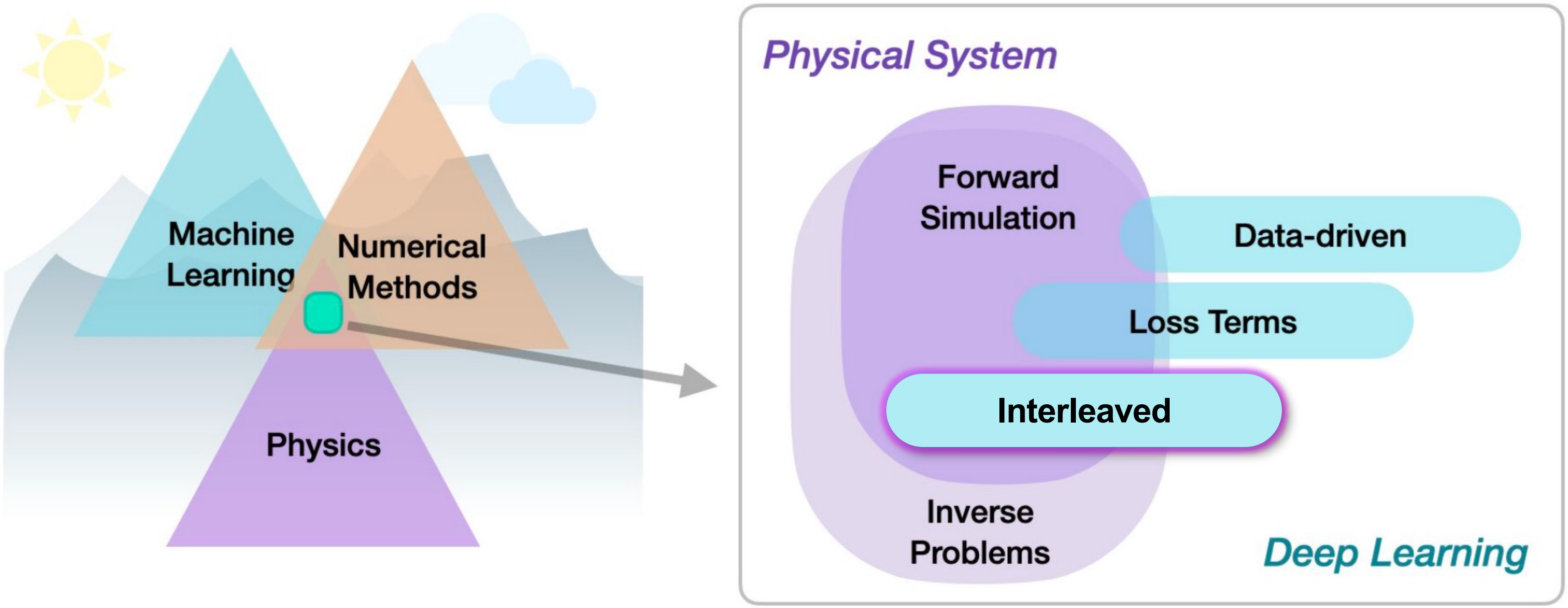
Pros	Cons
Easy setup with simple NN	Quite slow: need to evaluate NN at every grid point, i.e. “paint the image pixel by pixel”
PDE derivatives in physical loss can be computed with backpropagation	Accuracy of computed derivatives relies on learned representation
Popular for inverse problems: given certain measurements or observations (=training data), find a PDE solution	Does not combine well with numerical solvers e.g. for refining solution

ML & numerical methods: new possibilities





ML & numerical methods: new possibilities



Differentiable physics

- Differentiable physics = differentiable numerical simulations of physical systems

Differentiable physics

- Differentiable physics = differentiable numerical simulations of physical systems
- Equip classical numerical solvers (discretized PDE) with the ability to compute gradients with respect to their inputs

Differentiable physics

- Differentiable physics = differentiable numerical simulations of physical systems
- Equip classical numerical solvers (discretized PDE) with the ability to compute gradients with respect to their inputs
- This allows integration of numerical methods into the training process of an attached NN

Differentiable physics: differentiable solver example

- Linear advection equation with ***u***=***u***(***x***) and *d*=*d*(***x***,*t*)

$$\frac{\partial d}{\partial t} + \mathbf{u} \cdot \nabla d = 0$$

Differentiable physics: differentiable solver example

- Linear advection equation with ***u***=***u***(***x***) and *d*=*d*(***x***,*t*)

E.g. in 1D

$$\frac{\partial d}{\partial t} + \mathbf{u} \cdot \nabla d = 0 \Rightarrow \boxed{\frac{d(x_i, t+\Delta t) - d(x_i, t)}{\Delta t} = -u(x_i)\frac{d(x_i+\Delta x, t) - d(x_i, t)}{\Delta x}}$$

Differentiable physics: differentiable solver example

- Linear advection equation with ***u***=***u***(***x***) and *d*=*d*(***x***,*t*)

E.g. in 1D

$$\frac{\partial d}{\partial t} + \mathbf{u} \cdot \nabla d = 0 \Rightarrow \boxed{d(x_i, t + \Delta t) = d(x_i, t) - \Delta t \left[u(x_i) \frac{d(x_{i+1}, t) - d(x_i, t)}{\Delta x} \right]}$$

- Generally:

$$d(t + \Delta t) = \mathcal{P}(d(t), u, t + \Delta t)$$

Differentiable physics: differentiable solver example

- Linear advection equation with $\boldsymbol{u}=\boldsymbol{u}(\boldsymbol{x})$ and $d=d(\boldsymbol{x},t)$

E.g. in 1D

$$\frac{\partial d}{\partial t} + \mathbf{u} \cdot \nabla d = 0 \quad \Rightarrow \quad \boxed{d(x_i, t+\Delta t) = d(x_i, t) - \Delta t \left[u(x_i) \frac{d(x_{i+1}, t) - d(x_i, t)}{\Delta x} \right]}$$

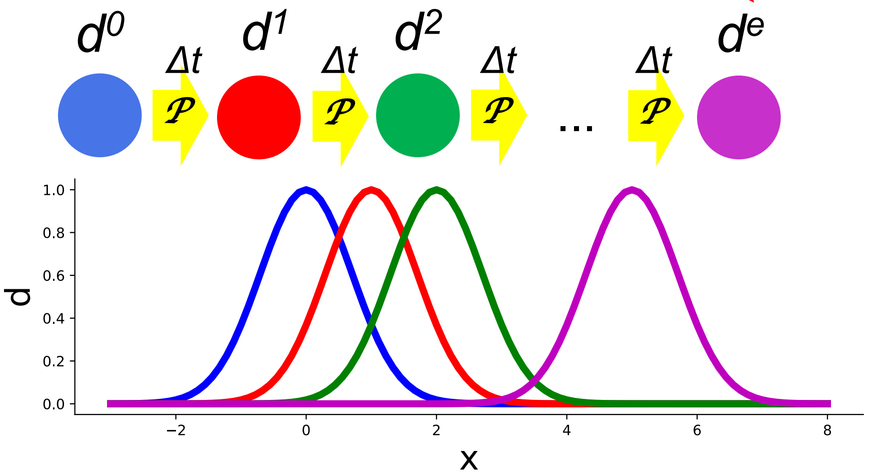
- Generally:

$$d(t+\Delta t) = \mathcal{P}(d(t), u, t+\Delta t)$$

- For N forward iterations:

$$d^e = \mathcal{P}(d^0, u, t + N\Delta t = t^e)$$

end time step



Differentiable physics: differentiable solver example

- Linear advection equation with ***u***=***u***(***x***) and *d*=*d*(***x***,*t*)

E.g. in 1D

$$\frac{\partial d}{\partial t} + \mathbf{u} \cdot \nabla d = 0 \Rightarrow \boxed{d(x_i, t+\Delta t) = d(x_i, t) - \Delta t \left[u(x_i) \frac{d(x_{i+1}, t) - d(x_i, t)}{\Delta x} \right]}$$

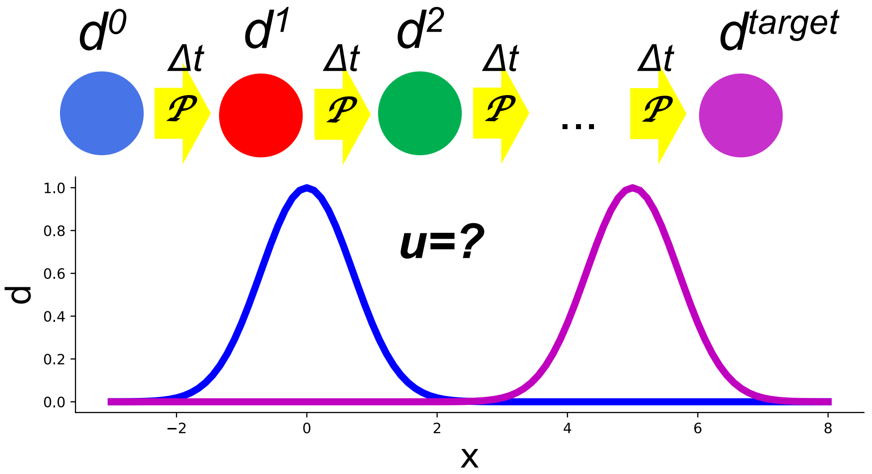
- Generally:

$$d(t+\Delta t) = \mathcal{P}(d(t), u, t+\Delta t)$$

- For N forward iterations:

$$d^e = \mathcal{P}(d^0, u, t + N\Delta t = t^e)$$

- Find velocity field ***u*** that brings a known initial density $\boldsymbol{d^0 = d(t^0=0)}$ into a known target density $\boldsymbol{d^{target} = d(t^e=t+N\Delta t)}$



Differentiable physics: differentiable solver example

- Linear advection equation with ***u***=***u***(***x***) and *d*=*d*(***x***,*t*)

E.g. in 1D

$$\frac{\partial d}{\partial t} + \mathbf{u} \cdot \nabla d = 0 \quad \Rightarrow \quad \boxed{d(x_i, t+\Delta t) = d(x_i, t) - \Delta t \left[u(x_i) \frac{d(x_{i+1}, t) - d(x_i, t)}{\Delta x} \right]}$$

- Generally:

$$d(t+\Delta t) = \mathcal{P}(d(t), u, t+\Delta t)$$

- For N forward iterations:

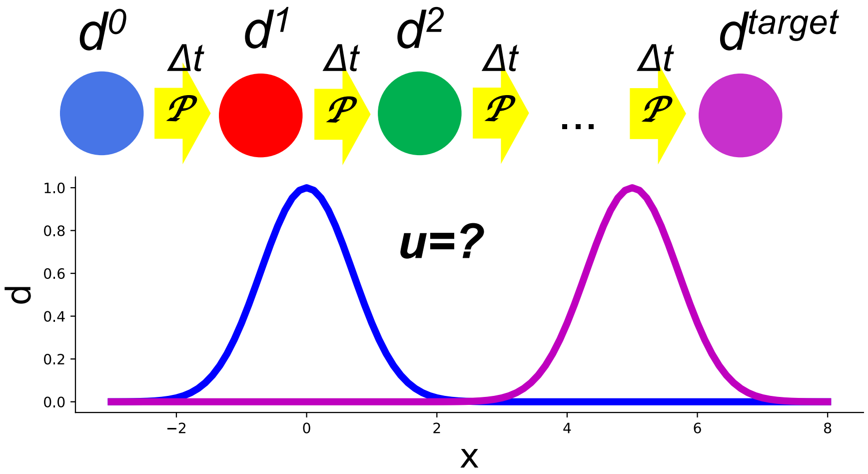
$$d^e = \mathcal{P}(d^0, u, t + N\Delta t = t^e)$$

- Optimization problem:

$$L = |d^e - d^{\text{target}}|^2$$

output of simulation   known from observation

- Find velocity field ***u*** that brings a known initial density ***d^0 = d(t^0=0)*** into a known target density ***d^{target} = d(t^e=t+NΔt)***



Differentiable physics: differentiable solver example

- Linear advection equation with ***u***=***u***(***x***) and *d*=*d*(***x***,*t*)

E.g. in 1D

$$\frac{\partial d}{\partial t} + \mathbf{u} \cdot \nabla d = 0 \quad \Rightarrow \quad \boxed{d(x_i, t + \Delta t) = d(x_i, t) - \Delta t \left[u(x_i) \frac{d(x_{i+1}, t) - d(x_i, t)}{\Delta x} \right]}$$

- Generally:

$$d(t + \Delta t) = \mathcal{P}(d(t), u, t + \Delta t)$$

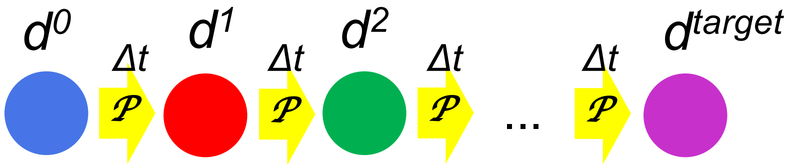
- For N forward iterations:

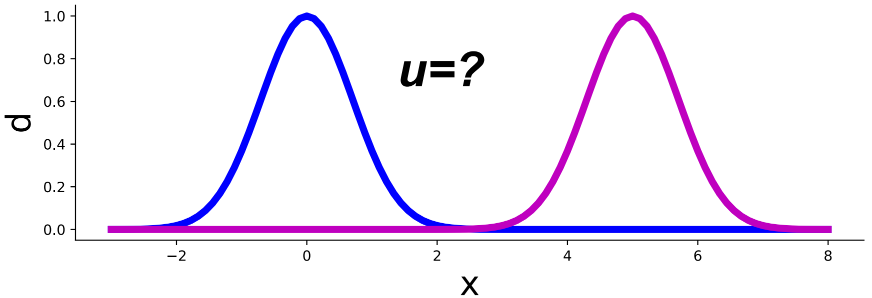
$$\boxed{d^e = \mathcal{P}(d^0, u, t + N\Delta t = t^e)}$$

- Optimization problem:

$$L = |d^e - d^{\text{target}}|^2 = |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

- Find velocity field ***u*** that brings a known initial density ***d⁰ = d(t⁰=0)*** into a known target density ***d^target = d(t^e=t+NΔt)***





Differentiable physics: differentiable solver example

- Linear advection equation with ***u*=*u*(*x*)** and *d*=*d*(***x***,*t*)

E.g. in 1D

$$\frac{\partial d}{\partial t} + \mathbf{u} \cdot \nabla d = 0 \quad \Rightarrow \quad \boxed{d(x_i, t+\Delta t) = d(x_i, t) - \Delta t \left[u(x_i) \frac{d(x_{i+1}, t) - d(x_i, t)}{\Delta x} \right]}$$

- Generally:

$$d(t+\Delta t) = \mathcal{P}(d(t), u, t+\Delta t)$$

- For N forward iterations:

$$d^e = \mathcal{P}(d^0, u, t + N\Delta t = t^e)$$

- Optimization problem:

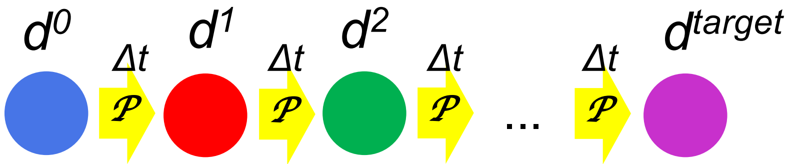
$$L = |d^e - d^{\text{target}}|^2 = |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

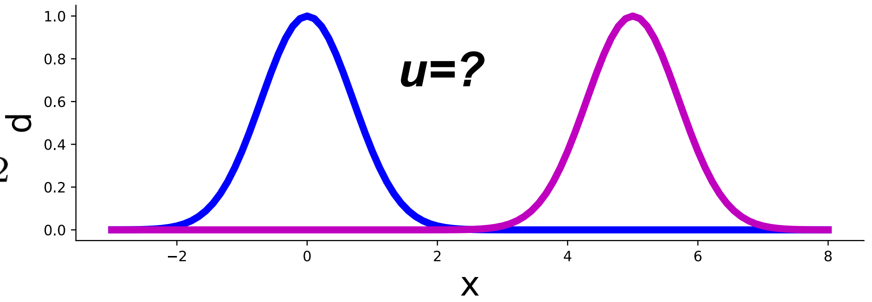
known reference quantities from observation

$$\underset{u}{\text{argmin}}\, L(u) = \underset{u}{\text{argmin}}\, |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

u unknown at start e.g. init with 0s

- Find velocity field ***u*** that brings a known initial density ***d^0 = $d(t^0=0)$*** into a known target density ***d^{target} = $d(t^e=t+N\Delta t)$***





$$\underset{u}{\operatorname{argmin}} L(u) = \underset{u}{\operatorname{argmin}} |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

learning rate

$$\Delta u_i = \frac{\partial L}{\partial u_i} = \frac{\partial \mathcal{P}_i}{\partial u_i} \frac{\partial L}{\partial \mathcal{P}_i} \qquad u_i^{\text{new}} = u_i - \eta \Delta u_i$$

Differentiable physics: differentiable solver example

$$\underset{u}{\operatorname{argmin}} L(u) = \underset{u}{\operatorname{argmin}} |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

learning rate

$$\Delta u_i = \frac{\partial L}{\partial u_i} = \frac{\partial \mathcal{P}_i}{\partial u_i} \boxed{\frac{\partial L}{\partial \mathcal{P}_i}} \qquad u_i^{\text{new}} = u_i - \eta \Delta u_i$$

$$\boxed{\sum_i |d_i^e - d_i^{\text{target}}|^2}$$

In case of a single forward step from t^{e-1} to t^e:

$$\boxed{\frac{\partial L}{\partial \mathcal{P}_i} = \frac{\partial |\mathcal{P} - d^{\text{target}}|^2}{\partial \mathcal{P}_i} = \frac{\partial L}{\partial d_i^e} = \frac{\partial |d^e - d^{\text{target}}|^2}{\partial d_i^e} = 2\left(d_i^e - d_i^{\text{target}}\right)}$$

Differentiable physics: differentiable solver example

$$\underset{u}{\operatorname{argmin}} L(u) = \underset{u}{\operatorname{argmin}} |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

learning rate

$$\Delta u_i = \frac{\partial L}{\partial u_i} = \boxed{\frac{\partial \mathcal{P}_i}{\partial u_i}} \boxed{\frac{\partial L}{\partial \mathcal{P}_i}} \qquad u_i^{\text{new}} = u_i - \eta \Delta u_i$$

In case of a single forward step from t^{e-1} to t^e:

$$\boxed{\frac{\partial L}{\partial \mathcal{P}_i} = \frac{\partial |\mathcal{P} - d^{\text{target}}|^2}{\partial \mathcal{P}_i} = \frac{\partial L}{\partial d_i^e} = \frac{\partial |d^e - d^{\text{target}}|^2}{\partial d_i^e} = 2\left(d_i^e - d_i^{\text{target}}\right)}$$

$$\boxed{\frac{\partial \mathcal{P}_i}{\partial u_i} = \frac{\partial d_i^e}{\partial u_i} = -\frac{\Delta t}{\Delta x}\left[d_{i+1}^{e-1} - d_i^{e-1}\right]}$$

From: $\boxed{d(x_i, t + \Delta t) = d(x_i, t) - \Delta t \left[u(x_i) \frac{d(x_{i+1}, t) - d(x_i, t)}{\Delta x} \right]}$ with: $t^e = t + \Delta t$, $t^{e-1} = t$

Differentiable physics: differentiable solver example

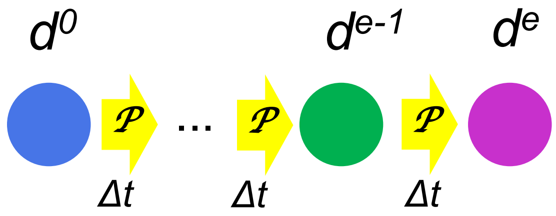
$$\underset{u}{\operatorname{argmin}} L(u) = \underset{u}{\operatorname{argmin}} |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

$$\Delta u_i = \frac{\partial L}{\partial u_i} = \frac{\partial \mathcal{P}_i}{\partial u_i} \frac{\partial L}{\partial \mathcal{P}_i} \qquad u_i^{\text{new}} = u_i - \eta \Delta u_i$$

In case of multiple forward steps from t^0 to t^e:

$$\Delta u_i = \frac{\partial d^e}{\partial u_i} \frac{\partial L}{\partial d^e} + \frac{\partial d^{e-1}}{\partial u_i} \frac{\partial d^e}{\partial d^{e-1}} \frac{\partial L}{\partial d^e} + \cdots + \left(\frac{\partial d^0}{\partial u_i} \cdots \frac{\partial d^{e-1}}{\partial d^{e-2}} \frac{\partial d^e}{\partial d^{e-1}} \frac{\partial L}{\partial d^e} \right)$$

- Final density $\boldsymbol{d^e}$ depends on velocity $\boldsymbol{u_i}$ through all previous density states:



Differentiable physics: differentiable solver example

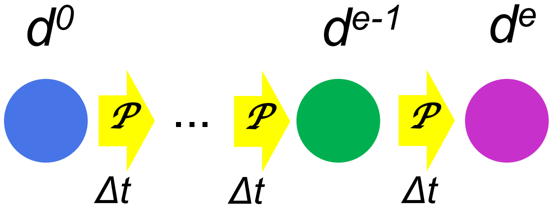
$$\operatorname*{argmin}_u L(u) = \operatorname*{argmin}_u |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

$$\Delta u_i = \frac{\partial L}{\partial u_i} = \frac{\partial \mathcal{P}_i}{\partial u_i}\frac{\partial L}{\partial \mathcal{P}_i} \qquad u_i^{\text{new}} = u_i - \eta \Delta u_i$$

In case of multiple forward steps from t^0 to t^e:

$$\begin{aligned}\Delta u_i = & \frac{\boxed{\partial d^e}}{\partial u_i}\frac{\partial L}{\boxed{\partial d^e}} \\ & + \frac{\boxed{\partial d^{e-1}}}{\partial u_i}\frac{\partial d^e}{\partial d^{e-1}}\frac{\partial L}{\boxed{\partial d^e}} \\ & + \cdots \\ & + \left(\frac{\boxed{\partial d^0}}{\partial u_i}\cdots\frac{\partial d^{e-1}}{\partial d^{e-2}}\frac{\partial d^e}{\partial d^{e-1}}\frac{\partial L}{\boxed{\partial d^e}}\right)\end{aligned}$$

- Final density $\boldsymbol{d^e}$ depends on velocity $\boldsymbol{u_i}$ through all previous density states:



Differentiable physics: differentiable solver example

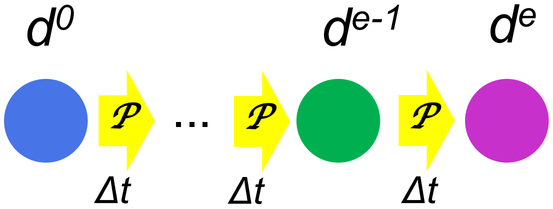
$$\operatorname*{argmin}_u L(u) = \operatorname*{argmin}_u |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

$$\Delta u_i = \frac{\partial L}{\partial u_i} = \frac{\partial \mathcal{P}_i}{\partial u_i}\frac{\partial L}{\partial \mathcal{P}_i} \qquad u_i^{\text{new}} = u_i - \eta \Delta u_i$$

In case of multiple forward steps from t^0 to t^e:

$$\Delta u_i = \frac{\partial d^e}{\partial u_i}\frac{\partial L}{\partial d^e}$$
$$+ \frac{\partial d^{e-1}}{\partial u_i}\frac{\partial d^e}{\partial d^{e-1}}\frac{\partial L}{\partial d^e}$$
$$+ \ \cdots$$
$$+ \left(\frac{\partial d^0}{\partial u_i} \cdots \frac{\partial d^{e-1}}{\partial d^{e-2}}\frac{\partial d^e}{\partial d^{e-1}}\frac{\partial L}{\partial d^e}\right)$$

- Final density $\boldsymbol{d^e}$ depends on velocity $\boldsymbol{u_i}$ through all previous density states:



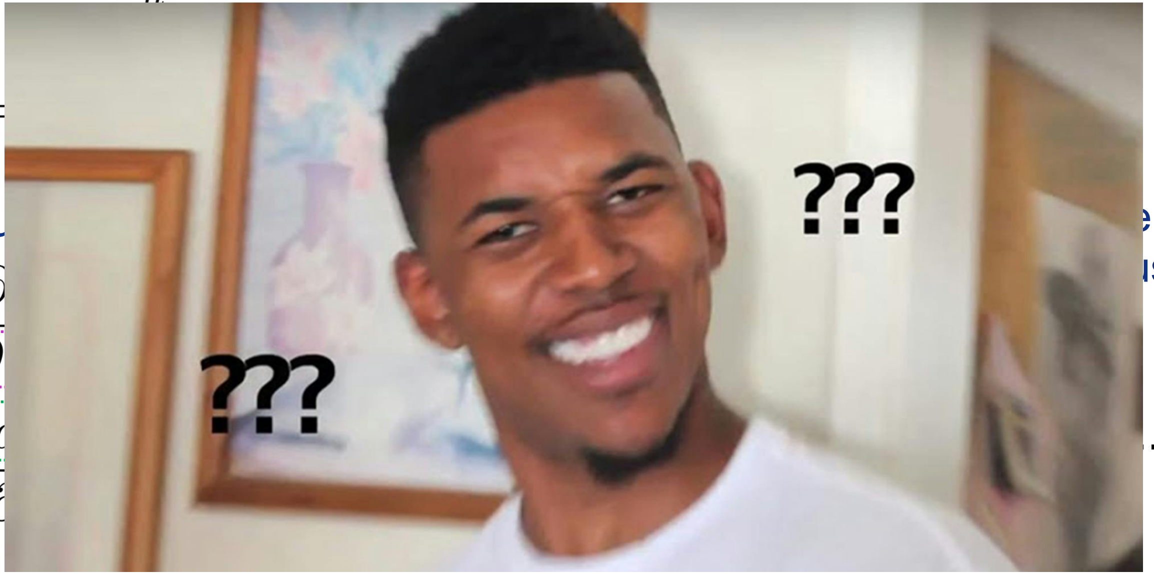
$$\boxed{\frac{\partial d_i^e}{\partial d_i^{e-1}} = 1 + \frac{\Delta t}{\Delta x} u_i}$$

*potential contributions from cells i+1, i-1 etc...

Differentiable physics: differentiable solver example

$$\operatorname*{argmin}_u L(u) = \operatorname*{argmin}_u |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

$$\Delta u_i = \frac{\partial L}{\partial u_i} = \ldots$$



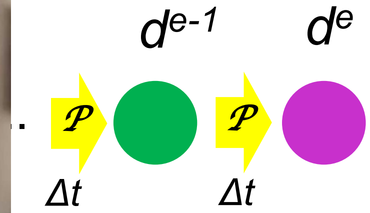
In case of mu… …epends on velocity $\boldsymbol{u_i}$ …us density states:

$$\Delta u_i = \frac{\partial d^e}{\partial u_i} \ldots$$

$$+ \ldots$$

$$+ \ldots$$

$$+ \left(\frac{\partial d^0}{\partial u_i} \cdots \frac{\partial d^{e-1}}{\partial d^{e-2}} \frac{\partial d^e}{\partial d^{e-1}} \frac{\partial L}{\partial d^e} \right)$$



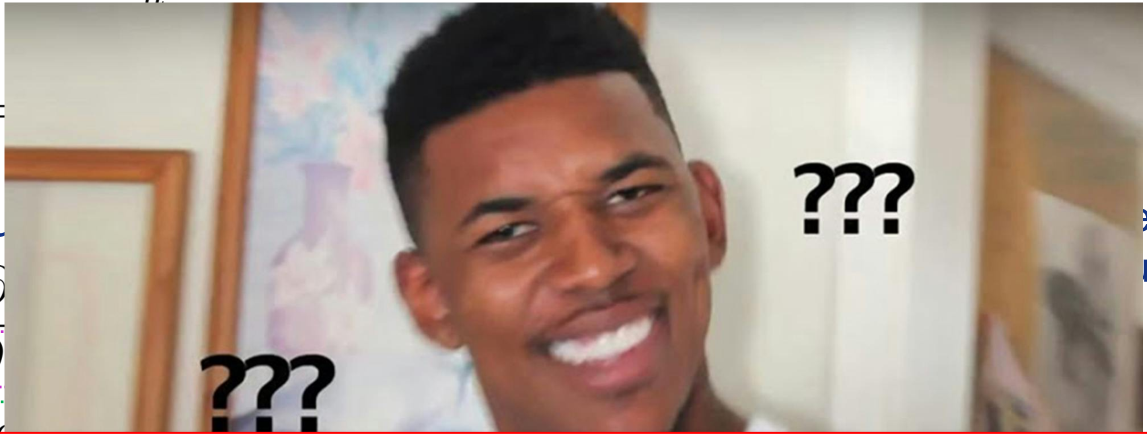
$$\boxed{\frac{\partial d_i^e}{\partial d_i^{e-1}} = 1 + \frac{\Delta t}{\Delta x} u_i}$$

*potential contributions from cells i+1, i-1 etc...

Differentiable physics: differentiable solver example

$$\underset{u}{\text{argmin}}\, L(u) = \underset{u}{\text{argmin}}\, |\mathcal{P}(d^0, u, t^e) - d^{\text{target}}|^2$$

$$\Delta u_i = \frac{\partial L}{\partial u_i} = \ldots$$



In case of mu... ...epends on velocity $\boldsymbol{u_i}$ ...us density states:

$$\Delta u_i = \frac{\partial d^e}{\partial u_i} \frac{\partial \ldots}{\partial \ldots} \ldots$$

d^{e-1} d^e

A typical PDE based numerical solver consists of arithmetic operations which are differentiable. The computation of gradients is typically not expensive and it can happen during forward simulation.

$$+ \left(\frac{\partial d^0}{\partial u_i} \cdots \frac{\partial d^{e-1}}{\partial d^{e-2}} \frac{\partial d^e}{\partial d^{e-1}} \frac{\partial L}{\partial d^e} \right)$$

$$\ldots \partial d_i^{e-1} \ldots \Delta x \ldots$$

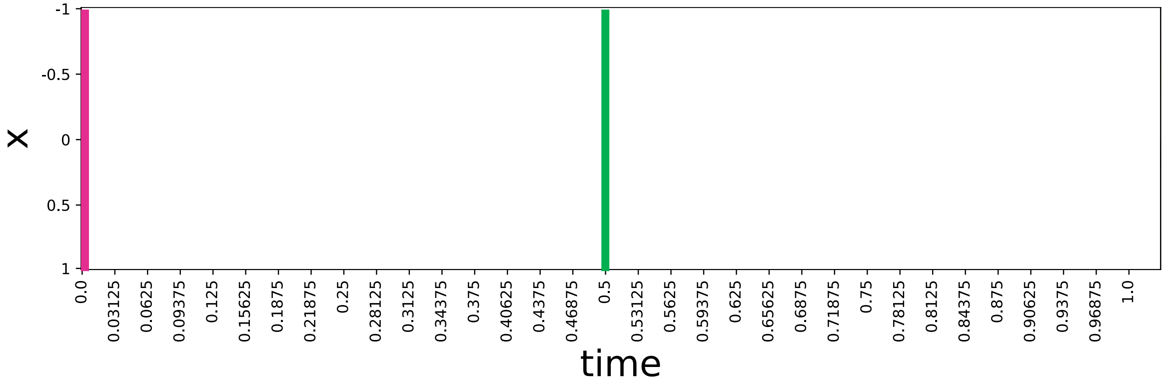
*potential contributions from cells i+1, i-1 etc...

Differentiable physics: solving Burger's equation with DP

- Classical gradient based optimization, no DL
- Start with $u(x,t=0)=0$, $u^{target}(x,t=0.5)$ known

Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$



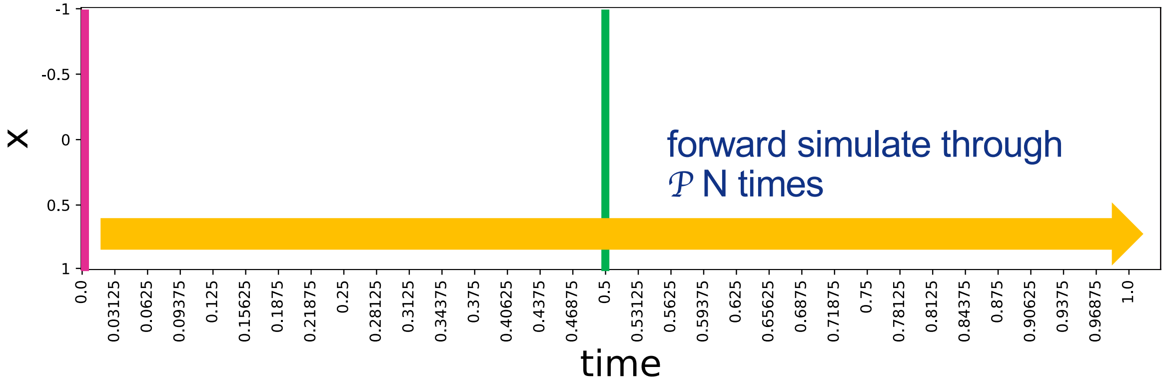
Differentiable physics: solving Burger's equation with DP

Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

- Classical gradient based optimization, no DL
- Start with u(x,t=0)=0, u^{target}(x,t=0.5) known

1. Simulate $\mathcal{P}$ (discretized PDE) from t=0 to t=1 in N=32 time steps



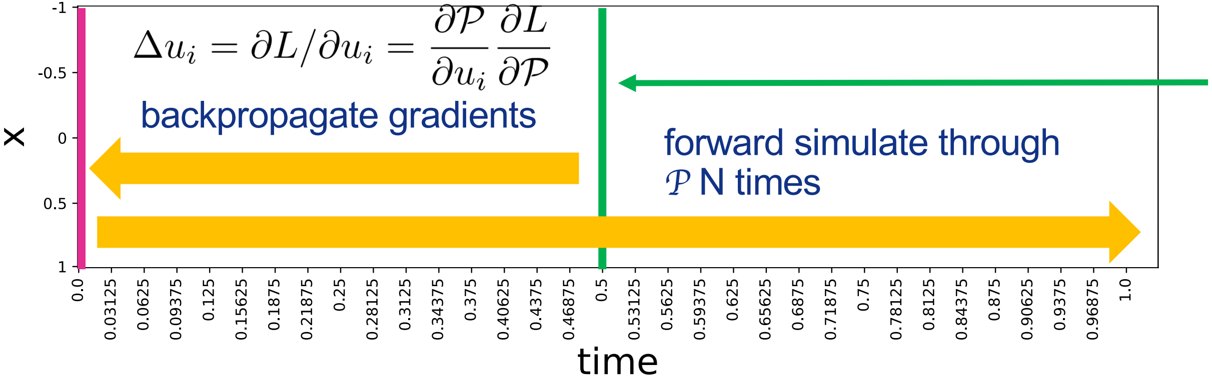
Differentiable physics: solving Burger's equation with DP

- Classical gradient based optimization, no DL
- Start with u(x,t=0)=0, u^{target}(x,t=0.5) known

Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

1. Simulate $\mathcal{P}$ (discretized PDE) from t=0 to t=1 in N=32 time steps
2. Backpropagate gradients from t=0.5, 16 steps back till first step at t=0



$$L = \sum_i |u_i(t=0.5) - u_i^{\text{target}}|^2$$

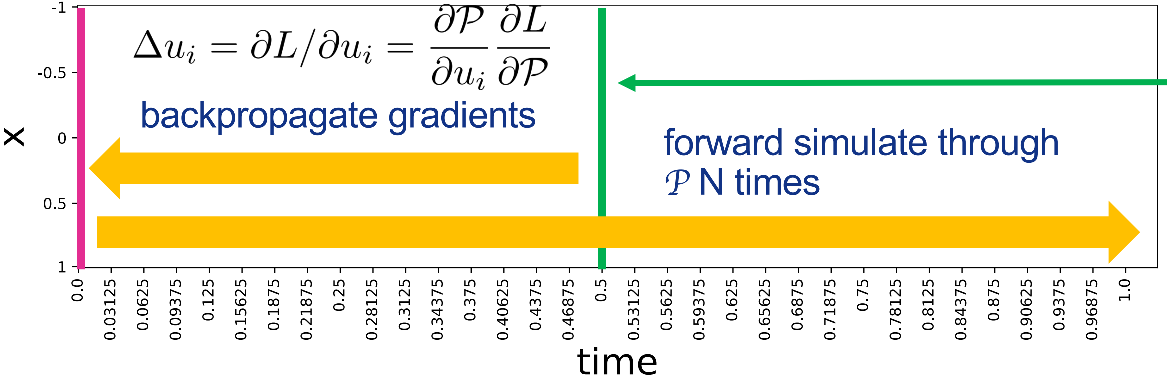
Differentiable physics: solving Burger’s equation with DP

- Classical gradient based optimization, no DL
- Start with u(x,t=0)=0, u^{target}(x,t=0.5) known

Burger’s equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

1. Simulate $\mathcal{P}$ (discretized PDE) from t=0 to t=1 in N=32 time steps
2. Backpropagate gradients from t=0.5, 16 steps back till first step at t=0
3. Update u(x,t=0) with gradients: $u_i^{\mathrm{new}} = u_i - \eta \Delta u_i$



$$L = \sum_i |u_i(t=0.5) - u_i^{\mathrm{target}}|^2$$

Differentiable physics: solving Burger's equation with DP

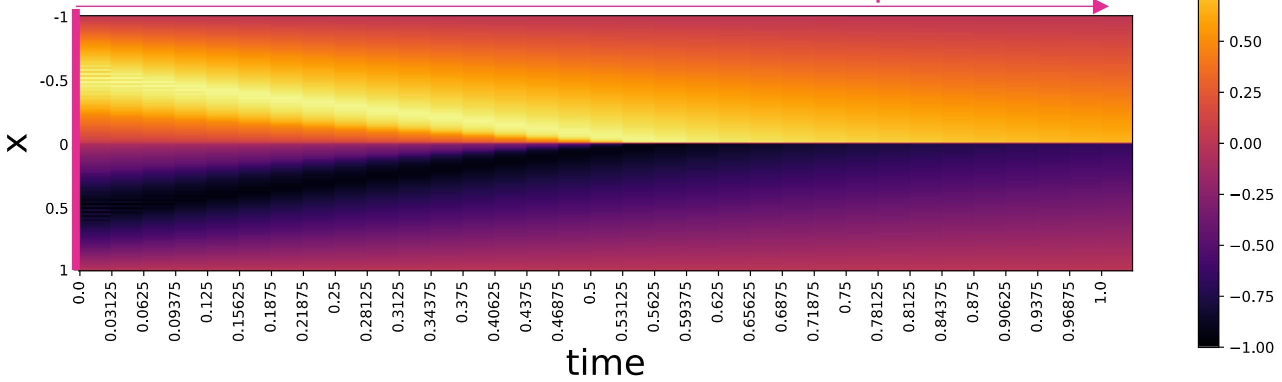
- Classical gradient based optimization, no DL
- Start with u(x,t=0)=0, u^{target}(x,t=0.5) known

Burger's equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$

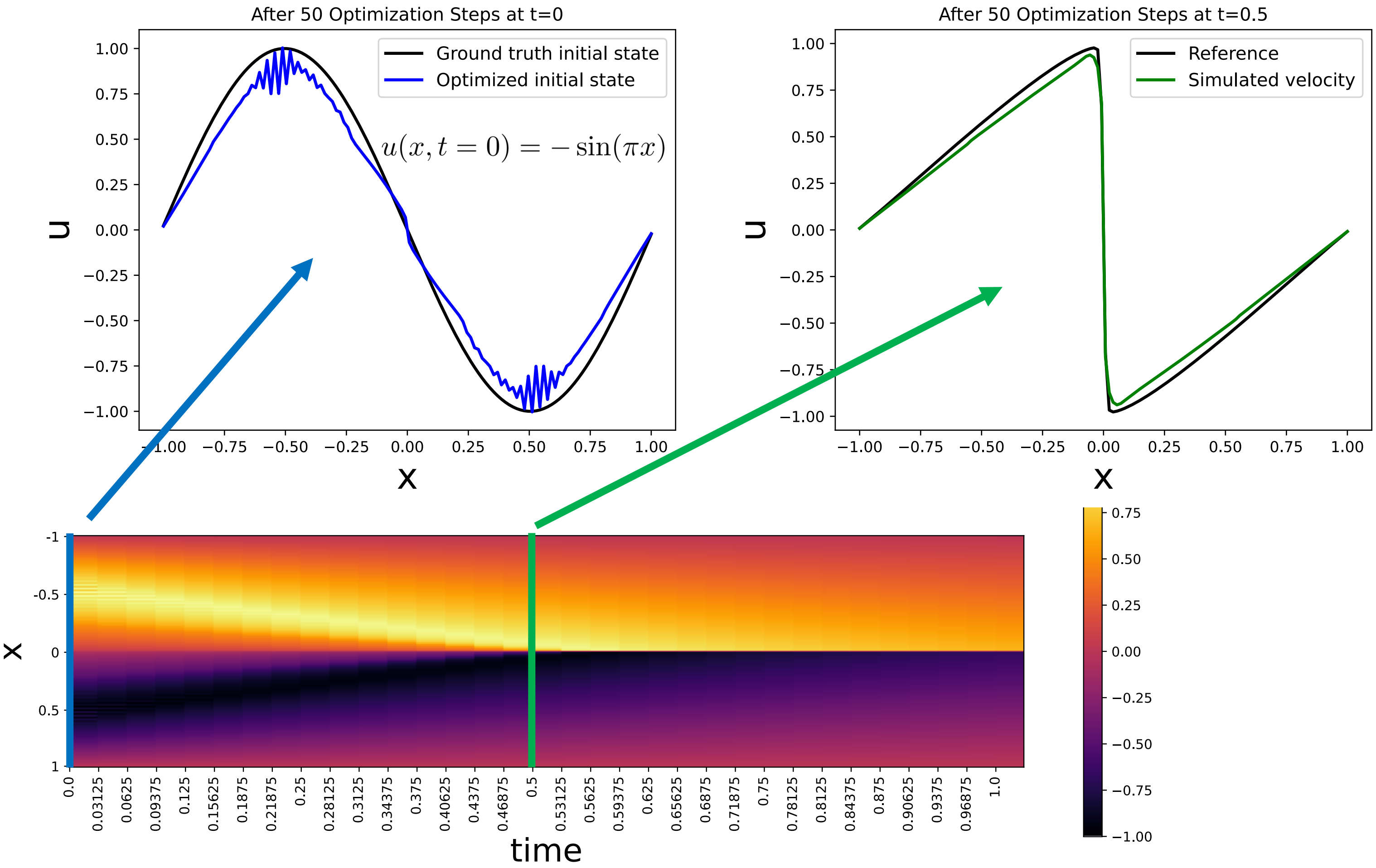
1. Simulate $\mathcal{P}$ (discretized PDE) from t=0 to t=1 in N=32 time steps
2. Backpropagate gradients from t=0.5, 16 steps back till first step at t=0
3. Update u(x,t=0) with gradients

forward sim. with optimized ***u***

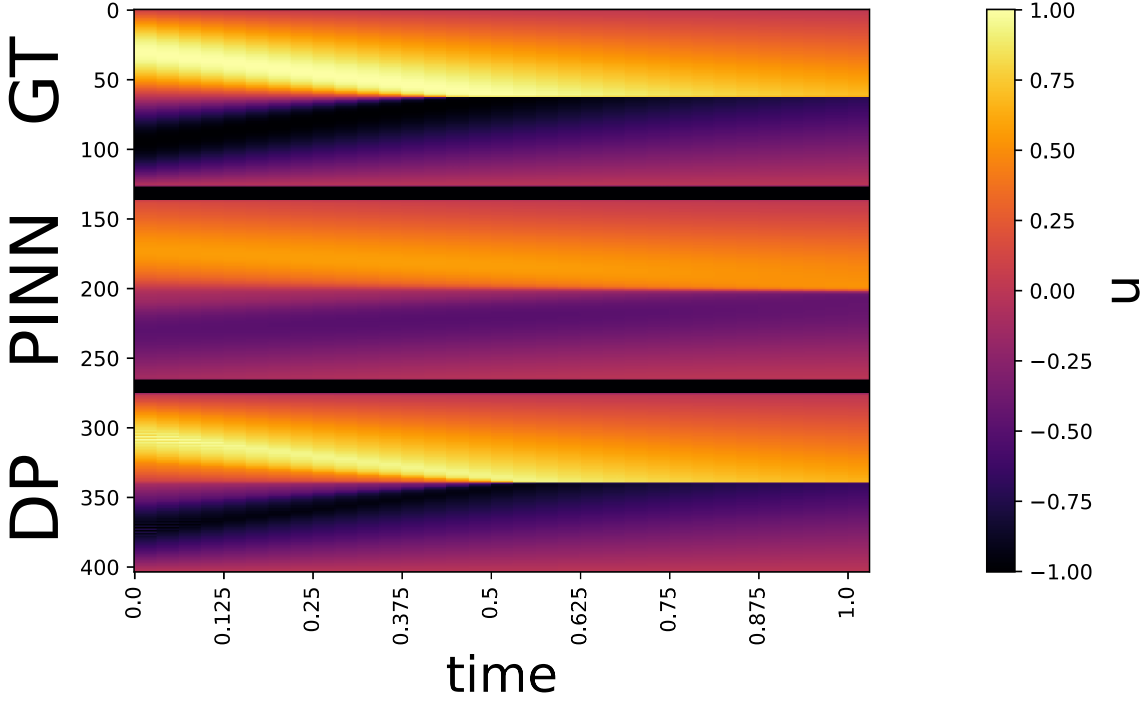


4. Repeat 50 times (~2 mins)

Differentiable physics: solving Burger's equation with DP



Differentiable physics: discussion of PINN vs DP



Physics informed NN (PINN)

- Recovers the overall shape of the solution
- Temporal constraints are at least partially fulfilled
- Poor reconstruction of amplitudes

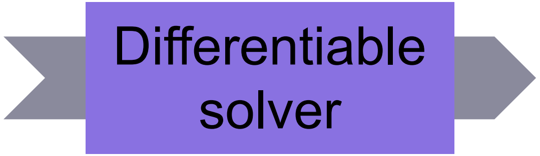
Differentiable physics solver (DP)

- Much closer to ground truth (GT) thanks to flow of gradients
- Difficulty with sharper features: artifacts

Differentiable physics: coupling differentiable solver with NN

Differentiable physics: coupling differentiable solver with NN

$$\Delta \mathbf{u} = \partial L / \partial \mathbf{u} = \frac{\partial \mathcal{P}}{\partial \mathbf{u}} \frac{\partial L}{\partial \mathcal{P}}$$



- Physical field **u** input to differentiable solver

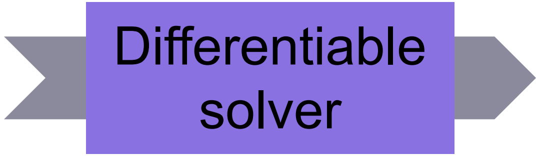
- Find optimal **u**

Differentiable physics: coupling differentiable solver with NN

gradient w.r.t. NN weights

$$\Delta \mathbf{u} = \partial L / \partial \mathbf{u} = \frac{\partial \mathcal{P}}{\partial \mathbf{u}} \frac{\partial L}{\partial \mathcal{P}}$$

$$\Delta \theta = \frac{\partial L}{\partial \theta} = \boxed{\frac{\partial \mathbf{u}}{\partial \theta}} \frac{\partial \mathcal{P}}{\partial \mathbf{u}} \frac{\partial L}{\partial \mathcal{P}}$$

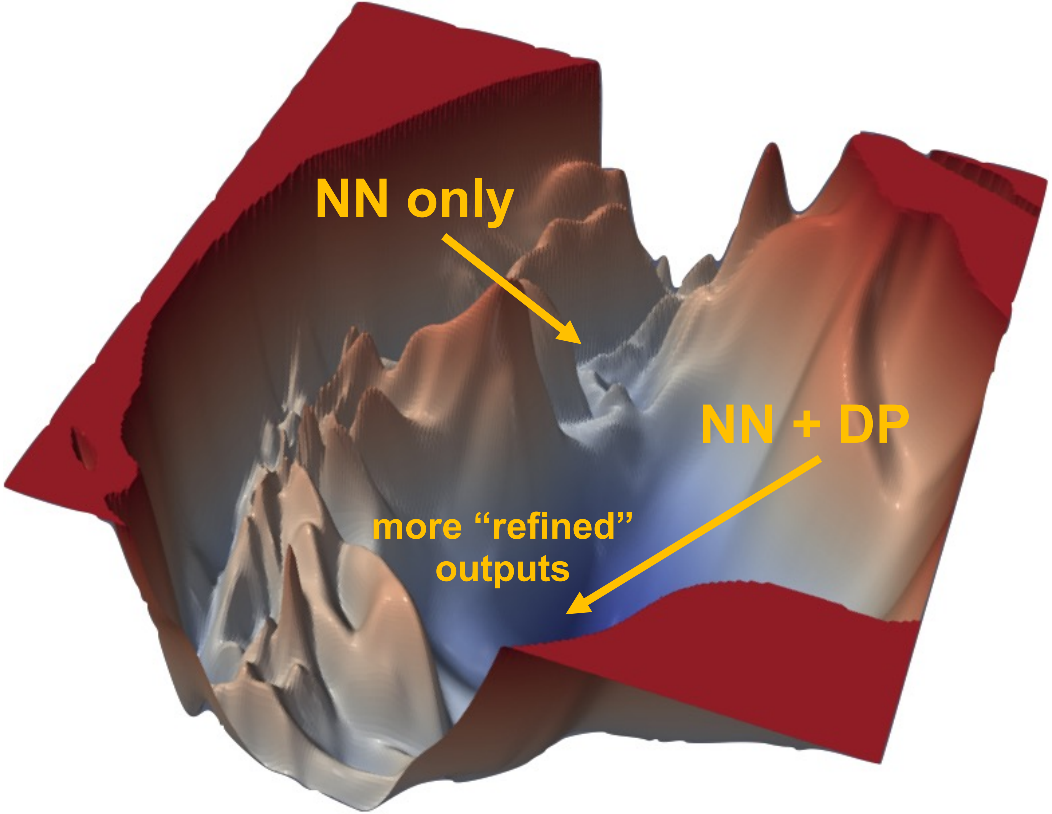


- Physical field **u** input to differentiable solver
- Find optimal **u**

- NN approximates physical field **u**
- Physical field **u** input to differentiable solver
- Find optimal NN weights: gradients are guided by solver

Differentiable physics: coupling differentiable solver with NN

Image source: [cs.umd.edu]



- Gradients from differentiable solver allow to access previously "hidden" parts of the loss landscape

Differentiable physics: pros & cons

- Uses existing numerical tools which can be coupled to the training of neural networks

- More complicated implementation

Differentiable physics: pros & cons

Uses existing numerical tools which can be coupled to the training of neural networks

Efficient evaluation of gradients is possible

More complicated implementation

Requires understanding of the physics to choose suitable discretization

Differentiable physics: pros & cons

Uses existing numerical tools which can be coupled to the training of neural networks

More complicated implementation

Efficient evaluation of gradients is possible

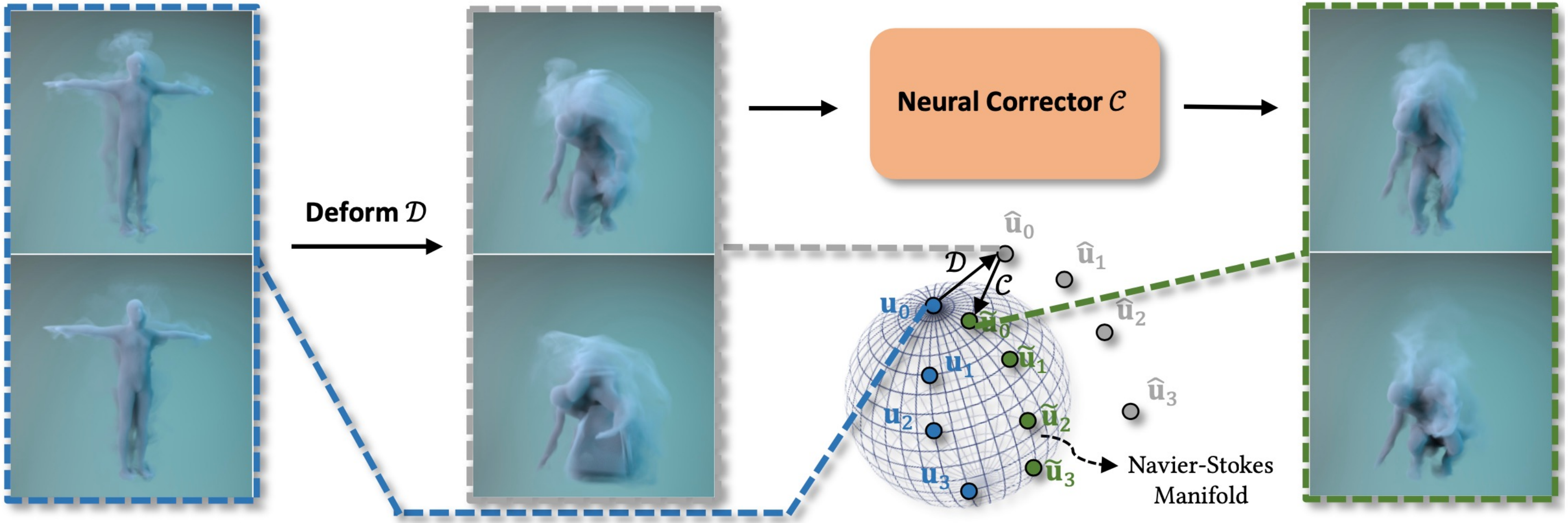
Requires understanding of the physics to choose suitable discretization

Choice of PDE discretization gives control on numerical accuracy

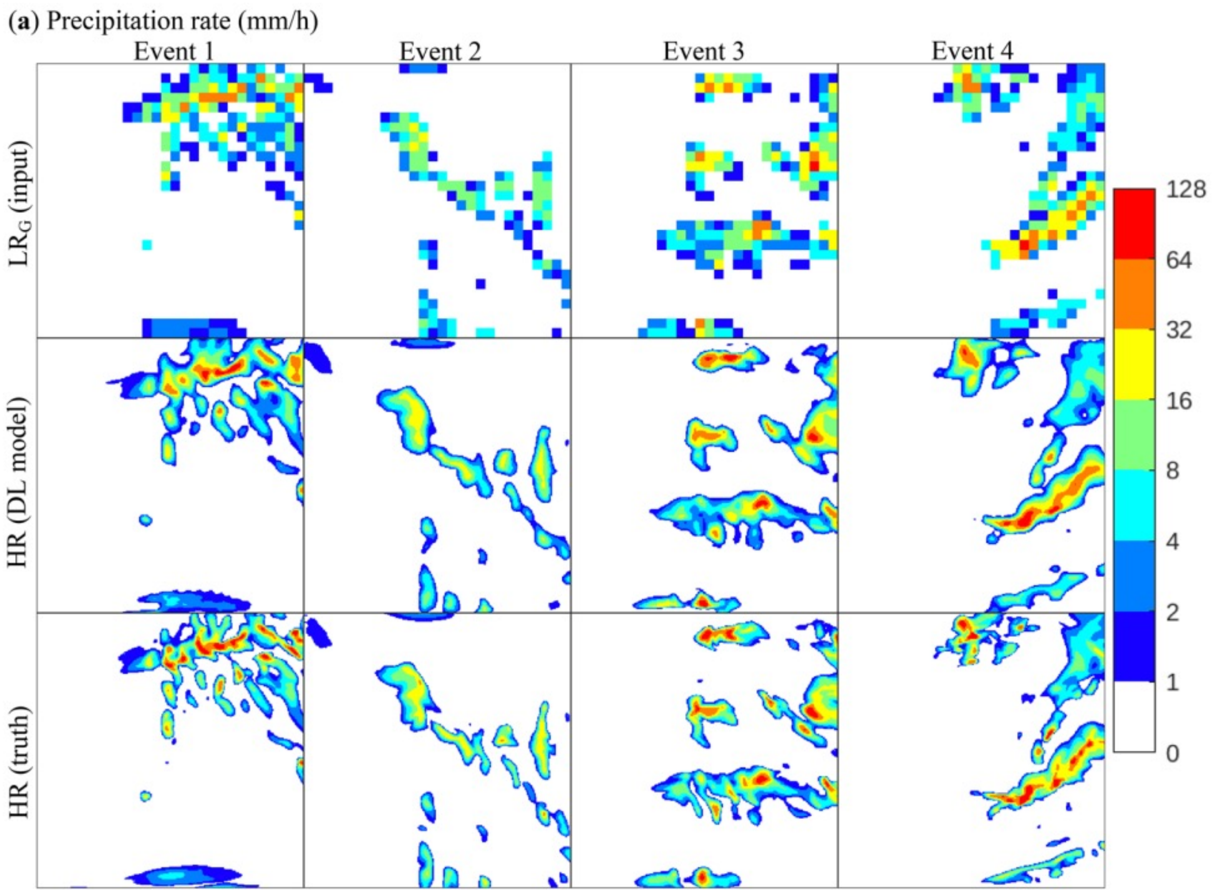
Computational cost and training difficulty scales with time look-ahead

Differentiable physics: application examples

Controlling fluid deformations & reduction of numerical errors, J. Tang et al. 2023 [link]



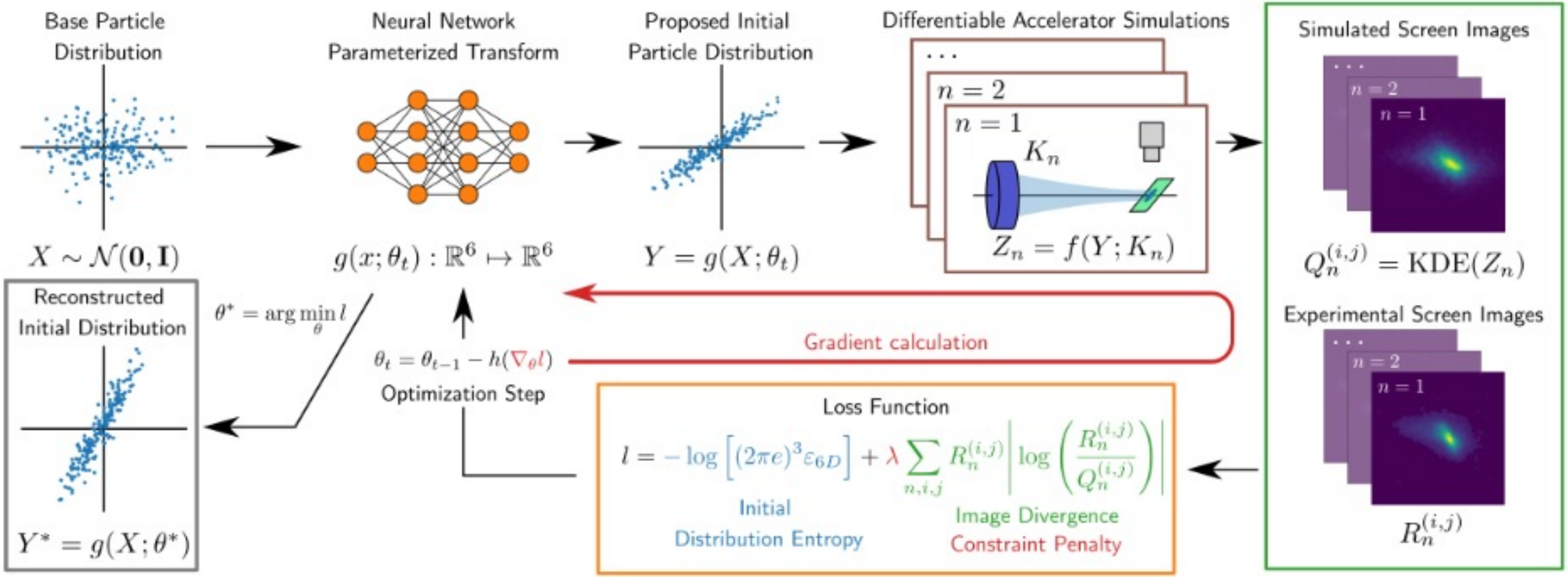
Differentiable physics: application examples



Super-resolution forecasting of precipitation rate, B. Teufel et al. 2023 [doi.org/10.1186/s40562-023-00272-z]

Reconstruction of 6D beam distribution in a particle accelerator,
R. Roussel et al., 2022, [arXiv:2211.09077]

SLAC NATIONAL ACCELERATOR LABORATORY



Summary

- Physics based deep learning is an emerging topic with many exciting possibilities

Summary

- Physics based deep learning is an emerging topic with many exciting possibilities

- AI will not replace classical numerical simulations!

Summary

- Physics based deep learning is an emerging topic with many exciting possibilities
- AI will not replace classical numerical simulations!

Fast simulations

Forecasting

Field reconstruction

Control problems

Numerical error reduction

Equation learning

Further read: examples in high energy physics

- A. Adelmann et al., 2022, New directions for surrogate models and differentiable programming for High Energy Physics detector simulation [doi.org/10.48550/arXiv.2203.08806]

- T. Dorigo et al., 2023, Toward the end-to-end optimization of particle physics instruments with differentiable programming [cds.cern.ch/record/2807001]

- MODE Collaboration, 2021, Toward Machine Learning Optimization of Experimental Design [inspirehep.net/literature/1850892]

- R. Lehe et al., 2020, Machine learning and surrogate models for simulation-based optimization of accelerator design [link]

Further read: examples in weather/climate modeling

- K. Kashinath et al., 2021, Physics-informed machine learning: case studies for weather and climate modelling [doi.org/10.1098/rsta.2020.0093]

- S. Rasp et al., 2021, Data-Driven Medium-Range Weather Prediction With a Resnet Pretrained on Climate Simulations: A New Model for WeatherBench [doi.org/10.1029/2020MS002405]

- J. Pathak et al., 2018, Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model [doi.org/10.1063/1.5028373]

Further read: examples in computer graphics

- S. Zhao et al., 2020, Physics-Based Differentiable Rendering A Comprehensive Introduction [link]
- N. Thuerey et al., 2019, Simulation & Animation [link]
- Y. Wang et al., 2023, Amortizing Samples in Physics-Based Inverse Rendering Using ReSTIR [link]

Further read: examples in DP + reinforcement learning

- J. Degrave et al., 2019, A Differentiable Physics Engine for Deep Learning in Robotics [doi.org/10.3389/fnbot.2019.00006]
- F. de Avila Belbute-Pere et al., 2018, End-to-End Differentiable Physics for Learning and Control [link]
- S. Chen et al., 2022, Imitation Learning via Differentiable Physics [doi.org/10.48550/arXiv.2206.04873]
- J. Lv et al., 2022, SAM-RL: Sensing-Aware Model-Based Reinforcement Learning via Differentiable Physics-Based Simulation and Rendering [doi.org/10.48550/arXiv.2210.15185]

Further read: examples in predictive control/maintenance

- J. Morton et al., 2018, Deep Dynamical Modeling and Control of Unsteady Fluid Flows [doi.org/10.48550/arXiv.1805.07472]

- L. G. Huber et al., 2023, Physics-Informed Machine Learning for Predictive Maintenance: Applied Use-Cases [doi.org/10.1109/SDS57534.2023.00016]

- D. Di Lorenzo et al., 2023, Physics informed and data-based augmented learning in structural health diagnosis [doi.org/10.1016/j.cma.2023.116186]

- V. Jadhav et al., 2022, Physics Informed Neural Network for Health Monitoring of an Air Preheater [doi.org/10.36001/phme.2022.v7i1.3343]