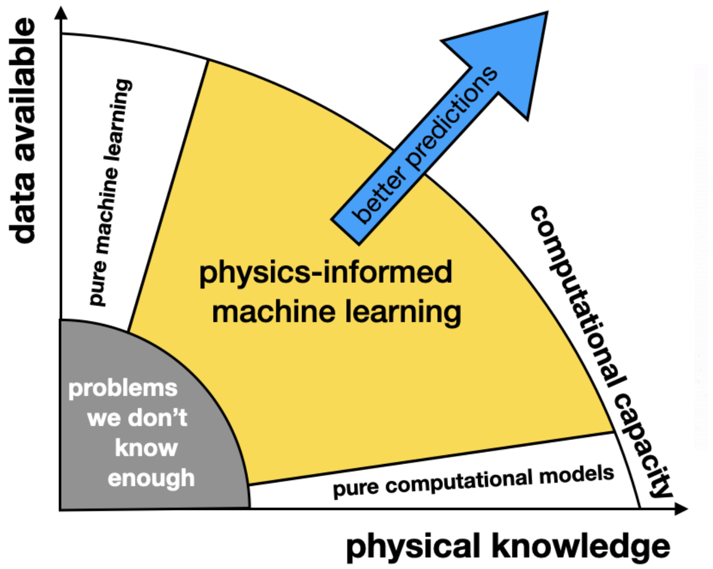
Introduction to Physics-informed AI

Francisco Sahli Costabal

The objective of this course is to provide the foundation to understand research on the field physics-informed machine learning. By end of this class, the attendant should be able to read and understand a current publication on the topic, or at least, know were to find resources to understand it.

This field, as its name suggest, is the intersection of two different areas of knowledge: physics, or more broadly speaking modeling, and machine learning. The idea is to get the best from both worlds. Over the past centuries, humanity has accumulated a vast body of knowledge in the form of mathematical models. Newton's laws are Einstein theory of general relativity are examples of models that can explain many phenomena and were created with little to none data. On the other hand, machine learning has emerged as powerful tool to learn relationships purely based on data. An example of this are voice assistants, trained on houndreds of thousends examples, that are capable of understanding human speech. However, there is a middle ground, where combining data with models can lead to more accurate or faster predictions. This is where physics-informed machine learning can help.



We will start by introducing some concepts both on the modeling side and the machine learning side to help us understand physics-informed machine learning.

Physics models concepts

Partial differential equations

A partial diffential equation (PDE) is an equation involving partial derivatives. What is a partial derivative?

$$\frac{\partial u(x_1, \ldots, x_i, \ldots, x_n)}{\partial x_i} = \lim_{\Delta x \to 0} \frac{u(x_1, \ldots, x_i + \Delta x, \ldots, x_n) - u(x_1, \ldots, x_i, \ldots, x_n)}{\Delta x}$$

Some notation of differential operators:

- Gradient: $\nabla(\circ) = \left[\frac{\partial(\circ)}{\partial x}, \frac{\partial(\circ)}{\partial y}, \frac{\partial(\circ)}{\partial z}\right]$
- Divergence: $\operatorname{div}(\boldsymbol{a}) = \nabla \cdot \boldsymbol{a} = \frac{\partial \boldsymbol{a}}{\partial x} + \frac{\partial \boldsymbol{a}}{\partial y} + \frac{\partial \boldsymbol{a}}{\partial z}$
- Laplacian: $\Delta(\circ) = \nabla \cdot \nabla(\circ) = \operatorname{div}(\nabla(\circ)) = \frac{\partial^2(\circ)}{\partial x^2} + \frac{\partial^2(\circ)}{\partial y^2} + \frac{\partial^2(\circ)}{\partial z^2}$

Some well known partial differential equations are the heat equation: $\frac{\partial u}{\partial t} = \operatorname{div}(k \nabla u)$

and the wave equation: $\frac{\partial^2 u}{\partial t^2} = \operatorname{div}(c^2 \nabla u)$.

However, there are an infinite amount of solutions to these equations. We get one solution, we need to specify **boundary conditions** to inform what is happening where the domain that we are considering ends. For second order PDEs (the most common), we have:

- Dirichlet boundary conditions: restrict the quantity itself, $u = g$.
- Neumann boundary conditions: restrict the gradient of the quantity in the direction normal to the boundary: $\nabla u \cdot \boldsymbol{n} = h$

There are some other boundary condition types that are combinations of these two. If we specify boundary conditions in the entire boundary, we have defined a **boundary value problem**, which in many cases can be solved. If the problem is time dependent, we also need to specify an initial condition on the entire domain.

Finally, most PDEs don't have an analytical solution, which is why we need to resort to numerical methods, such as:

- Finite elements
- Finite differences
- Finite volumes

Forward problem

Goes from the input to the output. This can be a boundary value problem, or an initial value problem. The input can be the boundary conditions or some parameters of the model. The output is the unknown variable.

Example: given boundary conditions and the heat equation, find the temperature inside the heat sink.

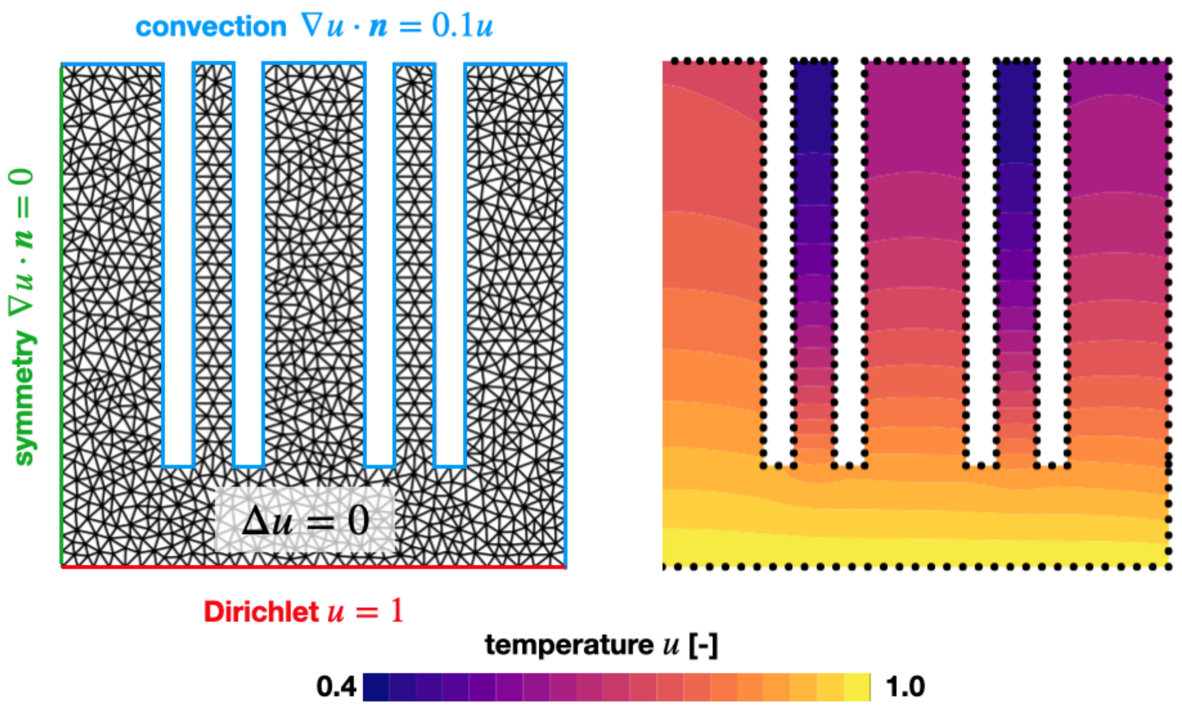

convection $\nabla u \cdot \boldsymbol{n} = 0.1u$
symmetry $\nabla u \cdot \boldsymbol{n} = 0$
$\Delta u = 0$
Dirichlet $u = 1$
temperature u [-]
0.4
1.0

Figure 1: Solution of a steady state heat equation in a heat sink

Inverse problem

Goes from the output to the input. Typically, we have partial observations of the output and we want to reconstruct both the entire output and part of the input. Or we have observed the entire output field and we want to recover the input. The input here can be parameters or boundary conditions as well. Note that these problems are often ill-posed (i.e. multiple solutions satisfy the problem statement) and regularization is used. Traditionally, these problems are solved via optimization of some functional:

$$\min_u D(u(\mathbf{x}), \hat{u}) + \alpha R(u(\mathbf{x}))$$

where D is distance function that measures the mismatch between the predicted function u and the measurements $\hat{u}$, and R is a regularization term. **Example:** magnetic resonance image reconstruction

Inverse problems constraint by partial differential equations: in this case, we know that there is a model in the form of partial differential equations that must be satisfied for any configuration. Here, we are more interested in learning some parameters θ of the phenomena, rather than the output quantity u. We can also phrase this as a minimization problem:

$$\min_\theta D(u(\mathbf{x}), \hat{u}) + \alpha R(\theta), \text{ s.t. } F(u, \theta) = 0$$

where $F(u, \theta) = 0$ represents the constraint that our model described with partial differential equations is satisfied at all times.

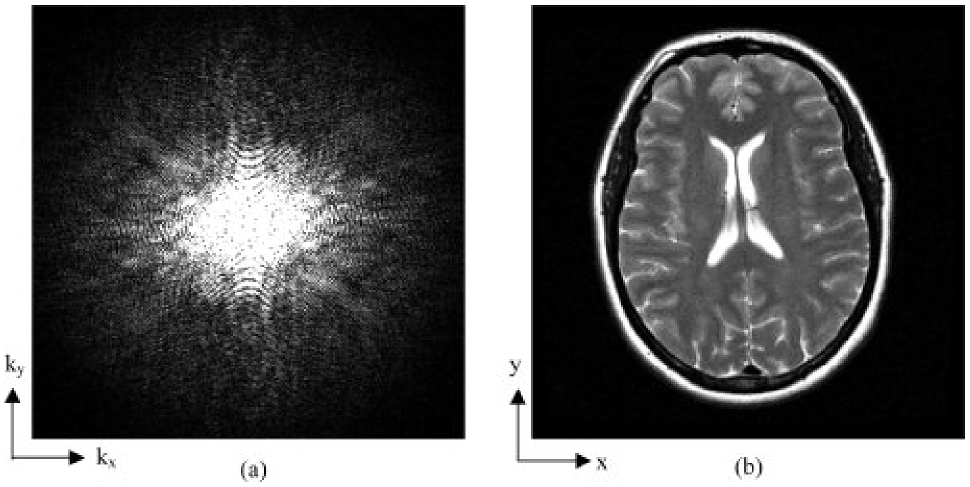


Figure 2: Reconstruction of MRI (right) from k-space. Image from: C. Paschal et al. Journal of magnetic resonance imaging (2004): JMRI. 19. 145-59.

Example: estimate active strain/stress in the heart from echocardiography images. The model F is the equations of non-linear elasticity.

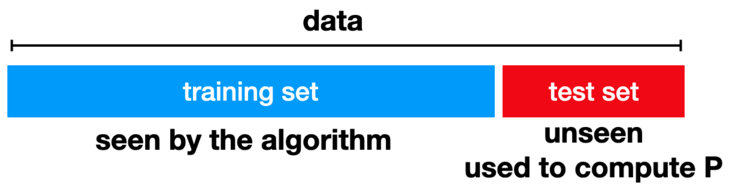
Machine learning concepts

This section is based on the deep learning book

What is machine learning?

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell 1997)

- the task represents the intended use of the machine learning algorithm. Nowadays, many tasks can be performed with machine learning algorithms:
  - translation (Google translate)
  - speech recognizition (Alexa, Siri, Google Assistant, etc…)
  - image generation from text
  - learning pressure distributions from velocities in vessels
- the performance measure P is the metric that we will use to assess how well the machine learning algorithm is performing. We need to leave reserve some data, called test set, to evaluate the performance of the algorithm.



- the experience E, typically represent the data used to train the algorithm. However, in the context of physics-informed machine learning, the physical model also becames part of the experience as the algorithm can also learn from it.

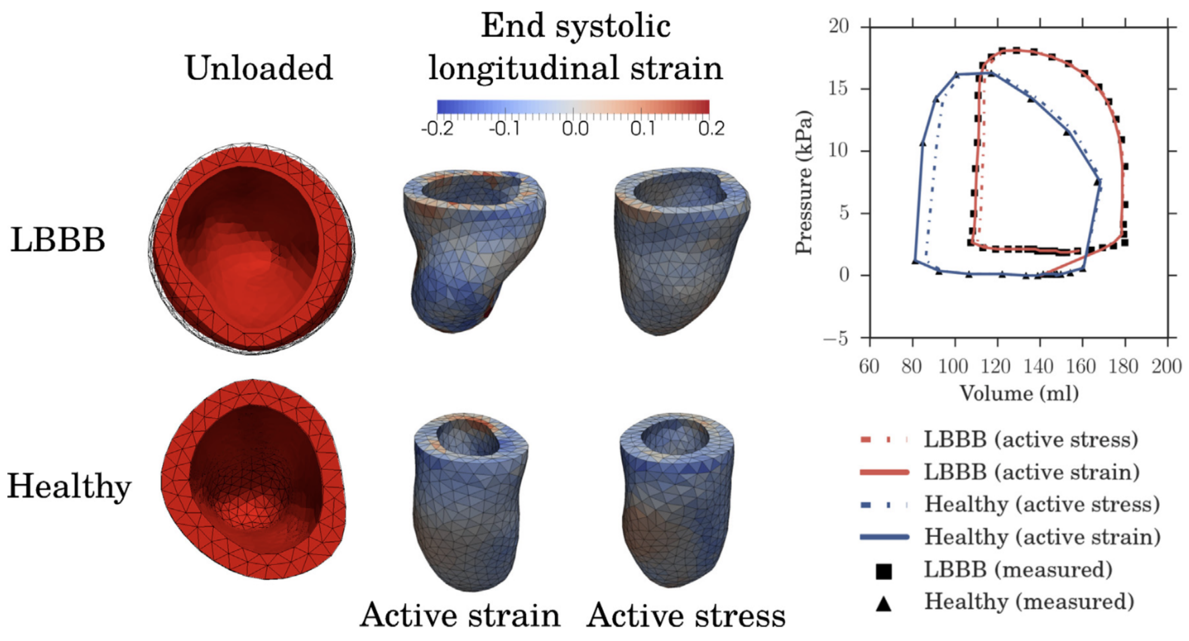
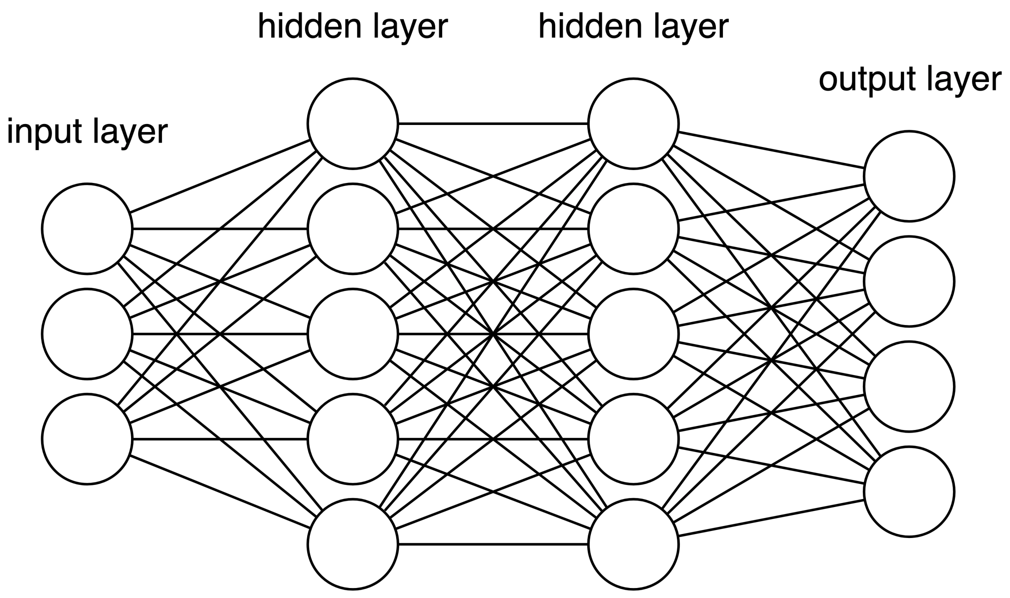


Figure 3: Image from: H. Finsberg et al. / Journal of Computational Science 24 (2018) 85–90

Deep learning

Deep learning a branch of machine learning, that uses neural networks as its primary algorithm. A feedforward neural network defines a function $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ that maps some input $\boldsymbol{x}$ to the ouput $\boldsymbol{y}$ and learns the value of the parameters $\boldsymbol{\theta}$ that results in the best approximation.



They are called feedforward because the flow of information is unidirectional. And they are called networks because they are represented by composing multiple functions (or layers):

$$f(\boldsymbol{x};\boldsymbol{\theta}) = f_n(...f_2(f_1(\boldsymbol{x};\boldsymbol{\theta}_1);\boldsymbol{\theta}_2);...;\boldsymbol{\theta}_n)$$

In the most simple architecture, each of the functions f_i is parametrized by a matrix of weights $\vec{W}_i$ and a vector of biases $\vec{b}_i$:

$$\vec{h}_i = f_i(\vec{h}_{i-1};\vec{\theta}_i = \{\vec{W}_i, \vec{b}_i\}) = g(\vec{W}_i^T \vec{h}_{i-1} + \vec{b}_i)$$

where $g(\circ)$ is a non-linear activation function applied element-wise. There are multiple activation functions and sometimes they can have a big influence on the performance of on network. The output layer will have a different activation function depending on the task. In physics-informed neural networks is very common to have a linear output layer $g(x) = x$, because we are estimating quantities that belong to the real numbers.

Training neural networks

So far we have described what is neural network, but we don't have to make them learn from data. To do this, we will define a loss function $\mathcal{L}$ that depends on the N input/output pairs that we have $\{\vec{x}_i, y_i\}_{i=1}^N$ and the network parameters $\vec{\theta}$. We will attempt to find the parameters $\vec{\theta}$ that minimize the loss function $\mathcal{L}$. For multiple reasons, a global minimum is almost never achieved, which is why we will be happy to find parameters that reduce the loss function significantly. One common loss function is the mean squared error, which is used when the output is a real number:

$$\mathcal{L}_{MSE}(\vec{\theta};\{\vec{x}_i, y_i\}_{i=1}^N) = \frac{1}{N}\sum_i^N (y_i - f(\vec{x}_i;\vec{\theta}))^2$$

To minimize the loss function we will use some flavor of an algorithm called **stochastic gradient descent**, which performs the following update for a certain amount of iterations:

$$\vec{\theta} \leftarrow \vec{\theta} - \epsilon \nabla_{\vec{\theta}} \mathcal{L}$$

where ϵ is the **learning rate** which can have a huge impact on the training process.

Physics-informed machine learning

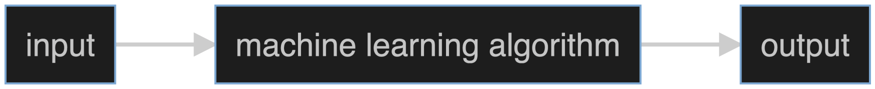
There are different approaches to physics-informed machine learning, with different level of integration between the model and the machine learning algorithm. We will start with the simplest.

Black box (surrogate model)

In this approach we will use some technique to solve the forward problem (finite differences, finite elements, etc…) and generate a dataset to train a machine learning algorithm (not necessarily a neural network). As usual, we need to split the dataset into training and test sets to assess the accuracy of our surrogate model.

Applications in forward problems

- Accelerated prediction
- Uncertainty quantification
- Sensitivity analysis



Here, the input can be model parameters, or boundary conditions; and the output can be a quantity of interested produced by running the model.

Example: fast prediction of electrocardiogram properties in response to drugs using a high resolution cardiac model and multi-fidelity Gaussian process regression.

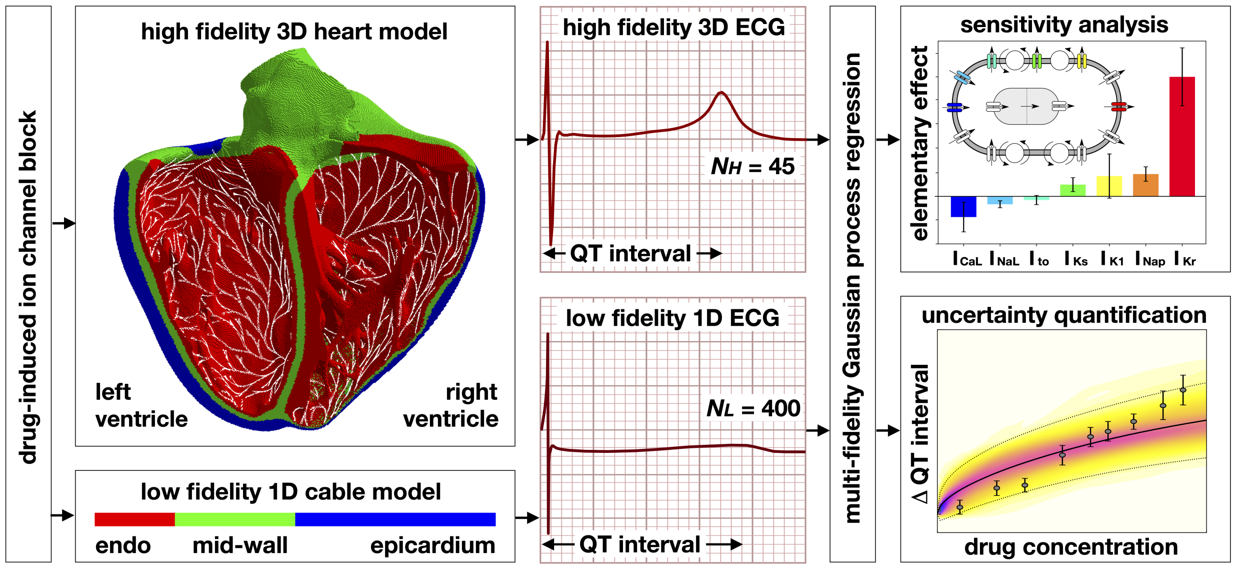
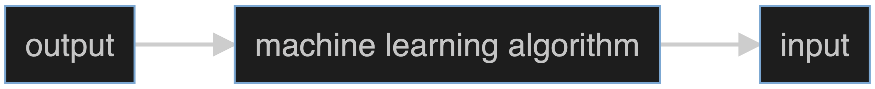

drug-induced ion channel block
high fidelity 3D heart model
left ventricle
right ventricle
low fidelity 1D cable model
endo
mid-wall
epicardium
high fidelity 3D ECG
N_H = 45
QT interval
low fidelity 1D ECG
N_L = 400
QT interval
multi-fidelity Gaussian process regression
sensitivity analysis
elementary effect
I_{CaL} I_{NaL} I_{to} I_{Ks} I_{K1} I_{Nap} I_{Kr}
uncertainty quantification
Δ QT interval
drug concentration

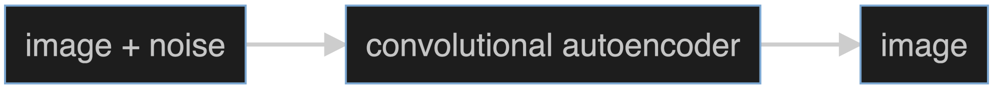
Figure 4: Image from: F. Sahli Costabal et al. / Computer Methods in Applied Mechanics and Engineering 348 (2019) 313–333

Applications in inverse problems

- Denoising
- Parameter identification



Example: denoising medical images with a convolutional autoencoder. Here, the model can be considered the type of noise that is generated during the image acquisition.



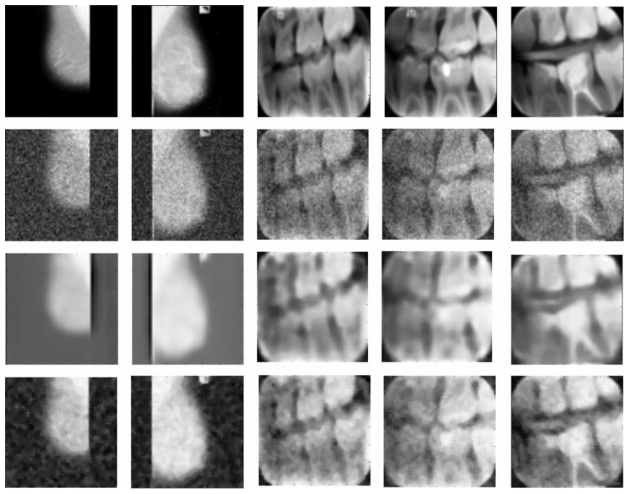


Figure 5: 1st row: orignal image, 2nd row: image corrupted with noise, 3rd row: results of autoencoder, 4th row: results of a median filter. Image from: Gondara, 2016 IEEE 16th International Conference on Data Mining Workshops.

Advantages of the black box approach:

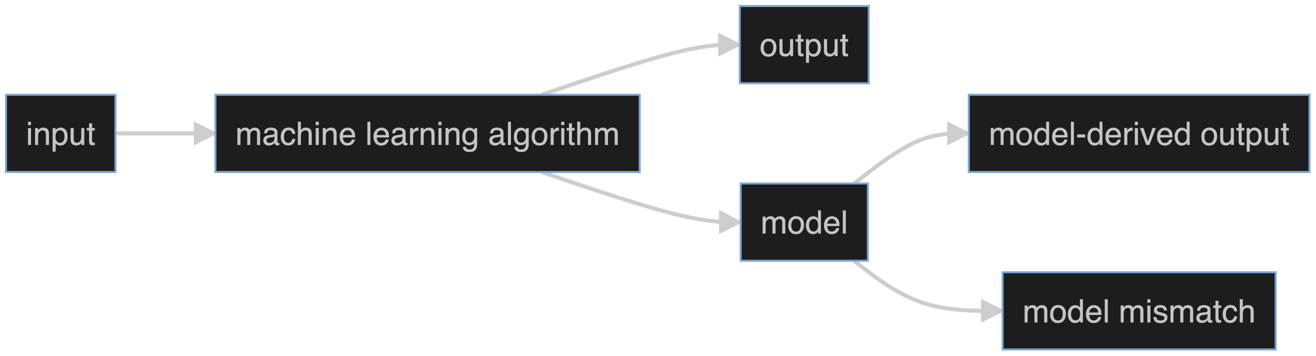
- Inexpensive to evaluate, thus it can be evaluated many times more than the original model with a fixed computational budget.
- No need to adapt the model, because it is treated as a black box.

Disadvantages:

- Creating the training set, specially when the input dimension is high, can be very expensive.
- We need to predefine an input space that will produce an output space, which might be different to the target output.

Intrusive approach

In this approach, we will incorporate the model into the machine learning model. Although this can be done with some machine learning algorithms, it most suited for neural networks because of their universal approximation properties. Although forward problems can be solved with this approach, they are typically not as efficient as specialized solvers (finite elements, finite differences, etc…). For inverse problems, we can have a general approach:



Example only using model-derived outputs: neural radiance fields (NeRF)

Physics-informed neural networks

We will focus more on this approach, which was introduced by Raissi, Perdikaris & Karniadakis in 2017. Let's consider a problem where we have partial observations of the unknown variable $u(\vec{x})$ and we have a model in the form of a PDE, which can be written as:

$$\mathcal{N}[u, \boldsymbol{\lambda}] = 0$$

where $\mathcal{N}[\circ, \boldsymbol{\lambda}]$ is, in general, a non-linear differential operator parametrized by λ.

Example: Poisson equation

$$\mathcal{N}[u, \lambda] = \Delta u - \lambda = 0$$

We have partial observations of the output u_i, which are located in domain at positions $\vec{x}_i$. We may also have data about the boundary conditions, such as Neumann boundary conditions $\nabla u \cdot \vec{n} = g_i$ at boundary locations $\vec{x}_i^b$.

We now approximate the unknown function with a neural network:

$$u(\boldsymbol{x}) \approx \hat{u}(\boldsymbol{x}; \boldsymbol{\theta}) = NN(\boldsymbol{x}; \boldsymbol{\theta})$$

parametrized with trainable parameters $\vec{\theta}$. We postulate a loss function composed of 3 terms:

$$\begin{aligned} \mathcal{L}(\{u_i, \boldsymbol{x}_i\}, \{g_i, \boldsymbol{x}_i^b\}, \boldsymbol{\theta}, \boldsymbol{\lambda}) &= MSE_{data}(\{u_i, \boldsymbol{x}_i\}, \boldsymbol{\theta}) \\ &+ MSE_{PDE}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \\ &+ MSE_{boundary}(\{g_i, \boldsymbol{x}_i^b\}, \boldsymbol{\theta}) \end{aligned}$$

The first term is just the mean squared error loss that we have seen previously, and encourages the network to learn the available data:

$$MSE_{data}(\{u_i, \boldsymbol{x}_i\}, \boldsymbol{\theta}) = \frac{1}{N} \sum_i^N (u_i - \hat{u}(\boldsymbol{x}_i; \boldsymbol{\theta}))^2$$

The second term can be seen as a model mismatch and encourages the network to satisfy the equation encoded in $\mathcal{N} = 0$:

$$MSE_{PDE}(\boldsymbol{\theta}, \boldsymbol{\lambda}) = \frac{1}{R} \sum_i^R (\mathcal{N}[\hat{u}(\boldsymbol{x}_i^r, \boldsymbol{\theta}; \boldsymbol{\lambda}])^2$$

Here, we will evaluate the PDE at locations $\vec{x}_i^r$, called "collocation points", that can be generated randomly at every optimization iteration or can have predefined values. Minimizing this term will ensure that the model is approximately satisfied in the entire domain. We can also leave the parameters of the PDE $\vec{\lambda}$ as trainable variables that will be learned during the optimization process.

Finally, we have a boundary term that ensures the boundary conditions are satisfied:

$$MSE_{boundary}(\{g_i, \boldsymbol{x}_i^b\}, \boldsymbol{\theta}) = \frac{1}{B} \sum_i^B (\nabla \hat{u}(\boldsymbol{x}_i^b; \boldsymbol{\theta}) \cdot \boldsymbol{n} - g_i)^2$$

Note that is we have Dirichlet boundary conditions we can simply add them as data. Now, we will train the network to minimize the loss function to obtain a solution that approximates the data, the model and boundary conditions.

Example: Burger's equation

$$\mathcal{N}[u(x,t),\lambda] = \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \lambda\frac{\partial^2 u}{\partial x^2} = 0$$

with $x \in [-1,1]$, $t \in [0,1]$, initital condition: $u(x,0) = -\sin(\pi x)$, boundary conditions: $u(-1,t) = u(1,t) = 0$. Note that we don't have Neumman boundary conditions, so we don't need the $MSE_{boundary}$ term in the loss function.

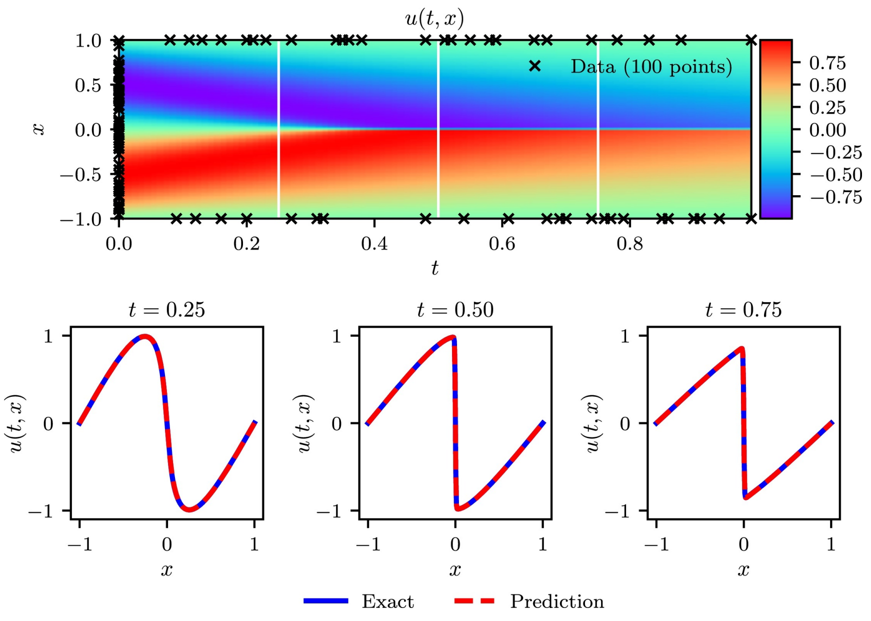


Figure 6: Solution of Burgers equation with physics-informed neural networks. Image from: M. Raissi et al. / Journal of Computational Physics 378 (2019) 686–707

Advantages of the intrusive approach:

- the neural network can "see" the entire model and learn from it with gradient-based optimization. Taking this gradient would be a nightmare to do by hand, but it is only one line of code with modern machine learning languages.
- can bypass some conditions that would make the inverse problem ill-posed, such as boundary conditions
- useful when we are not sure if the model is satisfied exactly.

Disadvantages:

- sometimes the models can be too complex to be coded in a machine learning language.
- can be slow and inaccurate to solve forward problems without any data.