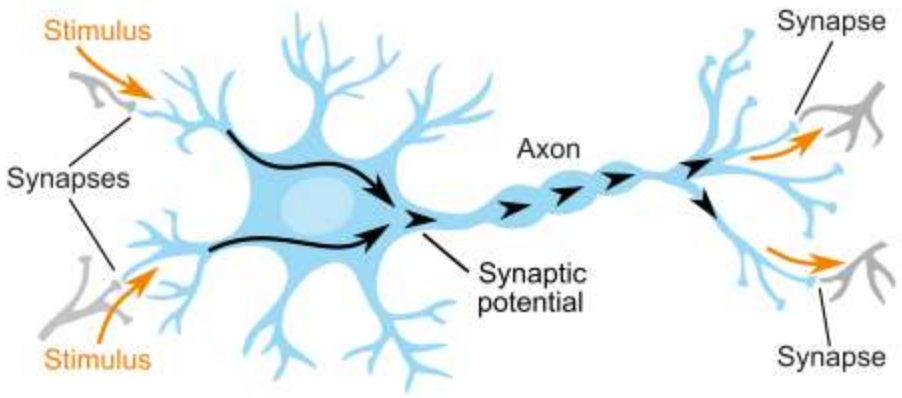
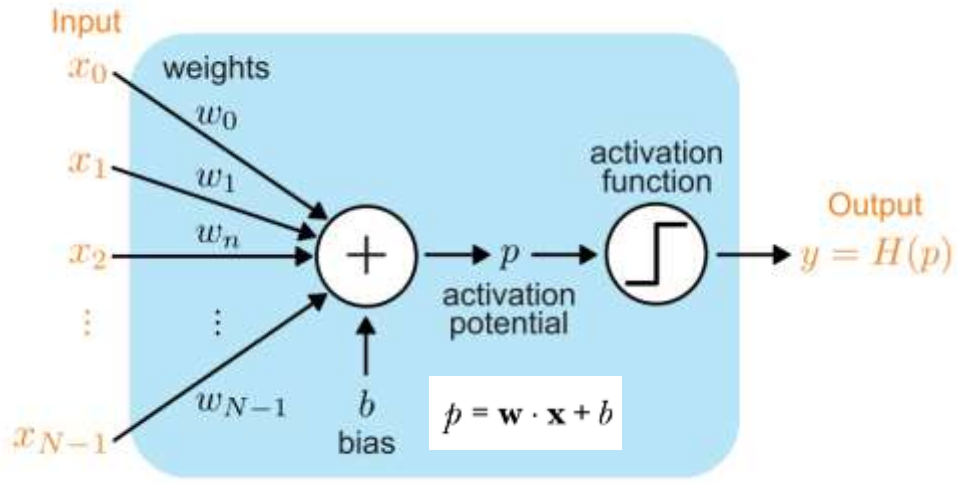
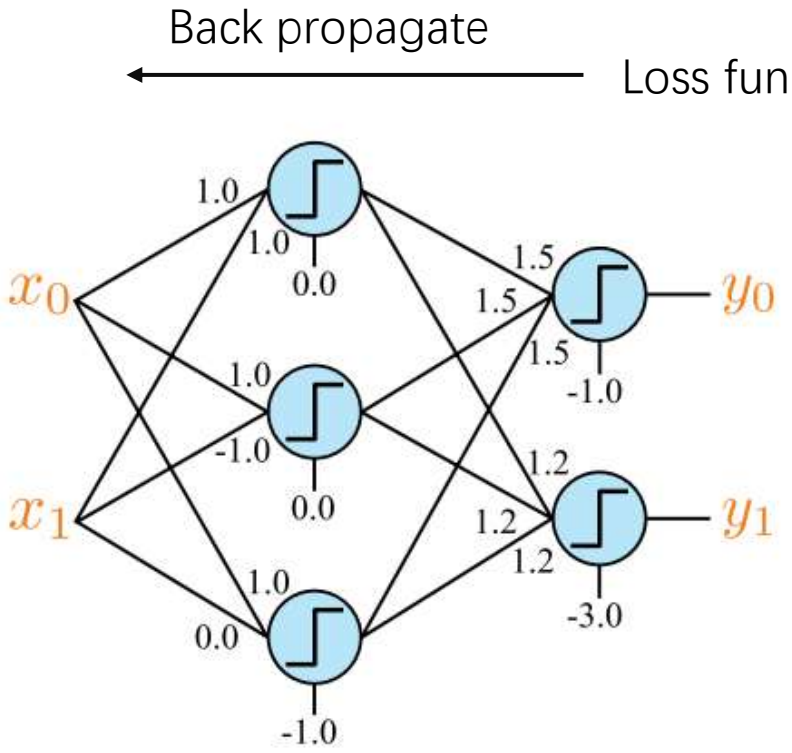
An Introduction of Physics- informed machine learning

Xinwen Zhang
2026/03/21

Artificial Neural Networks

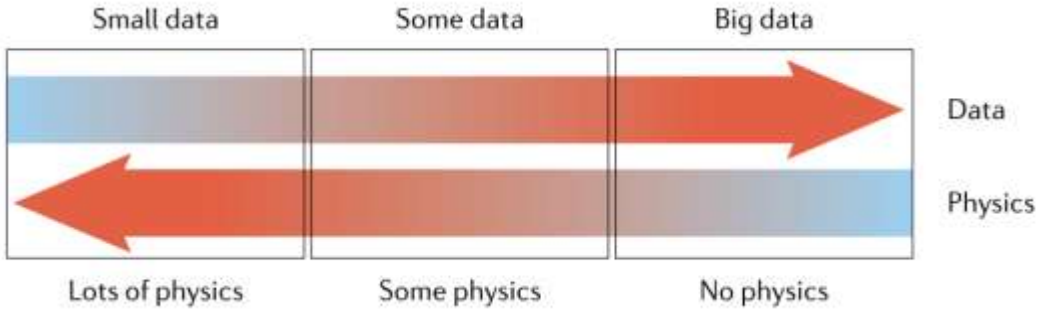




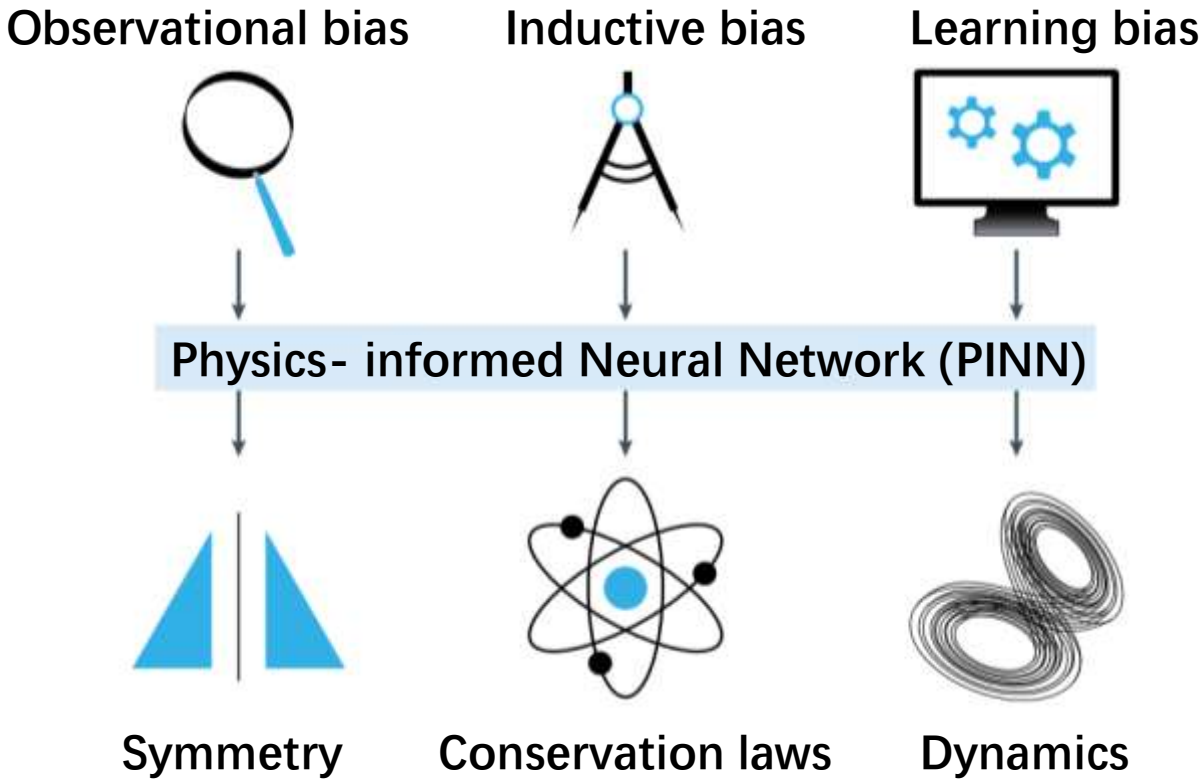


A two-layer dense neural network, fully connected neural networks or multi-layer perceptrons (MLP).

Principles of physics-informed learning

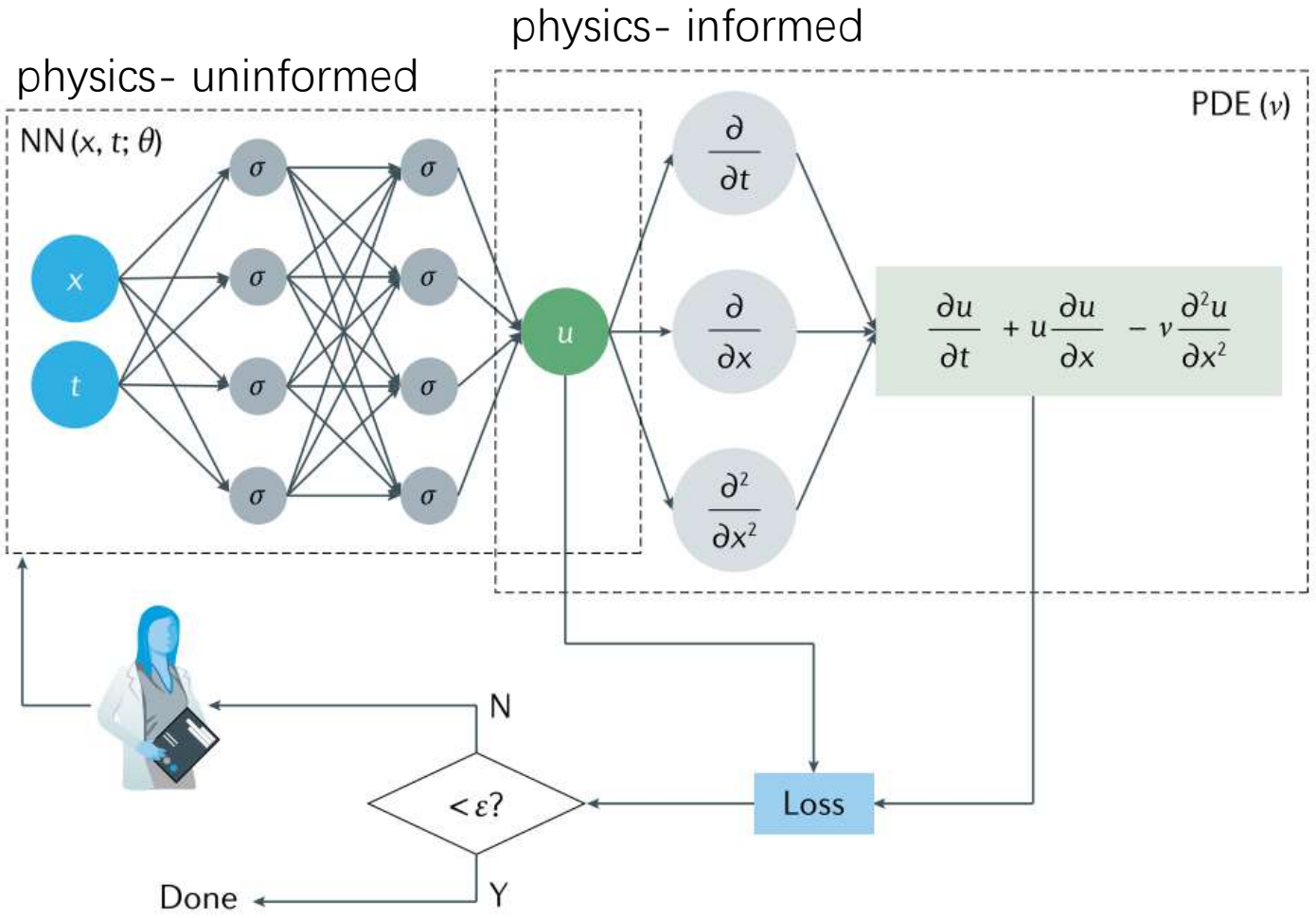


- Small data: e. g. PDE + known parameters
- Intermediate: e. g. PDE + unknown parameters
- Big data : e. g. unknown PDE



Karniadakis, G.E., Kevrekidis, I.G., Lu, L.*et al. Nat Rev Phys*, 2021

The standard framework of Physics-informed neural networks



Viscous Burgers' equation:

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

Loss function

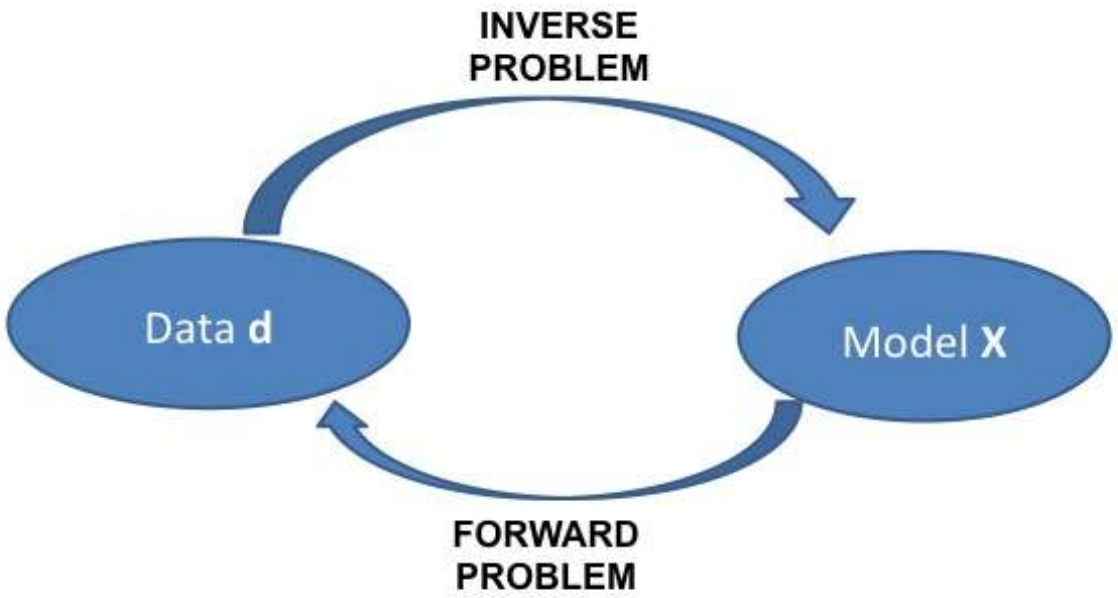
$$\mathcal{L}=w_{\text{data}}\mathcal{L}_{\text{data}}+w_{\text{PDE}}\mathcal{L}_{\text{PDE}},$$

$$\begin{cases}\mathcal{L}_{\text{data}}=\dfrac{1}{N_{\text{data}}}\sum_{i=1}^{N_{\text{data}}}\left(u(x_i,t_i)-u_i\right)^2 & \text{and}\\ \mathcal{L}_{\text{PDE}}=\dfrac{1}{N_{\text{PDE}}}\sum_{j=1}^{N_{\text{PDE}}}\left(\dfrac{\partial u}{\partial t}+u\dfrac{\partial u}{\partial x}-\nu\dfrac{\partial^2 u}{\partial x^2}\right)^2\Big|_{(x_j,t_j)}.\end{cases}$$

Note: L_{data} is from the points under initial conditions and boundary conditions

Optimation: Adam or L-BFGS

Forward problem and Inverse problem



Forward Problem: Given the model, parameters, and boundary conditions, how can we predict the state or time evolution of a system through simulation?

Inverse Problem: Given the observation results, how can we deduce the equation parameters?

Problem Setup

- Parameterized, nonlinear PDE(s)

$$u_t + \mathcal{N}[u;\lambda] = 0, x \in \Omega \subset \mathbb{R}^D,\ t \in [0,T]; \quad (\cdot)_t = \frac{\partial(\cdot)}{\partial t}$$

- where $u(t,x)$ denotes the latent (hidden) solution, $\mathcal{N}[\cdot;\lambda]$ is a nonlinear operator parametrized by λ
- The above setup covers a wide range of PDEs in math. physics, including conservation laws, diffusion, reac-diff-advec. PDE, kinetics etc. E.g., Burger's equation in 2D

$$\mathcal{N}[u;\lambda] = \lambda_1 u u_x - \lambda_2 u_{xx} \text{ and } \lambda = (\lambda_1, \lambda_2); \quad (\cdot)_x = \frac{\partial(\cdot)}{\partial x} \quad (\cdot)_{xx} = \frac{\partial^2(\cdot)}{\partial x^2}$$

- Given λ what is $u(t,x)$ (Inference, filtering and smoothing or simply **data-driven solutions of PDEs**) → Forward problem
- Find λ that best describes observations $u(t_i, x_j)$ (Learning, system identification, or **data-driven discovery of PDEs**) → Backward problem

M. Raissi, P. Perdikaris, G.E. Karniadakis, Journal of Computational Physics, 2018

Data-driven solutions of partial differential equations (Forward problem)

- Rewrite the PDE as $f(u; t, x) = 0$

$$f(u; t, x) \doteq u_t + \mathcal{N}[u], \quad \text{along with} \quad u = u_\theta(t, x)$$

- Along with the above constraint this gives *Physics-informed neural network* **parameterized by** θ

$$\mathcal{L} = \mathcal{L}_u + \mathcal{L}_f$$

$$\mathcal{L}_u = \frac{1}{N_u} \sum_{i=1}^{N_u} \left| u\left(t_u^i, x_u^i\right) - u^i \right|^2 ; \quad \mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \left| f\left(t_f^i, x_f^i\right) \right|^2$$

- $\left\{t_u^i, x_u^i, u^i\right\}_{i=1}^{N_u}$ denote the **initial and boundary training data** on $u(t, x)$
- $\left\{t_f^i, x_f^i\right\}_{i=1}^{N_f}$ specify the **collocation points** for $f(u; t, x)$
- $\mathcal{L}_u$ helps to enforce initial and boundary data accurately, while $\mathcal{L}_f$ imposes the structure of the PDE into the total loss

Data-driven discovery of partial differential equations (Inverse problem)

Define a PDE residual:

$$f := u_t + \mathcal{N}[u; \lambda]$$

The difference from forward problems is that the parameters in the differential operator directly become learnable parameters of the PINN.

Navier-Stokes Equations

- Describe the physics of many phenomena, such as weather, ocean currents, water flow in a pipe and air flow around a wing.

$$\begin{aligned} u_t + \lambda_1 (u u_x + v u_y) &= -p_x + \lambda_2 (u_{xx} + u_{yy}) \\ v_t + \lambda_1 (u v_x + v v_y) &= -p_y + \lambda_2 (v_{xx} + v_{yy}) \end{aligned}; \quad \text{where} \quad (\cdot)_x = \frac{\partial (\cdot)}{\partial x}$$

- $u(t, x, y)$ denotes the x-component of the velocity, $v(t, x, y)$ denotes the y component and $p(t, x, y)$ the pressure
- Conservation of mass: $u_x + v_y = 0 \implies u = \psi_y, \quad v = -\psi_x$
- Given a set of observations: $\left\{t^i, x^i, y^i, u^i, v^i\right\}_{i=1}^N$

$$\begin{aligned} f &\doteq u_t + \lambda_1 (u u_x + v u_y) + p_x - \lambda_2 (u_{xx} + u_{yy}) \\ g &\doteq v_t + \lambda_1 (u v_x + v v_y) + p_y - \lambda_2 (v_{xx} + v_{yy}) \end{aligned}$$

- Learn $\lambda = \{\lambda_1, \lambda_2\}$, and pressure field $p(t, x, y)$ by jointly approximating $[\psi(t, x, y) \quad p(t, x, y)]$ with a single NN with two outputs

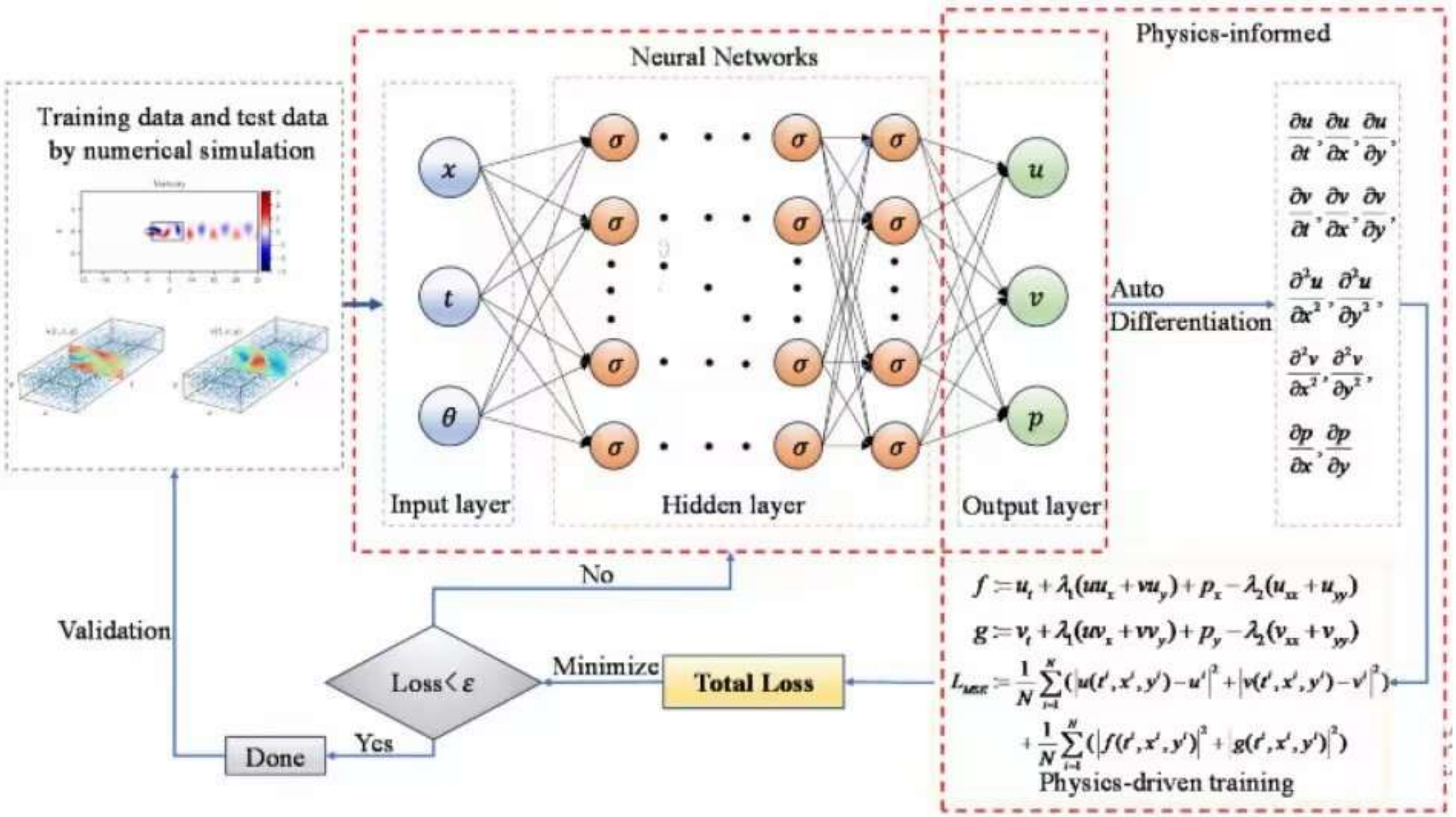
Navier-Stokes Equations

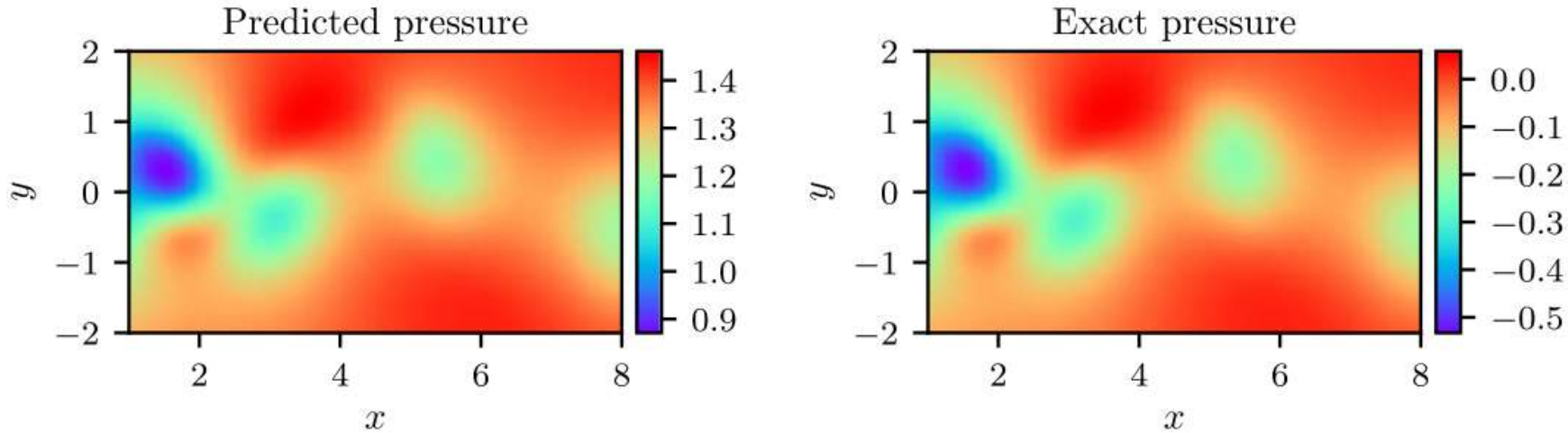
- Train by minimizing the total loss

$$\mathcal{L} \doteq \frac{1}{N}\sum_{i=1}^{N}\left(\left|u\left(t^i,x^i,y^i\right)-u^i\right|^2+\left|v\left(t^i,x^i,y^i\right)-v^i\right|^2\right)$$
$$+\frac{1}{N}\sum_{i=1}^{N}\left(\left|f\left(t^i,x^i,y^i\right)\right|^2+\left|g\left(t^i,x^i,y^i\right)\right|^2\right)$$

- Consider the prototypical problem of an incompressible flow past a (rigid) cylinder, with various values of the Reynolds number $Re = u_\infty D/\nu$
- Training data is generated using a high-res spectral solver (NekTar)

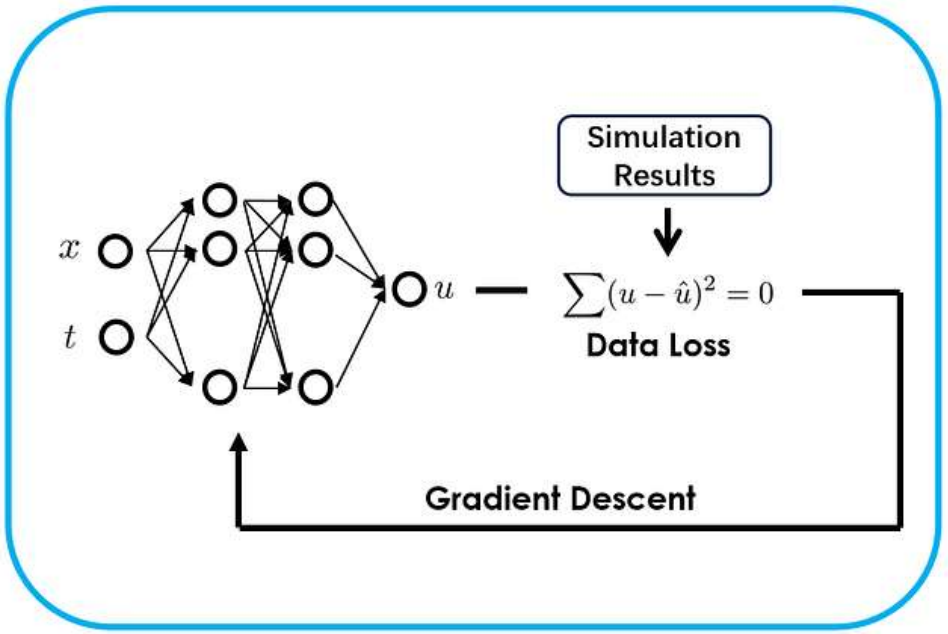
Navier-Stokes Inverse Problem (PINN structure)



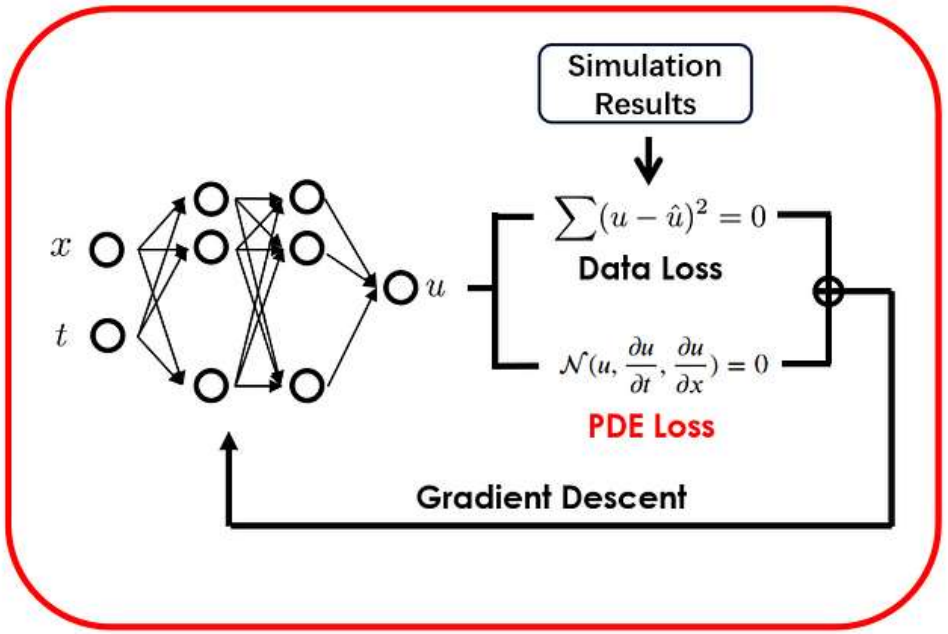


Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.01(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y + 0.01(v_{xx} + v_{yy})$
Identified PDE (clean data)	$u_t + 0.999(uu_x + vu_y) = -p_x + 0.01047(u_{xx} + u_{yy})$ $v_t + 0.999(uv_x + vv_y) = -p_y + 0.01047(v_{xx} + v_{yy})$
Identified PDE (1% noise)	$u_t + 0.998(uu_x + vu_y) = -p_x + 0.01057(u_{xx} + u_{yy})$ $v_t + 0.998(uv_x + vv_y) = -p_y + 0.01057(v_{xx} + v_{yy})$

Summary



Tranditional NN



Physics-informed NN

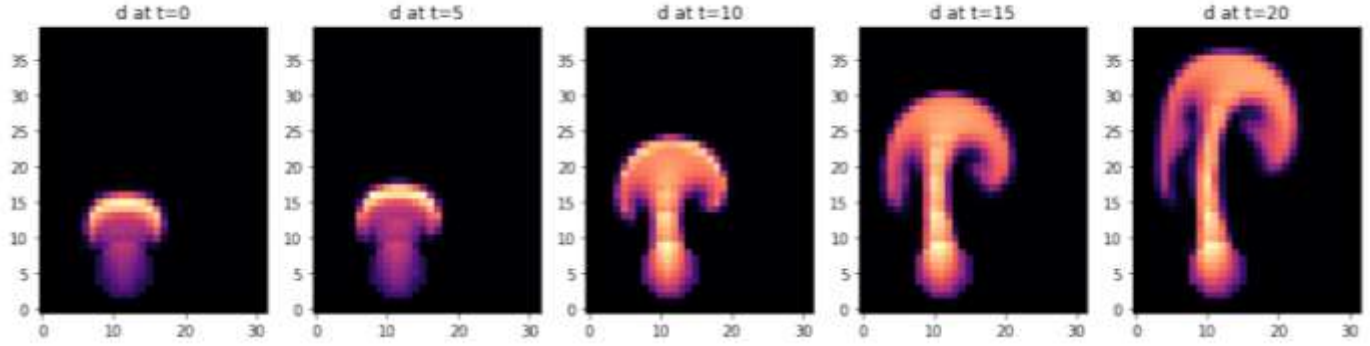
- Physics-informed neural networks is a class of universal function approximators that are capable of encoding any underlying physical laws that govern a given data-set
- They can be used to address two main types of problems: data-driven solution of partial differential equations and data-driven discovery of partial differential equations.

Navier-Stokes Forward Simulation (Classical)

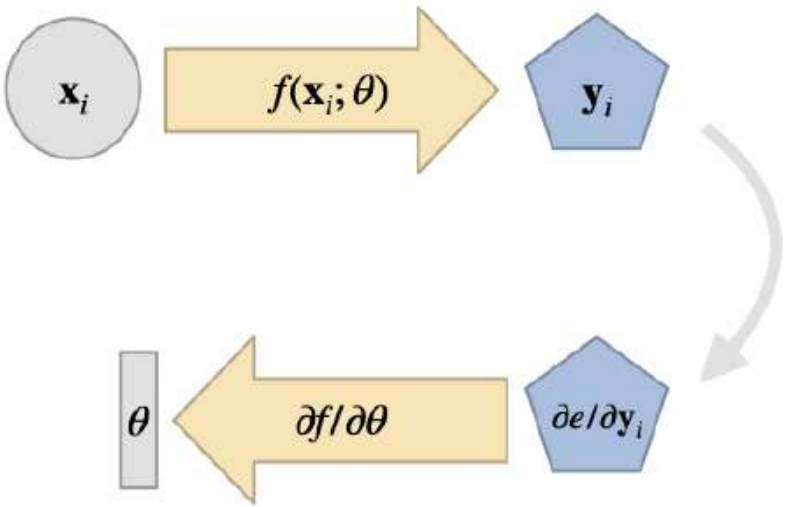
```
steps = [[ smoke.values, velocity.values.vector[0], velocity.values.vector[1] ]]
for time_step in range(20):
  if time_step<3 or time_step%10==0:
    print('Computing time step %d' % time_step)
  velocity, smoke, pressure = step(velocity, smoke, pressure, dt=DT)
  if time_step%5==0:
    steps.append( [smoke.values, velocity.values.vector[0], velocity.values.vector[1]] )

fig, axes = pylab.subplots(1, len(steps), figsize=(16, 5))
for i in range(len(steps)):
    axes[i].imshow(steps[i][0].numpy('y,x'), origin='lower', cmap='magma')
    axes[i].set_title(f"d at t={i*5}")
```

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho} \nabla p + \nu \nabla \cdot \nabla \mathbf{u} + (0,1)^T \xi d \quad \text{s.t.} \quad \nabla \cdot \mathbf{u} = 0,$$
$$\frac{\partial d}{\partial t} + \mathbf{u} \cdot \nabla d = 0$$

$$u_t + \lambda_1 (u u_x + v u_y) = -p_x + \lambda_2 (u_{xx} + u_{yy}),$$
$$v_t + \lambda_1 (u v_x + v v_y) = -p_y + \lambda_2 (v_{xx} + v_{yy}),$$



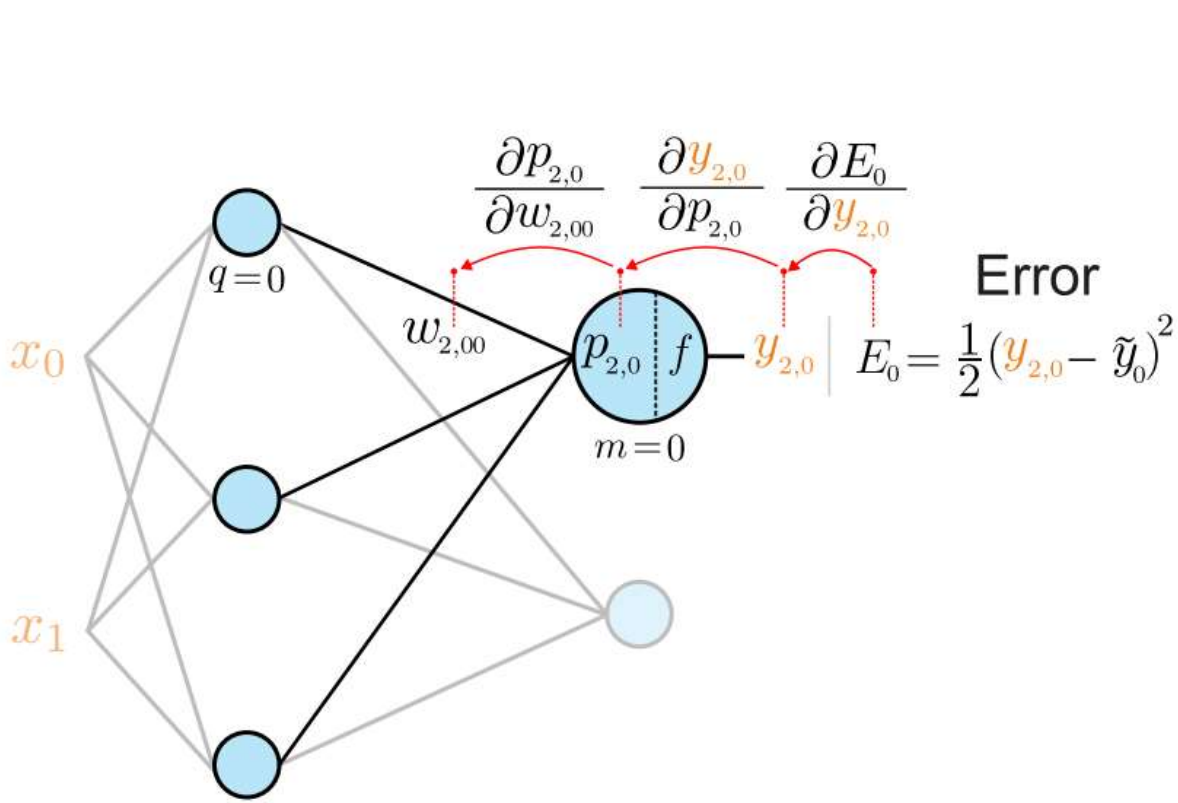
Supervised Training of PINN

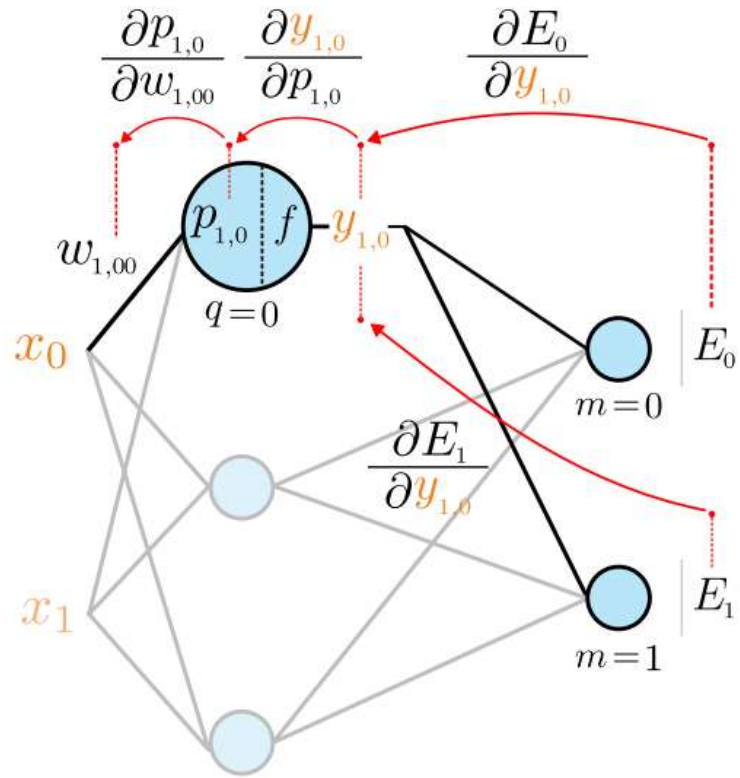


Supervised loss: $e(\mathbf{y}_i, \mathbf{y}_i^*)$

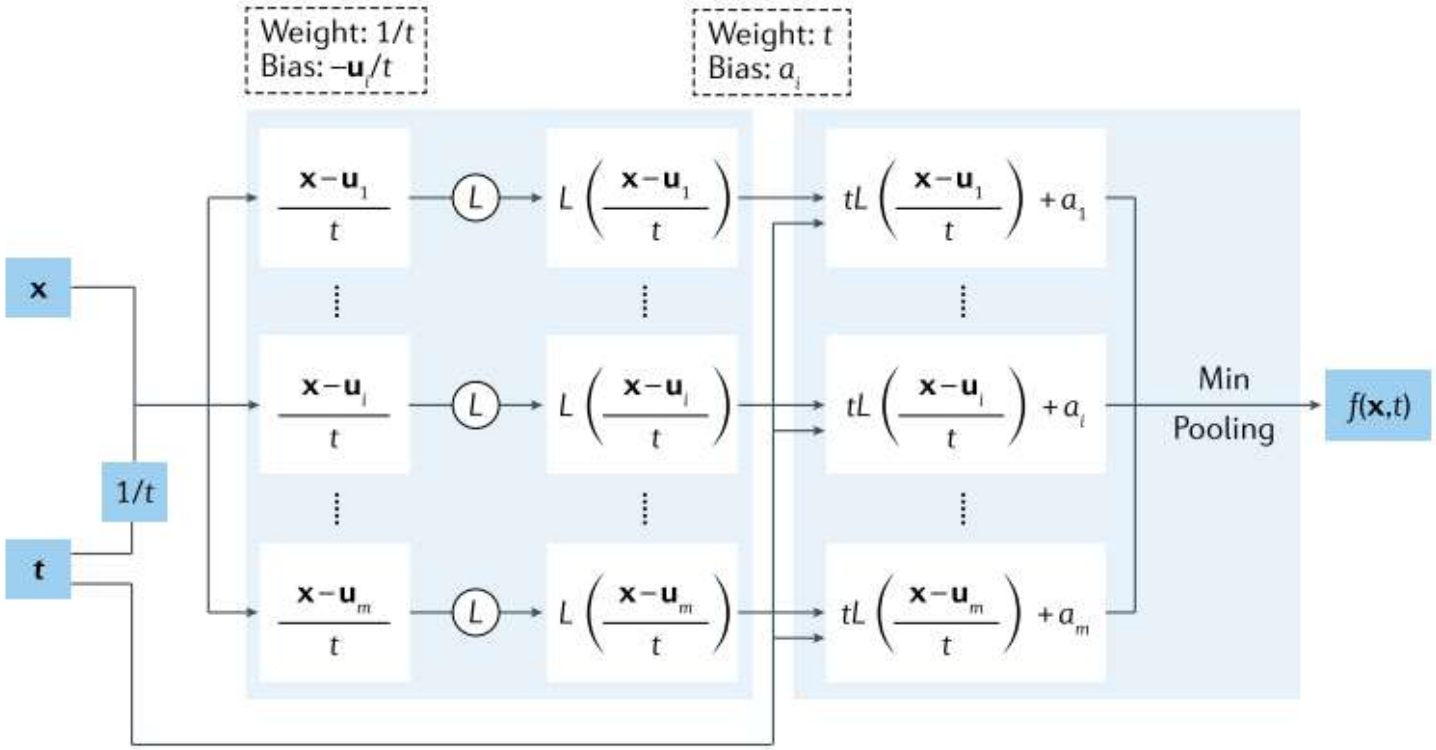
$$\arg\min_\theta \sum_i (f(x_i;\theta) - y_i^*)^2,$$

The training process uses a set of model equations, and approximates them numerically, in order to train the NN representation f.


$\frac{\partial p_{2,0}}{\partial w_{2,00}}$
$\frac{\partial y_{2,0}}{\partial p_{2,0}}$
$\frac{\partial E_0}{\partial y_{2,0}}$
$q=0$
x_0
x_1
$w_{2,00}$
$p_{2,0}$
f
$y_{2,0}$
$m=0$
Error
$E_0 = \frac{1}{2}(y_{2,0} - \tilde{y}_0)^2$


$\frac{\partial p_{1,0}}{\partial w_{1,00}}$
$\frac{\partial y_{1,0}}{\partial p_{1,0}}$
$\frac{\partial E_0}{\partial y_{1,0}}$
$w_{1,00}$
$p_{1,0}$
f
$y_{1,0}$
$q=0$
x_0
x_1
$m=0$
E_0
$\frac{\partial E_1}{\partial y_{1,0}}$
$m=1$
E_1

An example of Inductive bias in PINN



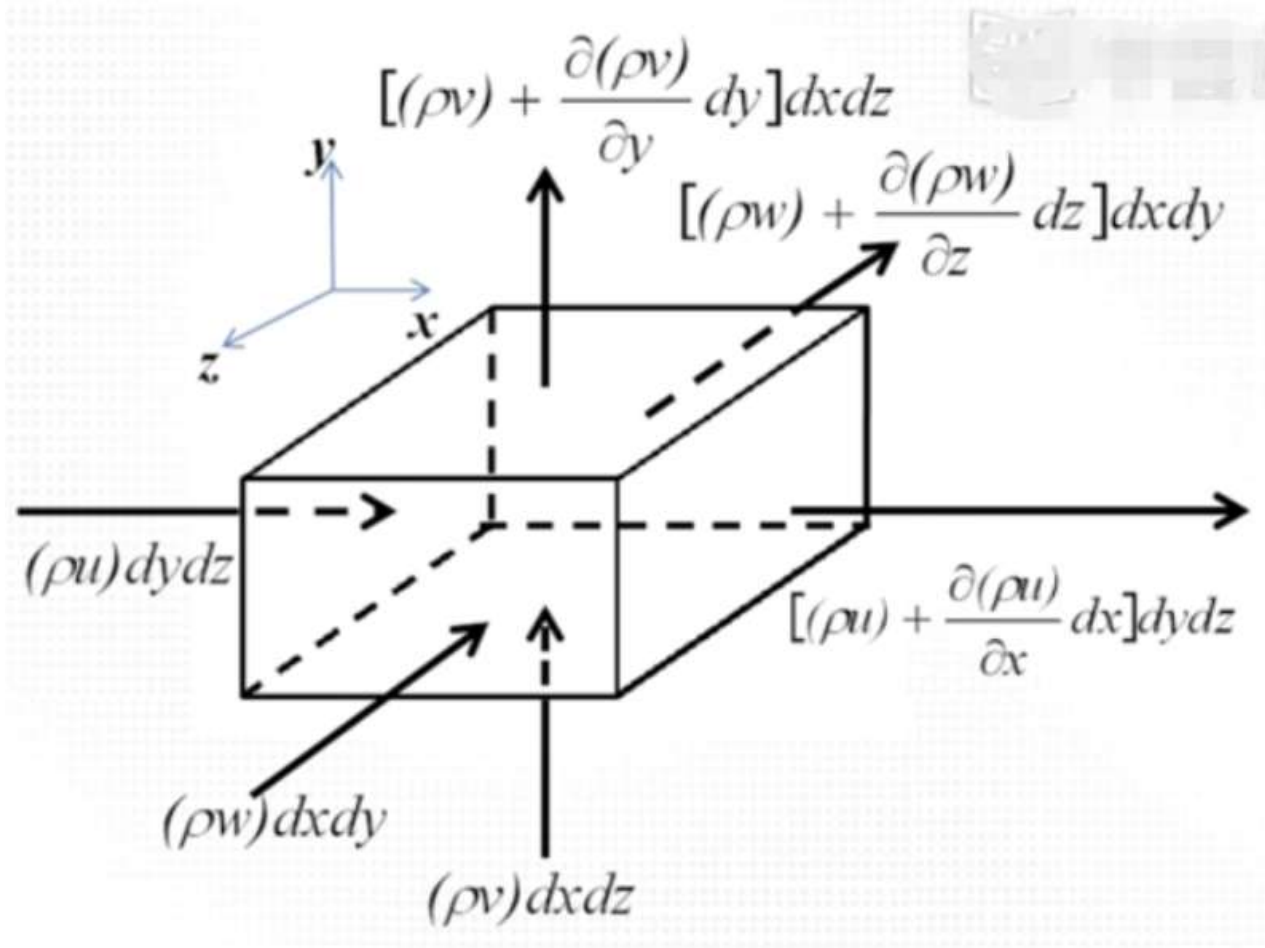
This two-layer architecture defines

$$f(x,t) := \min_{i\in\{1,\dots,m\}} \left\{ tL\left(\frac{x-u_i}{t}\right) + a_i \right\}$$

x, t : spatial and temporal variables
L : convex and Lipschitz activation function
a, u : NN parameters
m : number of neurons

$f(x,t)$ is the viscosity solution to the following Hamilton–Jacobi PDE:

$$\begin{cases} \dfrac{\partial f}{\partial t}(x,t) + H(\nabla_x f(x,t)) = 0 & x\in\mathbb{R}^n, t\in(0,+\infty), \\ f(x,0) = J(x) & x\in\mathbb{R}^n, \end{cases}$$


$[(\rho v) + \frac{\partial(\rho v)}{\partial y} dy]dxdz$
$[(\rho w) + \frac{\partial(\rho w)}{\partial z} dz]dxdy$
y
x
z
$(\rho u)dydz$
$[(\rho u) + \frac{\partial(\rho u)}{\partial x} dx]dydz$
$(\rho w)dxdy$
$(\rho v)dxdz$