# Physics-Informed Neural Networks (PINNs)

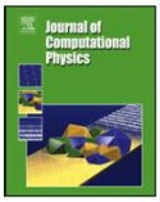
Tuesday, October 8, 2024 3:43 PM

Journal of Computational Physics 378 (2019) 686–707

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp



## Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations

M. Raissi [a], P. Perdikaris [b,*], G.E. Karniadakis [a]

[a] Division of Applied Mathematics, Brown University, Providence, RI, 02912, USA
[b] Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA, 19104, USA

Earlier work

- Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990
- Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. on neural networks*, 9(5):987–1000, 1998.

Some references:

- https://book.sciml.ai/notes/03-Introduction_to_Scientific_Machine_Learning_through_Physics-Informed_Neural_Networks/
- Deep Learning in Computational Mechanics, Kollmannsburger et al (Springer)
- https://www.nvidia.com/en-us/on-demand/deep-learning-for-science-and-engineering/ , co-developed by Karniadakis (PINNs and DeepOnets)
- Brittany Erickson's MATH 607 Seminar on Physics-Informed Deep Learning at U Oregon which did a lot of PINNs

I'll put more references on the github README files

* 1D means $x \in \mathbb{R}$ but $u: \mathbb{R}^2 \to \dots$ (space and time)

Context: PDE

$$u_t + N(u, \lambda) = 0, \quad x \in \Omega, \quad t \in [0, 1]$$

($\lambda$: parameter)

Ex: 1D Burger's Eq

$$u_t + \overbrace{\lambda_1 u u_x - \lambda_2 u_{xx}}^{N(u,\lambda)} = 0$$

See Nick Trefethen's "coffee table PDE" book for good, concise intro

# PINNs (p. 2)

Tuesday, October 8, 2024 4:23 PM

**Ex:** 1D nonlinear Schrödinger Eq $\Omega = [-5,5]$, $t \in [0, \pi/2]$

- $i u_t + \underbrace{\frac{1}{2} u_{xx} + |u|^2 \cdot u}_{i \cdot N(u)} = 0$ $\quad u : \mathbb{R}^2 \to \mathbb{C}$, $u(t,x) \in \mathbb{C}$

initial condition
- $u(t=0, x) = 2 \cdot \operatorname{sech}(x)$ ← this is the particular I.C. in the paper

boundary condition
$$\begin{cases} u(t, x=-5) = u(t, x=5) \\ u_x(t, x=-5) = u_x(t, x=-5) \end{cases}$$
(periodic B.C.)

**Ex:** Allen-Cahn Eq. Reaction-Diffusion (nonlinear)

$$u_t \underbrace{- 0.0001 \cdot u_{xx} + 5u^3 - 5u}_{N(u)} = 0 \qquad \Omega = [-1,1],\ t \in [0,1]$$

$u(t=0, x) = x^2 \cos(\pi x)$ ← from the paper I.C.

$$\left.\begin{aligned} u(t, x=-1) &= u(t, x=1) \\ u_x(t, x=-1) &= u_x(t, x=1) \end{aligned}\right\} \text{periodic B.C.}$$

**Ex:** Korteweg-de Vries (KdV) Eq. Burger's w/ dispersive term

$$u_t + \lambda_1 u u_x + \lambda_2 u_{xxx} = 0$$

**Ex:** (toy problem) ODE not PDE

$$u_t - g = 0 \quad \text{i.e.} \quad \frac{d}{dt} u(t) = g(t)$$

$u(0) = u_0$ I.C.

Solution via Fund. Thm. of Calculus: $u(t) = \int_0^t g(s)\, ds + u_0$

**Aside:** in general, how do you "check your answer"? i.e. have the true solution?

1) use an established, trusted numerical solver
2) use a problem with known analytical solution
   "manufactured solution" or look up in literature

# PINNs (p. 3)

Tuesday, October 8, 2024 4:38 PM

## Main Ideas

① Forward problem $u_t + N(u;\lambda) = 0$ $\quad t\in[0,T]$, $x\in\Omega$ + I.C. + B.C.

assume $\lambda$ parameters are known, so we won't write them explicitly

$f := u_t + N(u) = 0$

Idea: parameterize $u : [0,T]\times\Omega \to \mathbb{R}^{\cdots}$ as a neural net

same as "Implicit Neural Representation" (INR) of SIRENs and others

then fit neural net parameters so that:

1) $f(t_f^i, x_f^i) = 0$ for a "bunch" of collocation points $\{t_f^i, x_f^i\}_{i=1}^{N_f}$

2) $u(t_u^i, x_u^i) = u^i$ for a "bunch" of points at $t=0$ (initial conditions) and at $x\in\partial\Omega$ (boundary conditions)

we don't know $u$ "a priori" here

here we know $u$ since it's either I.C. or B.C.

then solve this in practice by minimizing

$$\text{Loss} = \frac{1}{N_u}\sum_{i=1}^{N_u}\left(u(t_u^i, x_u^i) - u^i\right)^2 + \frac{c}{N_f}\sum_{i=1}^{N_f} f(t_f^i, x_f^i)^2 \qquad c>0$$

Not exactly penalty method but similar.

Turned root-finding into minimization

find $F(x)=0$ $\quad$ $\min_x F(x)^2$

i.e. Loss = 0 iff all constraints satisfied

DETAILS

- How to choose collocation points?
  - in small dimensions, could use a grid
  - randomly ("Monte Carlo"), typically uniformly or more dense near boundary or interesting regions
  - quasi-Monte Carlo (now in SciPy!) mixture of grid vs. M.C. Aims to have high discrepancy
  - space-filling designs, experimental design Stratified, Latin Hypercube, etc.
  - sparse grids

Christoffel fcn sampling ... Area of approximation theory

ex: Gauss-Chebyshev (roots of polynomial) ...

# PINNs (p. 4)

Tuesday, October 8, 2024 4:54 PM

## How to optimize?

eg. 5 layers, 100 neurons each

Since neural net has small dimensional input and output, typically not too many weights ... so quasi-Newton (eg. L-BFGS) often work.

## How to evaluate $f$? And its gradients?

ex: Burger's Eq, $f = u_t + (\lambda_1 u \cdot u_x - \lambda_2 u_{xx})$

$u_t, u_x, u_{xx}$ via ... backpropagation (or any kind of AD)

$\begin{bmatrix} t \\ x \end{bmatrix}$ → layer 1, weights $W_1$ → $\sigma$ → layer 2 ... → output

we typically do AD for weights...

... but no technical reason we can't do AD w.r.t. these.

To train, take AD w.r.t. weights as usual

## (1a) Variant: Discrete-time solver (§3.2 in paper)

motivated by multi-stage methods like Runge-Kutta

Output at several stages, allows for implicit schemes

Very interesting / clever, a little weird, numerics not too convincing to me...

## Larger points:

- They do not advocate solving PDEs this way!
  - In high dim., they need too many collocation pts. (i.e. 3D)
  - In 1D, we already can solve PDE easily
  - PINNs need not conserve energy, mass, ...
- Advantages: :) "easy" / "experimentation" (i.e. Standard ML w/ few data, then add PDE to help)
  2) Easy to incorporate side information

# PINNs (p. 5)

Tuesday, October 8, 2024 5:18 PM

[A note about I.C. + B.C.]

Vanilla PINN: $\text{loss} = \text{loss}_u + c \cdot \text{loss}_f$ ($c>0$; $f(u,t)=0$ is the PDE)

$\text{loss}_u = \text{loss}_{IC} + \text{loss}_{BC}$ (Initial + Boundary Conditions)

Choose $c=1$ or anything $>0$ ... in practice, one loss term may converge faster, and you need to tune $c$ like in penalty method.

Alternative: hard-code IC and/or BC

(re-parameterization)

eg. $u(0,x) = u_0(x)$ I.C., $\Omega = [-1,1]$

$u(t,-1) = g_{left}(t)$

$u(t,+1) = g_{right}(t)$ B.C.

by continuity: $u_0(-1) = g_{left}(0)$, $u_0(1) = g_{right}(0)$

define

$$u(t,x) = \underbrace{t \cdot \cos\left(\tfrac{\pi}{2}x\right)}_{=0 \text{ if } t=0 \text{ or } x=\pm 1} \cdot \underbrace{\tilde{u}(t,x)}_{\text{parameterize by neural net}} + u_0(x)$$

$$+ \frac{1-x}{2}\left[g_{left}(t) - u_0(-1)\right] + \frac{1+x}{2}\left[g_{right}(t) - u_0(1)\right]$$

(or do a similar style trick)

→ more annoying to program, but easier to train

So... $u(t=0,x) = 0 + u_0(x) + \frac{1-x}{2}\underbrace{\left[g_{left}(0) - u_0(-1)\right]}_{=0} + \frac{1+x}{2}\underbrace{\left[g_{right}(0) - u_0(1)\right]}_{=0}$

$= u_0(x)$ ✓ (by consistency / continuity)

and

$u(t,x=1) = 0 + u_0(1) + \frac{1-1}{2}[\dots]$

$+ \frac{1+1}{2}\left[g_{right}(\pi) - u_0(1)\right] = g_{right}(\pi)$ ✓

... So we've defined $u$ so it automatically satisfies the constraints

[analogous to enforcing $u \geq 0$ by setting $u = \tilde{u}^2$]

and no need to fiddle with $c$

⚠ Caveat: I'm making this up, I don't have experience. Chris R's book.sciml.ai suggested the $u_0(x)$ I.C. trick, I added B.C. trick.

# PINNs (p. 6)

Tuesday, October 8, 2024 6:05 PM

## ② Inverse Problems

- Examples are relatively low-dim inverse problems
- Many alternatives (within and without SciML)
- Includes system identification and (W)SINDy approaches

Setup: $u_t + N(u, \lambda) = 0$ but this time we know $u$!

ex: take measurements of a physical phenomenon

Ex: $u(t,x)$ = # COVID infected people at time $t$ and location $x$

We may be interested in the equation (or really parameters in the equation like "$r_0$" in ODE SIR models) for modeling or interventions

Goal: discover $\lambda$

Same idea as before: parameterize $u(t,x)$ as a neural net,

but twist: in addition to training weights $w$, also train parameters $\lambda$

Since we have $u$, we set

$$u(t^i, x^i) = u^i \quad i = 1, \dots, n$$

collocation pts

$u^i$: known! (may be noisy)

or,

$$\min_{\lambda, w} \frac{1}{n} \sum_{i=1}^{n} \left( u(t^i, x^i) - u^i \right)^2$$

## ②a Discrete-time RK version

- skip -

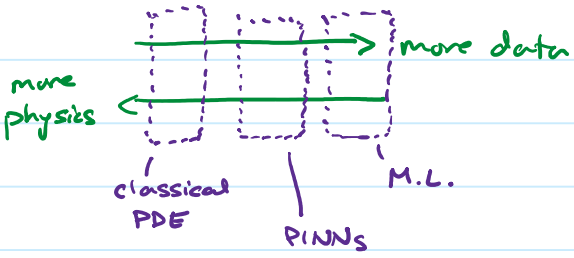
# PINNs (p. 7)

Wednesday, October 9, 2024 10:23 AM

## When are PINNs useful? See Karniadakis et al., Nature Rev. Phys., '21

- If you don't need (accurate) extrapolation
- Complicated scenarios, eg. multiphysics
- Avoid complicated codes with 1000's of lines of code
- For situations where classical methods struggle

  high-dim and/or stochastic PDE — often in finance, Black-Scholes Hamilton-Jacobi-Bellman
- For complicated geometries or to avoid meshing
- If you have missing/gappy/low-res data on boundaries or I.C.
- For some inverse problems where classical methods are slow due to forward simulation




## "Pre-built" Software in Python see Karniadakis for some (as of '21)

eg. "DeepXDE" (w/ TensorFlow) by Karniadakis' group / NVIDIA

NeuroDiffEq is one using PyTorch