Physics Informed Neural Networks

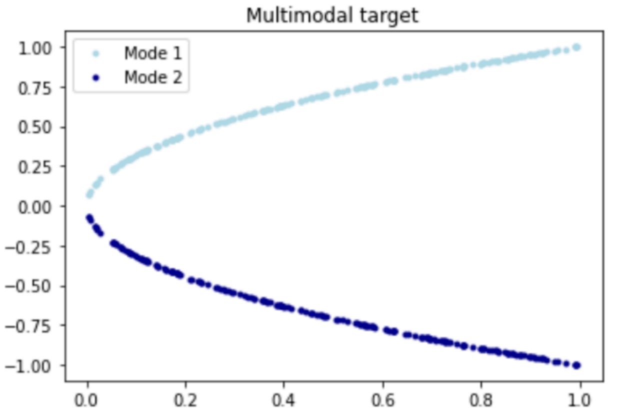
Nasy

Nov 18, 2022

Outline

Finding the inverse function of a parabola

Given a function $\mathcal{P} : y \to y^2$, where $y \in [0, 1]$, find a unknow function f that satisfies $\mathcal{P}(f(x)) = x, \ \forall x \in [0, 1]$.

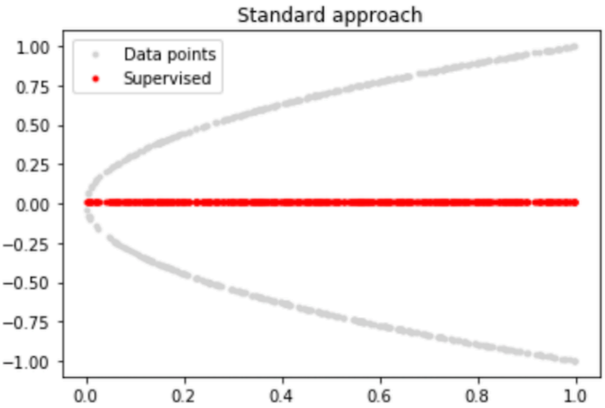


Classical

A classical approach is to use a neural network to approximate the data points Ω:

$$f_\theta(x) \approx y, \ \forall (x, y) \in \Omega$$

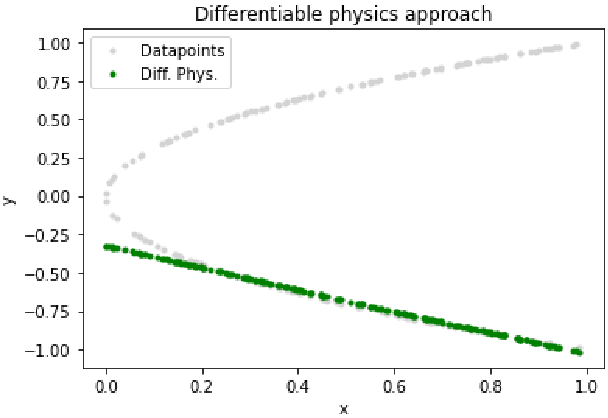
where θ is the parameters of the neural network.
However, nowhere near the solution.



Physics approach

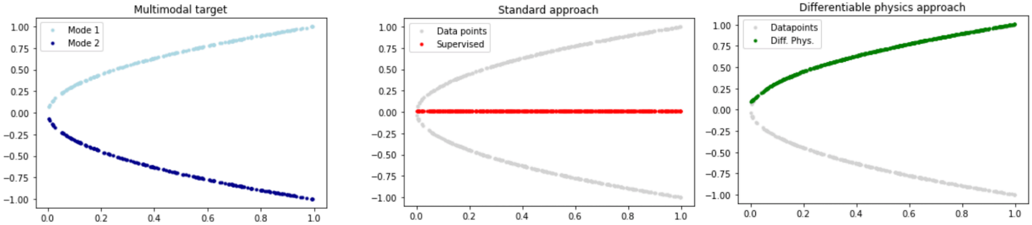
$$\mathcal{P}(f_\theta(x)) \approx y, \ \forall (x, y) \in \Omega$$

Now, minimize the error between $f_\theta^2(x)$ and y.



Physics Informed Neural Networks (PINNs) Definition

Neural networks that are trained to solve supervised learning tasks while respecting any given law of physics described by general nonlinear partial differential equations (PDE).



What is PDE?

- Equation containing unknown functions and its partial derivative.
- Describe the relationship between independent variables, unknown functions and partial derivative.

Example

- $f(x, y) = ax + by + c$, where a, b, c are unknown parameters.
- $u(x, y) = \alpha u(x, y) + \beta f(x, y)$ where u is the unknown function.
- $u_x(x, y) = \alpha u_y(x, y) + \beta f_{xy}(x, y)$ where u_x is the partial derivative of u with respect to x, u_y is the partial derivative of u with respect to y, and f_{xy} is the partial derivative of f with respect to x and y.

Notations

- $\dot{u} = \frac{\partial u}{\partial t}$
- $u_{xy} = \frac{\partial^2 u}{\partial y \partial x}$
- $\nabla u(x, y, z) = u_x + u_y + u_z$
- $\nabla \cdot \nabla u(x, y, z) = \Delta u(x, y, z) = u_{xx} + u_{yy} + u_{zz}$
- ∇ : nabla, or del.

Laplace's equation

$$\Delta \varphi = 0$$

or

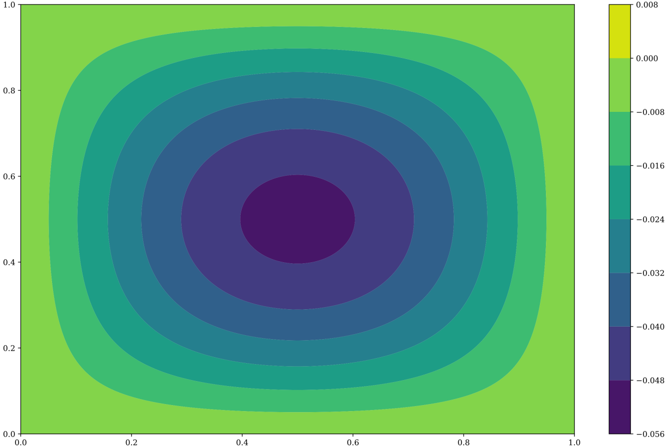
$$\nabla \cdot \nabla \varphi = 0$$

or, in a 3D space:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} = 0$$

Poisson's equation

$$\Delta\varphi = f$$



Heat equation

$$\dot{u} = \frac{\partial u}{\partial t} = \alpha \Delta u$$

where α is the thermal diffusivity.

Wave equation

$$\ddot{u} = c^2 \nabla^2 u$$

where c is the wave speed.

Burgers' equation

$$u_t + uu_x = \nu u_{xx}$$

t	temporal coordinate
x	spatial coordinate
$u(x,t)$	speed of fluid at the indicated spatial and temporal coordinates
ν	viscosity of fluid

Boundary conditions

For a equation $\nabla^2 y + y = 0$ in domain Ω.

- Dirichlet boundary condition: $y(x) = f(x) \quad \forall x \in \partial\Omega$
- Neumann boundary condition: $\frac{\partial y}{\partial \mathbf{n}}(\mathbf{x}) = f(\mathbf{x}) \quad \forall \mathbf{x} \in \partial\Omega$
  - Where f is a known scalar function defined on the boundary domain $\partial\Omega$, $\mathbf{n}$ denotes the (typically exterior) normal to the boundary.
  - The normal derivative, which shows up on the left side, is defined as $\frac{\partial y}{\partial \mathbf{n}}(\mathbf{x}) = \nabla y(\mathbf{x}) \cdot \widehat{\mathbf{n}}(\mathbf{x})$, where $\widehat{\mathbf{n}}$ is the unit normal.
- Robin boundary condition
  - Combine Dirichlet and Neumann boundary conditions.
- Periodic boundary condition

Paper

- Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations[1]
- Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations[2]

[1] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations*. Nov. 28, 2017. URL: http://arxiv.org/abs/1711.10561.

[2] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations*. Nov. 28, 2017. URL: http://arxiv.org/abs/1711.10566.

Problem

- Data-driven solution and data-driven discovery
- Continuous time and discrete time models

Data-driven solution with continuous time

General PDE Form:

$$u_t + \mathcal{N}[u] = 0, \;\; x \in \Omega, \;\; t \in [0, T]$$

where:

$\mathcal{N}[u]$ nonlinear differential operator

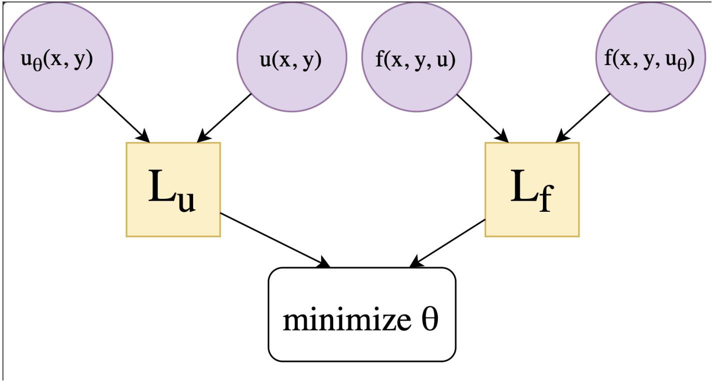
$u(t, x)$ unknown function (solution).

Ω spatial domain.

t time.

Physics informed neural network

- A neural network $u_\theta \approx u$, where θ is the parameters of the neural network.
- A physics informed neural network $f_\theta = u_{\theta t} + \mathcal{N}[u_\theta]$.
- Target: $f_\theta \approx u_t + \mathcal{N}[u]$ and $u_\theta \approx u$.
    - $\mathcal{L} = \mathcal{L}_f + \mathcal{L}_u$



Example (Burgers' Equation)

The equation:

$$u_t + uu_x = \nu u_{xx}$$

Here, already know $\nu = 0.01/\pi$, $x \in [-1, 1], t \in [0, 1]$,
Thus,

$$u_t + uu_x - 0.01/\pi u_{xx} = 0$$

And the equation along with Dirichlet boundary conditions can be written as:

- $u(0, x) = -\sin(\pi x)$
- $u(t, -1) = u(t, 1) = 0$

Target

- Data:
  - Boundary only data from boundary conditions.
- Input: $\{t, x\}$
- Output: $u(t, x)$
- Target: $f_\theta \approx u_t + \mathcal{N}[u]$ and $u_\theta \approx u$.
  - $\mathcal{L} = \mathcal{L}_f + \mathcal{L}_u$

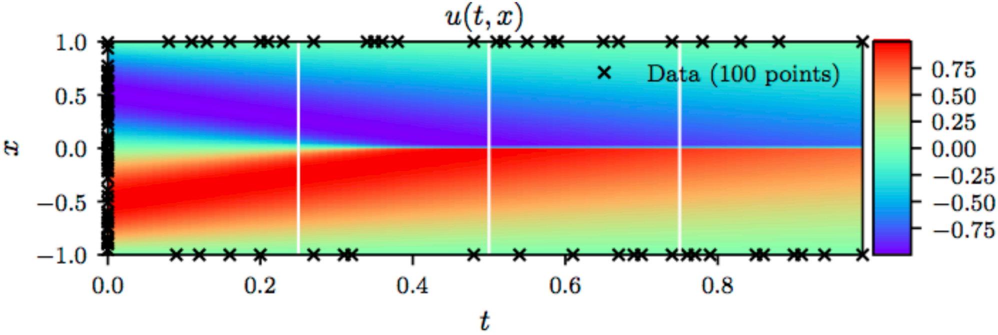
Example (Burgers' Equation) with codes

```
def u_theta(theta, t, x):
    # u_theta.apply(theta, t, x) to approx u(x, t)
    return net(theta, t, x)

def f_theta(theta, t, x):
    # See the auto diff cookbook
    # https://jax.readthedocs.io/en/latest/notebooks/autodiff_cookbook.html
    u = u_theta.apply
    u_t = jax.jacrev(u, argnums=1)(theta, t, x)
    u_x = jax.jacrev(u, argnums=2)(theta, t, x)
    u_xx = jax.hessian(u, argnums=2)(theta, t, x)
    # or jax.jacfwd(jax.jacrev(u, argnums=2), argnums=2)
    f = lambda: u_t + u * u_x - 0.01 * u_xx
    return f
```

Train and Results

- Train with MLPs with L-BFGS solver (quasi-newton method).
- Cannot use ReLU but tanh, because when we do the second order derivative, the ReLU will be 0.



Data-driven discovery with continuous time

General PDE Form:

$$u_t + \mathcal{N}[u; \lambda] = 0, \;\; x \in \Omega, \;\; t \in [0, T]$$

where:

- $\mathcal{N}[u; \lambda]$ nonlinear differential operator with parameters λ.
- $u(t, x)$ unknown function (solution).
- Ω spatial domain.
- t time.

Example (Incompressible Navier-Stokes Equation (convection–diffusion equations)) I

The equations:

$$u_t + \lambda_1(uu_x + vu_y) = -p_x + \lambda_2(u_{xx} + u_{yy}),$$
$$v_t + \lambda_1(uv_x + vv_y) = -p_y + \lambda_2(v_{xx} + v_{yy})$$

,
where:

$u(t, x, y)$ x-component of the velocity field,

$v(t, x, y)$ y-component of the velocity field,

$p(t, x, y)$ pressure,

λ the unknown parameters.

Additional physical constraints:

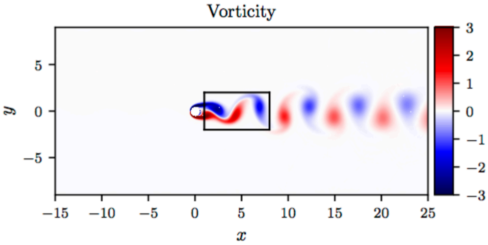
- Solutions to the Navier-Stokes equations are searched in the set of divergence-free functions, i.e.:

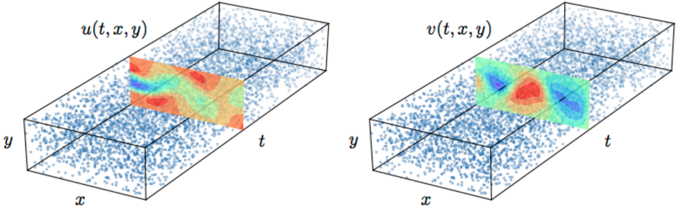
Example (Incompressible Navier-Stokes Equation (convection–diffusion equations)) II

- $u_x + u_y = 0$
- which describes the conservation of mass of the fluid

- u and v can written as a latent function $\psi(t, x, y)$ with an assumption:
  - $u = \psi_y, v = -\psi_x$

NS Equation figure

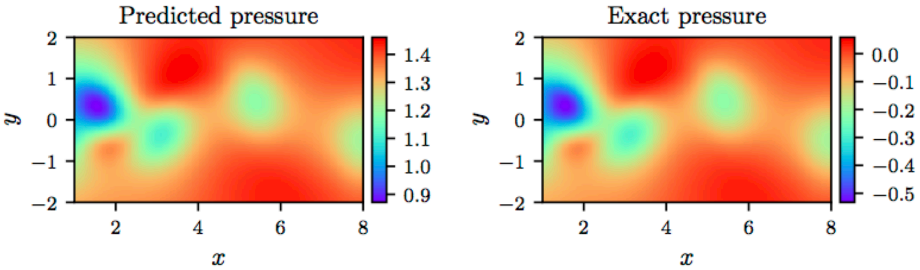




Example (Navier-Stokes Equation) – Target

- The neural network equations:
  - $f := u_t + \lambda_1(uu_x + vu_y) + p_x - \lambda_2(u_{xx} + u_{yy})$,
  - $g := v_t + \lambda_1(uv_x + vv_y) + p_y - \lambda_2(v_{xx} + v_{yy})$
- Inptu: $\{t, x, y, u, v\}$ with noisy.
- Output: $(\psi(t, x, y), p(t, x, y))$.
- Target:
  - $f_\theta \approx f$
  - $g_\theta \approx g$
  - $u_\theta \approx u$
  - $v_\theta \approx v$

Results



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)	$u_t + 0.998(uu_x + vu_y) = -p_x + 0.01057(u_{xx} + u_{yy})$ $v_t + 0.998(uv_x + vv_y) = -p_y + 0.01057(v_{xx} + v_{yy})$

Introduction

JAX is Autograd and XLA, brought together for high-performance numerical computing and machine learning research. It provides composable transformations of Python+NumPy programs: differentiate, vectorize, parallelize, Just-In-Time compile to GPU/TPU, and more.

Pure functional

- $f(x) = y$, always.
- non-pure function:
    - IO operator: `print`
    - No seed random function
    - `time`
    - Runtime error.

Ecosystem

- JAX (jax, jaxlib)
  - `jax`
  - `jax.numpy`
- Haiku (dm-haiku) from deepmind
  - Modules
- Optax (optax) from deepmind
  - Light
  - Linear system optimizers ($Ax = b$)
- JAXopt (jaxopt)
  - Other optimizers.
- Jraph (jraph)
  - Standardized data structures for graphs.
- JAX, M.D. (jax-md)
  - JAX and Molecular Dynamics
- RLax (rlax), and Coax (coax)
  - Reinforcement Learning

Example (def)

```
import jax
import jax.numpy as jnp
import haiku as hk

def _u(t, x):
    return hk.MLP(jnp.concatenate([t, x], axis=-1), [10, 10, 1])

u = hk.transform_with_state(_u)
```

Example (init)

```
fake_t = jnp.ones([batch, size])
fake_x = jnp.ones([batch, size])

# theta: params
# state: training state
# rng:   random number generator
params, state = u.init(rng, fake_t, fake_x)

hk.experimental.tabulate(u)(fake_t, fake_x)
```

Example (loss)

```
def loss_fn(config, ...):

    def _loss(params, t, x):
        u_theta = u.apply(params, t, x)
        ...
        loss = _f
        return loss

    return _loss

loss = loss_fn(config, ...)
```

Example (optim)

```
import optax

lr = optax.linear_schedule(
    0.001,          # init
    0.001 / 10,     # final
    1,              # steps change to final
    150             # start linear decay after steps
)

opt = optax.adam(learning_rate=lr)
opt = optax.adamax(learning_rate=lr)
```

Example (solver)

```
import jaxopt

# Linear solver
solver = jaxopt.OptaxSolver(
    loss,
    opt,
    maxiter=epochs,
    ...
)

# non-linear solver
solver = jaxopt.LBFGS(
    loss,
    maxiter=epochs,
    ...
)

opt_state = solver.init(params, state)
```

Example (train)

```python
# init
params, state, opt_state, update


for batch in data:
    params, state = update(params, state, batch)
```

Example (parallel)

```python
# Use pjit
from jax.experimental.maps import Mesh, ResourceEnv, thread_resources
from jax.experimental.pjit import PartitionSpec, pjit

mesh = Mesh(np.asarray(jax.devices(), dtype=object), ["data", ...])
thread_resources.env = ResourceEnv(physical_mesh=mesh, loops=())

update = pjit(
    solver.update,
    in_axis_resources=[
        None,  # params
        None,  # state
        PartitionSpec("data"),  # batch
    ],
    out_axis_resources=None,
)
```

Conclusion

- Find an inverse function of a parabola
  - Classical
  - Physics informed
- PDE
  - PDE example
  - PDE boundary
- PINNs
  - Data-driven solution with continuous time
    - Burgers' equation
  - Data-driven discovery with continuous time
    - Navier-Stokes equation

Refs I

[1] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations.* Nov. 28, 2017. URL: http://arxiv.org/abs/1711.10561.

[2] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations.* Nov. 28, 2017. URL: http://arxiv.org/abs/1711.10566.