Partial differential equations

Marcin Kuta

Types of PDEs

$$au_{xx} + bu_{xy} + cu_{yy} + du_x + eu_y + fu + g = 0 \tag{1}$$

- $b^2 - ac > 0$: hyperbolic
- $b^2 - ac = 0$: parabolic
- $b^2 - ac < 0$: elliptic

- Wave equation $u_{tt} = u_{xx}$
- Heat equation $u_t = u_{xx}$
- Laplace equation $u_{xx} + u_{yy} = 0$

Solving PDEs with neural networks

Advantages:

- fully implicit method
- mesh-free
- no time step
- no stability problems

Disadvantages:

- does not generalize beyond domain (no extrapolation)
- no guarantee of unique solutions
- may converge to different solutions from different network initial values

Boundary conditions

- Dirichlet (essential boundary conditions)
- Neumann (natural boundary conditions)
- Robin
- mixed
- general

Partial differential equations

$$u_t + \mathcal{N}_x[u] = 0, \qquad x \in \Omega, t \in [0, T] \tag{2}$$

$$u(x, 0) = h(x), \qquad x \in \Omega \tag{3}$$

$$u(x, t) = g(x, t), \qquad x \in \partial\Omega, t \in [0, T] \tag{4}$$

$$\hat{u}(x, t) = Z_l \circ a \circ Z_{l-1} \circ a \ldots a \circ Z_2 \circ a \circ Z_1(x, t) \tag{5}$$

Loss function

$$\mathcal{L} = \lambda_{\text{PDE}}\mathcal{L}_{\text{PDE}} + \lambda_{\text{BC}}\mathcal{L}_{\text{BC}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{REC}}\mathcal{L}_{\text{REC}} + \lambda_{\text{REG}}\mathcal{L}_{\text{REG}} \quad (6)$$

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_r}\sum_{i=1}^{N_r} \left|\hat{u}_t(x_i, t_i) + \mathcal{N}_x\left[\hat{u}(x_i, t_i)\right]\right|^2 \quad (7)$$

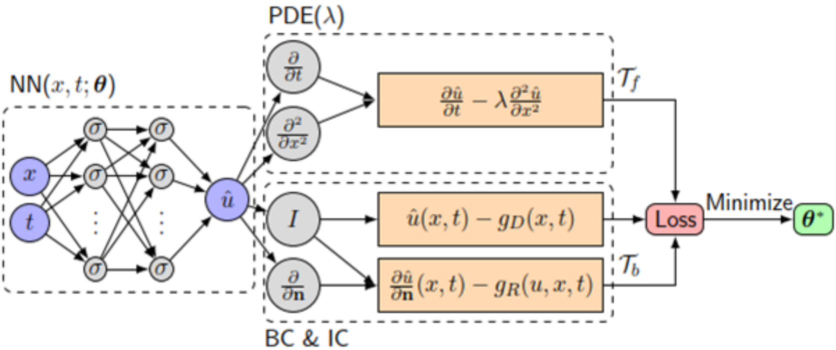
$$\mathcal{L}_{\text{BC}} = \frac{1}{N_b}\sum_{i=1}^{N_b} \left|\hat{u}(x_i, t_i) - g(x_i, t_i)\right|^2 \quad (8)$$

$$\mathcal{L}_{\text{IC}} = \frac{1}{N_0}\sum_{i=1}^{N_0} \left|\hat{u}(x_i, 0) - h(x_i, 0)\right|^2 \quad (9)$$

$$\mathcal{L}_{\text{REC}} = \frac{1}{N_d}\sum_{i=1}^{N_d} \left|\hat{u}(x_i, t_i) - u_i^{\text{true}}\right|^2 \quad (10)$$

$$\mathcal{L}_{\text{REG}} = \frac{1}{2}\sum_{i,j} \left|w_{ij}\right|^2 \quad (11)$$

Physics-informed neural network



Source: [2]

Loss function

$$\mathcal{L} = w_f \mathcal{L}_f + w_b \mathcal{L}_b \tag{12}$$

$$\mathcal{L}_f = \frac{1}{N_f} \sum_{x \in \Omega} ||f(x, \frac{\partial \hat{u}}{\partial x_1}, ..., \frac{\partial \hat{u}}{\partial x_d}, \frac{\partial^2 \hat{u}}{\partial x_1^2}, ...)||_2^2 \tag{13}$$

$$\mathcal{L}_b = \frac{1}{N_b} \sum_{x \in \partial\Omega} ||\mathcal{B}(\hat{u}, x)||_2^2 \tag{14}$$

Heat equation

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}, \;\; x \in [0,1], \;\; t \in [0,1] \tag{15}$$

$\alpha = 0.3$
Boundary conditions:

$$u(0,t) = u(1,t) = 0 \tag{16}$$

Initial conditions:

$$u(x,0) = \sin(\pi x) \tag{17}$$

Exact solution:

$$u(x,t) = \sin(\pi x) e^{-\pi^2 \alpha t} \tag{18}$$

$$NN(x,t) = \hat{u}(x,t) \approx u(x,t) \tag{19}$$

Definition of the problem

- domain
- PDE
- condition equations
- training data
- the architecture of the NN
- optimizer and initializer

Domain

```
import numpy as np
import deepxde as dde
```

```
geom = dde.geometry.Interval(0,1)
timedomain = dde.geometry.TimeDomain(0,1)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```

```
def pde(x, y):
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return dy_t - 0.3*dy_xx
```

Conditions and training data

```
bc = dde.icbc.DirichletBC(geomtime, lambda x: 0, \
                          lambda _, on_boundary: on_boundary)

ic = dde.icbc.IC(geomtime,
    lambda x:, np.sin(np.pi * x[:, 0:1]),
    lambda _, on_initial: on_initial,
)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc, ic],
    num_domain = 4000,
    num_boundary = 2000,
    num_initial = 1000,
    num_test = 1000,
)
```

Network

```
layer_size = [2] + [32]*3 + [1]
activation = "tanh"
initializer = "Glorot normal"
net = dde.nn.FNN(layer_size, activation, initializer)
```

Model

```
net = dde.nn.FNN(layer_size, activation, initializer)
model = dde.Model(data, net)
optimizer = "adam"
model.compile("adam", lr=0.001)

losshistory, train_state = model.train(iterations=10000)
```

Results

```
x_data = np.linspace(0,1,num=100)
t_data = np.linspace(0,1,num=100)
test_x, test_t = np.meshgrid(x_data, t_data)
X = np.vstack((np.ravel(test_x), \
                         np.ravel(test_y))).T
y_pred = model.predict(X)
residual = model.predict(X, operator=pde)
```

References

[1] Maziar Raissi, Paris Perdikaris, George Em Karniadakis
Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations

[2] Lu Lu, Xuhui Meng, Zhiping Mao, George Em Karniadakis
DeepXDE: A Deep Learning Library for Solving Differential Equations

[3] http://github.com/lululxvi/deepxde