Scientific Machine Learning

MAM5 - INUM, Polytech Nice Sophia

DeepXDE

Stéphane Descombes and Stéphane Lanteri
Stephane.Lanteri@inria.fr

Atlantis project-team
Inria Research Center at Université Côte d'Azur, France

Year 2024-2025

Outline

Outline

Outline

Example 9: Burgers equation

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=\nu\frac{\partial^2 u}{\partial x^2} \text{ with } u(-1,t)=u(1,t)=0 \text{ and } u(x,0)=-\sin(\pi x) \text{ for } x\in[-1,1]$$

1. The DeepXDE and NumPy modules are imported

```
import deepxde as dde
from deepxde.backend import tf
```

2. Define a computational domain by using the built-in classes Interval and TimeDomain

```
geom = dde.geometry.Interval(-1.0, 1.0)
timedomain = dde.geometry.TimeDomain(0.0, 0.99)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```

Example 9: Burgers equation

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=\nu\frac{\partial^2 u}{\partial x^2} \text{ with } u(-1,t)=u(1,t)=0 \text{ and } u(x,0)=-\sin(\pi x) \text{ for } x\in[-1,1]$$

3. Define the PDE residual

- The first argument to pde is 2-dimensional vector
- The first component(x[:,0]) is the x coordinate
- The second componenet (x[:,1]) is the t coordinate
- The second argument is the network output

```
def pde(x, y):
    dy_x = dde.grad.jacobian(y, x, i=0, j=0)
    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 + y*dy_x - 0.01/np.pi*dy_xx
```

Example 9: Burgers equation

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=\nu\frac{\partial^2 u}{\partial x^2} \text{ with } u(-1,t)=u(1,t)=0 \text{ and } u(x,0)=-\sin(\pi x) \text{ for } x\in[-1,1]$$

4. Define the boundary/initial condition

- on_boundary is chosen here to use the whole boundary of the computational domain
- Include the geomtime space-time geometry and on_boundary as the BCs in the DirichletBC
- Use the computational domain, initial function, and on_initial to specify the IC

```
bc = dde.icbc.DirichletBC(geomtime, lambda x: 0.0, lambda _,
                                    on_boundary: on_boundary)
ic = dde.icbc.IC(geomtime, lambda x: -np.sin(np.pi*x[:,0:1]),
                           lambda _, on_initial: on_initial)
```

Example 9: Burgers equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \text{ with } u(-1,t) = u(1,t) = 0 \text{ and } u(x,0) = -\sin(\pi x) \text{ for } x \in [-1,1]$$

5. Define the PDE problem

```
data = dde.data.TimePDE(geomtime, pde, [bc, ic],
                        num_domain=2540, num_boundary=80,
                        num_initial=160)
```

- 2540 is the number of training residual points sampled inside the domain
- 80 is the number of training points sampled on the boundary
- 160 is the number of residual points for the initial condition

6. Define the network

```
layer_size = [2] + [20]*3 + [1]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.nn.FNN(layer_size, activation, initializer)
```

Example 9: Burgers equation

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=\nu\frac{\partial^2 u}{\partial x^2} \text{ with } u(-1,t)=u(1,t)=0 \text{ and } u(x,0)=-\sin(\pi x) \text{ for } x\in[-1,1]$$

7. Build a model, choose the optimizer and learning rate

```
model = dde.Model(data, net)
model.compile("adam", lr=1.0e-3)
```

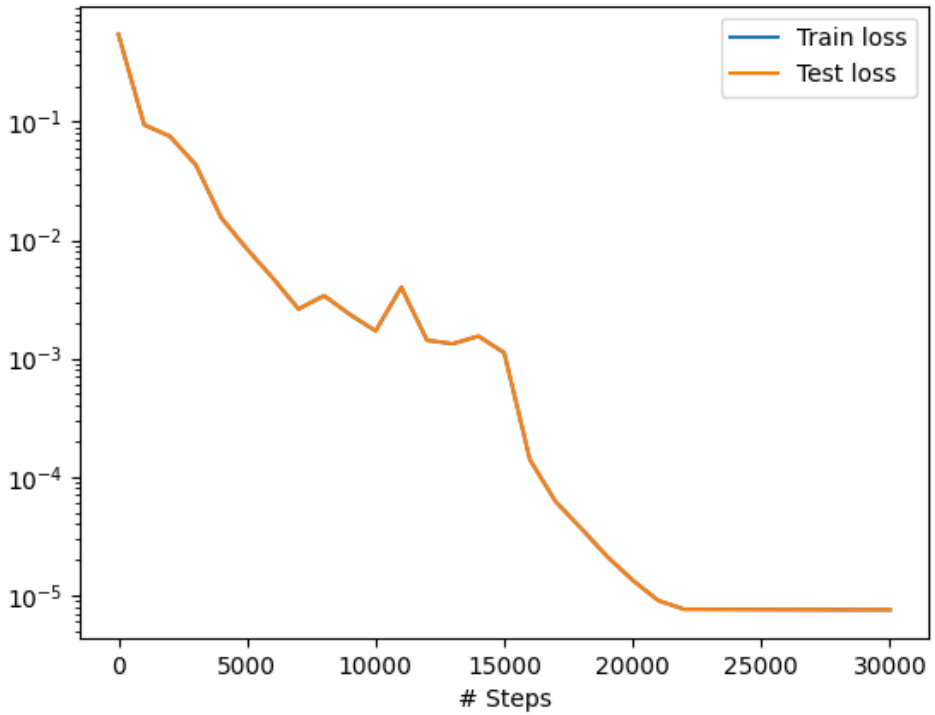
8. Train the model

```
losshistory, train_state = model.train(iterations=15000)
```

```
Using backend: pytorch

Compiling model...
'compile' took 0.000054 s

Training model...

Step      Train loss                        Test loss                         Test metric
15000     [1.94e-03, 3.18e-05, 9.73e-04]    [1.94e-03, 3.18e-05, 9.73e-04]    []
16000     [4.08e-04, 3.60e-06, 1.63e-04]    [4.08e-04, 3.60e-06, 1.63e-04]    []
17000     [1.37e-04, 5.28e-07, 7.64e-05]    [1.37e-04, 5.28e-07, 7.64e-05]    []
18000     [8.27e-05, 3.01e-07, 2.77e-05]    [8.27e-05, 3.01e-07, 2.77e-05]    []
19000     [5.41e-05, 3.03e-07, 1.06e-05]    [5.41e-05, 3.03e-07, 1.06e-05]    []
20000     [3.52e-05, 1.66e-07, 5.43e-06]    [3.52e-05, 1.66e-07, 5.43e-06]    []
21000     [2.56e-05, 1.16e-07, 2.36e-06]    [2.56e-05, 1.16e-07, 2.36e-06]    []
22000     [1.88e-05, 6.77e-08, 1.66e-06]    [1.88e-05, 6.77e-08, 1.66e-06]    []
23000     [1.42e-05, 3.12e-08, 9.50e-07]    [1.42e-05, 3.12e-08, 9.50e-07]    []
24000     [1.10e-05, 3.32e-08, 7.74e-07]    [1.10e-05, 3.32e-08, 7.74e-07]    []
25000     [8.77e-06, 1.93e-08, 7.78e-07]    [8.77e-06, 1.93e-08, 7.78e-07]    []
26000     [8.55e-06, 1.59e-08, 6.94e-07]    [8.55e-06, 1.59e-08, 6.94e-07]    []
27000     [8.54e-06, 1.65e-08, 6.93e-07]    [8.54e-06, 1.65e-08, 6.93e-07]    []
28000     [8.53e-06, 1.68e-08, 6.93e-07]    [8.53e-06, 1.68e-08, 6.93e-07]    []
29000     [8.52e-06, 1.71e-08, 6.93e-07]    [8.52e-06, 1.71e-08, 6.93e-07]    []
30000     [8.52e-06, 1.73e-08, 6.93e-07]    [8.52e-06, 1.73e-08, 6.93e-07]    []

Best model at step 30000:
  train loss: 9.23e-06
  test loss: 9.23e-06
  test metric: []

'train' took 55.118216 s
```

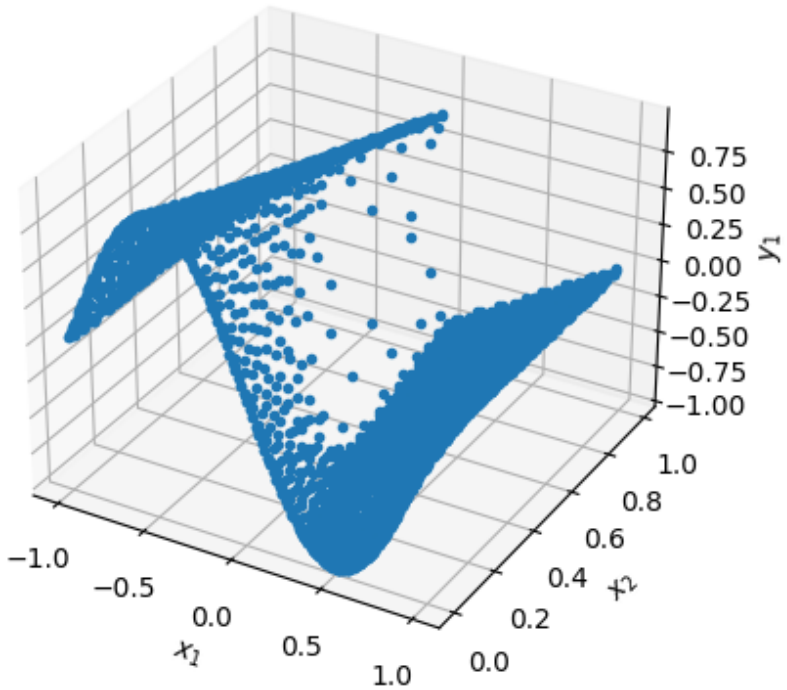
Example 9: Burgers equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \quad \text{with} \quad u(-1,t) = u(1,t) = 0 \quad \text{and} \quad u(x,0) = -\sin(\pi x) \quad \text{for} \quad x \in [-1,1]$$



Example 9: Burgers equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2} \quad \text{with} \quad u(-1,t) = u(1,t) = 0 \quad \text{and} \quad u(x,0) = -\sin(\pi x) \quad \text{for} \quad x \in [-1,1]$$



Example 10: Laplace equation in a polar coordinate system on a disk

$$r\frac{dy}{dr} + r^2\frac{d^2y}{dr^2} + \frac{d^2y}{d\theta^2} = 0$$

with $y(1,\theta) = \cos(\theta)$ (Dirichlet BC) and $y(r,\theta+2\pi) = y(r,\theta)$ (periodic BC)

for $r \in [0,1]$ and $\theta \in [0,2\pi]$

Reference solution: $y = r\cos(\theta)$

1. The DeepXDE and NumPy modules are imported

```
import deepxde as dde
from deepxde.backend import tf
```

2. Define a computational domain by using the built-in class Rectangle

```
geom = dde.geometry.Rectangle(xmin=[0.0, 0.0], xmax=[1.0, 2.0*np.pi])
```

Example 10: Laplace equation in a polar coordinate system on a disk

$$r\frac{dy}{dr} + r^2\frac{d^2y}{dr^2} + \frac{d^2y}{d\theta^2} = 0$$

with $y(1,\theta) = \cos(\theta)$ (Dirichlet BC) and $y(r,\theta+2\pi) = y(r,\theta)$ (periodic BC)

for $r \in [0,1]$ and $\theta \in [0,2\pi]$

Reference solution: $y = r\cos(\theta)$

3. Define the PDE residual

- The first argument to pde is 2-dimensional vector
- The first component(x[:,0]) is the r coordinate
- The second componenet (x[:,1]) is the θ coordinate
- The second argument is the network output $y(r,\theta)$

```
def pde(x, y):
    dy_r = dde.grad.jacobian(y, x, i=0, j=0)
    dy_rr = dde.grad.hessian(y, x, i=0, j=0)
    dy_thetatheta = dde.grad.hessian(y, x, i=1, j=1)
    return x[:,0:1]*dy_r + x[:,0:1]**2*dy_rr + dy_thetatheta
```

Example 10: Laplace equation in a polar coordinate system on a disk

$$r\frac{dy}{dr} + r^2\frac{d^2y}{dr^2} + \frac{d^2y}{d\theta^2} = 0$$

with $y(1,\theta) = \cos(\theta)$ (Dirichlet BC) and $y(r,\theta+2\pi) = y(r,\theta)$ (periodic BC)

for $r \in [0,1]$ and $\theta \in [0,2\pi]$

Reference solution: $y = r\cos(\theta)$

4. Define the Dirichlet boundary condition

```
bc_rad = dde.icbc.DirichletBC(geom,
         lambda x: np.cos(x[:,1:2]),
         lambda x,
         on_boundary: on_boundary and dde.utils.isclose(x[0], 1.0))
```

Example 10: Laplace equation in a polar coordinate system on a disk

$$r\frac{dy}{dr} + r^2\frac{d^2y}{dr^2} + \frac{d^2y}{d\theta^2} = 0$$

with $y(1,\theta) = \cos(\theta)$ (Dirichlet BC) and $y(r,\theta+2\pi) = y(r,\theta)$ (periodic BC)

for $r \in [0,1]$ and $\theta \in [0,2\pi]$

Reference solution: $y = r\cos(\theta)$

5. Define the PDE problem

```
data = dde.data.PDE(geom, pde, bc_rad,
                    num_domain=2540, num_boundary=80,
                    solution=solution)
```

- 2540 is the number of training residual points sampled inside the domain
- 80 is the number of training points sampled on the boundary
- The argument solution is the reference solution to compute the error

```
def solution(x):
    r, theta = x[:,0:1], x[:,1:]
    return r*np.cos(theta)
```

Example 10: Laplace equation in a polar coordinate system on a disk

$$r\frac{dy}{dr} + r^2\frac{d^2y}{dr^2} + \frac{d^2y}{d\theta^2} = 0$$

with $y(1,\theta) = \cos(\theta)$ (Dirichlet BC) and $y(r,\theta+2\pi) = y(r,\theta)$ (periodic BC)

for $r \in [0,1]$ and $\theta \in [0,2\pi]$

Reference solution: $y = r\cos(\theta)$

6. Define the network

```
layer_size = [2] + [20]*3 + [1]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.nn.FNN(layer_size, activation, initializer)
```

Example 10: Laplace equation in a polar coordinate system on a disk

$$r\frac{dy}{dr} + r^2\frac{d^2y}{dr^2} + \frac{d^2y}{d\theta^2} = 0$$

with $y(1,\theta) = \cos(\theta)$ (Dirichlet BC) and $y(r,\theta+2\pi) = y(r,\theta)$ (periodic BC)

for $r \in [0,1]$ and $\theta \in [0,2\pi]$

Reference solution: $y = r\cos(\theta)$

7. Transform the network inputs

- In cartesian coordinates, the variables are in the form of $[r\sin(\theta), r\cos(\theta)]$
- Use them as features to satisfy the underlying physical constraints, so that the network is automatically periodic along the θ coordinate and the period is 2π

```
def feature_transform(x):
    return tf.concat(
    [x[:,0:1]*tf.sin(x[:,1:2]), x[:,0:1]*tf.cos(x[:,1:2])], axis=1)
```

```
net.apply_feature_transform(feature_transform)
```

Example 10: Laplace equation in a polar coordinate system on a disk

$$r\frac{dy}{dr} + r^2\frac{d^2y}{dr^2} + \frac{d^2y}{d\theta^2} = 0$$

with $y(1,\theta) = \cos(\theta)$ (Dirichlet BC) and $y(r,\theta+2\pi) = y(r,\theta)$ (periodic BC)

for $r \in [0,1]$ and $\theta \in [0,2\pi]$

Reference solution: $y = r\cos(\theta)$

8. Build a model, choose the optimizer and learning rate

```
model = dde.Model(data, net)
model.compile("adam", lr=1.0e-3)
```

9. Train the model

```
losshistory, train_state = model.train(iterations=15000)
```

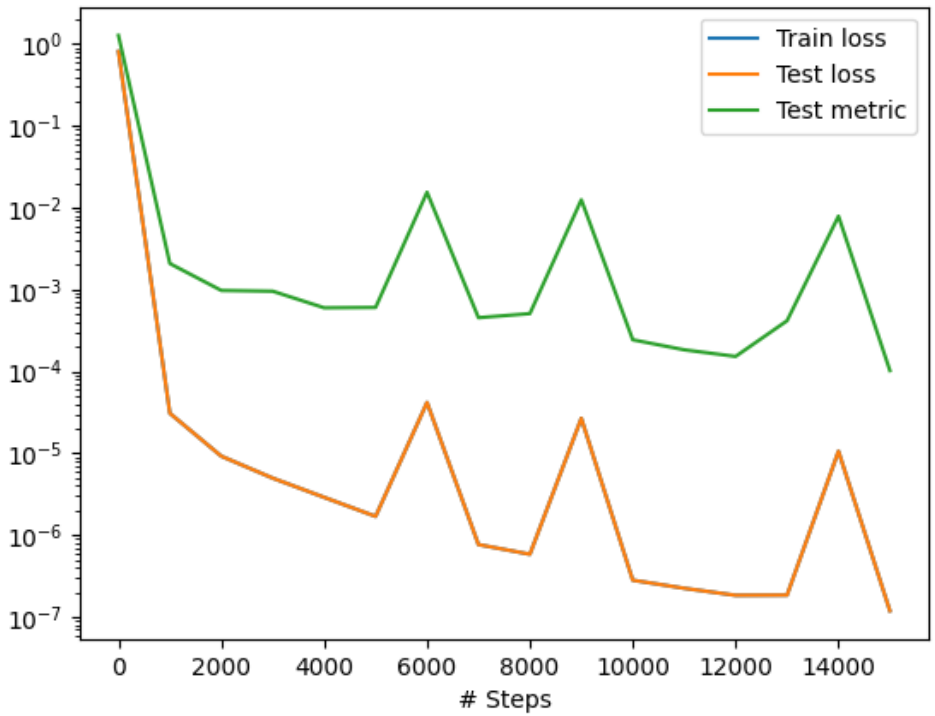
```
Using backend: pytorch

Compiling model...
'compile' took 0.000126 s

Training model...

Step      Train loss              Test loss               Test metric
0         [5.03e-02, 1.58e-01]    [5.03e-02, 1.58e-01]    [5.45e-01]
1000      [3.48e-05, 5.77e-06]    [3.48e-05, 5.77e-06]    [3.26e-03]
2000      [8.66e-06, 1.11e-06]    [8.66e-06, 1.11e-06]    [1.37e-03]
3000      [3.63e-06, 2.86e-07]    [3.63e-06, 2.86e-07]    [8.49e-04]
4000      [1.77e-06, 1.23e-07]    [1.77e-06, 1.23e-07]    [7.02e-04]
5000      [1.07e-06, 2.21e-07]    [1.07e-06, 2.21e-07]    [7.90e-04]
6000      [7.74e-07, 1.01e-06]    [7.74e-07, 1.01e-06]    [2.24e-03]
7000      [6.00e-07, 1.28e-06]    [6.00e-07, 1.28e-06]    [2.56e-03]
8000      [5.00e-07, 6.82e-08]    [5.00e-07, 6.82e-08]    [3.62e-04]
9000      [4.92e-07, 3.70e-06]    [4.92e-07, 3.70e-06]    [4.05e-03]
10000     [3.89e-07, 5.47e-08]    [3.89e-07, 5.47e-08]    [3.58e-04]
11000     [1.12e-06, 1.46e-05]    [1.12e-06, 1.46e-05]    [5.86e-03]
12000     [3.32e-07, 3.84e-08]    [3.32e-07, 3.84e-08]    [2.85e-04]
13000     [3.15e-07, 2.64e-08]    [3.15e-07, 2.64e-08]    [2.78e-04]
14000     [8.54e-07, 1.15e-05]    [8.54e-07, 1.15e-05]    [5.01e-03]
15000     [2.93e-07, 1.10e-07]    [2.93e-07, 1.10e-07]    [4.37e-04]

Best model at step 13000:
  train loss: 3.41e-07
  test loss: 3.41e-07
  test metric: [2.78e-04]

'train' took 85.780040 s
```

Example 10: Laplace equation in a polar coordinate system on a disk

$$r\frac{dy}{dr} + r^2\frac{d^2y}{dr^2} + \frac{d^2y}{d\theta^2} = 0$$

with $y(1,\theta) = \cos(\theta)$ (Dirichlet BC) and $y(r,\theta+2\pi) = y(r,\theta)$ (periodic BC)

for $r \in [0,1]$ and $\theta \in [0,2\pi]$

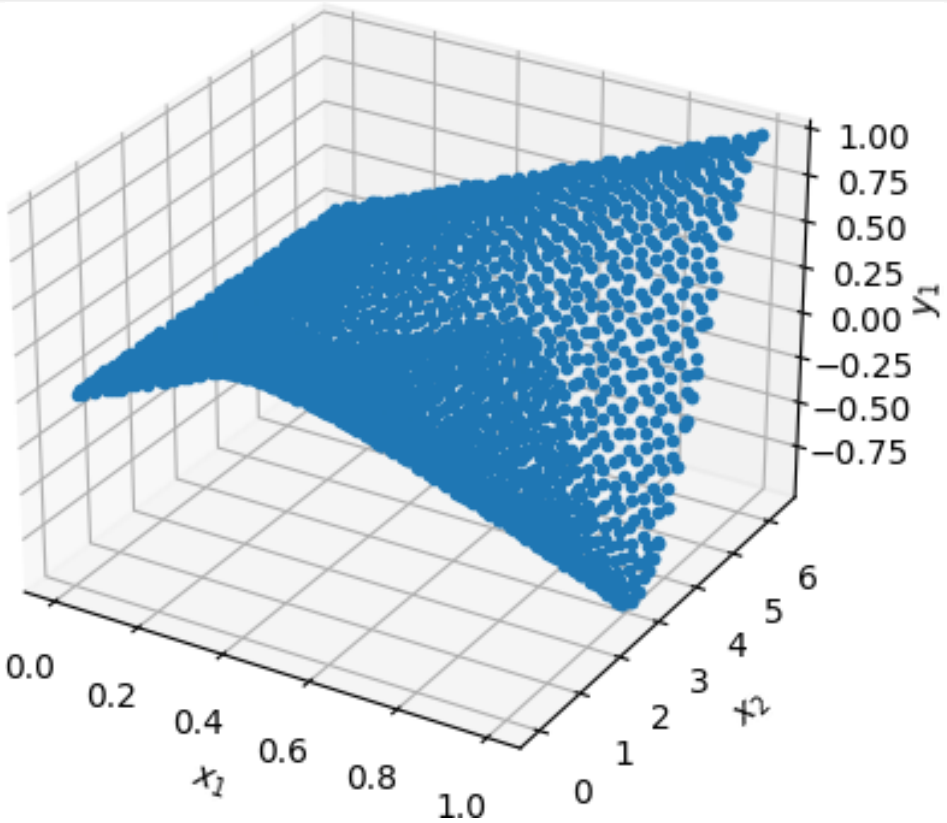


Example 10: Laplace equation in a polar coordinate system on a disk

$$r\frac{dy}{dr} + r^2\frac{d^2y}{dr^2} + \frac{d^2y}{d\theta^2} = 0$$

with $y(1,\theta) = \cos(\theta)$ (Dirichlet BC) and $y(r,\theta+2\pi) = y(r,\theta)$ (periodic BC)

for $r \in [0,1]$ and $\theta \in [0,2\pi]$



Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$
$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.

1. The DeepXDE and NumPy modules are imported

```
import deepxde as dde
from deepxde.backend import tf
import numpy as np
```

2. Define our three unknown variables for σ, ρ and β

```
C1 = dde.Variable(1.0)
C2 = dde.Variable(1.0)
C3 = dde.Variable(1.0)
```

Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$
$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.

3. Define a computational domain using built-in class TimeDomain

```
geom = dde.geometry.TimeDomain(0.0, 3.0)
```

Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$
$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.

4. Create the Lorenz system to solve

```
def Lorenz_system(x, y):
    y1, y2, y3 = y[:,0:1], y[:,1:2], y[:,2:]
    dy1_x = dde.grad.jacobian(y, x, i=0)
    dy2_x = dde.grad.jacobian(y, x, i=1)
    dy3_x = dde.grad.jacobian(y, x, i=2)
    return [
        dy1_x - C1*(y2 - y1),
        dy2_x - y1*(C2 - y3) + y2,
        dy3_x - y1*y2 + C3*y3]
```

Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$
$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.

5. Define the initial conditions

```
def boundary(_, on_initial):
    return on_initial
```

- The initial conditions are specified using the computational domain, initial function and boundary
- The argument component refers to if this IC is for the first component (x), the second component (y), or the third component (z).

```
ic1 = dde.icbc.IC(geom, lambda X: -8.0, boundary, component=0)
ic2 = dde.icbc.IC(geom, lambda X: 7.0 , boundary, component=1)
ic3 = dde.icbc.IC(geom, lambda X: 27.0, boundary, component=2)
```

Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$
$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.

6. Define the training data (1/2)

- Assign the data from Lorenz.npz to the corresponding t, x, y and z values for training
- First we retrieve the data to train the model
- The data is split into "t" and "y", which correspond to time datapoints and the cartesian coodrinate datapoints (x, y, and z), respectivly

```
def gen_traindata():
    data = np.load("dataset/Lorenz.npz")
    return data["t"], data["y"]
```

Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$
$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.

6. Define the training data (2/2)

- Then we organize and assign the train data

```
observe_t, ob_y = gen_traindata()
observe_y0 = dde.icbc.PointSetBC(observe_t, ob_y[:,0:1], component=0)
observe_y1 = dde.icbc.PointSetBC(observe_t, ob_y[:,1:2], component=1)
observe_y2 = dde.icbc.PointSetBC(observe_t, ob_y[:,2:3], component=2)
```

Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$
$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.

7. Define the PDE problem

- num_domain is the number of points inside the domain
- num_boundary is the number of points on the boundary
- anchors are extra points beyond num_domain and num_boundary used for training

```
data = dde.data.PDE(
    geom,
    Lorenz_system,
    [ic1, ic2, ic3, observe_y0, observe_y1, observe_y2],
    num_domain=400,
    num_boundary=2,
    anchors=observe_t)
```

Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$
$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.

8. Choose the network

```
net = dde.nn.FNN([1] + [40] * 3 + [3], "tanh", "Glorot uniform")
```

9. Build a model

```
model = dde.Model(data, net)
model.compile("adam", lr=0.001, external_trainable_variables=[C1, C2, C3])
variable = dde.callbacks.VariableValue(
            [C1, C2, C3], period=600, filename="variables.dat")
```

10. Train the model

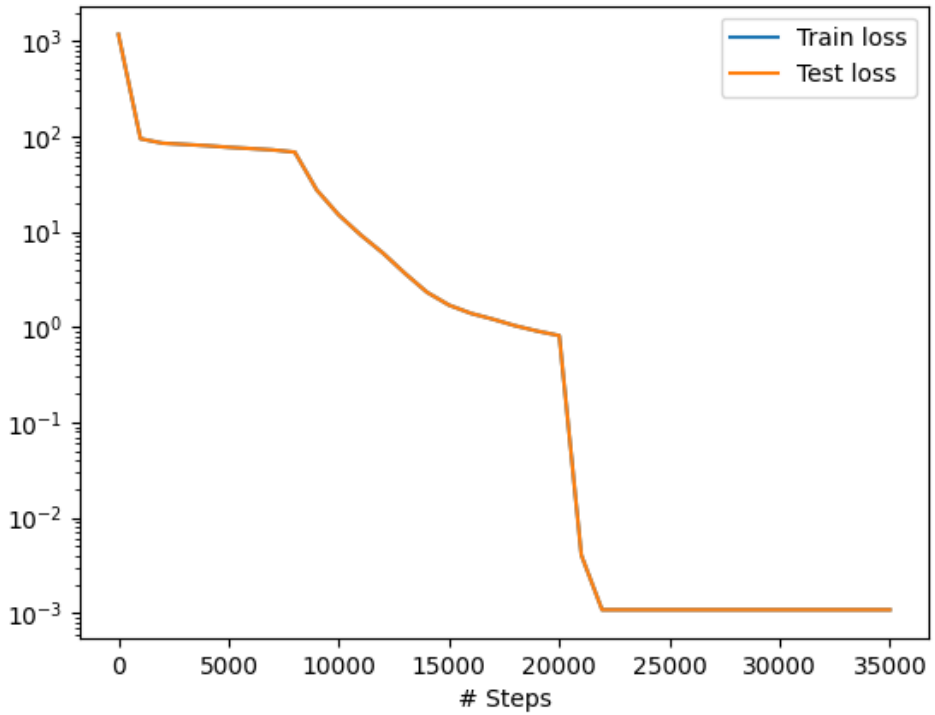
```
losshistory, train_state = model.train(iterations=60000, callbacks=[variable])
```

Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$

$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.

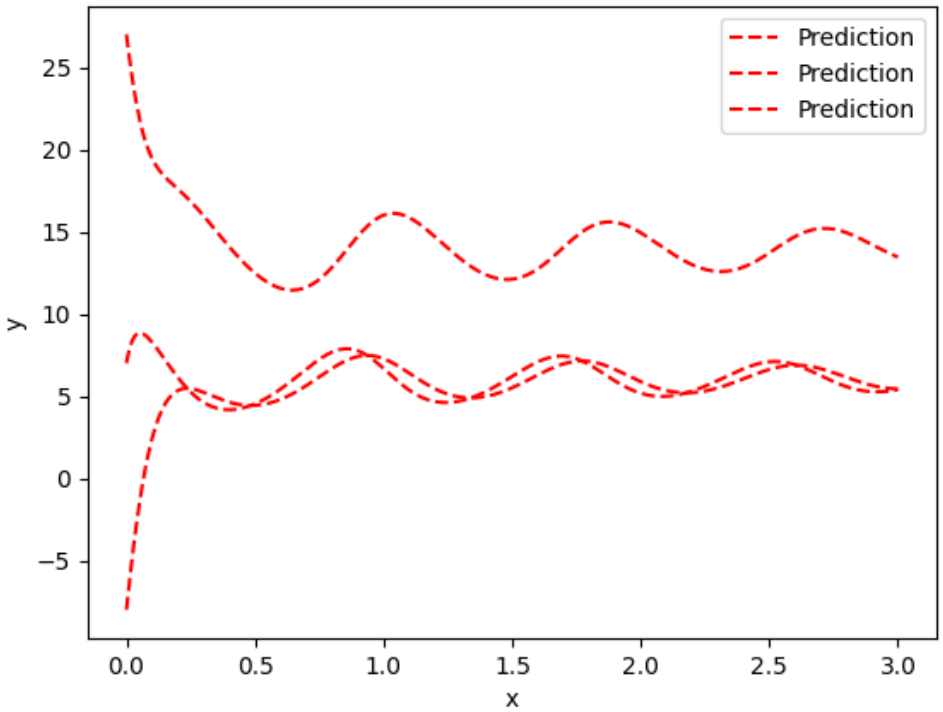


Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$

$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.

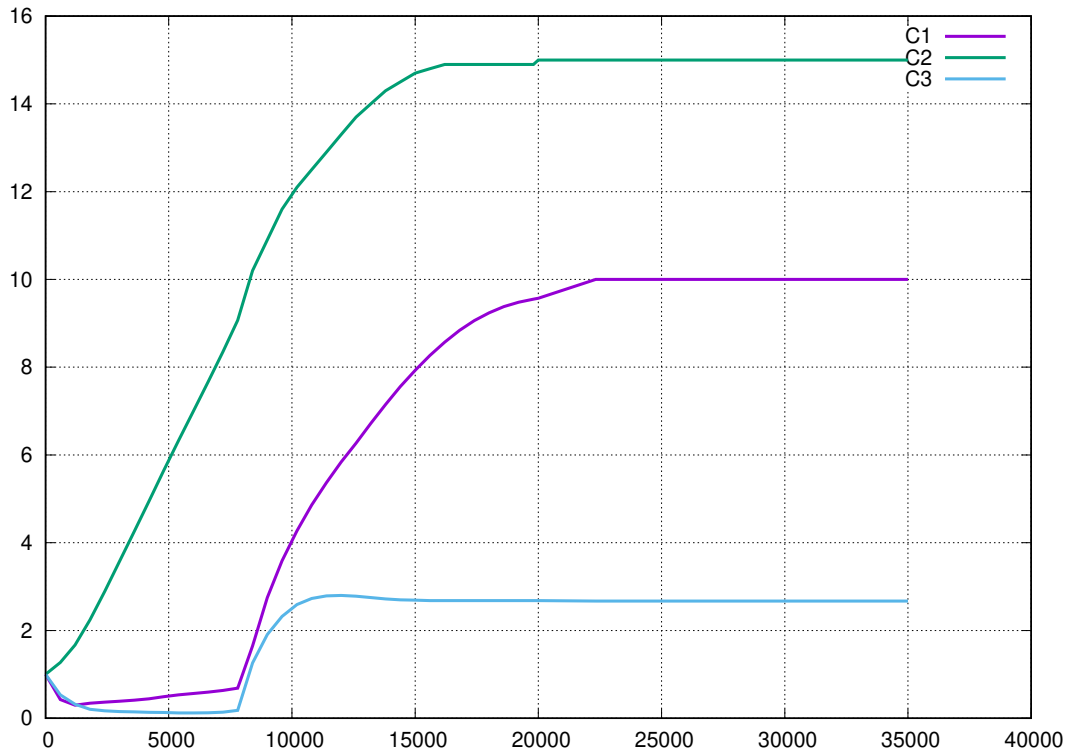


Example 11: Lorentz system

$$\frac{dx}{dt} = \sigma(y - x) \ , \quad \frac{dy}{dt} = x(\rho - z) - y \ , \quad \frac{dz}{dt} = xy - \beta z \quad \text{for} \quad t \in [0, 3]$$

$$\text{and with} \quad x(0) = -8, \quad y(0) = 7, \quad z(0) = 27$$

where the parameters σ, ρ and β are to be identified from observations of the system at certain times and whose true values are 10, 15, and 8/3, respectivly.



Example 12: Poisson equation with unknown forcing field

$$\frac{d^2u}{dx^2} = q(x) \quad \text{for} \quad x \in [-1, 1] \quad \text{and with} \quad u(-1) = u(1) = 0$$

where both $u(x)$ and $q(x)$ are unknown, and we have the measurement of $u(x)$ at 100 points. The reference solution is $u(x) = \sin(\pi x)$ and $q(x) = -\pi^2 \sin(\pi x)$.

1. The DeepXDE and NumPy modules are imported

```
import deepxde as dde
from deepxde.backend import tf
import numpy as np
```

2. Define the training data

- Define a function to generate num equally spaced points from -1 to 1

```
def gen_traindata(num):
    xvals = np.linspace(-1.0, 1.0, num).reshape(num, 1)
    uvals = np.sin(np.pi*xvals)
    return xvals, uvals
```

Example 12: Poisson equation with unknown forcing field

$$\frac{d^2u}{dx^2} = q(x) \quad \text{for} \quad x \in [-1, 1] \quad \text{and with} \quad u(-1) = u(1) = 0$$

where both $u(x)$ and $q(x)$ are unknown, and we have the measurement of $u(x)$ at 100 points. The reference solution is $u(x) = \sin(\pi x)$ and $q(x) = -\pi^2 \sin(\pi x)$.

3. Define a computational domain using built-in class Interval

```
geom = dde.geometry.Interval(-1.0, 1.0)
```

4. Define the PDE residual

```
def pde(x, y):
    u, q = y[:,0:1], y[:,1:2]
    du_xx = dde.grad.hessian(y, x, component=0, i=0, j=0)
    return -du_xx + q
```

Example 12: Poisson equation with unknown forcing field

$$\frac{d^2u}{dx^2} = q(x) \quad \text{for} \quad x \in [-1, 1] \quad \text{and with} \quad u(-1) = u(1) = 0$$

where both $u(x)$ and $q(x)$ are unknown, and we have the measurement of $u(x)$ at 100 points. The reference solution is $u(x) = \sin(\pi x)$ and $q(x) = -\pi^2 \sin(\pi x)$.

5. Define boundary conditions

```
def sol(x):
    return np.sin(np.pi*x)
```

- The function sol returns True if a point is on a boundary and False otherwise
- The component axis on which the boundary is satisfied is also selected

```
bc = dde.icbc.DirichletBC(geom, sol, lambda _,
                          on_boundary: on_boundary, component=0)
```

Example 12: Poisson equation with unknown forcing field

$$\frac{d^2u}{dx^2} = q(x) \quad \text{for} \quad x \in [-1, 1] \quad \text{and with} \quad u(-1) = u(1) = 0$$

where both $u(x)$ and $q(x)$ are unknown, and we have the measurement of $u(x)$ at 100 points. The reference solution is $u(x) = \sin(\pi x)$ and $q(x) = -\pi^2 \sin(\pi x)$.

6. Generate observation data

```
ob_x, ob_u = gen_traindata(100)
observe_u = dde.icbc.PointSetBC(ob_x, ob_u, component=0)
```

7. Define the PDE

```
data = dde.data.PDE(geom, pde,
        [bc, observe_u],
        num_domain=200,
        num_boundary=2,
        anchors=ob_x,
        num_test=1000)
```

- anchors are extra points beyond num_domain and num_boundary used for training

Example 12: Poisson equation with unknown forcing field

$$\frac{d^2u}{dx^2} = q(x) \quad \text{for} \quad x \in [-1, 1] \quad \text{and with} \quad u(-1) = u(1) = 0$$

where both $u(x)$ and $q(x)$ are unknown, and we have the measurement of $u(x)$ at 100 points. The reference solution is $u(x) = \sin(\pi x)$ and $q(x) = -\pi^2 \sin(\pi x)$.

8. Choose the networks

- Two networks: one to train for $u(x)$ and the other to train for $q(x)$
- Fully connected networks

```
net = dde.nn.PFNN([1, [20, 20], [20, 20], [20, 20], 2],
                  "tanh", "Glorot uniform")
```

9. Build a model

```
model = dde.Model(data, net)
model.compile("adam", lr=0.0001, loss_weights=[1, 100, 1000])
```

10. Train the model

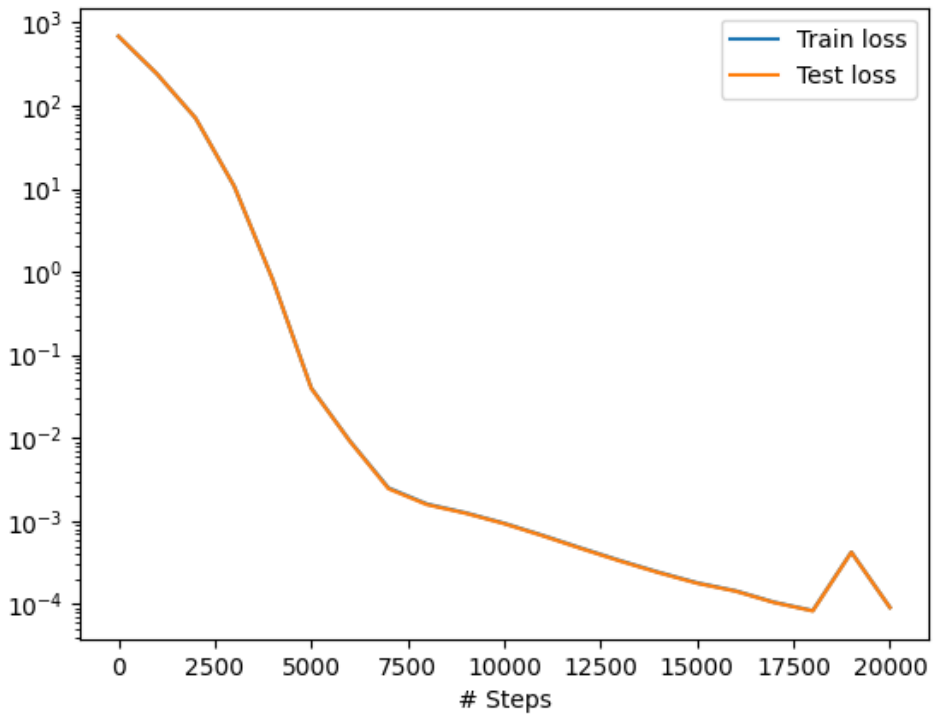
```
losshistory, train_state = model.train(iterations=20000)
```

```
Compiling model...
'compile' took 0.000129 s

Training model...

Step      Train loss                        Test loss
0         [9.84e-03, 5.93e+00, 6.76e+02]    [9.69e-03, 5.93e+00, 6.76e+02]
2000      [4.93e-01, 1.95e+01, 5.06e+01]    [4.96e-01, 1.95e+01, 5.06e+01]
4000      [9.21e-03, 2.19e-01, 5.75e-01]    [8.59e-03, 2.19e-01, 5.75e-01]
6000      [8.15e-04, 4.38e-04, 7.95e-03]    [7.34e-04, 4.38e-04, 7.95e-03]
8000      [1.28e-04, 6.10e-05, 1.41e-03]    [1.02e-04, 6.10e-05, 1.41e-03]
10000     [4.94e-05, 3.41e-05, 8.58e-04]    [3.85e-05, 3.41e-05, 8.58e-04]
12000     [2.51e-05, 1.16e-05, 4.36e-04]    [1.88e-05, 1.16e-05, 4.36e-04]
14000     [1.19e-05, 3.91e-06, 2.27e-04]    [8.74e-06, 3.91e-06, 2.27e-04]
16000     [5.95e-06, 2.02e-07, 1.39e-04]    [4.51e-06, 2.02e-07, 1.39e-04]
18000     [3.89e-06, 2.60e-07, 7.96e-05]    [3.16e-06, 2.60e-07, 7.96e-05]
20000     [5.24e-06, 8.44e-06, 7.77e-05]    [5.09e-06, 8.44e-06, 7.77e-05]

Best model at step 18000:
  train loss: 8.37e-05
  test loss: 8.30e-05
  test metric: []

'train' took 30.475348 s

l2 relative error for u: 0.0003947657
l2 relative error for q: 0.010026635
```

Example 12: Poisson equation with unknown forcing field

$$\frac{d^2u}{dx^2} = q(x) \quad \text{for} \quad x \in [-1, 1] \quad \text{and with} \quad u(-1) = u(1) = 0$$

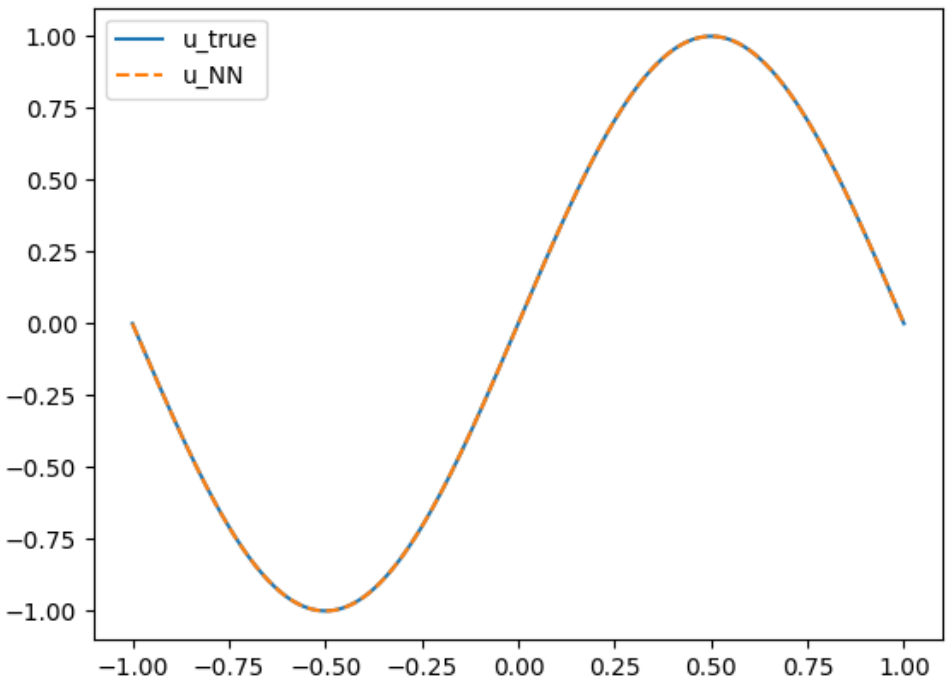
where both $u(x)$ and $q(x)$ are unknown, and we have the measurement of $u(x)$ at 100 points. The reference solution is $u(x) = \sin(\pi x)$ and $q(x) = -\pi^2 \sin(\pi x)$.



Example 12: Poisson equation with unknown forcing field

$$\frac{d^2u}{dx^2} = q(x) \quad \text{for} \quad x \in [-1, 1] \quad \text{and with} \quad u(-1) = u(1) = 0$$

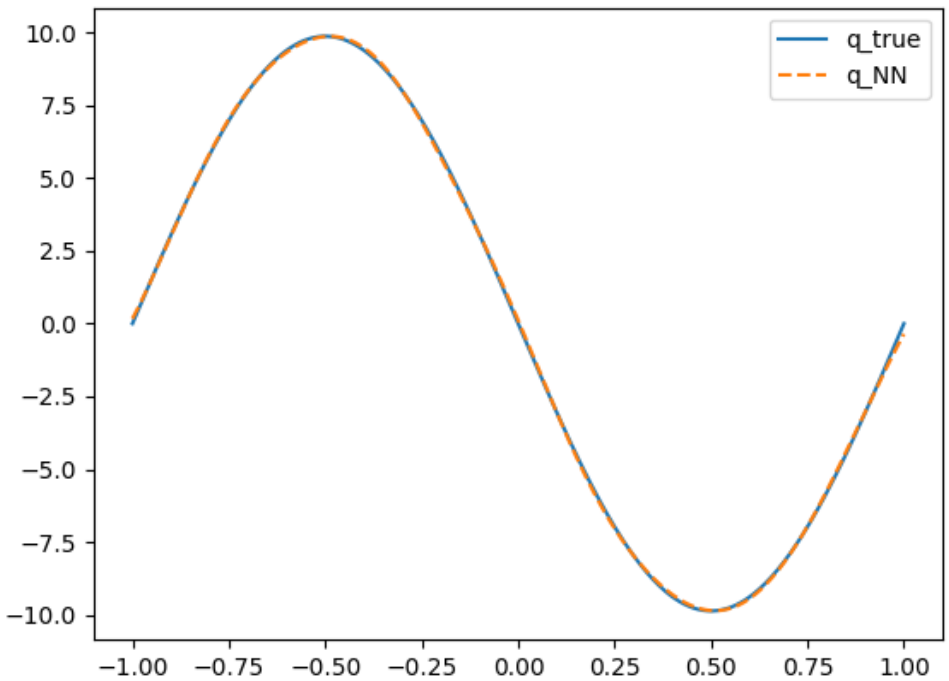
where both $u(x)$ and $q(x)$ are unknown, and we have the measurement of $u(x)$ at 100 points. The reference solution is $u(x) = \sin(\pi x)$ and $q(x) = -\pi^2 \sin(\pi x)$.



Example 12: Poisson equation with unknown forcing field

$$\frac{d^2u}{dx^2} = q(x) \quad \text{for} \quad x \in [-1, 1] \quad \text{and with} \quad u(-1) = u(1) = 0$$

where both $u(x)$ and $q(x)$ are unknown, and we have the measurement of $u(x)$ at 100 points. The reference solution is $u(x) = \sin(\pi x)$ and $q(x) = -\pi^2 \sin(\pi x)$.



Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

$$\text{for} \quad x \in [0,1] \quad \text{and} \quad t \in [0,10] \quad \text{and with} \quad C_A(x,0) = C_B(x,0) = \exp^{-20x}$$

$$\text{and} \quad C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.

1. The DeepXDE and NumPy modules are imported

```
import deepxde as dde
from deepxde.backend import tf
import numpy as np
```

2. Define the unknown variables D and k_f with initial guesses of 1.0 and 0.05 respectively

```
kf = dde.Variable(0.05)
D = dde.Variable(1.0)
```

Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

$$\text{for} \quad x \in [0,1] \quad \text{and} \quad t \in [0,10] \quad \text{and with} \quad C_A(x,0) = C_B(x,0) = \exp^{-20x}$$

$$\text{and} \quad C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.

3. Define a computational domain by using the built-in Interval and TimeDomain classes and combining them with GeometryXTime

```
geom = dde.geometry.Interval(0.0, 1.0)
timedomain = dde.geometry.TimeDomain(0.0, 10.0)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```

Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

for $x \in [0,1]$ and $t \in [0,10]$ and with $C_A(x,0) = C_B(x,0) = \exp^{-20x}$

and $C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.

4. Define the reaction-inverse PDE system

```
def pde(x, y):
    ca, cb = y[:,0:1], y[:,1:2]
    dca_t = dde.grad.jacobian(y, x, i=0, j=1)
    dca_xx = dde.grad.hessian(y, x, component=0, i=0, j=0)
    dcb_t = dde.grad.jacobian(y, x, i=1, j=1)
    dcb_xx = dde.grad.hessian(y, x, component=1, i=0, j=0)
    eq_a = dca_t - 1.0e-3*D*dca_xx + kf*ca*(cb**2)
    eq_b = dcb_t - 1.0e-3*D*dcb_xx + 2.0*kf*ca*(cb**2)
    return [eq_a, eq_b]
```

Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

$$\text{for} \quad x \in [0,1] \quad \text{and} \quad t \in [0,10] \quad \text{and with} \quad C_A(x,0) = C_B(x,0) = \exp^{-20x}$$

$$\text{and} \quad C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.

5. Dirichlet boundary condition

```
def fun_bc(x):
    return 1 - x[:,0:1]
```

5. Initial condition

```
def fun_init(x):
    return np.exp(-20.0*x[:,0:1])
```

Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

$$\text{for} \quad x \in [0,1] \quad \text{and} \quad t \in [0,10] \quad \text{and with} \quad C_A(x,0) = C_B(x,0) = \exp^{-20x}$$

$$\text{and} \quad C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.

6. Dirichlet boundary condition

```
bc_a = dde.icbc.DirichletBC(
        geomtime, fun_bc, lambda _, on_boundary: on_boundary, component=0)
bc_b = dde.icbc.DirichletBC(
        geomtime, fun_bc, lambda _, on_boundary: on_boundary, component=1)
ic1 = dde.icbc.IC(geomtime, fun_init, lambda _,
       on_initial: on_initial, component=0)
ic2 = dde.icbc.IC(geomtime, fun_init, lambda _,
       on_initial: on_initial, component=1)
```

Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

for $x \in [0,1]$ and $t \in [0,10]$ and with $C_A(x,0) = C_B(x,0) = \exp^{-20x}$

and $C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.

7. Generate the training data

```
def gen_traindata():
    data = np.load("reaction.npz")
    t, x, ca, cb = data["t"], data["x"], data["Ca"], data["Cb"]
    X, T = np.meshgrid(x, t)
    X = np.reshape(X, (-1.0, 1.0))
    T = np.reshape(T, (-1.0, 1.0))
    Ca = np.reshape(ca, (-1.0, 1.0))
    Cb = np.reshape(cb, (-1.0, 1.0))
    return np.hstack((X, T)), Ca, Cb
```

Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

$$\text{for} \quad x \in [0,1] \quad \text{and} \quad t \in [0,10] \quad \text{and with} \quad C_A(x,0) = C_B(x,0) = \exp^{-20x}$$

$$\text{and} \quad C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.

8. Organize the data

```
observe_x, Ca, Cb = gen_traindata()
observe_y1 = dde.icbc.PointSetBC(observe_x, Ca, component=0)
observe_y2 = dde.icbc.PointSetBC(observe_x, Cb, component=1)
```

9. Define the PDE problem

```
data = dde.data.TimePDE(
        geomtime, pde,
        [bc_a, bc_b, ic1, ic2, observe_y1, observe_y2],
        num_domain=2000, num_boundary=100,
        num_initial=100, anchors=observe_x, num_test=50000)
```

Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

$$\text{for} \quad x \in [0,1] \quad \text{and} \quad t \in [0,10] \quad \text{and with} \quad C_A(x,0) = C_B(x,0) = \exp^{-20x}$$

$$\text{and} \quad C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.

10. Create the network

```
net = dde.nn.FNN([2] + [20] * 3 + [2], "tanh", "Glorot uniform")
```

11. Create the model

```
model = dde.Model(data, net)
model.compile("adam", lr=0.001, external_trainable_variables=[kf, D])
variable = dde.callbacks.VariableValue([kf, D], period=1000,
            filename="variables.dat")
```

Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

$$\text{for} \quad x \in [0,1] \quad \text{and} \quad t \in [0,10] \quad \text{and with} \quad C_A(x,0) = C_B(x,0) = \exp^{-20x}$$

$$\text{and} \quad C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.

12. Train the network

```
losshistory, train_state = model.train(iterations=80000, callbacks=[variable])
dde.saveplot(losshistory, train_state, issave=True, isplot=True)
```

```
Compiling model...
'compile' took 0.000068 s

Training model...

Step      Train loss
0         [3.34e-02, 1.01e-02, 1.37e+00, 2.52e-01, 8.17e-02, 3.27e-02, 5.17e-01, 1.13e-01]
          Test loss
          [3.21e-02, 9.43e-03, 1.37e+00, 2.52e-01, 8.17e-02, 3.27e-02, 5.17e-01, 1.13e-01]

.....
.....

80000     [4.48e-07, 5.00e-07, 1.78e-07, 1.34e-07, 2.83e-07, 1.70e-07, 8.20e-08, 6.42e-08]
          [3.20e-07, 3.84e-07, 1.78e-07, 1.34e-07, 2.83e-07, 1.70e-07, 8.20e-08, 6.42e-08]

Best model at step 80000:
  train loss: 1.86e-06
  test loss: 1.62e-06
  test metric: []

'train' took 17340.235802 s
```

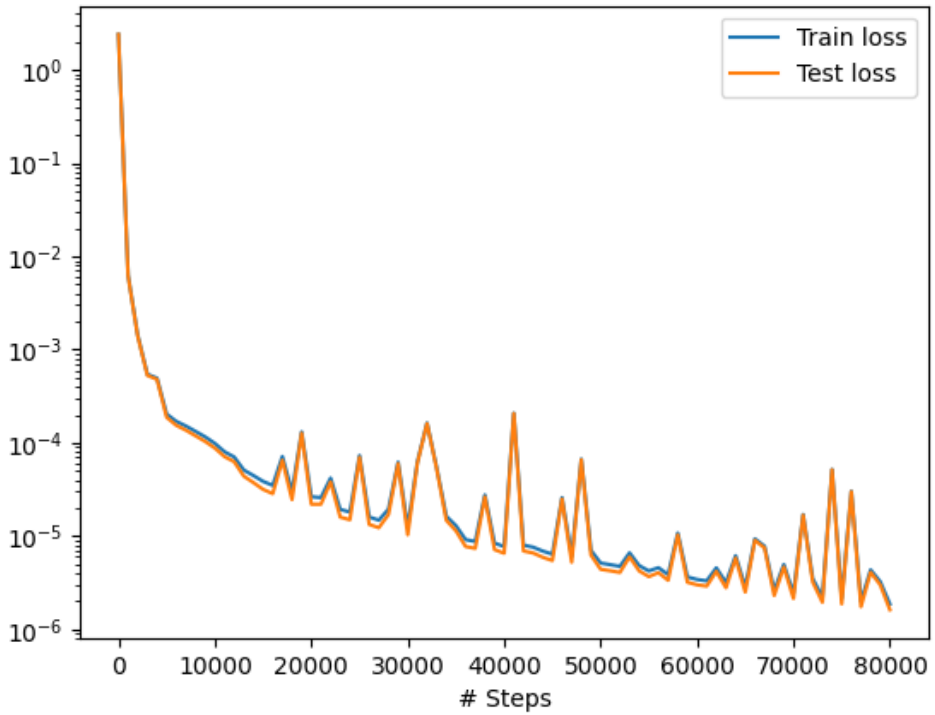
Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

for $x \in [0,1]$ and $t \in [0,10]$ and with $C_A(x,0) = C_B(x,0) = \exp^{-20x}$

and $C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.



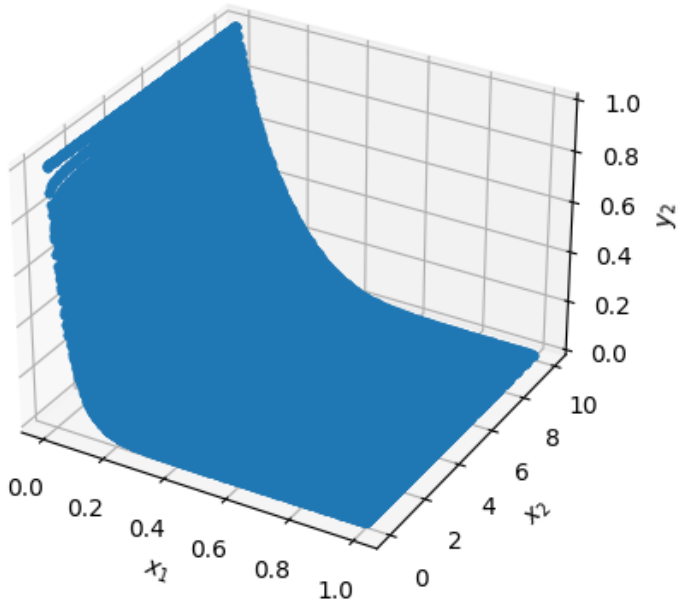
Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

for $x \in [0,1]$ and $t \in [0,10]$ and with $C_A(x,0) = C_B(x,0) = \exp^{-20x}$

and $C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.



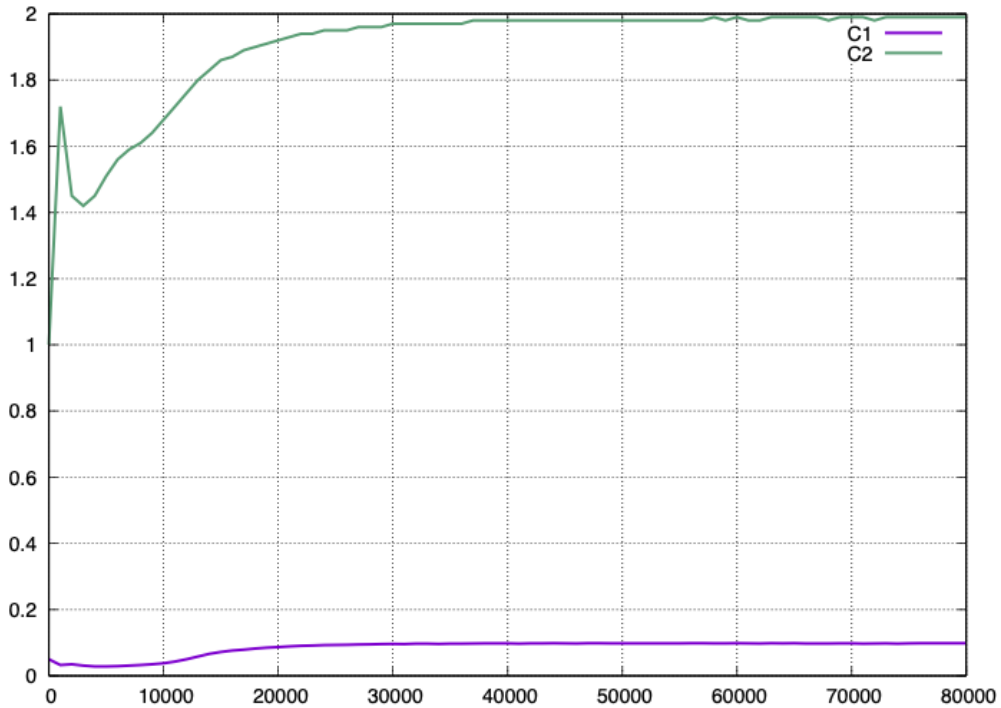
Example 13: diffusion-reaction system

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2 \quad , \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

for $x \in [0,1]$ and $t \in [0,10]$ and with $C_A(x,0) = C_B(x,0) = \exp^{-20x}$

and $C_A(0,t) = C_B(0,t) = 1 \quad , \quad C_A(1,t) = C_B(1,t) = 0$

The expected values of D and k_f are 2.10^{-3} and 0.1 respectively.



Outline

Universal approximation theorem of a NN

- NNs are universal approximators of continuous functions
- A less known and perhaps more powerful result is that a NN with a single hidden layer can approximate accurately any nonlinear continuous operator
- The theorem guarantees only a small approximation error for a sufficient large network, and does not consider the important optimization and generalization errors
- DeepONet: learn operators accurately and efficiently from a relatively small dataset
- A DeepONet consists of two sub-networks
- One for encoding the input function at a fixed number of sensors x_i , $i = 1, .., m$ (branch net)
- One for encoding the locations for the output functions (trunk net)

Universal approximation theorem of a NN

- NNs are universal approximators of continuous functions
- A less known and perhaps more powerful result is that a NN with a single hidden layer can approximate accurately any nonlinear continuous operator
- The theorem guarantees only a small approximation error for a sufficient large network, and does not consider the important optimization and generalization errors
- DeepONet: learn operators accurately and efficiently from a relatively small dataset
- A DeepONet consists of two sub-networks
- One for encoding the input function at a fixed number of sensors x_i , $i = 1, .., m$ (branch net)
- One for encoding the locations for the output functions (trunk net)

Universal approximation theorem of a NN

- A NN with a single hidden layer can approximate accurately any nonlinear continuous functional (a mapping from a space of functions into the real numbers) or (nonlinear) operator (a mapping from a space of functions into another space of functions)
- Let G be an operator taking an input function u
- $G(u)$ is the corresponding output function
- For any point y in the domain of $G(u)$, the output $G(u)(y)$ is a real number
- Hence, the network takes inputs composed of two parts: u and y, and outputs $G(u)(y)$
- Although our goal is to learn operators, which take a function as the input, we have to represent the input functions discretely, so that network approximations can be applied
- A straightforward and simple way, in practice, is to employ the function values at sufficient but finite many locations $x_1, x_2, .., x_m$; we call these locations as sensors

Main result

Theorem - Universal Approximation Theorem for Operator

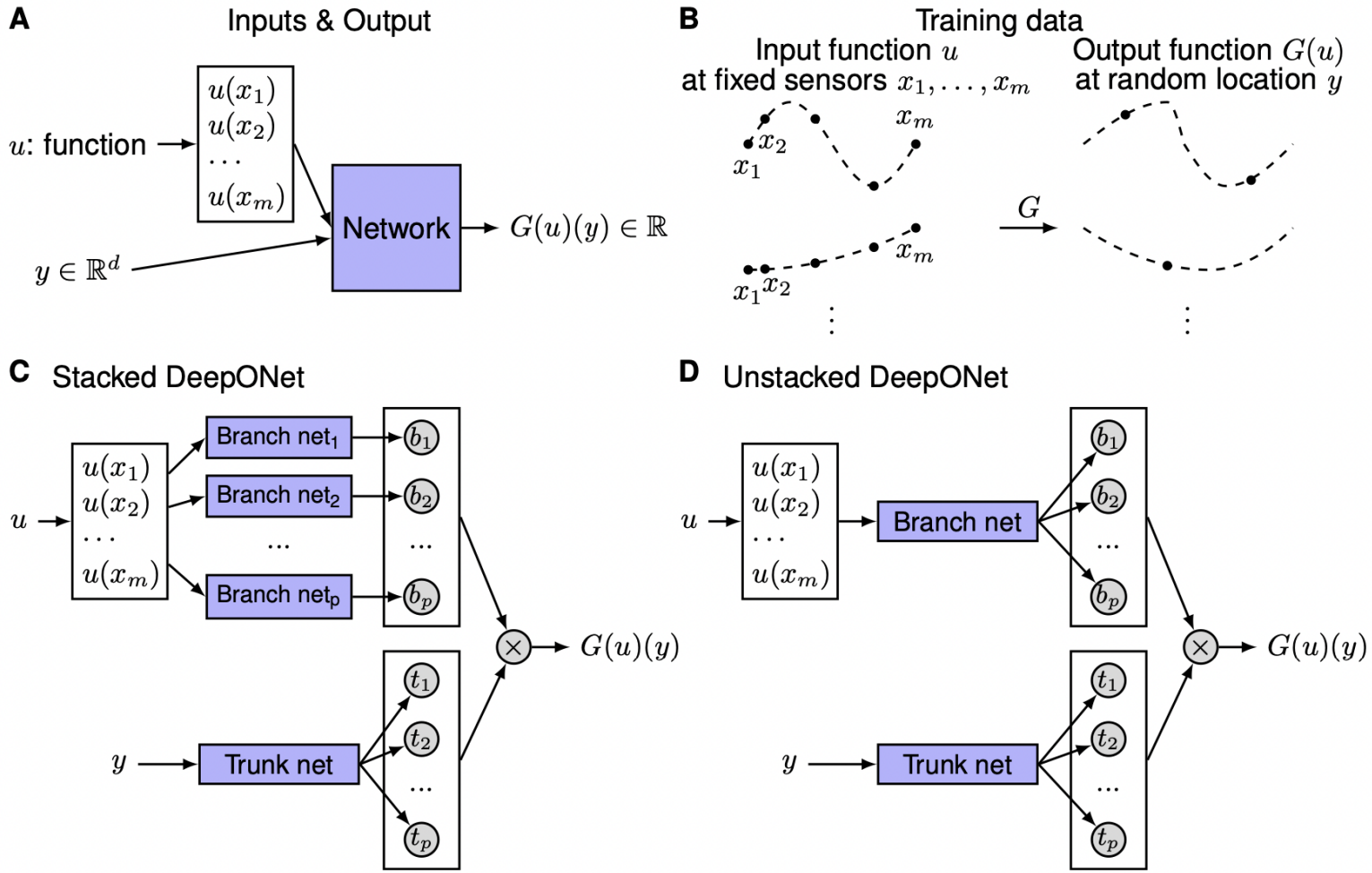
Suppose that σ is a continuous non-polynomial function, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2$. Then for any $\varepsilon > 0$, there are positiive integers n, p, m, constants c_i^k, ξ_{ij}^k, θ_i^k, $\zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, .., n$, $k = 1, ., p$, $j = 1, .., m$ such that

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon$$

holds forall $u \in V$ and $y \in K_2$.

T. Chen and H. Chen
Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks
IEEE Trans. Neur. Netw., Vol. 6, No. 4 (1995)

DeepONet: architectures



(A) The network to learn an operator $G : u \to G(u)$ takes two inputs $[u(x_1), u(x_2), .., u(x_m)]$ and y. (B) Illustration of the training data. For each input function u, we require that we have the same number of evaluations at the same scattered sensors $x_1, x_2, .., x_m$. However, we do not enforce any constraints on the number or locations for the evaluation of output functions. (C) The stacked DeepONet in Theorem has one trunk network and p stacked branch networks. (D) The unstacked DeepONet has one trunk network and one branch network.

Universal approximation theorem of a NN

- A NN with a single hidden layer can approximate accurately any nonlinear continuous functional (a mapping from a space of functions into the real numbers) or (nonlinear) operator (a mapping from a space of functions into another space of functions)
- This approximation theorem indicates the potential application of NNs to learn nonlinear operators from data, i.e., similar to what the DL community is currently doing, that is learning functions from data
- The accuracy of NNs can be characterized by dividing the total error into three main types: approximation, optimization, and generalization
- The universal approximation theorems only guarantee a small approximation error for a sufficiently large network, but they do not consider the optimization error and generalization error at all
- Useful networks should be easy to train, i.e., exhibit small optimization error, and generalize well to unseen data, i.e., exhibit small generalization error

Methodology

- The only requirement for the training dataset is the consistency of the sensors $x_1, x_2, .., x_m$ for input functions
- The network inputs consist of two separate components: $[u(x_1), u(x_2), .., u(x_m)]^T$ and y
- The goal is to achieve good performance by designing the network architecture
- In high dimensional problems, y is a vector with d components, so the dimension of y does not match the dimension of $u(x_i)$ for $i = 1, 2, .., m$
- At least two sub-networks are needed to handle $[u(x_1), u(x_2), .., u(x_m)]^T$ and y separately (Fig. C)
- The trunk network takes y as the input and outputs $[t_1, t_2, .., t_p]^T \in \mathbb{R}^p$
- p branch networks: each of them takes $[u(x_1), u(x_2), .., u(x_m)]^T$ as the input and outputs a scalar $b_k \in \mathbb{R}$
- Both networks are merged

$$G(u)(y) \approx \sum_{k=1}^{p} b_k t_k$$

Methodology

- In practice, p is at least of the order of 10, and using lots of branch networks is computationally and memory expensive
- Hence, we merge all the branch networks into one single branch network (Fig. D)
- A single network outputs $[b_1, b_2, .., b_p]^T \in \mathbb{R}^p$
- DeepONet is a high level network architecture without defining the architectures of its inner trunk and branch networks
- FNN is used as the baseline
- Embodying some prior knowledge into neural network architectures usually induces good generalization
- The success of DeepONet even using FNN as its sub-networks is also due to its strong inductive bias
- The output $G(u)(y)$ has two independent inputs u and y, and thus using the trunk and branch networks explicitly is consistent with this prior knowledge
- More broadly, $G(u)(y)$ can be viewed as a function of y conditioning on u

Methodology: data generation

- $u(x)$ of the process plays an important role in system identification
- The input signal is the only possibility to influence the process in order to gather information about its response
- The quality of the identification signal determines an upper bound on the accuracy that in the best case can be achieved by any model
- One can consider two function spaces: Gaussian random field (GRF) and orthogonal (Chebyshev) polynomials
- We use the mean-zero GRF: $u \approx G(0, k_l(x_1, x_2))$ where the covariance kernel

$$k_l(x_1, x_2) = \exp(-\frac{\| x_1 - x_2 \|^2}{2l^2})$$

is the radial-basis function (RBF) kernel with a length-scale parameter $l > 0$.

Methodology: data generation

- $u(x)$ of the process plays an important role in system identification
- The input signal is the only possibility to influence the process in order to gather information about its response
- The quality of the identification signal determines an upper bound on the accuracy that in the best case can be achieved by any model
- One can consider two function spaces: Gaussian random field (GRF) and orthogonal (Chebyshev) polynomials
- We use the mean-zero GRF: $u \approx G(0, k_l(x_1, x_2))$ where the covariance kernel

$$k_l(x_1, x_2) = \exp(-\frac{\| x_1 - x_2 \|^2}{2l^2})$$

is the radial-basis function (RBF) kernel with a length-scale parameter $l > 0$.

Methodology: data generation

- Let $M > 0$ and T_i are Chebyshev polynomials of the first kind; we define the orthogonal polynomials of degree N as

$$V_{\text{poly}} = \left\{ \sum_{i=0}^{N-1} a_i T_I(x) \ : \ |a_i| \leq M \right\}$$

- We generate the dataset from V_{poly} by randomly sampling a_i from $[-M, M]$ to get a sample of u
- After sampling u from the chosen function spaces, we solve the original ODE or PDE system by a certain numerical method to obtain the reference solutions
- We note that one data point is a triplet $(u, y, G(u)(y))$, and thus one specific input u may appear in multiple data points with different values of y
- For example, a dataset of size 10000 may only be generated from 100 u trajectories, and each evaluates $G(u)(y)$ for 100 y locations

Example 1: Poisson equation in 1D

We want to learn the operator $G : f \to u$ for the 1D Poisson problem

$$\frac{d^2u}{dx^2} = f(x) \quad \text{for} \quad x \in [0, 1] \quad \text{and with} \quad u(0) = u(1) = 0$$

The source term f is supposed to be an arbitrary continuous function.

1. The DeepXDE and NumPy modules are imported

```
import deepxde as dde
from deepxde.backend import tf
import numpy as np
```

Example 1: Poisson equation in 1D

We want to learn the operator $G : f \to u$ for the 1D Poisson problem

$$\frac{d^2u}{dx^2} = f(x) \quad \text{for} \quad x \in [0, 1] \quad \text{and with} \quad u(0) = u(1) = 0$$

The source term f is supposed to be an arbitrary continuous function.

2. Define the PDE with boundary conditions and the domain

```
def equation(x, y, f):
    dy_xx = dde.grad.hessian(y, x)
    return -dy_xx - f

geom = dde.geometry.Interval(0.0 , 1.0)

def u_boundary(_):
    return 0.0

def boundary(_, on_boundary):
    return on_boundary

bc = dde.icbc.DirichletBC(geom, u_boundary, boundary)

pde = dde.data.PDE(geom, equation, bc, num_domain=100, num_boundary=2)
```

Example 1: Poisson equation in 1D

We want to learn the operator $G : f \to u$ for the 1D Poisson problem

$$\frac{d^2u}{dx^2} = f(x) \quad \text{for} \quad x \in [0, 1] \quad \text{and with} \quad u(0) = u(1) = 0$$

The source term f is supposed to be an arbitrary continuous function.

3. Specify the function space for f and the corresponding evaluation points

- Use the dde.data.PowerSeries to get the function space of polynomials of degree 3
- With the PDE, the function space is used to define a PDEOperator dde.data.PDEOperatorCartesianProd that incorporates the PDE into the loss

```
degree = 3
space = dde.data.PowerSeries(N=degree + 1)
num_eval_points = 10
evaluation_points = geom.uniform_points(num_eval_points, boundary=True)
pde_op = dde.data.PDEOperatorCartesianProd(
                pde, space, evaluation_points, num_function=100)
```

Example 1: Poisson equation in 1D

We want to learn the operator $G : f \to u$ for the 1D Poisson problem

$$\frac{d^2u}{dx^2} = f(x) \quad \text{for} \quad x \in [0, 1] \quad \text{and with} \quad u(0) = u(1) = 0$$

The source term f is supposed to be an arbitrary continuous function.

4. Define the DeepONet using dde.nn.DeepONetCartesianProd

- The branch net is chosen as a fully connected neural network of size $[m, 32, p]$ x where $p = 32$
- The trunk net is a fully connected neural network of size $[dim\_x, 32, p]$

```
dim_x = 1

p = 32

net = dde.nn.DeepONetCartesianProd(
            [num_eval_points, 32, p], [dim_x, 32, p],
            activation="tanh",
            kernel_initializer="Glorot normal")
```

Example 1: Poisson equation in 1D

We want to learn the operator $G : f \to u$ for the 1D Poisson problem

$$\frac{d^2u}{dx^2} = f(x) \quad \text{for} \quad x \in [0, 1] \quad \text{and with} \quad u(0) = u(1) = 0$$

The source term f is supposed to be an arbitrary continuous function.

5. Define and train the model

```
model = dde.Model(pde_op, net)
dde.optimizers.set_LBFGS_options(maxiter=1000)
model.compile("L-BFGS")
model.train()
```

6. Use the trained model to predict the solution of the Poisson equation

```
n = 3
features = space.random(n)
fx = space.eval_batch(features, evaluation_points)
x = geom.uniform_points(100, boundary=True)
y = model.predict((fx, x))
```

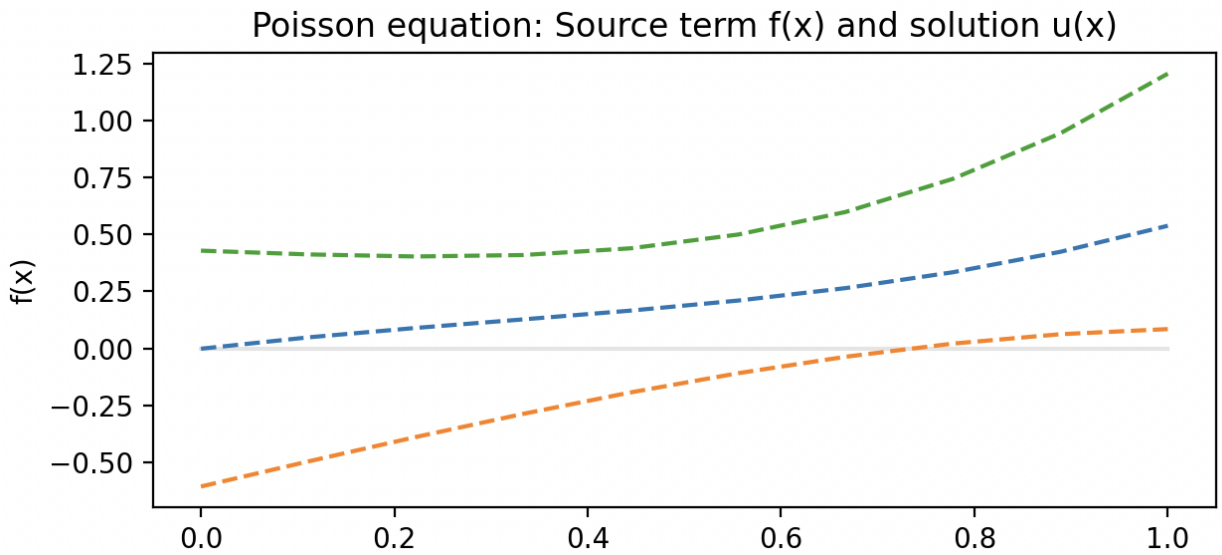
```
Using backend: pytorch

Compiling model...
'compile' took 0.000114 s

Training model...

Step      Train loss              Test loss               Test metric
0         [5.69e-01, 1.51e-02]    [5.69e-01, 1.51e-02]    []
1000      [3.00e-06, 1.43e-07]    [3.00e-06, 1.43e-07]    []

Best model at step 1000:
  train loss: 3.15e-06
  test loss: 3.15e-06
  test metric: []

'train' took 41.223631 s
```

Example 1: Poisson equation in 1D

We want to learn the operator $G : f \to u$ for the 1D Poisson problem

$$\frac{d^2u}{dx^2} = f(x) \quad \text{for} \quad x \in [0,1] \quad \text{and with} \quad u(0) = u(1) = 0$$

The source term f is supposed to be an arbitrary continuous function.



Example 1: Poisson equation in 1D

We want to learn the operator $G : f \to u$ for the 1D Poisson problem

$$\frac{d^2u}{dx^2} = f(x) \quad \text{for} \quad x \in [0, 1] \quad \text{and with} \quad u(0) = u(1) = 0$$

The source term f is supposed to be an arbitrary continuous function.