poisson3d

September 12, 2020

Author: Manu Jayadharan, University of Pittsburgh, 2020

1 Example 2: Using fluidlearn to solve an elliptic pde: 3d Poission equation.

- This is the second example in the series, intended to act as tutorial for fluidlearn package.
- New in this example: how to use one of in-built PDE models. We illustrate this by using the *Poisson* model from the fluidlearn.fluidmodels module.
- We also show how to manufacture boundary conditions easily using the fluidlearn.dataprocess module, for convergence testing and debugging.

Equation to solve: $-\Delta u - f = 0$ over domain Ω .

For demonstration purposes we take $f = -6(x_1 + x_2) - 2$ and $\Omega = [-2, 4] \times [0, 5] \times [-3, 3]$, so we can compare the results with the actual solution $u = x_1^3 + x_2^3 + x_3^2$.

[5]:
```
#Import fluidlearn package and classes
import fluidlearn
from fluidlearn import dataprocess
import numpy as np
```

[ ]:
```
#Import fluidlearn package and classes
import fluidlearn
from fluidlearn import dataprocess
```

1.0.1 Defining the domain and time interval for which the PDE needs to be solved.

This matters only for generating collocation points and if the user is feeding their own collocation points, they can skip this step.

[123]:
```
#domain range
X_1_domain = [-2, 2]
X_2_domain = [0, 1]
X_3_domain = [0,4]

#domain of the problem
domain_bounds = [X_1_domain, X_2_domain, X_3_domain]
```

1.0.2 Manufacturing the boundary data

- We use the fluidlearn.dataprocess.BcIcManufact class to generate points lying on the faces of the hypercube defined by the intervals given in domain_bounds. This is equivalent to randomly selecting points from the domain boundary, $\partial\Omega$.
- We then use our knowledge of the manufactured solution to manufacture the boundary conditions corresponding to these points.
- Note that for this example, we use uniform distribution to randomly select points.

[124]:
```
bc_data_size = 1000 #number of data points on boundary

#object to randomly generate points lying on the boundary
bc_generator = dataprocess.BcIcDataManufact(domain_bounds)
```

[125]:
```
X_data = bc_generator.generate_uniform_bc_ic(bc_data_size)
```

[126]:
```
num_intr_dat = 10
X_data = np.concatenate([X_data,np.random.uniform(0,1,3*num_intr_dat).
 ↪reshape((num_intr_dat,3))])
```

[127]:
```
#Note that we will have bc_data_size number of instances for each boundary
#face
X_data.shape
```

[127]:
```
(6010, 3)
```

Generating the boundary condition from random boundary points using the manufactured solution $u = x_1^3 + x_2^3 + x_3^2$

[130]:
```
import numpy as np
Y_data = (X_data[:,0]**3 + X_data[:,1]**3 + X_data[:,2]**2)[:,np.newaxis]
```

1.0.3 Defining the rhs function $f = -6(x_1 + x_2) - 2$ of the PDE.

[132]:
```
def rhs_function (args, time_dep=False):
        return -6*(args[0]+args[1]) -2
```

1.0.4 Defining the model architecture

[134]:
```
model_type = 'poisson'
space_dim = 3 #dimension of Omega
time_dependent_problem = False
n_hid_lay=6 #numberof hidden layers in the neural network
n_hid_nrn=30 #number of neurons in each hidden layer
act_func='tanh' #activation function used for hidden layers:  could be elu,␣
 ↪relu, sigmoid
```

```
loss_list='mse' #type of error function used for cost functin, we use mean␣
 ↪squared error.
optimizer='adam' #type of optimizer for cost function minimization
dom_bounds=domain_bounds #domain bounds where collocation points has to be␣
 ↪generated

distribution = 'uniform' #type of distribution used for generating the pde␣
 ↪collocation points.
number_of_collocation_points = 10000

batch_size = 32 #batch size for stochastic batch gradient type optimization
num_epochs = 10 #number of epochs used for trainng
```

1.0.5 Defining the fluidlearn solver

[135]:
```
#Instantiation of the fluidlearn.fluildlearn.Solver class
poisson3d_model = fluidlearn.Solver()
```

[136]:
```
poisson3d_model(model_type=model_type,
            space_dim=space_dim,
            time_dep=time_dependent_problem,
            output_dim=1,
            n_hid_lay=n_hid_lay,
            n_hid_nrn=n_hid_lay,
            act_func=act_func,
            rhs_func=rhs_function,
            loss_list=loss_list,
            optimizer=optimizer,
            dom_bounds=dom_bounds,
            load_model=False,
            model_path=None)
```

1.0.6 Fitting the model

[137]:
```
poisson3d_model.fit(
    x=X_data,
    y=Y_data,
    colloc_points=number_of_collocation_points,
    dist=distribution,
    batch_size=batch_size,
    epochs=num_epochs,
)
```

```
Epoch 1/10
501/501 [==============================] - 8s 15ms/step - loss: 31.9742 -
output_1_loss: 16.1745 - output_2_loss: 15.7997
```

```
Epoch 2/10
501/501 [==============================] - 8s 17ms/step - loss: 20.7385 -
output_1_loss: 12.2779 - output_2_loss: 8.4605
Epoch 3/10
501/501 [==============================] - 8s 16ms/step - loss: 11.8072 -
output_1_loss: 7.1775 - output_2_loss: 4.6297
Epoch 4/10
501/501 [==============================] - 9s 17ms/step - loss: 6.0623 -
output_1_loss: 3.9346 - output_2_loss: 2.1277
Epoch 5/10
501/501 [==============================] - 8s 17ms/step - loss: 3.7272 -
output_1_loss: 2.5057 - output_2_loss: 1.2215
Epoch 6/10
501/501 [==============================] - 8s 17ms/step - loss: 2.2637 -
output_1_loss: 1.5694 - output_2_loss: 0.6943
Epoch 7/10
501/501 [==============================] - 9s 18ms/step - loss: 1.5779 -
output_1_loss: 1.1181 - output_2_loss: 0.4598
Epoch 8/10
501/501 [==============================] - 9s 18ms/step - loss: 1.1895 -
output_1_loss: 0.8162 - output_2_loss: 0.3734
Epoch 9/10
501/501 [==============================] - 9s 17ms/step - loss: 0.8678 -
output_1_loss: 0.6147 - output_2_loss: 0.2531
Epoch 10/10
501/501 [==============================] - 10s 21ms/step - loss: 0.7599 -
output_1_loss: 0.5414 - output_2_loss: 0.2185
```

1.0.7 Resuming Training the model again for 50 more epochs

[138]:
```
poisson3d_model.fit(
    x=X_data,
    y=Y_data,
    colloc_points=number_of_collocation_points,
    dist=distribution,
    batch_size=batch_size,
    epochs=20,
)
```

```
Epoch 1/20
501/501 [==============================] - 8s 16ms/step - loss: 0.5891 -
output_1_loss: 0.4347 - output_2_loss: 0.1544
Epoch 2/20
501/501 [==============================] - 7s 15ms/step - loss: 0.5025 -
output_1_loss: 0.3705 - output_2_loss: 0.1320
Epoch 3/20
501/501 [==============================] - 8s 16ms/step - loss: 0.4789 -
output_1_loss: 0.3509 - output_2_loss: 0.1280
```

```
Epoch 4/20
501/501 [==============================] - 8s 16ms/step - loss: 0.4606 -
output_1_loss: 0.3293 - output_2_loss: 0.1313
Epoch 5/20
501/501 [==============================] - 10s 21ms/step - loss: 0.3550 -
output_1_loss: 0.2634 - output_2_loss: 0.0915
Epoch 6/20
501/501 [==============================] - 9s 18ms/step - loss: 0.3180 -
output_1_loss: 0.2331 - output_2_loss: 0.0849
Epoch 7/20
501/501 [==============================] - 9s 17ms/step - loss: 0.3354 -
output_1_loss: 0.2379 - output_2_loss: 0.0975
Epoch 8/20
501/501 [==============================] - 9s 18ms/step - loss: 0.2543 -
output_1_loss: 0.1872 - output_2_loss: 0.0671
Epoch 9/20
501/501 [==============================] - 8s 16ms/step - loss: 0.2204 -
output_1_loss: 0.1583 - output_2_loss: 0.0621
Epoch 10/20
501/501 [==============================] - 9s 18ms/step - loss: 0.2538 -
output_1_loss: 0.1765 - output_2_loss: 0.0773
Epoch 11/20
501/501 [==============================] - 9s 18ms/step - loss: 0.1924 -
output_1_loss: 0.1332 - output_2_loss: 0.0592
Epoch 12/20
501/501 [==============================] - 9s 18ms/step - loss: 0.2467 -
output_1_loss: 0.1605 - output_2_loss: 0.0862
Epoch 13/20
501/501 [==============================] - 9s 18ms/step - loss: 0.2176 -
output_1_loss: 0.1437 - output_2_loss: 0.0740
Epoch 14/20
501/501 [==============================] - 9s 18ms/step - loss: 0.1409 -
output_1_loss: 0.0966 - output_2_loss: 0.0443
Epoch 15/20
501/501 [==============================] - 9s 18ms/step - loss: 0.1919 -
output_1_loss: 0.1249 - output_2_loss: 0.0671
Epoch 16/20
501/501 [==============================] - 9s 18ms/step - loss: 0.1304 -
output_1_loss: 0.0883 - output_2_loss: 0.0421
Epoch 17/20
501/501 [==============================] - 9s 18ms/step - loss: 0.1378 -
output_1_loss: 0.0890 - output_2_loss: 0.0488
Epoch 18/20
501/501 [==============================] - 9s 18ms/step - loss: 0.0826 -
output_1_loss: 0.0556 - output_2_loss: 0.0270
Epoch 19/20
501/501 [==============================] - 9s 18ms/step - loss: 0.1493 -
output_1_loss: 0.0936 - output_2_loss: 0.0557
```

```
Epoch 20/20
501/501 [==============================] - 9s 18ms/step - loss: 0.0744 -
output_1_loss: 0.0499 - output_2_loss: 0.0245
```

1.0.8 Demo Using the trained model for predicton

[149]:
```
num_test_points = 500
X_test = np.random.uniform(0,1,3*num_test_points).reshape(num_test_points,3)
X_test = np.concatenate([np.random.
 ↪uniform(X_1_domain[0],X_1_domain[1],num_test_points).
 ↪reshape(num_test_points,1),
                          np.random.
 ↪uniform(X_2_domain[0],X_2_domain[1],num_test_points).
 ↪reshape(num_test_points,1),
                          np.random.
 ↪uniform(X_3_domain[0],X_3_domain[1],num_test_points).
 ↪reshape(num_test_points,1)],
                         axis=1)
```

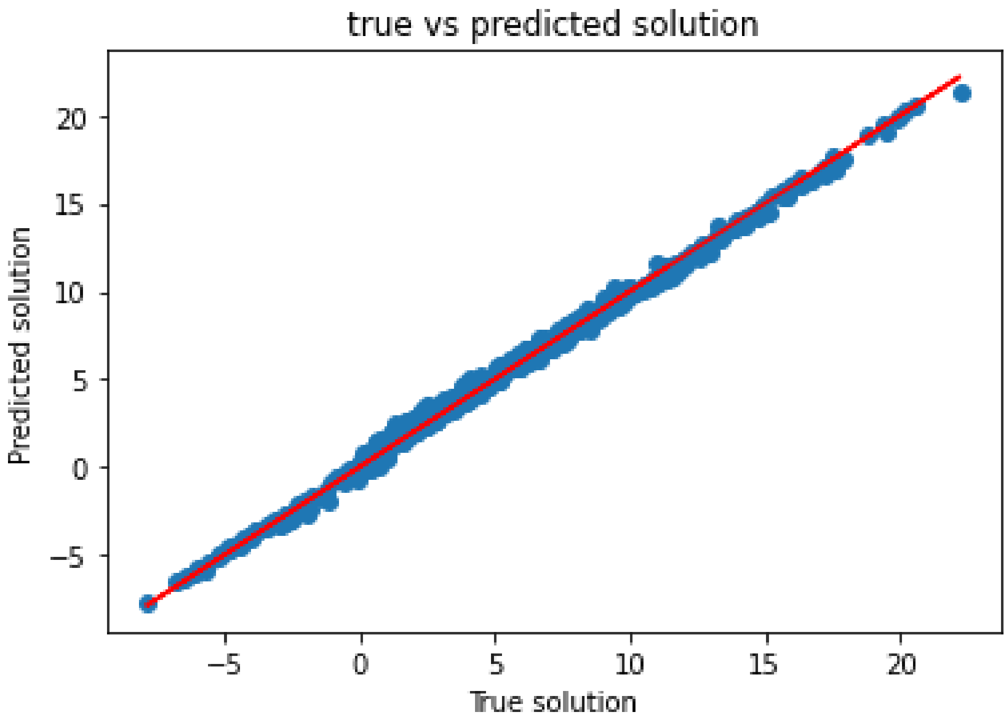
[150]:
```
Y_test = X_test[:,0]**3 + X_test[:,1]**3 + X_test[:,2]**2
```

[151]:
```
Y_pred = poisson3d_model.predict(X_test)
```

[152]:
```
Y_pred = np.squeeze(Y_pred,axis=1)
Y_pred.shape
```

[152]:
```
(500,)
```

[153]:
```
plt.plot(Y_test,Y_pred,'o',Y_test, Y_test,'r-')
plt.title("true vs predicted solution")
plt.xlabel("True solution")
plt.ylabel("Predicted solution")
plt.show()
```



[48]:
```python
path_to_save_model = "saved_model/model_name"
poisson3d_model.save_model(path_to_save_model)
```

```
WARNING:tensorflow:From C:\Users\manuj\anaconda3\envs\tf2\lib\site-
packages\tensorflow\python\training\tracking\tracking.py:111:
Model.state_updates (from tensorflow.python.keras.engine.training) is deprecated
and will be removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
WARNING:tensorflow:From C:\Users\manuj\anaconda3\envs\tf2\lib\site-
packages\tensorflow\python\training\tracking\tracking.py:111: Layer.updates
(from tensorflow.python.keras.engine.base_layer) is deprecated and will be
removed in a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
INFO:tensorflow:Assets written to: saved_model/model_name\assets
```