difussion_equation

August 17, 2020

Author: Manu Jayadharan, University of Pittsburgh, 2020

1 Using fluidlearn to solve diffusion equation

Equation to solve: $u_t - \Delta u - f = 0$ over domain Ω from time T_initial to T_final.

For demonstration purposes we take $f = sin(x_1 + x_2) + tsin(x_1 + x_2)$ and $\Omega = [-2, 2] \times [0, 1]$ and the time interval to be $[0, 1]$, so we can compare the results with the actual solution $u = tsin(x_1 + x_2)$.

[1]:
```
#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.

[2]:
```
#domain range
X_1_domain = [-2, 2]
X_2_domain = [0, 1]
#time range
T_initial = 0
T_final = 1
T_domain = [T_initial, T_final]

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

1.0.2 Loading data from a csv file

- We use the manufactured data with $u = tsin(x_1 + x_2)$ saved in a csv file.
- Data is saved in the format: $(x_1, x_2, t, u(x_1, x_2, t))$ as four columns.
- You could load either preprocess your data to be in this format or load your data from a csv file with similar format.

[3]:
```
path_to_data = "data_manufactured/t_sin_x_plus_y.csv"
X_data, Y_data = dataprocess.imp_from_csv(path_to_csv_file=path_to_data,
                                           x_y_combined=True, y_dim=1)
```

1.0.3 Defining the rhs function $f = sin(x_1 + x_2) + tsin(x_1 + x_2)$ of the PDE.

We use tensorflow.sin function instead of python functions, we could used numpy.sin as well.

[4]:
```
def rhs_function (args, time_dep=True):
        import tensorflow as tf
        if time_dep:
            space_inputs = args[:-1]
            time_inputs = args[-1]
        else:
            space_inputs = args

        return tf.sin(space_inputs[0]+space_inputs[1]) + 2*time_inputs*tf.
 ↪sin(space_inputs[0]+space_inputs[1])
```

1.0.4 Defining the model architecture

[8]:
```
model_type = 'forward'
space_dim = 2 #dimension of Omega
time_depedent_problem = True
n_hid_lay=3 #numberof hidden layers in the neural network
n_hid_nrn=20 #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 = 5000

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

[9]:
```
diffusion_model = fluidlearn.Solver()
```

[10]:
```
diffusion_model(model_type=model_type,
            space_dim=space_dim,
            time_dep=time_depedent_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

[16]:
```
diffusion_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
219/219 [==============================] - 0s 1ms/step - loss: 0.0012 -
output_1_loss: 5.9274e-04 - output_2_loss: 5.6584e-04
Epoch 2/10
219/219 [==============================] - 0s 1ms/step - loss: 0.0012 -
output_1_loss: 5.8445e-04 - output_2_loss: 5.7774e-04
Epoch 3/10
219/219 [==============================] - 0s 1ms/step - loss: 0.0012 -
output_1_loss: 5.8166e-04 - output_2_loss: 5.6921e-04
Epoch 4/10
219/219 [==============================] - 0s 1ms/step - loss: 0.0011 -
output_1_loss: 5.9059e-04 - output_2_loss: 5.3954e-04
Epoch 5/10
219/219 [==============================] - 0s 1ms/step - loss: 0.0011 -
output_1_loss: 5.7146e-04 - output_2_loss: 5.3106e-04
Epoch 6/10
219/219 [==============================] - 0s 1ms/step - loss: 0.0011 -
output_1_loss: 5.7799e-04 - output_2_loss: 5.1664e-04
Epoch 7/10
219/219 [==============================] - 0s 1ms/step - loss: 0.0011 -
output_1_loss: 5.8432e-04 - output_2_loss: 5.0667e-04
Epoch 8/10
219/219 [==============================] - 0s 1ms/step - loss: 0.0011 -
output_1_loss: 5.6643e-04 - output_2_loss: 4.9963e-04
Epoch 9/10
219/219 [==============================] - 0s 1ms/step - loss: 0.0010 -
output_1_loss: 5.6436e-04 - output_2_loss: 4.8168e-04
Epoch 10/10
```

```
219/219 [==============================] - 0s 1ms/step - loss: 0.0010 -
output_1_loss: 5.6857e-04 - output_2_loss: 4.7589e-04
```

1.0.7 Resuming Training the model again for 50 more epochs

[17]:
```python
diffusion_model.fit(
    x=X_data,
    y=Y_data,
    colloc_points=number_of_collocation_points,
    dist=distribution,
    batch_size=batch_size,
    epochs=50,
)
```

```
Epoch 1/50
219/219 [==============================] - 0s 1ms/step - loss: 0.0010 -
output_1_loss: 5.5980e-04 - output_2_loss: 4.7206e-04
Epoch 2/50
219/219 [==============================] - 0s 1ms/step - loss: 0.0010 -
output_1_loss: 5.6017e-04 - output_2_loss: 4.6657e-04
Epoch 3/50
219/219 [==============================] - 0s 1ms/step - loss: 0.0010 -
output_1_loss: 5.5690e-04 - output_2_loss: 4.6706e-04
Epoch 4/50
219/219 [==============================] - 0s 1ms/step - loss: 0.0010 -
output_1_loss: 5.6190e-04 - output_2_loss: 4.6779e-04
Epoch 5/50
219/219 [==============================] - 0s 1ms/step - loss: 9.9724e-04 -
output_1_loss: 5.5509e-04 - output_2_loss: 4.4215e-04
Epoch 6/50
219/219 [==============================] - 0s 1ms/step - loss: 9.8571e-04 -
output_1_loss: 5.5845e-04 - output_2_loss: 4.2726e-04
Epoch 7/50
219/219 [==============================] - 0s 1ms/step - loss: 9.9226e-04 -
output_1_loss: 5.5868e-04 - output_2_loss: 4.3358e-04
Epoch 8/50
219/219 [==============================] - 0s 1ms/step - loss: 9.7645e-04 -
output_1_loss: 5.5116e-04 - output_2_loss: 4.2529e-04
Epoch 9/50
219/219 [==============================] - 0s 1ms/step - loss: 9.4746e-04 -
output_1_loss: 5.4082e-04 - output_2_loss: 4.0664e-04
Epoch 10/50
219/219 [==============================] - 0s 1ms/step - loss: 9.5962e-04 -
output_1_loss: 5.5074e-04 - output_2_loss: 4.0888e-04
Epoch 11/50
219/219 [==============================] - 0s 1ms/step - loss: 9.7768e-04 -
output_1_loss: 5.6199e-04 - output_2_loss: 4.1570e-04
Epoch 12/50
```

```
219/219 [==============================] - 0s 1ms/step - loss: 9.5774e-04 -
output_1_loss: 5.4138e-04 - output_2_loss: 4.1637e-04
Epoch 13/50
219/219 [==============================] - 0s 1ms/step - loss: 9.3080e-04 -
output_1_loss: 5.3598e-04 - output_2_loss: 3.9482e-04
Epoch 14/50
219/219 [==============================] - 0s 1ms/step - loss: 9.2209e-04 -
output_1_loss: 5.3978e-04 - output_2_loss: 3.8232e-04
Epoch 15/50
219/219 [==============================] - 0s 1ms/step - loss: 9.2861e-04 -
output_1_loss: 5.3781e-04 - output_2_loss: 3.9080e-04
Epoch 16/50
219/219 [==============================] - 0s 1ms/step - loss: 9.1883e-04 -
output_1_loss: 5.3864e-04 - output_2_loss: 3.8019e-04
Epoch 17/50
219/219 [==============================] - 0s 1ms/step - loss: 9.1839e-04 -
output_1_loss: 5.4479e-04 - output_2_loss: 3.7360e-04
Epoch 18/50
219/219 [==============================] - 0s 1ms/step - loss: 8.9556e-04 -
output_1_loss: 5.2353e-04 - output_2_loss: 3.7202e-04
Epoch 19/50
219/219 [==============================] - 0s 1ms/step - loss: 9.3620e-04 -
output_1_loss: 5.3704e-04 - output_2_loss: 3.9916e-04
Epoch 20/50
219/219 [==============================] - 0s 1ms/step - loss: 9.0400e-04 -
output_1_loss: 5.2691e-04 - output_2_loss: 3.7709e-04
Epoch 21/50
219/219 [==============================] - 0s 1ms/step - loss: 9.1851e-04 -
output_1_loss: 5.4153e-04 - output_2_loss: 3.7698e-04
Epoch 22/50
219/219 [==============================] - 0s 1ms/step - loss: 8.8642e-04 -
output_1_loss: 5.3385e-04 - output_2_loss: 3.5256e-04
Epoch 23/50
219/219 [==============================] - 0s 1ms/step - loss: 8.9163e-04 -
output_1_loss: 5.2564e-04 - output_2_loss: 3.6599e-04
Epoch 24/50
219/219 [==============================] - 0s 1ms/step - loss: 8.9880e-04 -
output_1_loss: 5.3118e-04 - output_2_loss: 3.6761e-04
Epoch 25/50
219/219 [==============================] - 0s 1ms/step - loss: 8.8967e-04 -
output_1_loss: 5.2640e-04 - output_2_loss: 3.6327e-04
Epoch 26/50
219/219 [==============================] - 0s 1ms/step - loss: 8.9322e-04 -
output_1_loss: 5.3548e-04 - output_2_loss: 3.5774e-04
Epoch 27/50
219/219 [==============================] - 0s 1ms/step - loss: 8.7614e-04 -
output_1_loss: 5.2109e-04 - output_2_loss: 3.5505e-04
Epoch 28/50
```

```
219/219 [==============================] - 0s 1ms/step - loss: 8.6327e-04 -
output_1_loss: 5.2088e-04 - output_2_loss: 3.4240e-04
Epoch 29/50
219/219 [==============================] - 0s 1ms/step - loss: 8.6831e-04 -
output_1_loss: 5.1809e-04 - output_2_loss: 3.5022e-04
Epoch 30/50
219/219 [==============================] - 0s 1ms/step - loss: 8.6826e-04 -
output_1_loss: 5.2257e-04 - output_2_loss: 3.4569e-04
Epoch 31/50
219/219 [==============================] - 0s 1ms/step - loss: 8.7395e-04 -
output_1_loss: 5.2565e-04 - output_2_loss: 3.4830e-04
Epoch 32/50
219/219 [==============================] - 0s 1ms/step - loss: 8.5699e-04 -
output_1_loss: 5.0775e-04 - output_2_loss: 3.4923e-04
Epoch 33/50
219/219 [==============================] - 0s 1ms/step - loss: 8.6955e-04 -
output_1_loss: 5.2541e-04 - output_2_loss: 3.4414e-04
Epoch 34/50
219/219 [==============================] - 0s 1ms/step - loss: 8.5461e-04 -
output_1_loss: 5.2507e-04 - output_2_loss: 3.2954e-04
Epoch 35/50
219/219 [==============================] - 0s 1ms/step - loss: 8.2575e-04 -
output_1_loss: 5.0769e-04 - output_2_loss: 3.1806e-04
Epoch 36/50
219/219 [==============================] - 0s 1ms/step - loss: 8.4777e-04 -
output_1_loss: 5.2176e-04 - output_2_loss: 3.2600e-04
Epoch 37/50
219/219 [==============================] - 0s 1ms/step - loss: 8.1819e-04 -
output_1_loss: 5.0107e-04 - output_2_loss: 3.1712e-04
Epoch 38/50
219/219 [==============================] - 0s 1ms/step - loss: 8.2468e-04 -
output_1_loss: 5.0548e-04 - output_2_loss: 3.1920e-04
Epoch 39/50
219/219 [==============================] - 0s 1ms/step - loss: 8.2768e-04 -
output_1_loss: 5.0262e-04 - output_2_loss: 3.2506e-04
Epoch 40/50
219/219 [==============================] - 0s 1ms/step - loss: 8.0858e-04 -
output_1_loss: 4.9870e-04 - output_2_loss: 3.0988e-04
Epoch 41/50
219/219 [==============================] - 0s 1ms/step - loss: 8.0637e-04 -
output_1_loss: 4.9633e-04 - output_2_loss: 3.1005e-04
Epoch 42/50
219/219 [==============================] - 0s 1ms/step - loss: 7.9516e-04 -
output_1_loss: 4.9447e-04 - output_2_loss: 3.0069e-04
Epoch 43/50
219/219 [==============================] - 0s 1ms/step - loss: 8.0102e-04 -
output_1_loss: 4.9453e-04 - output_2_loss: 3.0649e-04
Epoch 44/50
```

```
219/219 [==============================] - 0s 1ms/step - loss: 7.9965e-04 -
output_1_loss: 5.0271e-04 - output_2_loss: 2.9695e-04
Epoch 45/50
219/219 [==============================] - 0s 1ms/step - loss: 7.7721e-04 -
output_1_loss: 4.9079e-04 - output_2_loss: 2.8642e-04
Epoch 46/50
219/219 [==============================] - 0s 1ms/step - loss: 7.7830e-04 -
output_1_loss: 4.8906e-04 - output_2_loss: 2.8924e-04
Epoch 47/50
219/219 [==============================] - 0s 1ms/step - loss: 8.1743e-04 -
output_1_loss: 5.1115e-04 - output_2_loss: 3.0628e-04
Epoch 48/50
219/219 [==============================] - 0s 1ms/step - loss: 7.9719e-04 -
output_1_loss: 4.9210e-04 - output_2_loss: 3.0509e-04
Epoch 49/50
219/219 [==============================] - 0s 1ms/step - loss: 7.7435e-04 -
output_1_loss: 4.9389e-04 - output_2_loss: 2.8046e-04
Epoch 50/50
219/219 [==============================] - 0s 1ms/step - loss: 7.7420e-04 -
output_1_loss: 4.8990e-04 - output_2_loss: 2.8430e-04
```

1.0.8 Demo Using the trained model for predicton

[32]:
```
#taking two points from the domain for time t=0.3 and t=0.76 respectively
x_test_points = [[-0.5,0.1,0.3],
                 [0.66,0.6,0.76]]
#Predicting the value
y_predicted = diffusion_model.predict(x_test_points)
```

[33]:
```
#finding the true y value for comparing
import numpy as np
x_test_points = np.array(x_test_points)
y_true = np.sin(x_test_points[:,0:1] + x_test_points[:,1:2]) * x_test_points[:
 ↪,2:3]
```

[36]:
```
#looking at predicted and true solution side by side.
```

[37]:
```
np.concatenate([y_predicted, y_true], axis=1)
```

[37]:
```
array([[-0.1297535 , -0.1168255 ],
       [ 0.70116615,  0.72358866]])
```

Note that we need more training for further improving the accuracy.

1.0.9 Saving the model to a specified location.

[48]:
```
path_to_save_model = "saved_model/model_name"
diffusion_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
```

1.0.10 Loading the saved model

[6]:
```
path_to_load_model = "saved_model/model_name"
```

[7]:
```
loaded_diffusion_model = fluidlearn.Solver()
```

[8]:
```
loaded_diffusion_model(space_dim=2,
    time_dep=True,
    load_model=True,
    model_path=path_to_load_model)
```

1.0.11 Predicting using loaded model

[9]:
```
y_predicted = loaded_diffusion_model.predict(X_data)
```

[10]:
```
y_predicted
```

[10]:
```
array([[-0.10157388],
       [-0.4190994 ],
       [-0.7965628 ],
       …,
       [-0.10375804],
       [ 0.05802408],
       [-0.00470909]], dtype=float32)
```

[ ]:
```

```