HW 3_Changhao Xu

Monday, February 27, 2023 14:00

1. **Environment setup**

   n_modes_height = n_modes_width = 12 and hidden_channels=32

```
[14]: tfno = TFNO2d(n_modes_height=12, n_modes_width=12, hidden_channels=32,
                    factorization=None, skip='linear')

[15]: tfno

[15]: TFNO2d(
        (convs): FactorizedSpectralConv2d(
          (weight): ModuleList(
            (0): ComplexDenseTensor(shape=torch.Size([32, 32, 6, 6]), rank=None)
            (1): ComplexDenseTensor(shape=torch.Size([32, 32, 6, 6]), rank=None)
            (2): ComplexDenseTensor(shape=torch.Size([32, 32, 6, 6]), rank=None)
            (3): ComplexDenseTensor(shape=torch.Size([32, 32, 6, 6]), rank=None)
            (4): ComplexDenseTensor(shape=torch.Size([32, 32, 6, 6]), rank=None)
            (5): ComplexDenseTensor(shape=torch.Size([32, 32, 6, 6]), rank=None)
            (6): ComplexDenseTensor(shape=torch.Size([32, 32, 6, 6]), rank=None)
            (7): ComplexDenseTensor(shape=torch.Size([32, 32, 6, 6]), rank=None)
          )
        )
        (fno_skips): ModuleList(
          (0): Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (2): Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): Conv2d(32, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
        )
        (lifting): Lifting(
          (fc): Conv2d(3, 32, kernel_size=(1, 1), stride=(1, 1))
        )
        (projection): Projection(
          (fc1): Conv2d(32, 256, kernel_size=(1, 1), stride=(1, 1))
          (fc2): Conv2d(256, 1, kernel_size=(1, 1), stride=(1, 1))
        )
      )
```

2. **Darcy Flow equation**

   After 150 epochs of training, the final training error is 0.03, and the testing h1 error on 32 resolution is 0.0765, testing l2 error on 32 resolution is 0.0374, testing h1 error on 64 resolution is 0.1596, testing l2 error on 64 resolution is 0.0554.

```
[137] time=2.00, avg_loss=0.6537, train_err=0.0327, 32_h1=0.0770, 32_l2=0.0383, 64_h1=0.1593, 64_l2=0.0536
[138] time=2.00, avg_loss=0.6539, train_err=0.0327, 32_h1=0.0766, 32_l2=0.0374, 64_h1=0.1564, 64_l2=0.0538
[139] time=2.00, avg_loss=0.6745, train_err=0.0337, 32_h1=0.0770, 32_l2=0.0374, 64_h1=0.1594, 64_l2=0.0538
[140] time=2.00, avg_loss=0.6575, train_err=0.0329, 32_h1=0.0765, 32_l2=0.0376, 64_h1=0.1603, 64_l2=0.0529
[141] time=2.02, avg_loss=0.6313, train_err=0.0316, 32_h1=0.0777, 32_l2=0.0380, 64_h1=0.1586, 64_l2=0.0530
[142] time=2.02, avg_loss=0.6365, train_err=0.0318, 32_h1=0.0763, 32_l2=0.0372, 64_h1=0.1594, 64_l2=0.0555
[143] time=2.03, avg_loss=0.6258, train_err=0.0313, 32_h1=0.0762, 32_l2=0.0365, 64_h1=0.1586, 64_l2=0.0533
[144] time=2.02, avg_loss=0.6438, train_err=0.0322, 32_h1=0.0763, 32_l2=0.0367, 64_h1=0.1580, 64_l2=0.0544
[145] time=2.03, avg_loss=0.6265, train_err=0.0313, 32_h1=0.0765, 32_l2=0.0374, 64_h1=0.1591, 64_l2=0.0552
[146] time=2.03, avg_loss=0.6416, train_err=0.0321, 32_h1=0.0768, 32_l2=0.0372, 64_h1=0.1572, 64_l2=0.0521
[147] time=2.02, avg_loss=0.6175, train_err=0.0309, 32_h1=0.0767, 32_l2=0.0378, 64_h1=0.1580, 64_l2=0.0550
[148] time=2.03, avg_loss=0.6193, train_err=0.0310, 32_h1=0.0762, 32_l2=0.0371, 64_h1=0.1573, 64_l2=0.0531
[149] time=2.03, avg_loss=0.5997, train_err=0.0300, 32_h1=0.0765, 32_l2=0.0374, 64_h1=0.1596, 64_l2=0.0554
```

   In order to further reduce testing l2 error on 64 resolution, in config file we change the training resolution to 64, as shown below:

   n_modes_height = n_modes_width = 64

   batch_size: 64

   test_resolutions: [64, 64]

   test_batch_sizes: [64, 64]

```
# FNO related
tfno2d:
  data_channels: 3
  n_modes_height: 64
  n_modes_width: 64
  hidden_channels: 64
  projection_channels: 256
  n_layers: 4
  domain_padding: None #0.078125
  domain_padding_mode: 'one-sided' #symmetric
  fft_norm: 'forward'
  norm: 'group_norm'
  skip: 'linear'
  implementation: 'factorized'
  separable: 0
  preactivation: 0
```

```
# Optimizer
opt:
  n_epochs: 150
  learning_rate: 5e-3
  training_loss: 'h1'
  weight_decay: 1e-4
  amp_autocast: False

  scheduler_T_max: 300 # For cosine only, typically take n_epochs
  scheduler_patience: 5 # For ReduceLROnPlateau only
  scheduler: 'CosineAnnealingLR' # Or 'CosineAnnealingLR' OR 'ReduceLROnPlateau' OR 'StepLR'
  step_size: 50
  gamma: 0.5

# Dataset related
data:
  folder: "/dli/task/bootcamp/data/darcy_flow/"
  batch_size: 64
  n_train: 4000
  train_resolution: 64
  n_tests: [500, 500]
  test_resolutions: [64, 64]
  test_batch_sizes: [64, 64]
  positional_encoding: True
```

And the error result from training 120-150 epoch is shown below (64_l2 = 0.0278 < 5%):

```
[120] time=9.41, avg_loss=0.8245, train_err=0.0309, 64_h1=0.0587, 64_l2=0.0308
[121] time=9.38, avg_loss=0.8243, train_err=0.0309, 64_h1=0.0567, 64_l2=0.0273
[122] time=9.37, avg_loss=0.8178, train_err=0.0307, 64_h1=0.0570, 64_l2=0.0274
[123] time=9.40, avg_loss=0.8530, train_err=0.0320, 64_h1=0.0565, 64_l2=0.0261
[124] time=9.37, avg_loss=0.8011, train_err=0.0300, 64_h1=0.0572, 64_l2=0.0277
[125] time=9.31, avg_loss=0.7708, train_err=0.0289, 64_h1=0.0551, 64_l2=0.0246
[126] time=9.32, avg_loss=0.7644, train_err=0.0287, 64_h1=0.0566, 64_l2=0.0284
[127] time=9.34, avg_loss=0.8241, train_err=0.0309, 64_h1=0.0564, 64_l2=0.0255
[128] time=9.32, avg_loss=0.7861, train_err=0.0295, 64_h1=0.0600, 64_l2=0.0339
[129] time=9.32, avg_loss=1.0524, train_err=0.0395, 64_h1=0.0578, 64_l2=0.0289
[130] time=9.32, avg_loss=0.7922, train_err=0.0297, 64_h1=0.0561, 64_l2=0.0261
[131] time=9.34, avg_loss=0.7597, train_err=0.0285, 64_h1=0.0555, 64_l2=0.0250
[132] time=9.40, avg_loss=0.7379, train_err=0.0277, 64_h1=0.0560, 64_l2=0.0256
[133] time=9.38, avg_loss=0.7372, train_err=0.0276, 64_h1=0.0563, 64_l2=0.0278
[134] time=9.36, avg_loss=0.7763, train_err=0.0291, 64_h1=0.0553, 64_l2=0.0253
[135] time=9.39, avg_loss=0.7192, train_err=0.0270, 64_h1=0.0552, 64_l2=0.0246
[136] time=9.40, avg_loss=0.7622, train_err=0.0286, 64_h1=0.0546, 64_l2=0.0238
[137] time=9.39, avg_loss=0.7320, train_err=0.0275, 64_h1=0.0560, 64_l2=0.0268
[138] time=9.38, avg_loss=0.7540, train_err=0.0283, 64_h1=0.0549, 64_l2=0.0242
[139] time=9.38, avg_loss=0.7377, train_err=0.0277, 64_h1=0.0581, 64_l2=0.0301
[140] time=9.38, avg_loss=0.7199, train_err=0.0270, 64_h1=0.0548, 64_l2=0.0243
[141] time=9.37, avg_loss=0.7003, train_err=0.0263, 64_h1=0.0551, 64_l2=0.0238
[142] time=9.32, avg_loss=0.7038, train_err=0.0264, 64_h1=0.0560, 64_l2=0.0276
[143] time=9.31, avg_loss=0.7203, train_err=0.0270, 64_h1=0.0558, 64_l2=0.0274
[144] time=9.34, avg_loss=0.7034, train_err=0.0264, 64_h1=0.0550, 64_l2=0.0246
[145] time=9.32, avg_loss=0.6896, train_err=0.0259, 64_h1=0.0548, 64_l2=0.0245
[146] time=9.33, avg_loss=0.6827, train_err=0.0256, 64_h1=0.0550, 64_l2=0.0253
[147] time=9.32, avg_loss=0.6664, train_err=0.0250, 64_h1=0.0551, 64_l2=0.0246
[148] time=9.32, avg_loss=0.6856, train_err=0.0257, 64_h1=0.0559, 64_l2=0.0265
[149] time=9.37, avg_loss=0.7495, train_err=0.0281, 64_h1=0.0572, 64_l2=0.0278
```

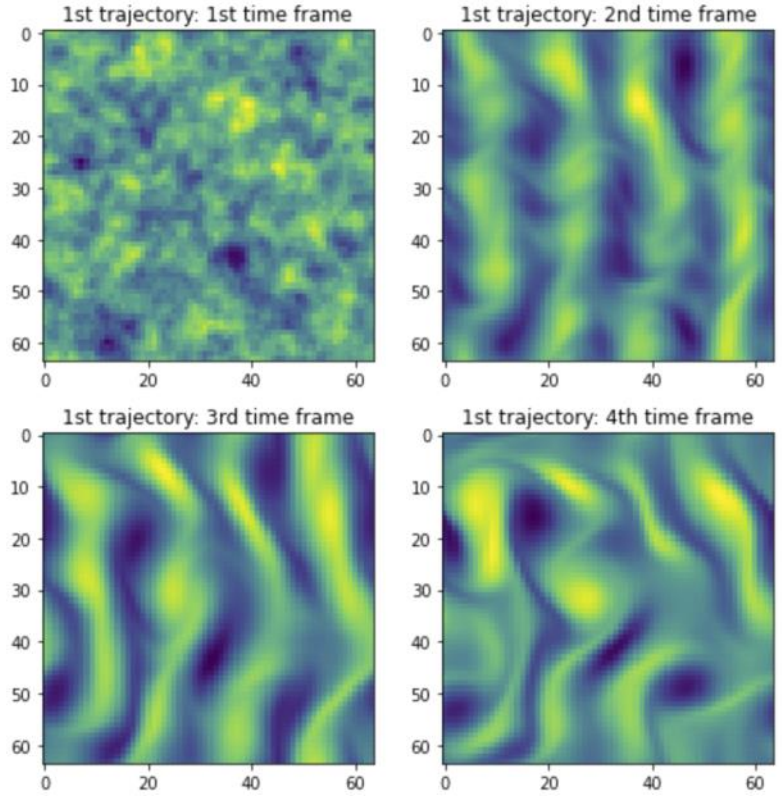
3. Navier-Stokes equation

a. Visualize the dataset

```
data.shape # 50 trajectories, each trajectory has 501 time frames, each time frame is represented as a 64 × 64 image
```

```
(50, 501, 64, 64)
```

```
fig = plt.figure(figsize=(7, 7))
ax = fig.add_subplot(2, 2, 1)
ax.imshow(data[0][0])
ax.set_title('1st trajectory: 1st time frame')
ax = fig.add_subplot(2, 2, 2)
ax.imshow(data[0][1])
ax.set_title('1st trajectory: 2nd time frame')
ax = fig.add_subplot(2, 2, 3)
ax.imshow(data[0][2])
ax.set_title('1st trajectory: 3rd time frame')
ax = fig.add_subplot(2, 2, 4)
ax.imshow(data[0][3])
ax.set_title('1st trajectory: 4th time frame')
plt.tight_layout()
fig.show()
```



b. Prepare the input and output dataset

```
ns_input = data[:, 0:500, :, :] # define the input as [50, 0:500, 64, 64]
ns_output = data[:, 1:501, :, :] # define the output one-frame off [50, 1:501, 64, 64]

x_train = torch.zeros(40*500, 64, 64) # training input
for i in range(40):
    for j in range(500):
        x_train[500*i+j, :, :] = torch.Tensor(ns_input[i, j, :, :])
        # mix the trajectory index and the time index to convert the time series into an image-to-image mapping
x_train = x_train.unsqueeze(1).clone()

y_train = torch.zeros(40*500, 64, 64) # training output
for i in range(40):
    for j in range(500):
        y_train[500*i+j, :, :] = torch.Tensor(ns_output[i, j, :, :])
        # mix the trajectory index and the time index to convert the time series into an image-to-image mapping
y_train = y_train.unsqueeze(1).clone()

x_test = torch.zeros(10*500, 64, 64) # testing input
for i in range(40, 50):
    for j in range(500):
        x_test[500*(i-40)+j, :, :] = torch.Tensor(ns_input[i, j, :, :])
        # mix the trajectory index and the time index to convert the time series into an image-to-image mapping
x_test = x_test.unsqueeze(1).clone()

y_test = torch.zeros(10*500, 64, 64) # testing output
for i in range(40, 50):
    for j in range(500):
        y_test[500*(i-40)+j, :, :] = torch.Tensor(ns_output[i, j, :, :])
        # mix the trajectory index and the time index to convert the time series into an image-to-image mapping
y_test = y_test.unsqueeze(1).clone()
```

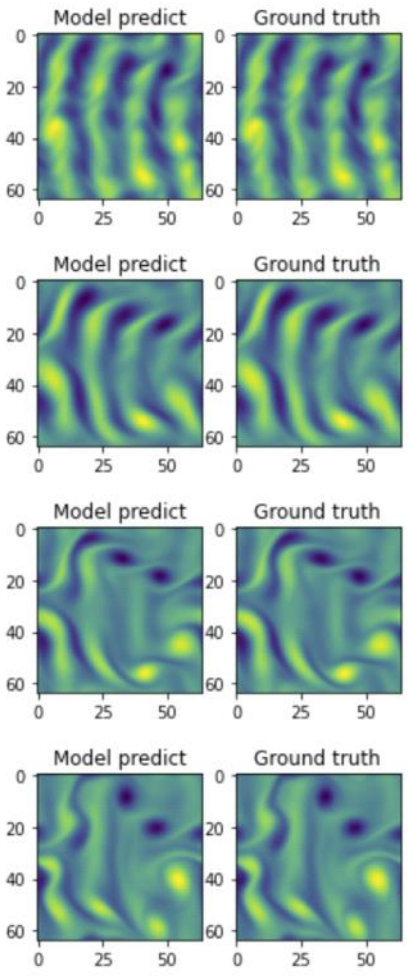
c. Train the model with n_modes_height = n_modes_width = 64, hidden_channels = 64, and n_epochs = 100, training_loss = 'h1'

```
[61] time=44.48, avg_loss=27.3987, train_err=0.1370, 64_h1=0.1652, 64_l2=0.1064
[62] time=44.50, avg_loss=27.7627, train_err=0.1388, 64_h1=0.1684, 64_l2=0.1027
[63] time=44.99, avg_loss=27.8384, train_err=0.1392, 64_h1=0.1462, 64_l2=0.0872
[64] time=44.48, avg_loss=27.3748, train_err=0.1369, 64_h1=0.1681, 64_l2=0.1009
[65] time=44.50, avg_loss=27.5805, train_err=0.1379, 64_h1=0.1540, 64_l2=0.0938
[66] time=44.55, avg_loss=27.4401, train_err=0.1372, 64_h1=0.1525, 64_l2=0.0900
[67] time=44.48, avg_loss=26.9206, train_err=0.1346, 64_h1=0.1401, 64_l2=0.0845
[68] time=44.49, avg_loss=26.6410, train_err=0.1332, 64_h1=0.1408, 64_l2=0.0878
[69] time=44.55, avg_loss=26.6667, train_err=0.1333, 64_h1=0.1340, 64_l2=0.0792
[70] time=44.46, avg_loss=26.5062, train_err=0.1325, 64_h1=0.1380, 64_l2=0.0807
[71] time=44.49, avg_loss=26.2284, train_err=0.1311, 64_h1=0.1488, 64_l2=0.0904
[72] time=44.50, avg_loss=27.2885, train_err=0.1364, 64_h1=0.1539, 64_l2=0.0938
[73] time=44.49, avg_loss=26.4348, train_err=0.1322, 64_h1=0.1400, 64_l2=0.0846
[74] time=44.48, avg_loss=25.9041, train_err=0.1295, 64_h1=0.1344, 64_l2=0.0839
[75] time=44.50, avg_loss=25.6931, train_err=0.1285, 64_h1=0.1377, 64_l2=0.0832
[76] time=44.49, avg_loss=26.1851, train_err=0.1309, 64_h1=0.1495, 64_l2=0.0891
[77] time=44.47, avg_loss=25.7731, train_err=0.1289, 64_h1=0.1348, 64_l2=0.0812
[78] time=44.50, avg_loss=25.6125, train_err=0.1281, 64_h1=0.1294, 64_l2=0.0763
[79] time=44.49, avg_loss=25.3951, train_err=0.1270, 64_h1=0.1362, 64_l2=0.0829
[80] time=44.54, avg_loss=25.6790, train_err=0.1284, 64_h1=0.1391, 64_l2=0.0853
[81] time=44.49, avg_loss=25.2162, train_err=0.1261, 64_h1=0.1366, 64_l2=0.0800
[82] time=44.55, avg_loss=25.3047, train_err=0.1265, 64_h1=0.1318, 64_l2=0.0815
[83] time=44.53, avg_loss=25.4373, train_err=0.1272, 64_h1=0.1378, 64_l2=0.0789
[84] time=44.49, avg_loss=24.9367, train_err=0.1247, 64_h1=0.1269, 64_l2=0.0728
[85] time=44.50, avg_loss=24.9338, train_err=0.1247, 64_h1=0.1395, 64_l2=0.0834
[86] time=44.49, avg_loss=24.9856, train_err=0.1249, 64_h1=0.1302, 64_l2=0.0813
[87] time=44.49, avg_loss=24.6145, train_err=0.1231, 64_h1=0.1367, 64_l2=0.0813
[88] time=44.51, avg_loss=24.2746, train_err=0.1214, 64_h1=0.1309, 64_l2=0.0759
[89] time=44.49, avg_loss=24.3921, train_err=0.1220, 64_h1=0.1368, 64_l2=0.0848
[90] time=44.47, avg_loss=24.3241, train_err=0.1216, 64_h1=0.1347, 64_l2=0.0805
[91] time=44.50, avg_loss=24.1044, train_err=0.1205, 64_h1=0.1417, 64_l2=0.0850
[92] time=44.55, avg_loss=24.5751, train_err=0.1229, 64_h1=0.1250, 64_l2=0.0723
[93] time=44.48, avg_loss=23.8719, train_err=0.1194, 64_h1=0.1387, 64_l2=0.0840
[94] time=44.49, avg_loss=23.7250, train_err=0.1186, 64_h1=0.1388, 64_l2=0.0847
[95] time=44.61, avg_loss=23.6799, train_err=0.1184, 64_h1=0.1336, 64_l2=0.0781
[96] time=44.50, avg_loss=23.3193, train_err=0.1166, 64_h1=0.1214, 64_l2=0.0730
[97] time=44.49, avg_loss=23.7447, train_err=0.1187, 64_h1=0.1262, 64_l2=0.0750
[98] time=44.50, avg_loss=23.3205, train_err=0.1166, 64_h1=0.1238, 64_l2=0.0750
[99] time=44.49, avg_loss=23.5175, train_err=0.1176, 64_h1=0.1315, 64_l2=0.0798
```

d. The error grows bigger over time, and result will diverge from the ground truth, as after each epoch, the tiny error will propagate and get amplified.
We further plot the one-step mapping and recurrent model predictions as a comparison:

If we plot one-step mapping prediction with ground truth, we can see that after 100 epochs of training, the model prediction was quite close to ground truth.

```
%matplotlib inline
import matplotlib.pyplot as plt
for sample in test_loader:
    x, y = sample['x'], sample['y']
    out = model(x.to(device))
    out = output_encoder.decode(out)
    out = out.cpu()
    y = output_encoder.decode(y.to(device))
    y = y.cpu()
    for i in range(4):
        fig = plt.figure(figsize=(4, 4))
        ax = fig.add_subplot(1, 2, 1)
        ax.imshow(out[0+i][0].detach().numpy())
        ax.set_title('Model predict')
        ax = fig.add_subplot(1, 2, 2)
        ax.imshow(y[0+i][0].detach().numpy())
        ax.set_title('Ground truth')
    break
```



Meanwhile, when we plot the recurrent model prediction that iteratively predicts the next time using this time's prediction as input, we can see the model diverge from ground truth, and after two epochs have become quite different from ground truth.

```
%matplotlib inline
import matplotlib.pyplot as plt
for sample in test_loader:
    x, y = sample['x'], sample['y']
    x_recurrent = x[:,-3:]
    for i in range(4):
        out = model(x_recurrent[:,-3:].to(device))
        out = output_encoder.decode(out)
        out = out.cpu()
        y = output_encoder.decode(y.to(device))
        y = y.cpu()
        fig = plt.figure(figsize=(4, 4))
        ax = fig.add_subplot(1, 2, 1)
        ax.imshow(out[0][0].detach().numpy())
        ax.set_title('Model predict')
        ax = fig.add_subplot(1, 2, 2)
        ax.imshow(y[0+i][0].detach().numpy())
        ax.set_title('Ground truth')
        x_recurrent = torch.cat((x_recurrent, out), 1)
    break
```