Export_models

December 12, 2022

Contents

[6]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 300
import random
import csv
import h5py
import pandas as pd
import torch
import torchvision
from torch import nn
from torch.utils.data import Dataset, DataLoader
import matplotlib.cm as cm
```

1 Export the NNC2P model

This is the model architecture:

[2]:
```
# Define hyperparameters of the model here. Will first of all put two hidden␣
 ↪layers
# total of 800 neurons for the one in the paper
device = "cpu"
size_HL_1 = 600
size_HL_2 = 200

# Implement neural network
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        #self.flatten = nn.Flatten()
        self.stack = nn.Sequential(
            nn.Linear(3, size_HL_1),
            nn.Sigmoid(),
            nn.Linear(size_HL_1, size_HL_2),
            nn.Sigmoid(),
            nn.Linear(size_HL_2, 1)
        )

    def forward(self, x):
        # No flatten needed, as our input and output are 1D?
        #x = self.flatten(x)
        logits = self.stack(x)
        return logits
```

We import NNC2Pv0, which was on par with the models in the paper.

[3]:
```
NNC2P = torch.load('Models/NNC2Pv0.pth')
model = NNC2P
```

Look at the parameter values:

[7]:
```
with torch.no_grad():
    for param in NNC2P.parameters():
        print(param)
```

We follow this guide from the PyTorch documentation.

1.1 Get the parameters as matrices

[10]:
```
# torch.save(NNC2P.state_dict(), "NNC2Pv0_params.h5")
```

[11]:
```
NNC2P.state_dict()
```

[11]:
```
OrderedDict([('stack.0.weight',
              tensor([[-0.3637,  0.4540, -0.4355],
                      [ 0.0066,  0.6949,  0.4879],
                      [ 0.1112, -0.0925,  0.1091],
                      …,
                      [ 0.5306, -0.4535, -0.3026],
                      [-0.4308, -0.1415,  0.2810],
                      [ 0.6349, -0.2947,  0.0561]])),
             ('stack.0.bias',
              tensor([ 0.5675,  0.2904, -0.7667, -0.3078, -0.1945,  0.0523,
0.0514, -0.4138,
                       0.2312, -0.5222,  0.2495, -0.3197, -0.4844, -0.5024,
-0.3668, -0.2699,
                       0.7860,  0.7489,  0.1024,  0.8798,  0.1536, -0.4353,
-0.3389, -0.5969,
                    -0.4334, -0.7355, -0.4756, -0.4140, -0.1220, -0.1788,
-0.7250, -0.0075,
                     0.2842,  0.1193,  0.5405, -0.1805, -0.0228, -0.3408,
-0.1134, -0.2822,
                     0.5498, -0.1406,  0.3311, -0.5858,  0.0567, -0.2661,
0.3879,  0.8417,
                    -0.2426,  0.5311,  0.0035,  0.1361, -0.3355,  0.2191,
-0.3657,  0.0739,
                    -0.7668, -0.7611, -0.4528,  0.7155,  0.4711,  0.1546,
-0.7966, -0.6006,
                     0.5338, -0.4438, -0.5507,  0.2647, -0.5531, -0.1843,
0.6857, -0.1058,
                    -0.2366,  0.5566, -0.2539, -0.0841, -0.2701,  0.1520,
-0.3656, -0.0887,
                    -0.3681, -0.4994,  0.1562,  0.0979, -0.1539, -0.2539,
-0.3159,  0.2476,
                     0.1437,  0.1037, -0.6092, -0.4861,  0.6079, -0.1717,
0.3969, -0.8278,
                    -0.7750,  0.4500,  0.1029,  0.0236,  0.3942, -0.0011,
-0.5502, -0.6392,
                    -0.1455, -0.5056, -0.4315, -0.6536, -0.8086,  0.8507,
-0.4151, -0.7212,
                    -0.0891,  0.1468, -0.0913,  0.1593, -0.3147, -0.7297,
0.2530, -0.1589,
                    -0.1999,  0.4665, -0.5153, -0.6170, -0.3868,  0.0854,
0.5496,  0.1570,
                    -0.5972,  0.1290, -0.2804, -0.1617, -0.4747, -0.1994,
0.1695, -0.2299,
                     0.5255, -0.7798,  0.7290, -0.1372, -0.0409,  0.4159,
0.2687, -0.6314,
                     0.1840, -0.6343, -0.7727,  0.0432,  0.1978,  0.0018,
-0.2912,  0.5889,
                     0.1239, -0.5980, -0.3289, -0.4699,  0.1432,  0.6450,
-0.4566,  0.6617,
                    -0.5549, -0.7374,  0.2306,  0.9800,  0.1920,  0.5020,
0.2284,  0.2587,
                     0.6900,  0.2306,  0.7923, -0.1113,  0.2198,  0.6304,
0.3187,  0.0511,
                    -0.5725, -0.6510, -0.7051, -0.3080, -0.2263, -0.5543,
-0.2684, -0.2800,
                    -0.5838,  0.6659, -0.0447, -0.3244, -0.2777,  0.1524,
0.8192, -0.0718,
                    -0.1331,  0.0362, -0.3517,  0.2572,  0.0893, -0.3430,
-0.6010, -0.2209,
                     0.4120, -0.5042,  0.1973,  0.0020,  0.2477,  0.2700,
-0.6794, -0.2675,
                    -0.3750,  0.3425, -0.0609, -0.0658,  0.3587, -0.2422,
0.3080, -0.7774,
                     -0.0425, -0.1093, -0.6006, -0.4135,  0.0222,  0.0549,
0.4497,  0.2517,
                     -0.0629,  0.4377,  0.3117,  0.3804, -0.8146, -0.1727,
-0.0757, -0.4479,
                     -0.3724,  0.2646,  0.2722,  0.2111, -0.4963, -0.7829,
-0.1263,  0.1045,
                     -0.4437, -0.4764,  0.0316, -0.6644,  0.0834,  0.5011,
0.3411,  0.3595,
                     -0.5658, -0.4027, -0.5273,  0.2064, -0.2696,  0.1704,
-0.7847, -0.4299,
                     -0.5457, -0.2170,  0.5040,  0.1638, -0.2259, -0.1841,
0.3940, -0.1587,
                     -0.0681, -0.5532,  0.0486, -0.0708, -0.0685, -0.1967,
-0.6578, -0.0085,
                     -0.5584,  0.3869, -0.3360,  0.0781, -0.4732, -0.4988,
0.5257, -0.0463,
                     -0.5861,  0.0443,  0.3502, -0.3827, -0.0767, -0.4918,
-0.0975,  0.0335,
                      0.0242, -0.1530,  0.2708,  0.3870, -0.0407, -0.7733,
-0.3965,  0.7103,
                     -0.5266, -0.8473, -0.2814,  0.0634, -0.0469,  0.2093,
-0.5929,  0.3147,
                      0.7441, -0.2883, -0.4244, -0.4688,  0.7391, -0.2475,
-0.2986, -0.7846,
                     -0.5749,  0.6449,  0.5729,  0.0330, -0.7806, -0.3968,
-0.1973,  0.8683,
                      0.2063,  0.0795, -0.2172, -0.3743, -0.1792,  0.0273,
-0.2719, -0.1724,
                      0.5487, -0.2173, -0.5166, -0.8283, -0.5187, -0.2308,
0.0458, -0.0205,
                     -0.0467, -0.6538, -0.0829,  0.0589,  0.0573,  0.3710,
0.1821, -0.6651,
                      0.0139,  0.1801, -0.3490, -0.5684,  0.5960,  0.6916,
-0.5211, -0.0705,
                     -0.0245,  0.5548, -0.4998,  0.1310,  0.0123,  0.1382,
0.5340, -0.1300,
                      0.0042,  0.0777, -0.8929, -0.2648,  0.1318,  0.1760,
0.0599, -0.4066,
                      0.3279,  0.2792, -0.3842,  0.1425, -0.0647, -0.6798,
-0.9598,  0.3412,
                     -0.4429, -0.3725,  0.2720,  0.5411,  0.0429, -0.7045,
0.4488,  0.2515,
                      0.4915, -0.2986, -0.0725,  0.8208,  0.0345, -0.4975,
0.2115, -0.3730,
                     -0.1543,  0.4633, -0.7425,  0.3975, -0.1460, -0.0902,
0.5782, -0.5746,
                      -0.0736, -0.8905, -0.1959,  0.3797,  0.6835,  0.4984,
-0.0769,  0.2039,
                       0.5143, -0.4893, -0.5451, -0.5868,  0.8137,  0.5941,
0.1640,  0.2265,
                      -0.6311,  0.3958, -0.2065, -0.4971, -0.0210, -0.3891,
-0.2294, -0.3468,
                       0.7438, -0.1030,  0.7179, -0.7436, -0.5150,  0.0701,
-0.2541,  0.5022,
                      -0.7572,  0.0990,  0.1417,  0.1436,  0.0180,  0.0168,
-0.4819,  0.8244,
                      -0.0125, -0.1109, -0.6625,  0.7918, -0.4478, -0.2006,
0.1864, -0.3666,
                       0.2405,  0.2242, -0.0725, -0.1479, -0.2050,  0.4549,
0.2757, -0.2656,
                       0.5447,  0.2885,  0.0163, -0.5062, -0.3655, -0.4252,
-0.2810, -0.6262,
                       0.4720, -0.5443, -0.2816,  0.7436,  0.7959, -0.2127,
0.6045,  0.2159,
                       0.0723,  0.8628,  0.0749,  0.1937, -0.5478, -0.1131,
0.3797,  0.4071,
                      -0.5809, -0.6407, -0.6400, -0.3935, -0.7474, -0.2790,
0.1554,  0.1401,
                       0.4752, -0.2307, -0.5861,  0.6426, -0.3433, -0.5701,
0.1752,  0.4724,
                      -0.3654,  0.4743,  0.5474,  0.2260, -0.3306, -0.1384,
0.3962, -0.3417,
                      -1.0276, -0.4299,  0.2657,  0.1818,  0.3824,  0.1642,
-0.1071, -0.1129,
                       0.1338,  0.3750, -0.0246,  0.2682,  0.6734, -0.4917,
-0.8268,  0.1484,
                      -0.6909, -0.3862,  0.1191,  0.2251,  0.4636, -0.0899,
0.5847, -0.5227,
                       0.0309, -0.1919, -0.4084,  0.0564,  0.2178,  0.1525,
-0.4559, -0.0342,
                      -0.1900, -0.2373, -0.0560,  0.3202, -0.2350, -0.1091,
-0.2436, -0.0595,
                      -0.0075,  0.0434,  0.4786,  0.4589, -0.7814, -0.4575,
-0.1438,  0.7816,
                       0.6213, -0.3059, -0.0335,  0.5486, -0.8782, -0.7016,
0.6680, -0.4792,
                       0.2301,  0.0706, -0.1901, -0.2882, -0.1218,  0.3371,
-0.1424, -0.5664,
                      -0.3493,  0.2683, -0.4209, -0.1263,  0.1663,  0.3661,
0.0221, -0.0802,
                       0.8377, -0.8028,  0.1312,  0.5930,  0.0925,  0.5772,
-0.3172, -0.2318,
                       0.3839, -0.3587, -0.1506, -0.2225, -0.3813,  0.3004,
0.5387, -0.0993,
                        0.1397, -0.2269, -0.4488,  0.6487, -0.3429,  0.7323,
-0.6757, -0.1690])),
              ('stack.2.weight',
               tensor([[-0.2088,  0.0383,  0.0544,  …, -0.1030,  0.0783,
-0.0014],
                       [ 0.0470, -0.0564, -0.0553,  …,  0.0203, -0.1165,
-0.0557],
                       [-0.0048, -0.0284, -0.0800,  …, -0.0789, -0.0413,
-0.0859],
                       …,
                       [ 0.0504, -0.0565, -0.0705,  …,  0.0052, -0.0812,
-0.0857],
                       [ 0.0676, -0.0898, -0.0730,  …,  0.0324, -0.0613,
-0.0054],
                       [-0.1756,  0.0118,  0.0710,  …, -0.0210,  0.0434,
0.0014]])),
              ('stack.2.bias',
               tensor([-0.0270, -0.0241, -0.0302, -0.0111, -0.0253,  0.0092,
-0.0516, -0.0816,
                        0.0336, -0.0337, -0.0481,  0.0434, -0.0189,  0.0027,
-0.0539, -0.0060,
                       -0.0646, -0.0440, -0.0354, -0.0596, -0.0734, -0.0540,
-0.0820, -0.0217,
                       -0.0141, -0.0055, -0.0674,  0.0044, -0.0344, -0.0741,
0.0176, -0.0616,
                       -0.0446, -0.0020, -0.0306, -0.0233, -0.0305, -0.0373,
-0.0475, -0.0744,
                        0.0541, -0.0632, -0.0144, -0.0232, -0.0255,  0.0226,
-0.0348, -0.0434,
                       -0.0581,  0.0095, -0.0401, -0.0386, -0.0368,  0.0169,
0.0336, -0.0220,
                        0.0518, -0.0205,  0.0081, -0.0749, -0.0333, -0.0069,
-0.0173,  0.0392,
                        0.0175, -0.0278,  0.0328,  0.0343, -0.0011, -0.0501,
-0.0517, -0.0325,
                       -0.0284, -0.0531,  0.0279, -0.0292,  0.0079, -0.0678,
-0.0238, -0.0258,
                       -0.0790,  0.0158, -0.0643,  0.0079, -0.0183,  0.0297,
0.0061,  0.0364,
                       -0.0228, -0.0035,  0.0068, -0.0856, -0.0804,  0.0039,
-0.0382, -0.0563,
                       -0.0724,  0.0061, -0.0240, -0.0852, -0.0255, -0.0267,
0.0112, -0.0661,
                       -0.0289, -0.0278, -0.0946,  0.0428, -0.0398, -0.0250,
-0.0035,  0.0147,
                       -0.0032, -0.0094, -0.0720, -0.0195, -0.0702, -0.0429,
0.0336, -0.0590,
                          0.0109,  0.0078, -0.0717, -0.0769, -0.0308, -0.0152,
0.0234,  0.0287,
                         -0.0626, -0.0299,  0.0259,  0.0166, -0.0485, -0.0169,
-0.0319,  0.0128,
                          0.0126,  0.0150, -0.0164,  0.0116, -0.0582,  0.0241,
-0.0376,  0.0374,
                         -0.0056, -0.0238, -0.0540,  0.0336, -0.0016, -0.0473,
-0.0338, -0.0415,
                         -0.0025, -0.0549,  0.0414, -0.0718, -0.0048, -0.0709,
0.0092, -0.0428,
                         -0.0446, -0.0539,  0.0246, -0.0199, -0.0679, -0.0330,
-0.0509, -0.0346,
                         -0.0404,  0.0587,  0.0257,  0.0199,  0.0176,  0.0247,
-0.0360,  0.0113,
                         -0.0526,  0.0746, -0.0126,  0.0148, -0.0180,  0.0308,
-0.0730,  0.0025,
                         -0.0178, -0.0758,  0.0204, -0.0438,  0.0013,  0.0851,
-0.0482, -0.0559,
                         -0.0076, -0.0415,  0.0245,  0.0066,  0.0124, -0.0645,
-0.0227, -0.0411])),
             ('stack.4.weight',
              tensor([[ 0.1627, -0.0709,  0.0123,  0.3133, -0.0664,  0.1463,
0.1598, -0.0656,
                           0.1418,  0.1590, -0.0636,  0.1596, -0.0613,  0.1870,
0.2146,  0.1589,
                          -0.0531,  0.1655,  0.2035, -0.0812,  0.2130, -0.0660,
-0.0634,  0.1614,
                          -0.0514,  0.1856, -0.0662, -0.0870, -0.0564, -0.0592,
-0.0726,  0.2121,
                          -0.0452, -0.0451, -0.0661,  0.1536, -0.0549, -0.0531,
-0.0767, -0.0604,
                           0.0218,  0.1820, -0.0688, -0.0404,  0.1659,  0.1630,
0.1711,  0.1666,
                          -0.0629,  0.1867,  0.1757, -0.0653, -0.0009, -0.0835,
0.1746,  0.1658,
                           0.1263,  0.1652,  0.1826,  0.2197, -0.0461, -0.0854,
0.1609,  0.1759,
                           0.1829,  0.1776,  0.1510,  0.1660,  0.1700,  0.2523,
-0.0616,  0.1798,
                           0.2320,  0.1671,  0.1840, -0.0574,  0.1818,  0.1700,
-0.0544, -0.0604,
                          -0.0597,  0.1765,  0.1576,  0.1636,  0.1670,  0.1743,
0.1496,  0.1805,
                          -0.0265, -0.0614,  0.1608, -0.0620, -0.0661, -0.0646,
-0.0664, -0.0510,
                          -0.0638,  0.1882,  0.1827, -0.0636,  0.1784,  0.1607,
0.1750, -0.0586,
                        0.2005,  0.1738, -0.0557,  0.1631, -0.0659, -0.0593,
0.1661,  0.2004,
                        0.1617, -0.0612, -0.0625, -0.0604, -0.0643,  0.1516,
0.1590, -0.0466,
                        0.1747, -0.0753,  0.2033, -0.0531, -0.0683, -0.0731,
0.1523,  0.1447,
                        0.1618, -0.0730,  0.1459,  0.1990, -0.0332, -0.0622,
-0.0709,  0.1701,
                        0.1877, -0.0721, -0.0712,  0.1784,  0.1617, -0.0757,
0.1658,  0.1736,
                        0.1609, -0.0745,  0.0118,  0.1497,  0.2044,  0.1691,
0.1669,  0.1960,
                       -0.0650,  0.1632,  0.1851, -0.0569,  0.1556,  0.1992,
0.1768, -0.0513,
                       -0.0658,  0.1782,  0.2142,  0.1588, -0.0646,  0.1558,
0.1586, -0.0601,
                       -0.0669,  0.0652,  0.1553, -0.0779,  0.1668,  0.1606,
-0.0585,  0.1972,
                       -0.0570,  0.2067,  0.1678,  0.1546, -0.0296,  0.1642,
-0.0804, -0.0662,
                       -0.0473, -0.0452,  0.1472, -0.0654,  0.1412,  0.1047,
-0.0688, -0.0658,
                       -0.0858,  0.2038,  0.1444,  0.1963,  0.1914, -0.0606,
-0.0622,  0.1813]])),
             ('stack.4.bias', tensor([0.1309]))])
```

1.1.1 First, make it easy: save the matrices as a CSV

[100]:
```python
param_names = NNC2P.state_dict().keys()
file_names = ["Models/NNC2Pv0_params_" + key + ".csv" for key in param_names]


for i, key in enumerate(param_names):
    # Get the key and name of the current parameter matrix that we are saving
    # key = param_names[i]
    name = file_names[i]
    print(name)
    # Get the value of these parameters:
    matrix = NNC2P.state_dict()[key]
    # Convert to Numpy array
    matrix_np = matrix.numpy()
    # Convert to a dataframe
    df = pd.DataFrame(matrix_np)
    # Save to file
    df.to_csv(name,index=False, header=False)
```

```
Models/NNC2Pv0_params_stack.0.weight.csv
```

```
Models/NNC2Pv0_params_stack.0.bias.csv
Models/NNC2Pv0_params_stack.2.weight.csv
Models/NNC2Pv0_params_stack.2.bias.csv
Models/NNC2Pv0_params_stack.4.weight.csv
Models/NNC2Pv0_params_stack.4.bias.csv
```

Read the files:

[101]:
```
weight0 = pd.read_csv("Models/NNC2Pv0_params_stack.0.weight.csv", header=None).
  ↪values
bias0   = pd.read_csv("Models/NNC2Pv0_params_stack.0.bias.csv", header=None).
  ↪values
weight2 = pd.read_csv("Models/NNC2Pv0_params_stack.2.weight.csv", header=None).
  ↪values
bias2   = pd.read_csv("Models/NNC2Pv0_params_stack.2.bias.csv", header=None).
  ↪values
weight4 = pd.read_csv("Models/NNC2Pv0_params_stack.4.weight.csv", header=None).
  ↪values
bias4   = pd.read_csv("Models/NNC2Pv0_params_stack.4.bias.csv", header=None).
  ↪values

weights_and_biases = [weight0, bias0, weight2, bias2, weight4, bias4]
```

[102]:
```
# Print the shape for each parameter:
for i in range(len(weights_and_biases)):
    print("For the file: ", file_names[i])
    # Read the values
    shape = np.shape(weights_and_biases[i])
    print("The shape is equal to ", shape)
```

```
For the file:  Models/NNC2Pv0_params_stack.0.weight.csv
The shape is equal to  (600, 3)
For the file:  Models/NNC2Pv0_params_stack.0.bias.csv
The shape is equal to  (600, 1)
For the file:  Models/NNC2Pv0_params_stack.2.weight.csv
The shape is equal to  (200, 600)
For the file:  Models/NNC2Pv0_params_stack.2.bias.csv
The shape is equal to  (200, 1)
For the file:  Models/NNC2Pv0_params_stack.4.weight.csv
The shape is equal to  (1, 200)
For the file:  Models/NNC2Pv0_params_stack.4.bias.csv
The shape is equal to  (1, 1)
```

[104]:
```
# Read the example file
example = pd.read_csv("Models/NNC2Pv0_params_stack.0.weight.csv", header=None).
  ↪values
print(example)
print(np.shape(example))
```

```
[[-0.36373898  0.45402282 -0.4355268 ]
 [ 0.00657238  0.69492054  0.4879491 ]
 [ 0.11119203 -0.09253014  0.10905483]
 …
 [ 0.5305801  -0.45353398 -0.3026449 ]
 [-0.4308225  -0.14152803  0.28101048]
 [ 0.63488334 -0.29469463  0.05608372]]
(600, 3)
```

1.1.2 How to use the parameters to make a prediction without any Pytorch tools

When we are going to implement this in the Gmunu code, we can no longer use any of the built-in tools of PyTorch.

[53]:
```
## One specific test case for the data
rho,eps,v,p,D,S,tau = 9.83632270803203,1.962038705851822,0.2660655147967911,12.
↪866163917605371,10.204131145455385,12.026584842282125,22.131296926293793
```

This is how the PyTorch implementation works:

[54]:
```
input_test = torch.tensor([D, S, tau])
exact_result = p
print(exact_result)
print(input_test)
with torch.no_grad():
    pred = model(input_test).item()
print(pred)
```

```
12.866163917605371
tensor([10.2041, 12.0266, 22.1313])
12.86711311340332
```

Now, we have to try and get the same output, but by defining all intermediate steps ourselves!

[76]:
```
def sigmoid(x):
    return 1/(1+np.exp(-x))

def compute_prediction(x):
    """Input is a np. array of size 1x3"""
    print(np.shape(x))
    x = np.matmul(weight0, x.T) + bias0
    print(np.shape(x))
    x = sigmoid(x)
    x = np.matmul(weight2, x) + bias2
    print(np.shape(x))
    x = sigmoid(x)
    x = np.matmul(weight4, x) + bias4
    print(np.shape(x))
    return x[0][0]
```

[77]:
```
input_test = np.array([[D, S, tau]])
print(np.shape(input_test))
```

```
(1, 3)
```

[78]:
```
our_prediction = compute_prediction(input_test)
print(our_prediction)
print(pred)
```

```
(1, 3)
(600, 1)
(200, 1)
(1, 1)
12.867113930614748
12.86711311340332
```

Now we compute rho and eps from this (see appendix A of central paper)

[81]:
```
v_star = S/(tau + D + our_prediction)
W_star = 1/np.sqrt(1-v_star**2)

rho_star = D/W_star
eps_star = (tau + D*(1 - W_star) + our_prediction*(1 - W_star**2))/(D*W_star)
print("Our calculations:")
print(rho_star, eps_star)
print("Exact results:")
print(rho, eps)
```

```
Our calculations:
9.836338457223192 1.9620408002397969
Exact results:
9.83632270803203 1.962038705851822
```

1.1.3 Now save it into an hdf5 file

[9]:
```
# # Open an HDF5 file for writing
# with h5py.File("NNC2Pv0_params.h5", "w") as f:
#     # Save the weights and biases of the network to the HDF5 file
#     f.create_dataset("NNC2Pv0_params", data=NNC2P.state_dict())
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [9], in <cell line: 2>()
      1 # Open an HDF5 file for writing
      2 with h5py.File("NNC2Pv0_params.h5", "w") as f:
      3     # Save the weights and biases of the network to the HDF5 file
----> 4     f.create_dataset("NNC2Pv0_params", data=NNC2P.state_dict())
```

```
File D:\Anaconda3\lib\site-packages\h5py\_hl\group.py:149, in Group.
 ↪create_dataset(self, name, shape, dtype, data, **kwds)
    146         parent_path, name = name.rsplit(b'/', 1)
    147         group = self.require_group(parent_path)
--> 149 dsid = dataset.make_new_dset(group, shape, dtype, data, name, **kwds)
    150 dset = dataset.Dataset(dsid)
    151 return dset

File D:\Anaconda3\lib\site-packages\h5py\_hl\dataset.py:91, in␣
 ↪make_new_dset(parent, shape, dtype, data, name, chunks, compression, shuffle,␣
 ↪fletcher32, maxshape, compression_opts, fillvalue, scaleoffset, track_times,␣
 ↪external, track_order, dcpl, allow_unknown_filter)
     89     else:
     90         dtype = numpy.dtype(dtype)
---> 91     tid = h5t.py_create(dtype, logical=1)
     93 # Legacy
     94 if any((compression, shuffle, fletcher32, maxshape, scaleoffset)) and␣
 ↪chunks is False:

File h5py\h5t.pyx:1663, in h5py.h5t.py_create()

File h5py\h5t.pyx:1687, in h5py.h5t.py_create()

File h5py\h5t.pyx:1747, in h5py.h5t.py_create()

TypeError: Object dtype dtype('O') has no native HDF5 equivalent
```

1.2 More advanced: using Torch script

There exist two ways of converting a PyTorch model to Torch Script. The first is known as tracing, a mechanism in which the structure of the model is captured by evaluating it once using example inputs, and recording the flow of those inputs through the model. This is suitable for models that make limited use of control flow. The second approach is to add explicit annotations to your model that inform the Torch Script compiler that it may directly parse and compile your model code, subject to the constraints imposed by the Torch Script language.

[10]:
```
example = torch.tensor([1, 1, 0.5])
example
```

[10]:
```
tensor([1.0000, 1.0000, 0.5000])
```

[12]:
```
traced_script_module = torch.jit.trace(model, example)
traced_script_module
```

[12]:
```
NeuralNetwork(
  original_name=NeuralNetwork
  (stack): Sequential(
    original_name=Sequential
    (0): Linear(original_name=Linear)
    (1): Sigmoid(original_name=Sigmoid)
    (2): Linear(original_name=Linear)
    (3): Sigmoid(original_name=Sigmoid)
    (4): Linear(original_name=Linear)
  )
)
```

[16]:
```
output = traced_script_module(torch.tensor([1,1,0.5]))
output
```

[16]:
```
tensor([0.0595], grad_fn=<AddBackward0>)
```

[17]:
```
traced_script_module.save("traced_NNC2P_model.pt")
```

To do: finish it

[ ]:
```
```