Assignment

April 26, 2023

1 Imports

[ ]:
```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from scipy.io import loadmat
import matplotlib.pyplot as plt
import os

plt.rcParams["figure.figsize"] = (10.0, 7.0)
plt.rcParams["font.size"] = 16
plt.rcParams["font.family"] = "Serif"
plt.rcParams["grid.linestyle"] = "--"
plt.rcParams["grid.linewidth"] = 0.5
```

[ ]:
```
DATA_DIR = 'data'
plots_dir = 'plots'
```

2 Loading Data

We will start by loading the data. We will use `scipy` to load the data. Then we will use `torch` to batch the dataset.

[ ]:
```
data = loadmat(os.path.join(DATA_DIR, 'burgers_data_R10.mat'))
data["a"].shape, data["u"].shape
```

[ ]:
```
((2048, 8192), (2048, 8192))
```

The original dataset has a shape of `(2048, 8192)`. We will subsample the dataset with factor of 8 on the second dimension to make it in a shape `(2048, 1024)`:

[ ]:
```
subsmaple = 2**3
h = 2**13 // subsmaple

x_data = data['a'][:,::subsmaple]
y_data = data['u'][:,::subsmaple]
```

```
x_data= x_data.astype(np.float32)
x_data= torch.from_numpy(x_data)
y_data= y_data.astype(np.float32)
y_data= torch.from_numpy(y_data)

x_data.shape, y_data.shape
```

[ ]: (torch.Size([2048, 1024]), torch.Size([2048, 1024]))

There are 2048 samples in the dataset. However, we will only use first 1000 samples for training and the last 100 samples for testing:

[ ]:
```
train_samples = 1000
test_smaples = 100
x_train = x_data[:train_samples,:]
y_train = y_data[:train_samples,:]
x_test = x_data[-test_smaples:,:]
y_test = y_data[-test_smaples:,:]
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

[ ]:
```
(torch.Size([1000, 1024]),
 torch.Size([1000, 1024]),
 torch.Size([100, 1024]),
 torch.Size([100, 1024]))
```

Finally, we will reshape the datasetnd create a batch dataloader:

[ ]:
```
batch_size = 20

x_train = x_train.reshape(train_samples,h,1)
x_test = x_test.reshape(test_smaples,h,1)

train_loader = torch.utils.data.DataLoader(torch.utils.data.
 ↪TensorDataset(x_train, y_train), batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(torch.utils.data.
 ↪TensorDataset(x_test, y_test), batch_size=batch_size, shuffle=False)
```

Great! Now we have data ready to go!

3 The Model

3.1 The Fourier Block

We will first create a class for one dimensional Fourier block as given in the PDF. Here is a class which implements this:

[ ]:
```
```

```
class SpectralConv1d(nn.Module):
    """The SpectralConv1d class implements a 1D Fourier layer. Does FFT, linear␣
↪transform and then does inverse FFT to\\
        return values in real space."""

    def __init__(self, in_channels, out_channels, modes):
        """ 1D Fourier layer. Does FFT, linear transform and then does inverse␣
↪FFT to\\
        return values in real space.

        Parameters
        ----------
        in_channels : int
            Number of input channels.
        out_channels : int
            Number of output channels.
        modes : int
            Number of Fourier modes to use, at most floor(N/2) + 1 where N is␣
↪the\\
            number of points in the signal.
        """
        super(SpectralConv1d, self).__init__()
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.modes = modes

        self.scale = 1 / (in_channels * out_channels)
        self.weights = nn.Parameter(
            self.scale
            * torch.rand(in_channels, out_channels, self.modes, dtype=torch.
↪cfloat)
        )

    def complex_multiplication(self, input, weights):
        """Does complex multiplication of two tensors for the linear␣
↪transformation.

        Parameters
        ----------
        input : torch.Tensor
            Input tensor of shape (batchsize, in_channels, input_size)
        weights : torch.Tensor
            Weights tensor of shape (in_channels, out_channels, modes)

        Returns
        -------
        torch.Tensor
```

```
            Output tensor of shape (batchsize, out_channels, input_size)
        """
        return torch.einsum("bix,iox->box", input, weights)

    def forward(self, x):
        """The forward pass of the layer.

        Parameters
        ----------
        x : torch.Tensor
            Input tensor of shape (batchsize, in_channels, input_size)

        Returns
        -------
        torch.Tensor
            Output tensor of shape (batchsize, out_channels, input_size)
        """
        batchsize = x.shape[0]
        x_ft = torch.fft.rfft(x)
        output_c_channels = x.size(-1) // 2 + 1
        if self.modes > output_c_channels:
            raise ValueError(
                f"Number of output modes must not be greater than the maximum␣
↪frequencies in input.\
                             \\Max frequency {output_c_channels} < {self.modes}␣
↪modes to choose."
            )
        out_ft = torch.zeros(
            batchsize,
            self.out_channels,
            output_c_channels,
            device=x.device,
            dtype=torch.cfloat,
        )
        out_ft[:, :, : self.modes] = self.complex_multiplication(
            x_ft[:, :, : self.modes], self.weights
        )

        x = torch.fft.irfft(out_ft, n=x.size(-1))
        return x
```

3.2 The FNO Model

Next, we create the model. The network consists of 4 spectral convolution layers defined above, each followed by a linear layer and a `GELU` activation function. The final layer is a linear layer with a single output.

[ ]:
```
class FNO1D(nn.Module):
    """FNO network in 1D
    The network consists of 4 spectral convolution layers, each followed by a␣
 ↪linear layer and a GELU activation\\
        function. The final layer is a linear layer with a single output.
    """

    def __init__(self, modes, width):
        """Initializes the FNO1D Class

        The network consists of 4 spectral convolution layers, each followed by␣
 ↪a linear layer and a GELU activation\\
        function. The final layer is a linear layer with a single output.

        Parameters
        ----------
        modes : int
            Number of Fourier modes to use, at most floor(N/2) + 1 where N is␣
 ↪the\\
            number of points in the signal.
        width : int
            Width of the network.
        """
        super(FNO1D, self).__init__()
        self.modes = modes
        self.width = width

        self.fc0 = nn.Linear(2, self.width)

        self.conv0 = SpectralConv1d(self.width, self.width, self.modes)
        self.conv1 = SpectralConv1d(self.width, self.width, self.modes)
        self.conv2 = SpectralConv1d(self.width, self.width, self.modes)
        self.conv3 = SpectralConv1d(self.width, self.width, self.modes)

        self.w0 = nn.Conv1d(self.width, self.width, 1)
        self.w1 = nn.Conv1d(self.width, self.width, 1)
        self.w2 = nn.Conv1d(self.width, self.width, 1)
        self.w3 = nn.Conv1d(self.width, self.width, 1)

        self.fc1 = nn.Linear(self.width, 128)
        self.fc2 = nn.Linear(128, 1)

    def forward(self, x):
        """Does a forward pass of the network.

        Parameters
        ----------
```

```
        x : torch.Tensor
            Input tensor of shape (batchsize, input_size, 2)

        Returns
        -------
        torch.Tensor
            Output tensor of shape (batchsize, input_size, 1)
        """
        grid = self.get_grid(x.shape, x.device)
        x = torch.cat((x, grid), dim=-1)
        x = self.fc0(x)
        x = x.permute(0, 2, 1)

        x1 = self.conv0(x)
        x2 = self.w0(x)
        x = x1 + x2
        x = F.gelu(x)

        x1 = self.conv1(x)
        x2 = self.w1(x)
        x = x1 + x2
        x = F.gelu(x)

        x1 = self.conv2(x)
        x2 = self.w2(x)
        x = x1 + x2
        x = F.gelu(x)

        x1 = self.conv3(x)
        x2 = self.w3(x)
        x = x1 + x2

        x = x.permute(0, 2, 1)
        x = self.fc1(x)
        x = F.gelu(x)
        x = self.fc2(x)
        return x

    def get_grid(self, shape, device):
        """Creates a grid of points in real space.

        Parameters
        ----------
        shape : tuple
            Shape of the input tensor.
        device : torch.device
            Device to use for the grid.
```

```
    Returns
    -------
    torch.Tensor
        Grid tensor of shape (batchsize, input_size, 1)
    """
    batchsize, size_x = shape[0], shape[1]
    gridx = torch.tensor(np.linspace(0, 1, size_x), dtype=torch.float)
    gridx = gridx.reshape(1, size_x, 1).repeat([batchsize, 1, 1])
    return gridx.to(device)
```

3.3 The LP Loss and Training the Model

Finally, we define the `LpLoss` object which will be used for optimization:

[ ]:
```
class LpLoss:
    """The Lp loss function."""
    def __init__(self, p=2):
        """Instantiates the LpLoss class.

        Parameters
        ----------
        p : int, optional
            The p in Lp norm. Defaults to 2.

        Raises
        ------
        ValueError
            If p < 1.
        """
        if p<1:
            raise ValueError(f"p must be greater than 1. Recieved {p}")
        self.p = p

    def call(self, x, y):
        num_examples = x.size()[0]

        diff_norms = torch.norm(x - y, dim=1, p=self.p)
        y_norms = torch.norm(y, dim=1, p=self.p)

        return torch.sum(diff_norms/y_norms)/num_examples

    def __call__(self, x, y):
        return self.call(x, y)
```

Let's see how the FNO architecture looks like. We will also calculate the number of parameters in the model:

[ ]:
```
def multiply(*args):
    m = 1
    for i in args:
        m*=i
    return m

def count_params(model):
    parameters = 0
    for p in model.parameters():
        if p.is_complex():
            parameters += 2*multiply(*list(p.size()))
        else:
            parameters+=multiply(*list(p.size()))
    return parameters

modes = 16
width = 64
model = FNO1D(modes, width)
print(model)
print("Total Number of Parameters: ", count_params(model))
```

```
FNO1D(
  (fc0): Linear(in_features=2, out_features=64, bias=True)
  (conv0): SpectralConv1d()
  (conv1): SpectralConv1d()
  (conv2): SpectralConv1d()
  (conv3): SpectralConv1d()
  (w0): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
  (w1): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
  (w2): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
  (w3): Conv1d(64, 64, kernel_size=(1,), stride=(1,))
  (fc1): Linear(in_features=64, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=1, bias=True)
)
Total Number of Parameters:  549569
```

[ ]:
```
epochs = 50
learning_rate = 0.001
step_size = train_samples//batch_size
gamma = 0.97

optimizer = torch.optim.Adam(params=model.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=step_size,␣
 ↪gamma=gamma)
train_mse_losses=[]
train_l2_losses=[]
test_mse_losses=[]
```

```python
test_l2_losses=[]

for ep in range(epochs):
    model.train()
    train_mse = 0
    train_l2 = 0
    j = 0
    for x, y in train_loader:
        j+=1
        optimizer.zero_grad()
        out = model(x)

        y = torch.unsqueeze(y, -1)
        mse = torch.mean((y-out)**2)
        l2 =  LpLoss(2)(y, out)
        l2.backward()

        optimizer.step()
        scheduler.step()
        for param_group in optimizer.param_groups:
            new_lr = param_group['lr']
            if new_lr != learning_rate:
                learning_rate = new_lr
                print("Changing Learning rate to: {}...".
↪format(param_group['lr']))
        train_mse += mse.item()
        train_l2 += l2.item()
        print(f"At Batch {j:>3d}/{step_size}: MSE Loss {(train_mse/j):.6f} ||␣
↪L2 Loss {(train_l2/j):.6f}", end="\r")
    model.eval()
    test_l2 = 0.0
    test_mse = 0.0
    with torch.no_grad():
        for x, y in test_loader:
            y = torch.unsqueeze(y, -1)
            out = model(x)
            test_mse += torch.mean((y-out)**2).item()
            test_l2 +=  LpLoss(2)(y, out).item()

    train_mse /= len(train_loader)
    train_l2/=len(train_loader)
    test_l2 /= len(test_loader)
    test_mse /=len(test_loader)
    train_l2_losses.append(train_l2)
    train_mse_losses.append(train_mse)
    test_l2_losses.append(test_l2)
    test_mse_losses.append(test_mse)
```

```
    print(f"Epoch {ep+1:>3d}/{epochs} || Train MSE {train_mse:6f} || Train L2␣
↪{train_l2:6f} || Test MSE {test_mse:6f} || Test L2 {test_l2:6f}")
```

```
Changing Learning rate to: 0.0009699999999999999…3918
Epoch   1/50 || Train MSE 26.675634 || Train L2 1.169040 || Test MSE 18.085400
|| Test L2 0.915313
Changing Learning rate to: 0.0009408999999999999…4972
Epoch   2/50 || Train MSE 1.546068 || Train L2 0.317364 || Test MSE 0.001131 ||
Test L2 0.060289
Changing Learning rate to: 0.0009126729999999999…756
Epoch   3/50 || Train MSE 0.000852 || Train L2 0.052458 || Test MSE 0.000490 ||
Test L2 0.029504
Changing Learning rate to: 0.0008852928099999999…169
Epoch   4/50 || Train MSE 0.000324 || Train L2 0.031170 || Test MSE 0.000350 ||
Test L2 0.034650
Changing Learning rate to: 0.0008587340256999998…613
Epoch   5/50 || Train MSE 0.000223 || Train L2 0.026663 || Test MSE 0.000186 ||
Test L2 0.026360
Changing Learning rate to: 0.0008329720049289999…405
Epoch   6/50 || Train MSE 0.000169 || Train L2 0.025349 || Test MSE 0.000237 ||
Test L2 0.027411
Changing Learning rate to: 0.0008079828447811299…578
Epoch   7/50 || Train MSE 0.000118 || Train L2 0.019559 || Test MSE 0.000111 ||
Test L2 0.022207
Changing Learning rate to: 0.0007837433594376959…638
Epoch   8/50 || Train MSE 0.000178 || Train L2 0.026633 || Test MSE 0.000105 ||
Test L2 0.019207
Changing Learning rate to: 0.0007602310586545650…8346
Epoch   9/50 || Train MSE 0.000083 || Train L2 0.018233 || Test MSE 0.000116 ||
Test L2 0.020253
Changing Learning rate to: 0.0007374241268949280…0223
Epoch  10/50 || Train MSE 0.000108 || Train L2 0.020199 || Test MSE 0.000040 ||
Test L2 0.012186
Changing Learning rate to: 0.0007153014030880802…374
Epoch  11/50 || Train MSE 0.000046 || Train L2 0.012359 || Test MSE 0.000100 ||
Test L2 0.017539
Changing Learning rate to: 0.0006938423609954377…984
Epoch  12/50 || Train MSE 0.000112 || Train L2 0.020987 || Test MSE 0.000232 ||
Test L2 0.025177
Changing Learning rate to: 0.0006730270901655745…790
Epoch  13/50 || Train MSE 0.000079 || Train L2 0.016933 || Test MSE 0.000196 ||
Test L2 0.027204
Changing Learning rate to: 0.0006528362774606073…261
Epoch  14/50 || Train MSE 0.000039 || Train L2 0.012281 || Test MSE 0.000050 ||
Test L2 0.013618
Changing Learning rate to: 0.0006332511891367891…511
```

```
Epoch  15/50 || Train MSE 0.000066 || Train L2 0.015422 || Test MSE 0.000045 ||
Test L2 0.013402
Changing Learning rate to: 0.0006142536534626854…528
Epoch  16/50 || Train MSE 0.000045 || Train L2 0.013503 || Test MSE 0.000027 ||
Test L2 0.009894
Changing Learning rate to: 0.0005958260438588048…674
Epoch  17/50 || Train MSE 0.000051 || Train L2 0.013949 || Test MSE 0.000251 ||
Test L2 0.031312
Changing Learning rate to: 0.0005779512625430406…174
Epoch  18/50 || Train MSE 0.000069 || Train L2 0.016039 || Test MSE 0.000023 ||
Test L2 0.009496
Changing Learning rate to: 0.0005606127246667494…394
Epoch  19/50 || Train MSE 0.000031 || Train L2 0.011406 || Test MSE 0.000022 ||
Test L2 0.009177
Changing Learning rate to: 0.0005437943429267469…578
Epoch  20/50 || Train MSE 0.000028 || Train L2 0.009540 || Test MSE 0.000038 ||
Test L2 0.011710
Changing Learning rate to: 0.0005274805126389445…242
Epoch  21/50 || Train MSE 0.000054 || Train L2 0.014352 || Test MSE 0.000054 ||
Test L2 0.015480
Changing Learning rate to: 0.0005116560972597762…811
Epoch  22/50 || Train MSE 0.000055 || Train L2 0.014778 || Test MSE 0.000032 ||
Test L2 0.008988
Changing Learning rate to: 0.0004963064143419829…322
Epoch  23/50 || Train MSE 0.000039 || Train L2 0.012270 || Test MSE 0.000050 ||
Test L2 0.013376
Changing Learning rate to: 0.00048141722191172336…62
Epoch  24/50 || Train MSE 0.000046 || Train L2 0.013176 || Test MSE 0.000031 ||
Test L2 0.012068
Changing Learning rate to: 0.00046697470525437166…45
Epoch  25/50 || Train MSE 0.000040 || Train L2 0.012852 || Test MSE 0.000017 ||
Test L2 0.007322
Changing Learning rate to: 0.0004529654640967405…455
Epoch  26/50 || Train MSE 0.000028 || Train L2 0.010437 || Test MSE 0.000033 ||
Test L2 0.010792
Changing Learning rate to: 0.0004393765001738382…096
Epoch  27/50 || Train MSE 0.000034 || Train L2 0.011026 || Test MSE 0.000012 ||
Test L2 0.007136
Changing Learning rate to: 0.00042619520516862307…87
Epoch  28/50 || Train MSE 0.000016 || Train L2 0.007631 || Test MSE 0.000014 ||
Test L2 0.006740
Changing Learning rate to: 0.00041340934901356436…38
Epoch  29/50 || Train MSE 0.000015 || Train L2 0.007381 || Test MSE 0.000012 ||
Test L2 0.006695
Changing Learning rate to: 0.0004010070685431574…737
Epoch  30/50 || Train MSE 0.000020 || Train L2 0.008751 || Test MSE 0.000045 ||
Test L2 0.010953
Changing Learning rate to: 0.0003889768564868627…691
```

```
Epoch  31/50 || Train MSE 0.000017 || Train L2 0.007725 || Test MSE 0.000031 ||
Test L2 0.012178
Changing Learning rate to: 0.0003773075507922568…119
Epoch  32/50 || Train MSE 0.000023 || Train L2 0.009026 || Test MSE 0.000007 ||
Test L2 0.003910
Changing Learning rate to: 0.0003659883242684891…837
Epoch  33/50 || Train MSE 0.000018 || Train L2 0.007867 || Test MSE 0.000021 ||
Test L2 0.009121
Changing Learning rate to: 0.00035500867454043444…12
Epoch  34/50 || Train MSE 0.000010 || Train L2 0.005793 || Test MSE 0.000017 ||
Test L2 0.007135
Changing Learning rate to: 0.0003443584143042214…380
Epoch  35/50 || Train MSE 0.000016 || Train L2 0.007366 || Test MSE 0.000014 ||
Test L2 0.007230
Changing Learning rate to: 0.00033402766187509475…08
Epoch  36/50 || Train MSE 0.000010 || Train L2 0.006124 || Test MSE 0.000026 ||
Test L2 0.009801
Changing Learning rate to: 0.0003240068320188419…250
Epoch  37/50 || Train MSE 0.000016 || Train L2 0.008301 || Test MSE 0.000015 ||
Test L2 0.007079
Changing Learning rate to: 0.00031428662705827666…10
Epoch  38/50 || Train MSE 0.000016 || Train L2 0.007729 || Test MSE 0.000005 ||
Test L2 0.003992
Changing Learning rate to: 0.00030485802824652835…72
Epoch  39/50 || Train MSE 0.000016 || Train L2 0.007495 || Test MSE 0.000006 ||
Test L2 0.004284
Changing Learning rate to: 0.0002957122873991325…647
Epoch  40/50 || Train MSE 0.000014 || Train L2 0.007653 || Test MSE 0.000015 ||
Test L2 0.006926
Changing Learning rate to: 0.00028684091877715853…70
Epoch  41/50 || Train MSE 0.000016 || Train L2 0.007895 || Test MSE 0.000016 ||
Test L2 0.007980
Changing Learning rate to: 0.00027823569121384375…75
Epoch  42/50 || Train MSE 0.000014 || Train L2 0.007388 || Test MSE 0.000009 ||
Test L2 0.005193
Changing Learning rate to: 0.0002698886204774284…518
Epoch  43/50 || Train MSE 0.000012 || Train L2 0.006489 || Test MSE 0.000006 ||
Test L2 0.003771
Changing Learning rate to: 0.00026179196186310554…61
Epoch  44/50 || Train MSE 0.000009 || Train L2 0.005542 || Test MSE 0.000019 ||
Test L2 0.007287
Changing Learning rate to: 0.0002539382030072124…545
Epoch  45/50 || Train MSE 0.000008 || Train L2 0.005539 || Test MSE 0.000008 ||
Test L2 0.004803
Changing Learning rate to: 0.000246320056916996…5302
Epoch  46/50 || Train MSE 0.000007 || Train L2 0.005313 || Test MSE 0.000017 ||
Test L2 0.006650
Changing Learning rate to: 0.0002389304552094861…425
```

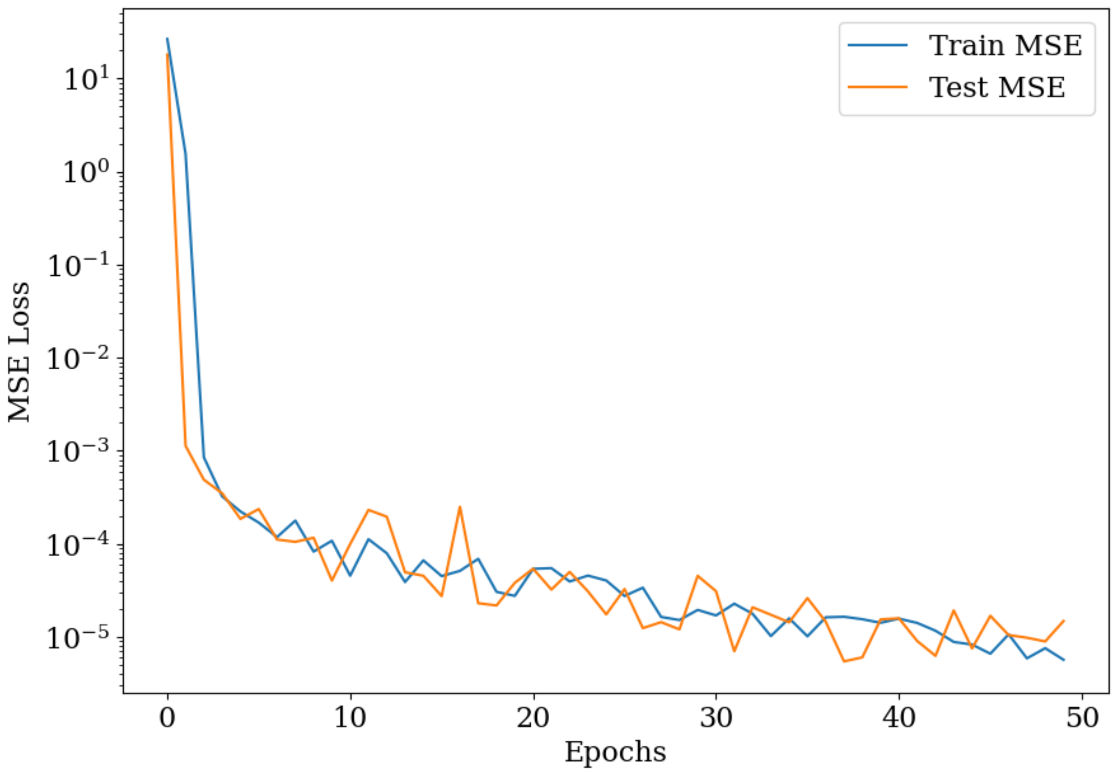
```
Epoch  47/50 || Train MSE 0.000011 || Train L2 0.006370 || Test MSE 0.000010 ||
Test L2 0.006454
Changing Learning rate to: 0.00023176254155320154…450
Epoch  48/50 || Train MSE 0.000006 || Train L2 0.004475 || Test MSE 0.000010 ||
Test L2 0.004950
Changing Learning rate to: 0.00022480966530660546…28
Epoch  49/50 || Train MSE 0.000008 || Train L2 0.005159 || Test MSE 0.000009 ||
Test L2 0.005330
Changing Learning rate to: 0.00021806537534740736…71
Epoch  50/50 || Train MSE 0.000006 || Train L2 0.004650 || Test MSE 0.000015 ||
Test L2 0.007091
```

[ ]:
```
torch.save(model.state_dict(), 'fno.pt')
```

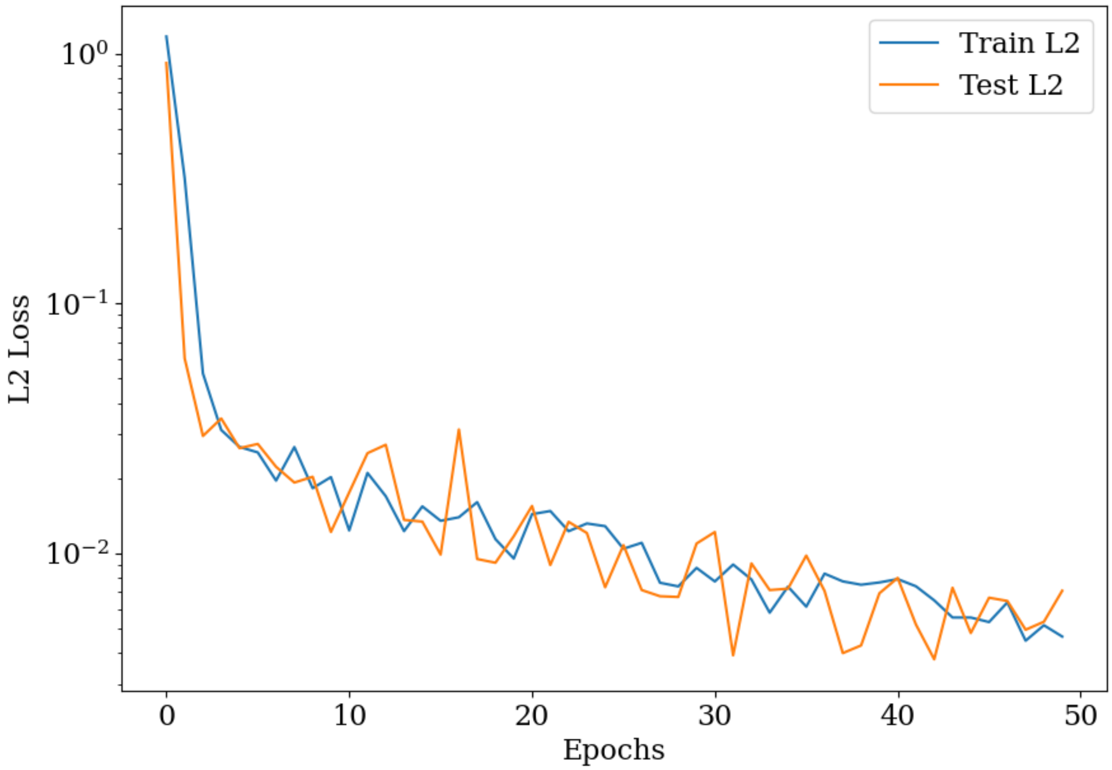
3.4 Plotting the Losses

Let's plot the losses:

[ ]:
```
plt.plot(train_mse_losses, label="Train MSE")
plt.plot(test_mse_losses, label="Test MSE")
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("MSE Loss")
plt.yscale("log")
plt.savefig(os.path.join("plots", "0101.png"))
plt.show()
```



[ ]:
```
plt.plot(train_l2_losses, label="Train L2")
plt.plot(test_l2_losses, label="Test L2")
plt.legend()
plt.xlabel("Epochs")
plt.ylabel("L2 Loss")
plt.yscale("log")
plt.savefig(os.path.join("plots", "0102.png"))
```


Train L2
Test L2
10^0
10^{-1}
10^{-2}
L2 Loss
0
10
20
30
40
50
Epochs