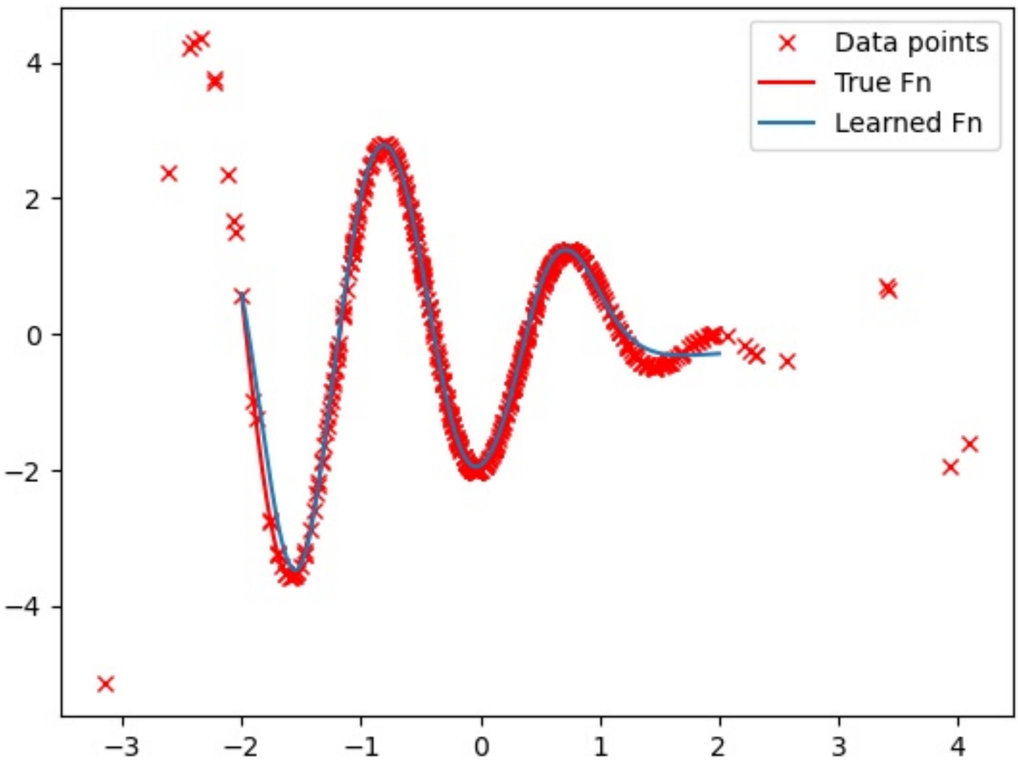
```
import torch
import numpy as np
from torch import nn, optim
from torch.nn import functional as F
from torch.utils.data import TensorDataset, DataLoader
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

alpha = 0.1
K = 1000
B = 128
N = 512

def f_true(x) :
    return (x-2) * np.cos(x*4)

torch.manual_seed(0)
X_train = torch.normal(0.0, 1.0, (N,))
y_train = f_true(X_train)
X_val = torch.normal(0.0, 1.0, (N//5,))
y_val = f_true(X_val)

train_dataloader = DataLoader(TensorDataset(X_train.unsqueeze(1), y_train.unsqueeze(1)),
batch_size=B, shuffle=True)
test_dataloader = DataLoader(TensorDataset(X_val.unsqueeze(1), y_val.unsqueeze(1)), batch_size=B)


# unsqueeze(1) reshapes the data into dimension [N,1],
# where is 1 the dimension of an data point.

# The batchsize of the test dataloader should not affect the test result
# so setting batch_size=N may simplify your code.
# In practice, however, the batchsize for the training dataloader
# is usually chosen to be as large as possible while not exceeding
# the memory size of the GPU. In such cases, it is not possible to
# use a larger batchsize for the test dataloader.


class MLP(nn.Module):
    def __init__(self, input_dim = 1):
        super().__init__()
        self.linear1 = nn.Linear(input_dim, 64, bias=True)
        self.linear2 = nn.Linear(64,64, bias=True)
        self.linear3 = nn.Linear(64,1, bias=True)
        self.layers = [self.linear1, nn.Sigmoid(), self.linear2, nn.Sigmoid() ,self.linear3]
    def forward(self, x):
        x = nn.Sequential(*self.layers)(x)
        return x

model = MLP()

for i in range(0,len(model.layers),2):
    layer = model.layers[i]
```

```
    layer.weight.data = torch.normal(0, 1, layer.weight.shape)
    layer.bias.data = torch.full(layer.bias.shape, 0.03)

loss_function = nn.MSELoss()

optimizer = torch.optim.SGD(model.parameters(), lr = alpha)

# training
for _ in range(K):
    for x, y in train_dataloader:
        optimizer.zero_grad()

        train_loss = loss_function(model(x), y)*0.5
        train_loss.backward()

        optimizer.step()

# plotting
with torch.no_grad():
    xx = torch.linspace(-2,2,1024).unsqueeze(1)
    plt.plot(X_train,y_train,'rx',label='Data points')
    plt.plot(xx,f_true(xx),'r',label='True Fn')
    plt.plot(xx, model(xx),label='Learned Fn')
plt.legend()
plt.savefig('p1.png')
plt.show()
```


Data points
True Fn
Learned Fn
4
2
0
−2
−4
−3
−2
−1
0
1
2
3
4