```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from sklearn.kernel_ridge import KernelRidge

import numpy as np
import pandas as pd
import seaborn as sns

DEBUG_MODE = True
def pp(s):
    if(DEBUG_MODE):
        print(s)

class Net(nn.Module):

    def __init__(self, depth, width, initVariance):
        super(Net, self).__init__()

        self.preLayers = nn.ModuleList()
        for _ in range(depth):
            newLayer = nn.Linear(width, width)
            nn.init.normal_(newLayer.weight, 0, np.sqrt(initVariance))
            self.preLayers.append(newLayer)

        self.outputLayer = nn.Linear(width, 1)

    def forward(self, x):
        pp(f"x dims: {x.shape}")
        for l in self.preLayers:
            x = F.relu(l(x))

        y = F.relu(self.outputLayer(x))
        return y


    def fit(self, xs, ys, learningRate, nEpics=5): # TODO: change back to 10

        optimizer = optim.SGD(self.parameters(), lr=learningRate)
        lossFn = nn.MSELoss()

        lossPerEpoch = []

        for _ in range(nEpics):
            self.train()

            optimizer.zero_grad()
            y_hat = self(xs)
```

```
            pp(f"y_hats : {y_hat}")
            pp(f"ys : {ys}")
            loss_output = lossFn(y_hat, ys)
            loss_output.backward()
            optimizer.step()
            lossPerEpoch.append(loss_output.item()/ys.shape[0])

        return lossPerEpoch

def genData(n, m, d):
    def relu(num):
        return 0 if num < 0 else num
    def genNSphere(r, d):
        v = np.random.normal(0, r, d)
        d = np.sum(v**2) **(0.5)
        return v/d

    xss = [genNSphere(1, d) for _ in range(n)]
    ys = []
    for xs in xss:
        ys.append(sum(map(relu, xs))/m)

    return (np.array(xss), np.array(ys))

def calcError(preds, targets):
    pp(f"predictions: {preds}")
    pp(f"targets: {targets}")
    return ((preds - targets)**2).mean()

def arcKernel(x1: np.array, x2: np.array):
    x1Tx2 = np.dot(x1, x2)
    x1Tx2 = 1.0 if x1Tx2 > 1.0 else x1Tx2
    x1Tx2 = -1.0 if x1Tx2 < -1.0 else x1Tx2
    return x1Tx2*(np.pi - np.arccos(x1Tx2))/(2*np.pi)

def plotResults(df):
    sns.color_palette("Set2")

    #df['Loss'] = np.log(df['Loss'].astype(float))

    title = f"NN vs Kernel Regression"
    plot = sns.lineplot(
        data=df, x="N", y="Loss", hue="ModelType", ci=None
    ).set_title(title)
    plot.figure.savefig(f"{title}.png")
    plot.figure.clf()

def addHistoryRow(df, n, loss, modelType):
    print(df.head())
    new_row = pd.DataFrame(columns=df.columns)
```

```
    new_row.loc[0] = [n, loss, modelType]
    df = pd.concat([df, new_row], ignore_index=True)
    return df

############ BEGIN APPLICATION ############

ns = [20, 40, 80, 160]
d = 10
m = 5
lr = 0.1

historyDf = pd.DataFrame(columns=[
    "N", "Loss", "ModelType"
])

for n in ns:
    trainXs, trainYs = genData(n, m, d)
    testXs, testYs = genData(100, m, d)

    #pp(trainXs)
    #pp(trainYs)
    net = Net(depth=m, width=d, initVariance=1/m)
    lossPerEpoch = net.fit(torch.Tensor(trainXs), torch.Tensor(trainYs), lr)
    pp(f"Loss per eopch: {lossPerEpoch}")
    nnError = calcError(net(torch.Tensor(testXs)).detach().numpy().flatten(), testYs)

    historyDf = addHistoryRow(historyDf, n, nnError, "Neureal Network")

    krr = KernelRidge(kernel=arcKernel)
    print(trainXs)
    krr.fit(trainXs, trainYs)
    krrError = calcError(krr.predict(testXs), testYs)
    historyDf = addHistoryRow(historyDf, n, krrError, "Kernel Regression")


plotResults(historyDf)
```