M7-L2-P2

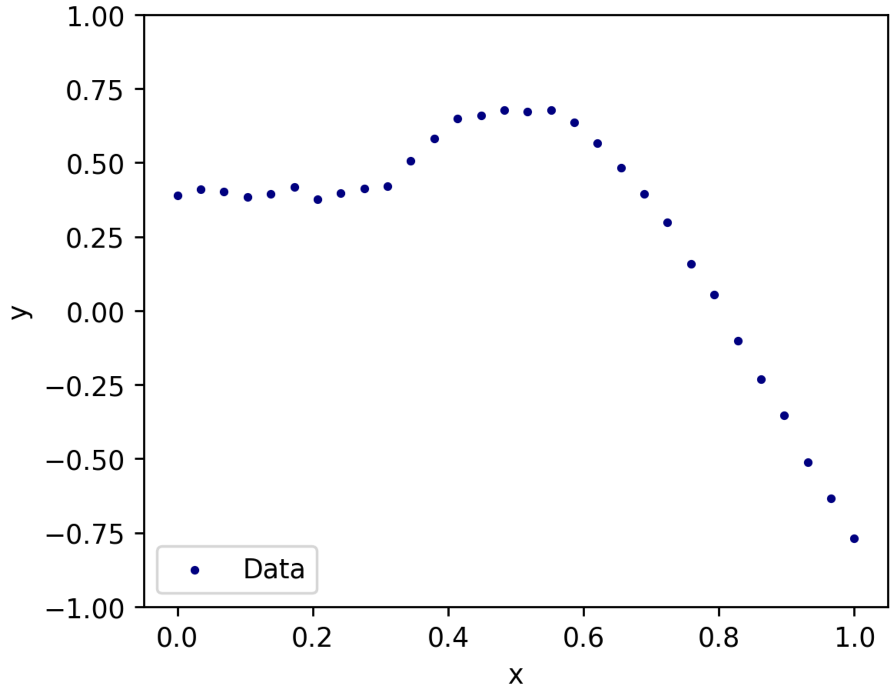
October 28, 2023

1 M7-L2 Problem 2

Here you will create a simple neural network for regression in PyTorch. PyTorch will give you a lot more control and flexibility for neural networks than SciKit-Learn, but there are some extra steps to learn.

Run the following cell to load our 1-D dataset:

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import optim, nn
import torch.nn.functional as F

x = np.array([0.        , 0.03448276, 0.06896552, 0.10344828, 0.13793103,0.
 ↪17241379, 0.20689655, 0.24137931, 0.27586207, 0.31034483,0.34482759, 0.
 ↪37931034, 0.4137931 , 0.44827586, 0.48275862,0.51724138, 0.55172414, 0.
 ↪5862069 , 0.62068966, 0.65517241,0.68965517, 0.72413793, 0.75862069, 0.
 ↪79310345, 0.82758621,0.86206897, 0.89655172, 0.93103448, 0.96551724, 1.
 ↪  ]).reshape(-1,1)
y = np.array([ 0.38914369,  0.40997345,  0.40282978,  0.38493705,  0.394214  ,0.
 ↪41651437,  0.37573321,  0.39571087,  0.41265936,  0.41953955,0.50596807,  0.
 ↪58059196,  0.6481607 ,  0.66050901,  0.67741369,0.67348567,  0.67696078,  0.
 ↪63537378,  0.56446933,  0.48265412,0.39540671,  0.29878237,  0.15893846,  0.
 ↪05525194, -0.10070259,-0.23055219, -0.35288448, -0.51317604, -0.63377544, -0.
 ↪76849408]).reshape(-1,1)


plt.figure(figsize=(5,4),dpi=250)
plt.scatter(x,y,s=5,c="navy",label="Data")
plt.legend(loc="lower left")
plt.ylim(-1,1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



1.1 PyTorch Tensors

PyTorch models only work with PyTorch Tensors, so we need to convert our dataset into a tensors.

To convert these back to numpy arrays we can use: - `x.detach().numpy()` - `y.detach().numpy()`

[ ]:
```
x = torch.Tensor(x)
y = torch.Tensor(y)
```

1.2 PyTorch Module

We create a subclass whose superclass is `nn.Module`, a basic predictive model, and we must define 2 methods.

`nn.Module` subclass: - `__init__()` - runs when creating a new model instance - includes the line `super().__init__()` to inherit parent methods from `nn.Module` - sets up all necessary model components/parameters - `forward()` - runs when calling a model instance - performs a forward pass through the network given an input tensor.

This class `Net_2_layer` is an MLP for regression with 2 layers. At initialization, the user inputs the number of hidden neurons per layer, the number of inputs and outputs, and the activation function.

[ ]:
```
class Net_2_layer(nn.Module):
    def __init__(self, N_hidden=6, N_in=1, N_out=1, activation = F.relu):
        super().__init__()
```

```
        # Linear transformations -- these have weights and biases as trainable
↪parameters,
        # so we must create them here.
        self.lin1 = nn.Linear(N_in, N_hidden)
        self.lin2 = nn.Linear(N_hidden, N_hidden)
        self.lin3 = nn.Linear(N_hidden, N_out)
        self.act = activation

    def forward(self,x):
        x = self.lin1(x)
        x = self.act(x)  # Activation of first hidden layer
        x = self.lin2(x)
        x = self.act(x)  # Activation at second hidden layer
        x = self.lin3(x) # (No activation at last layer)

        return x
```

1.3 Instantiate a model

This model has 6 neurons at each hidden layer, and it uses ReLU activation.

[ ]:
```
model = Net_2_layer(N_hidden = 6, activation = F.relu)
loss_curve = []
```

1.4 Training a model

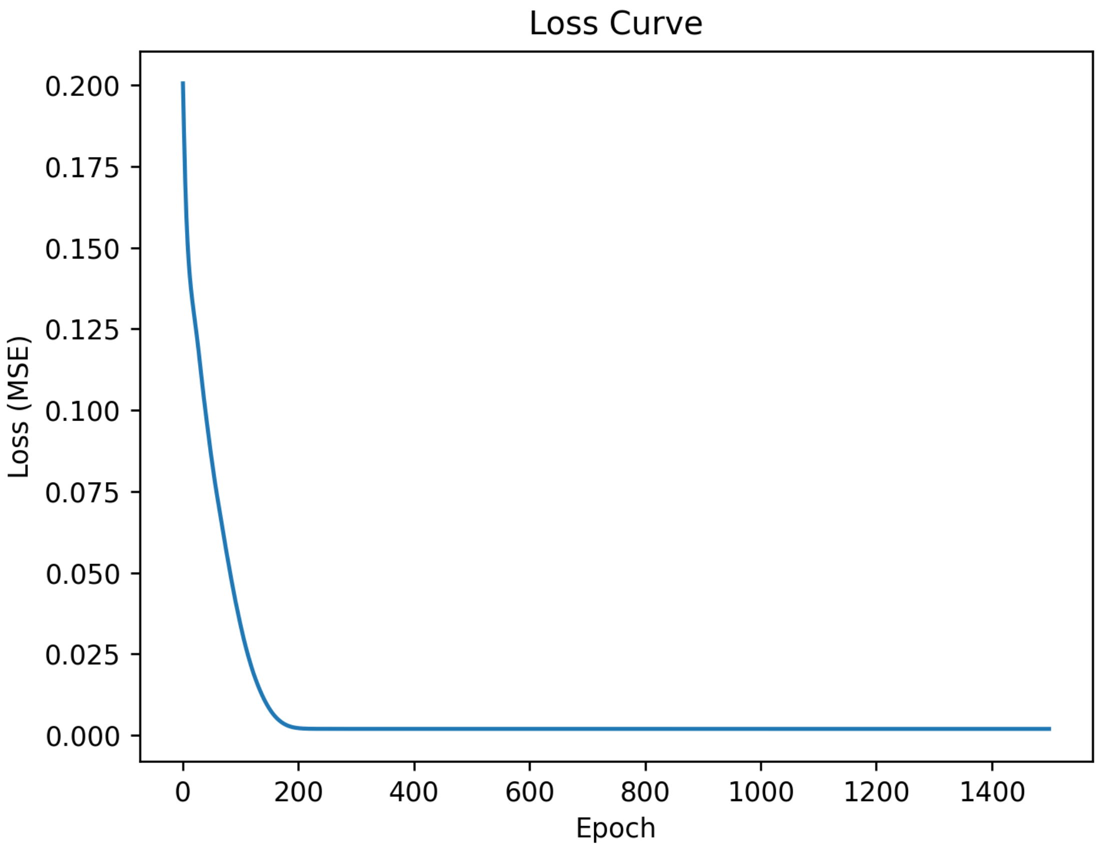
[ ]:
```
# Training parameters: Learning rate, number of epochs, loss function
# (These can be tuned)
lr = 0.005
epochs = 1500
loss_fcn = F.mse_loss

# Set up optimizer to optimize the model's parameters using Adam with the
↪selected learning rate
opt = optim.Adam(params = model.parameters(), lr=lr)

# Training loop
for epoch in range(epochs):
    out = model(x) # Evaluate the model
    loss = loss_fcn(out,y) # Calculate the loss -- error between network
↪prediction and y

    loss_curve.append(loss.item())

    # Print loss progress info 25 times during training
    if epoch % int(epochs / 25) == 0:
        print(f"Epoch {epoch} of {epochs}... \tAverage loss: {loss.item()}")
```

```
    # Move the model parameters 1 step closer to their optima:
    opt.zero_grad()
    loss.backward()
    opt.step()
```

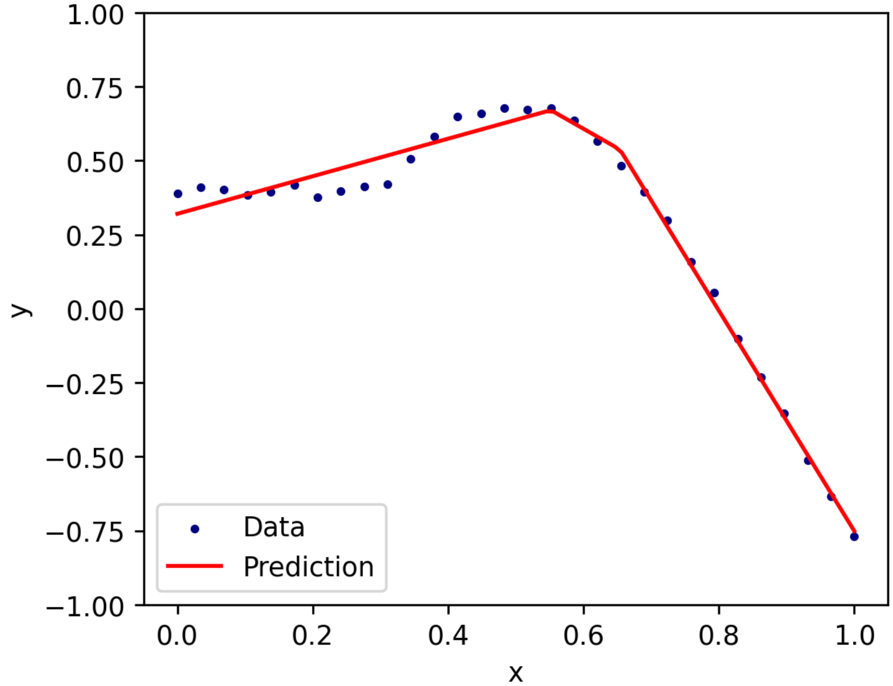
```
Epoch 0 of 1500…      Average loss: 0.20060913264751434
Epoch 60 of 1500…     Average loss: 0.07273828983306885
Epoch 120 of 1500…    Average loss: 0.020207548514008522
Epoch 180 of 1500…    Average loss: 0.0030261396896094084
Epoch 240 of 1500…    Average loss: 0.0019302271539345384
Epoch 300 of 1500…    Average loss: 0.0019199324306100607
Epoch 360 of 1500…    Average loss: 0.0019171799067407846
Epoch 420 of 1500…    Average loss: 0.0019168586004525423
Epoch 480 of 1500…    Average loss: 0.001916774665005505
Epoch 540 of 1500…    Average loss: 0.0019166858401149511
Epoch 600 of 1500…    Average loss: 0.0019165772246196866
Epoch 660 of 1500…    Average loss: 0.0019164994591847062
Epoch 720 of 1500…    Average loss: 0.0019163553370162845
Epoch 780 of 1500…    Average loss: 0.0019162431126460433
Epoch 840 of 1500…    Average loss: 0.0019161227392032743
Epoch 900 of 1500…    Average loss: 0.0019159989897161722
Epoch 960 of 1500…    Average loss: 0.001915863249450922
Epoch 1020 of 1500…   Average loss: 0.0019157796632498503
Epoch 1080 of 1500…   Average loss: 0.0019156151684001088
Epoch 1140 of 1500…   Average loss: 0.0019154789624735713
Epoch 1200 of 1500…   Average loss: 0.0019153128378093243
Epoch 1260 of 1500…   Average loss: 0.0019152022432535887
Epoch 1320 of 1500…   Average loss: 0.001915033208206296
Epoch 1380 of 1500…   Average loss: 0.0019148987485095859
Epoch 1440 of 1500…   Average loss: 0.0019147639395669103
```

[ ]:

```
plt.figure(dpi=250)
plt.plot(loss_curve)
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.title('Loss Curve')
plt.show()
```



[ ]:
```
xs = torch.linspace(0,1,100).reshape(-1,1)
ys = model(xs)

plt.figure(figsize=(5,4),dpi=250)
plt.scatter(x,y,s=5,c="navy",label="Data")
plt.plot(xs.detach().numpy(), ys.detach().numpy(),"r-",label="Prediction")
plt.legend(loc="lower left")
plt.ylim(-1,1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



1.5 Your Turn

In the cells below, create a new instance of `Net_2_layer`. This time, use 20 neurons per hidden layer, and an activation of `F.tanh`. Plot the loss curve and a visualization of the prediction with the data.

[ ]:
```
model = Net_2_layer(N_hidden = 20, activation = F.tanh)
loss_curve = []
```

[ ]:
```
# Training parameters: Learning rate, number of epochs, loss function
# (These can be tuned)
lr = 0.005
epochs = 1500
loss_fcn = F.mse_loss

# Set up optimizer to optimize the model's parameters using Adam with the␣
↪selected learning rate
opt = optim.Adam(params = model.parameters(), lr=lr)

# Training loop
for epoch in range(epochs):
    out = model(x) # Evaluate the model
    loss = loss_fcn(out,y) # Calculate the loss -- error between network␣
↪prediction and y
```

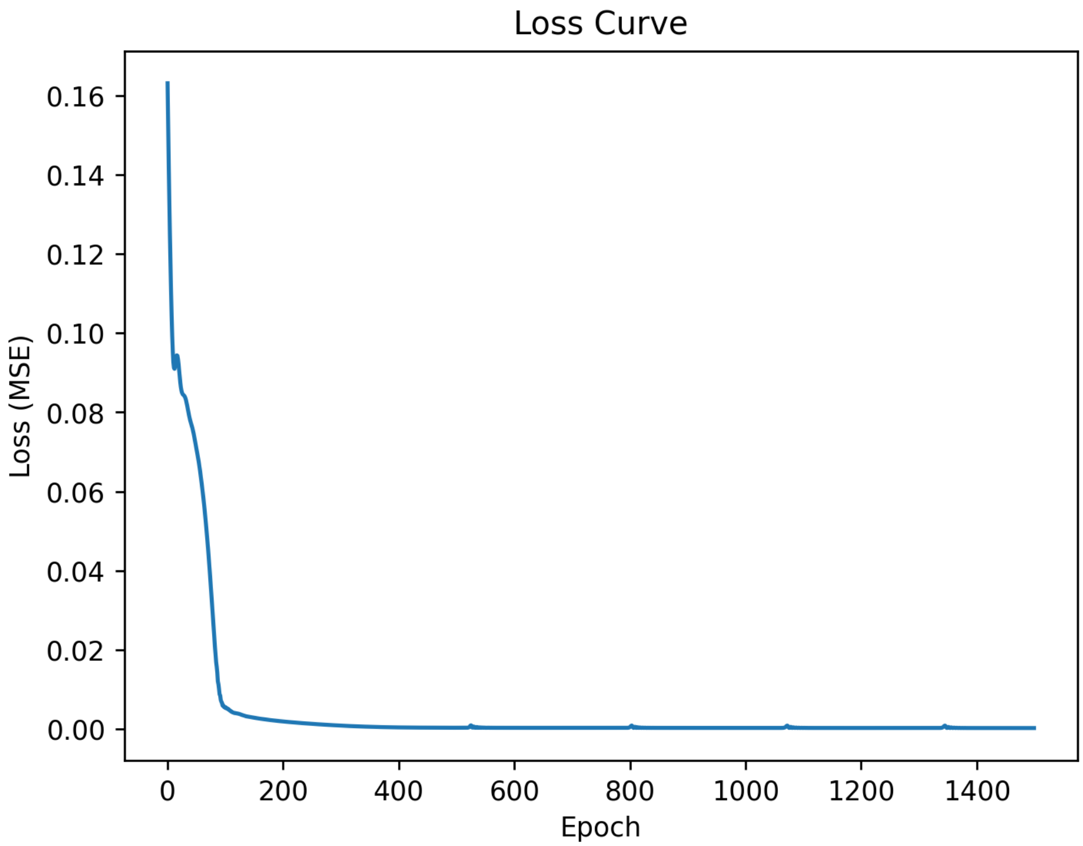
```
    loss_curve.append(loss.item())

    # Print loss progress info 25 times during training
    if epoch % int(epochs / 25) == 0:
        print(f"Epoch {epoch} of {epochs}... \tAverage loss: {loss.item()}")

    # Move the model parameters 1 step closer to their optima:
    opt.zero_grad()
    loss.backward()
    opt.step()
```

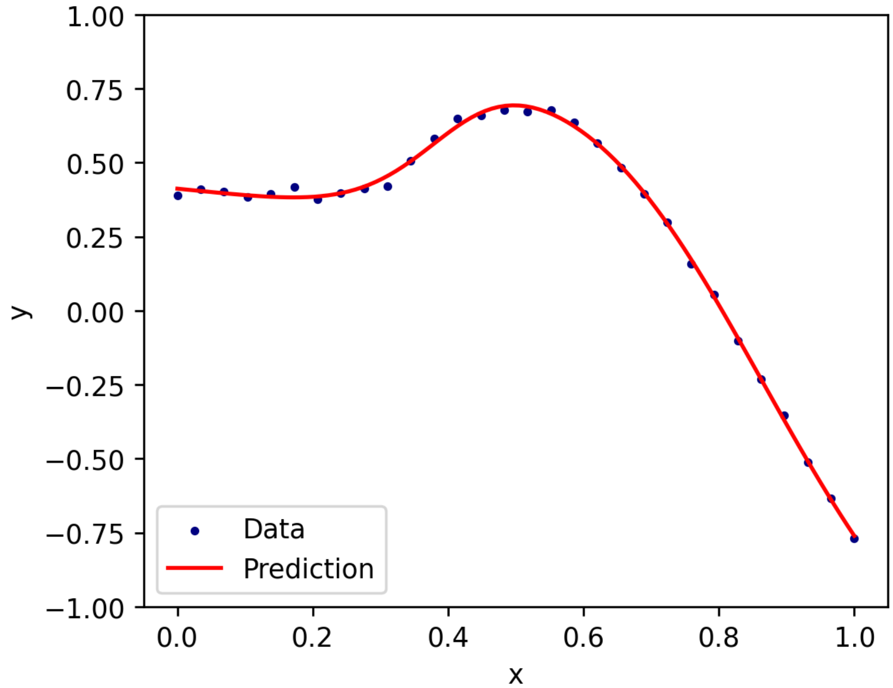
```
Epoch 0 of 1500…      Average loss: 0.16317762434482574
Epoch 60 of 1500…     Average loss: 0.06078961864113808
Epoch 120 of 1500…    Average loss: 0.003894438734278083
Epoch 180 of 1500…    Average loss: 0.0021709902212023735
Epoch 240 of 1500…    Average loss: 0.0013601431855931878
Epoch 300 of 1500…    Average loss: 0.0008099587867036462
Epoch 360 of 1500…    Average loss: 0.0004718304262496531
Epoch 420 of 1500…    Average loss: 0.00032416018075309694
Epoch 480 of 1500…    Average loss: 0.0002800915972329676
Epoch 540 of 1500…    Average loss: 0.00028401173767633736
Epoch 600 of 1500…    Average loss: 0.000254952086834237
Epoch 660 of 1500…    Average loss: 0.00024991866666823626
Epoch 720 of 1500…    Average loss: 0.0002460372052155435
Epoch 780 of 1500…    Average loss: 0.0002424498670734465
Epoch 840 of 1500…    Average loss: 0.00024211336858570576
Epoch 900 of 1500…    Average loss: 0.00023554654035251588
Epoch 960 of 1500…    Average loss: 0.0002320874627912417
Epoch 1020 of 1500…   Average loss: 0.00022848552907817066
Epoch 1080 of 1500…   Average loss: 0.0002608615905046463
Epoch 1140 of 1500…   Average loss: 0.00022104621166363358
Epoch 1200 of 1500…   Average loss: 0.0002172699460061267
Epoch 1260 of 1500…   Average loss: 0.0002132928348146379
Epoch 1320 of 1500…   Average loss: 0.00020909849263262004
Epoch 1380 of 1500…   Average loss: 0.00021068908972665668
Epoch 1440 of 1500…   Average loss: 0.000200624592253007
```

[ ]:

```
plt.figure(dpi=250)
plt.plot(loss_curve)
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.title('Loss Curve')
plt.show()

xs = torch.linspace(0,1,100).reshape(-1,1)
ys = model(xs)
```

```
plt.figure(figsize=(5,4),dpi=250)
plt.scatter(x,y,s=5,c="navy",label="Data")
plt.plot(xs.detach().numpy(), ys.detach().numpy(),"r-",label="Prediction")
plt.legend(loc="lower left")
plt.ylim(-1,1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```




1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
0.0
0.2
0.4
0.6
0.8
1.0
x
Data
Prediction