M10-HW1

November 18, 2023

1 Problem 1

1.1 Problem Description

In this problem you will fit a neural network to solve a simple regression problem. You will use 5 fold cross validation, plotting training and validation loss curves, as well as model predictions for each of the folds. You will compare between results for 3 neural networks, trained for 100, 500, and 2000 epochs respectively.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

- Visualization of provided data
- `trainModel()` function
- 15 figures containing two subplots (loss curves and model prediction) across all 5 folds for the 3 models
- Average MSE across all folds for the 3 models
- Discussion and comparison of model performance, and the importance of cross validation for evaluating model performance.

Imports and Utility Functions:

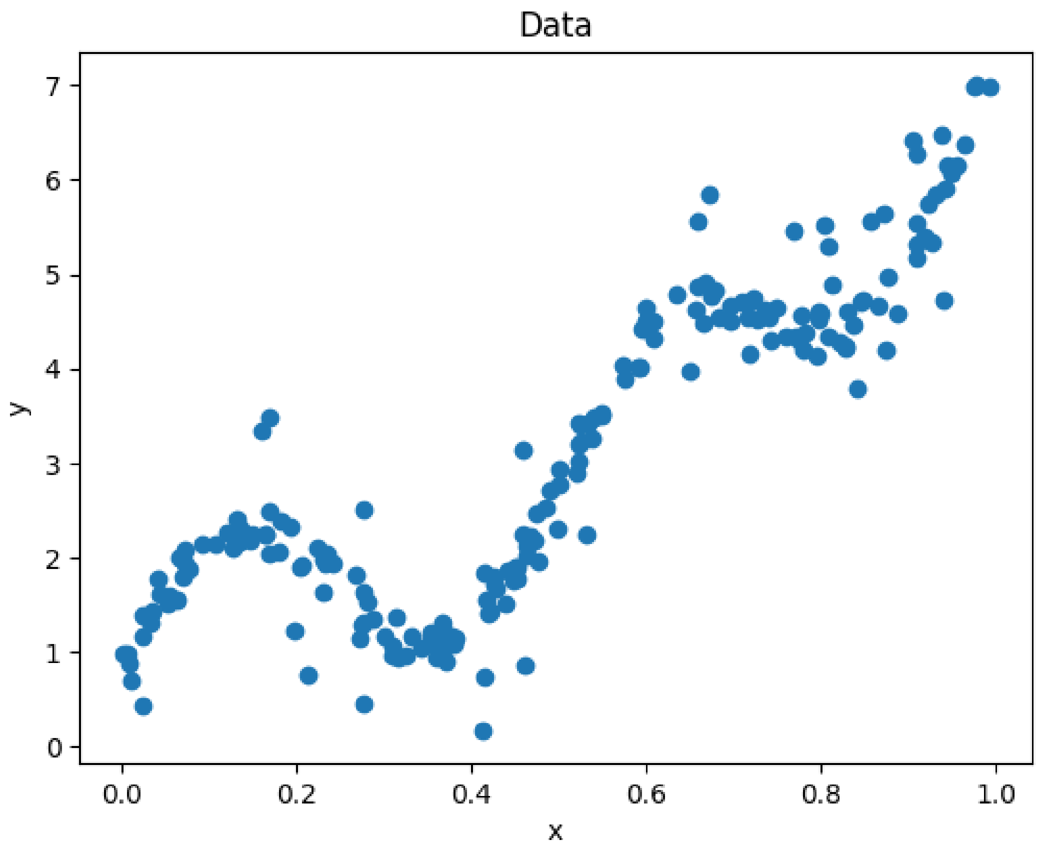
[ ]:
```
import torch
from torch import nn, optim

import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import mean_squared_error

def plotLoss(ax, train_curve, val_curve):
    ax.plot(train_curve, label = 'Training')
    ax.plot(val_curve, label = 'Validation')
    ax.set_xlabel('Epoch')
    ax.set_ylabel('Loss')
    ax.legend()
```

```
def plotModel(ax, model, x, y, idx_train, idx_test):
    xs = torch.linspace(min(x).item(), max(x).item(), 200).reshape(-1,1)
    ys = model(xs)
    ax.scatter(x[idx_train], y[idx_train], c = 'blue', alpha = 0.5, label =␣
 ↪'Training Data')
    ax.scatter(x[idx_test], y[idx_test], c = 'green', alpha = 0.5, label =␣
 ↪'Test Data')
    ax.plot(xs.detach().numpy(), ys.detach().numpy(), 'k--', label = 'Fitted␣
 ↪Function')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.legend()
```

1.2 Load and visualize the data

Data can be loaded from the `m10-hw1-data.txt` file using `np.loadtxt()`. The first column of the data corresponds to the x values and the second column corresponds to the y values. Visualize the data using a scatter plot.

[ ]:
```
data = np.loadtxt("data/m10-hw1-data.txt")
x = torch.tensor(data[:,0])
y = torch.tensor(data[:,1])
plt.figure()
plt.scatter(x, y)
plt.title("Data")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



1.3 Create Neural Network and NN Training Function

Create a neural network to predict the underlying function of the data using fully connected layers and tanh activation functions, with no activation on the output layer. The network should have 4 hidden layers, with the following shape: $[64, 128, 128, 64]$.

Since we are going to train many models throughout k-fold cross validation, you will create a function `trainModel(x, y, n_epoch)` that returns `model,train_curve,val_curve`, where `model` is the trained PyTorch model, `train_curve` and `val_curve` are lists of the training and validation loss at each epoch throughout the training, respectively. Use `nn.MSELoss()` as the loss function. Use the `torch.optim.Adam()` optimizer with a learning rate of 0.01. You will instantiate your neural network inside of the training function, as we train a new model with each of the k folds. The x and y which we pass the model will be split into training and validation sets using `train_test_split()` from sklearn, with a `test_size` of 0.25. Note: since we already split the train/test data 80/20 with each k fold, 25% of the remaining training data will correspond to 20% of the total data. Thus for any given fold, we have 60% of the data for training, 20% for validation, and 20% for testing.

[ ]:
```
class Network(nn.Module):
    def __init__(self, n_inputs, n_outputs, hidden_layer_sizes,␣
 ↪activation_functions):
```

```
        super().__init__()
        self.seq = nn.Sequential(
            nn.Linear(n_inputs,hidden_layer_sizes[0], dtype=torch.float32),
        )
        for i in range(len(hidden_layer_sizes[:-1])):
            self.seq.append(activation_functions[i])
            self.seq.append(nn.Linear(hidden_layer_sizes[i],␣
↪hidden_layer_sizes[i+1], dtype=torch.float32))
        self.seq.append(activation_functions[-1])
        self.seq.append(nn.Linear(hidden_layer_sizes[-1], n_outputs,␣
↪dtype=torch.float32))
    def forward(self, x):
        return self.seq(x)

def trainModel(x: torch.tensor, y: torch.tensor, n_epochs: int):
    train_curve = []
    val_curve = []

    loss_fcn = nn.MSELoss()

    n_inputs = 1
    n_outputs = 1
    hidden_layer_sizes = [64, 128, 128, 64]
    model = Network(n_inputs, n_outputs, hidden_layer_sizes, [nn.Tanh() for i␣
↪in range(len(hidden_layer_sizes))])

    opt = optim.Adam(params = model.parameters(), lr=0.01)

    x_train, x_val, y_train, y_val = [out.to(torch.float32) for out in␣
↪train_test_split(x, y, test_size=0.25)]

    for epoch in range(n_epochs):

        model.train()
        x_out_train = model(x_train)
        loss_train = loss_fcn(x_out_train, y_train)

        x_out_val = model(x_val)
        loss_val = loss_fcn(x_out_val, y_val)

        opt.zero_grad()
        loss_train.backward()
        opt.step()

        train_curve.append(loss_train.item())
        val_curve.append(loss_val.item())
```

```
        # if epoch % int(n_epochs / 25) == 0:
        #     print(f"Epoch {epoch:>4} of {n_epochs}:   Train Loss =␣
↪{loss_train.item():.6f}")

    return model, train_curve, val_curve
```

1.4 K-Fold Cross Validation

Now we will compare across three models trained for $[100, 500, 2000]$ epochs using 5-fold cross validation. We will use the `KFold()` function from sklearn to get indices of the training and test sets for the 5 folds. Then use your `trainModel()` function from the previous section to train a network for each fold.

For each fold, generate a figure with two subplots: training and validation curves on one, and the model prediction plotted with the training and test data on the other. The training and validation curves can be generated using the provided `plotLoss()` function which takes in a subplot axes handle, `ax`, and the training and validation loss lists, `train_curve` and `val_curve`. The model prediction can be plotted using the `plotModel()` function which takes in a subplot axes handle, `ax`, the trained model, `model`, the complete datasets `x` and `y`, and `idx_train` and `idx_test`, the indices of of the training and test data for that specific fold.

The generated figure should also be titled with the MSE of the trained model on the test data using `suptitle()` from matplotlib, such that the title is centered above the two subplots. The MSE can be computed using the `mean_squared_error` function from sklearn or `MSELoss` from PyTorch.

Average the MSE loss on the test set across the 5 folds, and report a single MSE loss for each of the three models.

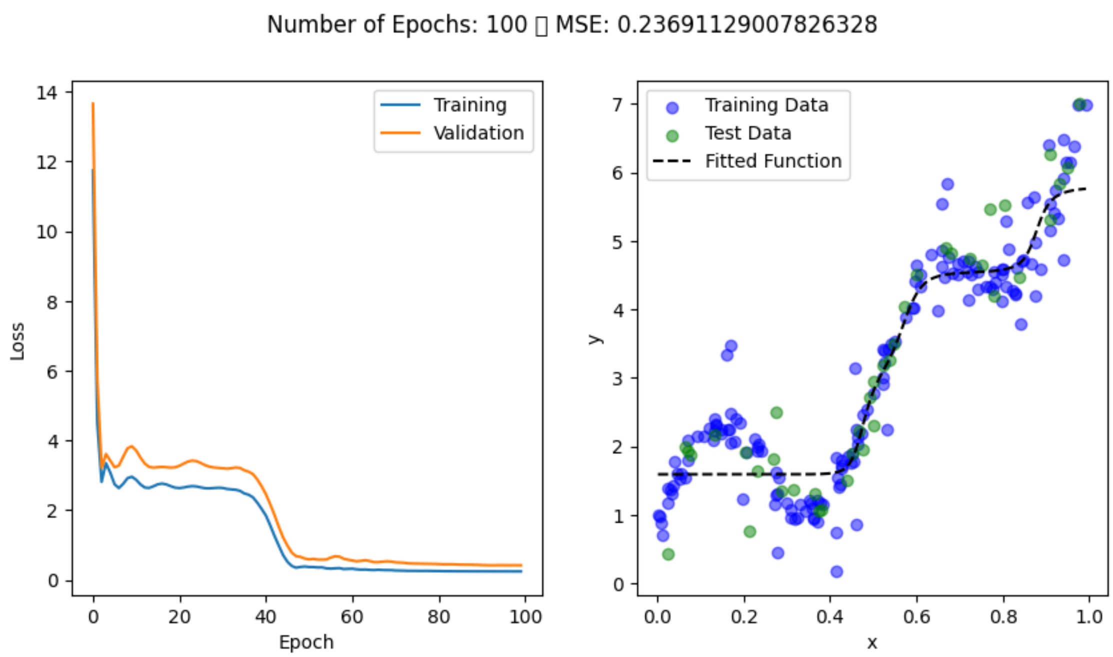
Since there are three models and we are using 5-fold cross validation, you should output 15 figures, with two subplots each.

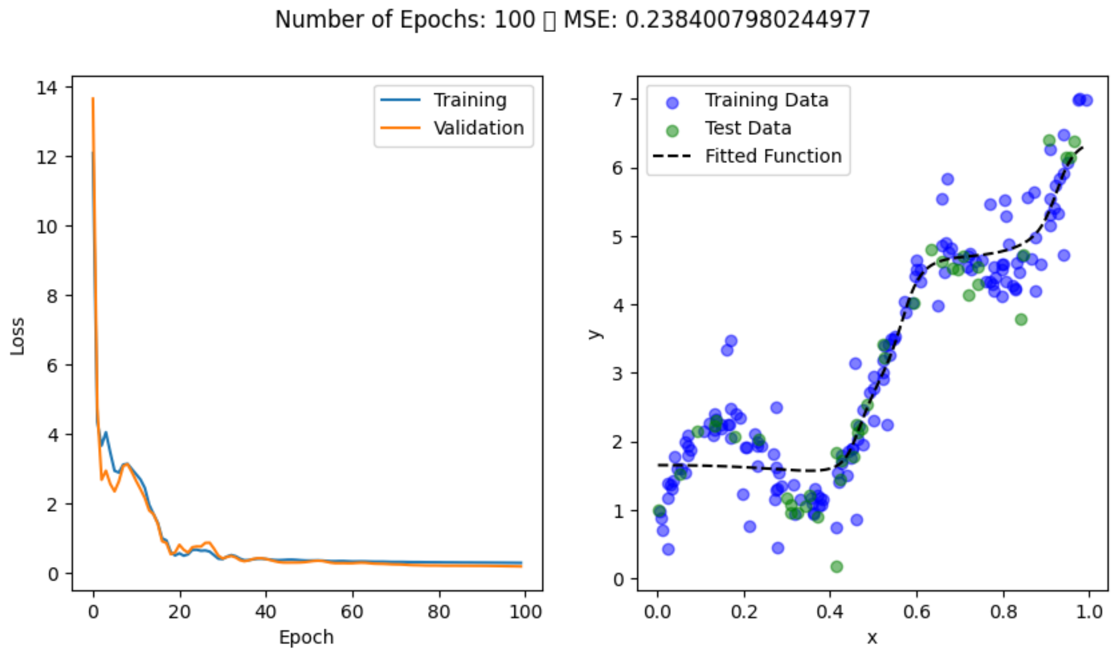
[ ]:
```
kf = KFold(n_splits=5)

for n_epochs in [100, 500, 2000]:
    mse_average = []
    for i, (train_index, test_index) in enumerate(kf.split(x)):
        model, train_curve, val_curve = trainModel(x[train_index].
↪reshape(-1,1), y[train_index].reshape(-1,1), n_epochs)

        mse_average.append(mean_squared_error(model(x[test_index].reshape(-1,1).
↪to(torch.float32)).detach().numpy(), y[test_index].reshape(-1,1).detach().
↪numpy()))
        fig, axs = plt.subplots(1, 2)
        fig.set_figwidth(10)
        fig.set_figheight(5)
        fig.suptitle(f"Number of Epochs: {n_epochs} \t MSE: {mse_average[i]}")

        plotLoss(axs[0], train_curve, val_curve)
        plotModel(axs[1], model, x, y, train_index, test_index)
```

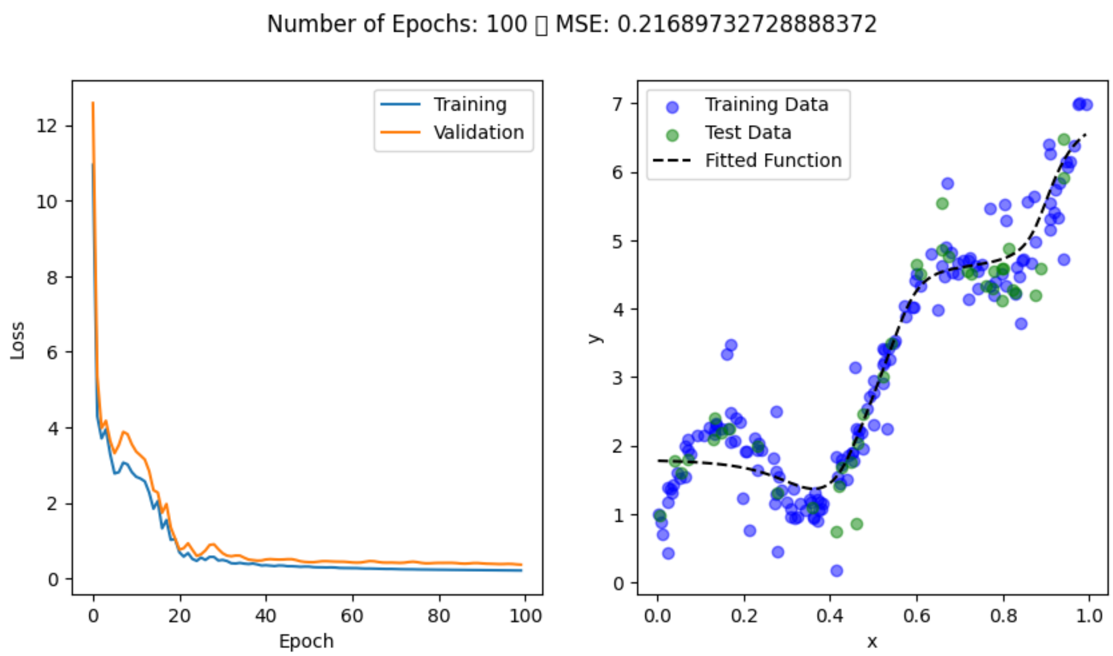
```
    print(f"MSE Average for {n_epochs} epochs = {np.average(mse_average)}")
```

```
MSE Average for 100 epochs = 0.2611703494112058
MSE Average for 500 epochs = 0.20418395547426432
MSE Average for 2000 epochs = 0.21787915186360046
```

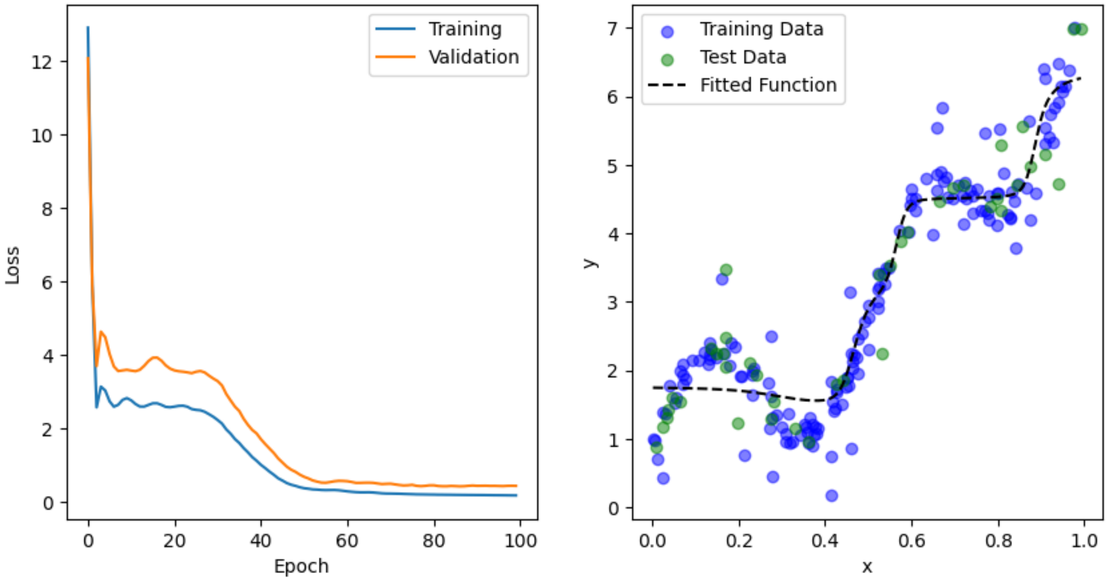
```
/opt/miniconda3/lib/python3.8/site-packages/IPython/core/events.py:89:
UserWarning: Glyph 9 (	) missing from current font.
  func(*args, **kwargs)
/opt/miniconda3/lib/python3.8/site-packages/IPython/core/pylabtools.py:152:
UserWarning: Glyph 9 (	) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```



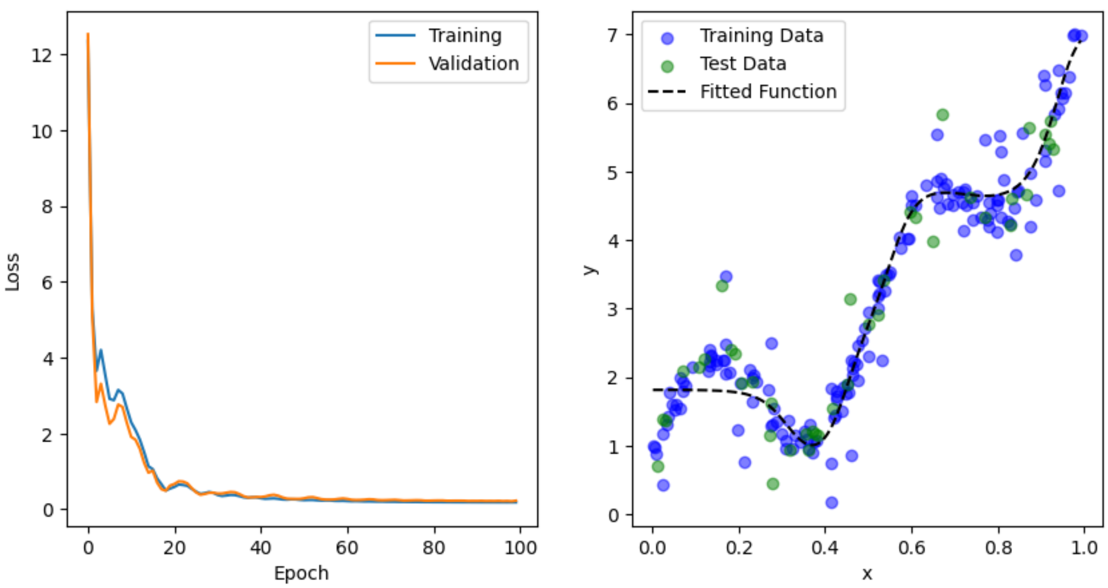

Number of Epochs: 100 □ MSE: 0.2384007980244977
Training
Validation
Loss
Epoch
Training Data
Test Data
Fitted Function
y
x

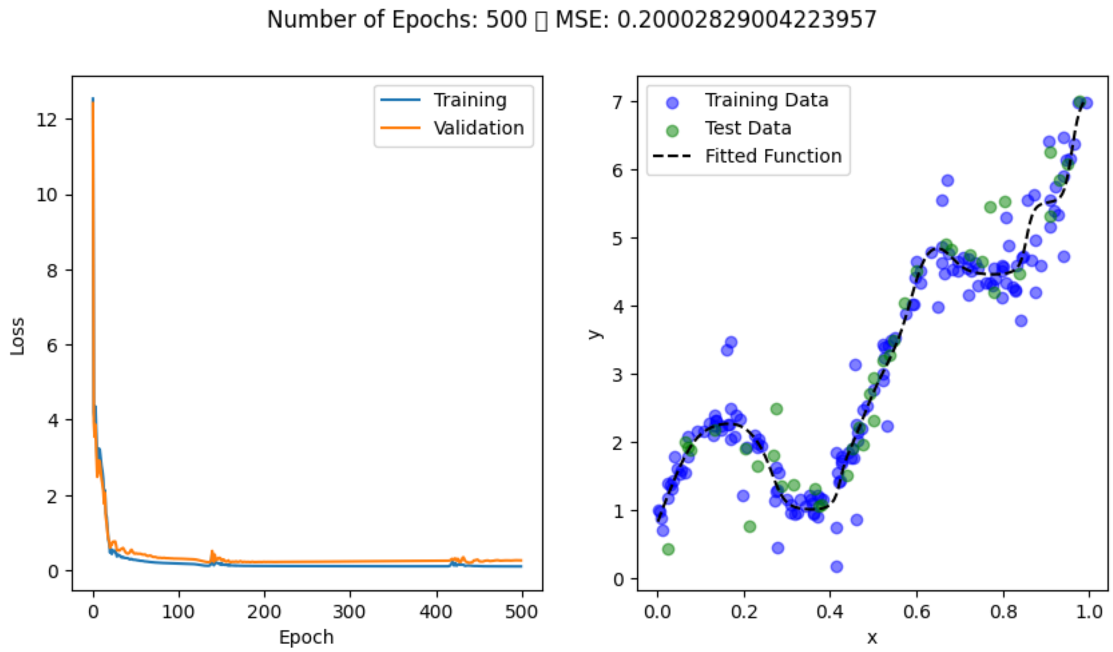

Number of Epochs: 100 □ MSE: 0.21689732728888372
Training
Validation
Loss
Epoch
Training Data
Test Data
Fitted Function
y
x

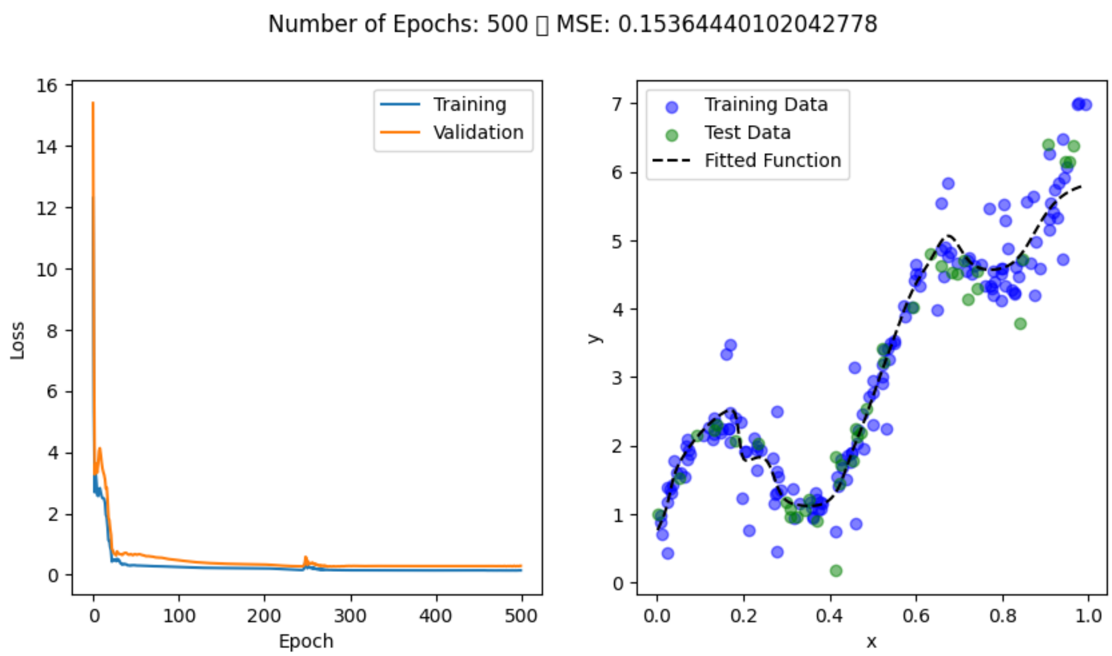
Number of Epochs: 100 ▯ MSE: 0.3355758543266182

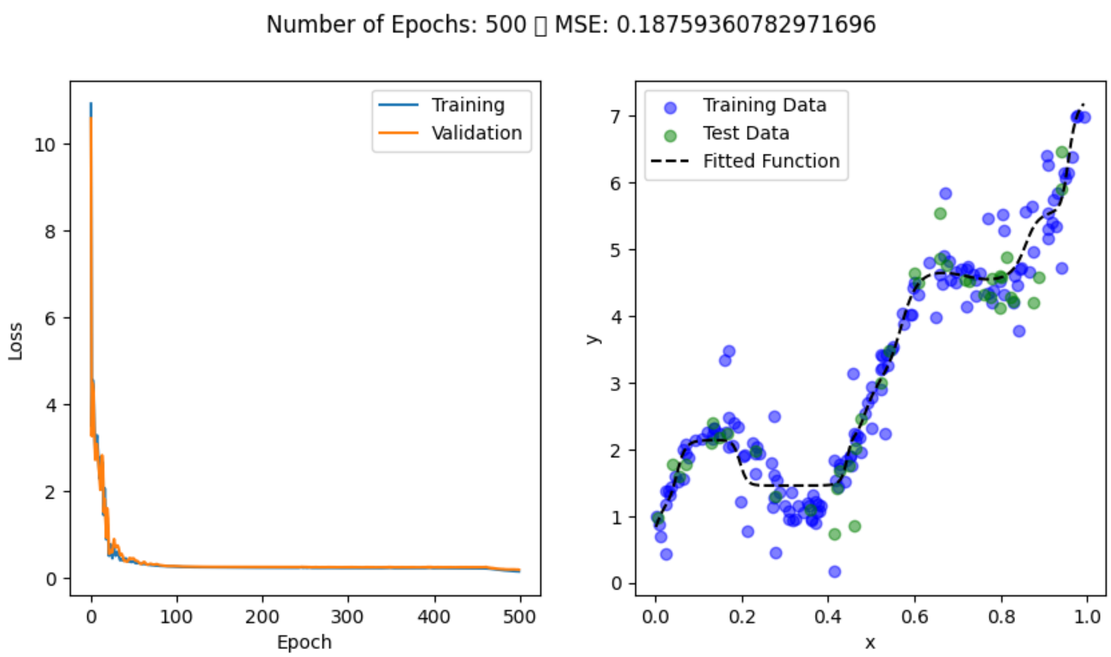


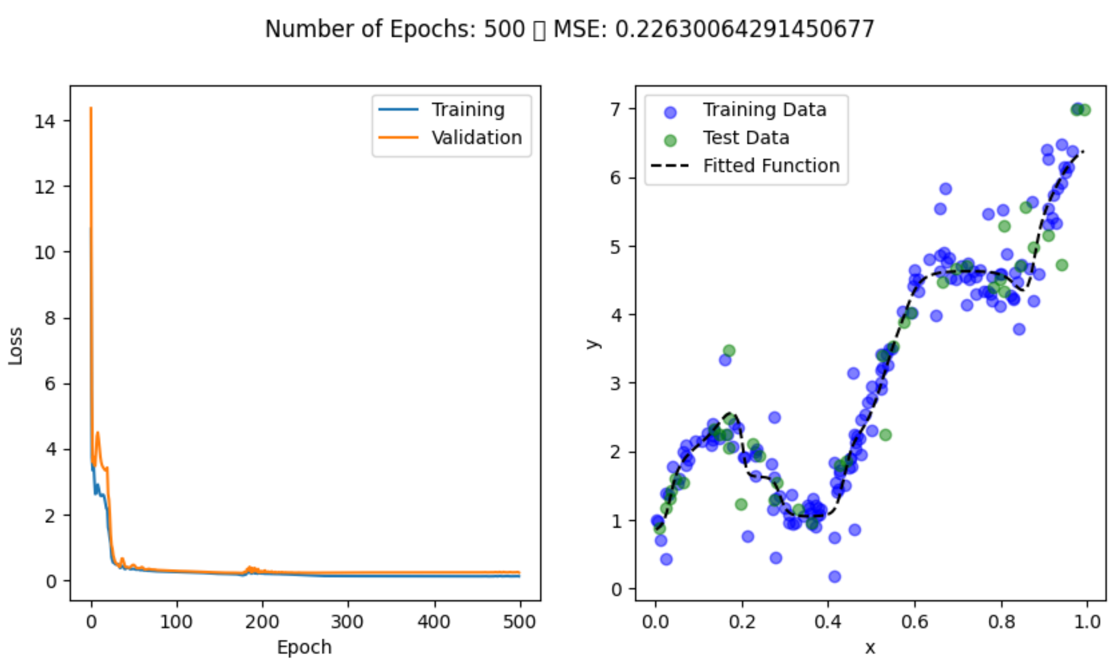
Number of Epochs: 100 ▯ MSE: 0.278066477337766



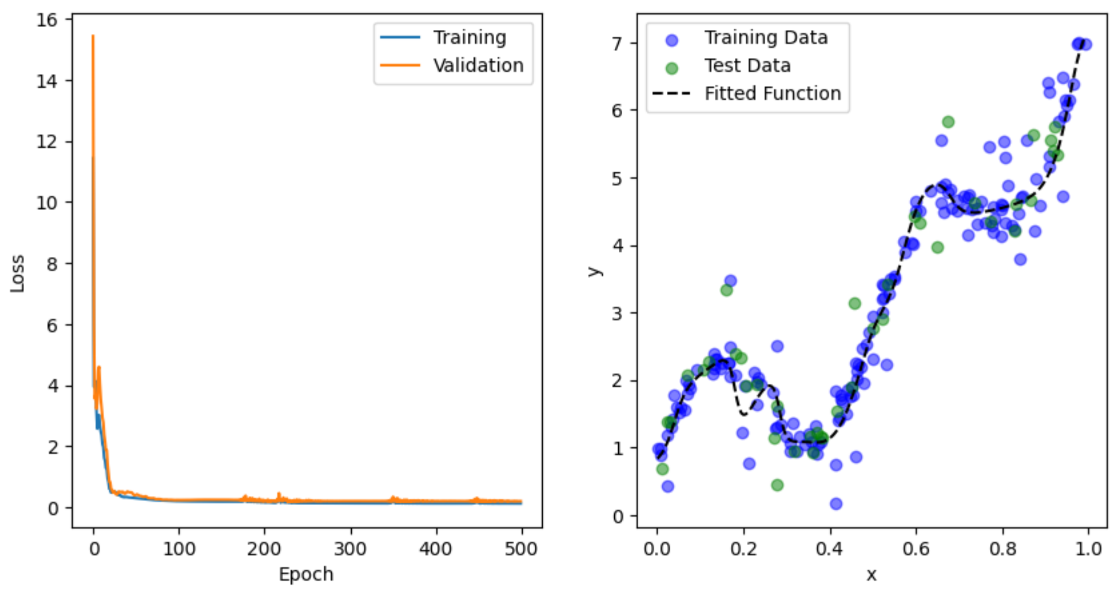

Number of Epochs: 500 □ MSE: 0.20002829004223957
Training
Validation
Loss
Epoch
Training Data
Test Data
Fitted Function
y
x


Number of Epochs: 500 □ MSE: 0.15364440102042778
Training
Validation
Loss
Epoch
Training Data
Test Data
Fitted Function
y
x

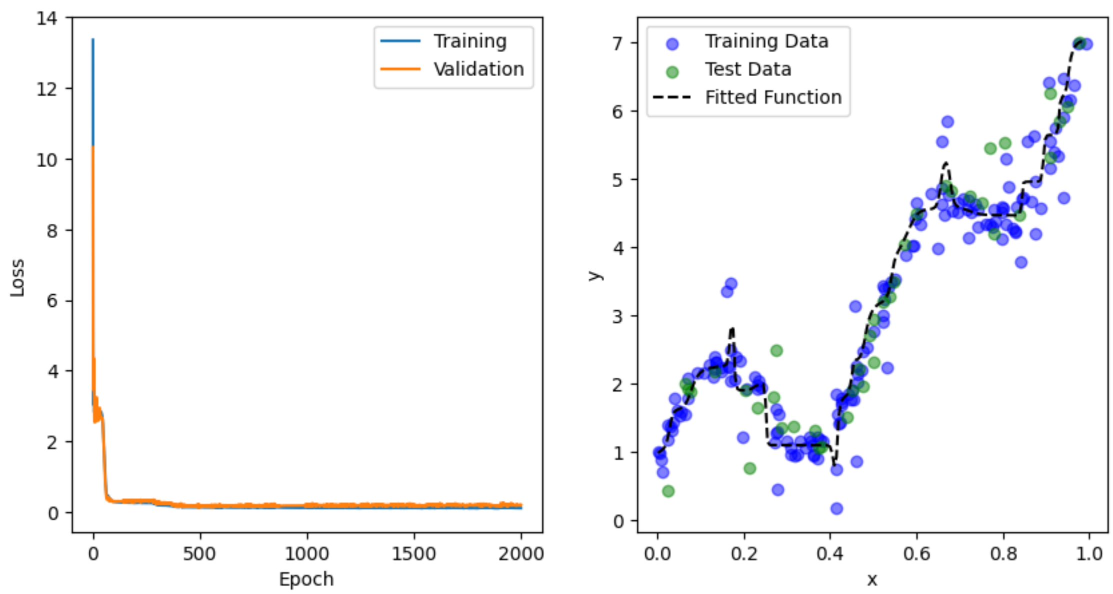

Number of Epochs: 500 □ MSE: 0.18759360782971696
Training
Validation
Loss
Epoch
Training Data
Test Data
Fitted Function
y
x


Number of Epochs: 500 □ MSE: 0.22630064291450677
Training
Validation
Loss
Epoch
Training Data
Test Data
Fitted Function
y
x

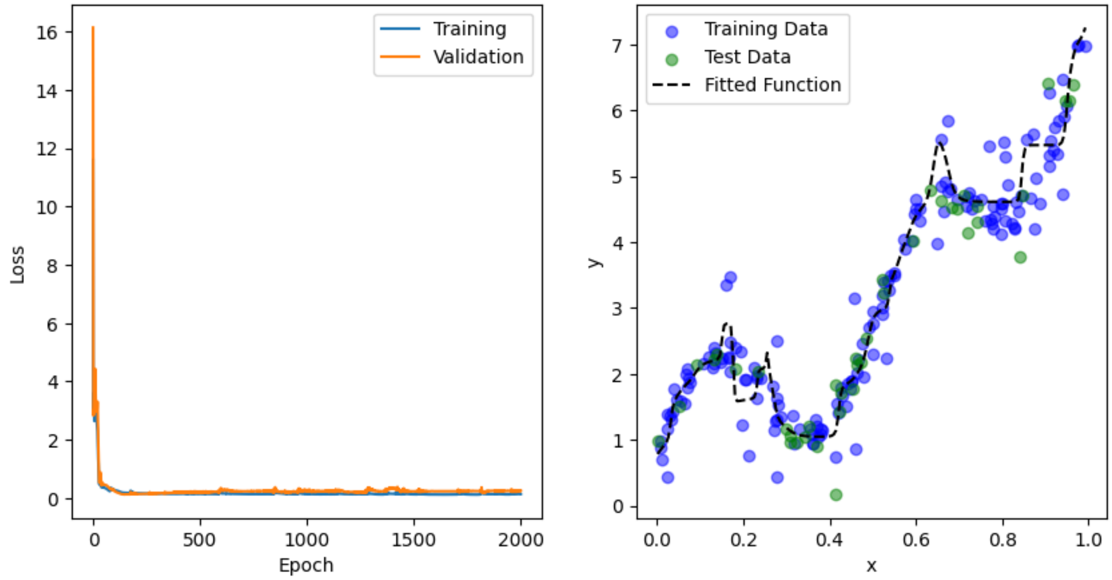
Number of Epochs: 500 □ MSE: 0.25335283556443045



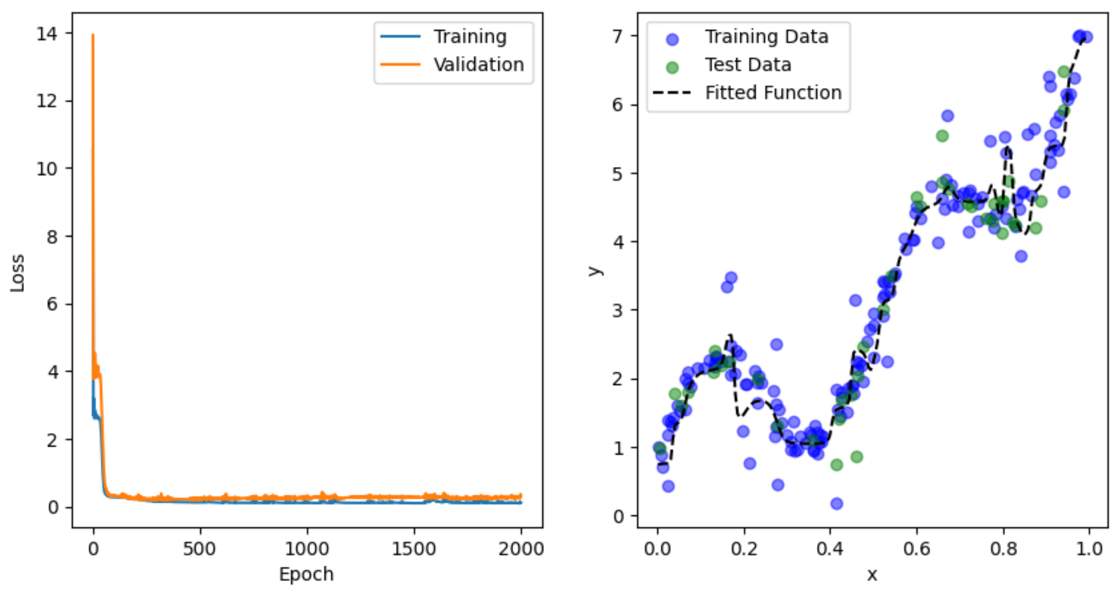
Number of Epochs: 2000 □ MSE: 0.23032430381731506

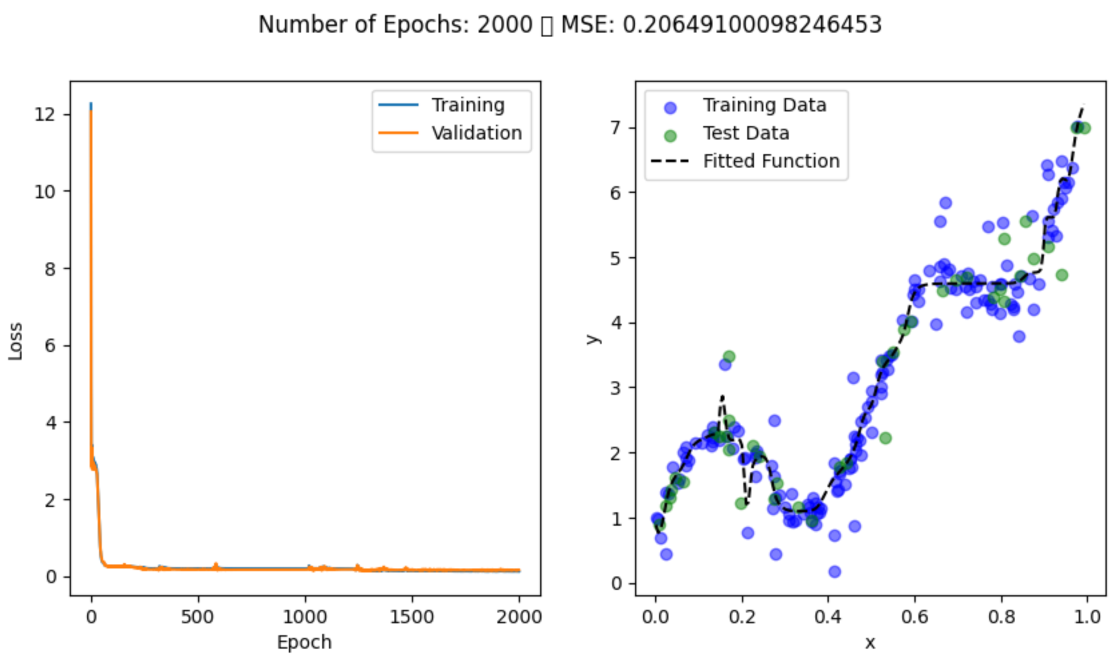


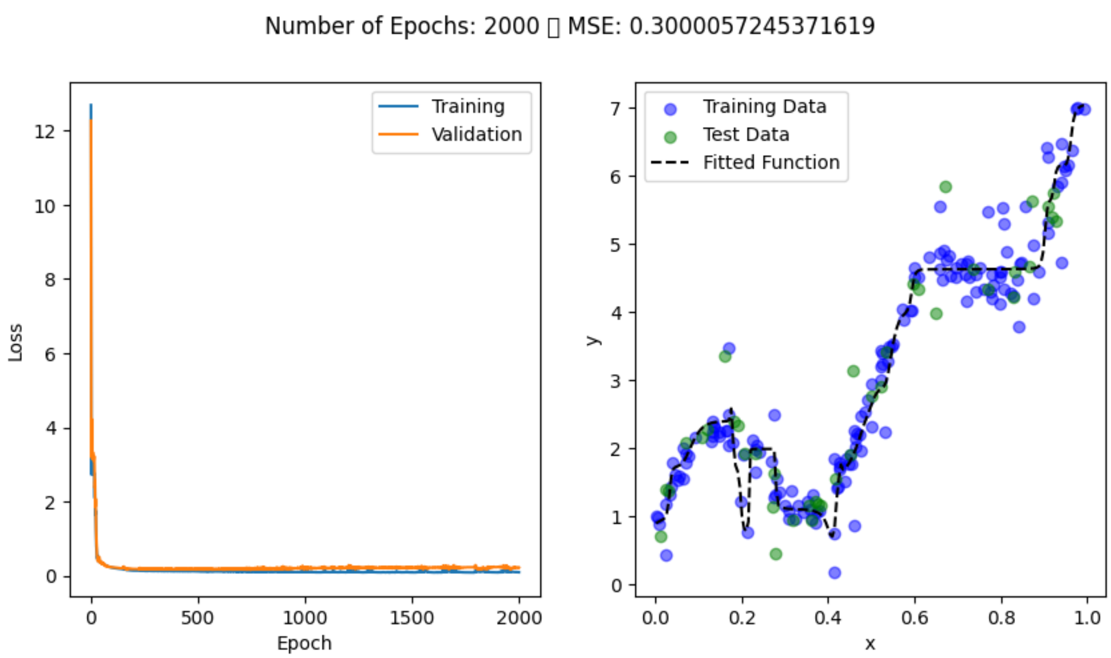
Number of Epochs: 2000 ▯ MSE: 0.15203049382591766



Number of Epochs: 2000 ▯ MSE: 0.20054423615514333







1.5 Discussion

Compare the averaged MSE result for the three different models, and comment on which number of epochs is most optimal. Why is it important that we perform cross validation when evaluating a model? For a given number of epochs, are all 5 of the k-fold models similar, or is there significant variation? Are some models underfit, overfit?

The 500 epoch trainig scheme produced the best overall fits with the lowest average MSE. This is also noticebale in the fits as well where the 100 epoch models underfit the data and the 2000 epoch fits over fit the data. Cross validation is very important for this reason because it allows for visualization as to whether we are underfitting, overfitting, or any combination. There was a clear increase in performacnce from 100 to 500 but then a clear decrease from 500 to 2000. Across the 5 k-fold models, there is slight variation. Across the board though it seems that overfitting or underfitting is pretty determinant on the number of epochs and not necessarily linked to the k-folds. Overall for each epoch number there is slight variation but not much.