Problem 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:

In [77]:
```
import torch.nn as nn
import torch

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 = 'Traini
    ax.scatter(x[idx_test], y[idx_test], c = 'green', alpha = 0.5, label = 'Test Da
```

```
    ax.plot(xs.detach().numpy(), ys.detach().numpy(), 'k--', label = 'Fitted Functi
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.legend()
```

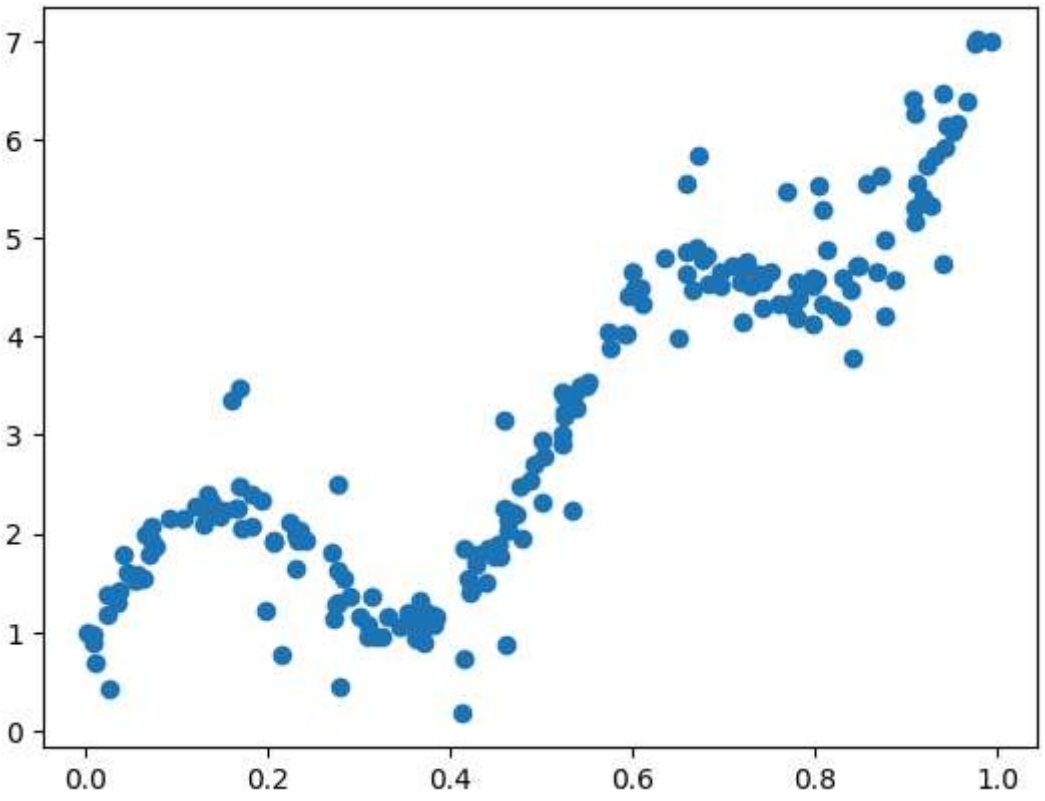
Load and visualize the data

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.

In [78]:
```
## YOUR CODE GOES HERE
dataset = np.loadtxt(r"C:\Users\zsqu4\Desktop\ML HW\ML-for-engineers\hw10\data\m10-
x = dataset[:,0]
y = dataset[:,1]
print(dataset.shape)
plt.scatter(x,y)
```

```
(200, 2)
```

Out[78]: <matplotlib.collections.PathCollection at 0x22f80e41350>



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.

In [79]:
```
## YOUR CODE GOES HERE
import torch.nn.functional as F

class NN(nn.Module):
    # YOUR CODE GOES HERE
    def __init__(self):
        super().__init__()
        input_size = 1
        output_size = 1

        self.model = nn.Sequential(
            nn.Linear(input_size,64),
            nn.Tanh(),
            nn.Linear(64,128),
            nn.Tanh(),
            nn.Linear(128,128),
            nn.Tanh(),
            nn.Linear(128,64),
            nn.Tanh(),
            nn.Linear(64,output_size)
            )
    def forward(self, xy):
        return self.model(xy)

def train(x,y,n_epoch = 100):
    x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=0.25, random_

    model = NN()
    lr= 0.01
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)
    lossfun = torch.nn.MSELoss()

    x_train_tensor = torch.tensor(x_train, dtype=torch.float32).unsqueeze(1)
    y_train_tensor = torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)
```

```
    x_val_tensor = torch.tensor(x_val, dtype=torch.float32).unsqueeze(1)
    y_val_tensor = torch.tensor(y_val, dtype=torch.float32).unsqueeze(1)

    # print(x_train_tensor.shape,y_train_tensor.shape,x_val_tensor.shape,y_val_tens

    train_curve = []
    val_curve = []

    for epoch in range(n_epoch):
        # Training step
        model.train()
        optimizer.zero_grad()
        train_pred = model(x_train_tensor)
        train_loss = lossfun(train_pred, y_train_tensor)
        train_loss.backward()
        optimizer.step()
        train_curve.append(train_loss.item())

        model.eval()
        # with torch.no_grad():
        val_pred = model(x_val_tensor)
        val_loss = lossfun(val_pred, y_val_tensor)
        val_loss.backward()
        optimizer.step()
        val_curve.append(val_loss.item())



    return model,train_curve,val_curve
```

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.

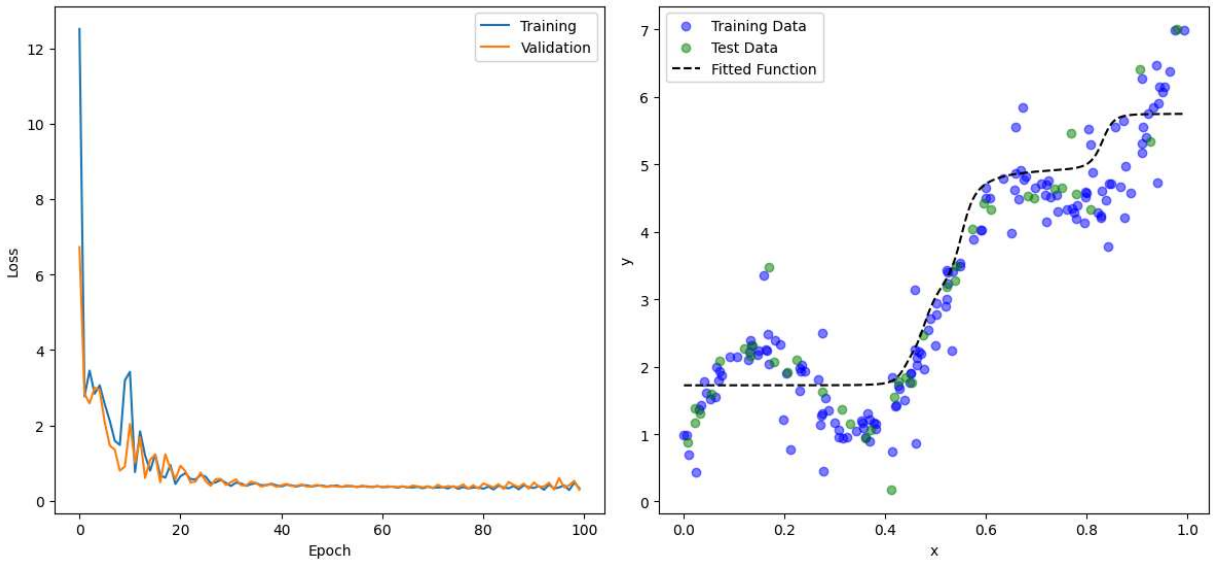
The generated figure should also be titled with the MSE of the trained model on the test data using `suptitle()` from matplotlib, such that the title is centered above the two subplots. The MSE can be computed using the `mean_squared_error` function from sklearn or `MSELoss` from PyTorch.

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

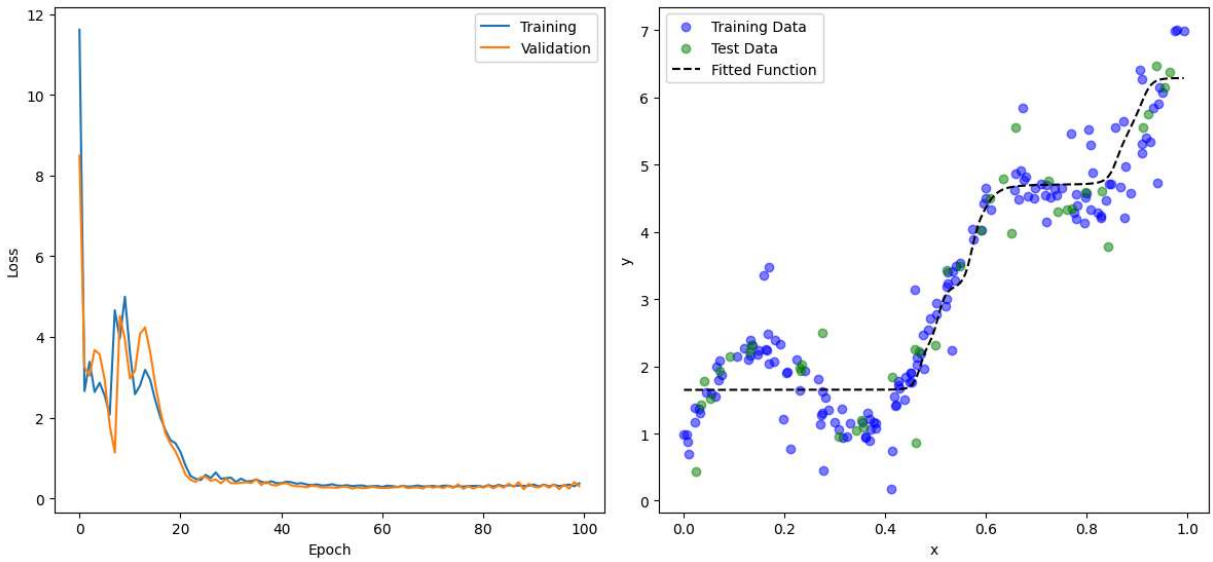
Since there are three models and we are using 5-fold cross validation, you should output 15 figures, with two subplots each.

In [80]:
```
## YOUR CODE GOES HERE

folds = KFold(n_splits=5 ,random_state = 62,shuffle=True)
epochs = [100,500,2000]
mse_res = []
for epoch in epochs:
    mse_loss = []
    for fold, (train_idx, test_idx) in enumerate(folds.split(x)):
        x_train, x_test = x[train_idx], x[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]

        # print(x_train.shape,y_train.shape)
        model, train_curve, val_curve = train(x_train, y_train, n_epoch=epoch)

        x_train_tensor = torch.tensor(x_train, dtype=torch.float32).unsqueeze(1)
        y_train_tensor = torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)
        x_test_tensor = torch.tensor(x_test, dtype=torch.float32).unsqueeze(1)
        y_test_tensor = torch.tensor(y_test, dtype=torch.float32).unsqueeze(1)

        y_pred = model(x_test_tensor).detach().numpy()
        mse = mean_squared_error(y_test, y_pred)
        mse_loss.append(mse)

        fig, ax = plt.subplots(1, 2, figsize=(12, 6))
        plotLoss(ax[0],train_curve,val_curve)


        plotModel(ax[1], model, x, y, train_idx, test_idx)

        fig.suptitle(f"{epoch} Epochs | Fold {fold+1} | MSE: {mse:.4f}", fontsize=1
        plt.tight_layout()
        plt.show()

    avg_mse = np.mean(mse_loss)
    mse_res.append(avg_mse)
```

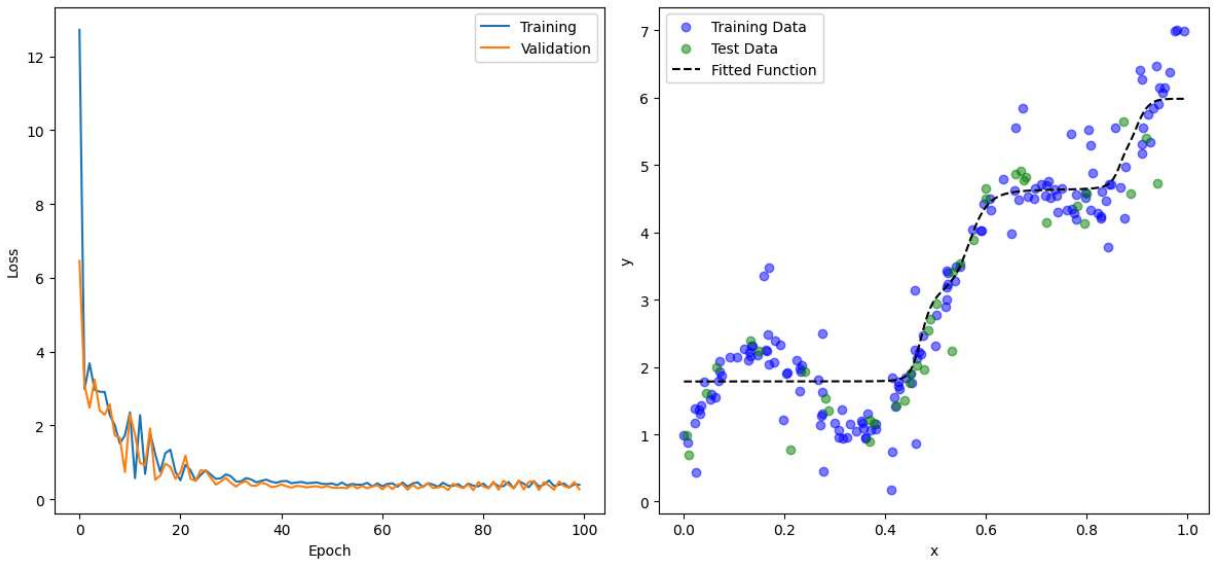
100 Epochs | Fold 1 | MSE: 0.3555



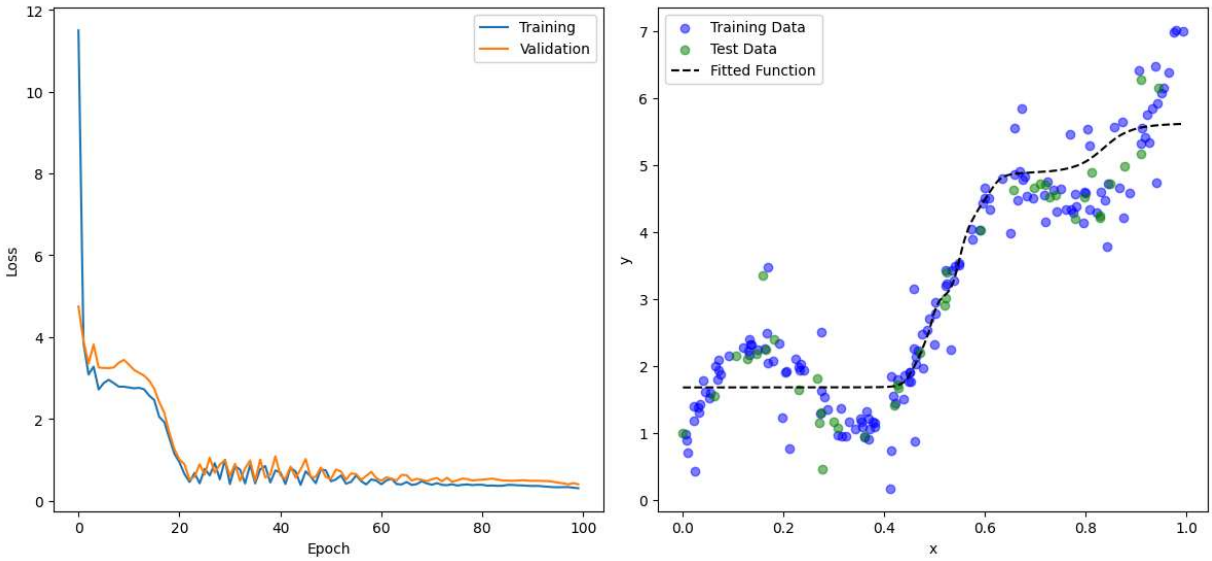
100 Epochs | Fold 2 | MSE: 0.2443



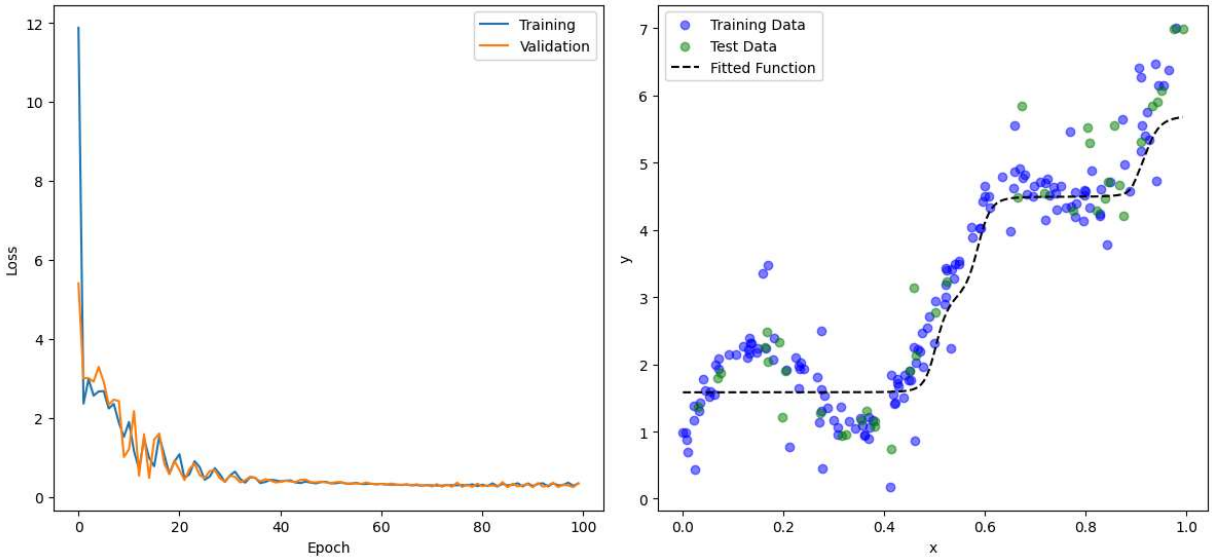
100 Epochs | Fold 3 | MSE: 0.2675



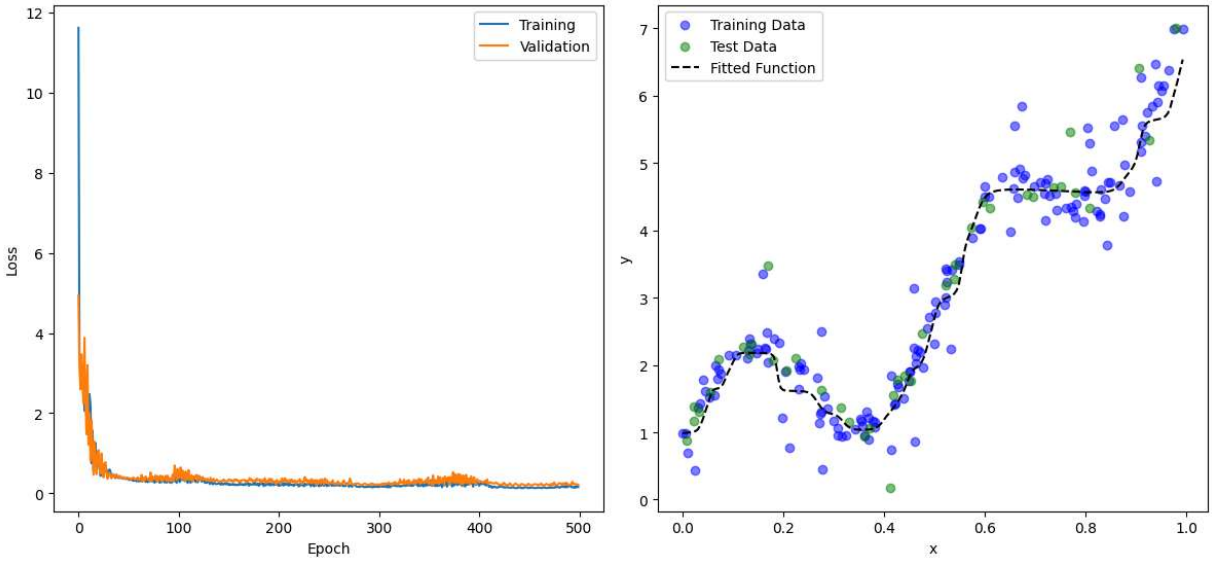
100 Epochs | Fold 4 | MSE: 0.3286



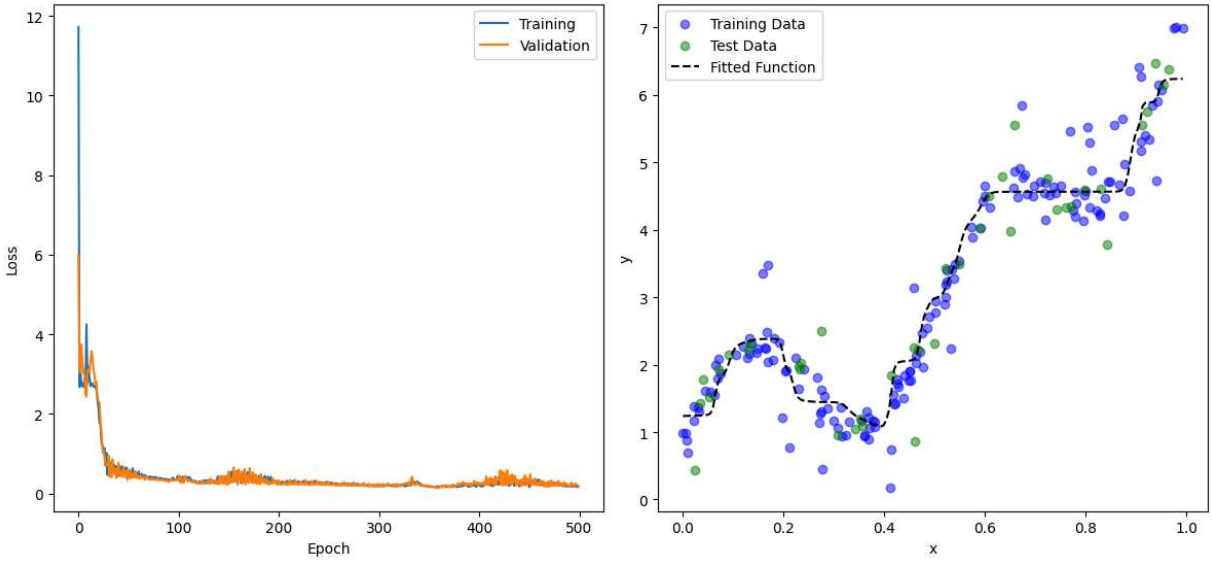
100 Epochs | Fold 5 | MSE: 0.4273



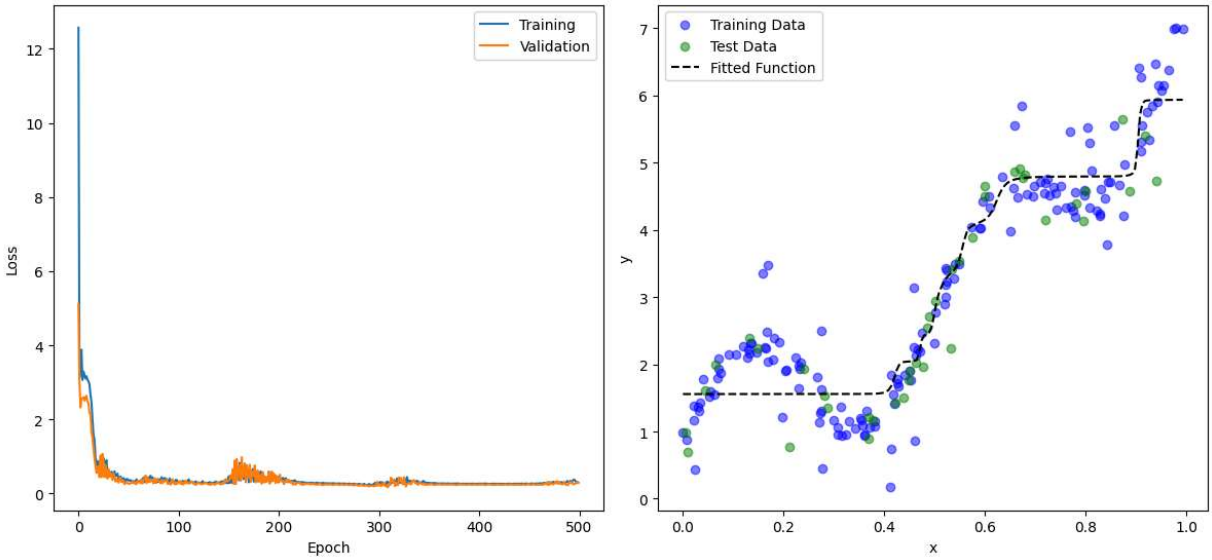
500 Epochs | Fold 1 | MSE: 0.1918



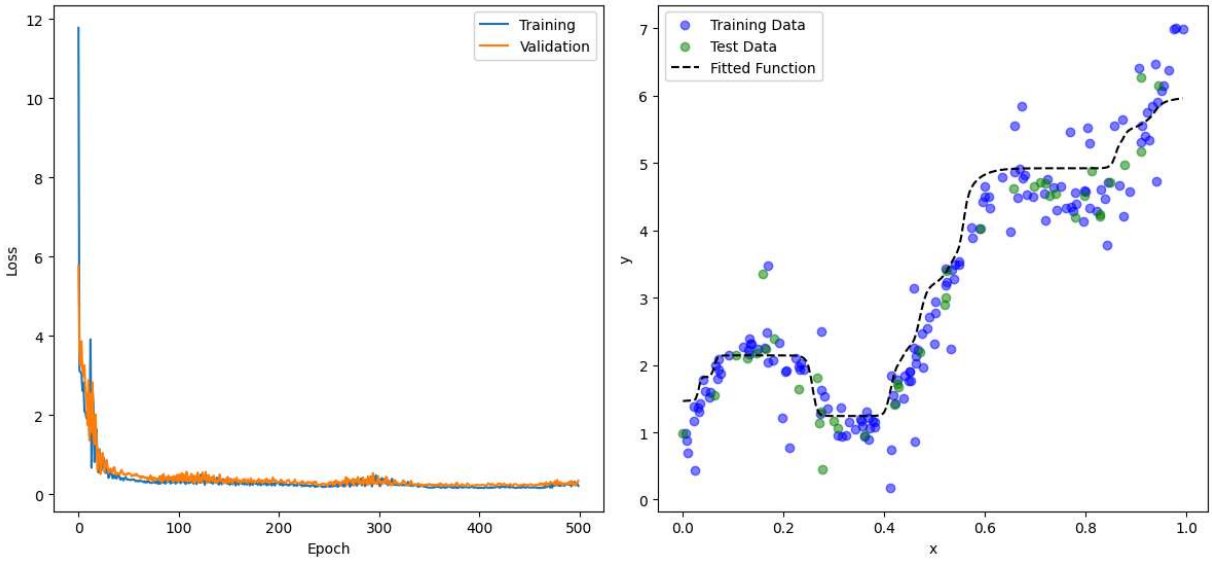
500 Epochs | Fold 2 | MSE: 0.2006



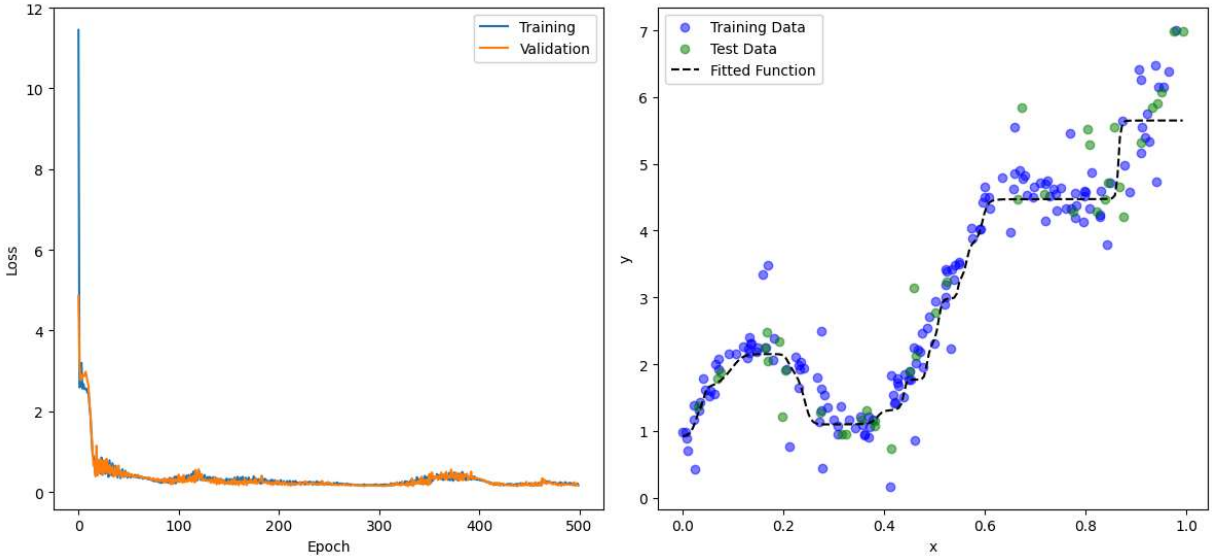
500 Epochs | Fold 3 | MSE: 0.2500



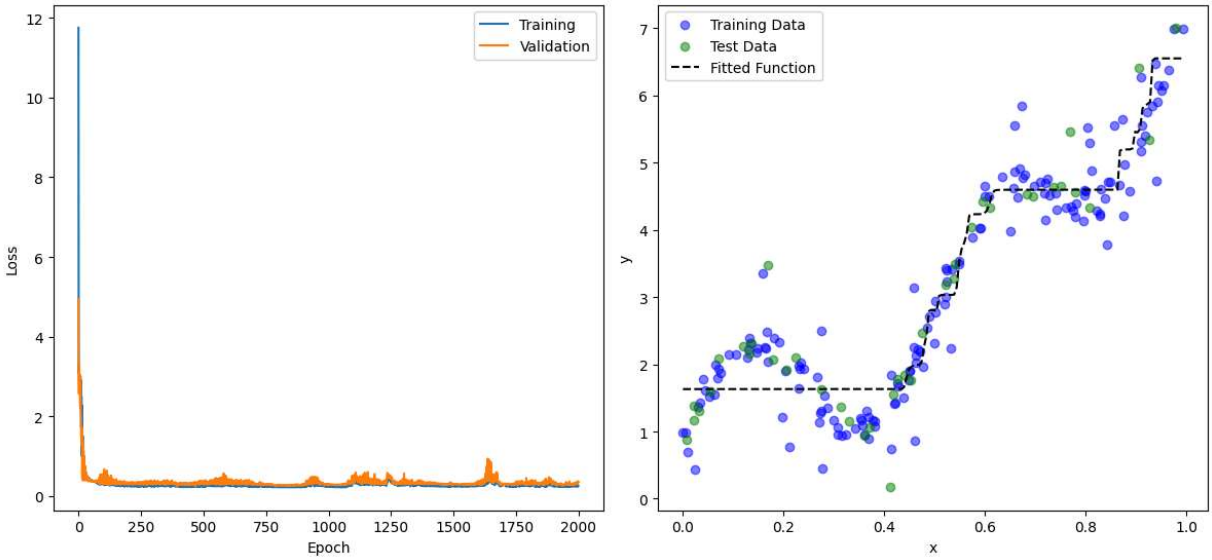
500 Epochs | Fold 4 | MSE: 0.2009



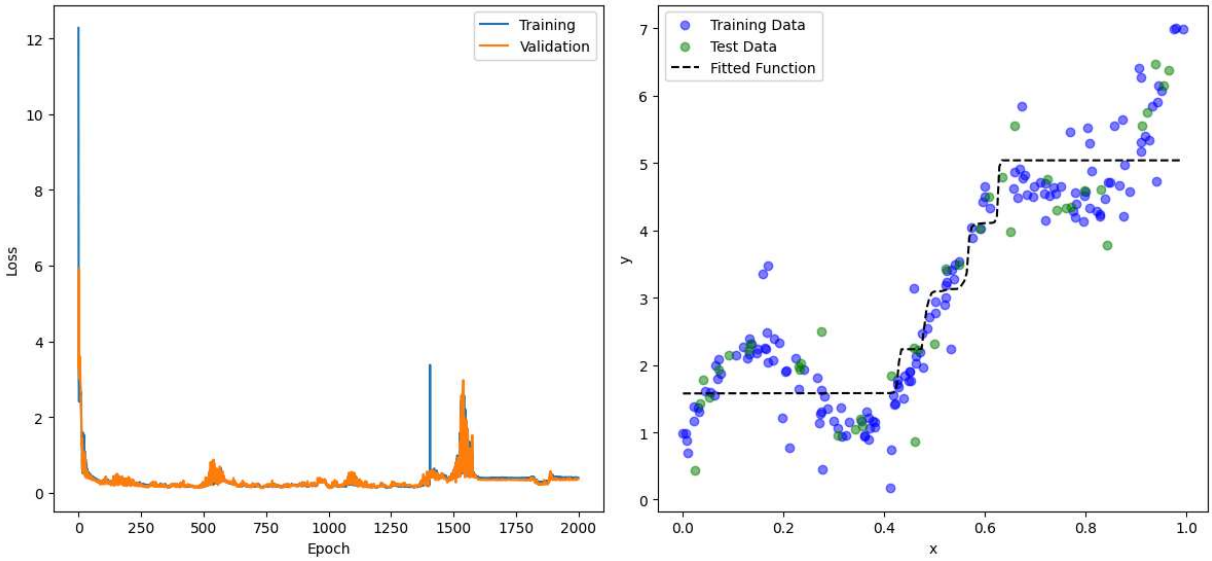
500 Epochs | Fold 5 | MSE: 0.3740



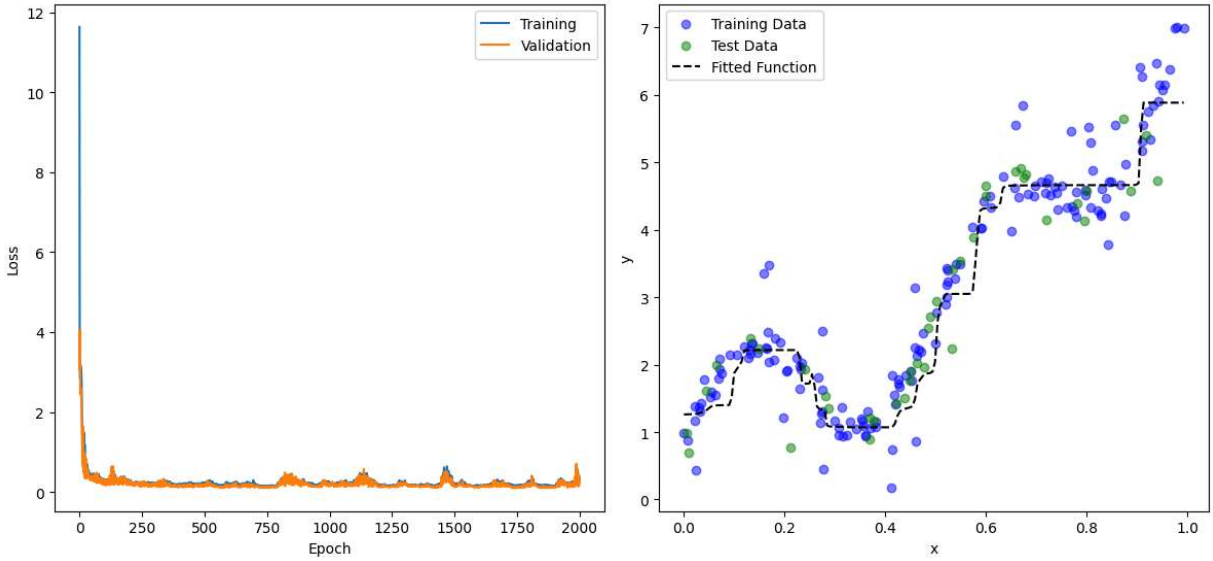
2000 Epochs | Fold 1 | MSE: 0.3060



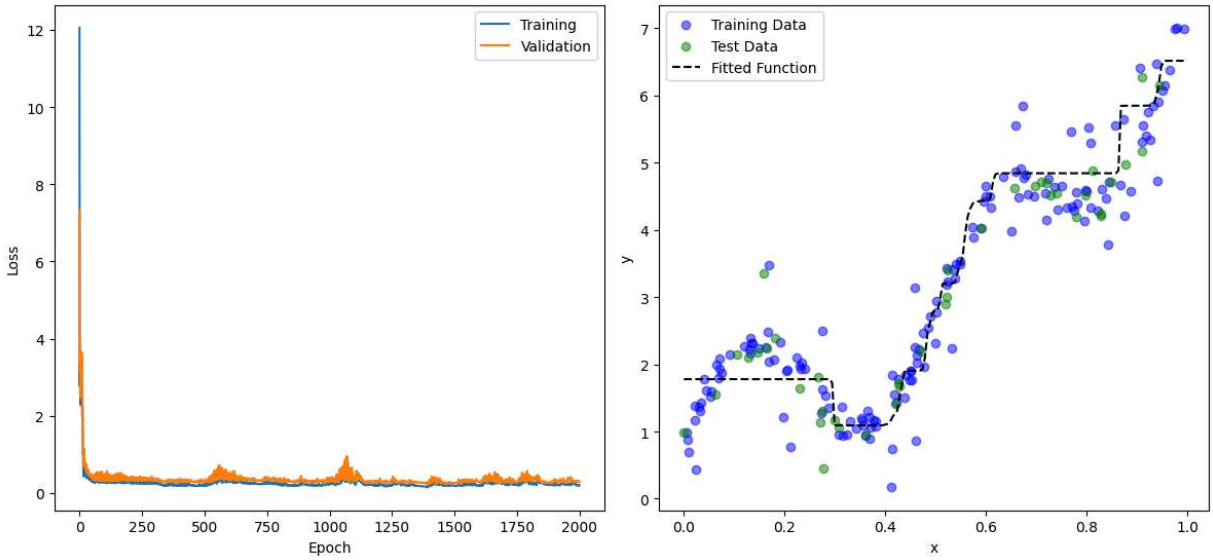
2000 Epochs | Fold 2 | MSE: 0.4750



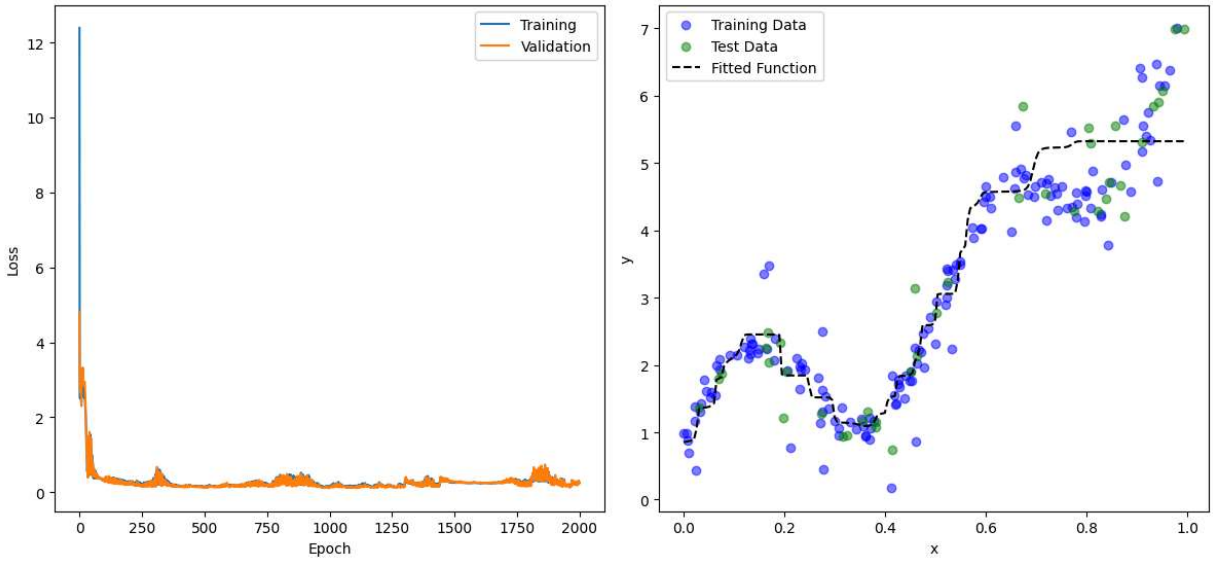
2000 Epochs | Fold 3 | MSE: 0.2667



2000 Epochs | Fold 4 | MSE: 0.2546



2000 Epochs | Fold 5 | MSE: 0.4130



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?

Your response goes here

Across all the models, I think the one trained with 500 epochs is the best. It has a low error and the fitted function curve is very smooth, which means it's not underfitting and overfitting. With only 100 epochs, the model is underfitting, which is indicated by the higher error. On the other hand, with 2000 epochs the model is overfitting. This is shown by the acute angles in the fitted function curve and the fluctuating error.

Cross validation ensures the performance of the model is not biased by train and test the model with different subsets of data. It reduces the risk of overfitting and maximizes the potential of the dataset.

Not all k-fold models are the same, indicating that the data is not super well balanced.