24-787: Machine Learning and Artificial Intelligence for Engineers
Ryan Wu
ID: weihuanw
Homework 10
Due: April 06, 2024

Concept Questions:

Problem 1
1. Model 1 is low bias but high variance, Model 2 is low variance but high bias.

Problem 2
4. K-fold cross validation partitions the data into k equal sized subsets, and trains k models, each time using on subset as the validation data and the rest as the training data

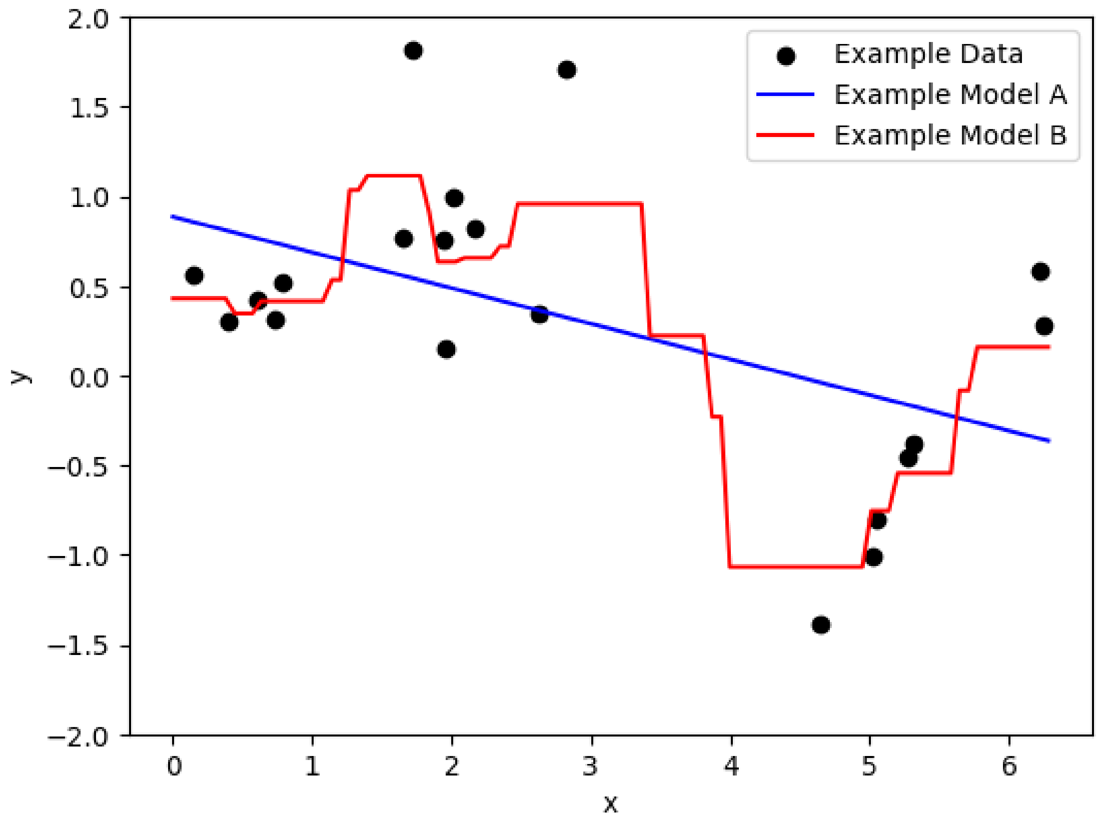
M10-L1 Problem 1

In this problem you will look compare models with lower/higher variance/bias by computing bias and variance at a single point.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor

def plot_model(model,color="blue"):
    x = np.linspace(0, np.pi*2, 100)
    y = model.predict(x.reshape(-1,1))
    plt.plot(x, y, color=color)
    plt.xlabel("x")
    plt.ylabel("y")

def plot_data(x, y):
    plt.scatter(x,y,color="black")

def eval_model_at_point(model, x):
    return model.predict(np.array([[x]])).item()

def train_models():
    x = np.random.uniform(0,np.pi*2,20).reshape(-1,1)
    y = np.random.normal(np.sin(x),0.5).flatten()

    modelA = LinearRegression()
    modelB = KNeighborsRegressor(3)
    modelA.fit(x,y)
    modelB.fit(x,y)
    return modelA, modelB, x, y
```

The function `train_models` gets 20 new data points and trains two models on these data points. Model A is a linear regression model, while model B is a 3-nearest neighbor regressor.

```python
modelA, modelB, x, y = train_models()
plt.figure()
plot_data(x,y)
plot_model(modelA,"blue")
plot_model(modelB,"red")
plt.legend(["Example Data", "Example Model A", "Example Model B"])
plt.ylim([-2,2])
plt.show()
```



Training models

First, train 50 instances of model A and 50 instances of model B. Store all 100 total models for use in the next few cells. Generate these models with the function: `modelA, modelB, x, y = train_models()`.

```python
# YOUR CODE GOES HERE
# train 50 instances of modelA and modelB

modelA_list = []
modelB_list = []

for i in range(50):
    modelA, modelB, x, y = train_models()
    modelA_list.append(modelA)
    modelB_list.append(modelB)
```

Bias and Variance

Now we will use the definitions of bias and variance to compute the bias and variance of each type of model. You will focus on the point x = 1.57 only. First, compute the prediction for each model at x. (You can use the function `eval_model_at_point(model, x)`).

```
x = 1.57

# YOUR CODE GOES HERE
# evaluate the models at x = 1.57
modelA_values = []
modelB_values = []

for i in range(50):
    modelA_values.append(eval_model_at_point(modelA_list[i], x))
    modelB_values.append(eval_model_at_point(modelB_list[i], x))
```

In this cell, use the values you computed above to compute and print the bias and variance of model A at the point x = 1.57. The true function value `y_GT` is given as 1 for x=1.57.

```
yGT = 1

# YOUR CODE GOES HERE
# calculate the bias and variance of modelA and modelB
bias_A = np.mean(np.array(modelA_values)) - yGT
bias_B = np.mean(np.array(modelB_values)) - yGT

var_A = np.var(np.array(modelA_values))
var_B = np.var(np.array(modelB_values))

print(f"Model A:   Bias = {bias_A:.3f},  Variance = {var_A:.3f}")
print(f"Model B:   Bias = {bias_B:.3f},  Variance = {var_B:.3f}")
```

```
Model A:   Bias = -0.524,  Variance = 0.035
Model B:   Bias = -0.076,  Variance = 0.096
```

Questions

1. Which model has smaller bias at $x = 1.57$?

   At $x = 1.57$, Model B has a smaller bias at -0.076.

2. Which model has lower variance at $x = 1.57$?

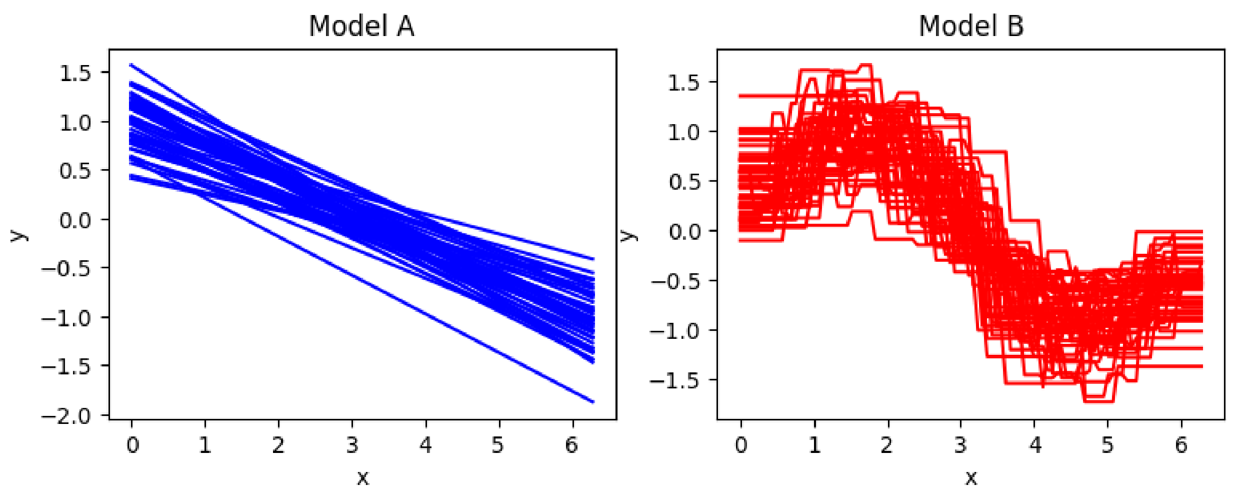
   At $x = 1.57$, Model A has a smaller variance at 0.035.

Plotting models

Now use the `plot_model` function to overlay all Model A predictions on one plot and all Model B predictions on another. Notice the spread of each model.

```
plt.figure(figsize=(9,3))

plt.subplot(1,2,1)
plt.title("Model A")
# YOUR CODE GOES HERE
```

```
for i in range(50):
    plot_model(modelA_list[i], "blue")


plt.subplot(1,2,2)
plt.title("Model B")
# YOUR CODE GOES HERE
for i in range(50):
    plot_model(modelB_list[i], "red")

plt.show()
```

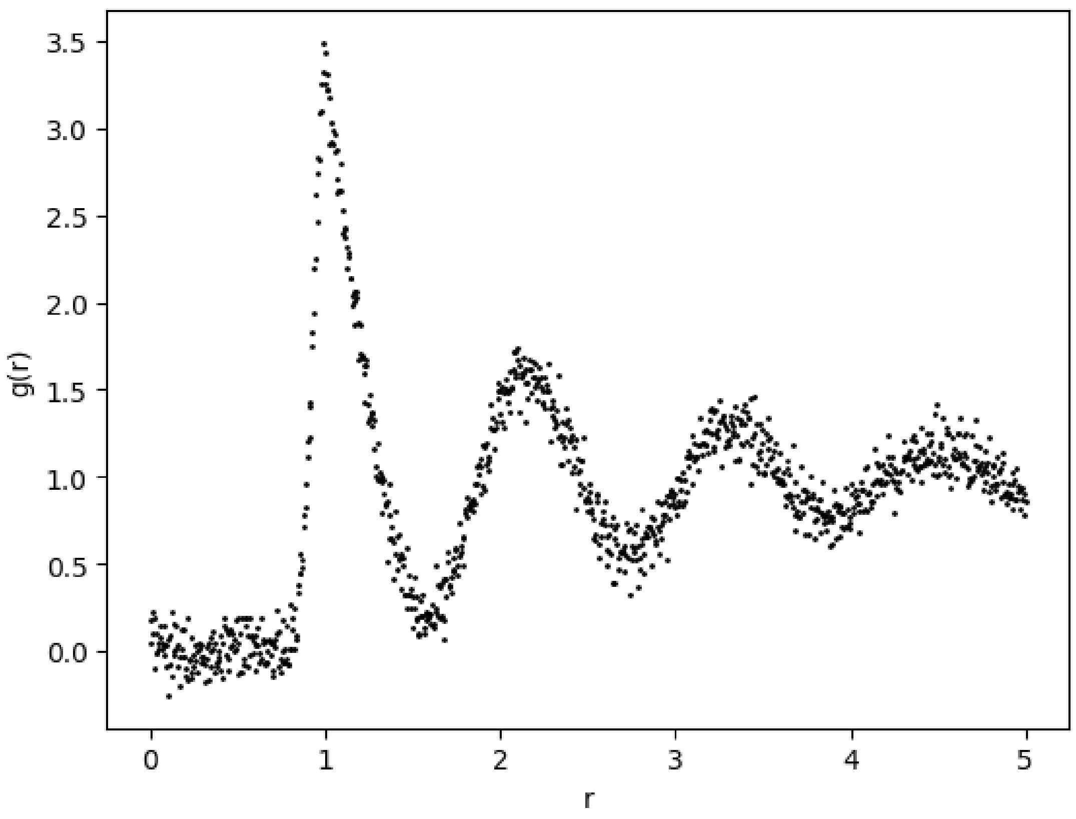


M10-L2 Problem 1

In this problem, you will perform 10-fold cross validation to find the best of 3 regression models.

You are given a dataset with testing and training data of another radial distribution function (measuring 'g(r)', the probability of a particle being a certain distance 'r' from another particle): `X_train, X_test, y_train, y_test`

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split, KFold
from sklearn.base import clone

def get_gr(r):
    a, b, L, m, t, d = 0.54, 5.4, 1.2, 7.4, 100, 3.3
    g1 = 1 + (r+1e-9)**(-m) * (d-1-L) + (r-1+L)/(r+1e-9)*np.exp(-a*(r-
1))*np.cos(b*(r-1))
    g2 = d * np.exp(-t*(r-1)**2)
    g = g1*(r>=1) + g2*(r<1)
    return g

def plot_model(model,color="blue"):
    x = np.linspace(0, 5, 1000)
    y = model.predict(x.reshape(-1,1))
    plt.plot(x, y, color=color, linewidth=2, zorder=2)
    plt.xlabel("r")
    plt.ylabel("g(r)")

def plot_data(x, y):
    plt.scatter(x,y,s=1, color="black")
    plt.xlabel("r")
    plt.ylabel("g(r)")


np.random.seed(0)
X = np.linspace(0,5,1000).reshape(-1,1)
y = np.random.normal(get_gr(X.flatten()),0.1)

X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=0, train_size=800)


plt.figure()
plot_data(X,y)
plt.show()
```



Models

Below we define 3 sklearn neural network models `model1`, `model2`, and `model3`. Your goal is to find which is best using 10-fold cross-validation.

```
model1 = MLPRegressor([24], random_state=0, activation="tanh",
max_iter=1000)
model2 = MLPRegressor([48,48], random_state=0, activation="tanh",
max_iter=1000)
model3 = MLPRegressor([64,64, 64], random_state=0, activation="relu",
max_iter=1000)


models = [model1, model2, model3]
for model in models:
    model.fit(X_train, y_train)
```

Cross-validation folds

This cell creates 10-fold iterator objects in sklearn. Make note of how this is done.

We also provide code for computing the cross-validation score for average R^2 over validation folds. Note that the model is retrained on each fold, and weights/biases are reset each time with `sklearn.base.clone()`

```
folds = KFold(n_splits=10,random_state=0,shuffle=True)


scores1 = []
for train_idx, val_idx in folds.split(X_train):
    model1 = clone(model1)
    model1.fit(X_train[train_idx,:],y_train[train_idx])
    score = model1.score(X_train[val_idx,:],y_train[val_idx])
    scores1.append(score)
    print(f"Validation score: {score}")


score1 = np.mean(np.array(scores1))
print(f"Average validation score for Model 1: {score1}")
```

```
Validation score: 0.17567883199274337
Validation score: 0.19279856417941255
Validation score: 0.277493724970579
Validation score: 0.3104352357647894
Validation score: 0.20608404129798263
Validation score: 0.03790122395449669
Validation score: 0.1676244803676995
Validation score: 0.22025003724477432
Validation score: 0.14423712046918646
Validation score: 0.19894361702001595
Average validation score for Model 1: 0.193144687726168
```

Your turn: validating models 2 and 3

Now follow the same procedure to get the average R^2 scores for `model2` and `model3` on validation folds. You can use the same KFold iterator.

```
# YOUR CODE GOES HERE
# model 2 validation
scores2 = []
for train_idx, val_idx in folds.split(X_train):
    model2 = clone(model2)
    model2.fit(X_train[train_idx,:],y_train[train_idx])
    score = model2.score(X_train[val_idx,:],y_train[val_idx])
    scores2.append(score)
    print(f"Validation score: {score}")

score2 = np.mean(np.array(scores2))
print(f"Average validation score for Model 2: {score2}")
```

```
# model 3 validation
scores3 = []
for train_idx, val_idx in folds.split(X_train):
    model3 = clone(model3)
    model3.fit(X_train[train_idx,:],y_train[train_idx])
    score = model3.score(X_train[val_idx,:],y_train[val_idx])
    scores3.append(score)
    print(f"Validation score: {score}")

score3 = np.mean(np.array(scores3))
print(f"Average validation score for Model 3: {score3}")
```

```
Validation score: 0.9135256064394238
Validation score: 0.92381162019413
Validation score: 0.9109428377428712
Validation score: 0.916683295227516
Validation score: 0.8980936123083956
Validation score: 0.9208009063665946
Validation score: 0.9123834705950664
Validation score: 0.8780032287365068
Validation score: 0.9281564779069267
Validation score: 0.95771300087561
Average validation score for Model 2: 0.9160114056393042
Validation score: 0.9629033605148654
Validation score: 0.9466107686883457
Validation score: 0.9518315048355763
Validation score: 0.9514051770741325
Validation score: 0.9229643307655356
Validation score: 0.9501422202077937
Validation score: 0.9322229519501164
Validation score: 0.9238931238090651
Validation score: 0.9461292855545796
Validation score: 0.9611128180031758
Average validation score for Model 3: 0.9449215541403186
```

Comparing models

Which model had the best performance according to your validation study?

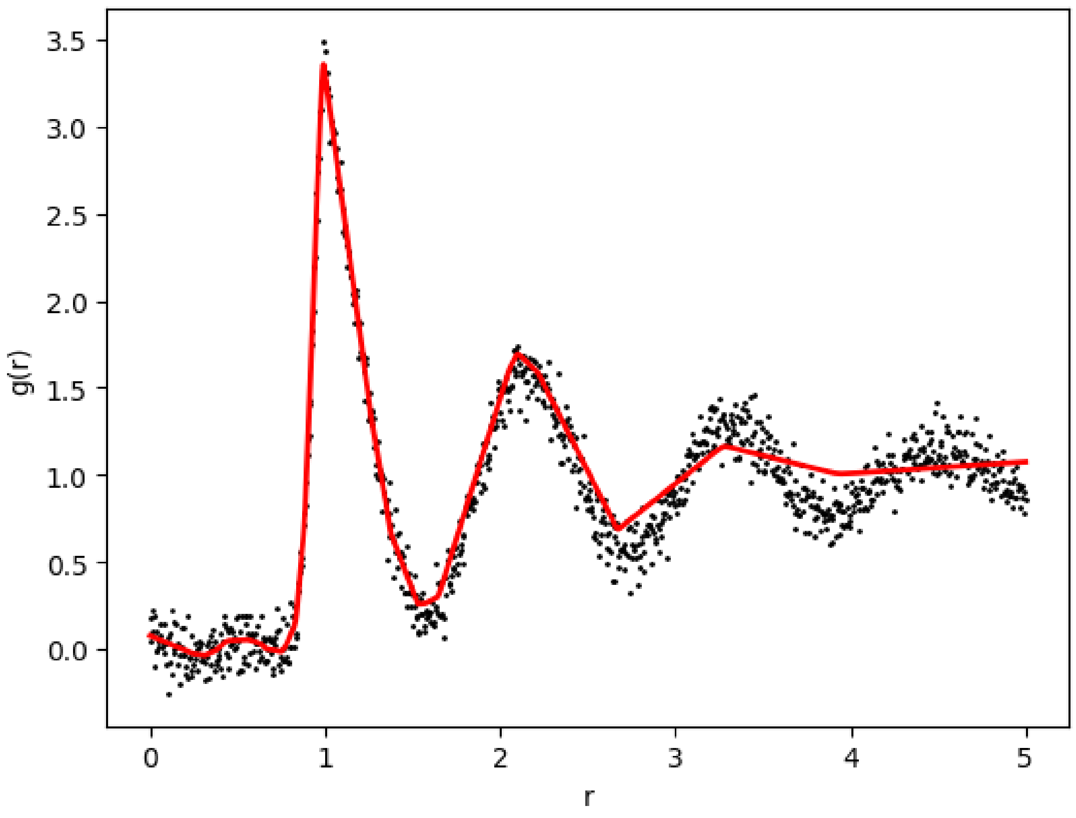
```
From the validation study, Model 3 had the best performance.
```

Retrain this model on the full training dataset and report the R2 score on training and testing data. Then complete the code to plot the model prediction with the data using the `plot_model` function.

```
# YOUR CODE GOES HERE
# retrain Model 3 on the full training set
```

```
model3.fit(X_train, y_train)
r2_score_test = model3.score(X_test, y_test)
r2_score_train = model3.score(X_train, y_train)
print(f"R2 score for Model 3 testing data: {r2_score_test}")
print(f"R2 score for Model 3 training data: {r2_score_train}")


plt.figure()
plot_data(X,y)

# YOUR CODE GOES HERE
plot_model(model3, color="red")

plt.show()
```

```
R2 score for Model 3 testing data: 0.9371864974414208
R2 score for Model 3 training data: 0.9547034601442719
```



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.

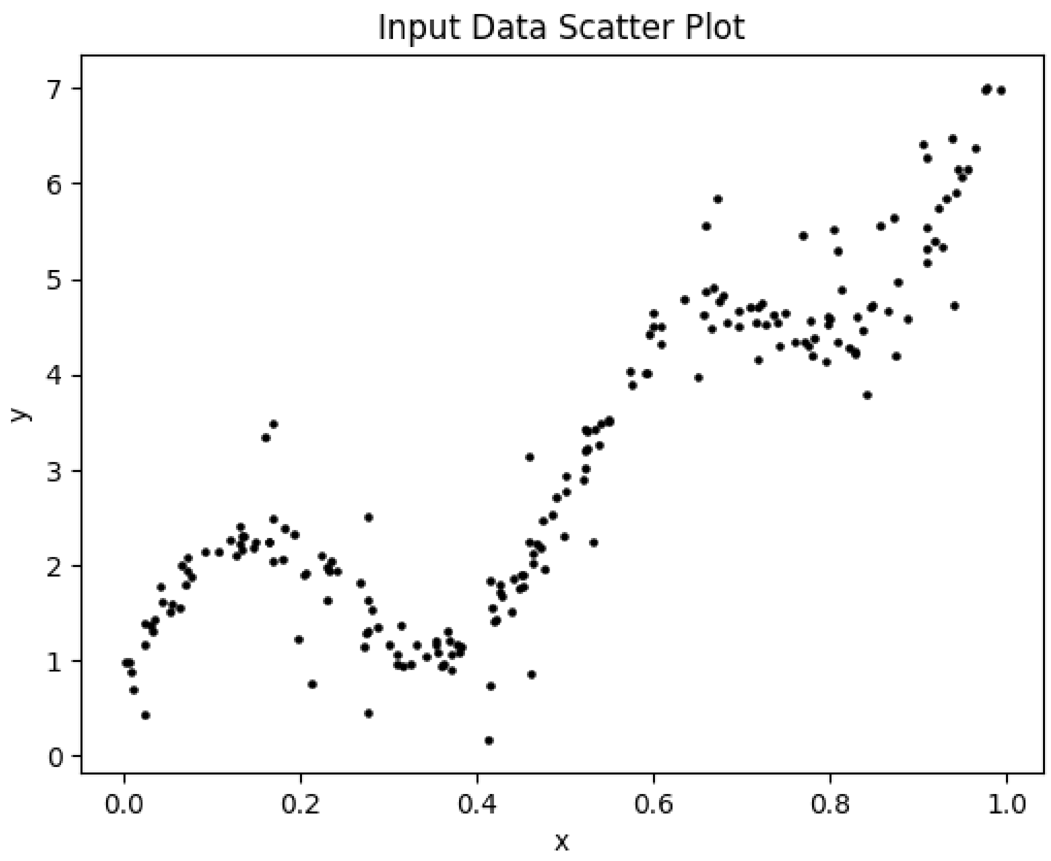
Imports and Utility Functions:

```
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 = '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()
```

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.

```
## YOUR CODE GOES HERE
data = np.loadtxt('data/m10-hw1-data.txt')
# data first column is x, second column is y
x = torch.tensor(data[:,0]).reshape(-1,1)
y = torch.tensor(data[:,1]).reshape(-1,1)

# plot the data using scatter plot
plt.scatter(x,y,s=5, color="black")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Input Data Scatter Plot")
plt.show()
```



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.

```
## YOUR CODE GOES HERE
from torch import optim
import torch.nn.functional as F
```

```
# fully connected laters and tanh activation function, no activation
function for the output layer
# 4 hidden layers with [64, 128, 128, 64]

class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(1, 64)
        self.fc2 = nn.Linear(64, 128)
        self.fc3 = nn.Linear(128, 128)
        self.fc4 = nn.Linear(128, 64)
        self.fc5 = nn.Linear(64, 1)
        self.activation = nn.Tanh()

    def forward(self, x):
        x = self.activation(self.fc1(x))
        x = self.activation(self.fc2(x))
        x = self.activation(self.fc3(x))
        x = self.activation(self.fc4(x))
        x = self.fc5(x)
        return x

    def trainModel(x, y, n_epoch):
        # split data into training and testing
        x_train, x_val, y_train, y_val = train_test_split(x, y,
test_size=0.25)
        # convert to tensor
        x_train_tensor = torch.tensor(x_train).reshape(-1,1).float()
        x_val_tensor = torch.tensor(x_val).reshape(-1,1).float()
        y_train_tensor = torch.tensor(y_train).reshape(-1,1).float()
        y_val_tensor = torch.tensor(y_val).reshape(-1,1).float()

        model = NeuralNetwork()
        lossfun = nn.MSELoss()
        opt = optim.Adam(model.parameters(), lr=0.01)

        train_curve = []
        val_curve = []

        for epoch in range(n_epoch):
            model.train()
            opt.zero_grad()
            train_pred = model(x_train_tensor)
            train_loss = lossfun(train_pred, y_train_tensor)
            train_loss.backward()
            val_pred = model(x_val_tensor)
            val_loss = lossfun(val_pred, y_val_tensor)
            val_loss.backward()
            opt.step()
            train_curve.append(train_loss.item())
            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.

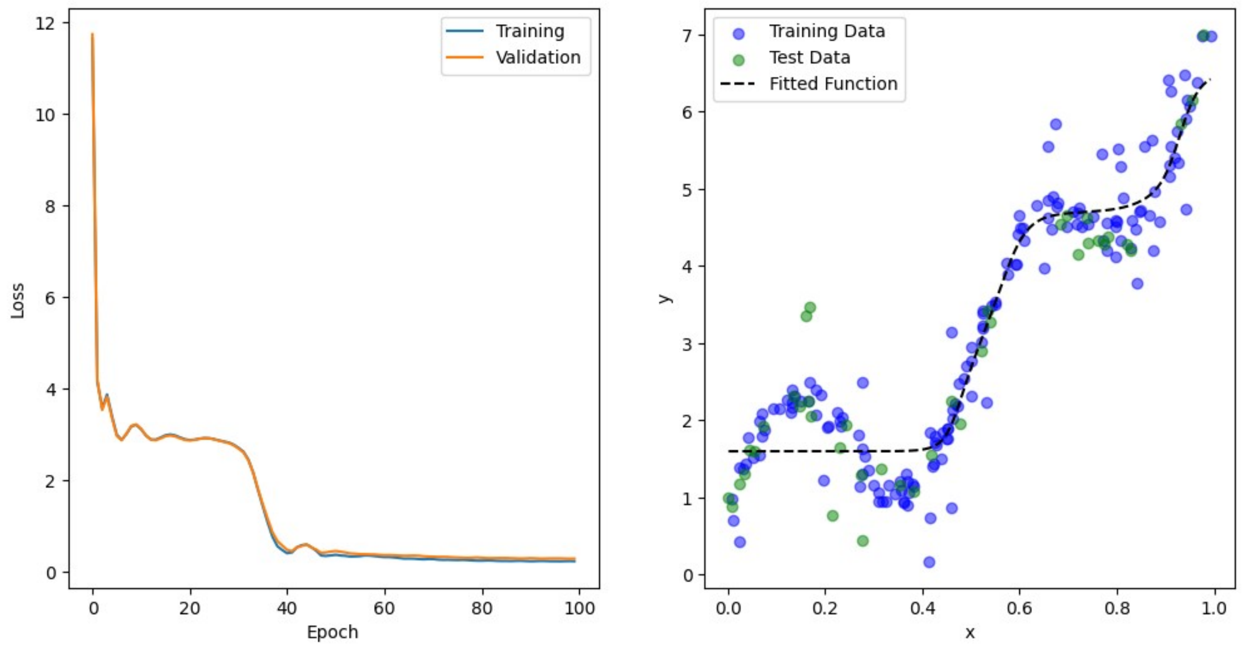
```
## YOUR CODE GOES HERE
from sklearn.base import clone

def crossValidation(x, y):
    folds = KFold(n_splits=5, random_state=0, shuffle=True)
    n_epochs = [100, 500, 2000]
    lossfun = nn.MSELoss()
    models_mse = []

    for n_epoch in n_epochs:
        model_mses = []
        for train_idx, test_idx in folds.split(x):
            x_train, x_test = x[train_idx], x[test_idx]
            y_train, y_test = y[train_idx], y[test_idx]
            x_train_tensor = torch.tensor(x_train).reshape(-
1,1).float()
            x_test_tensor = torch.tensor(x_test).reshape(-1,1).float()
            y_train_tensor = torch.tensor(y_train).reshape(-
1,1).float()
```

```
            model, train_curve, val_curve =
NeuralNetwork.trainModel(x_train, y_train, n_epoch)
            model.eval()
            yhat = model(x_test_tensor)
            mse = lossfun(yhat, y_test).item()
            model_mses.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)
            plt.suptitle(f'MSE of the Trained Model on the Test Data:
{model_mses[-1]:.4f}')
            plt.show()

        models_mse.append(np.mean(model_mses))

    return models_mse

# cross validation for 100, 500, 2000 epochs
n_epochs = [100, 500, 2000]
models_mse = crossValidation(x, y)
for i, mse in enumerate(models_mse):
    print(f"MSE for model trained for {n_epochs[i]} epochs:
{mse:.4f}")
# print("MSE for each model (100, 500, 2000):", models_mse)
```

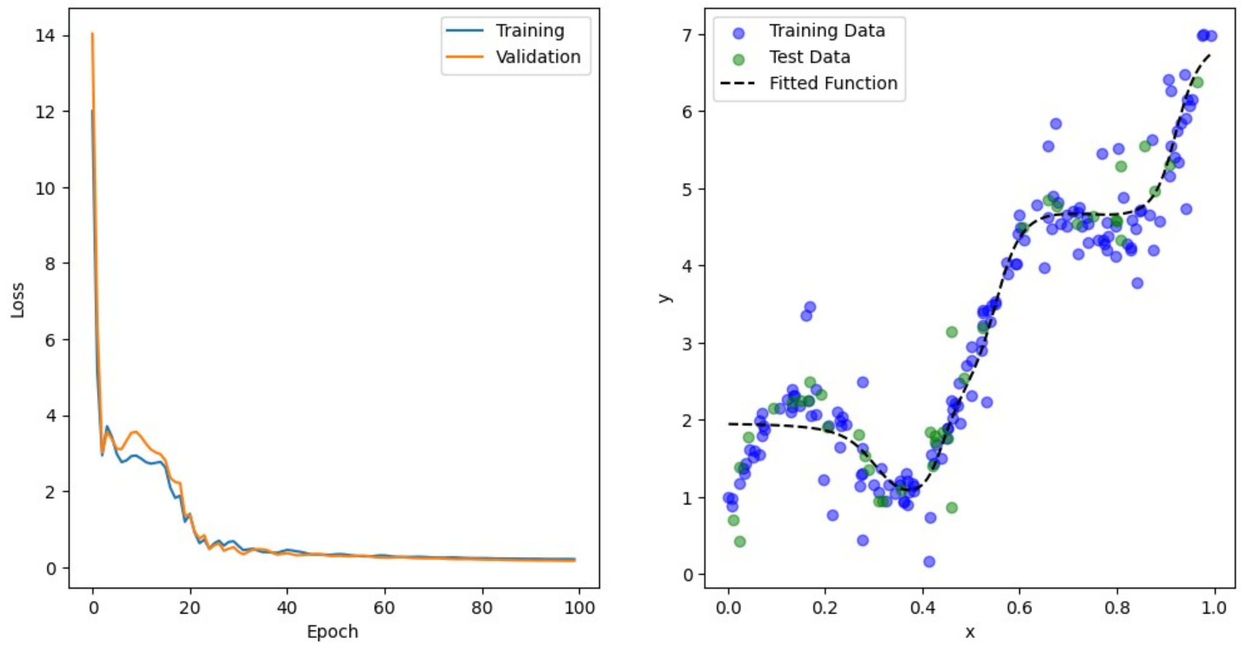
```
/var/folders/nv/06hggmyd00v8n19qbkz7vxh00000gn/T/
ipykernel_87963/3682784502.py:15: UserWarning: To copy construct from
a tensor, it is recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  x_train_tensor = torch.tensor(x_train).reshape(-1,1).float()
/var/folders/nv/06hggmyd00v8n19qbkz7vxh00000gn/T/ipykernel_87963/36827
84502.py:16: UserWarning: To copy construct from a tensor, it is
recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  x_test_tensor = torch.tensor(x_test).reshape(-1,1).float()
/var/folders/nv/06hggmyd00v8n19qbkz7vxh00000gn/T/ipykernel_87963/36827
84502.py:17: UserWarning: To copy construct from a tensor, it is
recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  y_train_tensor = torch.tensor(y_train).reshape(-1,1).float()
/var/folders/nv/06hggmyd00v8n19qbkz7vxh00000gn/T/ipykernel_87963/42742
74295.py:30: UserWarning: To copy construct from a tensor, it is
recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
```

```
  x_train_tensor = torch.tensor(x_train).reshape(-1,1).float()
/var/folders/nv/06hggmyd00v8n19qbkz7vxh00000gn/T/ipykernel_87963/42742
74295.py:31: UserWarning: To copy construct from a tensor, it is
recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  x_val_tensor = torch.tensor(x_val).reshape(-1,1).float()
/var/folders/nv/06hggmyd00v8n19qbkz7vxh00000gn/T/ipykernel_87963/42742
74295.py:32: UserWarning: To copy construct from a tensor, it is
recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  y_train_tensor = torch.tensor(y_train).reshape(-1,1).float()
/var/folders/nv/06hggmyd00v8n19qbkz7vxh00000gn/T/ipykernel_87963/42742
74295.py:33: UserWarning: To copy construct from a tensor, it is
recommended to use sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  y_val_tensor = torch.tensor(y_val).reshape(-1,1).float()
```

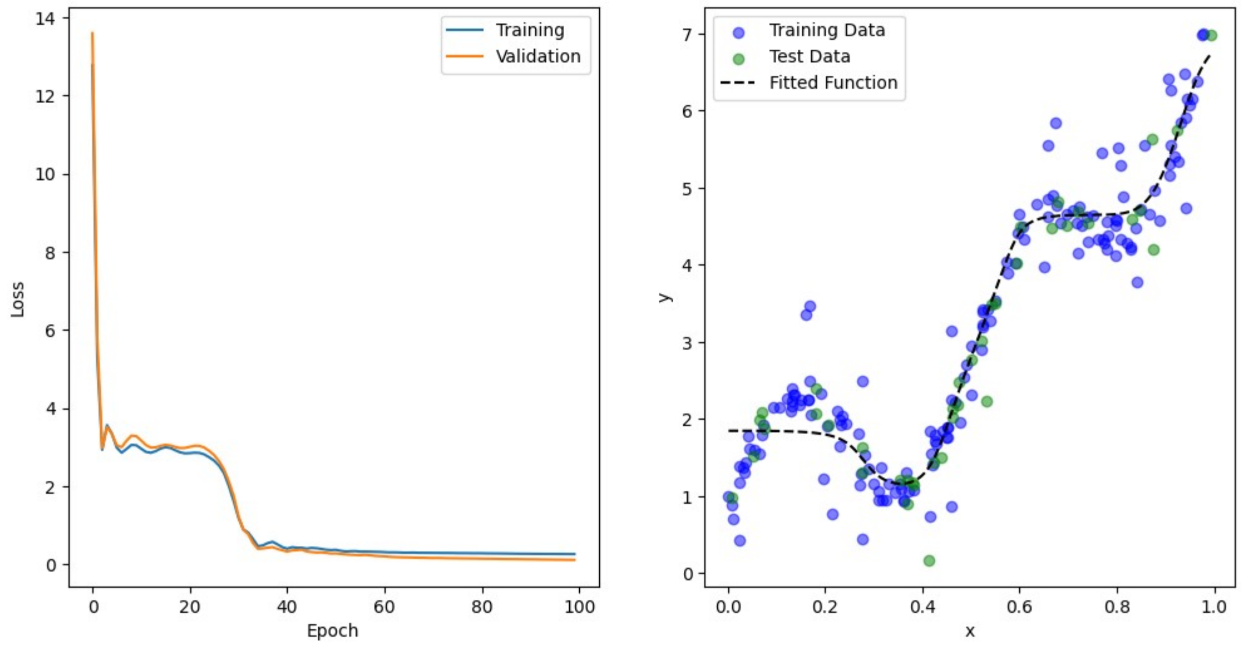
MSE of the Trained Model on the Test Data: 0.3563



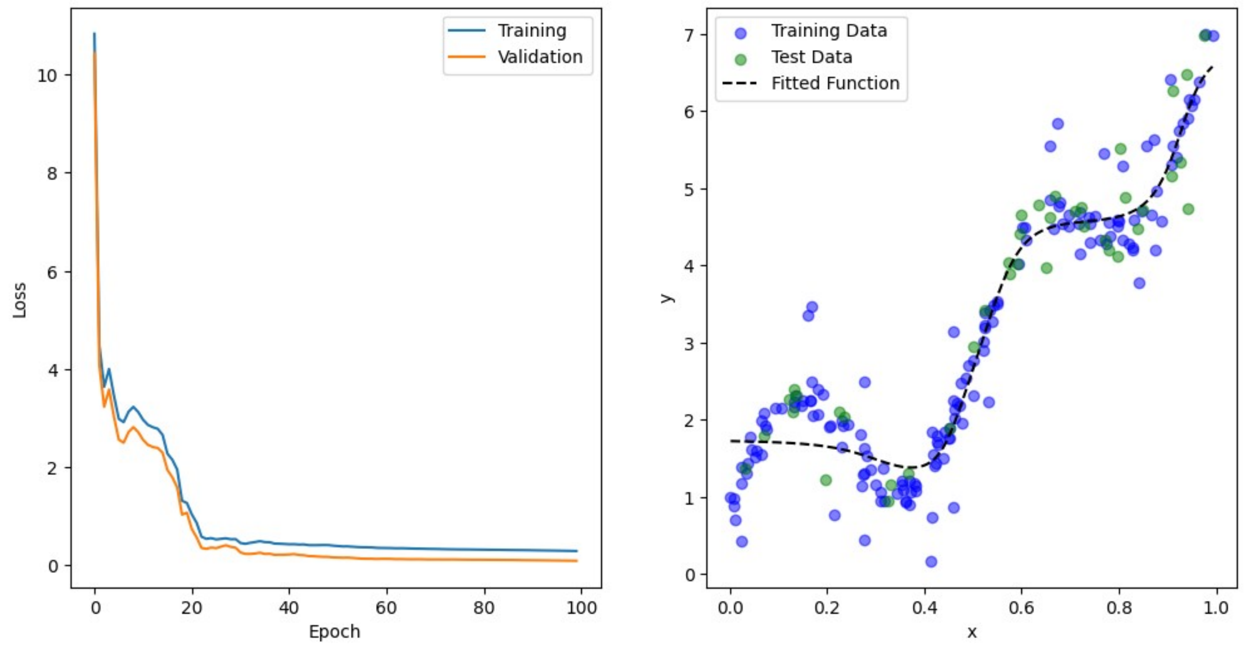
MSE of the Trained Model on the Test Data: 0.2503



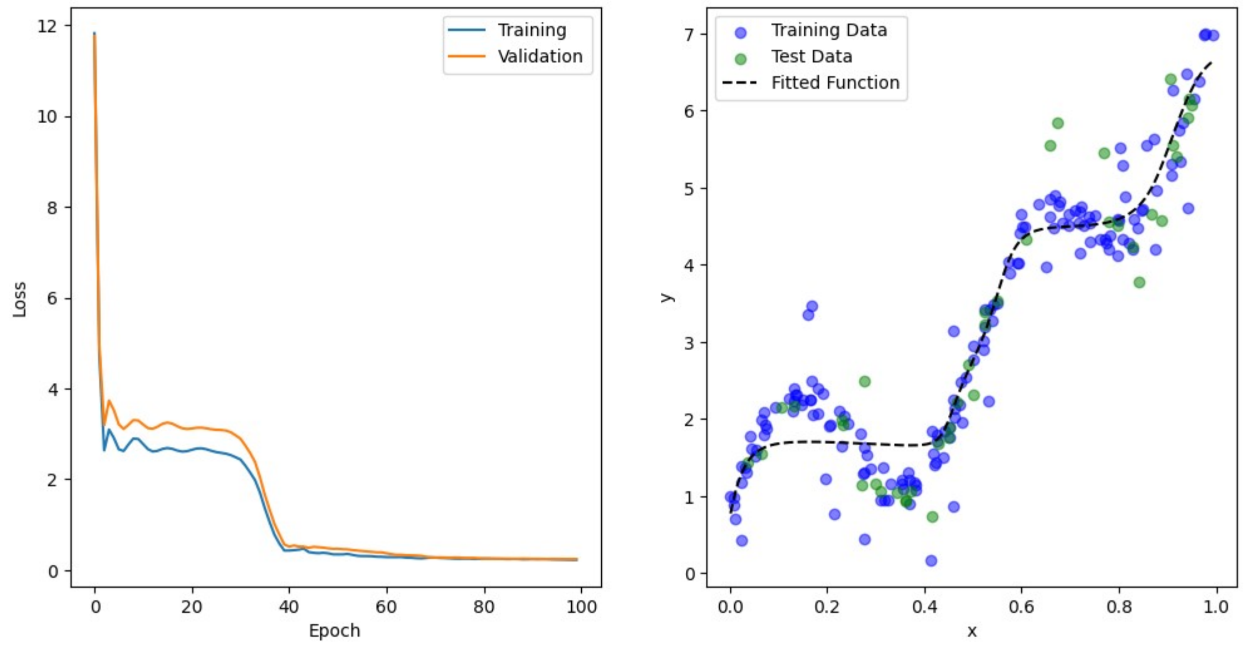
MSE of the Trained Model on the Test Data: 0.1411



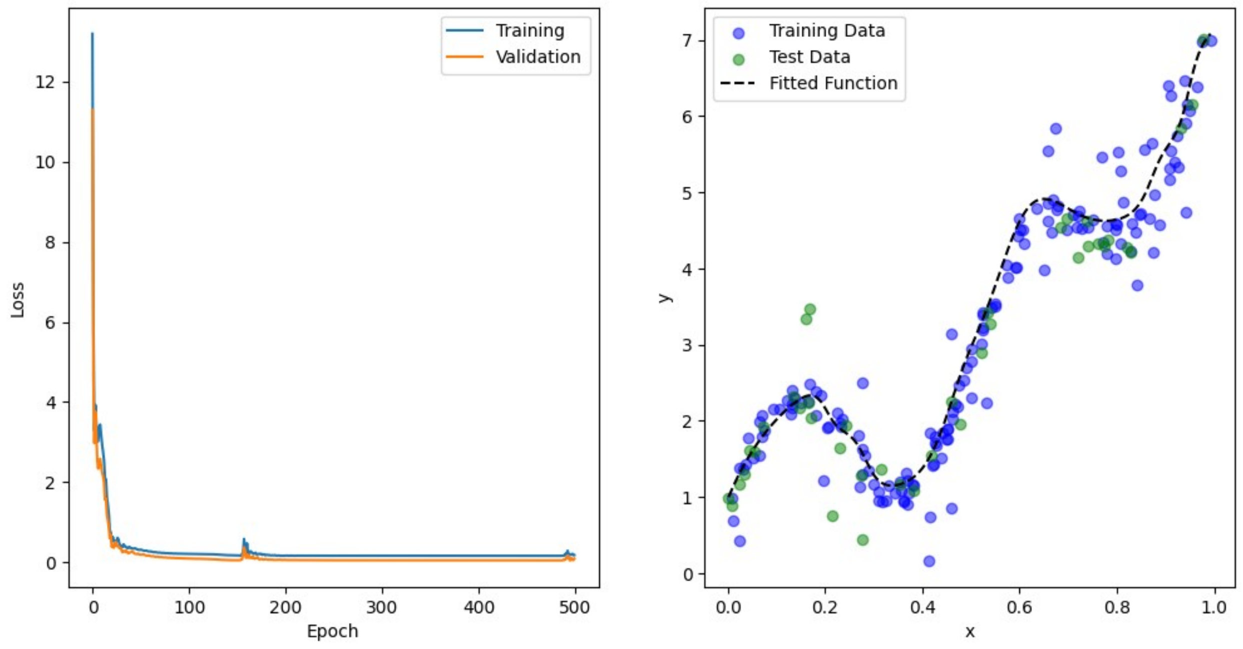
MSE of the Trained Model on the Test Data: 0.1975



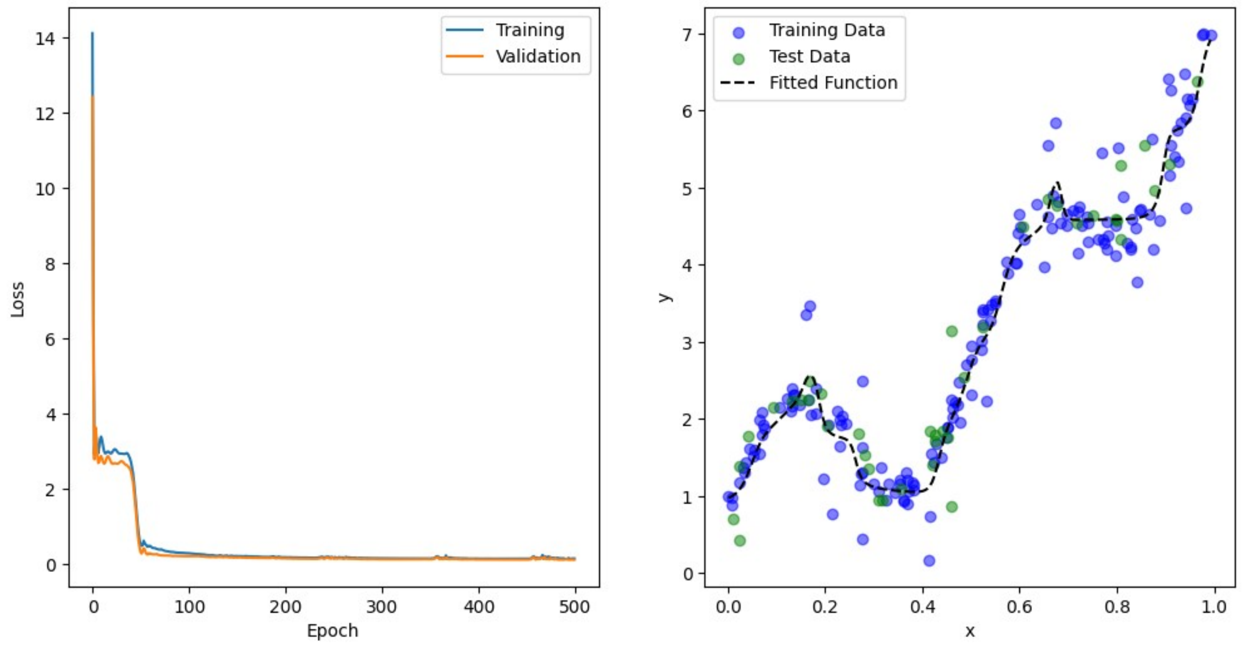
MSE of the Trained Model on the Test Data: 0.2931



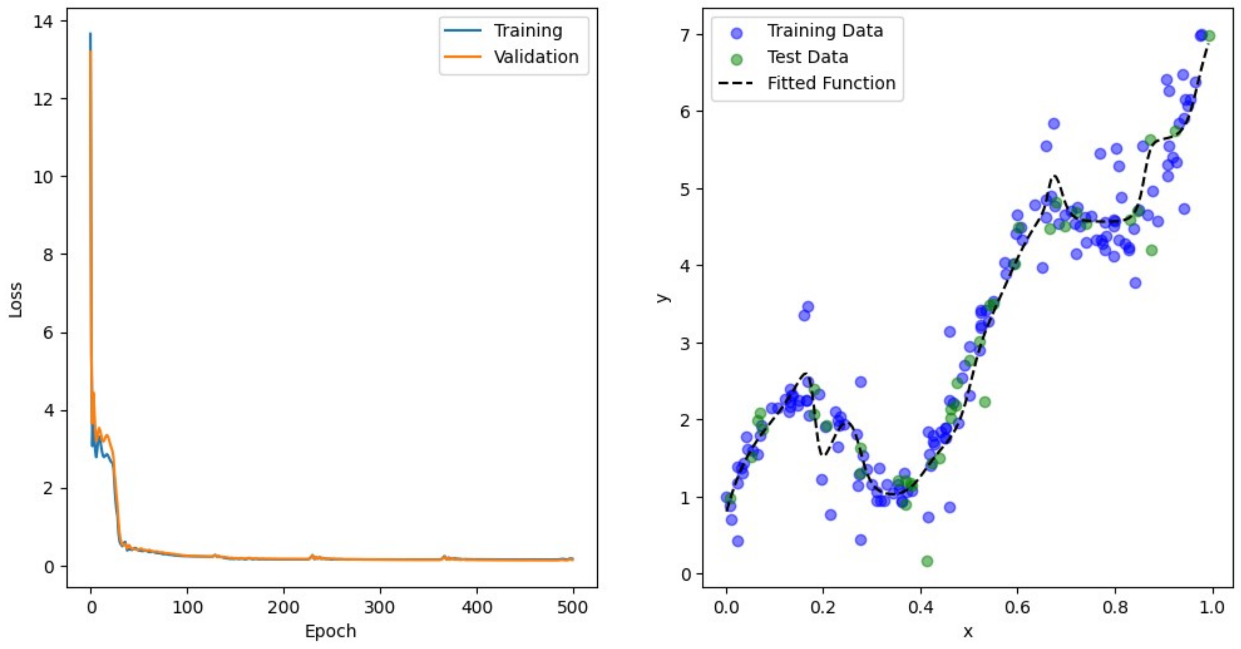
MSE of the Trained Model on the Test Data: 0.1925



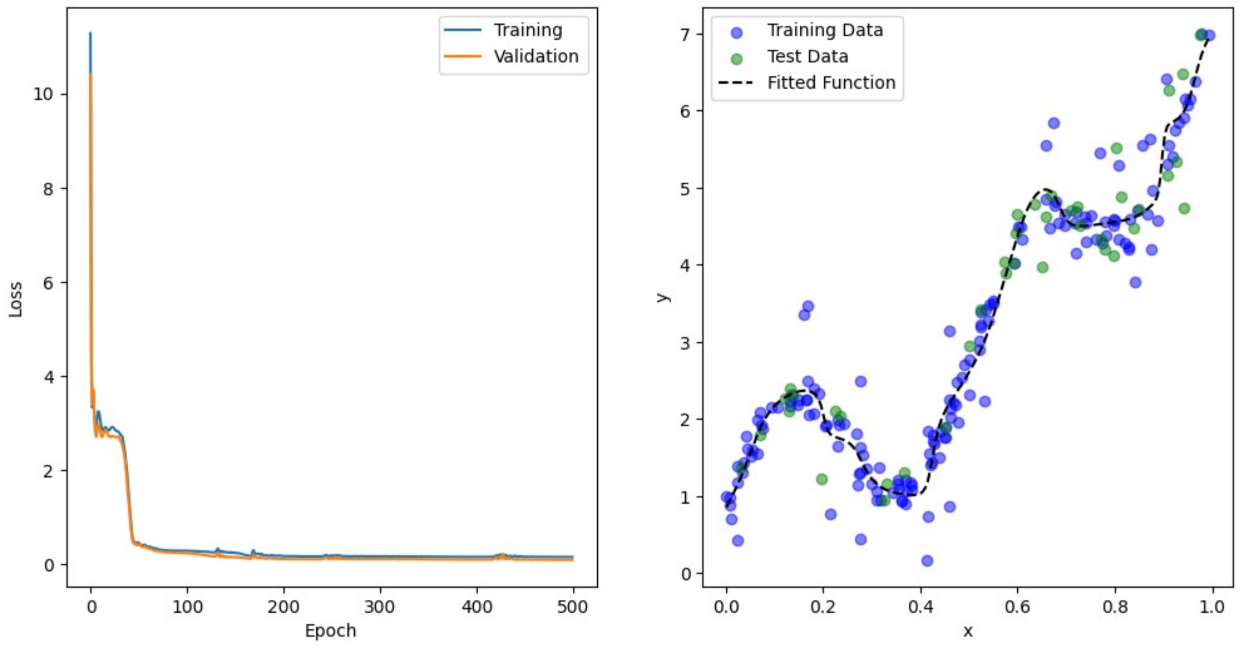
MSE of the Trained Model on the Test Data: 0.1761



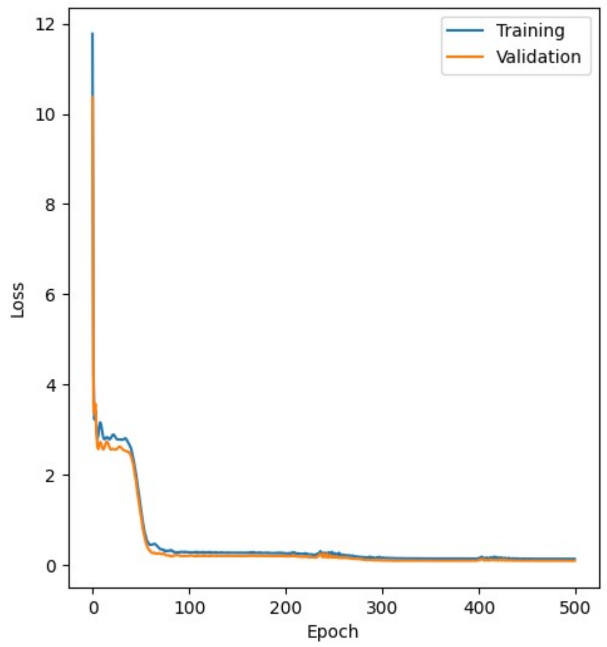
MSE of the Trained Model on the Test Data: 0.1515

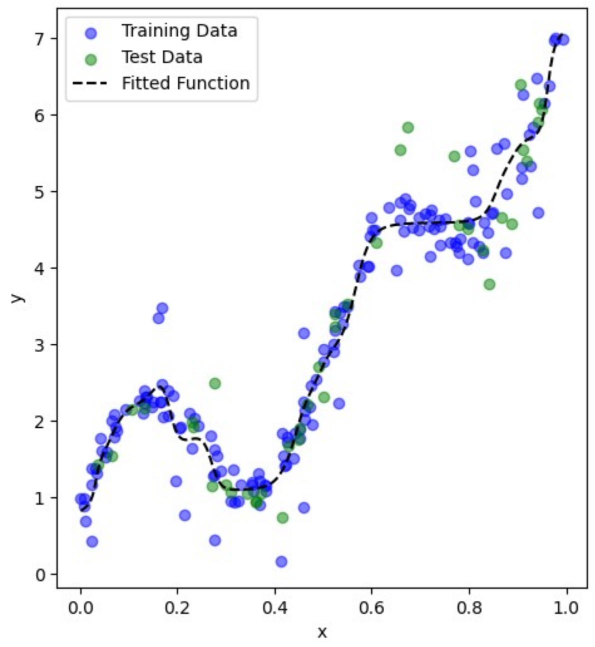


MSE of the Trained Model on the Test Data: 0.1817

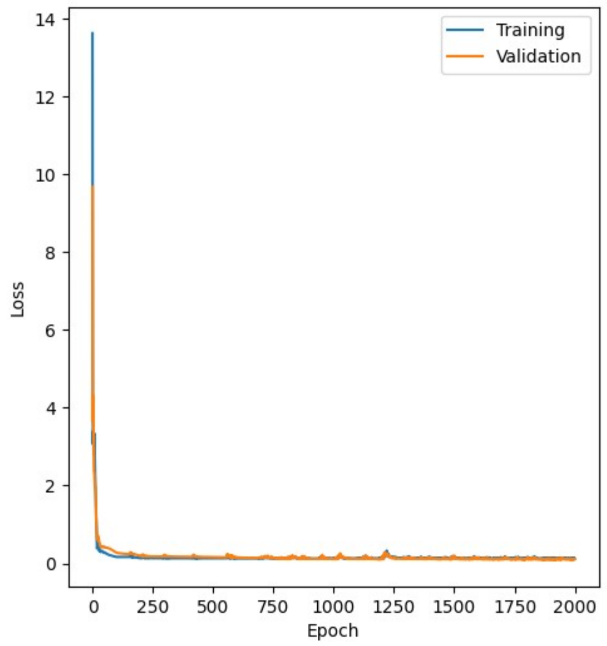


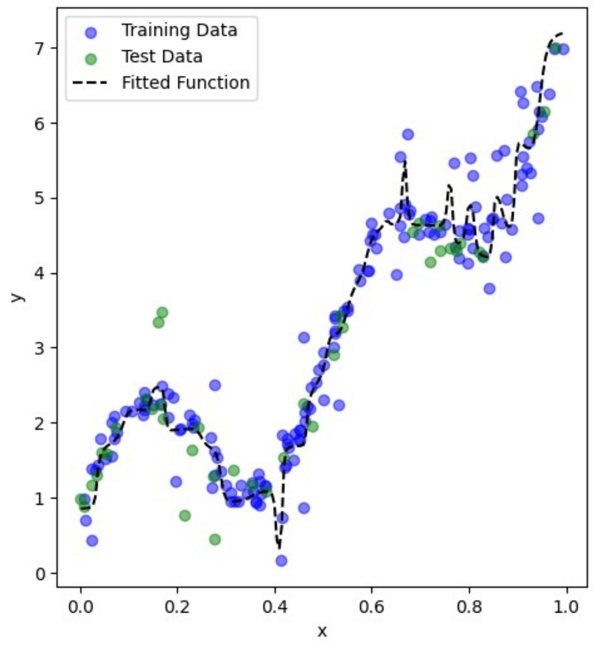
MSE of the Trained Model on the Test Data: 0.2230



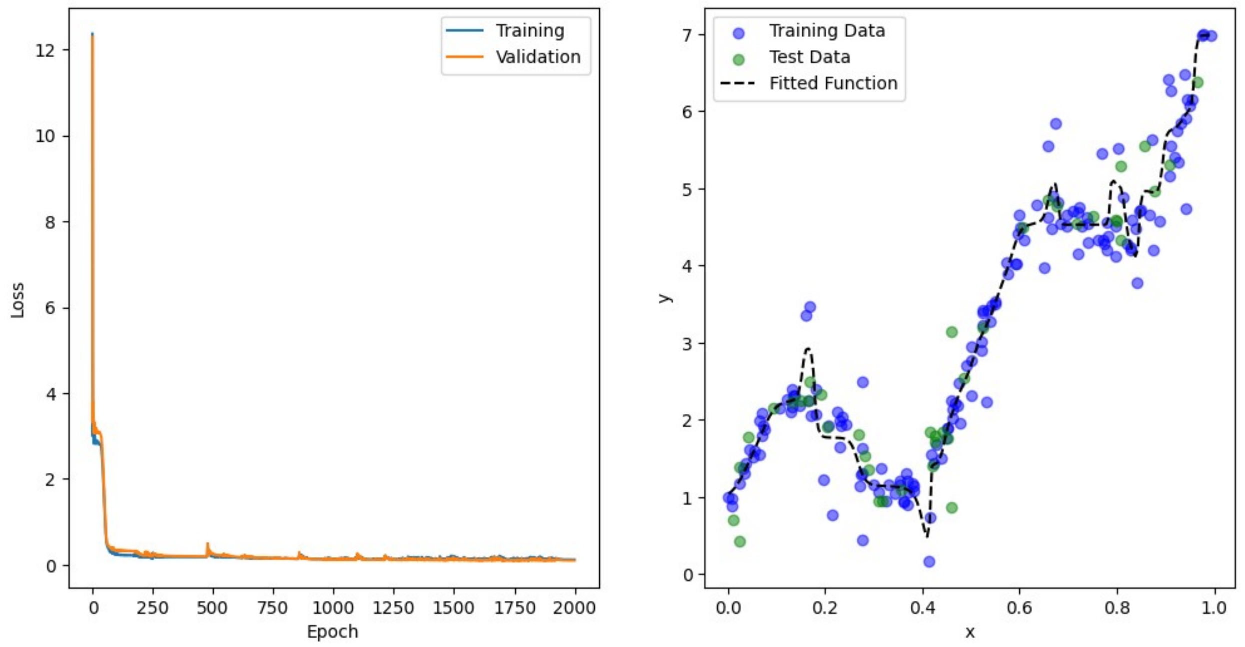


MSE of the Trained Model on the Test Data: 0.1904

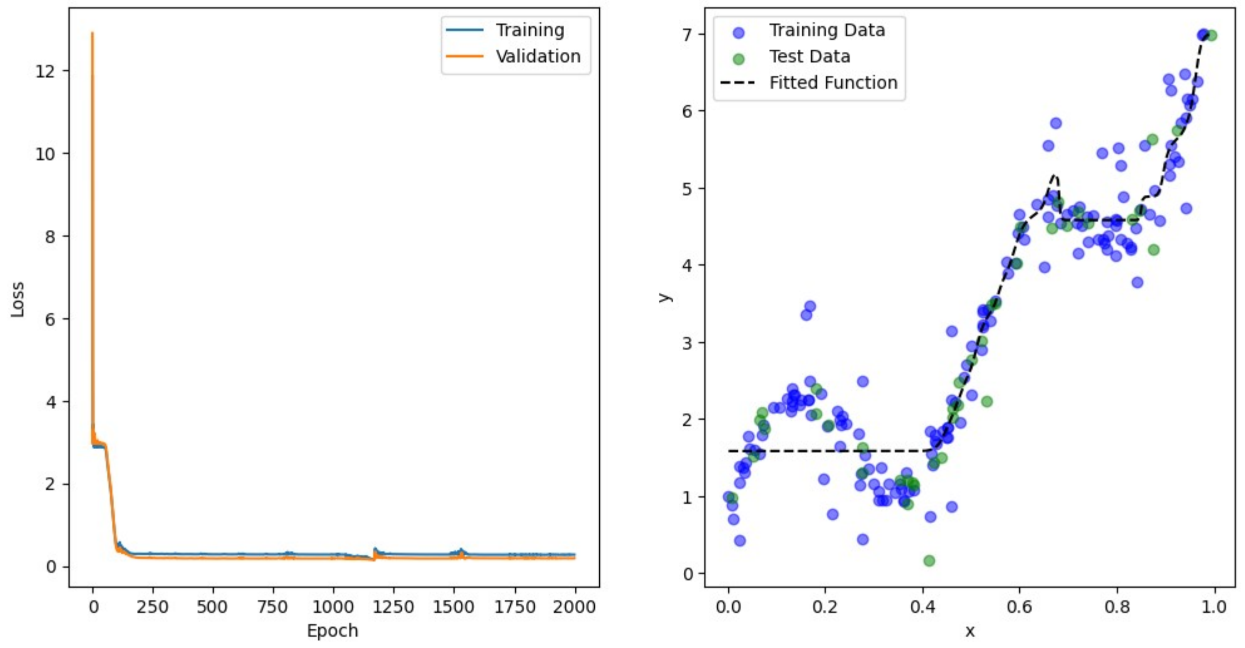




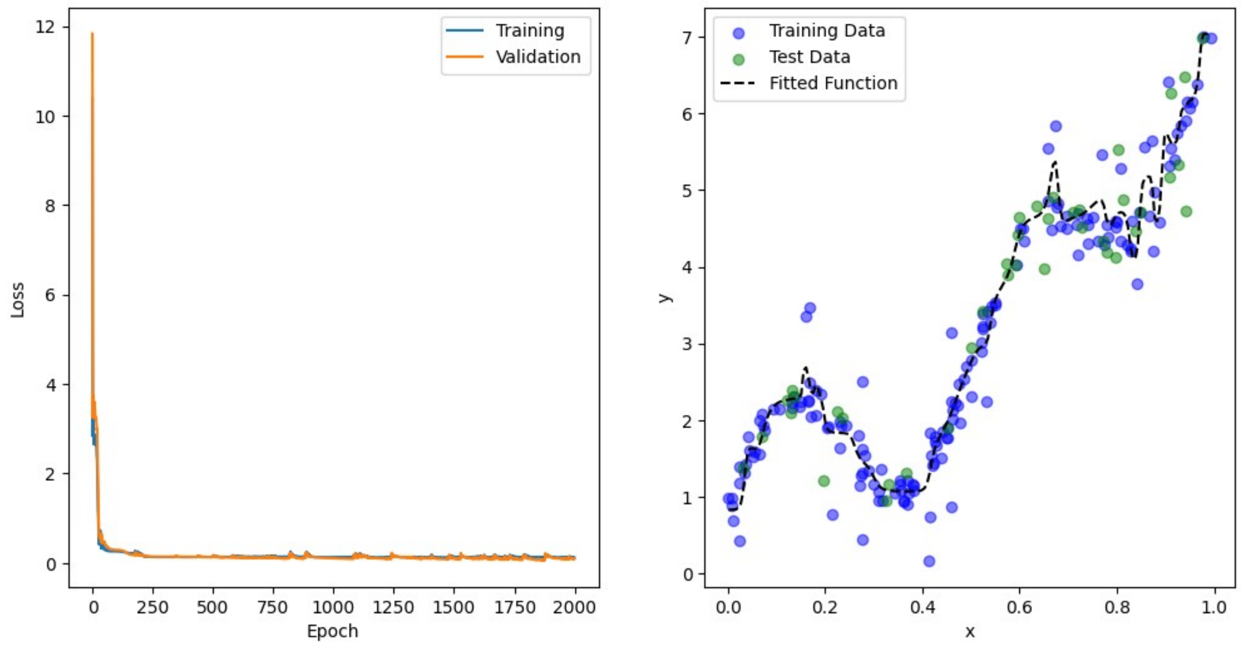
MSE of the Trained Model on the Test Data: 0.2130



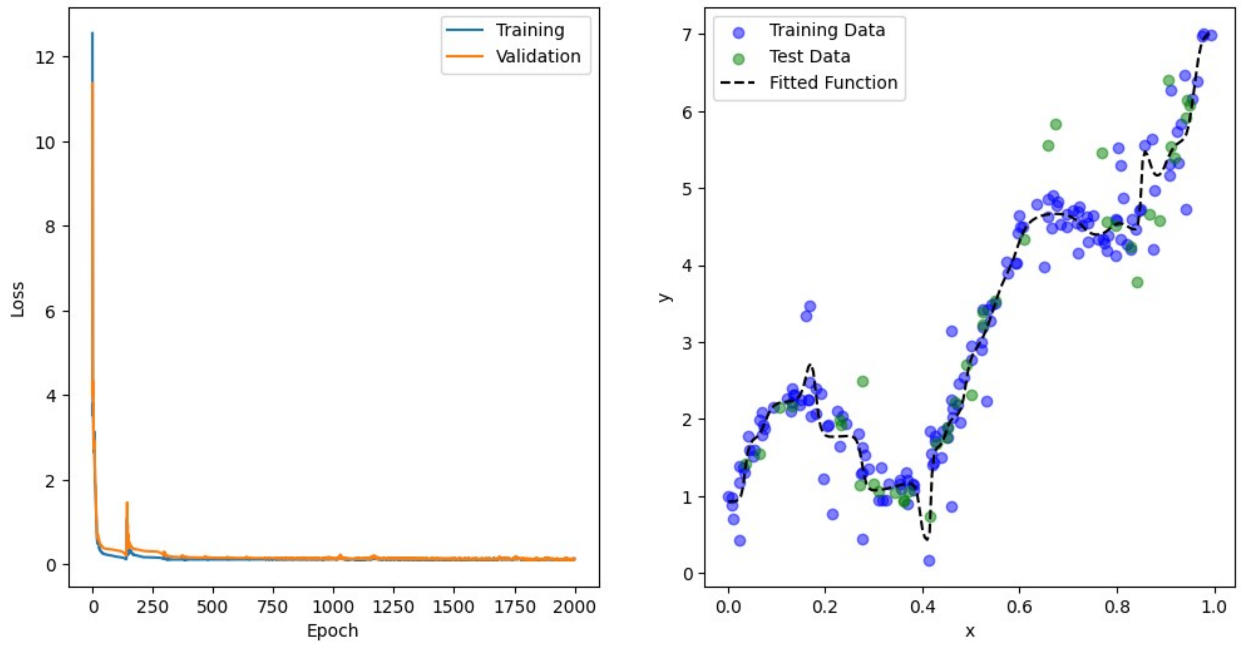
MSE of the Trained Model on the Test Data: 0.1999



MSE of the Trained Model on the Test Data: 0.1688



MSE of the Trained Model on the Test Data: 0.2024



```
MSE for model trained for 100 epochs: 0.2477
MSE for model trained for 500 epochs: 0.1850
MSE for model trained for 2000 epochs: 0.1949
```

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

For the model trained for 100 epochs, the average MSE is around 0.2477. For the model trained for 500 epochs, the average MSE is around 0.1850. For the model trained for 2000 epochs, the average MSE is around 0.1949. From the above results, the average MSE performance can be ranked from best to worst as the following: 500 epochs, 2000 epochs, and 100 epochs. The most optimal number of epochs would be 500 epochs.

It is important to perform cross-validation when evaluating a model because it helps the model reduce the tendency of overfitting. It also gives us a better idea of our model's prediction compared to unseen data.

For a given number of epochs, there can be slight variations depending on the subsets of data used in each fold. However, the variation should not be significant. Models trained for 100 epochs show indications of underfitting. On the other hand, models trained for 2000 epochs show indications of overfitting.