SRECHARAN SELVAM

Homework 10

Instructions

This homework contains **2** concepts and **3** programming questions. In MS word or a similar text editor, write down the problem number and your answer for each problem. Combine all answers for concept questions in a single PDF file. Export/print the Jupyter notebook as a PDF file including the code you implemented and the outputs of the program. Make sure all plots and outputs are visible in the PDF.

Combine all answers into a single PDF named andrewID_hw10.pdf and submit it to Gradescope before the due date. Refer to the syllabus for late homework policy. Please assign each question a page by using the "Assign Questions and Pages" feature in Gradescope.

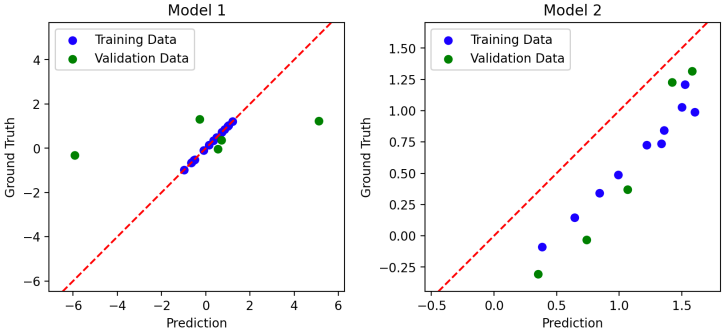
Here is a breakdown of the points for programming questions:

Name	Points
M10-L1-P1	15
M10-L2-P1	15
M10-HW1	60

Problem 1 (5 points)

The two linear least squares regression models are fit on the

same exact training and validation datasets. Below are the R^2 plots for the two models. Which of the following can be said about the models?



(Multiple choice - choose one)

1. Model 1 is low bias but high variance, Model 2 is low variance but high bias
2. Model 1 is high bias but low variance, Model 2 is high variance but low bias
3. Model 1 is low bias but high variance, Model 2 is high bias
4. Model 1 is high bias but low variance, Model 2 is high variance

Problem 2 (5 points)

Which of the following statements is true of k-fold cross validation?

Multiple choice (choose one)

1. K-fold cross validation helps us determine how to partition the data to obtain optimal model performance

2. K-fold cross validation trains k individual models and combines their predictions to generate a better performing model
3. K-fold cross validation trains k models to find the optimal set of hyperparameters for a given dataset
4. K-fold cross validation partitions the data into k equal sized subsets, and trains k models, each time using one subset as the validation data and the rest as the training data

ANSWER:

PROBLEM 1 :-

ANSWER : OPTION 1

PROBLEM 2 :-

ANSWER : OPTION 4

M10-L1 Problem 1

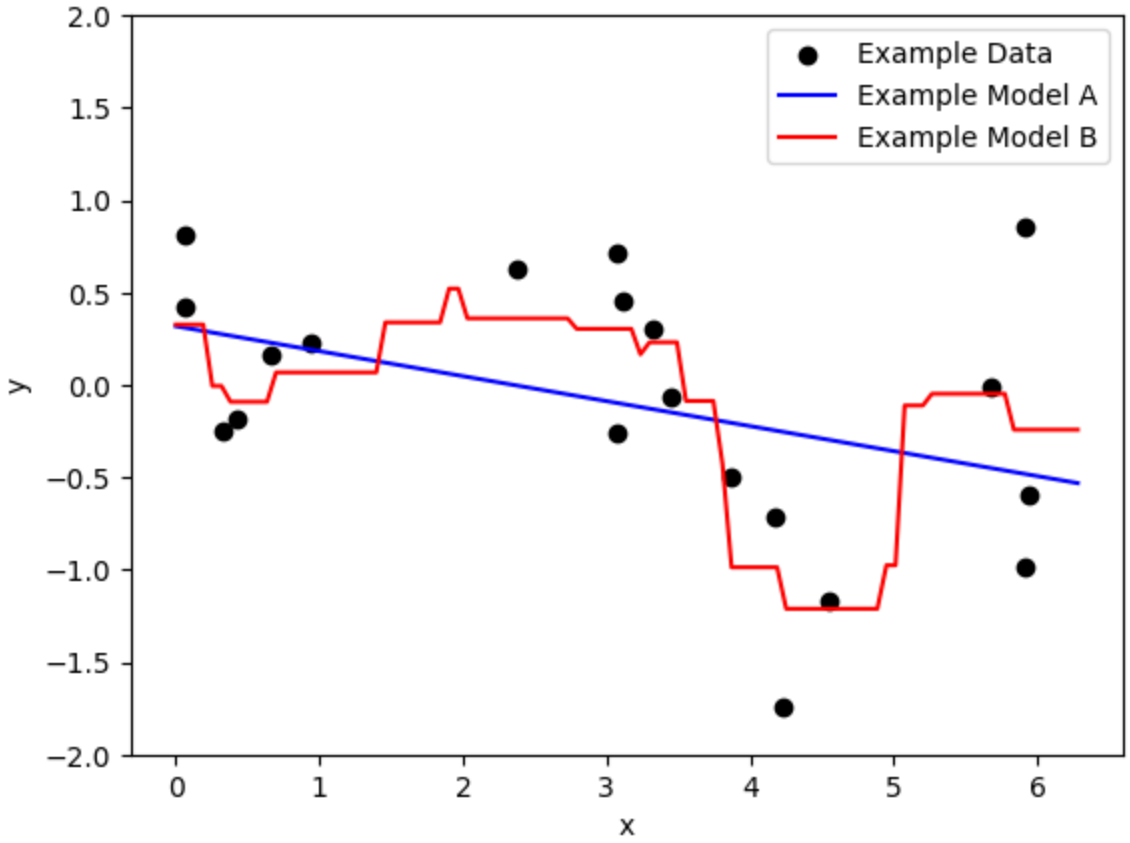
In this problem you will look compare models with lower/higher variance/bias by computing bias and variance at a single point.

In [63]:
```
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.

In [64]:
```
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()` .

In [65]:
```
# YOUR CODE GOES HERE
model_A = []
model_B = []

for i in range(50):
    modelA, modelB, i,i = train_models()
    model_A.append(modelA)
    model_B.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)`).

In [66]:
```
x = 1.57

# YOUR CODE GOES HERE
pred_A = [eval_model_at_point(model,x) for model in model_A]
pred_B = [eval_model_at_point(model,x) for model in model_B]
```

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.

In [69]:
```
yGT = 1

# YOUR CODE GOES HERE
bias_A = np.abs(np.mean(pred_A) - yGT)
var_A = np.var(pred_A)
bias_B = np.abs(np.mean(pred_B) - yGT)
var_B = np.var(pred_B)
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 = 0.500 |  Variance = 0.041
Model B |   Bias = 0.127 |  Variance = 0.091
```

Questions

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

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

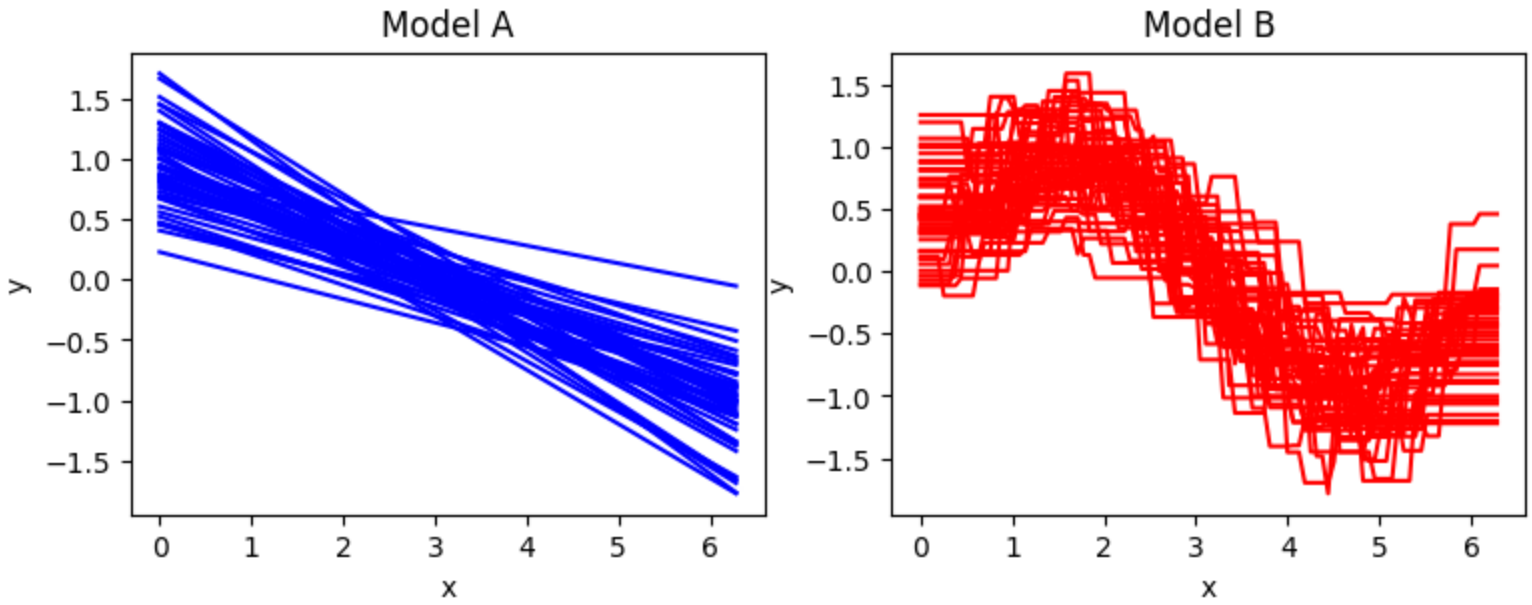
In [70]:
```
print("1) Model B (K-Neighbors Regressor) has a smaller bias compared to Model A.")
print("2) Model A (Linear Regression) has lower variance compared to Model B.")
```

```
1) Model B (K-Neighbors Regressor) has a smaller bias compared to Model A.
2) Model A (Linear Regression) has lower variance compared to Model B.
```

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.

In [75]:
```
plt.figure(figsize=(9,3))

plt.subplot(1,2,1)
plt.title("Model A")
# YOUR CODE GOES HERE
for model in models_A:
    plot_model(model, color="blue")

plt.subplot(1,2,2)
plt.title("Model B")
# YOUR CODE GOES HERE
for model in models_B:
    plot_model(model, color="red")
plt.show()
```


Model A
Model B
y
x
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
1
2
3
4
5
6

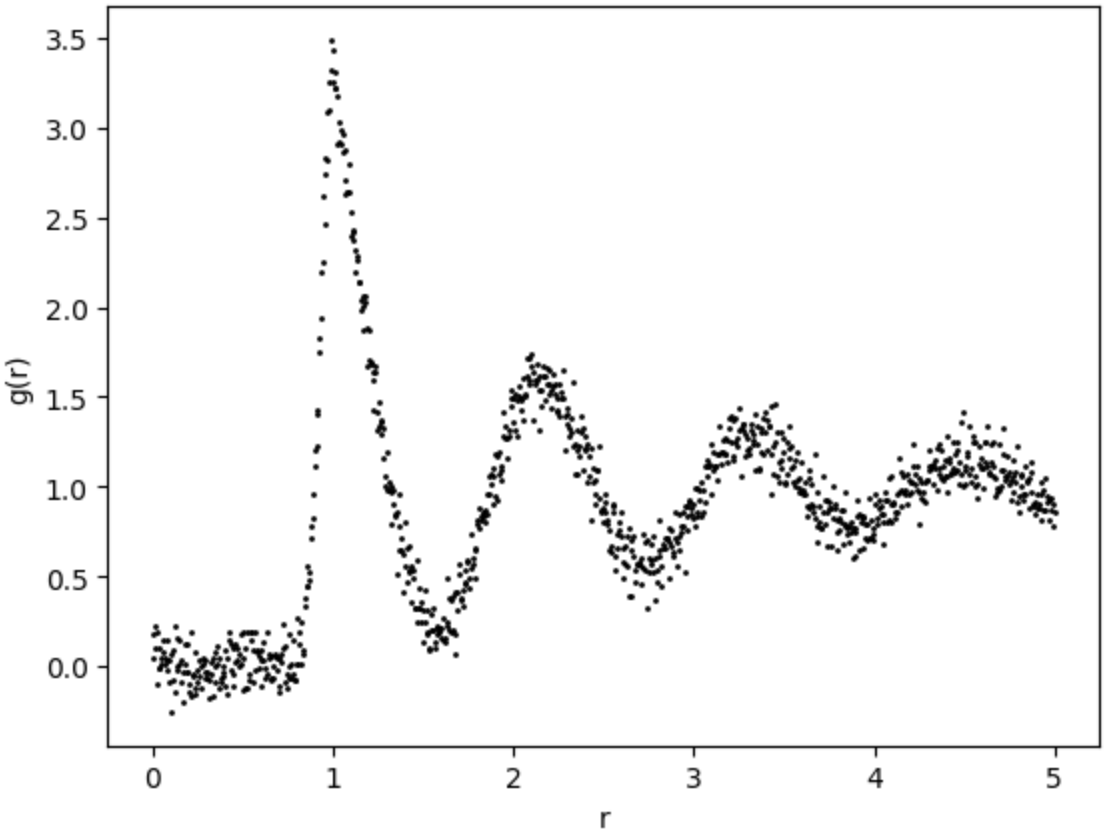
M10-L2 Problem 1

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`

In [10]:
```
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.

In [11]:
```
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()`

In [12]:
```
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: 0.19279856417941277
Validation score: 0.2774937249705789
Validation score: 0.3104352357647894
Validation score: 0.20608404129798263
Validation score: 0.0379012239544968
Validation score: 0.1676244803676994
Validation score: 0.22025003724477443
Validation score: 0.14423712046918657
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.

In [15]:
```
# YOUR CODE GOES HERE
scores2 = []
for train_idx, val_idx in folds.split(X_train):
    model2c = clone(model2)
    model2c.fit(X_train[train_idx,:], y_train[train_idx])
    score = model2c.score(X_train[val_idx,:], y_train[val_idx])
    scores2.append(score)
    print(f"Validation score for Model 2: {score}")

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

scores3 = []
for train_idx, val_idx in folds.split(X_train):
    model3c = clone(model3)
    model3c.fit(X_train[train_idx,:], y_train[train_idx])
    score = model3c.score(X_train[val_idx,:], y_train[val_idx])
    scores3.append(score)
    print(f"Validation score for model 3 : {score}")

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

```
Validation score for Model 2: 0.9135256064394238
Validation score for Model 2: 0.92381162019413
Validation score for Model 2: 0.9109428377428712
Validation score for Model 2: 0.916683295227516
Validation score for Model 2: 0.8980936123083956
Validation score for Model 2: 0.9208009063665946
Validation score for Model 2: 0.9123834705950664
Validation score for Model 2: 0.8780032287365068
Validation score for Model 2: 0.9281564779069266
Validation score for Model 2: 0.95771300087561
Average validation score for Model 2: 0.9160114056393042
Validation score for model 3 : 0.9629033605148644
Validation score for model 3 : 0.9466107686883456
Validation score for model 3 : 0.9518315048355763
Validation score for model 3 : 0.9514051770741325
Validation score for model 3 : 0.9229643307655355
Validation score for model 3 : 0.9501422202077937
Validation score for model 3 : 0.9322229519501163
Validation score for model 3 : 0.9238931238090653
Validation score for model 3 : 0.9461292855545793
Validation score for model 3 : 0.9611128180031755
Average validation score for Model 3: 0.9449215541403186
```

Comparing models

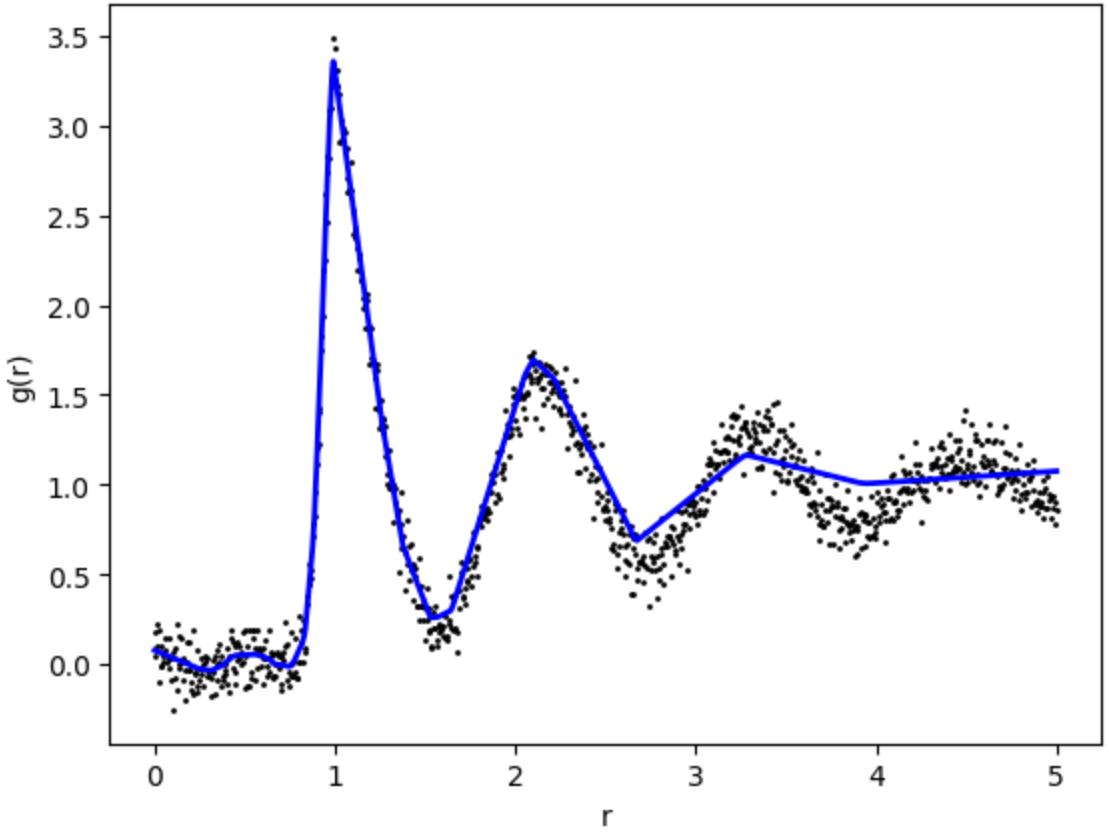
Which model had the best performance according to your validation study?

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.

In [16]:
```
# YOUR CODE GOES HERE
# Finding the best model
best_model, best_score = max(zip(models, [score1, score2, score3]), key=lambda pair: pair[1])

# Retrain on the full training dataset
best_model.fit(X_train, y_train)

# Report R2 score on training and testing data
r2_train = best_model.score(X_train, y_train)
```

```
r2_test = best_model.score(X_test, y_test)
print(f"The R2 score on training data is = {r2_train}")
print(f"The R2 score on testing data is = {r2_test}")

plt.figure()
plot_data(X,y)

# YOUR CODE GOES HERE
plot_model(best_model)
plt.show()
```

```
The R2 score on training data is = 0.9547034601442717
The R2 score on testing data is = 0.9371864974414205
```



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:

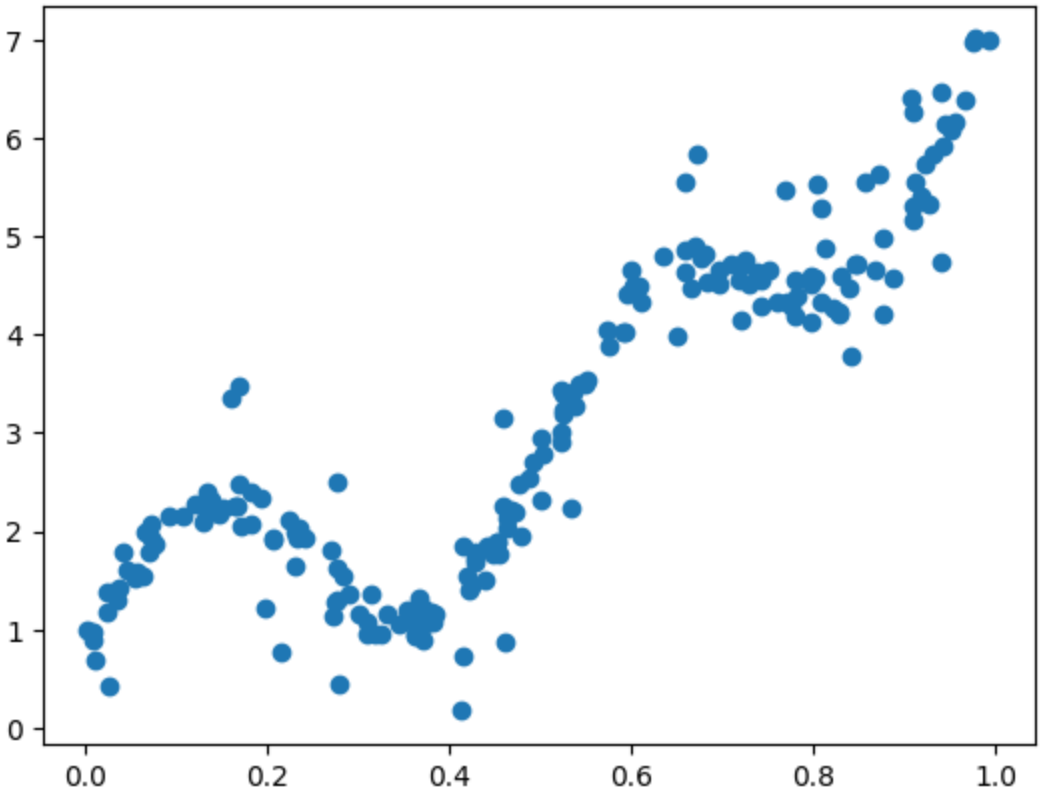
In [21]:
```
import torch.nn as nn
import torch
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt

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.

In [23]:
```
## YOUR CODE GOES HERE
# Load data
data = np.loadtxt("C:/Users/srech/Downloads/m10-hw1-data.txt")
x, y = data[:,0], data[:,1]
plt.scatter(x, y)
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.

In [24]:
```
## YOUR CODE GOES HERE
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(1, 64),
            nn.Tanh(),
            nn.Linear(64, 128),
            nn.Tanh(),
            nn.Linear(128, 128),
            nn.Tanh(),
            nn.Linear(128, 64),
            nn.Tanh(),
            nn.Linear(64, 1)
        )
    def forward(self, x):
        return self.layers(x)

def trainModel(x, y, n_epoch):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
    x_train, x_test, y_train, y_test = torch.tensor(x_train, dtype=torch.float32).view(-1, 1), torch.tensor(x_test, d
    model = Net()
    loss_fn = nn.MSELoss()
    opt = torch.optim.Adam(model.parameters(), lr=0.01)
    train_curve = []
    val_curve = []

    for i in range(n_epoch):
        model.train()
        opt.zero_grad()
        y_pred = model(x_train)
        loss = loss_fn(y_pred, y_train)
        loss.backward()
        opt.step()
        train_curve.append(loss.item())
        model.eval()
        with torch.no_grad():
            val_loss = loss_fn(model(x_test), y_test)
```

```
            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.

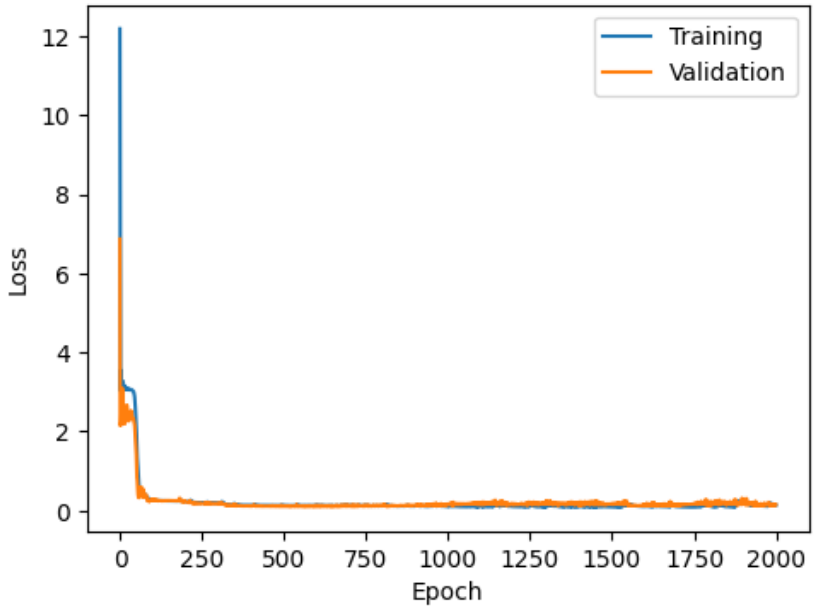
Average the MSE loss on the test set across the 5 folds, and report a single MSE loss for each of the three models.

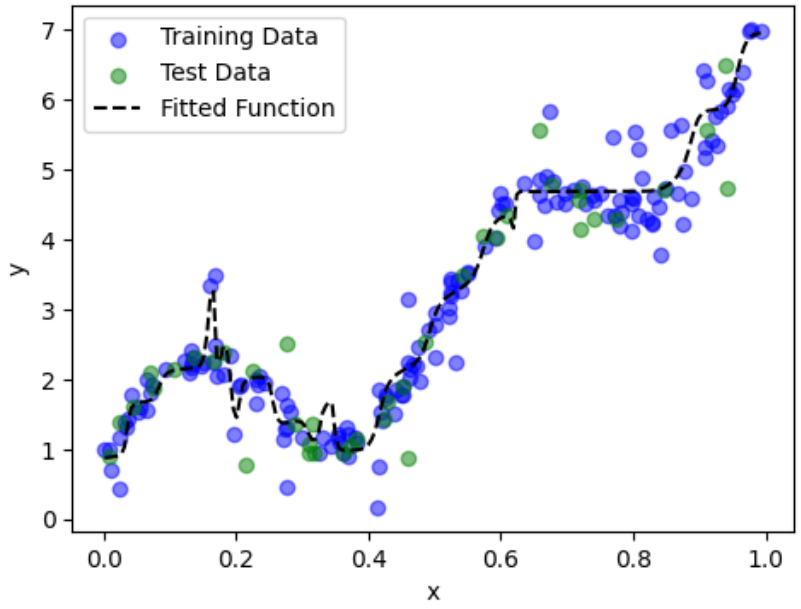
Since there are three models and we are using 5-fold cross validation, you should output 15 figures, with two subplots each.

In [25]:
```
## YOUR CODE GOES HERE
epochs = [100, 500, 2000]
kf = KFold(n_splits=5, shuffle=True)

def train_and_plot(x, y, train_index, test_index, epochs):
    model, train_curve, val_curve = trainModel(x[train_index], y[train_index], epochs)
    plotLoss(ax[0], train_curve, val_curve)
    plotModel(ax[1], model, x, y, train_index, test_index)
    x_test = torch.from_numpy(x[test_index]).float().unsqueeze(0)
    y_test = torch.from_numpy(y[test_index]).float()
    y_preds = model(x_test.T)
```

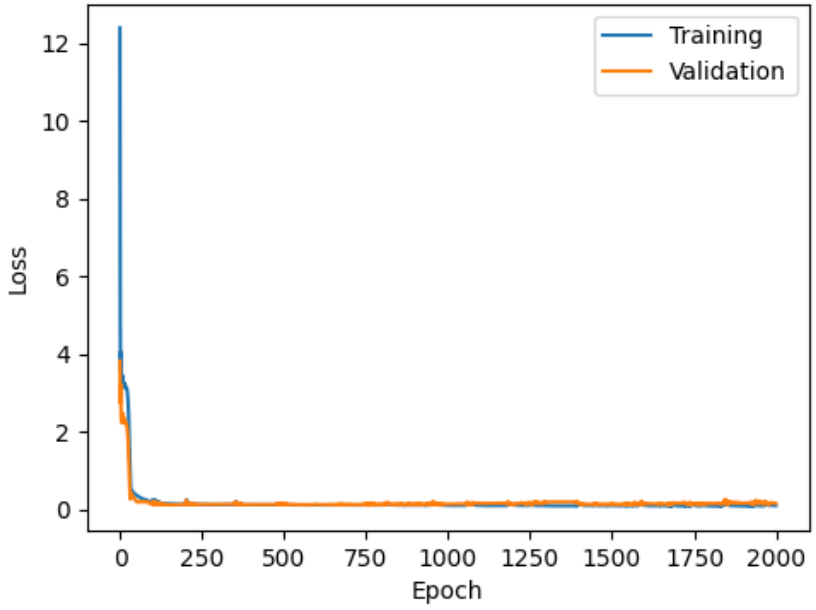
```
    test_loss = F.mse_loss(y_preds.squeeze(), y_test)
    return test_loss.item()
test_loss1 = []
test_loss2 = []
test_loss3 = []

for i in epochs:
    for train_index, test_index in kf.split(x):
        fig, ax = plt.subplots(1, 2, figsize=(12, 4))
        test_loss = train_and_plot(x, y, train_index, test_index, epoch)
        plt.suptitle('Epochs: ' + str(i) + ', MSE: ' + str(test_loss))
        if i == 100:
            test_loss1.append(test_loss)
        elif i == 500:
            test_loss2.append(test_loss)
        else:
            test_loss3.append(test_loss)
        plt.show()

print('The Average MSE loss for model trained with 100 epochs is = ', np.mean(test_loss1))
print('The Average MSE loss for model trained with 500 epochs is =', np.mean(test_loss2))
print('The Average MSE loss for model trained with 2000 epochs is = ', np.mean(test_loss3))
```

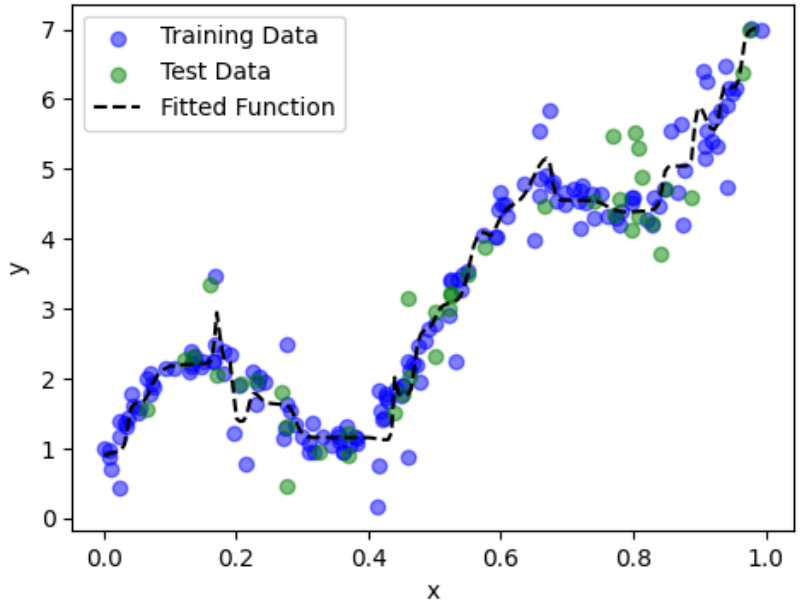
Epochs: 100, MSE: 0.24364838004112244



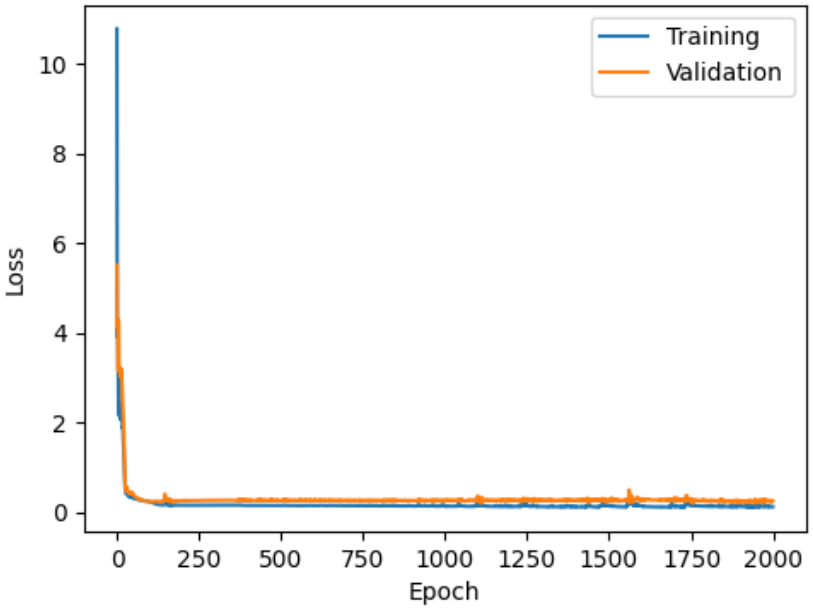


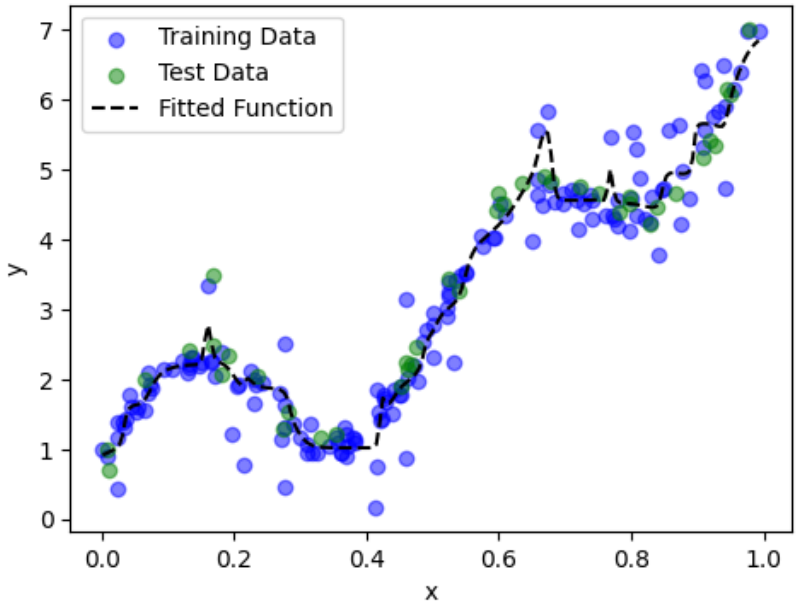
Epochs: 100, MSE: 0.2876756191253662



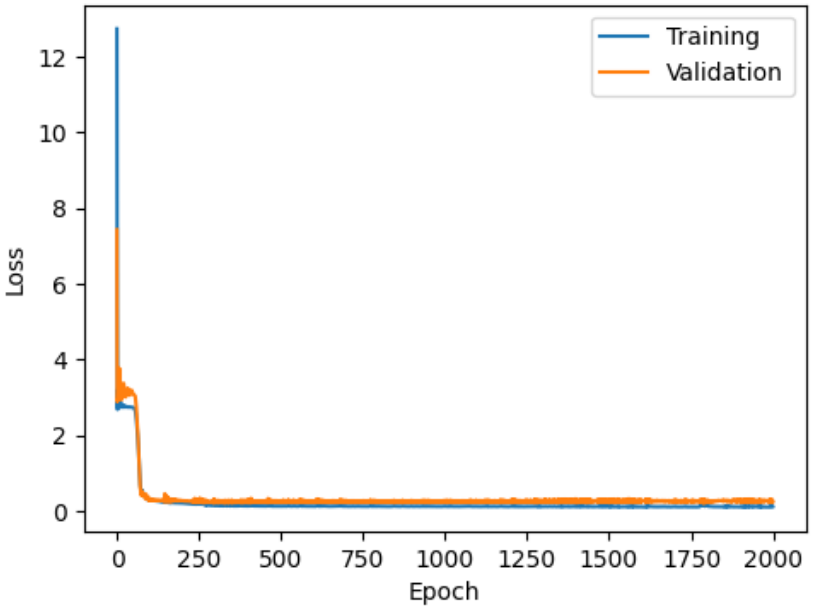


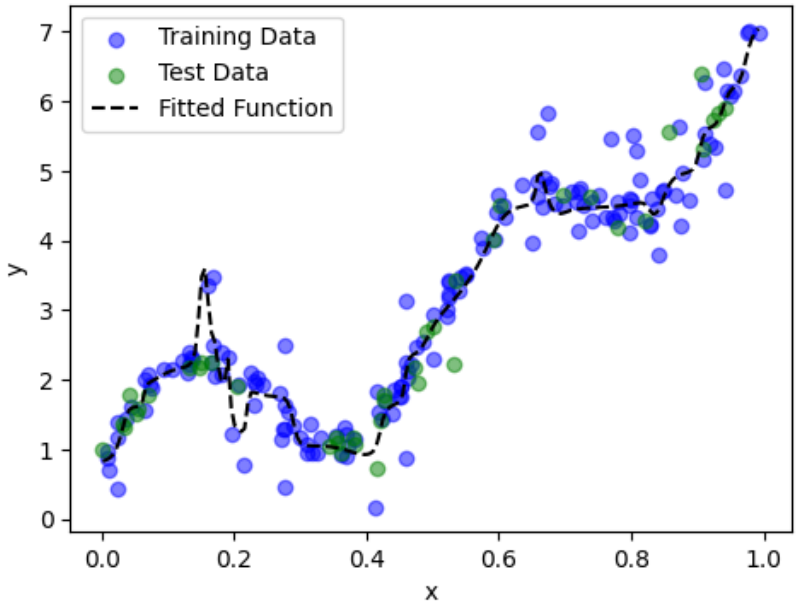
Epochs: 100, MSE: 0.10234446823596954



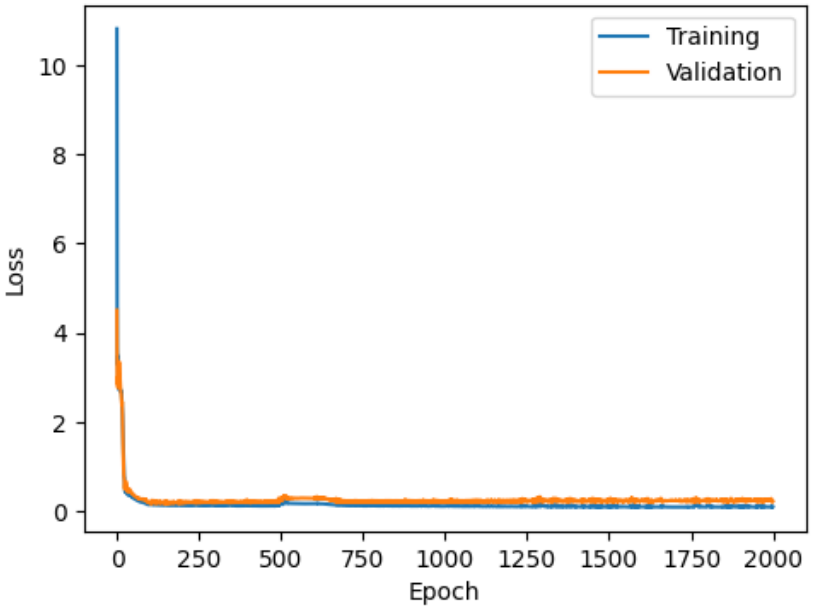


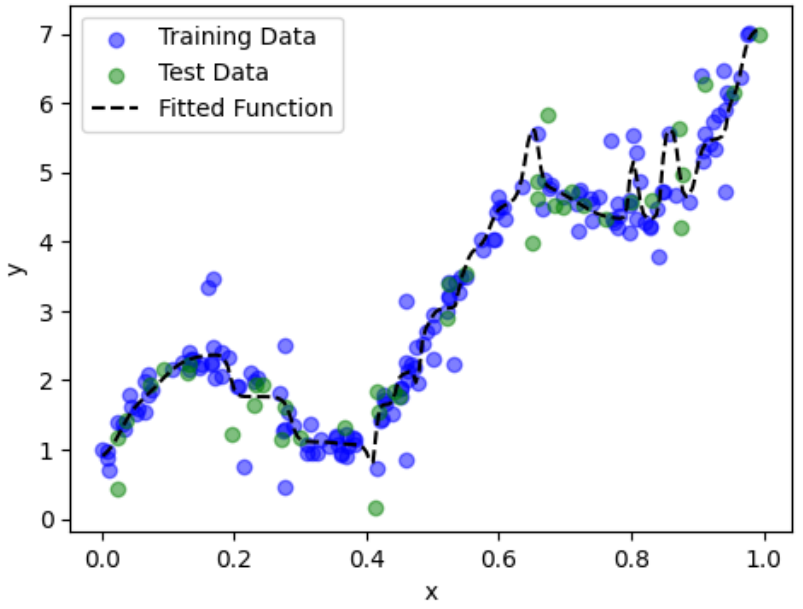
Epochs: 100, MSE: 0.1522085964679718



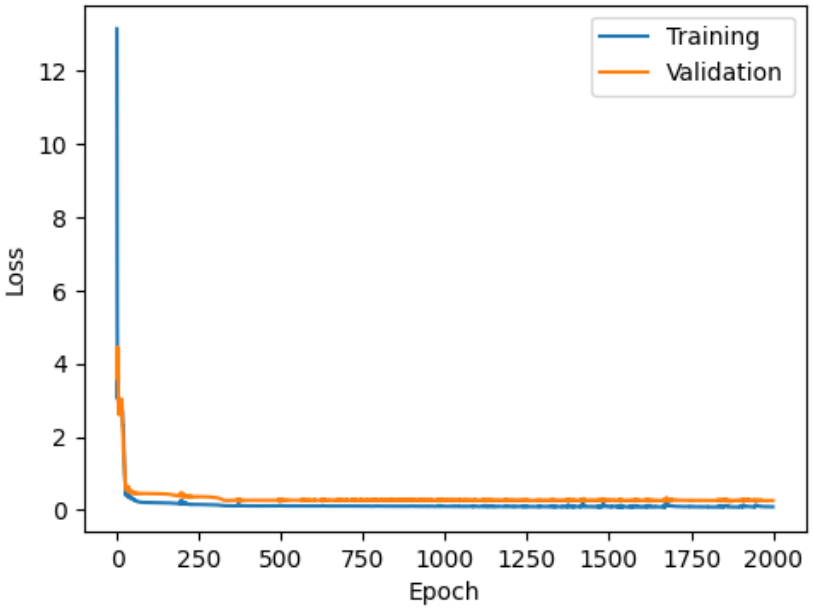


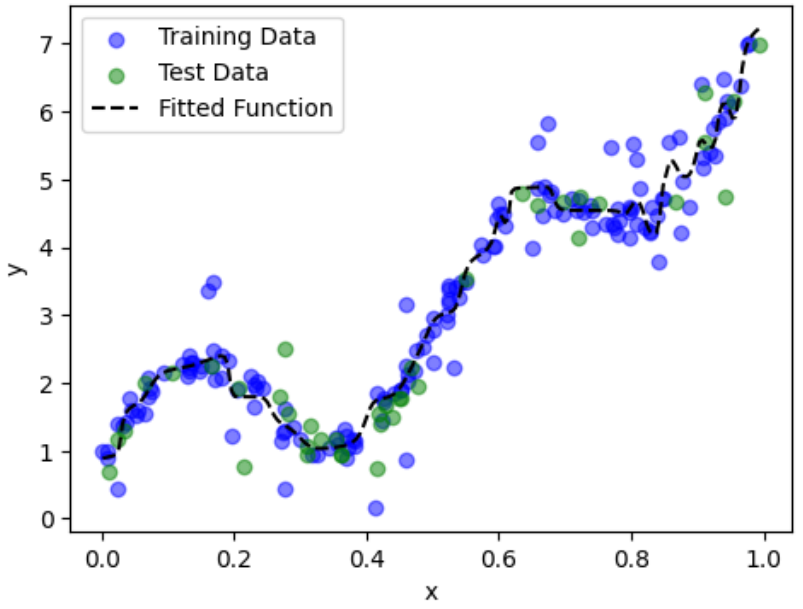
Epochs: 100, MSE: 0.24329355359077454



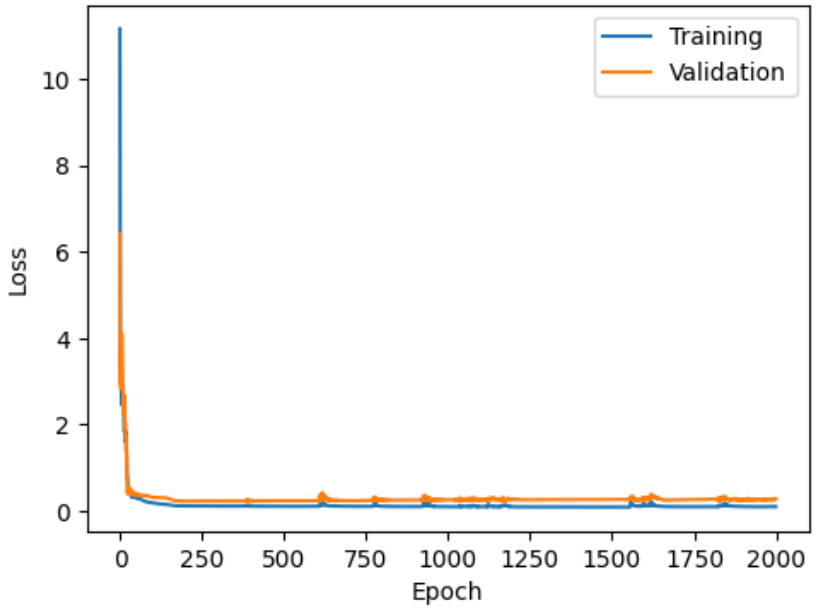


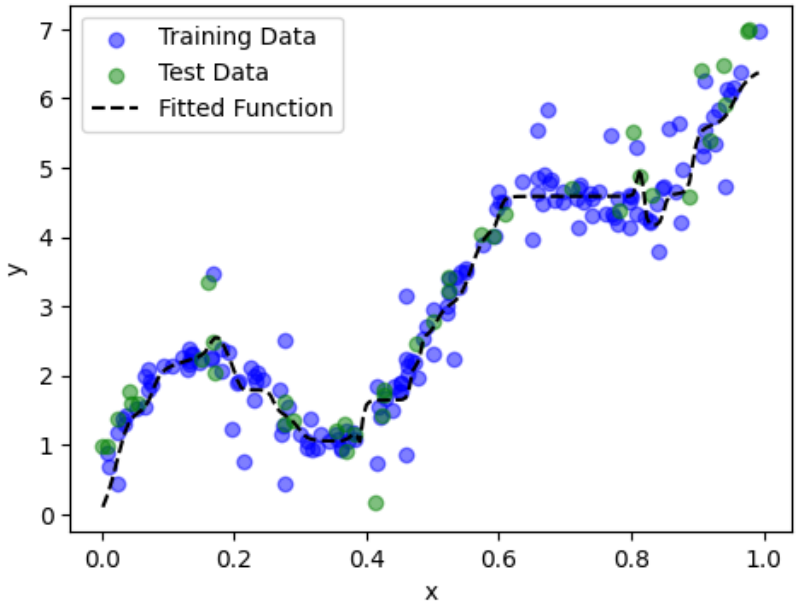
Epochs: 500, MSE: 0.18943825364112854



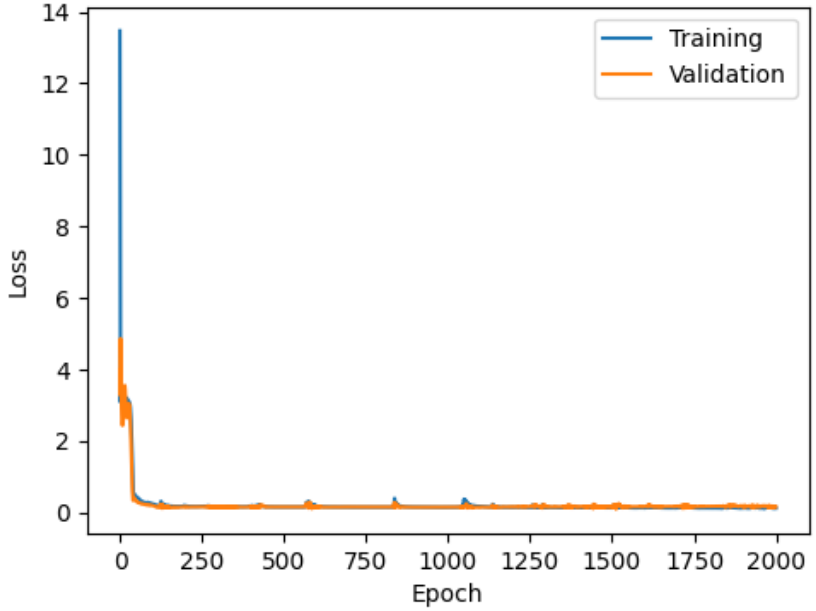


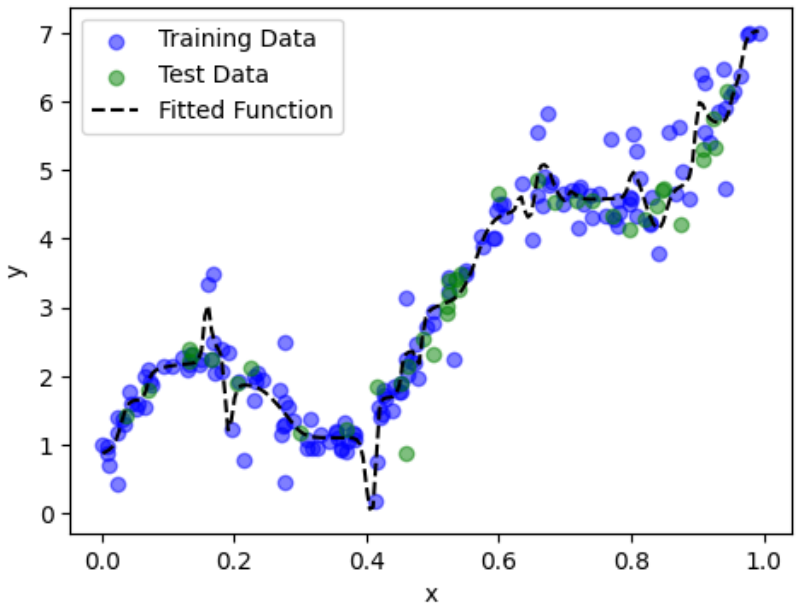
Epochs: 500, MSE: 0.23792941868305206



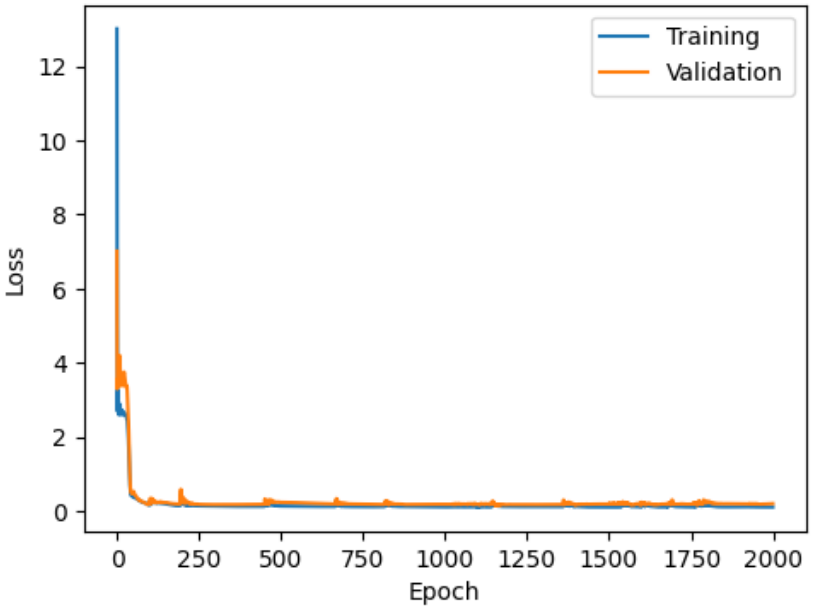


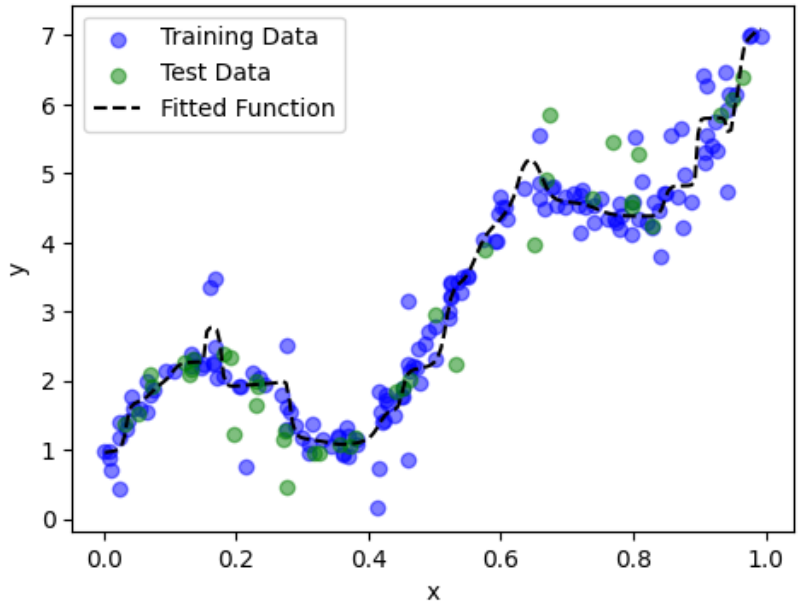
Epochs: 500, MSE: 0.18107804656028748



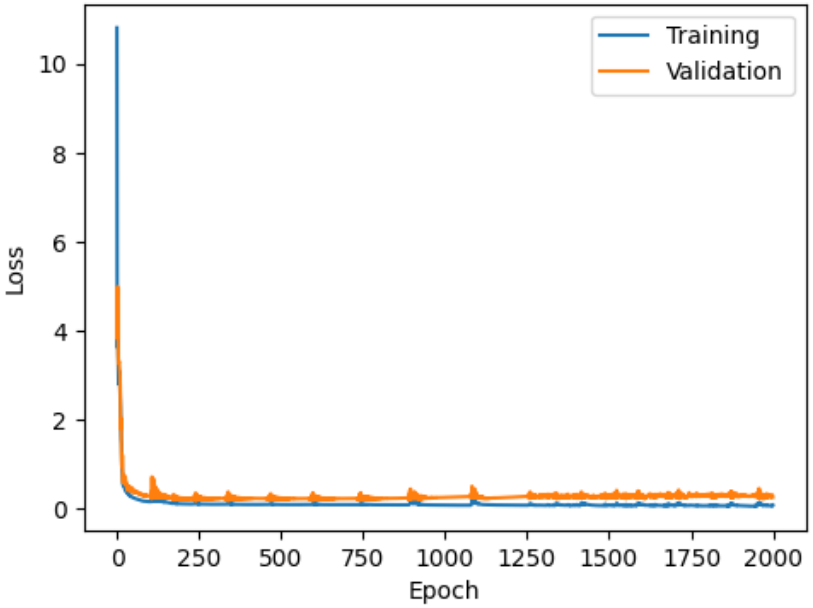


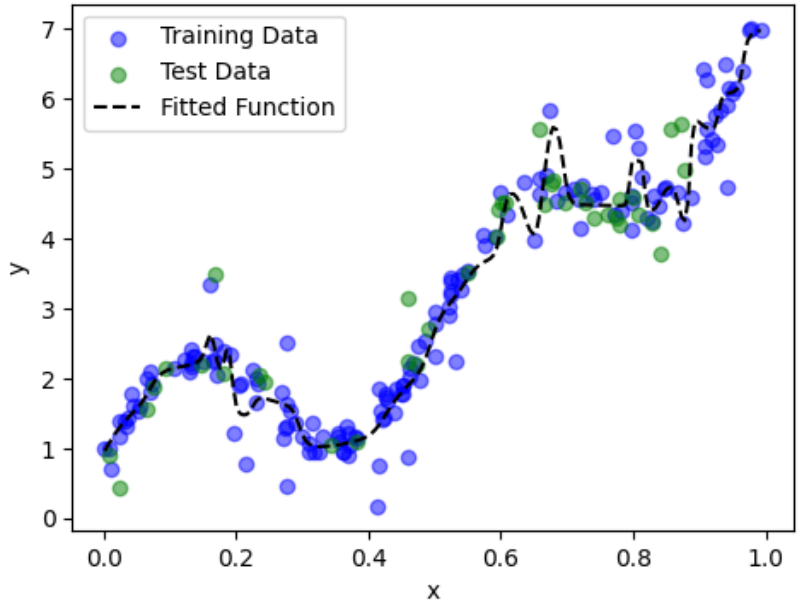
Epochs: 500, MSE: 0.2810816168785095



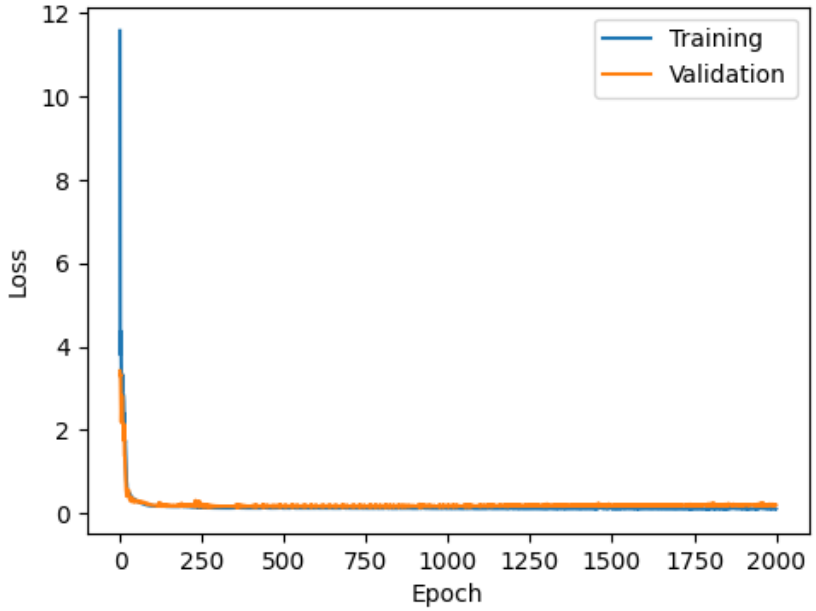


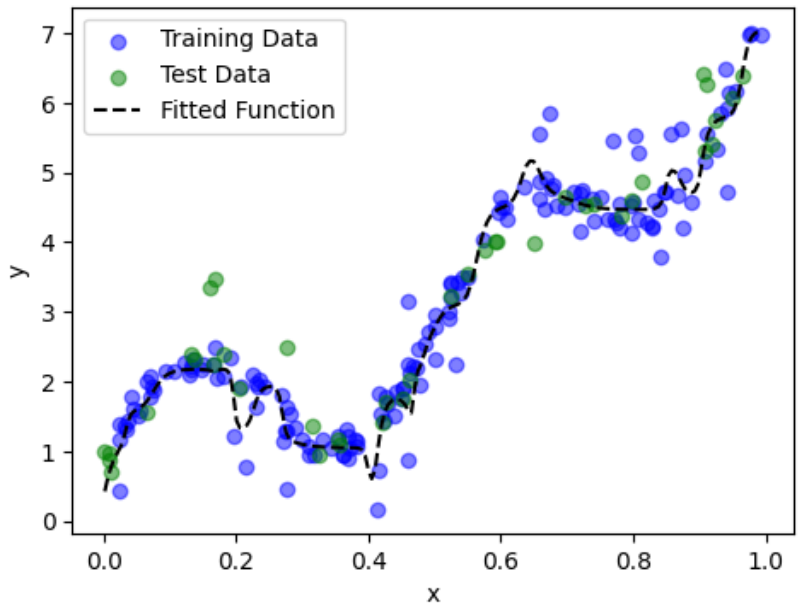
Epochs: 500, MSE: 0.303709089756011 96



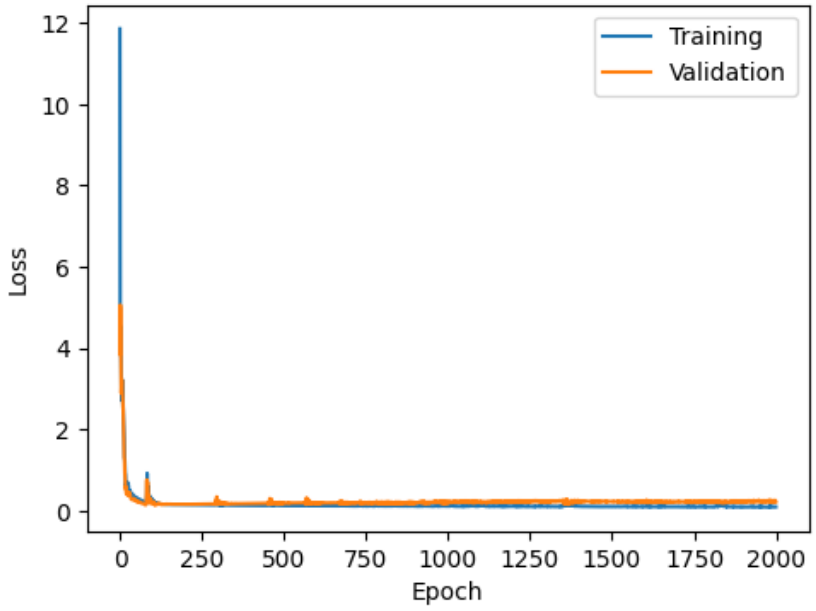


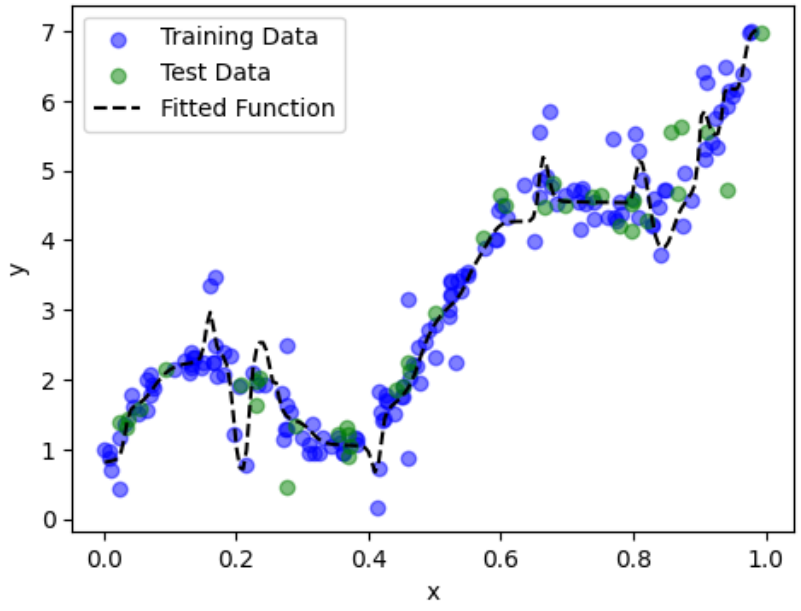
Epochs: 2000, MSE: 0.25476619601249695



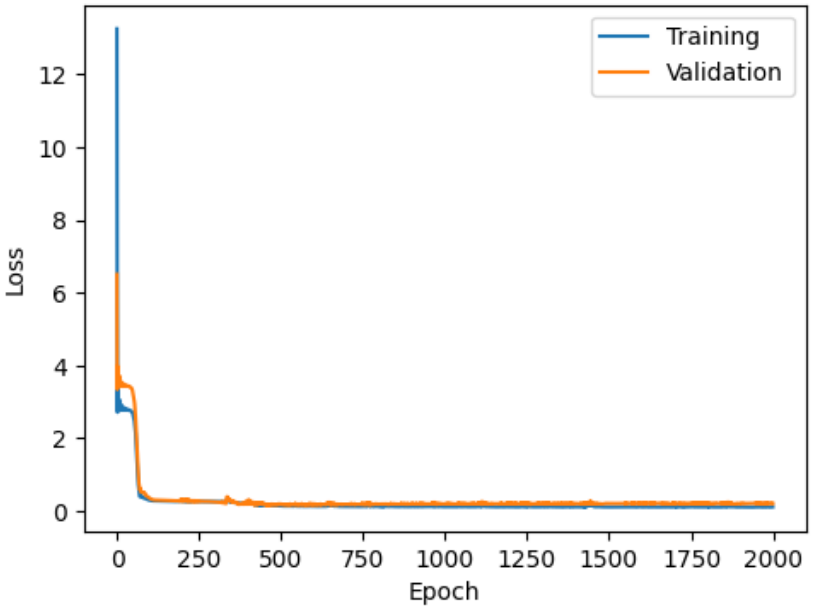


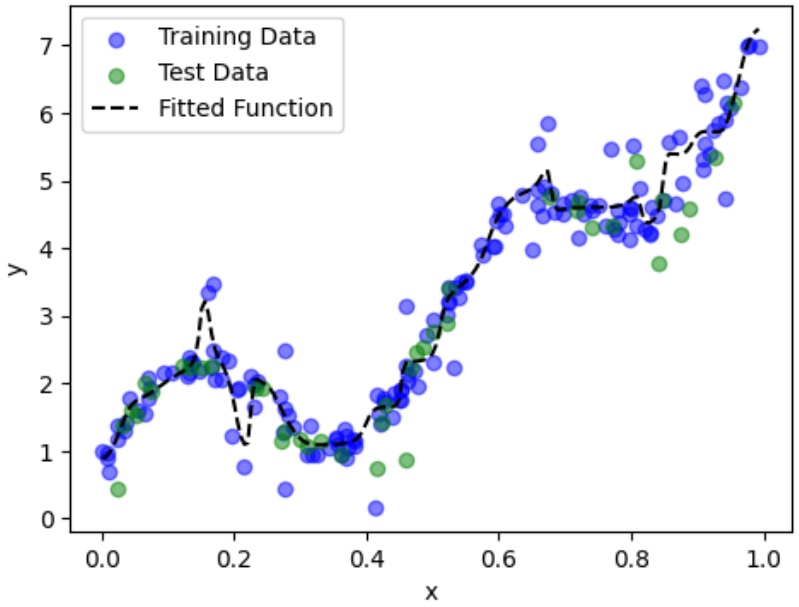
Epochs: 2000, MSE: 0.28744250535964966



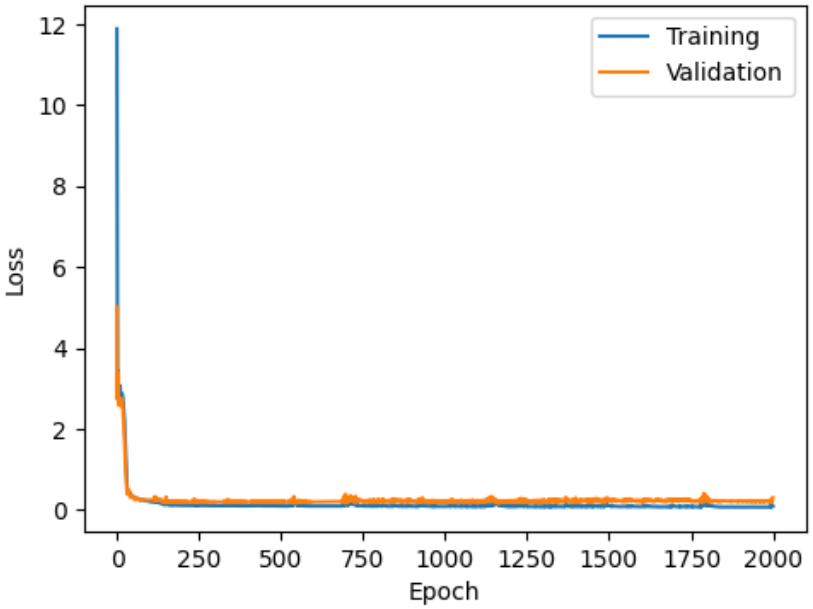


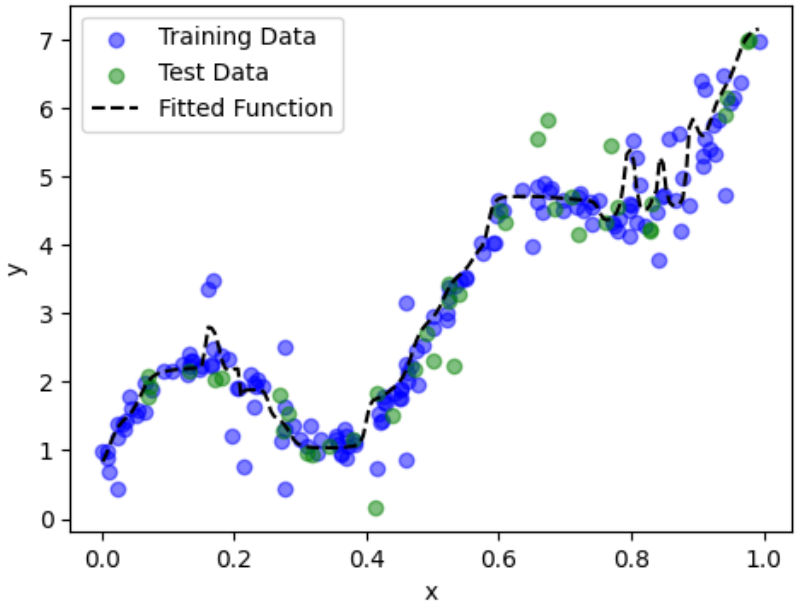
Epochs: 2000, MSE: 0.21675749123096466

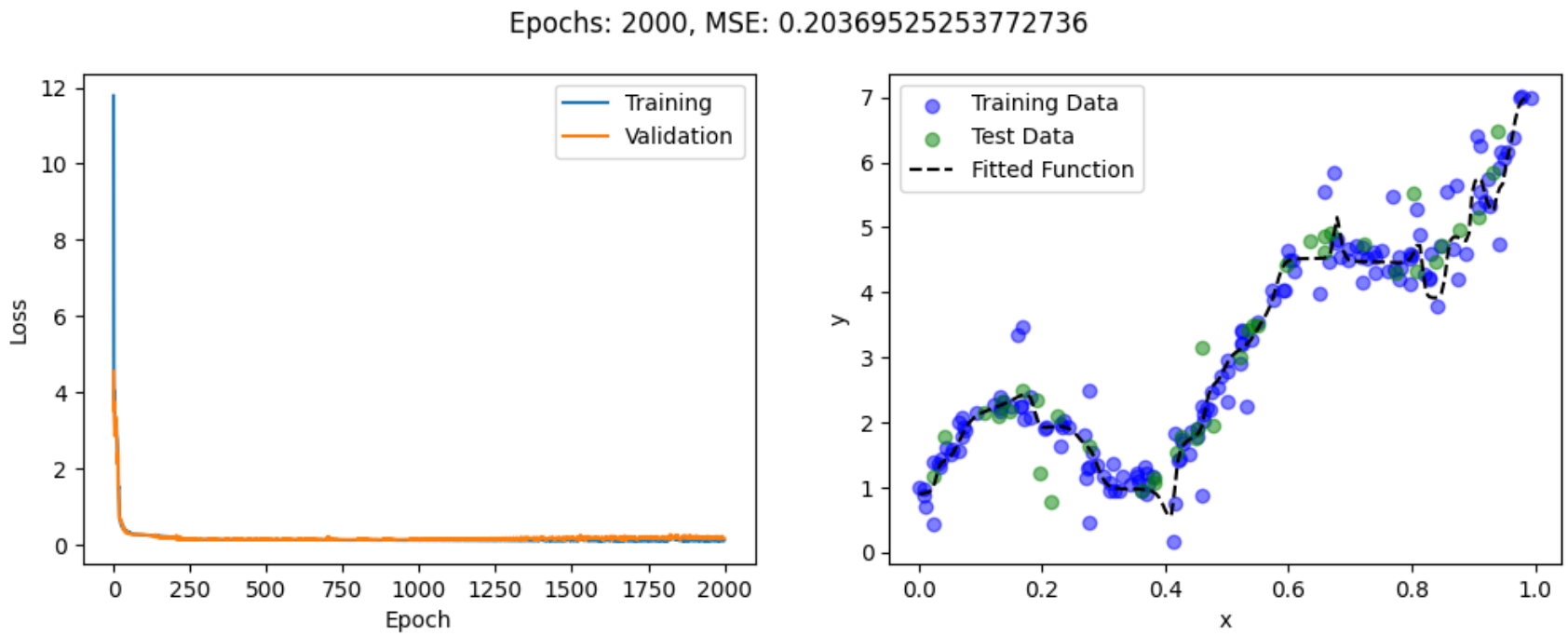




Epochs: 2000, MSE: 0.2449280470609665








```
The Average MSE loss for model trained with 100 epochs is =  0.2058341234922409
The Average MSE loss for model trained with 500 epochs is = 0.2386472851037979
The Average MSE loss for model trained with 2000 epochs is =  0.24151789844036103
```

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

Based on the average MSE results, training for 500 epochs yielded the most optimal model, suggesting it has appropriately captured the underlying data patterns without fitting to noise. Cross-validation enhances the evaluation process by providing insights into the model's ability to generalize across different subsets of data, which is essential for real-world applications. The consistency across all 5 folds for a given number of epochs indicates that the model is stable, exhibiting only minor variations in performance, which is indicative of good generalizability. In contrast, the model trained for 100 epochs likely didn't learn enough to

capture the complexity (underfit), while the model with 2000 epochs probably learned too much noise from the training data (overfit).