

Problem 1:

- 1

Problem 2:

- 4

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

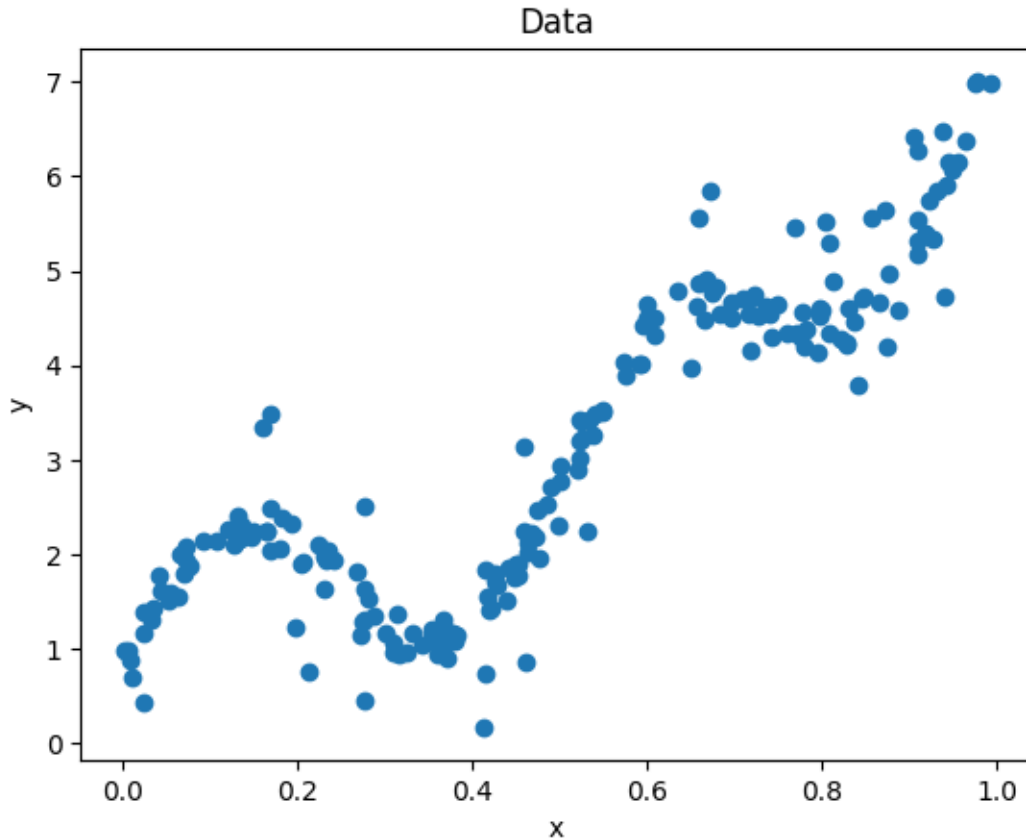
def plotLoss(ax, train_curve, val_curve):
    ax.plot(train_curve, label = 'Training')
    ax.plot(val_curve, label = 'Validation')
    ax.set_xlabel('Epoch')
    ax.set_ylabel('Loss')
    ax.legend()
```

```
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 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.

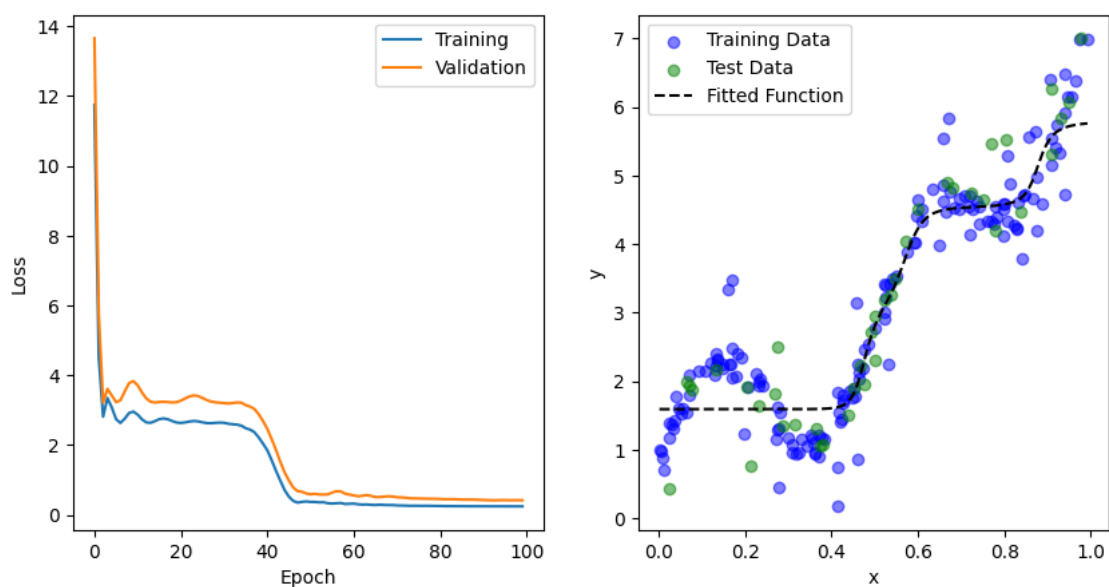
func(*args, **kwargs)

/opt/miniconda3/lib/python3.8/site-packages/IPython/core/pylabtools.py:152:

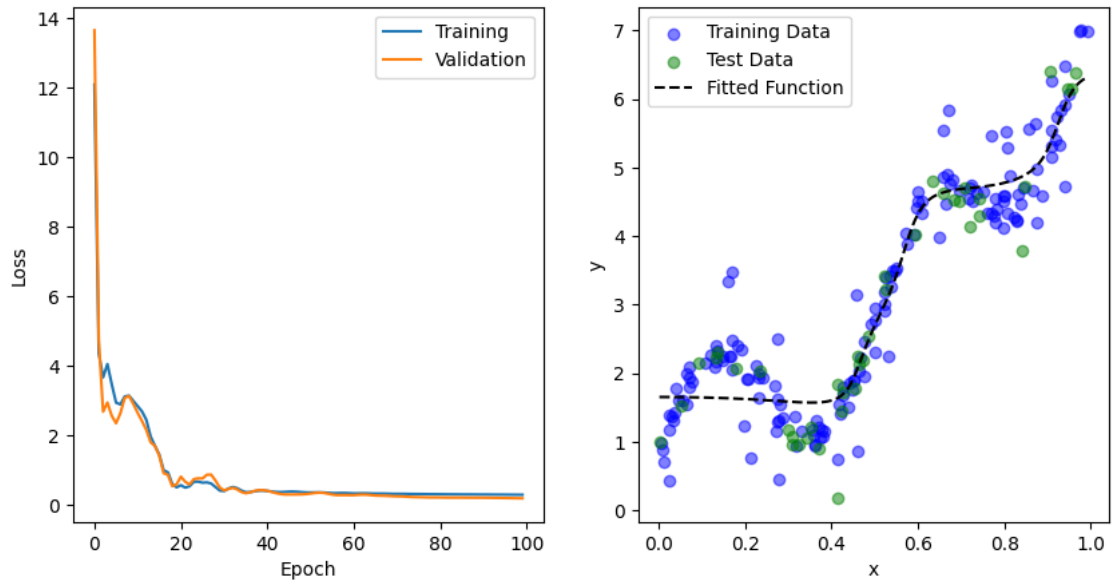
UserWarning: Glyph 9 () missing from current font.

fig.canvas.print_figure(bytes_io, **kw)

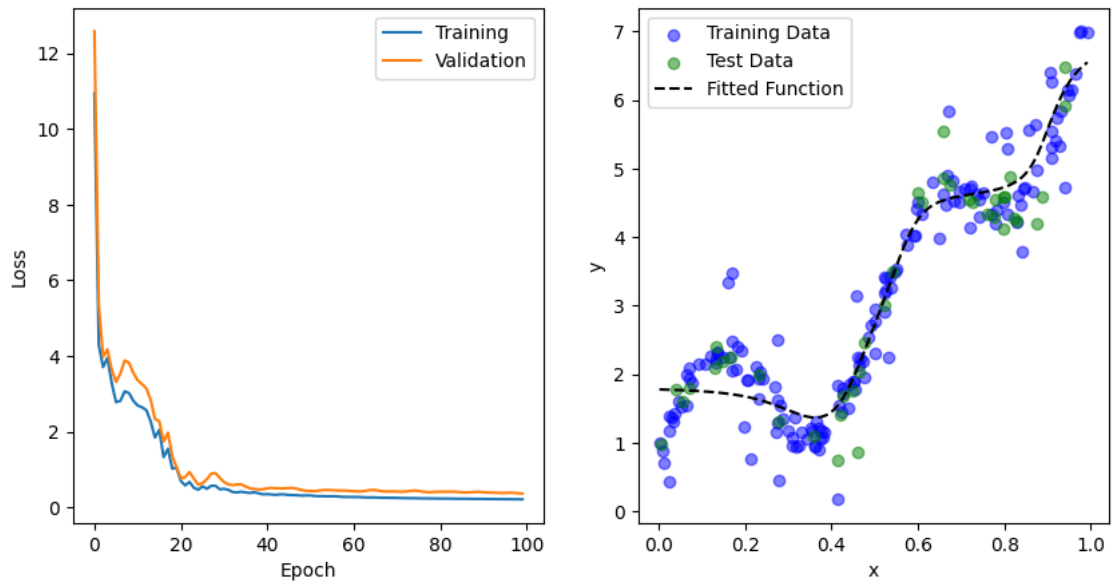
Number of Epochs: 100 □ MSE: 0.23691129007826328



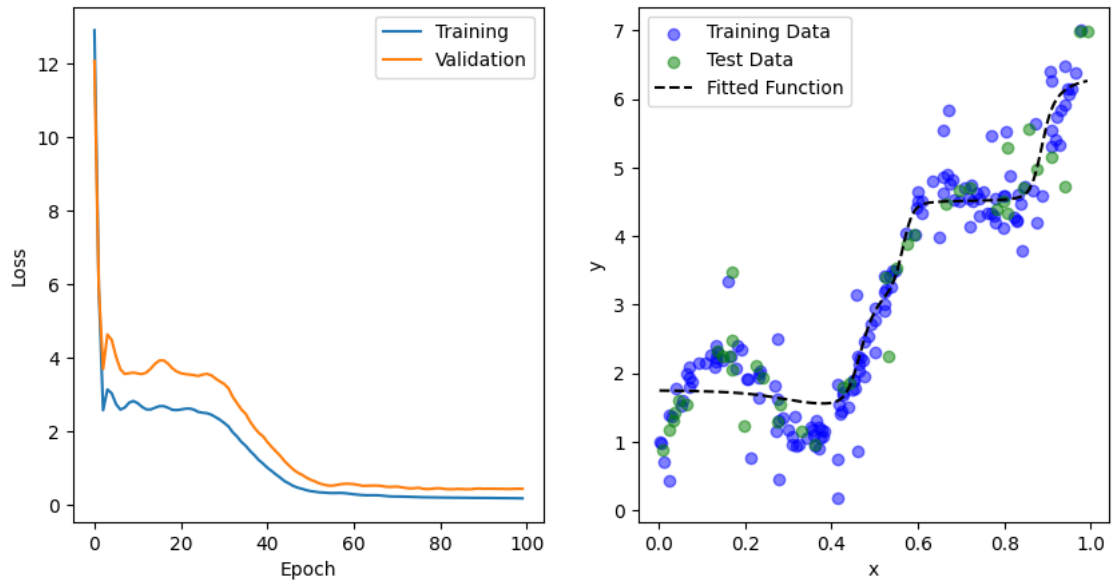
Number of Epochs: 100 \square MSE: 0.2384007980244977



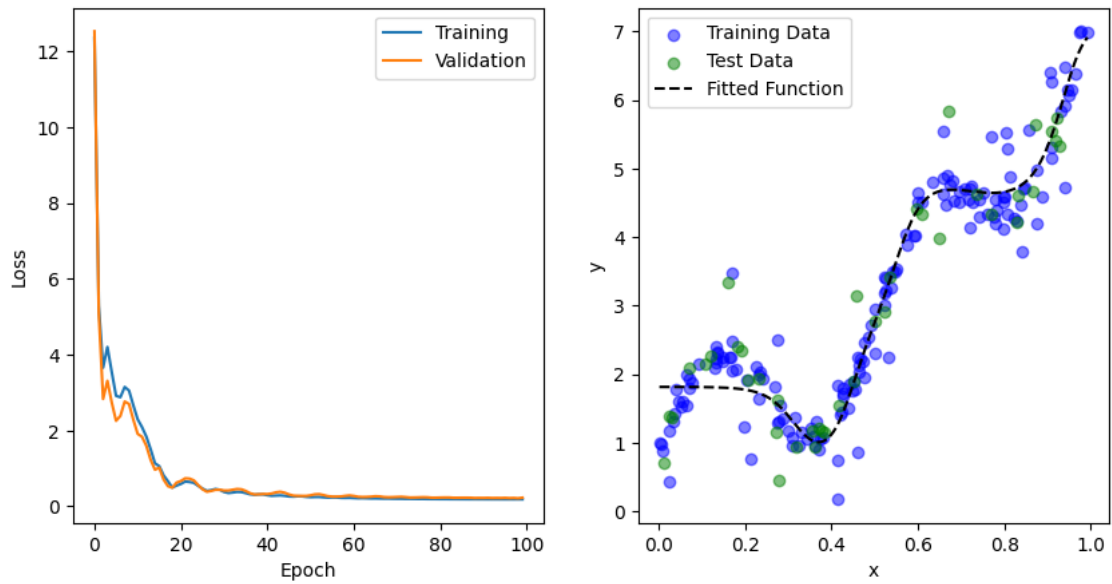
Number of Epochs: 100 \square MSE: 0.21689732728888372



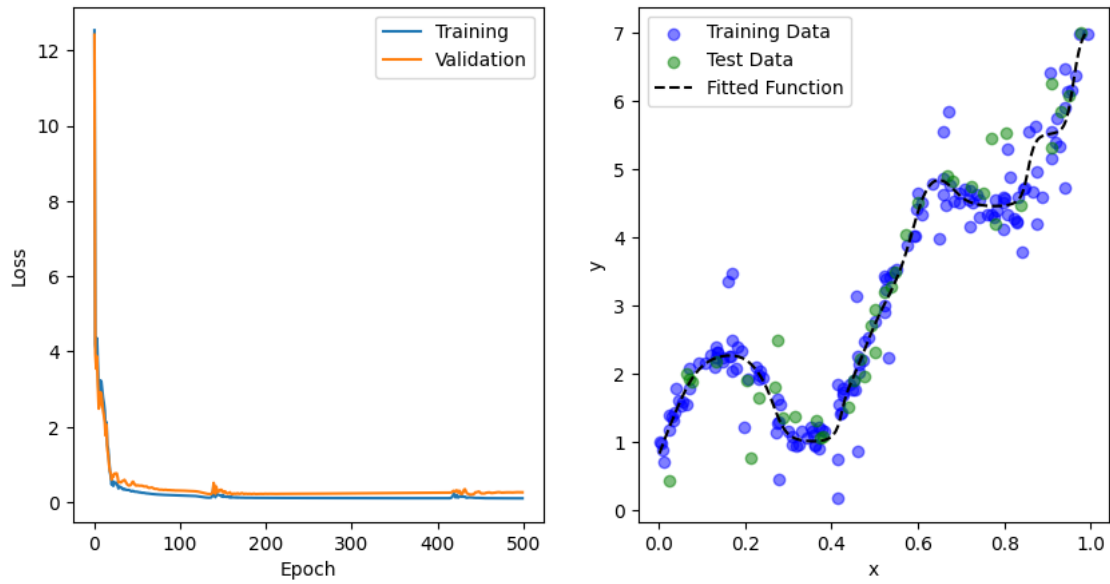
Number of Epochs: 100 \square MSE: 0.3355758543266182



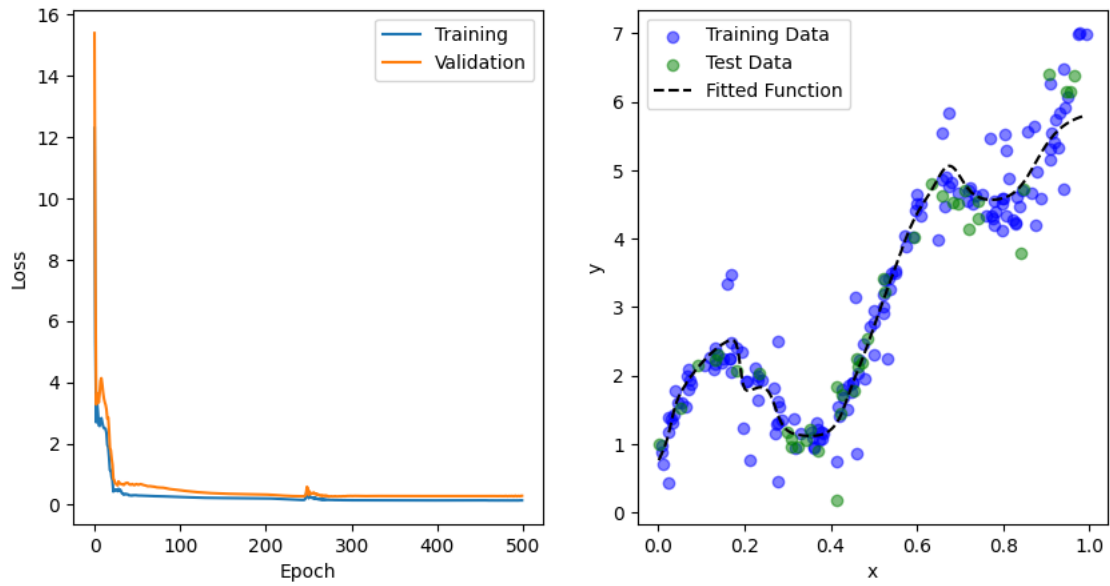
Number of Epochs: 100 \square MSE: 0.278066477337766



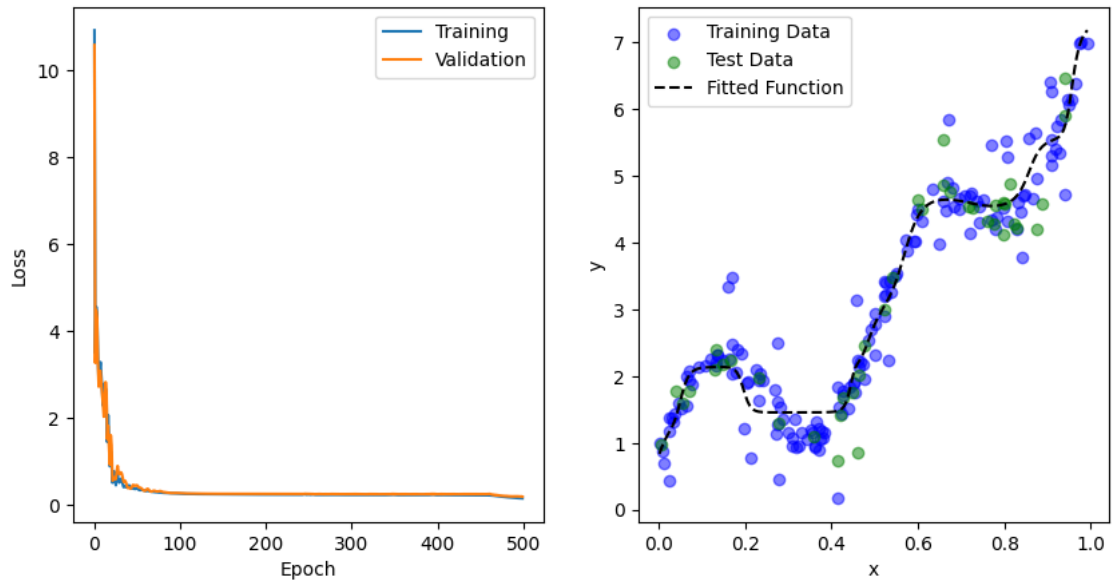
Number of Epochs: 500 \square MSE: 0.20002829004223957



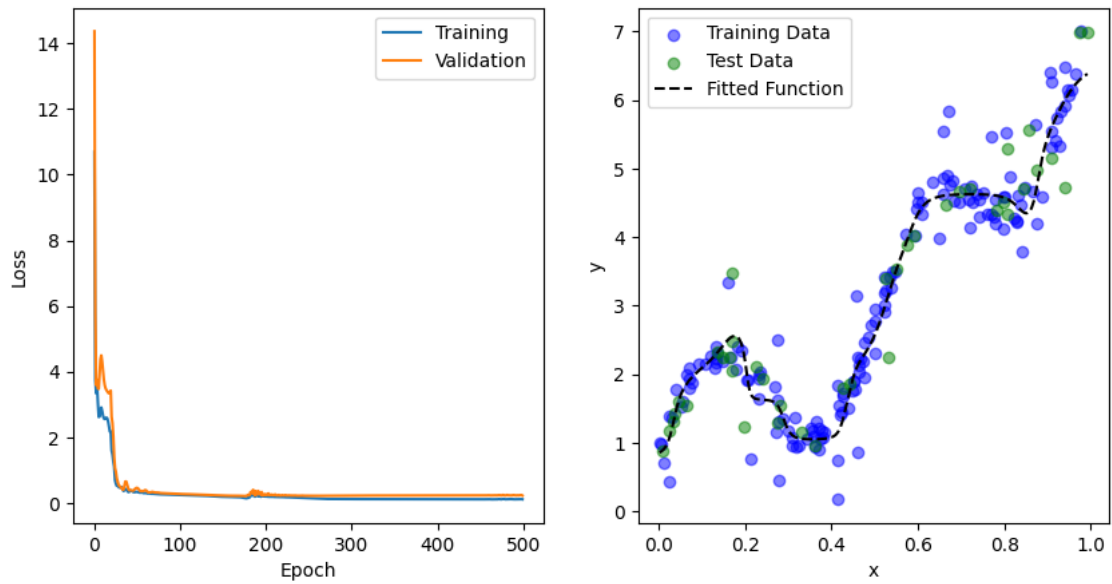
Number of Epochs: 500 \square MSE: 0.15364440102042778



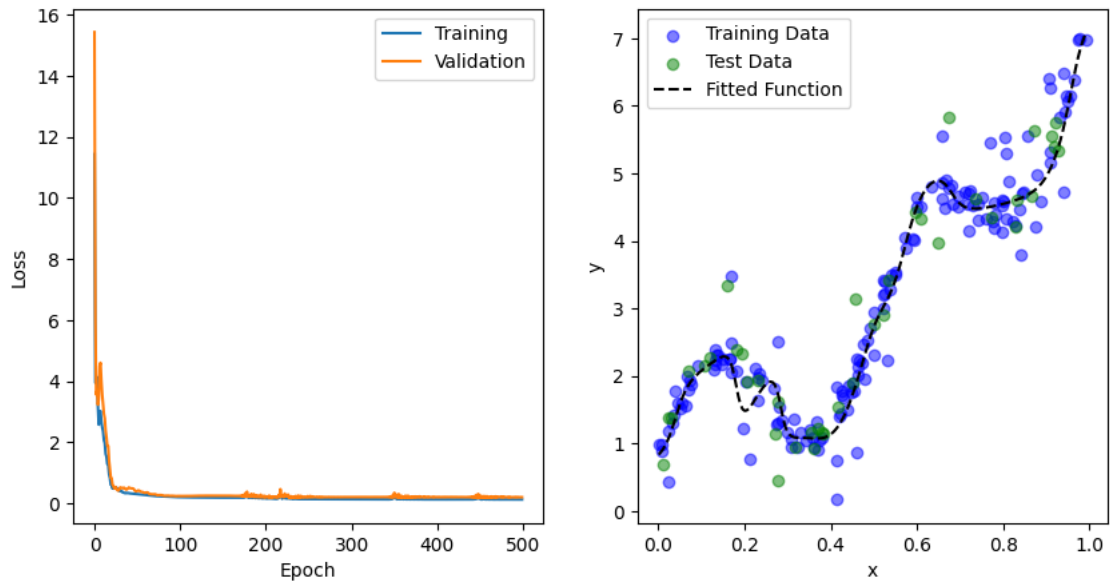
Number of Epochs: 500 \square MSE: 0.18759360782971696



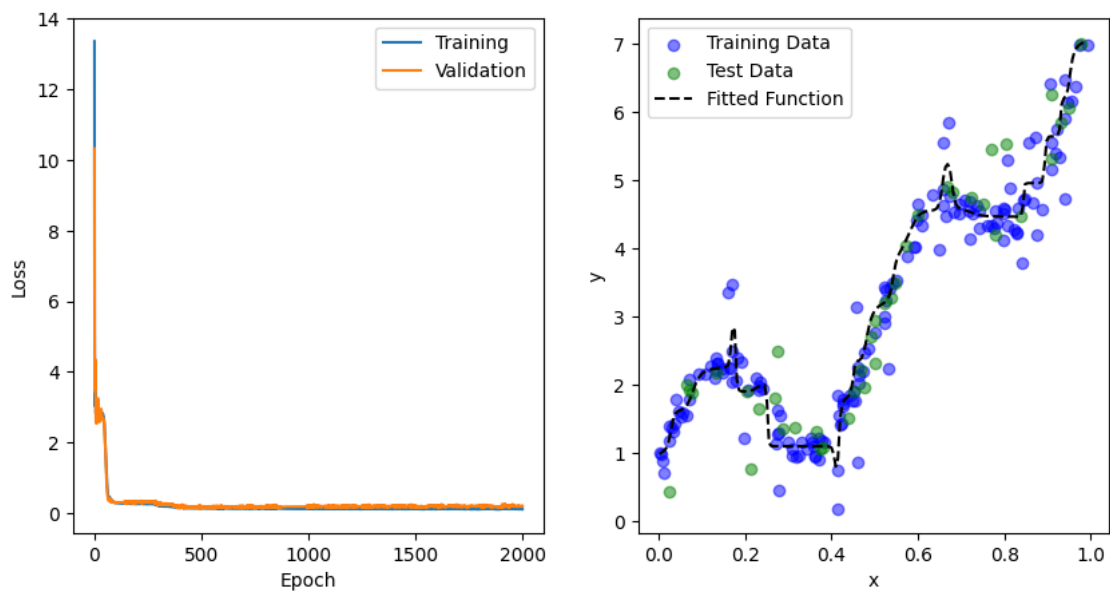
Number of Epochs: 500 \square MSE: 0.22630064291450677



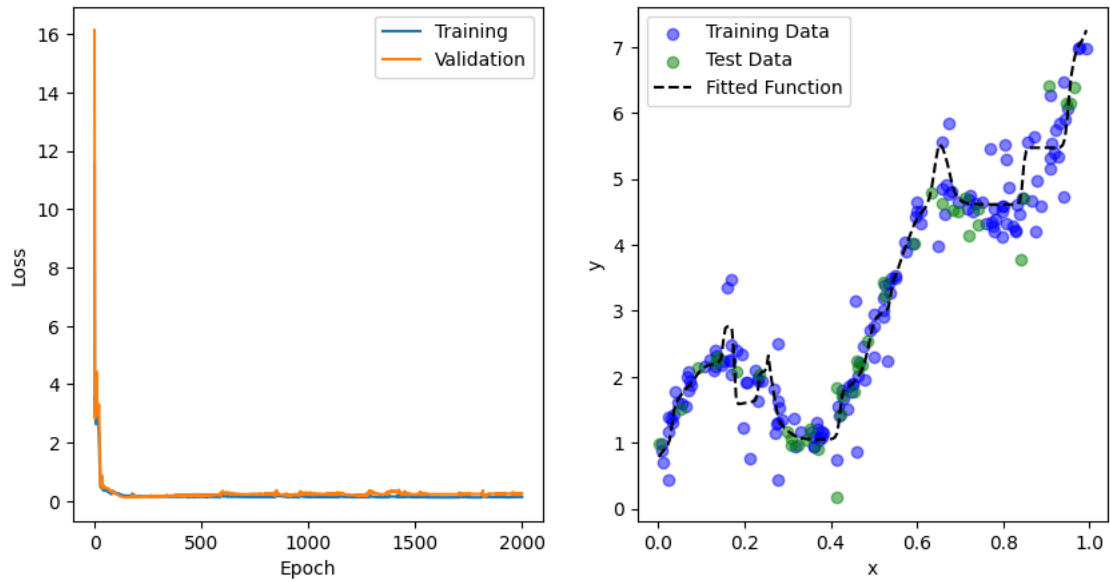
Number of Epochs: 500 \square MSE: 0.25335283556443045



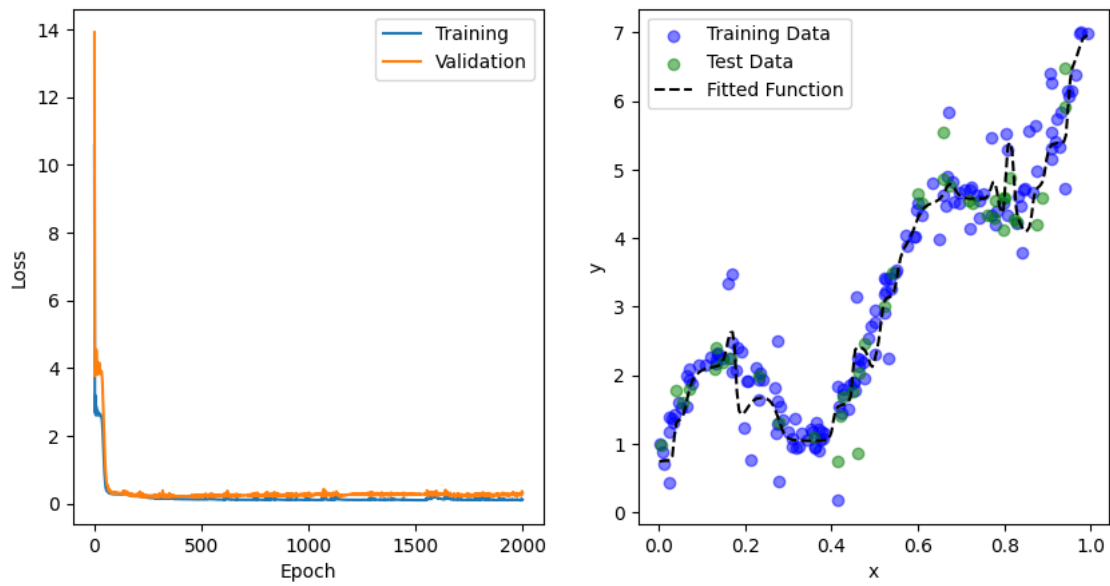
Number of Epochs: 2000 \square MSE: 0.23032430381731506



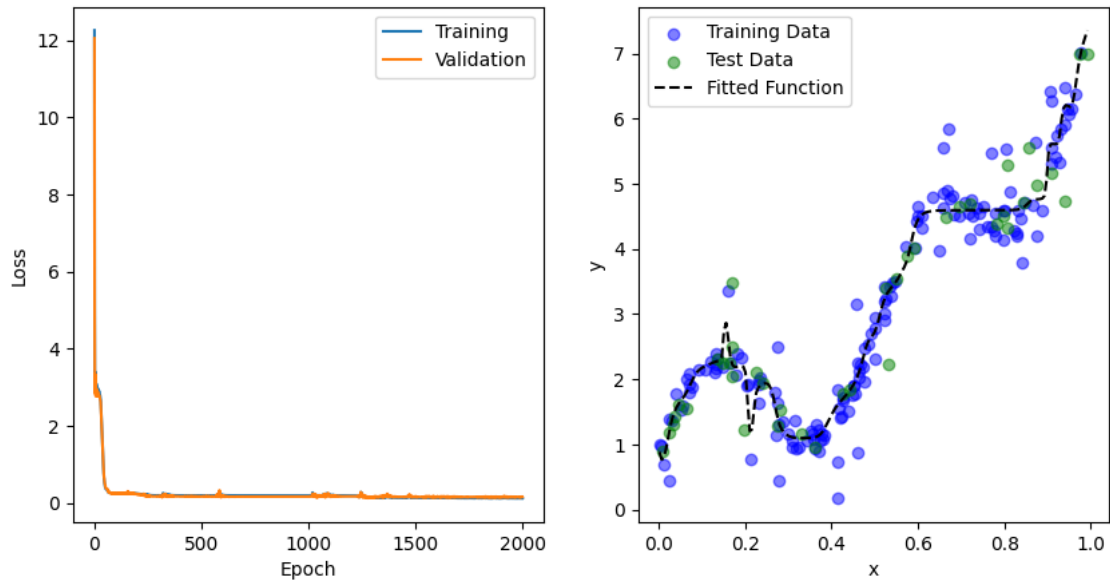
Number of Epochs: 2000 \square MSE: 0.15203049382591766



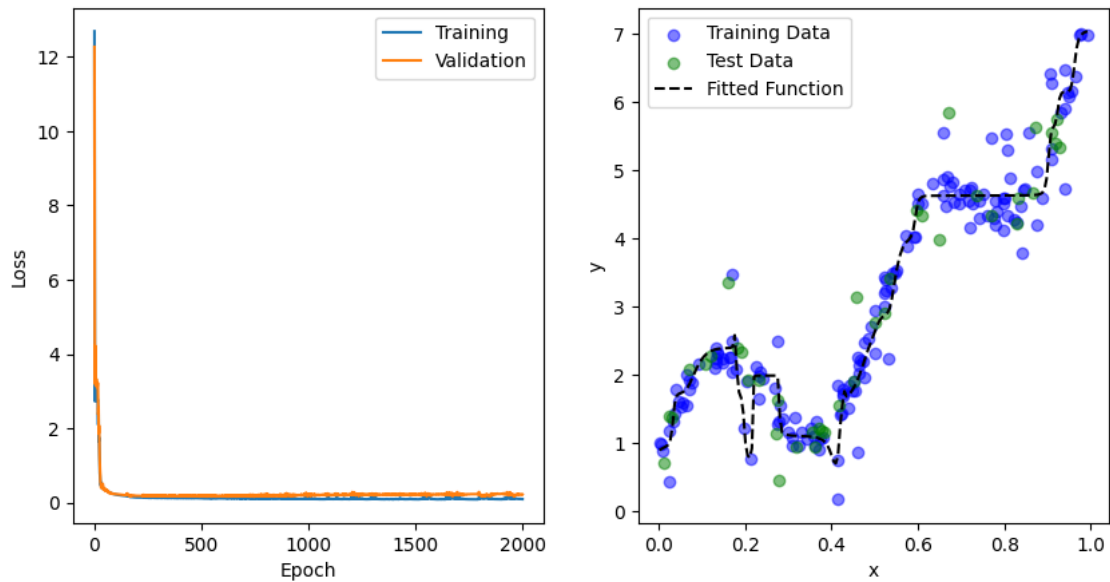
Number of Epochs: 2000 \square MSE: 0.20054423615514333



Number of Epochs: 2000 \square MSE: 0.20649100098246453



Number of Epochs: 2000 \square MSE: 0.3000057245371619



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 training scheme produced the best overall fits with the lowest average MSE. This is also noticeable 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 performance 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.

M10-L1-P1

November 18, 2023

1 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.

```
[ ]: 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.

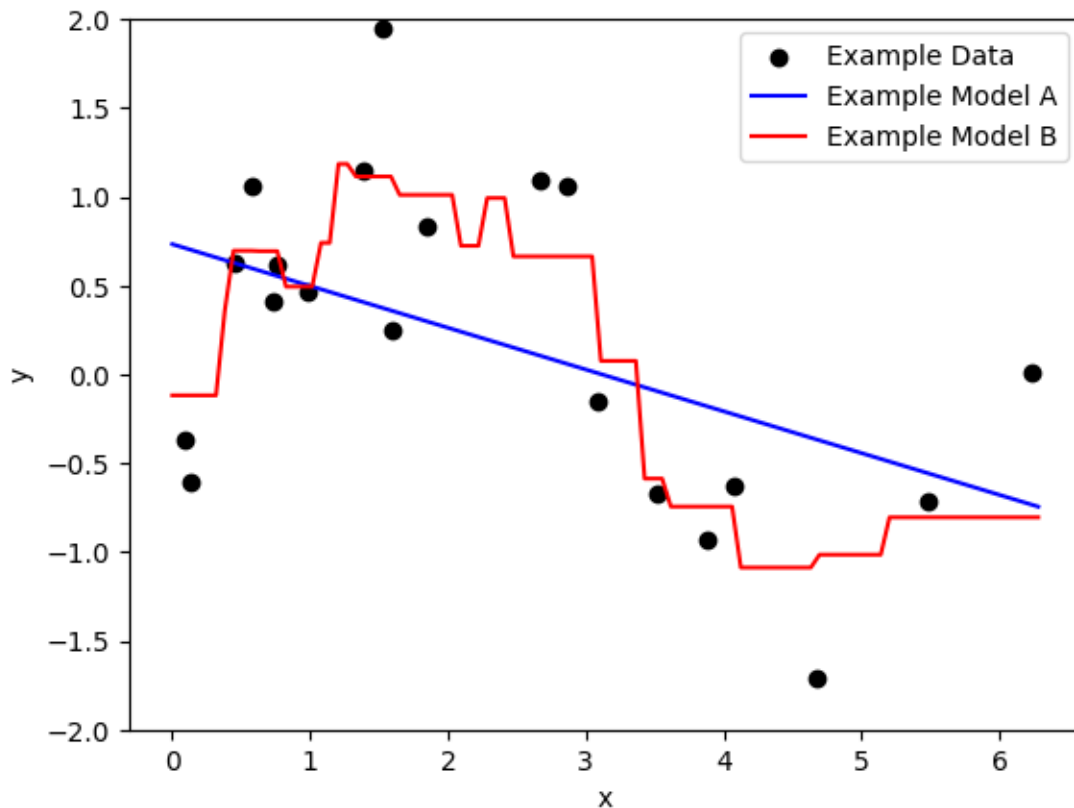
```
[ ]: modelA, modelB, x, y = train_models()
plt.figure()
```



```

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()

```



1.1 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()`.

```
[ ]: modelA, modelB, x, y = zip(*[train_models() for i in range(50)])
```

1.2 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

x_A = [eval_model_at_point(model, x) for model in modelA]
x_B = [eval_model_at_point(model, x) for model in modelB]
```

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

bias_A = np.mean(x_A) - yGT
var_A = np.mean(np.power(np.subtract(x_A, np.mean(x_A)), 2))

bias_B = np.mean(x_B) - yGT
var_B = np.mean(np.power(np.subtract(x_B, np.mean(x_B)), 2))

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 = -0.471,   Variance = 0.039
Model B:   Bias = 0.012,   Variance = 0.082
```

Questions

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

Model B has a smaller bias because it has a lower magnitude bias.

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

Model A has the lower variance at this point.

1.3 Plotting models

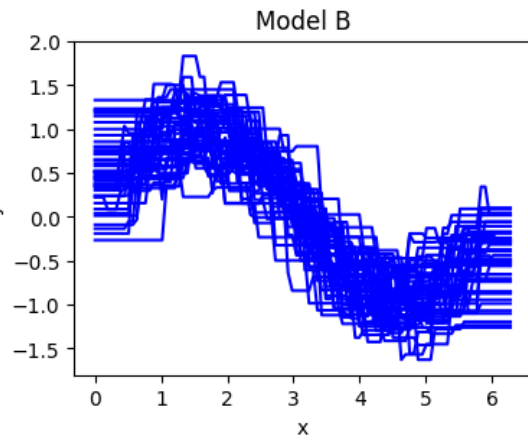
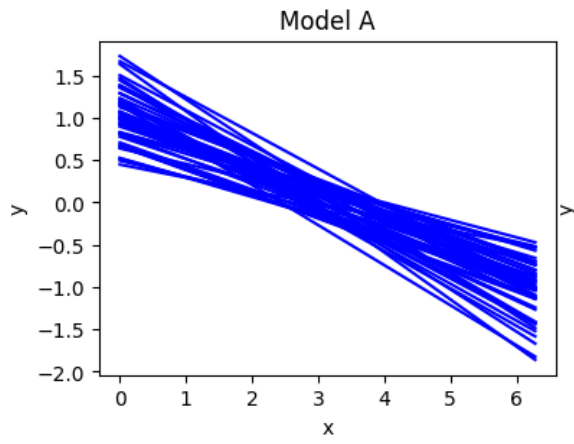
Now use the `plot_model` function to overlay all Model A predictions on one plot and all Model B predictions on another. Notice the spread of each model.

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

plt.subplot(1,2,1)
plt.title("Model A")
for model in modelA: plot_model(model, color="blue")

plt.subplot(1,2,2)
plt.title("Model B")
for model in modelB: plot_model(model, color="blue")

plt.show()
```



M10-L2-P1

November 18, 2023

1 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

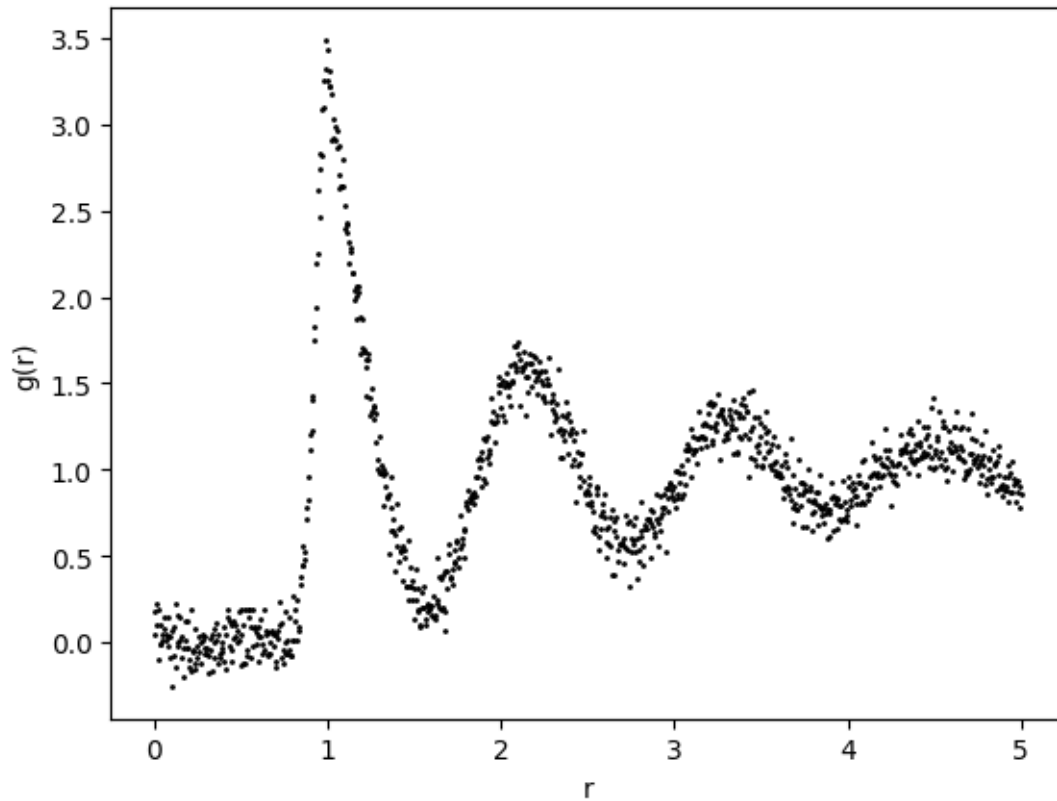
def plot_model(model,color="blue"):
    x = np.linspace(0, 5, 1000)
    y = model.predict(x.reshape(-1,1))
    plt.plot(x, y, color=color, linewidth=2, zorder=2)
    plt.xlabel("r")
    plt.ylabel("g(r)")

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()
```



1.1 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)
```

1.2 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: 0.2774937249705789
Validation score: 0.3104352357647895
Validation score: 0.20608404129798275
Validation score: 0.03790122395449669
Validation score: 0.1676244803676994
Validation score: 0.2202500372447742
Validation score: 0.14423712046918635
Validation score: 0.19894361702001595
Average validation score for Model 1: 0.19314468772616794
```

1.3 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.

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

scores2 = []
scores3 = []
```

```

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}")

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

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: 0.9135256064394239
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: 0.9281564779069266
Validation score: 0.95771300087561
Average validation score for Model 2: 0.9160114056393042
Validation score: 0.9629033605148645
Validation score: 0.9466107686883456
Validation score: 0.9518315048355762
Validation score: 0.9514051770741325
Validation score: 0.9229643307655355
Validation score: 0.9501422202077937
Validation score: 0.9322229519501161
Validation score: 0.9238931238090652
Validation score: 0.9461292855545795
Validation score: 0.9611128180031757
Average validation score for Model 3: 0.9449215541403186

```

2 Comparing models

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

Model 3 had the best performance.

Retrain this model on the full training dataset and report the R^2 score on training and testing data. Then complete the code to plot the model prediction with the data using the `plot_model` function.

```
[ ]: model3.fit(X_train, y_train)

print(f"Model 3 Train  $R^2$ : {model3.score(X_train, y_train)}")
print(f"Model 3 Train  $R^2$ : {model3.score(X_test, y_test)}")

plt.figure()
plot_data(X,y)
plot_model(model3)
plt.show()
```

Model 3 Train R^2 : 0.9547034601442719

Model 3 Train R^2 : 0.937186497441421

