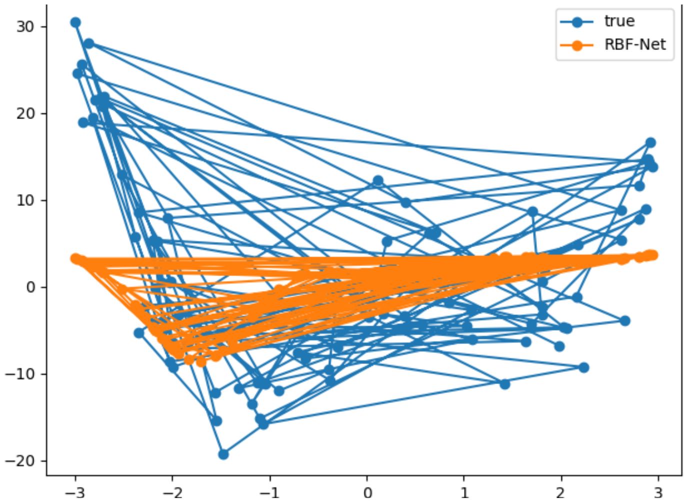
Homework

K is the number of cluster centers. that’s a hyperparameter
we have a complicated function, then we use a larger number of clusters. If we use larger number of clusters, then we’ll start overfitting.

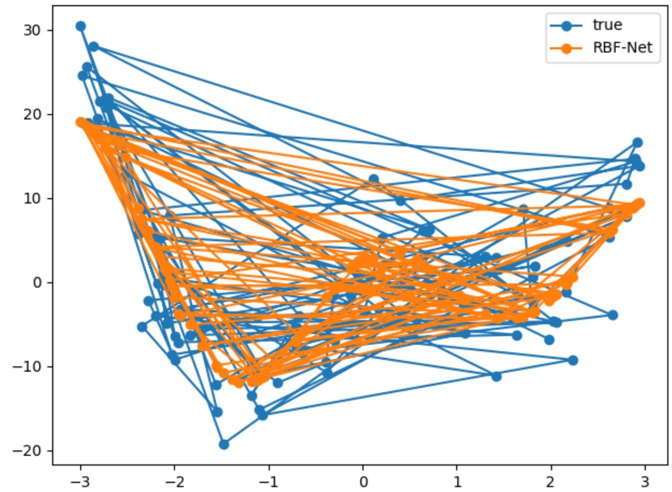
k= 3 Test MSE: 96.38560629475441
k=10 Test MSE: 90.87133355645386
k=20 Test MSE: 70.82392084282304
These are underfitting

k=60 Test MSE: 27.606281902204206
k=80 Test MSE: 49.481431377026794
Well-fitting

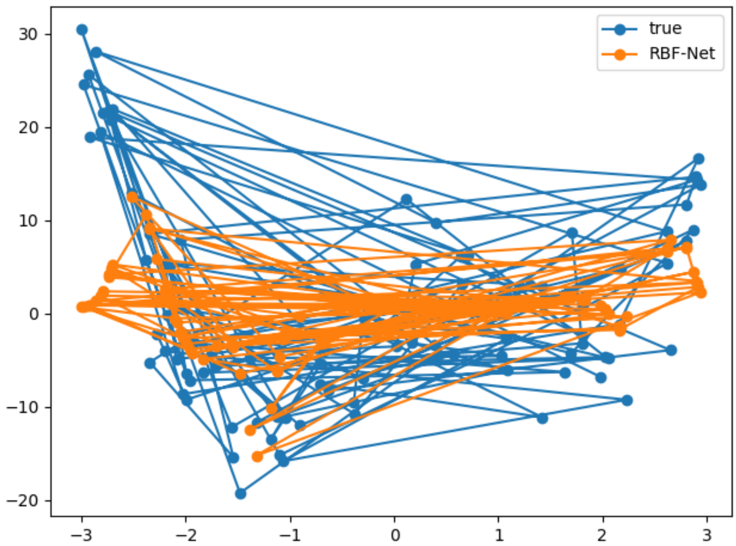
k=100 Test MSE: 103.51367245266853
overfitting



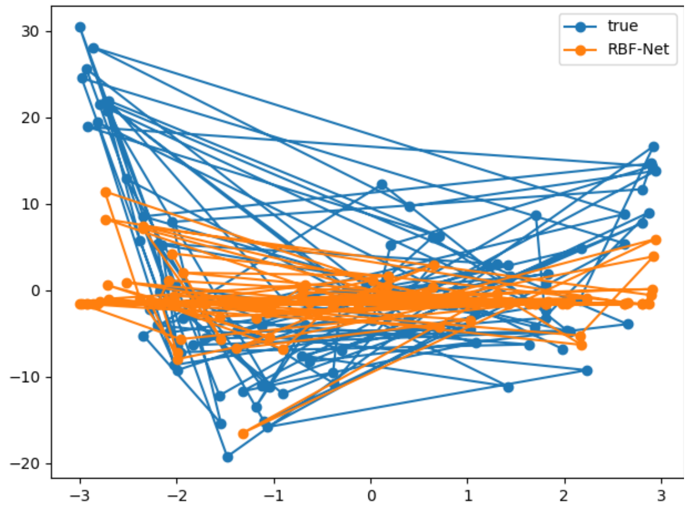
k=3



k=60



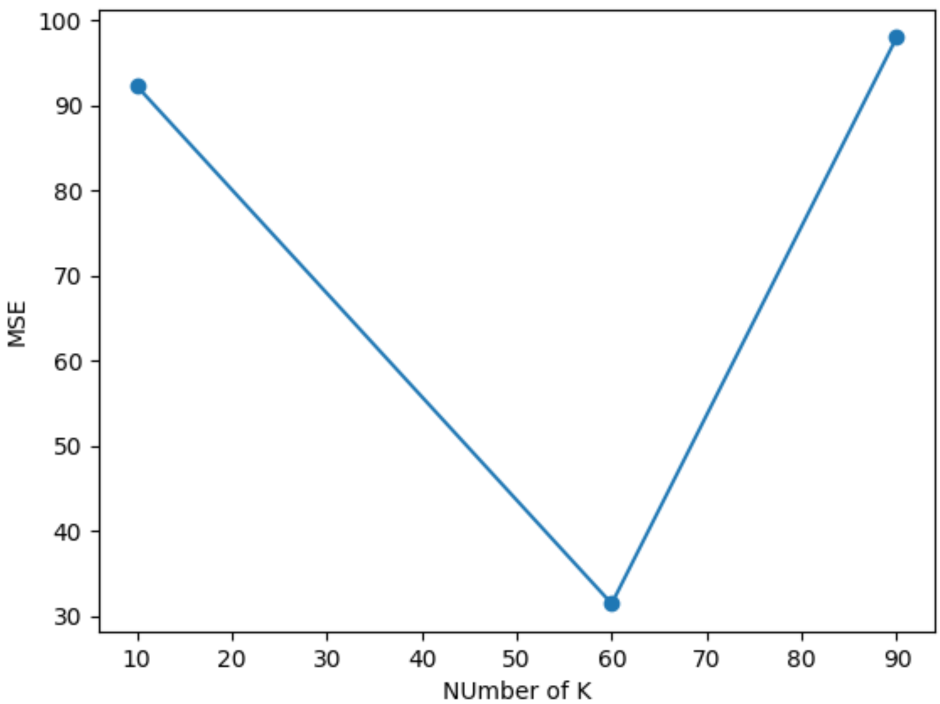
k=10



k=100

X = [10,60,90].   Number of K

Y = [92.25040699759252, 31.416488600678257, 97.96565589657301]. Average MSE



```
import numpy as np

def rbf(x, c, s):
    return np.exp(-1 / (2 * s**2) * (x-c)**2)

def kmeans(X, k):
    clusters = np.random.choice(np.squeeze(X), size=k)
    prevClusters = clusters.copy()
    stds = np.zeros(k)
    converged = False
    while not converged:
        distances = np.squeeze(np.abs(X[:, np.newaxis] - clusters[np.newaxis, :]))
        # find the cluster that's closest to each point
        closestCluster = np.argmin(distances, axis=1)
        # update clusters by taking the mean of all of the points assigned to that cluster
        for i in range(k):
            pointsForCluster = X[closestCluster == i]
            if len(pointsForCluster) > 0:
                clusters[i] = np.mean(pointsForCluster, axis=0)
        # converge if clusters haven't moved
        converged = np.linalg.norm(clusters - prevClusters) < 1e-6
        prevClusters = clusters.copy()
    distances = np.squeeze(np.abs(X[:, np.newaxis] - clusters[np.newaxis, :]))
    closestCluster = np.argmin(distances, axis=1)
    clustersWithNoPoints = []
    for i in range(k):
        pointsForCluster = X[closestCluster == i]
        if len(pointsForCluster) < 2:
            # keep track of clusters with no points or 1 point
            clustersWithNoPoints.append(i)
            continue
        else:
            stds[i] = np.std(X[closestCluster == i])
    # if there are clusters with 0 or 1 points, take the mean std of the other clusters
    if len(clustersWithNoPoints) > 0:
        pointsToAverage = []
        for i in range(k):
            if i not in clustersWithNoPoints:
                pointsToAverage.append(X[closestCluster == i])
        pointsToAverage = np.concatenate(pointsToAverage).ravel()
        stds[clustersWithNoPoints] = np.mean(np.std(pointsToAverage))
    return clusters, stds


class RBFNet(object):

    def __init__(self, k=2, lr=0.01, epochs=100, rbf=rbf, inferStds=True):
        self.k = k
        self.lr = lr
        self.epochs = epochs
        self.rbf = rbf
        self.inferStds = inferStds
        self.w = np.random.randn(k)
        self.b = np.random.randn(1)


    def fit(self, X, y):
        if self.inferStds:
            # compute stds from data
```

```
            self.centers, self.stds = kmeans(X, self.k)
        else:
            # use a fixed std
            self.centers, _ = kmeans(X, self.k)
            dMax = max([np.abs(c1 - c2) for c1 in self.centers for c2 in self.centers])
            self.stds = np.repeat(dMax / np.sqrt(2*self.k), self.k)
        # training
        for epoch in range(self.epochs):
            for i in range(X.shape[0]):
                # forward pass
                a = np.array([self.rbf(X[i], c, s) for c, s, in zip(self.centers, self.stds)])
                F = a.T.dot(self.w) + self.b
                loss = (y[i] - F).flatten() ** 2
                print('Loss: {0:.2f}'.format(loss[0]))
                # backward pass
                error = -(y[i] - F).flatten()
                # online update
                self.w = self.w - self.lr * a * error
                self.b = self.b - self.lr * error

    def predict(self, X):
        y_pred = []
        for i in range(X.shape[0]):
            a = np.array([self.rbf(X[i], c, s) for c, s, in zip(self.centers, self.stds)])
            F = a.T.dot(self.w) + self.b
            y_pred.append(F)
        return np.array(y_pred)

def load_txt(filename):
     dataset = list()
     with open(filename) as txt_file:
         my_list = txt_file.readlines()
         for row in my_list:
             dataset.append(row.split())
     return dataset

def convert_str_to_float(dataset, column):
    for row in dataset:
        row[column] = float(row[column].strip())


file_name_train = 'd_reg_tra.txt'
dataset_train = load_txt(file_name_train)

file_name_val = 'd_reg_val.txt'
dataset_val = load_txt(file_name_val)

column_numbers = len(dataset_train[0])
for i in range(column_numbers):
    convert_str_to_float(dataset_train, i)
    convert_str_to_float(dataset_val, i)

import pandas as pd
df_train = pd.DataFrame(dataset_train, columns = ['X', 'y'])
df_val = pd.DataFrame(dataset_val, columns = ['X', 'y'])


X_train= df_train.drop("y", axis= 1)
Y_train= df_train["y"]
X_test= df_val.drop("y", axis=1)
```

```
Y_test= df_val["y"]

X_train= X_train.to_numpy()
Y_train = Y_train.to_numpy()
X_test = X_test.squeeze()

rbfnet = RBFNet(lr=1e-2, k=80)
rbfnet.fit(X_train, Y_train)

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# mse =0
# for i in range(11):
y_pred2 = rbfnet.predict(X_test)
mse = mean_squared_error(Y_test, y_pred2[:,0])
# avg_mse = mse / 10
print("Test MSE:", mean_squared_error(Y_test, y_pred2[:,0]))
# print("Taverage test MSE:", mean_squared_error(Y_test, y_pred2[:,0]))

import matplotlib.pyplot as plt
plt.plot(X_test, Y_test, '-o', label='true')
plt.plot(X_test, y_pred2[:,0], '-o', label='RBF-Net')
plt.legend()
plt.tight_layout()
plt.show()


# # import matplotlib.pyplot as plt
# #  x values are crated by using above for loop
# Y = [92.25040699759252,  31.416488600678257, 97.96565589657301 ]
# X = [10,60,90]
# plt.plot(X, Y, '-o', label='true')
# plt.xlabel('NUmber of K')
# plt.ylabel('MSE')
# plt.show()
```