HW 3

Total points: 80

Shashank Rao - sr55952

Sarthak Shivnani -ss223347

If you prefer, you can work in groups of two. Please note that only one student per team needs to submit the assignment but make sure to include both students' names and UT EIDs.

For any question that requires a handwritten solution, you may upload scanned images of your solution in the notebook or attach them to the assignment . You may write your solution using markdown as well.

Please make sure your code runs and the graphs and figures are displayed in your notebook and PDF before submitting.

Q1. (15 points) Tensorflow Playground

In this question, you will be playing with Tensorflow Playground.

Select Classification as the Problem Type. Among the four datasets shown in DATA, please select the top left dataset.

Use the following settings as the DEFAULT settings for all subquestions:

- **Learning rate = 0.01**
- **Activation = ReLU**
- **Regularization = None**
- **Ratio of training to test data = 50%**
- **Noise = 0**
- **Batch Size = 30**
- **Input features as X_1 and X_2**
- **One hidden layer with 4 neurons**

For all questions below, it is ok if you did not run to the exact number of epochs specified as long as it is within $\pm$ 20 epochs. For example, if you are asked to run 1000 epochs and you ran 1012 epochs, it is fine because it's within 980 and 1020.

Part 1. (4 pts) Effect of activation function (keep other settings the same as in DEFAULT)

- **Using ReLU as the activation function**
- **Using the Linear activation function**

(a) Report the train and test losses for both at the end of 1000 epochs.\ (b) What qualitative difference do you observe in the decision boundaries obtained and what do you think is the reason for this?

Answer

(a)

Activation	ReLU	Linear
Train Loss	0.004	0.500
Test Loss	0.007	0.499

(b) When we use Linear as an activation function the decision boundary in linear , in this perticular senario where the data set looks circular the activation fuction may not be able to capture the complexity in the data While, When we use ReLU as an activation function it introduces a some non-linearity making the decision boundary non linear. This inturn can capture the complixity of the data which is reflected in its low train and test loss

In [ ]:

Part 2. (4 pts) **Effect of number of hidden units (keep other settings the same as in DEFAULT)**

- **Use 2 neurons in the hidden layer**
- **Use 8 neurons in the hidden layer**

(a) Report the train and test losses at the end of 1000 epochs.\ (b) What do you observe in terms of the decision boundary obtained as the number of neurons increases and what do you think is the reason for this?

Answer

(a)

# Hidden Units	2	8
Train Loss	0.277	0.006
Test Loss	0.278	0.007

(b)

- **when we used 2 neurons in the hidden layer decision boundary was more linear or simple, potentially underfitting the data.**
- **When we use 8 neurons it increases the model's capacity to learn complex dataset. decision boundary is more accuratley classifying**

Part 3. (4 pts) **Effect of Learning rate and number of epochs (keep other settings the same as in DEFAULT)**

Set the learning rate to the following numbers

- **10**
- **0.1**
- **0.0001**

(a) Report the train and test losses at the end of 100 epochs, 500 epochs and 1000 epochs respectively.\ (b) What do you observe from the change of loss vs learning rate, and the change of loss vs epoch numbers? Also report your observations on the training and test loss curve (observe if you see noise for certain learning rates and reason why this is happening).

Answer

(a)

Learning rate: 10

Epoch	100	500	1000
Train Loss	0.984	0.984	0.661
Test Loss	1.016	1.016	0.672

Learning rate: 0.1

Epoch	100	500	1000
Train Loss	0.005	0.001	0.000

Epoch	100	500	1000
Test Loss	.007	0.003	0.004

Learning rate: 0.0001

Epoch	100	500	1000
Train Loss	.442	0.429	0.411
Test Loss	0.434	0.418	0.398

Loss vs. Learning Rate

- Learning 10- observe high fluctuations or "noise" in the loss because the large step size overshoots the optimal point, leading to unstable training. It does not converge well, and in our case, the loss does not even increase over time.
- Learning 0.1 This learning rate is more stable and provides asmoother convergence. we see a decrease in both training and test loss over time.
- Learning 0.0001 Due to such a small learning rate, the model converges very slowly. Even after 1000 epochs, the loss does not decrease as much beacuse the gradient updates are small.

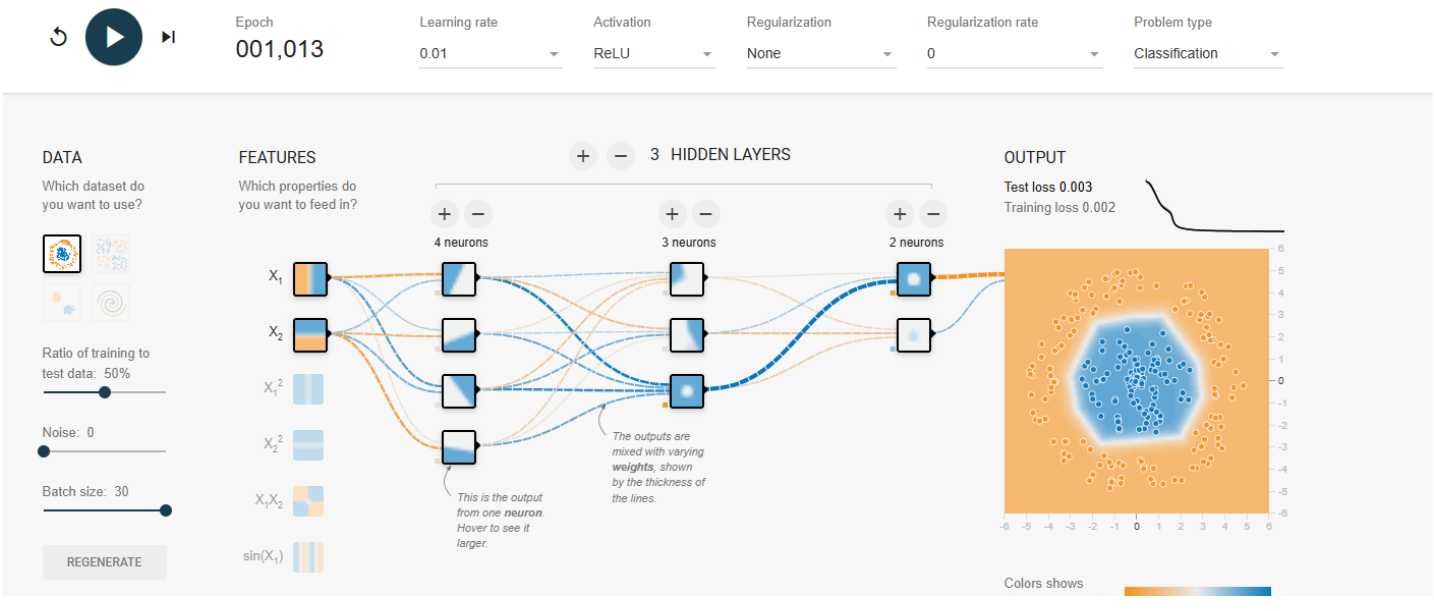
loss vs epoch numbers

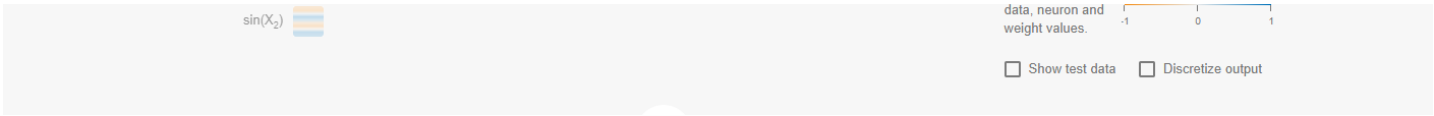
- At 100 epochs, losses are high for most learning rates as the model is still in early training.
- At 500 epochs, the losses decrease, especially for moderate learning rates (e.g., 0.1), as the model gradually learns the data.
- At 1000 epochs, depending on the learning rate, the model might have fully converged or still be learning in the case of very low learning rates.
- we observe noisy loss curves, there is a lot of fluctuation in the training and test loss. This may happen because the updates are too large, and the model is struggling to converge to a optimal point.

Noise: we observed Noisy curve when we used a high learning rate i.e. 10 high Noise: When the learning rate is this large, the model takes very big steps when updating the weights during gradient descent. This often results in the model overshooting the optimal point in the loss landscape. Instead of steadily decreasing, the loss fluctuated a lot, causing a noisy loss curve. The model may jumped between areas of higher and lower loss. This may happen because gradient descent is unable to make fine adjustments, and the learning path becomes unstable. Large learning rates can also lead to divergence, where the loss doesn't decrease over time.

Part 4. (3 pts) Use the DEFAULT setting but choose a configuration to modify (e.g. network depth, input features, noise, etc.)

(a) State the change you made and attach the screenshot showing your full network, output and the parameters.\
(b) Report the best train and test loss you obtain.\ (c) Briefly justify why the modification you chose lead to those results.





(a) For this we Added an additional hidden layer (Network Dept=3)

(b) Train Loss = 0.002 || Test Loss = 0.003

(c) Adding an extra hidden layer increases the model's capacity to learn more complex representations of the data.

Q2. (30 Points) - Coding Neural Networks with PyTorch

In this question we will code a simple MLP network with PyTorch and train it on the house cost dataset (provided in the supplementary files as *house_cost_data.csv*) that we used in HW1. We will then compare the results with linear regression. If you want to run it locally, please check out this link to install PyTorch. Otherwise, you can just use Google Colab.

Here is a tutorial for you to quickly to get familiar with PyTorch and finish the problems below.

This is a programming question. Please read through each subpart of this question carefully. You are required to add lines of code as specified in the code cells. Please carefully read through the comments in the code cells to identify what code is to be written, where it is to be written and how many lines of code are required. Code is to be added between the ## START CODE ## and ## END CODE ## comments and in place of the keyword None. In certain cases, the number of lines of code that are to be written will be specified. For example, ## START CODE ## (1 line of code) specifies that only 1 line of code is to be added between the ## START CODE ## and ## END CODE ## comments. In case there is no information on the required number of lines, you are allowed to add any number of lines of code.

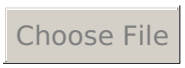
The following question covers a dataset for house cost and linear models in python. The categorical variables and rows with missing variables are removed to make it easier to run the models.

NOTE

- **Only use the following code block if you are using Google Colab. If you are using Jupyter Notebook, please ignore this code block. You can directly upload the file to your Jupyter Notebook file systems.**
- **It will prompt you to select a local file. Click on "Choose Files" then select and upload the file. Wait for the file to be 100% uploaded. You should see the name of the file once Colab has uploaded it.**

In [1]:

```
from google.colab import files
uploaded = files.upload()
```



No file selected

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving house_cost_data (1).csv to house_cost_data (1).csv
```

In [2]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
import tqdm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
import copy
```

```
%matplotlib inline
pd.options.mode.chained_assignment = None

df = pd.read_csv('/content/house_cost_data (1).csv')
X = df.drop(['house_cost'],axis=1).to_numpy()
Y = df['house_cost'].to_numpy()
```

Part 1 (2pts) - Preprocessing

Split the dataset into train(75% of data) and test(25% data) using the train_test_split function with random_state = 50.

In [3]:

```
##  START CODE  ## (1 line of code)
train_X, test_X, train_Y, test_Y = train_test_split(X, Y, test_size=0.25, random_state=5
0)
##  END CODE  ##
```

Scale the data (not including target) so that each of the independent variables would have zero mean and unit variance. You can use the StandardScaler() class for this.

In [4]:

```
##  START CODE  ##
scaler = StandardScaler()
train_X = scaler.fit_transform(train_X)
test_X = scaler.transform(test_X)
##  END CODE  ##
```

Split the train dataset into train(85% of train data) and validation(15% of train data) using the train_test_split function with random_state = 50.

In [5]:

```
##  START CODE  ## (1 line of code)
X_train, X_valid, y_train, y_valid = train_test_split(train_X, train_Y, test_size=0.15,
random_state=50)
##  END CODE  ##
```

Convert the numpy arrays to torch tensors. Make sure that the tensors are of dtype torch.float32. Also make sure that the target tensors (y) are 2-dimensional by using .reshape(-1,1) on the tensors.

In [6]:

```
# Convert to 2D PyTorch tensors
##  START CODE  ##
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).reshape(-1,1)

X_valid = torch.tensor(X_valid, dtype=torch.float32)
y_valid = torch.tensor(y_valid, dtype=torch.float32).reshape(-1,1)

X_test = torch.tensor(test_X, dtype=torch.float32)
y_test = torch.tensor(test_Y, dtype=torch.float32).reshape(-1,1)
##  END CODE  ##
X_train.shape, y_train.shape, X_valid.shape, y_valid.shape, X_test.shape, y_test.shape

##  END CODE  ##
```

Out[6]:

```
(torch.Size([13777, 15]),
 torch.Size([13777, 1]),
 torch.Size([2432, 15]),
 torch.Size([2432, 1]),
```

```
 torch.Size([5404, 15]),
 torch.Size([5404, 1]))
```

Part 2 (13 pts) - Train and Eval functions

(3 pts) First we will write and eval function which takes a model, input(X) and target(y) and evaluates the MSE and R2.

In [7]:

```
def eval(model, X, y):
    model.eval()
    ##  START CODE  ##
    with torch.no_grad():
        # Predict using the model
        predictions = model(X)

        # Convert predictions and targets to numpy for metrics calculation
        y_true = y.numpy()
        y_pred = predictions.numpy()

        # Calculate MSE and R² score
        mse = mean_squared_error(y_true, y_pred)
        r2 = r2_score(y_true, y_pred)
    ##  END CODE  ##
    return mse, r2
```

(10 pts) Now we will write the train function. Refer here for the tutorial. Make sure to understand what the optimizer does.

In [8]:

```
def train(model, X_train, y_train, X_valid, y_valid, n_epochs, batch_size, optimizer, lo
ss_fn):
    # parameters to store the best model on validation MSE
    best_mse = np.inf   # init to infinity
    best_weights = None
    train_history = []
    valid_history = []
    batch_start = torch.arange(0, len(X_train), batch_size)

    for epoch in range(n_epochs):
        model.train()
        with tqdm.tqdm(batch_start, unit="batch", mininterval=0, disable=False) as bar:
            bar.set_description(f"Epoch {epoch}")
            train_losses = []
            for start in bar:
                # take a batch with the help of 'start' variable
                ##  START CODE  ## (2 lines of code)
                X_batch = X_train[start:start+batch_size]
                y_batch = y_train[start:start+batch_size]
                ##  END CODE  ##

                # forward pass on the batch
                ##  START CODE  ## (1 line of code)
                y_pred = model(X_batch)
                ##  END CODE  ##

                # calculate loss using loss_fn
                ##  START CODE  ## (1 line of code)
                loss = loss_fn(y_pred, y_batch)
                ##  END CODE  ##

                # backward pass
                ##  START CODE  ## (2 lines of code)
                optimizer.zero_grad()
                loss.backward()
                ##  END CODE  ##
```

```
                # update weights (1 line of code)
                ##  START CODE  ##
                optimizer.step()
                ##  END CODE  ##

                # print progress
                bar.set_postfix(mse=float(loss))
                train_losses.append(float(loss))
        train_history.append(sum(train_losses)/len(train_losses))
        # evaluate validation MSE and R2 at end of each epoch
        ##  START CODE  ## (1 line of code)
        mse, r2 = eval(model, X_valid, y_valid)
        ##  END CODE  ##

        # store the best model depending on MSE
        valid_history.append(mse)
        if mse < best_mse:
            ##  START CODE  ## (when storing best model weights make sure to use copy.dee
pcopy())
            best_mse = mse
            best_weights = copy.deepcopy(model.state_dict())
            ##  END CODE  ##

    print("Validation MSE: %.2f" % best_mse)
    print("Validation RMSE: %.2f" % np.sqrt(best_mse))
    return best_weights, train_history, valid_history, best_mse
```

Part 3 (15 pts) - Train the model and analyze results

(5 pts) Initialize a model with 2 hidden layers of size 64 and 16 and ReLU activation function. Use the MSELoss and Adam optimizer from PyTorch to train the model. Sweep learning rates in [10, 1, 0.1, 0.01] and find the best learning rate with the smaller validation set MSE. For the best model, evaluate the performance on the test set and plot training curves(loss vs epochs).

In [9]:

```
best_valid_mse = np.inf
best_weights = None
best_train_history = None
best_valid_history = None
best_lr = None
for lr in [10, 1, 0.1, 0.01]:
    print("Running with LR: ", lr)
    # Define the model using nn.Sequential
    ##  START CODE  ##
    model = nn.Sequential(
        nn.Linear(X_train.shape[1], 64),
        nn.ReLU(),
        nn.Linear(64, 16),
        nn.ReLU(),
        nn.Linear(16, 1)
    )
    ##  END CODE  ##

    #  Define loss function and optimizer
    ##  START CODE  ##
    loss_fn = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)
    ##  END CODE  ##

    n_epochs = 50   # number of epochs to run
    batch_size = 256  # size of each batch

    # train the model
    weights, train_history, valid_history, valid_mse = train(model, X_train, y_train, X_
valid, y_valid, n_epochs, batch_size, optimizer, loss_fn)

    # store values for best LR
```

```
    if valid_mse < best_valid_mse:
        #TODO
        best_valid_mse = valid_mse
        best_weights = weights
        best_train_history = train_history
        best_valid_history = valid_history
        best_lr = lr

print("Best LR: ", best_lr)
print("Best Validation MSE: ", best_valid_mse)
```

```
Running with LR:  10

Epoch 0: 100%|██████████| 54/54 [00:00<00:00, 79.46batch/s, mse=2.99e+10]
Epoch 1: 100%|██████████| 54/54 [00:00<00:00, 96.45batch/s, mse=2.36e+10]
Epoch 2: 100%|██████████| 54/54 [00:00<00:00, 94.86batch/s, mse=3.63e+10]
Epoch 3: 100%|██████████| 54/54 [00:00<00:00, 86.42batch/s, mse=2.61e+10]
Epoch 4: 100%|██████████| 54/54 [00:00<00:00, 77.51batch/s, mse=2.55e+10]
Epoch 5: 100%|██████████| 54/54 [00:00<00:00, 89.87batch/s, mse=2.48e+10]
Epoch 6: 100%|██████████| 54/54 [00:00<00:00, 84.95batch/s, mse=2.67e+10]
Epoch 7: 100%|██████████| 54/54 [00:00<00:00, 89.48batch/s, mse=2.2e+10]
Epoch 8: 100%|██████████| 54/54 [00:00<00:00, 83.14batch/s, mse=2.25e+10]
Epoch 9: 100%|██████████| 54/54 [00:00<00:00, 88.21batch/s, mse=2.53e+10]
Epoch 10: 100%|██████████| 54/54 [00:00<00:00, 80.80batch/s, mse=2.37e+10]
Epoch 11: 100%|██████████| 54/54 [00:00<00:00, 97.06batch/s, mse=2.03e+10]
Epoch 12: 100%|██████████| 54/54 [00:00<00:00, 103.57batch/s, mse=1.9e+10]
Epoch 13: 100%|██████████| 54/54 [00:00<00:00, 95.18batch/s, mse=2.26e+10]
Epoch 14: 100%|██████████| 54/54 [00:00<00:00, 95.85batch/s, mse=2.05e+10]
Epoch 15: 100%|██████████| 54/54 [00:00<00:00, 106.85batch/s, mse=2.18e+10]
Epoch 16: 100%|██████████| 54/54 [00:00<00:00, 105.22batch/s, mse=2.62e+10]
Epoch 17: 100%|██████████| 54/54 [00:00<00:00, 100.79batch/s, mse=2.07e+10]
Epoch 18: 100%|██████████| 54/54 [00:00<00:00, 94.91batch/s, mse=2.31e+10]
Epoch 19: 100%|██████████| 54/54 [00:00<00:00, 96.61batch/s, mse=2.18e+10]
Epoch 20: 100%|██████████| 54/54 [00:00<00:00, 88.32batch/s, mse=2.24e+10]
Epoch 21: 100%|██████████| 54/54 [00:00<00:00, 97.48batch/s, mse=2.24e+10]
Epoch 22: 100%|██████████| 54/54 [00:00<00:00, 95.19batch/s, mse=2.52e+10]
Epoch 23: 100%|██████████| 54/54 [00:00<00:00, 95.47batch/s, mse=1.86e+10]
Epoch 24: 100%|██████████| 54/54 [00:00<00:00, 105.07batch/s, mse=2.22e+10]
Epoch 25: 100%|██████████| 54/54 [00:00<00:00, 105.64batch/s, mse=2.19e+10]
Epoch 26: 100%|██████████| 54/54 [00:00<00:00, 91.99batch/s, mse=2.38e+10]
Epoch 27: 100%|██████████| 54/54 [00:00<00:00, 93.86batch/s, mse=2.36e+10]
Epoch 28: 100%|██████████| 54/54 [00:00<00:00, 96.76batch/s, mse=2e+10]
Epoch 29: 100%|██████████| 54/54 [00:00<00:00, 99.19batch/s, mse=2.39e+10]
Epoch 30: 100%|██████████| 54/54 [00:00<00:00, 98.07batch/s, mse=2.29e+10]
Epoch 31: 100%|██████████| 54/54 [00:00<00:00, 100.04batch/s, mse=2.38e+10]
Epoch 32: 100%|██████████| 54/54 [00:00<00:00, 115.24batch/s, mse=2.27e+10]
Epoch 33: 100%|██████████| 54/54 [00:00<00:00, 94.86batch/s, mse=2.67e+10]
Epoch 34: 100%|██████████| 54/54 [00:00<00:00, 95.85batch/s, mse=2.01e+10]
Epoch 35: 100%|██████████| 54/54 [00:00<00:00, 92.37batch/s, mse=1.88e+10]
Epoch 36: 100%|██████████| 54/54 [00:00<00:00, 98.19batch/s, mse=1.97e+10]
Epoch 37: 100%|██████████| 54/54 [00:00<00:00, 92.79batch/s, mse=2.59e+10]
Epoch 38: 100%|██████████| 54/54 [00:00<00:00, 89.72batch/s, mse=2e+10]
Epoch 39: 100%|██████████| 54/54 [00:00<00:00, 80.30batch/s, mse=1.91e+10]
Epoch 40: 100%|██████████| 54/54 [00:00<00:00, 92.23batch/s, mse=2.27e+10]
Epoch 41: 100%|██████████| 54/54 [00:00<00:00, 87.21batch/s, mse=2.12e+10]
Epoch 42: 100%|██████████| 54/54 [00:00<00:00, 108.73batch/s, mse=2.12e+10]
Epoch 43: 100%|██████████| 54/54 [00:00<00:00, 100.76batch/s, mse=2.17e+10]
Epoch 44: 100%|██████████| 54/54 [00:00<00:00, 99.83batch/s, mse=2.3e+10]
Epoch 45: 100%|██████████| 54/54 [00:00<00:00, 98.72batch/s, mse=1.97e+10]
Epoch 46: 100%|██████████| 54/54 [00:00<00:00, 100.12batch/s, mse=1.82e+10]
Epoch 47: 100%|██████████| 54/54 [00:00<00:00, 90.93batch/s, mse=2.15e+10]
Epoch 48: 100%|██████████| 54/54 [00:00<00:00, 99.29batch/s, mse=1.97e+10]
Epoch 49: 100%|██████████| 54/54 [00:00<00:00, 96.07batch/s, mse=1.76e+10]

Validation MSE: 30998196224.00
Validation RMSE: 176063.05
Running with LR:  1

Epoch 0: 100%|██████████| 54/54 [00:00<00:00, 75.02batch/s, mse=2.75e+10]
Epoch 1: 100%|██████████| 54/54 [00:00<00:00, 91.15batch/s, mse=2.8e+10]
Epoch 2: 100%|██████████| 54/54 [00:00<00:00, 85.50batch/s, mse=3.1e+10]
Epoch 3: 100%|██████████| 54/54 [00:00<00:00, 64.96batch/s, mse=3.31e+10]
Epoch 4: 100%|██████████| 54/54 [00:00<00:00, 79.32batch/s, mse=3.38e+10]
```

```
Epoch 5: 100%|██████████| 54/54 [00:00<00:00, 97.33batch/s, mse=3.37e+10]
Epoch 6: 100%|██████████| 54/54 [00:00<00:00, 94.36batch/s, mse=3.34e+10]
Epoch 7: 100%|██████████| 54/54 [00:00<00:00, 98.64batch/s, mse=3.34e+10]
Epoch 8: 100%|██████████| 54/54 [00:00<00:00, 106.17batch/s, mse=3.33e+10]
Epoch 9: 100%|██████████| 54/54 [00:00<00:00, 98.19batch/s, mse=3.33e+10]
Epoch 10: 100%|██████████| 54/54 [00:00<00:00, 104.98batch/s, mse=3.32e+10]
Epoch 11: 100%|██████████| 54/54 [00:00<00:00, 105.89batch/s, mse=3.31e+10]
Epoch 12: 100%|██████████| 54/54 [00:00<00:00, 101.23batch/s, mse=3.31e+10]
Epoch 13: 100%|██████████| 54/54 [00:00<00:00, 104.44batch/s, mse=3.26e+10]
Epoch 14: 100%|██████████| 54/54 [00:00<00:00, 108.42batch/s, mse=3.27e+10]
Epoch 15: 100%|██████████| 54/54 [00:00<00:00, 103.97batch/s, mse=3.31e+10]
Epoch 16: 100%|██████████| 54/54 [00:00<00:00, 99.31batch/s, mse=3.38e+10]
Epoch 17: 100%|██████████| 54/54 [00:00<00:00, 100.85batch/s, mse=3.42e+10]
Epoch 18: 100%|██████████| 54/54 [00:00<00:00, 100.15batch/s, mse=3.33e+10]
Epoch 19: 100%|██████████| 54/54 [00:00<00:00, 93.26batch/s, mse=3.34e+10]
Epoch 20: 100%|██████████| 54/54 [00:00<00:00, 102.04batch/s, mse=3.27e+10]
Epoch 21: 100%|██████████| 54/54 [00:00<00:00, 112.78batch/s, mse=3.12e+10]
Epoch 22: 100%|██████████| 54/54 [00:00<00:00, 100.87batch/s, mse=3.14e+10]
Epoch 23: 100%|██████████| 54/54 [00:00<00:00, 104.92batch/s, mse=3.13e+10]
Epoch 24: 100%|██████████| 54/54 [00:00<00:00, 118.33batch/s, mse=3.19e+10]
Epoch 25: 100%|██████████| 54/54 [00:00<00:00, 112.02batch/s, mse=2.99e+10]
Epoch 26: 100%|██████████| 54/54 [00:00<00:00, 121.88batch/s, mse=2.96e+10]
Epoch 27: 100%|██████████| 54/54 [00:00<00:00, 117.43batch/s, mse=2.74e+10]
Epoch 28: 100%|██████████| 54/54 [00:00<00:00, 112.24batch/s, mse=2.47e+10]
Epoch 29: 100%|██████████| 54/54 [00:00<00:00, 119.20batch/s, mse=2.35e+10]
Epoch 30: 100%|██████████| 54/54 [00:01<00:00, 49.94batch/s, mse=2.23e+10]
Epoch 31: 100%|██████████| 54/54 [00:01<00:00, 37.43batch/s, mse=2.27e+10]
Epoch 32: 100%|██████████| 54/54 [00:00<00:00, 75.14batch/s, mse=2.29e+10]
Epoch 33: 100%|██████████| 54/54 [00:00<00:00, 81.03batch/s, mse=2.24e+10]
Epoch 34: 100%|██████████| 54/54 [00:00<00:00, 90.45batch/s, mse=2.25e+10]
Epoch 35: 100%|██████████| 54/54 [00:00<00:00, 89.38batch/s, mse=2.13e+10]
Epoch 36: 100%|██████████| 54/54 [00:00<00:00, 89.10batch/s, mse=2.08e+10]
Epoch 37: 100%|██████████| 54/54 [00:00<00:00, 73.59batch/s, mse=1.98e+10]
Epoch 38: 100%|██████████| 54/54 [00:00<00:00, 84.11batch/s, mse=1.86e+10]
Epoch 39: 100%|██████████| 54/54 [00:00<00:00, 99.79batch/s, mse=1.98e+10]
Epoch 40: 100%|██████████| 54/54 [00:00<00:00, 89.66batch/s, mse=1.82e+10]
Epoch 41: 100%|██████████| 54/54 [00:00<00:00, 100.83batch/s, mse=1.79e+10]
Epoch 42: 100%|██████████| 54/54 [00:00<00:00, 97.73batch/s, mse=1.8e+10]
Epoch 43: 100%|██████████| 54/54 [00:00<00:00, 101.18batch/s, mse=1.81e+10]
Epoch 44: 100%|██████████| 54/54 [00:00<00:00, 98.24batch/s, mse=1.73e+10]
Epoch 45: 100%|██████████| 54/54 [00:00<00:00, 58.33batch/s, mse=1.78e+10]
Epoch 46: 100%|██████████| 54/54 [00:00<00:00, 97.98batch/s, mse=1.77e+10]
Epoch 47: 100%|██████████| 54/54 [00:00<00:00, 91.84batch/s, mse=1.62e+10]
Epoch 48: 100%|██████████| 54/54 [00:00<00:00, 93.42batch/s, mse=1.73e+10]
Epoch 49: 100%|██████████| 54/54 [00:00<00:00, 102.81batch/s, mse=1.63e+10]

Validation MSE: 24129601536.00
Validation RMSE: 155337.06
Running with LR:  0.1

Epoch 0: 100%|██████████| 54/54 [00:00<00:00, 94.62batch/s, mse=8.96e+10]
Epoch 1: 100%|██████████| 54/54 [00:00<00:00, 98.44batch/s, mse=4.25e+10]
Epoch 2: 100%|██████████| 54/54 [00:00<00:00, 96.05batch/s, mse=2.79e+10]
Epoch 3: 100%|██████████| 54/54 [00:00<00:00, 101.44batch/s, mse=2.57e+10]
Epoch 4: 100%|██████████| 54/54 [00:00<00:00, 98.05batch/s, mse=2.52e+10]
Epoch 5: 100%|██████████| 54/54 [00:00<00:00, 102.39batch/s, mse=2.51e+10]
Epoch 6: 100%|██████████| 54/54 [00:00<00:00, 84.25batch/s, mse=2.5e+10]
Epoch 7: 100%|██████████| 54/54 [00:00<00:00, 73.53batch/s, mse=2.49e+10]
Epoch 8: 100%|██████████| 54/54 [00:00<00:00, 99.61batch/s, mse=2.48e+10]
Epoch 9: 100%|██████████| 54/54 [00:00<00:00, 105.25batch/s, mse=2.48e+10]
Epoch 10: 100%|██████████| 54/54 [00:00<00:00, 108.63batch/s, mse=2.47e+10]
Epoch 11: 100%|██████████| 54/54 [00:00<00:00, 101.25batch/s, mse=2.46e+10]
Epoch 12: 100%|██████████| 54/54 [00:00<00:00, 99.21batch/s, mse=2.44e+10]
Epoch 13: 100%|██████████| 54/54 [00:00<00:00, 97.20batch/s, mse=2.44e+10]
Epoch 14: 100%|██████████| 54/54 [00:00<00:00, 80.21batch/s, mse=2.42e+10]
Epoch 15: 100%|██████████| 54/54 [00:00<00:00, 89.61batch/s, mse=2.41e+10]
Epoch 16: 100%|██████████| 54/54 [00:00<00:00, 100.03batch/s, mse=2.41e+10]
Epoch 17: 100%|██████████| 54/54 [00:00<00:00, 99.66batch/s, mse=2.39e+10]
Epoch 18: 100%|██████████| 54/54 [00:00<00:00, 100.06batch/s, mse=2.39e+10]
Epoch 19: 100%|██████████| 54/54 [00:00<00:00, 106.11batch/s, mse=2.37e+10]
Epoch 20: 100%|██████████| 54/54 [00:00<00:00, 103.91batch/s, mse=2.37e+10]
Epoch 21: 100%|██████████| 54/54 [00:00<00:00, 74.53batch/s, mse=2.36e+10]
```

```
Epoch 22: 100%|██████████| 54/54 [00:00<00:00, 96.53batch/s, mse=2.35e+10]
Epoch 23: 100%|██████████| 54/54 [00:00<00:00, 90.64batch/s, mse=2.33e+10]
Epoch 24: 100%|██████████| 54/54 [00:00<00:00, 101.71batch/s, mse=2.32e+10]
Epoch 25: 100%|██████████| 54/54 [00:00<00:00, 106.62batch/s, mse=2.31e+10]
Epoch 26: 100%|██████████| 54/54 [00:00<00:00, 98.69batch/s, mse=2.29e+10]
Epoch 27: 100%|██████████| 54/54 [00:00<00:00, 102.89batch/s, mse=2.27e+10]
Epoch 28: 100%|██████████| 54/54 [00:00<00:00, 99.70batch/s, mse=2.26e+10]
Epoch 29: 100%|██████████| 54/54 [00:00<00:00, 96.12batch/s, mse=2.24e+10]
Epoch 30: 100%|██████████| 54/54 [00:00<00:00, 97.11batch/s, mse=2.23e+10]
Epoch 31: 100%|██████████| 54/54 [00:00<00:00, 95.34batch/s, mse=2.22e+10]
Epoch 32: 100%|██████████| 54/54 [00:00<00:00, 103.11batch/s, mse=2.21e+10]
Epoch 33: 100%|██████████| 54/54 [00:00<00:00, 100.58batch/s, mse=2.2e+10]
Epoch 34: 100%|██████████| 54/54 [00:00<00:00, 107.59batch/s, mse=2.19e+10]
Epoch 35: 100%|██████████| 54/54 [00:00<00:00, 119.99batch/s, mse=2.19e+10]
Epoch 36: 100%|██████████| 54/54 [00:00<00:00, 97.11batch/s, mse=2.18e+10]
Epoch 37: 100%|██████████| 54/54 [00:00<00:00, 75.86batch/s, mse=2.18e+10]
Epoch 38: 100%|██████████| 54/54 [00:00<00:00, 67.86batch/s, mse=2.17e+10]
Epoch 39: 100%|██████████| 54/54 [00:00<00:00, 62.06batch/s, mse=2.16e+10]
Epoch 40: 100%|██████████| 54/54 [00:00<00:00, 61.98batch/s, mse=2.16e+10]
Epoch 41: 100%|██████████| 54/54 [00:00<00:00, 65.29batch/s, mse=2.15e+10]
Epoch 42: 100%|██████████| 54/54 [00:00<00:00, 111.53batch/s, mse=2.15e+10]
Epoch 43: 100%|██████████| 54/54 [00:00<00:00, 113.26batch/s, mse=2.15e+10]
Epoch 44: 100%|██████████| 54/54 [00:00<00:00, 99.29batch/s, mse=2.15e+10]
Epoch 45: 100%|██████████| 54/54 [00:00<00:00, 96.64batch/s, mse=2.15e+10]
Epoch 46: 100%|██████████| 54/54 [00:00<00:00, 98.02batch/s, mse=2.14e+10]
Epoch 47: 100%|██████████| 54/54 [00:00<00:00, 85.07batch/s, mse=2.13e+10]
Epoch 48: 100%|██████████| 54/54 [00:00<00:00, 90.93batch/s, mse=2.13e+10]
Epoch 49: 100%|██████████| 54/54 [00:00<00:00, 85.19batch/s, mse=2.12e+10]
```

```
Validation MSE: 35533664256.00
Validation RMSE: 188503.75
Running with LR:  0.01
```

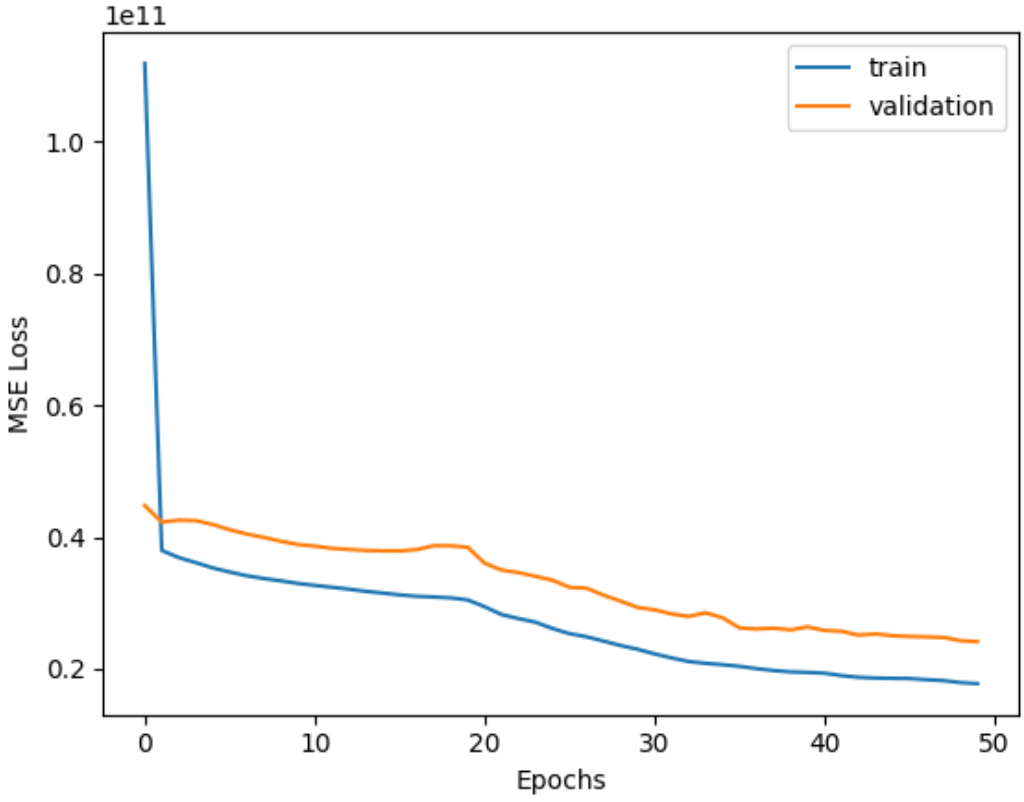
```
Epoch 0: 100%|██████████| 54/54 [00:00<00:00, 67.36batch/s, mse=4.16e+11]
Epoch 1: 100%|██████████| 54/54 [00:00<00:00, 73.33batch/s, mse=4.02e+11]
Epoch 2: 100%|██████████| 54/54 [00:00<00:00, 97.83batch/s, mse=3.52e+11]
Epoch 3: 100%|██████████| 54/54 [00:00<00:00, 101.30batch/s, mse=2.59e+11]
Epoch 4: 100%|██████████| 54/54 [00:00<00:00, 102.18batch/s, mse=1.58e+11]
Epoch 5: 100%|██████████| 54/54 [00:00<00:00, 103.02batch/s, mse=9.64e+10]
Epoch 6: 100%|██████████| 54/54 [00:00<00:00, 110.07batch/s, mse=7.61e+10]
Epoch 7: 100%|██████████| 54/54 [00:00<00:00, 109.29batch/s, mse=6.98e+10]
Epoch 8: 100%|██████████| 54/54 [00:00<00:00, 104.81batch/s, mse=6.63e+10]
Epoch 9: 100%|██████████| 54/54 [00:00<00:00, 106.81batch/s, mse=6.37e+10]
Epoch 10: 100%|██████████| 54/54 [00:00<00:00, 106.28batch/s, mse=6.12e+10]
Epoch 11: 100%|██████████| 54/54 [00:00<00:00, 103.87batch/s, mse=5.88e+10]
Epoch 12: 100%|██████████| 54/54 [00:00<00:00, 82.71batch/s, mse=5.62e+10]
Epoch 13: 100%|██████████| 54/54 [00:00<00:00, 106.71batch/s, mse=5.37e+10]
Epoch 14: 100%|██████████| 54/54 [00:00<00:00, 102.76batch/s, mse=5.11e+10]
Epoch 15: 100%|██████████| 54/54 [00:00<00:00, 93.84batch/s, mse=4.85e+10]
Epoch 16: 100%|██████████| 54/54 [00:00<00:00, 98.01batch/s, mse=4.6e+10]
Epoch 17: 100%|██████████| 54/54 [00:00<00:00, 95.34batch/s, mse=4.35e+10]
Epoch 18: 100%|██████████| 54/54 [00:00<00:00, 103.59batch/s, mse=4.1e+10]
Epoch 19: 100%|██████████| 54/54 [00:00<00:00, 99.90batch/s, mse=3.88e+10]
Epoch 20: 100%|██████████| 54/54 [00:00<00:00, 98.33batch/s, mse=3.67e+10]
Epoch 21: 100%|██████████| 54/54 [00:00<00:00, 109.33batch/s, mse=3.49e+10]
Epoch 22: 100%|██████████| 54/54 [00:00<00:00, 91.79batch/s, mse=3.32e+10]
Epoch 23: 100%|██████████| 54/54 [00:00<00:00, 93.44batch/s, mse=3.19e+10]
Epoch 24: 100%|██████████| 54/54 [00:00<00:00, 103.83batch/s, mse=3.09e+10]
Epoch 25: 100%|██████████| 54/54 [00:00<00:00, 101.36batch/s, mse=3.01e+10]
Epoch 26: 100%|██████████| 54/54 [00:00<00:00, 91.52batch/s, mse=2.94e+10]
Epoch 27: 100%|██████████| 54/54 [00:00<00:00, 104.00batch/s, mse=2.89e+10]
Epoch 28: 100%|██████████| 54/54 [00:00<00:00, 93.71batch/s, mse=2.84e+10]
Epoch 29: 100%|██████████| 54/54 [00:00<00:00, 88.89batch/s, mse=2.81e+10]
Epoch 30: 100%|██████████| 54/54 [00:00<00:00, 89.18batch/s, mse=2.78e+10]
Epoch 31: 100%|██████████| 54/54 [00:00<00:00, 107.30batch/s, mse=2.75e+10]
Epoch 32: 100%|██████████| 54/54 [00:00<00:00, 103.43batch/s, mse=2.72e+10]
Epoch 33: 100%|██████████| 54/54 [00:00<00:00, 80.96batch/s, mse=2.7e+10]
Epoch 34: 100%|██████████| 54/54 [00:00<00:00, 81.06batch/s, mse=2.68e+10]
Epoch 35: 100%|██████████| 54/54 [00:00<00:00, 78.34batch/s, mse=2.67e+10]
Epoch 36: 100%|██████████| 54/54 [00:00<00:00, 78.30batch/s, mse=2.65e+10]
Epoch 37: 100%|██████████| 54/54 [00:00<00:00, 88.22batch/s, mse=2.64e+10]
Epoch 38: 100%|██████████| 54/54 [00:00<00:00, 72.56batch/s, mse=2.63e+10]
Epoch 39: 100%|██████████| 54/54 [00:00<00:00, 80.53batch/s, mse=2.62e+10]
```

```
Epoch 40: 100%|██████████| 54/54 [00:00<00:00, 111.21batch/s, mse=2.61e+10]
Epoch 41: 100%|██████████| 54/54 [00:00<00:00, 107.70batch/s, mse=2.6e+10]
Epoch 42: 100%|██████████| 54/54 [00:00<00:00, 97.87batch/s, mse=2.59e+10]
Epoch 43: 100%|██████████| 54/54 [00:00<00:00, 96.83batch/s, mse=2.59e+10]
Epoch 44: 100%|██████████| 54/54 [00:00<00:00, 107.42batch/s, mse=2.58e+10]
Epoch 45: 100%|██████████| 54/54 [00:00<00:00, 109.62batch/s, mse=2.57e+10]
Epoch 46: 100%|██████████| 54/54 [00:00<00:00, 109.32batch/s, mse=2.57e+10]
Epoch 47: 100%|██████████| 54/54 [00:00<00:00, 108.81batch/s, mse=2.56e+10]
Epoch 48: 100%|██████████| 54/54 [00:00<00:00, 109.59batch/s, mse=2.56e+10]
Epoch 49: 100%|██████████| 54/54 [00:00<00:00, 101.37batch/s, mse=2.55e+10]

Validation MSE: 41104048128.00
Validation RMSE: 202741.33
Best LR:  1
Best Validation MSE:  24129602000.0
```

In [10]:

```
# restore model using the .load_state_dict() function
##  START CODE  ## (1 line of code)
model.load_state_dict(best_weights)

##  END CODE  ##

# calculate test performance using the eval function we defined above (MSE and R2)
##  START CODE  ##
test_mse, test_r2 = eval(model, X_test, y_test)
##  END CODE  ##

print("Test MSE: %.2f" % test_mse)
print("Test R2: %.2f" % test_r2)
plt.plot(best_train_history, label='train')
plt.plot(best_valid_history, label='validation')
plt.legend(loc="upper right")
plt.xlabel('Epochs')
plt.ylabel('MSE Loss')
plt.show()
```

```
Test MSE: 24662196224.00
Test R2: 0.82
```



(5 pts) Now we will analyze the performance of a smaller model on the same dataset. Define a smaller model with just 1 hidden layer of size 32 and the ReLU activation function. Use the same loss function and optimizer.

Sweep learning rates in [1, 0.1, 0.01, 0.001] to find the best learning rate. Finally, for the best model, evaluate the performance on the test set and plot training curves(loss vs epochs).

In [11]:

```
small_model_best_valid_mse = np.inf
small_model_best_weights = None
small_model_best_train_history = None
small_model_best_valid_history = None
best_lr = None
for lr in [1, 0.1, 0.01, 0.001]:
    print("Running with LR: ", lr)
    # Define the model using nn.Sequential
    ##  START CODE  ##
    small_model = nn.Sequential(
        nn.Linear(X_train.shape[1], 32),
        nn.ReLU(),
        nn.Linear(32, 1)
    )
    ##  END CODE  ##

    # Define loss function and optimizer
    ##  START CODE  ##
    loss_fn = nn.MSELoss()
    optimizer = optim.Adam(small_model.parameters(), lr=lr)
    ##  END CODE  ##

    n_epochs = 50   # number of epochs to run
    batch_size = 256  # size of each batch

    # train the model
    weights, train_history, valid_history, valid_mse \
    = train(small_model, X_train, y_train, X_valid, y_valid, n_epochs, batch_size, optim
izer, loss_fn)

    # store values for best LR
    if valid_mse < small_model_best_valid_mse:
        #TODO
        small_model_best_valid_mse = valid_mse
        small_model_best_weights = weights
        small_model_best_train_history = train_history
        small_model_best_valid_history = valid_history
        best_lr = lr

print("Best LR: ", best_lr)
print("Best Validation MSE: ", small_model_best_valid_mse)
```

```
Running with LR:  1
Epoch 0: 100%|██████████| 54/54 [00:00<00:00, 108.61batch/s, mse=1.21e+11]
Epoch 1: 100%|██████████| 54/54 [00:00<00:00, 109.93batch/s, mse=5.44e+10]
Epoch 2: 100%|██████████| 54/54 [00:00<00:00, 103.34batch/s, mse=3.92e+10]
Epoch 3: 100%|██████████| 54/54 [00:00<00:00, 112.84batch/s, mse=3.24e+10]
Epoch 4: 100%|██████████| 54/54 [00:00<00:00, 115.34batch/s, mse=2.89e+10]
Epoch 5: 100%|██████████| 54/54 [00:00<00:00, 108.09batch/s, mse=2.73e+10]
Epoch 6: 100%|██████████| 54/54 [00:00<00:00, 120.61batch/s, mse=2.65e+10]
Epoch 7: 100%|██████████| 54/54 [00:00<00:00, 114.79batch/s, mse=2.6e+10]
Epoch 8: 100%|██████████| 54/54 [00:00<00:00, 111.35batch/s, mse=2.56e+10]
Epoch 9: 100%|██████████| 54/54 [00:00<00:00, 103.56batch/s, mse=2.53e+10]
Epoch 10: 100%|██████████| 54/54 [00:00<00:00, 109.76batch/s, mse=2.51e+10]
Epoch 11: 100%|██████████| 54/54 [00:00<00:00, 114.89batch/s, mse=2.49e+10]
Epoch 12: 100%|██████████| 54/54 [00:00<00:00, 114.24batch/s, mse=2.48e+10]
Epoch 13: 100%|██████████| 54/54 [00:00<00:00, 116.01batch/s, mse=2.46e+10]
Epoch 14: 100%|██████████| 54/54 [00:00<00:00, 114.76batch/s, mse=2.44e+10]
Epoch 15: 100%|██████████| 54/54 [00:00<00:00, 114.99batch/s, mse=2.42e+10]
Epoch 16: 100%|██████████| 54/54 [00:00<00:00, 116.69batch/s, mse=2.41e+10]
Epoch 17: 100%|██████████| 54/54 [00:00<00:00, 117.09batch/s, mse=2.4e+10]
Epoch 18: 100%|██████████| 54/54 [00:00<00:00, 112.28batch/s, mse=2.38e+10]
Epoch 19: 100%|██████████| 54/54 [00:00<00:00, 119.19batch/s, mse=2.37e+10]
Epoch 20: 100%|██████████| 54/54 [00:00<00:00, 86.53batch/s, mse=2.36e+10]
Epoch 21: 100%|██████████| 54/54 [00:00<00:00, 85.54batch/s, mse=2.35e+10]
Epoch 22: 100%|██████████| 54/54 [00:00<00:00, 110.83batch/s, mse=2.34e+10]
```

```
Epoch 23: 100%|██████████| 54/54 [00:00<00:00, 117.30batch/s, mse=2.33e+10]
Epoch 24: 100%|██████████| 54/54 [00:00<00:00, 105.94batch/s, mse=2.32e+10]
Epoch 25: 100%|██████████| 54/54 [00:00<00:00, 111.79batch/s, mse=2.32e+10]
Epoch 26: 100%|██████████| 54/54 [00:00<00:00, 118.73batch/s, mse=2.32e+10]
Epoch 27: 100%|██████████| 54/54 [00:00<00:00, 107.88batch/s, mse=2.31e+10]
Epoch 28: 100%|██████████| 54/54 [00:00<00:00, 111.48batch/s, mse=2.3e+10]
Epoch 29: 100%|██████████| 54/54 [00:00<00:00, 113.83batch/s, mse=2.3e+10]
Epoch 30: 100%|██████████| 54/54 [00:00<00:00, 105.44batch/s, mse=2.29e+10]
Epoch 31: 100%|██████████| 54/54 [00:00<00:00, 113.48batch/s, mse=2.29e+10]
Epoch 32: 100%|██████████| 54/54 [00:00<00:00, 115.08batch/s, mse=2.28e+10]
Epoch 33: 100%|██████████| 54/54 [00:00<00:00, 113.37batch/s, mse=2.27e+10]
Epoch 34: 100%|██████████| 54/54 [00:00<00:00, 111.94batch/s, mse=2.26e+10]
Epoch 35: 100%|██████████| 54/54 [00:00<00:00, 112.27batch/s, mse=2.26e+10]
Epoch 36: 100%|██████████| 54/54 [00:00<00:00, 109.40batch/s, mse=2.25e+10]
Epoch 37: 100%|██████████| 54/54 [00:00<00:00, 93.18batch/s, mse=2.24e+10]
Epoch 38: 100%|██████████| 54/54 [00:00<00:00, 108.06batch/s, mse=2.22e+10]
Epoch 39: 100%|██████████| 54/54 [00:00<00:00, 103.33batch/s, mse=2.21e+10]
Epoch 40: 100%|██████████| 54/54 [00:00<00:00, 94.86batch/s, mse=2.2e+10]
Epoch 41: 100%|██████████| 54/54 [00:00<00:00, 97.44batch/s, mse=2.19e+10]
Epoch 42: 100%|██████████| 54/54 [00:00<00:00, 109.86batch/s, mse=2.18e+10]
Epoch 43: 100%|██████████| 54/54 [00:00<00:00, 106.79batch/s, mse=2.17e+10]
Epoch 44: 100%|██████████| 54/54 [00:00<00:00, 89.25batch/s, mse=2.16e+10]
Epoch 45: 100%|██████████| 54/54 [00:00<00:00, 105.29batch/s, mse=2.15e+10]
Epoch 46: 100%|██████████| 54/54 [00:00<00:00, 101.92batch/s, mse=2.14e+10]
Epoch 47: 100%|██████████| 54/54 [00:00<00:00, 105.96batch/s, mse=2.13e+10]
Epoch 48: 100%|██████████| 54/54 [00:00<00:00, 114.79batch/s, mse=2.12e+10]
Epoch 49: 100%|██████████| 54/54 [00:00<00:00, 106.96batch/s, mse=2.12e+10]

Validation MSE: 36408078336.00
Validation RMSE: 190809.02
Running with LR:  0.1

Epoch 0: 100%|██████████| 54/54 [00:00<00:00, 120.57batch/s, mse=4.11e+11]
Epoch 1: 100%|██████████| 54/54 [00:00<00:00, 109.39batch/s, mse=3.88e+11]
Epoch 2: 100%|██████████| 54/54 [00:00<00:00, 126.81batch/s, mse=3.46e+11]
Epoch 3: 100%|██████████| 54/54 [00:00<00:00, 110.78batch/s, mse=2.94e+11]
Epoch 4: 100%|██████████| 54/54 [00:00<00:00, 108.16batch/s, mse=2.39e+11]
Epoch 5: 100%|██████████| 54/54 [00:00<00:00, 102.18batch/s, mse=1.89e+11]
Epoch 6: 100%|██████████| 54/54 [00:00<00:00, 107.09batch/s, mse=1.49e+11]
Epoch 7: 100%|██████████| 54/54 [00:00<00:00, 83.32batch/s, mse=1.18e+11]
Epoch 8: 100%|██████████| 54/54 [00:00<00:00, 111.99batch/s, mse=9.61e+10]
Epoch 9: 100%|██████████| 54/54 [00:00<00:00, 101.43batch/s, mse=8.04e+10]
Epoch 10: 100%|██████████| 54/54 [00:00<00:00, 108.06batch/s, mse=7.05e+10]
Epoch 11: 100%|██████████| 54/54 [00:00<00:00, 91.08batch/s, mse=6.47e+10]
Epoch 12: 100%|██████████| 54/54 [00:00<00:00, 66.23batch/s, mse=6.09e+10]
Epoch 13: 100%|██████████| 54/54 [00:00<00:00, 118.89batch/s, mse=5.8e+10]
Epoch 14: 100%|██████████| 54/54 [00:00<00:00, 113.28batch/s, mse=5.55e+10]
Epoch 15: 100%|██████████| 54/54 [00:00<00:00, 115.21batch/s, mse=5.34e+10]
Epoch 16: 100%|██████████| 54/54 [00:00<00:00, 105.87batch/s, mse=5.14e+10]
Epoch 17: 100%|██████████| 54/54 [00:00<00:00, 111.00batch/s, mse=4.96e+10]
Epoch 18: 100%|██████████| 54/54 [00:00<00:00, 110.68batch/s, mse=4.79e+10]
Epoch 19: 100%|██████████| 54/54 [00:00<00:00, 82.88batch/s, mse=4.63e+10]
Epoch 20: 100%|██████████| 54/54 [00:00<00:00, 82.63batch/s, mse=4.48e+10]
Epoch 21: 100%|██████████| 54/54 [00:00<00:00, 111.10batch/s, mse=4.33e+10]
Epoch 22: 100%|██████████| 54/54 [00:00<00:00, 106.51batch/s, mse=4.18e+10]
Epoch 23: 100%|██████████| 54/54 [00:00<00:00, 119.13batch/s, mse=4.04e+10]
Epoch 24: 100%|██████████| 54/54 [00:00<00:00, 116.85batch/s, mse=3.9e+10]
Epoch 25: 100%|██████████| 54/54 [00:00<00:00, 112.95batch/s, mse=3.76e+10]
Epoch 26: 100%|██████████| 54/54 [00:00<00:00, 114.97batch/s, mse=3.64e+10]
Epoch 27: 100%|██████████| 54/54 [00:00<00:00, 116.23batch/s, mse=3.53e+10]
Epoch 28: 100%|██████████| 54/54 [00:00<00:00, 111.61batch/s, mse=3.44e+10]
Epoch 29: 100%|██████████| 54/54 [00:00<00:00, 111.41batch/s, mse=3.36e+10]
Epoch 30: 100%|██████████| 54/54 [00:00<00:00, 110.12batch/s, mse=3.29e+10]
Epoch 31: 100%|██████████| 54/54 [00:00<00:00, 112.47batch/s, mse=3.23e+10]
Epoch 32: 100%|██████████| 54/54 [00:00<00:00, 107.34batch/s, mse=3.17e+10]
Epoch 33: 100%|██████████| 54/54 [00:00<00:00, 114.85batch/s, mse=3.12e+10]
Epoch 34: 100%|██████████| 54/54 [00:00<00:00, 117.47batch/s, mse=3.08e+10]
Epoch 35: 100%|██████████| 54/54 [00:00<00:00, 108.92batch/s, mse=3.04e+10]
Epoch 36: 100%|██████████| 54/54 [00:00<00:00, 99.41batch/s, mse=3.01e+10]
Epoch 37: 100%|██████████| 54/54 [00:00<00:00, 108.49batch/s, mse=2.97e+10]
Epoch 38: 100%|██████████| 54/54 [00:00<00:00, 115.16batch/s, mse=2.94e+10]
Epoch 39: 100%|██████████| 54/54 [00:00<00:00, 100.19batch/s, mse=2.91e+10]
Epoch 40: 100%|██████████| 54/54 [00:00<00:00, 96.31batch/s, mse=2.87e+10]
```

```
Epoch 41: 100%|██████████| 54/54 [00:00<00:00, 111.37batch/s, mse=2.84e+10]
Epoch 42: 100%|██████████| 54/54 [00:00<00:00, 93.16batch/s, mse=2.82e+10]
Epoch 43: 100%|██████████| 54/54 [00:00<00:00, 116.91batch/s, mse=2.8e+10]
Epoch 44: 100%|██████████| 54/54 [00:00<00:00, 99.98batch/s, mse=2.79e+10]
Epoch 45: 100%|██████████| 54/54 [00:00<00:00, 82.42batch/s, mse=2.78e+10]
Epoch 46: 100%|██████████| 54/54 [00:00<00:00, 91.97batch/s, mse=2.76e+10]
Epoch 47: 100%|██████████| 54/54 [00:00<00:00, 96.57batch/s, mse=2.75e+10]
Epoch 48: 100%|██████████| 54/54 [00:00<00:00, 88.89batch/s, mse=2.74e+10]
Epoch 49: 100%|██████████| 54/54 [00:00<00:00, 84.72batch/s, mse=2.73e+10]

Validation MSE: 42700611584.00
Validation RMSE: 206641.27
Running with LR:  0.01

Epoch 0: 100%|██████████| 54/54 [00:00<00:00, 56.11batch/s, mse=4.17e+11]
Epoch 1: 100%|██████████| 54/54 [00:00<00:00, 116.26batch/s, mse=4.17e+11]
Epoch 2: 100%|██████████| 54/54 [00:00<00:00, 117.30batch/s, mse=4.16e+11]
Epoch 3: 100%|██████████| 54/54 [00:00<00:00, 113.66batch/s, mse=4.15e+11]
Epoch 4: 100%|██████████| 54/54 [00:00<00:00, 110.95batch/s, mse=4.14e+11]
Epoch 5: 100%|██████████| 54/54 [00:00<00:00, 115.90batch/s, mse=4.13e+11]
Epoch 6: 100%|██████████| 54/54 [00:00<00:00, 113.39batch/s, mse=4.12e+11]
Epoch 7: 100%|██████████| 54/54 [00:00<00:00, 84.42batch/s, mse=4.1e+11]
Epoch 8: 100%|██████████| 54/54 [00:00<00:00, 115.68batch/s, mse=4.08e+11]
Epoch 9: 100%|██████████| 54/54 [00:00<00:00, 114.02batch/s, mse=4.06e+11]
Epoch 10: 100%|██████████| 54/54 [00:00<00:00, 112.22batch/s, mse=4.04e+11]
Epoch 11: 100%|██████████| 54/54 [00:00<00:00, 110.83batch/s, mse=4.02e+11]
Epoch 12: 100%|██████████| 54/54 [00:00<00:00, 109.57batch/s, mse=3.99e+11]
Epoch 13: 100%|██████████| 54/54 [00:00<00:00, 124.38batch/s, mse=3.96e+11]
Epoch 14: 100%|██████████| 54/54 [00:00<00:00, 121.85batch/s, mse=3.93e+11]
Epoch 15: 100%|██████████| 54/54 [00:00<00:00, 117.79batch/s, mse=3.9e+11]
Epoch 16: 100%|██████████| 54/54 [00:00<00:00, 121.98batch/s, mse=3.87e+11]
Epoch 17: 100%|██████████| 54/54 [00:00<00:00, 109.37batch/s, mse=3.83e+11]
Epoch 18: 100%|██████████| 54/54 [00:00<00:00, 103.79batch/s, mse=3.8e+11]
Epoch 19: 100%|██████████| 54/54 [00:00<00:00, 121.38batch/s, mse=3.76e+11]
Epoch 20: 100%|██████████| 54/54 [00:00<00:00, 96.62batch/s, mse=3.73e+11]
Epoch 21: 100%|██████████| 54/54 [00:00<00:00, 116.51batch/s, mse=3.69e+11]
Epoch 22: 100%|██████████| 54/54 [00:00<00:00, 99.49batch/s, mse=3.65e+11]
Epoch 23: 100%|██████████| 54/54 [00:00<00:00, 97.02batch/s, mse=3.61e+11]
Epoch 24: 100%|██████████| 54/54 [00:00<00:00, 108.32batch/s, mse=3.56e+11]
Epoch 25: 100%|██████████| 54/54 [00:00<00:00, 111.73batch/s, mse=3.52e+11]
Epoch 26: 100%|██████████| 54/54 [00:00<00:00, 102.46batch/s, mse=3.48e+11]
Epoch 27: 100%|██████████| 54/54 [00:00<00:00, 111.32batch/s, mse=3.43e+11]
Epoch 28: 100%|██████████| 54/54 [00:00<00:00, 108.52batch/s, mse=3.39e+11]
Epoch 29: 100%|██████████| 54/54 [00:00<00:00, 97.14batch/s, mse=3.34e+11]
Epoch 30: 100%|██████████| 54/54 [00:00<00:00, 105.29batch/s, mse=3.3e+11]
Epoch 31: 100%|██████████| 54/54 [00:00<00:00, 101.05batch/s, mse=3.25e+11]
Epoch 32: 100%|██████████| 54/54 [00:00<00:00, 109.18batch/s, mse=3.21e+11]
Epoch 33: 100%|██████████| 54/54 [00:00<00:00, 109.43batch/s, mse=3.16e+11]
Epoch 34: 100%|██████████| 54/54 [00:00<00:00, 109.26batch/s, mse=3.11e+11]
Epoch 35: 100%|██████████| 54/54 [00:00<00:00, 114.80batch/s, mse=3.06e+11]
Epoch 36: 100%|██████████| 54/54 [00:00<00:00, 114.54batch/s, mse=3.02e+11]
Epoch 37: 100%|██████████| 54/54 [00:00<00:00, 122.06batch/s, mse=2.97e+11]
Epoch 38: 100%|██████████| 54/54 [00:00<00:00, 117.58batch/s, mse=2.92e+11]
Epoch 39: 100%|██████████| 54/54 [00:00<00:00, 121.64batch/s, mse=2.87e+11]
Epoch 40: 100%|██████████| 54/54 [00:00<00:00, 119.44batch/s, mse=2.82e+11]
Epoch 41: 100%|██████████| 54/54 [00:00<00:00, 120.28batch/s, mse=2.77e+11]
Epoch 42: 100%|██████████| 54/54 [00:00<00:00, 113.27batch/s, mse=2.73e+11]
Epoch 43: 100%|██████████| 54/54 [00:00<00:00, 110.80batch/s, mse=2.68e+11]
Epoch 44: 100%|██████████| 54/54 [00:00<00:00, 113.19batch/s, mse=2.63e+11]
Epoch 45: 100%|██████████| 54/54 [00:00<00:00, 105.95batch/s, mse=2.58e+11]
Epoch 46: 100%|██████████| 54/54 [00:00<00:00, 107.27batch/s, mse=2.54e+11]
Epoch 47: 100%|██████████| 54/54 [00:00<00:00, 107.61batch/s, mse=2.49e+11]
Epoch 48: 100%|██████████| 54/54 [00:00<00:00, 107.75batch/s, mse=2.44e+11]
Epoch 49: 100%|██████████| 54/54 [00:00<00:00, 81.97batch/s, mse=2.4e+11]

Validation MSE: 262300106752.00
Validation RMSE: 512152.44
Running with LR:  0.001

Epoch 0: 100%|██████████| 54/54 [00:00<00:00, 89.78batch/s, mse=4.17e+11]
Epoch 1: 100%|██████████| 54/54 [00:00<00:00, 92.84batch/s, mse=4.17e+11]
Epoch 2: 100%|██████████| 54/54 [00:00<00:00, 84.94batch/s, mse=4.17e+11]
```

```
Epoch 3: 100%|██████████| 54/54 [00:00<00:00, 58.91batch/s, mse=4.17e+11]
Epoch 4: 100%|██████████| 54/54 [00:00<00:00, 87.25batch/s, mse=4.17e+11]
Epoch 5: 100%|██████████| 54/54 [00:00<00:00, 112.54batch/s, mse=4.17e+11]
Epoch 6: 100%|██████████| 54/54 [00:00<00:00, 108.32batch/s, mse=4.17e+11]
Epoch 7: 100%|██████████| 54/54 [00:00<00:00, 117.89batch/s, mse=4.17e+11]
Epoch 8: 100%|██████████| 54/54 [00:00<00:00, 109.42batch/s, mse=4.17e+11]
Epoch 9: 100%|██████████| 54/54 [00:00<00:00, 95.45batch/s, mse=4.17e+11]
Epoch 10: 100%|██████████| 54/54 [00:00<00:00, 76.43batch/s, mse=4.17e+11]
Epoch 11: 100%|██████████| 54/54 [00:00<00:00, 119.68batch/s, mse=4.17e+11]
Epoch 12: 100%|██████████| 54/54 [00:00<00:00, 105.76batch/s, mse=4.17e+11]
Epoch 13: 100%|██████████| 54/54 [00:00<00:00, 108.84batch/s, mse=4.17e+11]
Epoch 14: 100%|██████████| 54/54 [00:00<00:00, 100.63batch/s, mse=4.17e+11]
Epoch 15: 100%|██████████| 54/54 [00:00<00:00, 90.80batch/s, mse=4.17e+11]
Epoch 16: 100%|██████████| 54/54 [00:00<00:00, 99.71batch/s, mse=4.17e+11]
Epoch 17: 100%|██████████| 54/54 [00:00<00:00, 114.14batch/s, mse=4.17e+11]
Epoch 18: 100%|██████████| 54/54 [00:00<00:00, 107.94batch/s, mse=4.17e+11]
Epoch 19: 100%|██████████| 54/54 [00:00<00:00, 99.95batch/s, mse=4.17e+11]
Epoch 20: 100%|██████████| 54/54 [00:00<00:00, 103.16batch/s, mse=4.16e+11]
Epoch 21: 100%|██████████| 54/54 [00:00<00:00, 108.96batch/s, mse=4.16e+11]
Epoch 22: 100%|██████████| 54/54 [00:00<00:00, 99.50batch/s, mse=4.16e+11]
Epoch 23: 100%|██████████| 54/54 [00:00<00:00, 111.76batch/s, mse=4.16e+11]
Epoch 24: 100%|██████████| 54/54 [00:00<00:00, 86.50batch/s, mse=4.16e+11]
Epoch 25: 100%|██████████| 54/54 [00:00<00:00, 101.15batch/s, mse=4.16e+11]
Epoch 26: 100%|██████████| 54/54 [00:00<00:00, 83.17batch/s, mse=4.16e+11]
Epoch 27: 100%|██████████| 54/54 [00:00<00:00, 91.27batch/s, mse=4.16e+11]
Epoch 28: 100%|██████████| 54/54 [00:00<00:00, 86.63batch/s, mse=4.16e+11]
Epoch 29: 100%|██████████| 54/54 [00:00<00:00, 78.97batch/s, mse=4.16e+11]
Epoch 30: 100%|██████████| 54/54 [00:00<00:00, 87.45batch/s, mse=4.16e+11]
Epoch 31: 100%|██████████| 54/54 [00:00<00:00, 72.74batch/s, mse=4.16e+11]
Epoch 32: 100%|██████████| 54/54 [00:00<00:00, 73.90batch/s, mse=4.16e+11]
Epoch 33: 100%|██████████| 54/54 [00:00<00:00, 91.07batch/s, mse=4.16e+11]
Epoch 34: 100%|██████████| 54/54 [00:01<00:00, 51.54batch/s, mse=4.16e+11]
Epoch 35: 100%|██████████| 54/54 [00:00<00:00, 100.10batch/s, mse=4.16e+11]
Epoch 36: 100%|██████████| 54/54 [00:00<00:00, 92.54batch/s, mse=4.15e+11]
Epoch 37: 100%|██████████| 54/54 [00:00<00:00, 117.92batch/s, mse=4.15e+11]
Epoch 38: 100%|██████████| 54/54 [00:00<00:00, 109.11batch/s, mse=4.15e+11]
Epoch 39: 100%|██████████| 54/54 [00:00<00:00, 112.86batch/s, mse=4.15e+11]
Epoch 40: 100%|██████████| 54/54 [00:00<00:00, 113.21batch/s, mse=4.15e+11]
Epoch 41: 100%|██████████| 54/54 [00:00<00:00, 116.49batch/s, mse=4.15e+11]
Epoch 42: 100%|██████████| 54/54 [00:00<00:00, 113.59batch/s, mse=4.15e+11]
Epoch 43: 100%|██████████| 54/54 [00:00<00:00, 112.52batch/s, mse=4.15e+11]
Epoch 44: 100%|██████████| 54/54 [00:00<00:00, 114.34batch/s, mse=4.15e+11]
Epoch 45: 100%|██████████| 54/54 [00:00<00:00, 100.75batch/s, mse=4.15e+11]
Epoch 46: 100%|██████████| 54/54 [00:00<00:00, 77.81batch/s, mse=4.15e+11]
Epoch 47: 100%|██████████| 54/54 [00:00<00:00, 91.13batch/s, mse=4.15e+11]
Epoch 48: 100%|██████████| 54/54 [00:00<00:00, 109.31batch/s, mse=4.14e+11]
Epoch 49: 100%|██████████| 54/54 [00:00<00:00, 114.75batch/s, mse=4.14e+11]
```

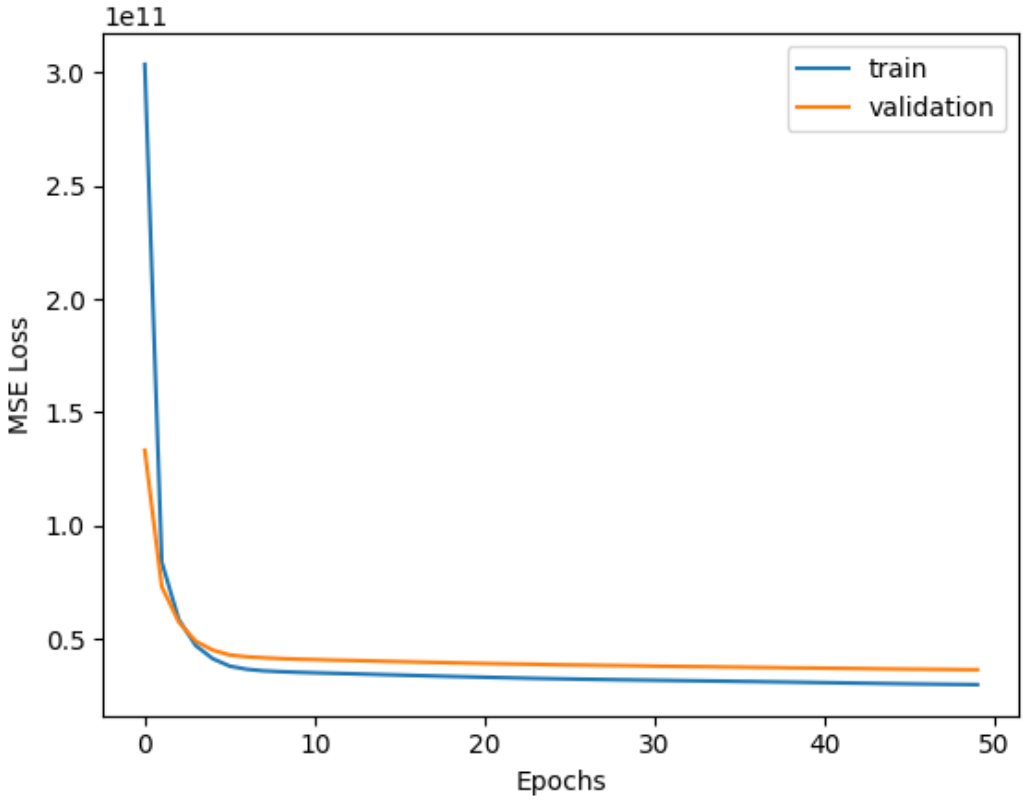
```
Validation MSE: 445796417536.00
Validation RMSE: 667679.88
Best LR:  1
Best Validation MSE:  36408080000.0
```

In [12]:

```
# restore model using the .load_state_dict() function
##  START CODE  ## (1 line of code)
small_model.load_state_dict(small_model_best_weights)

##  END CODE  ##

# calculate test performance using the eval function we defined above (MSE and R2)
##  START CODE  ##
test_mse, test_r2 = eval(small_model, X_test, y_test)
##  END CODE  ##

print("Test MSE: %.2f" % test_mse)
print("Test R2: %.2f" % test_r2)
plt.plot(small_model_best_train_history, label='train')
plt.plot(small_model_best_valid_history, label='validation')
plt.legend(loc="upper right")
plt.xlabel('Epochs')
```

```
plt.ylabel('MSE Loss')
plt.show()
```

```
Test MSE: 36540039168.00
Test R2: 0.74
```



How do the performances of the two MLPs and the linear regression model(from HW1) compare, and what do you think causes the difference?

Answer

Linear Regression performs poorly in scenarios where data is non-linear (like the concentric data from google playground). It may have lower variance, but its bias is high due to the assumption of linearity. MLPs outperform linear regression on non-linear datasets because they can capture complex relationships. An MLP with two hidden layers should outperform the one with a single hidden layer, as it has more capacity to model the complexity in the data. However, this comes at the cost of longer training times and a higher risk of overfitting.

(5 pts) Finally, we will train a bigger MLP model on the same dataset. Define a bigger model with 3 hidden layers with sizes (256, 32, 8) and the ReLU activation function. Use the same loss function and optimizer. For the best model on the validation data, evaluate the performance on the test set and plot training curves(loss vs epochs).

In [13]:

```
# Define the model using nn.Sequential
##  START CODE  ##
big_model = nn.Sequential(
        nn.Linear(X_train.shape[1], 256),
        nn.ReLU(),
        nn.Linear(256, 32),
        nn.ReLU(),
        nn.Linear(32, 8),
        nn.ReLU(),
        nn.Linear(8, 1)
    )
##  END CODE  ##

# Define loss function and optimizer
##  START CODE  ##
loss_fn = nn.MSELoss()
optimizer = optim.Adam(big_model.parameters(), lr=0.001) # we have choosen lr=0.01 as it
```

```
gave the best r^2 compared to 0.001 and .1 was overfitting showing a diversion
#lr of 0.01 will give us a better r sqaure but for the the sake of simplicity we have tak
en lr=0.001
##  END CODE  ##

n_epochs = 200   # number of epochs to run
batch_size = 256  # size of each batch

# train the model
big_model_best_weights, big_model_train_history, big_model_valid_history, big_model_best_
valid_mse \
= train(big_model, X_train, y_train, X_valid, y_valid, n_epochs, batch_size, optimizer,
loss_fn)
```

```
Epoch 0: 100%|██████████| 54/54 [00:00<00:00, 87.59batch/s, mse=4.17e+11]
Epoch 1: 100%|██████████| 54/54 [00:00<00:00, 88.15batch/s, mse=4.17e+11]
Epoch 2: 100%|██████████| 54/54 [00:00<00:00, 86.85batch/s, mse=4.16e+11]
Epoch 3: 100%|██████████| 54/54 [00:00<00:00, 84.46batch/s, mse=4.14e+11]
Epoch 4: 100%|██████████| 54/54 [00:00<00:00, 83.17batch/s, mse=4.09e+11]
Epoch 5: 100%|██████████| 54/54 [00:00<00:00, 84.85batch/s, mse=3.99e+11]
Epoch 6: 100%|██████████| 54/54 [00:00<00:00, 85.81batch/s, mse=3.81e+11]
Epoch 7: 100%|██████████| 54/54 [00:00<00:00, 86.18batch/s, mse=3.54e+11]
Epoch 8: 100%|██████████| 54/54 [00:00<00:00, 80.89batch/s, mse=3.17e+11]
Epoch 9: 100%|██████████| 54/54 [00:00<00:00, 79.48batch/s, mse=2.71e+11]
Epoch 10: 100%|██████████| 54/54 [00:00<00:00, 71.16batch/s, mse=2.22e+11]
Epoch 11: 100%|██████████| 54/54 [00:00<00:00, 67.16batch/s, mse=1.76e+11]
Epoch 12: 100%|██████████| 54/54 [00:00<00:00, 71.01batch/s, mse=1.39e+11]
Epoch 13: 100%|██████████| 54/54 [00:00<00:00, 64.14batch/s, mse=1.13e+11]
Epoch 14: 100%|██████████| 54/54 [00:00<00:00, 72.76batch/s, mse=9.72e+10]
Epoch 15: 100%|██████████| 54/54 [00:00<00:00, 67.06batch/s, mse=8.75e+10]
Epoch 16: 100%|██████████| 54/54 [00:00<00:00, 61.67batch/s, mse=8.11e+10]
Epoch 17: 100%|██████████| 54/54 [00:00<00:00, 80.18batch/s, mse=7.65e+10]
Epoch 18: 100%|██████████| 54/54 [00:00<00:00, 82.58batch/s, mse=7.3e+10]
Epoch 19: 100%|██████████| 54/54 [00:00<00:00, 94.26batch/s, mse=7.03e+10]
Epoch 20: 100%|██████████| 54/54 [00:00<00:00, 96.54batch/s, mse=6.81e+10]
Epoch 21: 100%|██████████| 54/54 [00:00<00:00, 82.10batch/s, mse=6.63e+10]
Epoch 22: 100%|██████████| 54/54 [00:00<00:00, 84.48batch/s, mse=6.48e+10]
Epoch 23: 100%|██████████| 54/54 [00:00<00:00, 83.82batch/s, mse=6.36e+10]
Epoch 24: 100%|██████████| 54/54 [00:00<00:00, 79.68batch/s, mse=6.25e+10]
Epoch 25: 100%|██████████| 54/54 [00:00<00:00, 85.30batch/s, mse=6.15e+10]
Epoch 26: 100%|██████████| 54/54 [00:00<00:00, 83.87batch/s, mse=6.05e+10]
Epoch 27: 100%|██████████| 54/54 [00:00<00:00, 87.20batch/s, mse=5.97e+10]
Epoch 28: 100%|██████████| 54/54 [00:00<00:00, 86.97batch/s, mse=5.88e+10]
Epoch 29: 100%|██████████| 54/54 [00:00<00:00, 82.12batch/s, mse=5.8e+10]
Epoch 30: 100%|██████████| 54/54 [00:00<00:00, 80.66batch/s, mse=5.71e+10]
Epoch 31: 100%|██████████| 54/54 [00:00<00:00, 88.99batch/s, mse=5.63e+10]
Epoch 32: 100%|██████████| 54/54 [00:00<00:00, 72.73batch/s, mse=5.55e+10]
Epoch 33: 100%|██████████| 54/54 [00:00<00:00, 62.27batch/s, mse=5.47e+10]
Epoch 34: 100%|██████████| 54/54 [00:00<00:00, 65.33batch/s, mse=5.39e+10]
Epoch 35: 100%|██████████| 54/54 [00:00<00:00, 65.15batch/s, mse=5.31e+10]
Epoch 36: 100%|██████████| 54/54 [00:00<00:00, 68.53batch/s, mse=5.22e+10]
Epoch 37: 100%|██████████| 54/54 [00:00<00:00, 67.65batch/s, mse=5.14e+10]
Epoch 38: 100%|██████████| 54/54 [00:00<00:00, 72.12batch/s, mse=5.06e+10]
Epoch 39: 100%|██████████| 54/54 [00:00<00:00, 58.45batch/s, mse=4.97e+10]
Epoch 40: 100%|██████████| 54/54 [00:00<00:00, 68.49batch/s, mse=4.89e+10]
Epoch 41: 100%|██████████| 54/54 [00:00<00:00, 85.14batch/s, mse=4.81e+10]
Epoch 42: 100%|██████████| 54/54 [00:00<00:00, 87.94batch/s, mse=4.72e+10]
Epoch 43: 100%|██████████| 54/54 [00:00<00:00, 89.17batch/s, mse=4.64e+10]
Epoch 44: 100%|██████████| 54/54 [00:00<00:00, 81.90batch/s, mse=4.56e+10]
Epoch 45: 100%|██████████| 54/54 [00:00<00:00, 83.03batch/s, mse=4.47e+10]
Epoch 46: 100%|██████████| 54/54 [00:00<00:00, 80.48batch/s, mse=4.39e+10]
Epoch 47: 100%|██████████| 54/54 [00:00<00:00, 87.34batch/s, mse=4.31e+10]
Epoch 48: 100%|██████████| 54/54 [00:00<00:00, 79.70batch/s, mse=4.23e+10]
Epoch 49: 100%|██████████| 54/54 [00:00<00:00, 81.51batch/s, mse=4.14e+10]
Epoch 50: 100%|██████████| 54/54 [00:00<00:00, 78.86batch/s, mse=4.06e+10]
Epoch 51: 100%|██████████| 54/54 [00:00<00:00, 80.31batch/s, mse=3.99e+10]
Epoch 52: 100%|██████████| 54/54 [00:00<00:00, 84.06batch/s, mse=3.91e+10]
Epoch 53: 100%|██████████| 54/54 [00:00<00:00, 80.26batch/s, mse=3.83e+10]
Epoch 54: 100%|██████████| 54/54 [00:00<00:00, 84.21batch/s, mse=3.76e+10]
Epoch 55: 100%|██████████| 54/54 [00:00<00:00, 73.00batch/s, mse=3.69e+10]
Epoch 56: 100%|██████████| 54/54 [00:00<00:00, 88.31batch/s, mse=3.62e+10]
Epoch 57: 100%|██████████| 54/54 [00:00<00:00, 65.73batch/s, mse=3.56e+10]
```

```
Epoch 58: 100%|██████████| 54/54 [00:00<00:00, 62.00batch/s, mse=3.49e+10]
Epoch 59: 100%|██████████| 54/54 [00:00<00:00, 71.02batch/s, mse=3.43e+10]
Epoch 60: 100%|██████████| 54/54 [00:00<00:00, 61.45batch/s, mse=3.37e+10]
Epoch 61: 100%|██████████| 54/54 [00:00<00:00, 77.31batch/s, mse=3.32e+10]
Epoch 62: 100%|██████████| 54/54 [00:00<00:00, 67.85batch/s, mse=3.27e+10]
Epoch 63: 100%|██████████| 54/54 [00:00<00:00, 67.35batch/s, mse=3.22e+10]
Epoch 64: 100%|██████████| 54/54 [00:00<00:00, 61.89batch/s, mse=3.17e+10]
Epoch 65: 100%|██████████| 54/54 [00:00<00:00, 74.04batch/s, mse=3.13e+10]
Epoch 66: 100%|██████████| 54/54 [00:00<00:00, 76.44batch/s, mse=3.09e+10]
Epoch 67: 100%|██████████| 54/54 [00:00<00:00, 80.85batch/s, mse=3.05e+10]
Epoch 68: 100%|██████████| 54/54 [00:00<00:00, 82.05batch/s, mse=3.01e+10]
Epoch 69: 100%|██████████| 54/54 [00:00<00:00, 86.51batch/s, mse=2.98e+10]
Epoch 70: 100%|██████████| 54/54 [00:00<00:00, 88.63batch/s, mse=2.95e+10]
Epoch 71: 100%|██████████| 54/54 [00:00<00:00, 73.07batch/s, mse=2.92e+10]
Epoch 72: 100%|██████████| 54/54 [00:00<00:00, 73.54batch/s, mse=2.89e+10]
Epoch 73: 100%|██████████| 54/54 [00:00<00:00, 82.03batch/s, mse=2.86e+10]
Epoch 74: 100%|██████████| 54/54 [00:00<00:00, 91.03batch/s, mse=2.84e+10]
Epoch 75: 100%|██████████| 54/54 [00:00<00:00, 75.69batch/s, mse=2.82e+10]
Epoch 76: 100%|██████████| 54/54 [00:00<00:00, 84.53batch/s, mse=2.8e+10]
Epoch 77: 100%|██████████| 54/54 [00:00<00:00, 80.43batch/s, mse=2.79e+10]
Epoch 78: 100%|██████████| 54/54 [00:00<00:00, 91.56batch/s, mse=2.77e+10]
Epoch 79: 100%|██████████| 54/54 [00:00<00:00, 80.65batch/s, mse=2.76e+10]
Epoch 80: 100%|██████████| 54/54 [00:00<00:00, 81.23batch/s, mse=2.75e+10]
Epoch 81: 100%|██████████| 54/54 [00:00<00:00, 64.99batch/s, mse=2.74e+10]
Epoch 82: 100%|██████████| 54/54 [00:00<00:00, 59.18batch/s, mse=2.73e+10]
Epoch 83: 100%|██████████| 54/54 [00:00<00:00, 63.19batch/s, mse=2.72e+10]
Epoch 84: 100%|██████████| 54/54 [00:00<00:00, 65.05batch/s, mse=2.71e+10]
Epoch 85: 100%|██████████| 54/54 [00:00<00:00, 67.88batch/s, mse=2.7e+10]
Epoch 86: 100%|██████████| 54/54 [00:00<00:00, 69.98batch/s, mse=2.69e+10]
Epoch 87: 100%|██████████| 54/54 [00:00<00:00, 59.02batch/s, mse=2.69e+10]
Epoch 88: 100%|██████████| 54/54 [00:00<00:00, 60.73batch/s, mse=2.68e+10]
Epoch 89: 100%|██████████| 54/54 [00:00<00:00, 57.97batch/s, mse=2.68e+10]
Epoch 90: 100%|██████████| 54/54 [00:00<00:00, 84.76batch/s, mse=2.67e+10]
Epoch 91: 100%|██████████| 54/54 [00:00<00:00, 70.46batch/s, mse=2.67e+10]
Epoch 92: 100%|██████████| 54/54 [00:00<00:00, 74.76batch/s, mse=2.67e+10]
Epoch 93: 100%|██████████| 54/54 [00:00<00:00, 79.16batch/s, mse=2.66e+10]
Epoch 94: 100%|██████████| 54/54 [00:00<00:00, 76.99batch/s, mse=2.66e+10]
Epoch 95: 100%|██████████| 54/54 [00:00<00:00, 84.28batch/s, mse=2.66e+10]
Epoch 96: 100%|██████████| 54/54 [00:00<00:00, 75.96batch/s, mse=2.65e+10]
Epoch 97: 100%|██████████| 54/54 [00:00<00:00, 79.07batch/s, mse=2.65e+10]
Epoch 98: 100%|██████████| 54/54 [00:00<00:00, 79.07batch/s, mse=2.65e+10]
Epoch 99: 100%|██████████| 54/54 [00:00<00:00, 75.10batch/s, mse=2.65e+10]
Epoch 100: 100%|██████████| 54/54 [00:00<00:00, 79.54batch/s, mse=2.64e+10]
Epoch 101: 100%|██████████| 54/54 [00:00<00:00, 75.75batch/s, mse=2.64e+10]
Epoch 102: 100%|██████████| 54/54 [00:00<00:00, 83.90batch/s, mse=2.64e+10]
Epoch 103: 100%|██████████| 54/54 [00:00<00:00, 90.85batch/s, mse=2.64e+10]
Epoch 104: 100%|██████████| 54/54 [00:00<00:00, 96.56batch/s, mse=2.63e+10]
Epoch 105: 100%|██████████| 54/54 [00:00<00:00, 95.38batch/s, mse=2.63e+10]
Epoch 106: 100%|██████████| 54/54 [00:00<00:00, 62.22batch/s, mse=2.63e+10]
Epoch 107: 100%|██████████| 54/54 [00:00<00:00, 59.94batch/s, mse=2.63e+10]
Epoch 108: 100%|██████████| 54/54 [00:00<00:00, 59.36batch/s, mse=2.63e+10]
Epoch 109: 100%|██████████| 54/54 [00:00<00:00, 62.98batch/s, mse=2.62e+10]
Epoch 110: 100%|██████████| 54/54 [00:00<00:00, 59.04batch/s, mse=2.62e+10]
Epoch 111: 100%|██████████| 54/54 [00:00<00:00, 56.59batch/s, mse=2.62e+10]
Epoch 112: 100%|██████████| 54/54 [00:00<00:00, 56.62batch/s, mse=2.62e+10]
Epoch 113: 100%|██████████| 54/54 [00:00<00:00, 56.78batch/s, mse=2.62e+10]
Epoch 114: 100%|██████████| 54/54 [00:00<00:00, 57.86batch/s, mse=2.61e+10]
Epoch 115: 100%|██████████| 54/54 [00:00<00:00, 67.51batch/s, mse=2.61e+10]
Epoch 116: 100%|██████████| 54/54 [00:00<00:00, 74.16batch/s, mse=2.61e+10]
Epoch 117: 100%|██████████| 54/54 [00:00<00:00, 80.00batch/s, mse=2.61e+10]
Epoch 118: 100%|██████████| 54/54 [00:00<00:00, 76.78batch/s, mse=2.61e+10]
Epoch 119: 100%|██████████| 54/54 [00:00<00:00, 74.64batch/s, mse=2.6e+10]
Epoch 120: 100%|██████████| 54/54 [00:00<00:00, 76.53batch/s, mse=2.6e+10]
Epoch 121: 100%|██████████| 54/54 [00:00<00:00, 86.64batch/s, mse=2.6e+10]
Epoch 122: 100%|██████████| 54/54 [00:00<00:00, 93.44batch/s, mse=2.6e+10]
Epoch 123: 100%|██████████| 54/54 [00:00<00:00, 83.26batch/s, mse=2.6e+10]
Epoch 124: 100%|██████████| 54/54 [00:00<00:00, 65.42batch/s, mse=2.6e+10]
Epoch 125: 100%|██████████| 54/54 [00:00<00:00, 85.08batch/s, mse=2.59e+10]
Epoch 126: 100%|██████████| 54/54 [00:00<00:00, 70.53batch/s, mse=2.59e+10]
Epoch 127: 100%|██████████| 54/54 [00:00<00:00, 65.82batch/s, mse=2.59e+10]
Epoch 128: 100%|██████████| 54/54 [00:00<00:00, 82.97batch/s, mse=2.59e+10]
Epoch 129: 100%|██████████| 54/54 [00:00<00:00, 80.60batch/s, mse=2.59e+10]
```

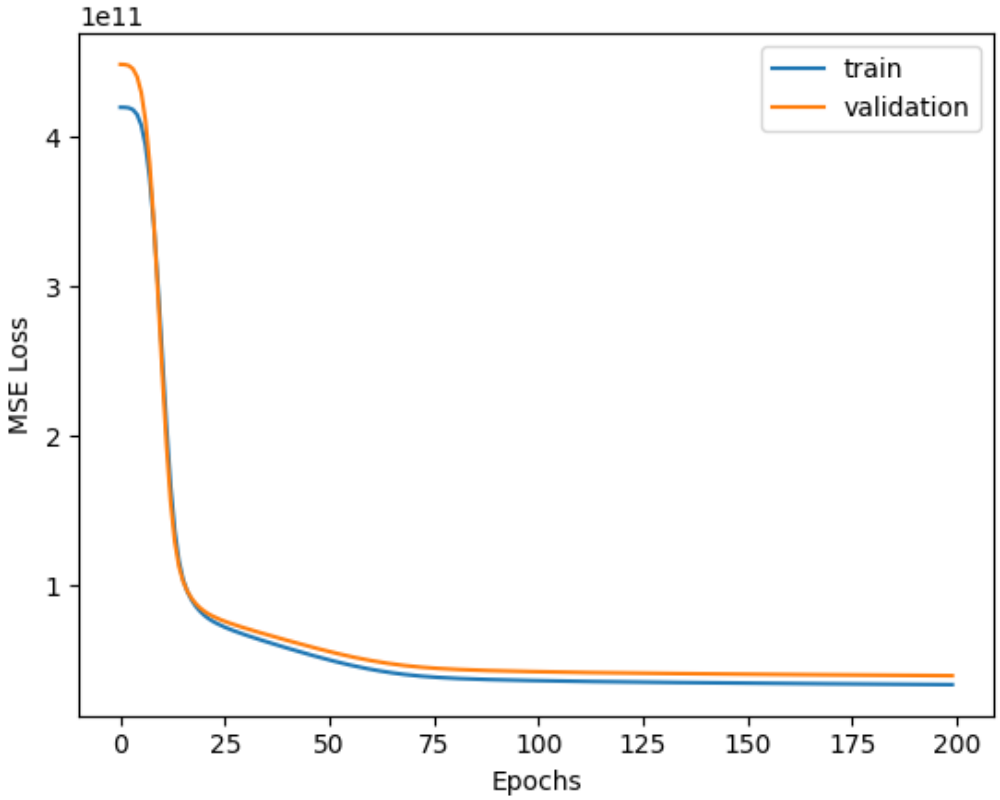
```
Epoch 130: 100%|██████████| 54/54 [00:00<00:00, 70.23batch/s, mse=2.59e+10]
Epoch 131: 100%|██████████| 54/54 [00:00<00:00, 54.28batch/s, mse=2.58e+10]
Epoch 132: 100%|██████████| 54/54 [00:00<00:00, 55.10batch/s, mse=2.58e+10]
Epoch 133: 100%|██████████| 54/54 [00:00<00:00, 57.16batch/s, mse=2.58e+10]
Epoch 134: 100%|██████████| 54/54 [00:00<00:00, 54.64batch/s, mse=2.58e+10]
Epoch 135: 100%|██████████| 54/54 [00:01<00:00, 52.07batch/s, mse=2.58e+10]
Epoch 136: 100%|██████████| 54/54 [00:00<00:00, 55.20batch/s, mse=2.58e+10]
Epoch 137: 100%|██████████| 54/54 [00:00<00:00, 54.74batch/s, mse=2.57e+10]
Epoch 138: 100%|██████████| 54/54 [00:00<00:00, 68.88batch/s, mse=2.57e+10]
Epoch 139: 100%|██████████| 54/54 [00:00<00:00, 61.30batch/s, mse=2.57e+10]
Epoch 140: 100%|██████████| 54/54 [00:00<00:00, 64.95batch/s, mse=2.57e+10]
Epoch 141: 100%|██████████| 54/54 [00:00<00:00, 75.99batch/s, mse=2.57e+10]
Epoch 142: 100%|██████████| 54/54 [00:00<00:00, 86.07batch/s, mse=2.56e+10]
Epoch 143: 100%|██████████| 54/54 [00:00<00:00, 64.62batch/s, mse=2.56e+10]
Epoch 144: 100%|██████████| 54/54 [00:00<00:00, 77.44batch/s, mse=2.56e+10]
Epoch 145: 100%|██████████| 54/54 [00:00<00:00, 75.96batch/s, mse=2.56e+10]
Epoch 146: 100%|██████████| 54/54 [00:00<00:00, 78.23batch/s, mse=2.56e+10]
Epoch 147: 100%|██████████| 54/54 [00:00<00:00, 74.50batch/s, mse=2.56e+10]
Epoch 148: 100%|██████████| 54/54 [00:00<00:00, 77.96batch/s, mse=2.55e+10]
Epoch 149: 100%|██████████| 54/54 [00:00<00:00, 71.50batch/s, mse=2.55e+10]
Epoch 150: 100%|██████████| 54/54 [00:00<00:00, 71.28batch/s, mse=2.55e+10]
Epoch 151: 100%|██████████| 54/54 [00:00<00:00, 76.84batch/s, mse=2.55e+10]
Epoch 152: 100%|██████████| 54/54 [00:00<00:00, 86.05batch/s, mse=2.55e+10]
Epoch 153: 100%|██████████| 54/54 [00:00<00:00, 91.55batch/s, mse=2.54e+10]
Epoch 154: 100%|██████████| 54/54 [00:00<00:00, 90.60batch/s, mse=2.54e+10]
Epoch 155: 100%|██████████| 54/54 [00:00<00:00, 60.28batch/s, mse=2.54e+10]
Epoch 156: 100%|██████████| 54/54 [00:00<00:00, 65.28batch/s, mse=2.54e+10]
Epoch 157: 100%|██████████| 54/54 [00:00<00:00, 63.64batch/s, mse=2.54e+10]
Epoch 158: 100%|██████████| 54/54 [00:00<00:00, 56.96batch/s, mse=2.54e+10]
Epoch 159: 100%|██████████| 54/54 [00:00<00:00, 65.51batch/s, mse=2.53e+10]
Epoch 160: 100%|██████████| 54/54 [00:00<00:00, 57.93batch/s, mse=2.53e+10]
Epoch 161: 100%|██████████| 54/54 [00:01<00:00, 53.73batch/s, mse=2.53e+10]
Epoch 162: 100%|██████████| 54/54 [00:00<00:00, 61.95batch/s, mse=2.53e+10]
Epoch 163: 100%|██████████| 54/54 [00:00<00:00, 63.06batch/s, mse=2.53e+10]
Epoch 164: 100%|██████████| 54/54 [00:00<00:00, 67.13batch/s, mse=2.52e+10]
Epoch 165: 100%|██████████| 54/54 [00:00<00:00, 79.57batch/s, mse=2.52e+10]
Epoch 166: 100%|██████████| 54/54 [00:00<00:00, 77.76batch/s, mse=2.52e+10]
Epoch 167: 100%|██████████| 54/54 [00:00<00:00, 69.25batch/s, mse=2.52e+10]
Epoch 168: 100%|██████████| 54/54 [00:00<00:00, 67.64batch/s, mse=2.52e+10]
Epoch 169: 100%|██████████| 54/54 [00:00<00:00, 88.00batch/s, mse=2.52e+10]
Epoch 170: 100%|██████████| 54/54 [00:00<00:00, 66.67batch/s, mse=2.51e+10]
Epoch 171: 100%|██████████| 54/54 [00:00<00:00, 63.70batch/s, mse=2.51e+10]
Epoch 172: 100%|██████████| 54/54 [00:00<00:00, 66.13batch/s, mse=2.51e+10]
Epoch 173: 100%|██████████| 54/54 [00:00<00:00, 65.49batch/s, mse=2.51e+10]
Epoch 174: 100%|██████████| 54/54 [00:00<00:00, 65.81batch/s, mse=2.51e+10]
Epoch 175: 100%|██████████| 54/54 [00:00<00:00, 74.74batch/s, mse=2.5e+10]
Epoch 176: 100%|██████████| 54/54 [00:00<00:00, 64.98batch/s, mse=2.5e+10]
Epoch 177: 100%|██████████| 54/54 [00:00<00:00, 57.87batch/s, mse=2.5e+10]
Epoch 178: 100%|██████████| 54/54 [00:00<00:00, 58.83batch/s, mse=2.5e+10]
Epoch 179: 100%|██████████| 54/54 [00:00<00:00, 61.07batch/s, mse=2.5e+10]
Epoch 180: 100%|██████████| 54/54 [00:00<00:00, 60.16batch/s, mse=2.49e+10]
Epoch 181: 100%|██████████| 54/54 [00:00<00:00, 60.81batch/s, mse=2.49e+10]
Epoch 182: 100%|██████████| 54/54 [00:00<00:00, 58.75batch/s, mse=2.49e+10]
Epoch 183: 100%|██████████| 54/54 [00:00<00:00, 60.54batch/s, mse=2.49e+10]
Epoch 184: 100%|██████████| 54/54 [00:00<00:00, 56.99batch/s, mse=2.49e+10]
Epoch 185: 100%|██████████| 54/54 [00:00<00:00, 57.59batch/s, mse=2.49e+10]
Epoch 186: 100%|██████████| 54/54 [00:00<00:00, 67.23batch/s, mse=2.48e+10]
Epoch 187: 100%|██████████| 54/54 [00:00<00:00, 69.42batch/s, mse=2.48e+10]
Epoch 188: 100%|██████████| 54/54 [00:00<00:00, 73.74batch/s, mse=2.48e+10]
Epoch 189: 100%|██████████| 54/54 [00:00<00:00, 70.93batch/s, mse=2.48e+10]
Epoch 190: 100%|██████████| 54/54 [00:00<00:00, 72.14batch/s, mse=2.48e+10]
Epoch 191: 100%|██████████| 54/54 [00:00<00:00, 93.75batch/s, mse=2.48e+10]
Epoch 192: 100%|██████████| 54/54 [00:00<00:00, 92.26batch/s, mse=2.47e+10]
Epoch 193: 100%|██████████| 54/54 [00:00<00:00, 99.20batch/s, mse=2.47e+10]
Epoch 194: 100%|██████████| 54/54 [00:00<00:00, 82.08batch/s, mse=2.47e+10]
Epoch 195: 100%|██████████| 54/54 [00:00<00:00, 67.30batch/s, mse=2.47e+10]
Epoch 196: 100%|██████████| 54/54 [00:00<00:00, 65.87batch/s, mse=2.47e+10]
Epoch 197: 100%|██████████| 54/54 [00:00<00:00, 69.32batch/s, mse=2.47e+10]
Epoch 198: 100%|██████████| 54/54 [00:00<00:00, 67.20batch/s, mse=2.46e+10]
Epoch 199: 100%|██████████| 54/54 [00:00<00:00, 55.53batch/s, mse=2.46e+10]

Validation MSE: 39553253376.00
Validation RMSE: 198880.00
```

Validation RMSE: 196880.00

In [14]:

```
# restore model using the .load_state_dict() function
##  START CODE  ## (1 line of code)
big_model.load_state_dict(big_model_best_weights)
##  END CODE  ##

# calculate test performance using the eval function we defined above (MSE and R2)
##  START CODE  ##
test_mse, test_r2 = eval(big_model, X_test, y_test)
##  END CODE  ##

print("Test MSE: %.2f" % test_mse)
print("Test R2: %.2f" % test_r2)
plt.plot(big_model_train_history, label='train')
plt.plot(big_model_valid_history, label='validation')
plt.legend(loc="upper right")
plt.xlabel('Epochs')
plt.ylabel('MSE Loss')
plt.show()
```

```
Test MSE: 38974840832.00
Test R2: 0.72
```



What do you notice in the curve above? In particular, what happens to the train loss vs validation loss as you keep training the model? Why does this happen and what technique(s) can you use to avoid this?

Answer

Both the training and validation loss curves decrease and converge without diverging, suggesting the model is not overfitting. Both losses seem to reach a point where further training does not significantly improve performance.

Usually when you keep training the model it starts to byheart all the noise which makes it overfit and wrongly predict on unknown data depicted by a divergence between the curves.

to avoid Overfitting- we can

- **Early Stopping: Monitor the validation loss during training and stop when it begins to increase, as this would be a sign of overfitting**

be a sign of overfitting.

- **Regularization - Apply techniques like L2 regularization**
- **Cross-validation**

Q3. (10 points) - Principal Component Analysis

Consider a set of data points $\{x_1, x_2, \dots x_N\}$ where each $x_i \in \mathbb{R}^d$, given to you after centering, i.e., $\frac{1}{N}\sum_{i=1}^{N} x_i = 0$.

Now suppose you want to project this data on a single unit vector given by u by learning an appropriate u. Show that, for learning u, minimizing the residual of the projections computed by the squared error between the projected data and the original data is equivalent to maximizing the variance of the projections. Hint : the projection of an x on a unit vector u is given by $(x^T u)u$.

Given:

We have a set of data points $\{x_1, x_2, \dots, x_N\}$ where each $x_i \in \mathbb{R}^d$, and the data has been centered, i.e.,

$$\frac{1}{N}\sum_{i=1}^{N} x_i = 0.$$

We aim to project this data onto a single unit vector u, where $u \in \mathbb{R}^d$ and $\|u\| = 1$. The projection of a data point x_i on the vector u is given by:

$$\text{Projection of } x_i \text{ on } u = (x_i^T u)u.$$

We are tasked with showing that minimizing the residual between the projected data and the original data (in terms of squared error) is equivalent to maximizing the variance of the projections.

Step 1: Residual Minimization (Squared Error)

The residual between the original data point x_i and its projection onto u is:

$$r_i = x_i - (x_i^T u)u.$$

The squared residual (error) for all data points is the sum of squared errors:

$$R(u) = \sum_{i=1}^{N} \|x_i - (x_i^T u)u\|^2.$$

Now, we expand this expression for the residual:

$$R(u) = \sum_{i=1}^{N} \left(x_i^T x_i - 2(x_i^T u)(u^T x_i) + (x_i^T u)^2 \right).$$

Using the fact that $u^T u = 1$ and simplifying:

$$R(u) = \sum_{i=1}^{N} \left(\|x_i\|^2 - (x_i^T u)^2 \right).$$

Thus, the total squared error is:

$$R(u) = \sum_{i=1}^{N} \|x_i\|^2 - \sum_{i=1}^{N} (x_i^T u)^2$$

$$\sum_{i=1}^{N}(x_i - u)^2.$$

Step 2: Maximizing Variance of Projections

The variance of the projections onto u is:

$$\text{Var}(u) = \frac{1}{N}$$

$$\sum_{i=1}^{N}(x_i^T u)^2.$$

We want to maximize this variance with respect to u. Notice that the term we want to maximize, $\sum_{i=1}^{N}(x_i^T u)^2$, is already part of the squared residual error expression.

Step 3: Connecting Minimization and Maximization

From the residual error expression:

$$R(u) = \sum_{i=1}^{N}\|x_i\|^2$$

$$-\sum_{i=1}^{N}(x_i^T u)^2,$$

we see that minimizing the residual error $R(u)$ is equivalent to maximizing the term $\sum_{i=1}^{N}(x_i^T u)^2$, which is directly related to the variance of the projections. Therefore, minimizing the squared error is equivalent to maximizing the variance of the projections.

Conclusion:

Minimizing the squared error of the projections is equivalent to maximizing the variance of the projections because the residual error can be decomposed into a constant term minus the variance of the projections. By minimizing the residual error, we are effectively maximizing the variance of the projections.

Q4. (25 points) - Principal Component Analysis

In this problem we will be applying PCA and T-SNE on the Superconductivity Dataset. More details on the dataset is present here. The goal here is to predict the critical temperature of a superconductor based on the features extracted.

First use Principal Component Analysis (PCA) to solve this problem.

- **Part 1. (5 points)** Perform the following steps to prepare the dataset:
  - Load the dataset from the "Q4data.csv" file provided as a dataframe df.
  - Select the **'critical_temp'** column as the target column and the rest of the columns from the dataframe df as **X.**
  - Split the dataset into train and test set with 35% data in test set and random_state = 42
  - Perform Standard Scaling on the dataset. Remember that when we have training and testing data, we fit preprocessing parameters on training data and apply them to all testing data. You should scale only the features (independent variables), not the target variable y.

    `Note: X should have 81 features.`

In [ ]:

```
# Only use this code block if you are using Google Colab.
# If you are using Jupyter Notebook, please ignore this code block. You can directly uplo
ad the file to your Jupyter Notebook file systems.
from google.colab import files

## It will prompt you to select a local file. Click on "Choose Files" then select and upl
```

```
oad the file.
## Wait for the file to be 100% uploaded. You should see the name of the file once Colab has uploaded it.
uploaded = files.upload()
```

Choose File **No file selected**

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Q4data.csv to Q4data.csv
```

In [ ]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SequentialFeatureSelector
import os, sys, re
import time
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler,StandardScaler
import matplotlib.pyplot as plt
df = pd.read_csv("/content/Q4data.csv")
```

In [ ]:

```
y = df["critical_temp"]
X = df.drop(columns=["critical_temp"])

X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.35, random_state=42)

scalar = StandardScaler()

### START CODE ###
### Scale the dataset
X_train = scalar.fit_transform(X_train)
X_test = scalar.transform(X_test)
### END CODE ###
```

- **Part 2a (3 points) Use PCA and reduce the dimension of X_train to the following number of components: `[3,20,40,60,81]`. For each of the five datasets, print the cumulative variance explained by the principal components `N = [3,20,40,60,81]` .(i.e. what percentage of variance in the original dataset is explained if we transform the dataset to have 3,20,40,60 and 81 principal components respectively).**

  `Note : PCA should be fit on X_train and the components thus learnt should be later used to transform X_test`

In [ ]:

```
from sklearn.decomposition import PCA

nums = [3, 20, 40, 60, 81]
res = []

for num in nums:
    ### START CODE ###
    ## Fit PCA
    pca = PCA(n_components=num)
    pca_fitter = pca.fit(X_train)
    ### END CODE ###

    ### START CODE ###
    ## Transform Data
    X_train_new = pca_fitter.transform(X_train)
    ### END CODE ###
```

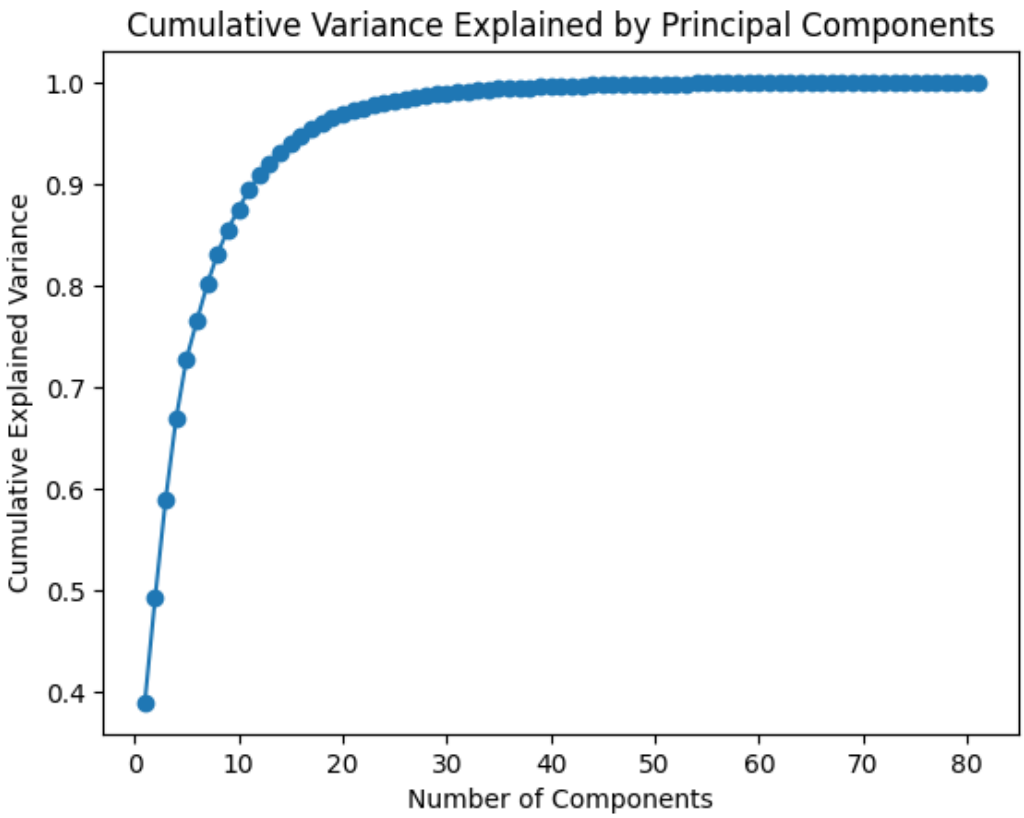
```
    ### START CODE ###
    ## Compute explained variance
    var = pca_fitter.explained_variance_ratio_.cumsum()
    ### END CODE ###

    print("Cumulative variance explained by {} components is {}".format(num, var[-1]))
```

```
Cumulative variance explained by 3 components is 0.5894367932307206
Cumulative variance explained by 20 components is 0.9694250665596833
Cumulative variance explained by 40 components is 0.9961465484729853
Cumulative variance explained by 60 components is 0.9995333375490648
Cumulative variance explained by 81 components is 1.0000000000000004
```

- **Part 2b. (2 points)** **Plot the cumulative variance explained by the principal components using the training data. The plot should display the number of components on X-axis and the cumulative explained variance on the y-axis. What do you understand from the plot obtained?**

In [ ]:

```
### START CODE ###
## Plot the explained variance vs number of components
cumulative_variance = []
nums = list(range(1, 82))

for num in nums:
    pca = PCA(n_components=num)
    pca_fitter = pca.fit(X_train)
    var = pca_fitter.explained_variance_ratio_.cumsum()
    cumulative_variance.append(var[-1])  # Store final cumulative variance for each num

### Plot the explained variance vs. number of components
plt.plot(nums, cumulative_variance, marker='o')
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.title("Cumulative Variance Explained by Principal Components")
plt.show()
### END CODE ###

plt.show()
```



Sharp Increase in Variance Explained: Initially, there is a sharp increase in the cumulative variance explained as the number of principal components increases, especially within the first 20 components. This suggests that a

small number of components captures most of the variability in the data.

Diminishing Returns: After about 20 to 30 components, the cumulative variance explained begins to level off, reaching close to 1 (or 100% of the total variance). This indicates that the additional principal components beyond this point contribute very little to explaining more variance.

Choosing the Optimal Number of Components: The plot helps in selecting the optimal number of components to reduce dimensionality while retaining most of the information. Based on this plot, we can conclude that using the first 20-30 components will capture nearly all of the variance in the data, meaning further components may not be necessary for effective dimensionality reduction.

This insight is useful for deciding on how much dimensionality reduction is appropriate without losing too much information from the original data.

- **Part 3. (5 points)** **For each of the reduced dataset, obtained in part 2.2, fit a linear regression model on the train data and show how adjusted R^2 varies as a function of # of components.(There will be a total of 5 R^2 score).**

In [ ]:

```
from sklearn.decomposition import PCA
nums = [3, 20, 40, 60, 81]
res = []

for num in nums:
    ### START CODE ###
    ## Fit PCA components
    pca = PCA(n_components=num)
    pca_fitter = pca.fit(X_train)
    ### END CODE ###

    ### START CODE ###
    ## Transform train and test data
    X_train_new = pca_fitter.transform(X_train)
    X_test_new = pca_fitter.transform(X_test)
    ### END CODE ###

    ### START CODE ###
    ## Compute explained variance
    percent_variance = pca_fitter.explained_variance_ratio_
    var = percent_variance.cumsum()  # Cumulative variance explained
    ### END CODE ###

    ### START CODE ###
    ## Fit LR and compute R-square and adjusted R-squared
    lr = LinearRegression()
    lr.fit(X_train_new, Y_train)  # Train the model
    r_squared = lr.score(X_test_new, Y_test)  # Compute R^2 score
    print("R^2 score {} with num_components {}".format(r_squared, num))  # hide
    ### END CODE ###

    adjusted_r_squared = 1 - (1 - r_squared) * (len(Y_test) - 1) / (len(Y_test) - X_test
_new.shape[1] - 1)
    print("Adjusted R^2", adjusted_r_squared)
```

```
R^2 score 0.49295596980634504 with num_components 3
Adjusted R^2 0.4927514890843957
R^2 score 0.6250421742885688 with num_components 20
Adjusted R^2 0.6240317786385784
R^2 score 0.6899454132683164 with num_components 40
Adjusted R^2 0.6882698953719009
R^2 score 0.7178670621821962 with num_components 60
Adjusted R^2 0.715573919907871
R^2 score 0.7307154603807391 with num_components 81
Adjusted R^2 0.7277522695494444
```

- **Part 4. T-SNE (3 points)** **Now apply T-SNE to the dataset given above. You are required to carry out the following tasks:**

1. **Initialize a t-SNE model with number of dimensions = 3, perplexity = 300, number of iterations = 300 and random state = 42**
2. **Apply the t-SNE model to the training dataset**

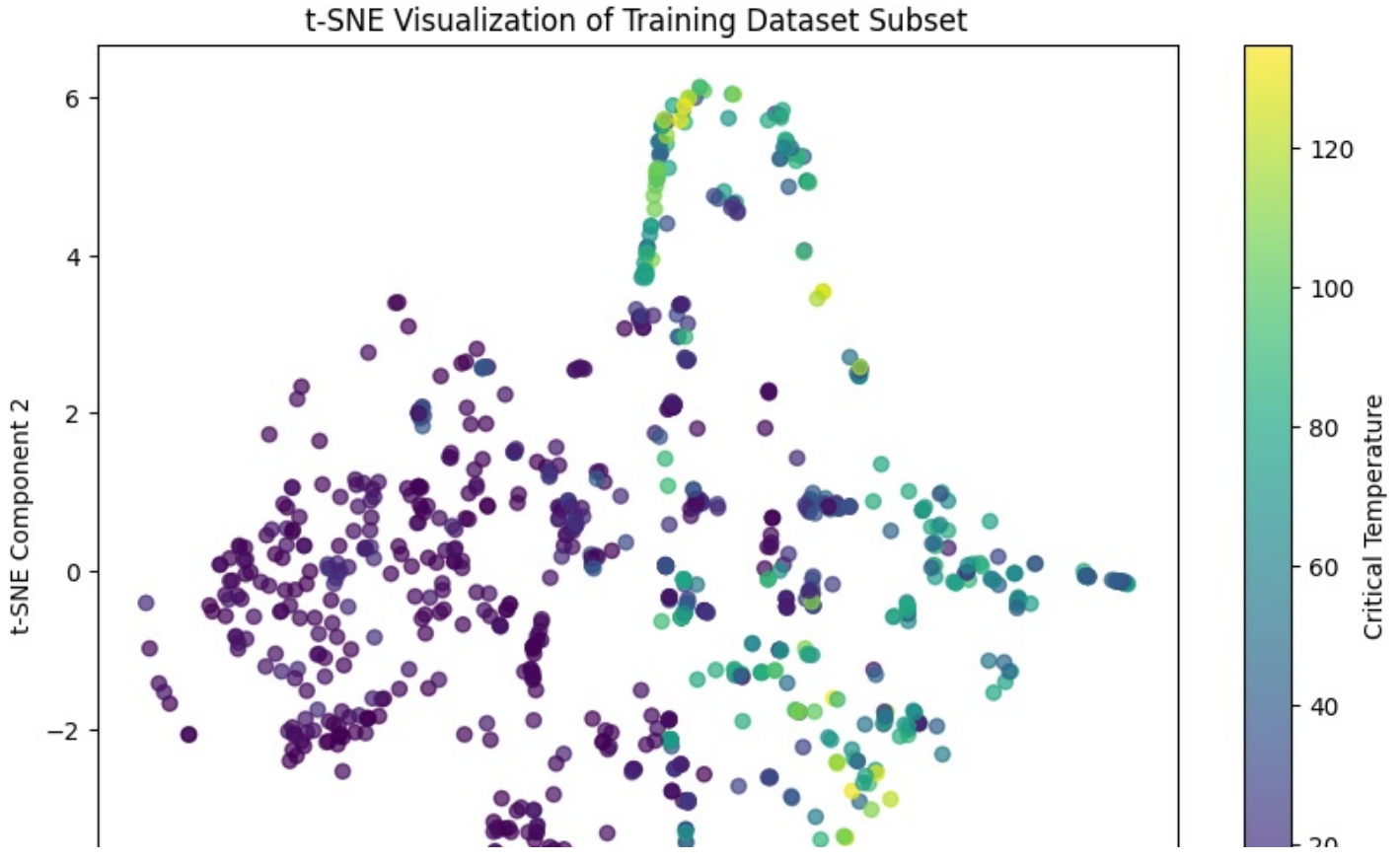
In [ ]:

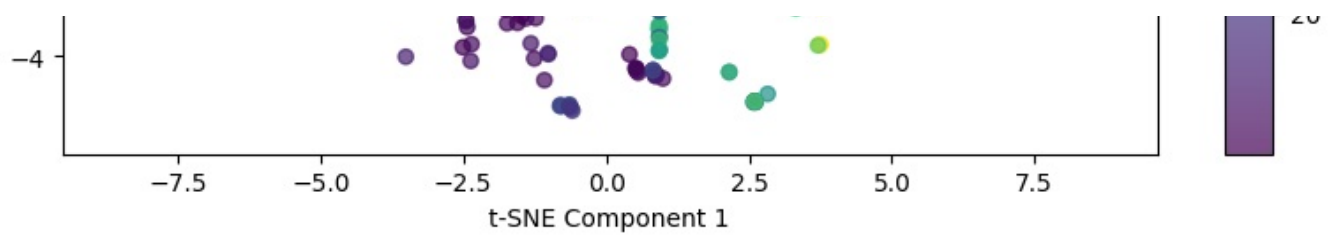
```
from sklearn.manifold import TSNE

### START CODE ###
## Initialize t-SNE model
tsne = TSNE(n_components=3, perplexity=300, max_iter=300, random_state=42)
### END CODE ###

### START CODE ###
## Fit and transform the data
X_tsne = tsne.fit_transform(X_train)
### END CODE ###
```

- **Part 5. (3 points) For this part use a small subset of 1000 samples of the training dataset and plot these data points w.r.t any two t-SNE components, and color of each point depicting the y-value**

In [ ]:

```
### START CODE
# Select a subset of 1000 samples
X_tsne_subset = X_tsne[:1000]  # Select first 1000 samples from t-SNE output
y_subset = Y_train[:1000]  # Corresponding y-values

# Create a scatter plot
plt.figure(figsize=(10, 7))
scatter = plt.scatter(X_tsne_subset[:, 0], X_tsne_subset[:, 1], c=y_subset, cmap='viridis
', alpha=0.7)
plt.colorbar(scatter, label='Critical Temperature')  # Color bar to indicate the temperat
ure values
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.title('t-SNE Visualization of Training Dataset Subset')
plt.show()
### END CODE
```

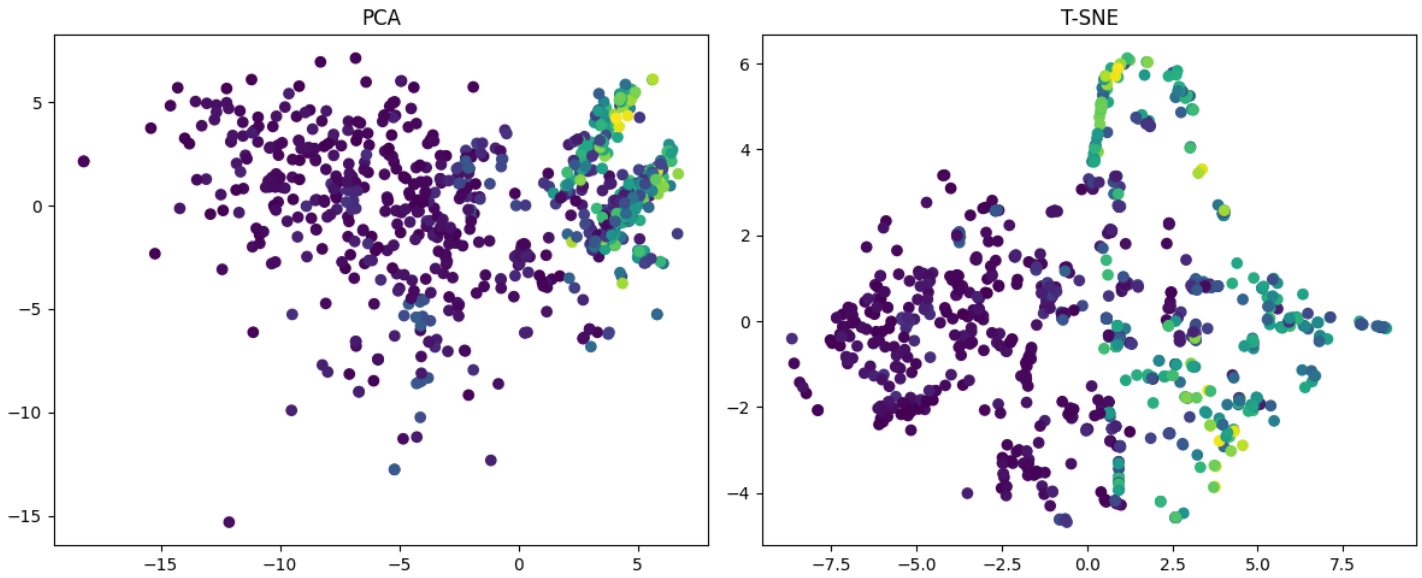




- **Part 6. (4 points)** **Now we will plot the PCA and t-SNE projections of the data and compare the plots side-by-side to see the difference in scatters created by the two methods. You can use first 1000 data points for this.**

In [ ]:

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

plt.figure(figsize=(12, 5))  # Adjust the figure size as needed

# First subplot (left)
### START CODE ###
### Obtain components from PCA and plot
pca = PCA(n_components=2)  # Use 2 components for 2D plotting
pca_fitter = pca.fit(X_train[:1000])  # Fit PCA on the first 1000 data points
x_pca = pca_fitter.transform(X_train[:1000])  # Transform the first 1000 data points
### END CODE ###

plt.subplot(1, 2, 1)  # 1 row, 2 columns, select the first subplot
plt.scatter(x_pca[:, 0], x_pca[:, 1],  c=y_subset, cmap='viridis')  # Plot the first two
PCA components
plt.title('PCA')

# Second subplot (right)
plt.subplot(1, 2, 2)  # 1 row, 2 columns, select the second subplot

### START CODE ###
### Scatter plot for t-SNE
plt.scatter(X_tsne[:1000, 0], X_tsne[:1000, 1],c=y_subset, cmap='viridis')  # Plot the f
irst two t-SNE components
### END CODE ###

plt.title('T-SNE')

# Adjust spacing between subplots
plt.tight_layout()

# Show the plots
plt.show()
```



Q5. (0 points) Outlier detection using PyOD - UNGRADED