04_exercise_gradient_descent_robertson

February 17, 2025

Exercise 4: Gradient Descent for Linear Regression

CPSC 381/581: Machine Learning

Yale University

Instructor: Alex Wong

Student: Hailey Robertson

Prerequisites:

1. Enable Google Colaboratory as an app on your Google Drive account
2. Create a new Google Colab notebook, this will also create a "Colab Notebooks" directory under "MyDrive" i.e.

```
/content/drive/MyDrive/Colab Notebooks
```

3. Create the following directory structure in your Google Drive

```
/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Exercises
```

4. Move the 04_exercise_gradient_descent.ipynb into

```
/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Exercises
```

so that its absolute path is

```
/content/drive/MyDrive/Colab Notebooks/CPSC 381-581: Machine Learning/Exercises/04_exercise_gra
```

In this exercise, we will optimize a linear function for the regression task using the gradient descent for mean squared and half mean squared losses. We will test them on several datasets.

Submission:

1. Implement all TODOs in the code blocks below.
2. Report your training, validation, and testing scores.

```
Report validation and testing scores here.

For full credit, your mean squared error scores for models optimized using mean_squared and hal

***** Results of scikit-learn linear regression model on Diabetes dataset *****
Training set mean squared error: 25472.6663
Training set r-squared scores: -3.2907
Validation set mean squared error: 30376.7403
```

```
Validation set r-squared scores: -5.1176
Testing set mean squared error: 27024.0792
Testing set r-squared scores: -3.2604
***** Results of our linear regression model trained with mean_squared loss, alpha=0.0001 and T
Training set mean squared error: 28034.7946
Training set r-squared scores: -3.7223
Validation set mean squared error: 34538.4871
Validation set r-squared scores: -5.9557
Testing set mean squared error: 29713.4975
Testing set r-squared scores: -3.6844
***** Results of our linear regression model trained with half_mean_squared loss, alpha=0.0002
Training set mean squared error: 28034.7946
Training set r-squared scores: -3.7223
Validation set mean squared error: 34538.4871
Validation set r-squared scores: -5.9557
Testing set mean squared error: 29713.4975
Testing set r-squared scores: -3.6844
***** Results of scikit-learn linear regression model on California housing prices dataset ***
Training set mean squared error: 0.5989
Training set r-squared scores: 0.5463
Validation set mean squared error: 0.5922
Validation set r-squared scores: 0.5721
Testing set mean squared error: 0.6590
Testing set r-squared scores: 0.5185
***** Results of our linear regression model trained with mean_squared loss, alpha=1e-07 and T=
Training set mean squared error: 0.6496
Training set r-squared scores: 0.5080
Validation set mean squared error: 0.6348
Validation set r-squared scores: 0.5413
Testing set mean squared error: 0.7028
Testing set r-squared scores: 0.4865
***** Results of our linear regression model trained with half_mean_squared loss, alpha=2e-07 a
Training set mean squared error: 0.6496
Training set r-squared scores: 0.5080
Validation set mean squared error: 0.6348
Validation set r-squared scores: 0.5413
Testing set mean squared error: 0.7028
Testing set r-squared scores: 0.4865
```

3. List any collaborators.

```
Collaborators: Doe, Jane (Please write names in <Last Name, First Name> format)

Collaboration details: Discussed ... implementation details with Jane Doe.
```

Import packages

[1]:
```python
import numpy as np
import sklearn.datasets as skdata
```

```python
import sklearn.metrics as skmetrics
from sklearn.linear_model import LinearRegression as LinearRegressionSciKit
import warnings

warnings.filterwarnings(action='ignore')
np.random.seed = 1
```

Implementation of our Gradient Descent optimizer for mean squared and half mean squared loss

[2]:
```python
class GradientDescentOptimizer(object):

    def __init__(self):
        pass

    def __compute_gradients(self, w, x, y, loss_func):
        '''
        Returns the gradient of mean squared or half mean squared loss

        Arg(s):
            w : numpy[float32]
                d x 1 weight vector
            x : numpy[float32]
                d x N feature vector
            y : numpy[float32]
                1 x N groundtruth vector
            loss_func : str
                loss type either mean_squared', or 'half_mean_squared'
        Returns:
            numpy[float32] : d x 1 gradients
        '''

        # DONE: Implements the __compute_gradients function
        if loss_func == 'mean_squared':
            # DONE: Implements gradients for mean squared loss

            '''
            Using for-loop

            gradients = np.zeros(x.shape)

            for n in range((x.shape[1])):
                x_n = x[:, n]
                gradients[:, n] = (np.matmul(w.T, x_n) - y[n]) * x_n
            '''

            # Using matrix multiplication
            gradients = (np.matmul(w.T, x) - y) * x
```

```
            # Note: Set keepdims=True to keep the dimension of 1 (otherwise it␣
↪will get squashed by mean operation)
            return 2.0 * np.mean(gradients, axis=1, keepdims=True)
        elif loss_func == 'half_mean_squared':
            # DONE: Implements gradients for half mean squared loss

            gradients = (np.matmul(w.T, x) - y) * x

            return np.mean(gradients, axis=1, keepdims=True)
        else:
            raise ValueError('Unsupported loss function: {}'.format(loss_func))

    def update(self, w, x, y, alpha, loss_func):
        '''
        Updates the weight vector based on mean squared or half mean squared␣
↪loss

        Arg(s):
            w : numpy[float32]
                d x 1 weight vector
            x : numpy[float32]
                d x N feature vector
            y : numpy[float32]
                1 x N groundtruth vector
            alpha : float
                learning rate
            loss_func : str
                loss type either 'mean_squared', or 'half_mean_squared'
        Returns:
            numpy[float32] : d x 1 weights
        '''

        # DONE: Implement the optimizer update function

        return w - alpha * self.__compute_gradients(w, x, y, loss_func)
```

Implementation of Linear Regression with Gradient Descent optimizer

```
[3]: class LinearRegressionGradientDescent(object):

    def __init__(self):
        # Define private variables
        self.__weights = None
        self.__optimizer = GradientDescentOptimizer()

    def fit(self, x, y, T, alpha, loss_func='mean_squared'):
```

```
        '''
        Fits the model to x and y by updating the weight vector
        using gradient descent

        Arg(s):
            x : numpy[float32]
                d x N feature vector
            y : numpy[float32]
                1 x N groundtruth vector
            T : int
                number of iterations to train
            alpha : float
                learning rate
            loss_func : str
                loss function to use
        '''

        # DONE: Implement the fit function
        self.__weights = np.zeros([x.shape[0], 1])

        for t in range(1, T + 1):

            # DONE: Compute loss function
            loss = self.__compute_loss(
                x=x,
                y=y,
                loss_func=loss_func)

            if (t % 10000) == 0:
                print('Step={}  Loss={:.4f}'.format(t, loss))

            # DONE: Update weights
            self.__weights = self.__optimizer.update(
                w=self.__weights,
                x=x,
                y=y,
                alpha=alpha,
                loss_func=loss_func)

    def predict(self, x):
        '''
        Predicts the label for each feature vector x

        Arg(s):
            x : numpy[float32]
                d x N feature vector
        Returns:
```

```
            numpy[float32] : 1 x N vector
        '''

        # DONE: Implements the predict function

        return np.matmul(self.__weights.T, x)

    def __compute_loss(self, x, y, loss_func):
        '''
        Returns the gradient of the mean squared or half mean squared loss

        Arg(s):
            x : numpy[float32]
                d x N feature vector
            y : numpy[float32]
                1 x N groundtruth vector
            loss_func : str
                loss type either 'mean_squared', or 'half_mean_squared'
        Returns:
            float : loss
        '''

        # DONE: Implements the __compute_loss function
        predictions = self.predict(x)

        if loss_func == 'mean_squared':
            # DONE: Implements loss for mean squared loss
            loss = np.mean((predictions - y) ** 2)
        elif loss_func == 'half_mean_squared':
            # DONE: Implements loss for half mean squared loss
            loss = 0.50 * np.mean((predictions - y) ** 2)
        else:
            raise ValueError('Unsupported loss function: {}'.format(loss_func))

        return loss
```

Implementing training and validation loop for linear regression

[4]:
```
# Load Diabetes and California housing prices dataset
datasets = [
    skdata.load_diabetes(),
    skdata.fetch_california_housing()
]
dataset_names = [
    'Diabetes',
    'California housing prices'
]
```

```
# Loss functions to minimize
dataset_loss_funcs = [
    ['mean_squared', 'half_mean_squared'],
    ['mean_squared', 'half_mean_squared']
]

# TODO: Select learning rates (alpha) for mean squared and half mean squared␣
↪loss
dataset_alphas = [
    [1e-4, 2e-4],
    [1e-7, 2e-7]
]

# TODO: Select number of steps (T) to train for mean squared and half mean␣
↪squared loss
dataset_Ts = [
    [30000, 30000],
    [2500000, 2500000]
]

for dataset_options in zip(datasets, dataset_names, dataset_loss_funcs,␣
↪dataset_alphas, dataset_Ts):

    dataset, dataset_name, loss_funcs, alphas, Ts = dataset_options

    '''
    Create the training, validation and testing splits
    '''
    x = dataset.data
    y = dataset.target

    # Shuffle the dataset based on sample indices
    shuffled_indices = np.random.permutation(x.shape[0])

    # Choose the first 80% as training set, next 10% as validation and the rest␣
↪as testing
    train_split_idx = int(0.80 * x.shape[0])
    val_split_idx = int(0.90 * x.shape[0])

    train_indices = shuffled_indices[0:train_split_idx]
    val_indices = shuffled_indices[train_split_idx:val_split_idx]
    test_indices = shuffled_indices[val_split_idx:]

    # Select the examples from x and y to construct our training, validation,␣
↪testing sets
    x_train, y_train = x[train_indices, :], y[train_indices]
```

```
    x_val, y_val = x[val_indices, :], y[val_indices]
    x_test, y_test = x[test_indices, :], y[test_indices]

    '''
    Trains and tests Linear Regression model from scikit-learn
    '''
    # DONE: Initialize scikit-learn linear regression model without bias
    model_scikit = LinearRegressionSciKit(fit_intercept=False)

    # DONE: Trains scikit-learn linear regression model
    model_scikit.fit(x_train, y_train)


    print('***** Results of scikit-learn linear regression model on {} dataset␣
↪*****'.format(
        dataset_name))

    # DONE: Test model on training set
    predictions_train = model_scikit.predict(x_train)

    score_mse_train = skmetrics.mean_squared_error(y_train, predictions_train)
    print('Training set mean squared error: {:.4f}'.format(score_mse_train))

    score_r2_train = skmetrics.r2_score(y_train, predictions_train)
    print('Training set r-squared scores: {:.4f}'.format(score_r2_train))

    # DONE: Test model on validation set
    predictions_val = model_scikit.predict(x_val)

    score_mse_val = skmetrics.mean_squared_error(y_val, predictions_val)
    print('Validation set mean squared error: {:.4f}'.format(score_mse_val))

    score_r2_val = skmetrics.r2_score(y_val, predictions_val)
    print('Validation set r-squared scores: {:.4f}'.format(score_r2_val))

    # DONE: Test model on testing set
    predictions_test = model_scikit.predict(x_test)

    score_mse_test = skmetrics.mean_squared_error(y_test, predictions_test)
    print('Testing set mean squared error: {:.4f}'.format(score_mse_test))

    score_r2_test = skmetrics.r2_score(y_test, predictions_test)
    print('Testing set r-squared scores: {:.4f}'.format(score_r2_test))

    '''
    Trains and tests our linear regression model using different solvers
    '''
```

```python
    # Take the transpose of the dataset to match the dimensions discussed in␣
↪lecture
    # i.e., (N x d) to (d x N)
    x_train = np.transpose(x_train, axes=(1, 0))
    x_val = np.transpose(x_val, axes=(1, 0))
    x_test = np.transpose(x_test, axes=(1, 0))
    y_train = np.expand_dims(y_train, axis=0)
    y_val = np.expand_dims(y_val, axis=0)
    y_test = np.expand_dims(y_test, axis=0)

    for loss_func, alpha, T in zip(loss_funcs, alphas, Ts):

        # DONE: Initialize our linear regression model
        model_ours = LinearRegressionGradientDescent()

        print('***** Results of our linear regression model trained with {}␣
↪loss, alpha={} and T={} on {} dataset *****'.format(
            loss_func, alpha, T, dataset_name))

        # DONE: Train model on training set
        model_ours.fit(
            x=x_train,
            y=y_train,
            T=T,
            alpha=alpha,
            loss_func=loss_func)

        # DONE: Test model on training set using mean squared error and␣
↪r-squared
        predictions_train = model_ours.predict(x_train)

        # Squeeze to remove extra dimensions before passing to r2_score
        y_train_squeezed = np.squeeze(y_train)
        predictions_train_squeezed = np.squeeze(predictions_train)

        score_mse_train = skmetrics.mean_squared_error(y_train_squeezed,␣
↪predictions_train_squeezed)
        print('Training set mean squared error: {:.4f}'.format(score_mse_train))

        score_r2_train = skmetrics.r2_score(y_train_squeezed,␣
↪predictions_train_squeezed)
        print('Training set r-squared scores: {:.4f}'.format(score_r2_train))

        # DONE: Test model on validation set using mean squared error and␣
↪r-squared
```

```
        predictions_val = model_ours.predict(x_val)

        # Squeeze to remove extra dimensions
        y_val_squeezed = np.squeeze(y_val)
        predictions_val_squeezed = np.squeeze(predictions_val)

        score_mse_val = skmetrics.mean_squared_error(y_val_squeezed,␣
↪predictions_val_squeezed)
        print('Validation set mean squared error: {:.4f}'.format(score_mse_val))

        score_r2_val = skmetrics.r2_score(y_val_squeezed,␣
↪predictions_val_squeezed)
        print('Validation set r-squared scores: {:.4f}'.format(score_r2_val))

        # DONE: Test model on testing set using mean squared error and r-squared
        predictions_test = model_ours.predict(x_test)

        # Squeeze to remove extra dimensions
        y_test_squeezed = np.squeeze(y_test)
        predictions_test_squeezed = np.squeeze(predictions_test)

        score_mse_test = skmetrics.mean_squared_error(y_test_squeezed,␣
↪predictions_test_squeezed)
        print('Testing set mean squared error: {:.4f}'.format(score_mse_test))

        score_r2_test = skmetrics.r2_score(y_test_squeezed,␣
↪predictions_test_squeezed)
        print('Testing set r-squared scores: {:.4f}'.format(score_r2_test))
```

```
***** Results of scikit-learn linear regression model on Diabetes dataset *****
Training set mean squared error: 25472.6663
Training set r-squared scores: -3.2907
Validation set mean squared error: 30376.7403
Validation set r-squared scores: -5.1176
Testing set mean squared error: 27024.0792
Testing set r-squared scores: -3.2604
***** Results of our linear regression model trained with mean_squared loss,
alpha=0.0001 and T=30000 on Diabetes dataset *****
Step=10000  Loss=28180.3086
Step=20000  Loss=28106.3236
Step=30000  Loss=28034.8016
Training set mean squared error: 28034.7946
Training set r-squared scores: -3.7223
Validation set mean squared error: 34538.4871
Validation set r-squared scores: -5.9557
Testing set mean squared error: 29713.4975
Testing set r-squared scores: -3.6844
```

```
***** Results of our linear regression model trained with half_mean_squared
loss, alpha=0.0002 and T=30000 on Diabetes dataset *****
Step=10000  Loss=14090.1543
Step=20000  Loss=14053.1618
Step=30000  Loss=14017.4008
Training set mean squared error: 28034.7946
Training set r-squared scores: -3.7223
Validation set mean squared error: 34538.4871
Validation set r-squared scores: -5.9557
Testing set mean squared error: 29713.4975
Testing set r-squared scores: -3.6844
***** Results of scikit-learn linear regression model on California housing
prices dataset *****
Training set mean squared error: 0.5989
Training set r-squared scores: 0.5463
Validation set mean squared error: 0.5922
Validation set r-squared scores: 0.5721
Testing set mean squared error: 0.6590
Testing set r-squared scores: 0.5185
***** Results of our linear regression model trained with mean_squared loss,
alpha=1e-07 and T=2500000 on California housing prices dataset *****
Step=10000  Loss=1.2972
Step=20000  Loss=1.2855
Step=30000  Loss=1.2744
Step=40000  Loss=1.2638
Step=50000  Loss=1.2536
Step=60000  Loss=1.2437
Step=70000  Loss=1.2340
Step=80000  Loss=1.2245
Step=90000  Loss=1.2151
Step=100000  Loss=1.2060
Step=110000  Loss=1.1970
Step=120000  Loss=1.1882
Step=130000  Loss=1.1796
Step=140000  Loss=1.1711
Step=150000  Loss=1.1628
Step=160000  Loss=1.1546
Step=170000  Loss=1.1465
Step=180000  Loss=1.1386
Step=190000  Loss=1.1308
Step=200000  Loss=1.1232
Step=210000  Loss=1.1157
Step=220000  Loss=1.1083
Step=230000  Loss=1.1010
Step=240000  Loss=1.0939
Step=250000  Loss=1.0869
Step=260000  Loss=1.0800
Step=270000  Loss=1.0732
```

```
Step=280000  Loss=1.0666
Step=290000  Loss=1.0600
Step=300000  Loss=1.0536
Step=310000  Loss=1.0473
Step=320000  Loss=1.0410
Step=330000  Loss=1.0349
Step=340000  Loss=1.0289
Step=350000  Loss=1.0230
Step=360000  Loss=1.0172
Step=370000  Loss=1.0114
Step=380000  Loss=1.0058
Step=390000  Loss=1.0003
Step=400000  Loss=0.9948
Step=410000  Loss=0.9895
Step=420000  Loss=0.9842
Step=430000  Loss=0.9790
Step=440000  Loss=0.9739
Step=450000  Loss=0.9689
Step=460000  Loss=0.9640
Step=470000  Loss=0.9591
Step=480000  Loss=0.9543
Step=490000  Loss=0.9496
Step=500000  Loss=0.9450
Step=510000  Loss=0.9404
Step=520000  Loss=0.9360
Step=530000  Loss=0.9315
Step=540000  Loss=0.9272
Step=550000  Loss=0.9229
Step=560000  Loss=0.9187
Step=570000  Loss=0.9146
Step=580000  Loss=0.9105
Step=590000  Loss=0.9065
Step=600000  Loss=0.9026
Step=610000  Loss=0.8987
Step=620000  Loss=0.8948
Step=630000  Loss=0.8911
Step=640000  Loss=0.8874
Step=650000  Loss=0.8837
Step=660000  Loss=0.8801
Step=670000  Loss=0.8766
Step=680000  Loss=0.8731
Step=690000  Loss=0.8697
Step=700000  Loss=0.8663
Step=710000  Loss=0.8630
Step=720000  Loss=0.8597
Step=730000  Loss=0.8564
Step=740000  Loss=0.8533
Step=750000  Loss=0.8501
```

```
Step=760000  Loss=0.8471
Step=770000  Loss=0.8440
Step=780000  Loss=0.8410
Step=790000  Loss=0.8381
Step=800000  Loss=0.8352
Step=810000  Loss=0.8323
Step=820000  Loss=0.8295
Step=830000  Loss=0.8267
Step=840000  Loss=0.8240
Step=850000  Loss=0.8213
Step=860000  Loss=0.8186
Step=870000  Loss=0.8160
Step=880000  Loss=0.8135
Step=890000  Loss=0.8109
Step=900000  Loss=0.8084
Step=910000  Loss=0.8059
Step=920000  Loss=0.8035
Step=930000  Loss=0.8011
Step=940000  Loss=0.7988
Step=950000  Loss=0.7964
Step=960000  Loss=0.7942
Step=970000  Loss=0.7919
Step=980000  Loss=0.7897
Step=990000  Loss=0.7875
Step=1000000  Loss=0.7853
Step=1010000  Loss=0.7832
Step=1020000  Loss=0.7811
Step=1030000  Loss=0.7790
Step=1040000  Loss=0.7770
Step=1050000  Loss=0.7750
Step=1060000  Loss=0.7730
Step=1070000  Loss=0.7710
Step=1080000  Loss=0.7691
Step=1090000  Loss=0.7672
Step=1100000  Loss=0.7654
Step=1110000  Loss=0.7635
Step=1120000  Loss=0.7617
Step=1130000  Loss=0.7599
Step=1140000  Loss=0.7581
Step=1150000  Loss=0.7564
Step=1160000  Loss=0.7547
Step=1170000  Loss=0.7530
Step=1180000  Loss=0.7513
Step=1190000  Loss=0.7497
Step=1200000  Loss=0.7480
Step=1210000  Loss=0.7464
Step=1220000  Loss=0.7449
Step=1230000  Loss=0.7433
```

```
Step=1240000  Loss=0.7418
Step=1250000  Loss=0.7403
Step=1260000  Loss=0.7388
Step=1270000  Loss=0.7373
Step=1280000  Loss=0.7359
Step=1290000  Loss=0.7344
Step=1300000  Loss=0.7330
Step=1310000  Loss=0.7316
Step=1320000  Loss=0.7302
Step=1330000  Loss=0.7289
Step=1340000  Loss=0.7276
Step=1350000  Loss=0.7262
Step=1360000  Loss=0.7249
Step=1370000  Loss=0.7237
Step=1380000  Loss=0.7224
Step=1390000  Loss=0.7212
Step=1400000  Loss=0.7199
Step=1410000  Loss=0.7187
Step=1420000  Loss=0.7175
Step=1430000  Loss=0.7163
Step=1440000  Loss=0.7152
Step=1450000  Loss=0.7140
Step=1460000  Loss=0.7129
Step=1470000  Loss=0.7118
Step=1480000  Loss=0.7107
Step=1490000  Loss=0.7096
Step=1500000  Loss=0.7085
Step=1510000  Loss=0.7075
Step=1520000  Loss=0.7064
Step=1530000  Loss=0.7054
Step=1540000  Loss=0.7044
Step=1550000  Loss=0.7034
Step=1560000  Loss=0.7024
Step=1570000  Loss=0.7014
Step=1580000  Loss=0.7005
Step=1590000  Loss=0.6995
Step=1600000  Loss=0.6986
Step=1610000  Loss=0.6976
Step=1620000  Loss=0.6967
Step=1630000  Loss=0.6958
Step=1640000  Loss=0.6949
Step=1650000  Loss=0.6941
Step=1660000  Loss=0.6932
Step=1670000  Loss=0.6924
Step=1680000  Loss=0.6915
Step=1690000  Loss=0.6907
Step=1700000  Loss=0.6899
Step=1710000  Loss=0.6891
```

```
Step=1720000  Loss=0.6883
Step=1730000  Loss=0.6875
Step=1740000  Loss=0.6867
Step=1750000  Loss=0.6859
Step=1760000  Loss=0.6852
Step=1770000  Loss=0.6844
Step=1780000  Loss=0.6837
Step=1790000  Loss=0.6830
Step=1800000  Loss=0.6823
Step=1810000  Loss=0.6815
Step=1820000  Loss=0.6809
Step=1830000  Loss=0.6802
Step=1840000  Loss=0.6795
Step=1850000  Loss=0.6788
Step=1860000  Loss=0.6781
Step=1870000  Loss=0.6775
Step=1880000  Loss=0.6769
Step=1890000  Loss=0.6762
Step=1900000  Loss=0.6756
Step=1910000  Loss=0.6750
Step=1920000  Loss=0.6744
Step=1930000  Loss=0.6738
Step=1940000  Loss=0.6732
Step=1950000  Loss=0.6726
Step=1960000  Loss=0.6720
Step=1970000  Loss=0.6714
Step=1980000  Loss=0.6708
Step=1990000  Loss=0.6703
Step=2000000  Loss=0.6697
Step=2010000  Loss=0.6692
Step=2020000  Loss=0.6687
Step=2030000  Loss=0.6681
Step=2040000  Loss=0.6676
Step=2050000  Loss=0.6671
Step=2060000  Loss=0.6666
Step=2070000  Loss=0.6661
Step=2080000  Loss=0.6656
Step=2090000  Loss=0.6651
Step=2100000  Loss=0.6646
Step=2110000  Loss=0.6641
Step=2120000  Loss=0.6637
Step=2130000  Loss=0.6632
Step=2140000  Loss=0.6627
Step=2150000  Loss=0.6623
Step=2160000  Loss=0.6618
Step=2170000  Loss=0.6614
Step=2180000  Loss=0.6610
Step=2190000  Loss=0.6605
```

```
Step=2200000  Loss=0.6601
Step=2210000  Loss=0.6597
Step=2220000  Loss=0.6593
Step=2230000  Loss=0.6589
Step=2240000  Loss=0.6585
Step=2250000  Loss=0.6581
Step=2260000  Loss=0.6577
Step=2270000  Loss=0.6573
Step=2280000  Loss=0.6569
Step=2290000  Loss=0.6565
Step=2300000  Loss=0.6561
Step=2310000  Loss=0.6558
Step=2320000  Loss=0.6554
Step=2330000  Loss=0.6550
Step=2340000  Loss=0.6547
Step=2350000  Loss=0.6543
Step=2360000  Loss=0.6540
Step=2370000  Loss=0.6536
Step=2380000  Loss=0.6533
Step=2390000  Loss=0.6530
Step=2400000  Loss=0.6526
Step=2410000  Loss=0.6523
Step=2420000  Loss=0.6520
Step=2430000  Loss=0.6517
Step=2440000  Loss=0.6514
Step=2450000  Loss=0.6511
Step=2460000  Loss=0.6508
Step=2470000  Loss=0.6505
Step=2480000  Loss=0.6502
Step=2490000  Loss=0.6499
Step=2500000  Loss=0.6496
Training set mean squared error: 0.6496
Training set r-squared scores: 0.5080
Validation set mean squared error: 0.6348
Validation set r-squared scores: 0.5413
Testing set mean squared error: 0.7028
Testing set r-squared scores: 0.4865
***** Results of our linear regression model trained with half_mean_squared
loss, alpha=2e-07 and T=2500000 on California housing prices dataset *****
Step=10000  Loss=0.6486
Step=20000  Loss=0.6427
Step=30000  Loss=0.6372
Step=40000  Loss=0.6319
Step=50000  Loss=0.6268
Step=60000  Loss=0.6218
Step=70000  Loss=0.6170
Step=80000  Loss=0.6122
Step=90000  Loss=0.6076
```

```
Step=100000  Loss=0.6030
Step=110000  Loss=0.5985
Step=120000  Loss=0.5941
Step=130000  Loss=0.5898
Step=140000  Loss=0.5855
Step=150000  Loss=0.5814
Step=160000  Loss=0.5773
Step=170000  Loss=0.5733
Step=180000  Loss=0.5693
Step=190000  Loss=0.5654
Step=200000  Loss=0.5616
Step=210000  Loss=0.5578
Step=220000  Loss=0.5541
Step=230000  Loss=0.5505
Step=240000  Loss=0.5469
Step=250000  Loss=0.5434
Step=260000  Loss=0.5400
Step=270000  Loss=0.5366
Step=280000  Loss=0.5333
Step=290000  Loss=0.5300
Step=300000  Loss=0.5268
Step=310000  Loss=0.5236
Step=320000  Loss=0.5205
Step=330000  Loss=0.5175
Step=340000  Loss=0.5144
Step=350000  Loss=0.5115
Step=360000  Loss=0.5086
Step=370000  Loss=0.5057
Step=380000  Loss=0.5029
Step=390000  Loss=0.5001
Step=400000  Loss=0.4974
Step=410000  Loss=0.4947
Step=420000  Loss=0.4921
Step=430000  Loss=0.4895
Step=440000  Loss=0.4870
Step=450000  Loss=0.4844
Step=460000  Loss=0.4820
Step=470000  Loss=0.4795
Step=480000  Loss=0.4772
Step=490000  Loss=0.4748
Step=500000  Loss=0.4725
Step=510000  Loss=0.4702
Step=520000  Loss=0.4680
Step=530000  Loss=0.4658
Step=540000  Loss=0.4636
Step=550000  Loss=0.4615
Step=560000  Loss=0.4594
Step=570000  Loss=0.4573
```

```
Step=580000  Loss=0.4553
Step=590000  Loss=0.4532
Step=600000  Loss=0.4513
Step=610000  Loss=0.4493
Step=620000  Loss=0.4474
Step=630000  Loss=0.4455
Step=640000  Loss=0.4437
Step=650000  Loss=0.4419
Step=660000  Loss=0.4401
Step=670000  Loss=0.4383
Step=680000  Loss=0.4365
Step=690000  Loss=0.4348
Step=700000  Loss=0.4331
Step=710000  Loss=0.4315
Step=720000  Loss=0.4298
Step=730000  Loss=0.4282
Step=740000  Loss=0.4266
Step=750000  Loss=0.4251
Step=760000  Loss=0.4235
Step=770000  Loss=0.4220
Step=780000  Loss=0.4205
Step=790000  Loss=0.4190
Step=800000  Loss=0.4176
Step=810000  Loss=0.4162
Step=820000  Loss=0.4147
Step=830000  Loss=0.4134
Step=840000  Loss=0.4120
Step=850000  Loss=0.4106
Step=860000  Loss=0.4093
Step=870000  Loss=0.4080
Step=880000  Loss=0.4067
Step=890000  Loss=0.4055
Step=900000  Loss=0.4042
Step=910000  Loss=0.4030
Step=920000  Loss=0.4018
Step=930000  Loss=0.4006
Step=940000  Loss=0.3994
Step=950000  Loss=0.3982
Step=960000  Loss=0.3971
Step=970000  Loss=0.3959
Step=980000  Loss=0.3948
Step=990000  Loss=0.3937
Step=1000000  Loss=0.3927
Step=1010000  Loss=0.3916
Step=1020000  Loss=0.3905
Step=1030000  Loss=0.3895
Step=1040000  Loss=0.3885
Step=1050000  Loss=0.3875
```

```
Step=1060000  Loss=0.3865
Step=1070000  Loss=0.3855
Step=1080000  Loss=0.3846
Step=1090000  Loss=0.3836
Step=1100000  Loss=0.3827
Step=1110000  Loss=0.3818
Step=1120000  Loss=0.3808
Step=1130000  Loss=0.3800
Step=1140000  Loss=0.3791
Step=1150000  Loss=0.3782
Step=1160000  Loss=0.3773
Step=1170000  Loss=0.3765
Step=1180000  Loss=0.3757
Step=1190000  Loss=0.3748
Step=1200000  Loss=0.3740
Step=1210000  Loss=0.3732
Step=1220000  Loss=0.3724
Step=1230000  Loss=0.3717
Step=1240000  Loss=0.3709
Step=1250000  Loss=0.3701
Step=1260000  Loss=0.3694
Step=1270000  Loss=0.3687
Step=1280000  Loss=0.3679
Step=1290000  Loss=0.3672
Step=1300000  Loss=0.3665
Step=1310000  Loss=0.3658
Step=1320000  Loss=0.3651
Step=1330000  Loss=0.3644
Step=1340000  Loss=0.3638
Step=1350000  Loss=0.3631
Step=1360000  Loss=0.3625
Step=1370000  Loss=0.3618
Step=1380000  Loss=0.3612
Step=1390000  Loss=0.3606
Step=1400000  Loss=0.3600
Step=1410000  Loss=0.3594
Step=1420000  Loss=0.3588
Step=1430000  Loss=0.3582
Step=1440000  Loss=0.3576
Step=1450000  Loss=0.3570
Step=1460000  Loss=0.3564
Step=1470000  Loss=0.3559
Step=1480000  Loss=0.3553
Step=1490000  Loss=0.3548
Step=1500000  Loss=0.3543
Step=1510000  Loss=0.3537
Step=1520000  Loss=0.3532
Step=1530000  Loss=0.3527
```

```
Step=1540000  Loss=0.3522
Step=1550000  Loss=0.3517
Step=1560000  Loss=0.3512
Step=1570000  Loss=0.3507
Step=1580000  Loss=0.3502
Step=1590000  Loss=0.3498
Step=1600000  Loss=0.3493
Step=1610000  Loss=0.3488
Step=1620000  Loss=0.3484
Step=1630000  Loss=0.3479
Step=1640000  Loss=0.3475
Step=1650000  Loss=0.3470
Step=1660000  Loss=0.3466
Step=1670000  Loss=0.3462
Step=1680000  Loss=0.3458
Step=1690000  Loss=0.3453
Step=1700000  Loss=0.3449
Step=1710000  Loss=0.3445
Step=1720000  Loss=0.3441
Step=1730000  Loss=0.3437
Step=1740000  Loss=0.3434
Step=1750000  Loss=0.3430
Step=1760000  Loss=0.3426
Step=1770000  Loss=0.3422
Step=1780000  Loss=0.3418
Step=1790000  Loss=0.3415
Step=1800000  Loss=0.3411
Step=1810000  Loss=0.3408
Step=1820000  Loss=0.3404
Step=1830000  Loss=0.3401
Step=1840000  Loss=0.3397
Step=1850000  Loss=0.3394
Step=1860000  Loss=0.3391
Step=1870000  Loss=0.3387
Step=1880000  Loss=0.3384
Step=1890000  Loss=0.3381
Step=1900000  Loss=0.3378
Step=1910000  Loss=0.3375
Step=1920000  Loss=0.3372
Step=1930000  Loss=0.3369
Step=1940000  Loss=0.3366
Step=1950000  Loss=0.3363
Step=1960000  Loss=0.3360
Step=1970000  Loss=0.3357
Step=1980000  Loss=0.3354
Step=1990000  Loss=0.3351
Step=2000000  Loss=0.3349
Step=2010000  Loss=0.3346
```

```
Step=2020000  Loss=0.3343
Step=2030000  Loss=0.3341
Step=2040000  Loss=0.3338
Step=2050000  Loss=0.3335
Step=2060000  Loss=0.3333
Step=2070000  Loss=0.3330
Step=2080000  Loss=0.3328
Step=2090000  Loss=0.3325
Step=2100000  Loss=0.3323
Step=2110000  Loss=0.3321
Step=2120000  Loss=0.3318
Step=2130000  Loss=0.3316
Step=2140000  Loss=0.3314
Step=2150000  Loss=0.3311
Step=2160000  Loss=0.3309
Step=2170000  Loss=0.3307
Step=2180000  Loss=0.3305
Step=2190000  Loss=0.3303
Step=2200000  Loss=0.3300
Step=2210000  Loss=0.3298
Step=2220000  Loss=0.3296
Step=2230000  Loss=0.3294
Step=2240000  Loss=0.3292
Step=2250000  Loss=0.3290
Step=2260000  Loss=0.3288
Step=2270000  Loss=0.3286
Step=2280000  Loss=0.3284
Step=2290000  Loss=0.3283
Step=2300000  Loss=0.3281
Step=2310000  Loss=0.3279
Step=2320000  Loss=0.3277
Step=2330000  Loss=0.3275
Step=2340000  Loss=0.3273
Step=2350000  Loss=0.3272
Step=2360000  Loss=0.3270
Step=2370000  Loss=0.3268
Step=2380000  Loss=0.3267
Step=2390000  Loss=0.3265
Step=2400000  Loss=0.3263
Step=2410000  Loss=0.3262
Step=2420000  Loss=0.3260
Step=2430000  Loss=0.3258
Step=2440000  Loss=0.3257
Step=2450000  Loss=0.3255
Step=2460000  Loss=0.3254
Step=2470000  Loss=0.3252
Step=2480000  Loss=0.3251
Step=2490000  Loss=0.3249
```

```
Step=2500000  Loss=0.3248
Training set mean squared error: 0.6496
Training set r-squared scores: 0.5080
Validation set mean squared error: 0.6348
Validation set r-squared scores: 0.5413
Testing set mean squared error: 0.7028
Testing set r-squared scores: 0.4865
```

[ ]: