Exercise 4: Gradient Descent for Linear Regression

CPSC 381/581: Machine Learning

Yale University

Instructor: Alex Wong

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_gradient_descent.ipynb
```

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 half_mean_squared losses on Diabetes dataset
should be no more than 15% worse the mean squared error scores
achieved by sci-kit learn's linear regression model across training,
validation and testing splits. Your mean squared error scores on
California housing price dataset should be no more than 20% worse.
```

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

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

```
In [ ]: 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
                '''

                # TODO: Implements the _compute_gradients function
                if loss_func == 'mean_squared':
                    gradients = (np.matmul(w.T, x) - y) * x

                    return 2.0 * np.mean(gradients, axis=1, keepdims=True)
                elif loss_func == 'half_mean_squared':
                    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
    '''

    # TODO: Implement the optimizer update function

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

Implementation of Linear Regression with Gradient Descent optimizer

In [ ]:
```
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
        '''

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

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

            # TODO: 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))

            # TODO: 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
        '''

        # TODO: 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
        '''

        # TODO: Implements the _compute_loss function
        predictions = self.predict(x)

        if loss_func == 'mean_squared':
            # TODO: Implements loss for mean squared loss
            loss = np.mean((predictions - y) ** 2)
        elif loss_func == 'half_mean_squared':
            # TODO: 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

In [ ]:

```
# 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 = [
    [1, 1],
    [1e-7, 2.5e-7]
]

# TODO: Select number of steps (T) to train for mean squared and half mean squared lo
dataset_Ts = [
```

```
    [100000, 100000],
    [2000000, 100000]
]

for dataset_options in zip(datasets, dataset_names, dataset_loss_funcs, dataset_alpha

    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 te
    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
    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
    '''
    # TODO: Initialize scikit-learn linear regression model without bias
    model_scikit = LinearRegressionSciKit(fit_intercept=False)

    # TODO: Trains scikit-learn linear regression model

    model_scikit.fit(x_train, y_train)


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

    # TODO: 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))

    # TODO: 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))

    # TODO: 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
    '''

    # 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):

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

        print('***** Results of our linear regression model trained with {} loss, alp
            loss_func, alpha, T, dataset_name))

        # TODO: Train model on training set
        model_ours.fit(x_train, y_train, T, alpha, loss_func)
        # TODO: Make pedictions
        predictions_train = model_ours.predict(x_train)

        # TODO: Test model on training set using mean squared error and r-squared
        score_mse_train = model_ours._compute_loss(x_train, y_train, loss_func)
        print('Training set mean squared error: {:.4f}'.format(score_mse_train))
        score_r2_train = skmetrics.r2_score(np.squeeze(y_train), np.squeeze(predictio
        print('Training set r-squared scores: {:.4f}'.format(score_r2_train))

        # TODO: Test model on validation set using mean squared error and r-squared
        predictions_val = model_ours.predict(x_val)

        score_mse_val = model_ours._compute_loss(x_val, y_val, loss_func)
        print('Validation set mean squared error: {:.4f}'.format(score_mse_val))
        score_r2_val = skmetrics.r2_score(np.squeeze(y_val), np.squeeze(predictions_v

        print('Validation set r-squared scores: {:.4f}'.format(score_r2_val))

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

        score_mse_test = model_ours._compute_loss(x_test, y_test, loss_func)
        print('Testing set mean squared error: {:.4f}'.format(score_mse_test))
        score_r2_test = skmetrics.r2_score(np.squeeze(y_test), np.squeeze(predictions_
        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: 25924.5928
Training set r-squared scores: -3.2354
Validation set mean squared error: 28170.6028
Validation set r-squared scores: -4.0233
Testing set mean squared error: 27257.1553
Testing set r-squared scores: -4.9347
***** Results of our linear regression model trained with mean_squared loss, alpha=1 a
nd T=100000 on Diabetes dataset *****
Step=10000  Loss=25940.3928
Step=20000  Loss=25931.6810
Step=30000  Loss=25927.7733
Step=40000  Loss=25926.0199
Step=50000  Loss=25925.2332
Step=60000  Loss=25924.8801
Step=70000  Loss=25924.7217
Step=80000  Loss=25924.6507
Step=90000  Loss=25924.6188
Step=100000  Loss=25924.6045
Training set mean squared error: 25924.6045
Training set r-squared scores: -3.2354
Validation set mean squared error: 28162.2910
Validation set r-squared scores: -4.0219
Testing set mean squared error: 27261.0160
Testing set r-squared scores: -4.9355
***** Results of our linear regression model trained with half_mean_squared loss, alph
a=1 and T=100000 on Diabetes dataset *****
Step=10000  Loss=12974.1281
Step=20000  Loss=12970.1962
Step=30000  Loss=12967.5872
Step=40000  Loss=12965.8404
Step=50000  Loss=12964.6704
Step=60000  Loss=12963.8866
Step=70000  Loss=12963.3616
Step=80000  Loss=12963.0100
Step=90000  Loss=12962.7744
Step=100000  Loss=12962.6166
Training set mean squared error: 12962.6166
Training set r-squared scores: -3.2355
Validation set mean squared error: 14054.8160
Validation set r-squared scores: -4.0125
Testing set mean squared error: 13643.0420
Testing set r-squared scores: -4.9410
***** Results of scikit-learn linear regression model on California housing prices dat
aset *****
Training set mean squared error: 0.5998
Training set r-squared scores: 0.5481
Validation set mean squared error: 0.6166
Validation set r-squared scores: 0.5598
Testing set mean squared error: 0.6364
Testing set r-squared scores: 0.5082
***** Results of our linear regression model trained with mean_squared loss, alpha=1e-
07 and T=2000000 on California housing prices dataset *****
Step=10000  Loss=1.3036
Step=20000  Loss=1.2917
Step=30000  Loss=1.2805
Step=40000  Loss=1.2698
Step=50000  Loss=1.2594
Step=60000  Loss=1.2494
Step=70000  Loss=1.2396
Step=80000  Loss=1.2299
Step=90000  Loss=1.2205
Step=100000  Loss=1.2113
Step=110000  Loss=1.2022
Step=120000  Loss=1.1933
Step=130000  Loss=1.1846
```

```
Step=140000  Loss=1.1760
Step=150000  Loss=1.1675
Step=160000  Loss=1.1592
Step=170000  Loss=1.1510
Step=180000  Loss=1.1430
Step=190000  Loss=1.1351
Step=200000  Loss=1.1274
Step=210000  Loss=1.1198
Step=220000  Loss=1.1123
Step=230000  Loss=1.1049
Step=240000  Loss=1.0977
Step=250000  Loss=1.0906
Step=260000  Loss=1.0836
Step=270000  Loss=1.0767
Step=280000  Loss=1.0700
Step=290000  Loss=1.0633
Step=300000  Loss=1.0568
Step=310000  Loss=1.0504
Step=320000  Loss=1.0441
Step=330000  Loss=1.0379
Step=340000  Loss=1.0318
Step=350000  Loss=1.0258
Step=360000  Loss=1.0199
Step=370000  Loss=1.0141
Step=380000  Loss=1.0084
Step=390000  Loss=1.0027
Step=400000  Loss=0.9972
Step=410000  Loss=0.9918
Step=420000  Loss=0.9864
Step=430000  Loss=0.9812
Step=440000  Loss=0.9760
Step=450000  Loss=0.9709
Step=460000  Loss=0.9659
Step=470000  Loss=0.9610
Step=480000  Loss=0.9562
Step=490000  Loss=0.9514
Step=500000  Loss=0.9467
Step=510000  Loss=0.9421
Step=520000  Loss=0.9376
Step=530000  Loss=0.9331
Step=540000  Loss=0.9287
Step=550000  Loss=0.9244
Step=560000  Loss=0.9201
Step=570000  Loss=0.9159
Step=580000  Loss=0.9118
Step=590000  Loss=0.9078
Step=600000  Loss=0.9038
Step=610000  Loss=0.8998
Step=620000  Loss=0.8960
Step=630000  Loss=0.8922
Step=640000  Loss=0.8884
Step=650000  Loss=0.8847
Step=660000  Loss=0.8811
Step=670000  Loss=0.8775
Step=680000  Loss=0.8740
Step=690000  Loss=0.8706
Step=700000  Loss=0.8671
Step=710000  Loss=0.8638
Step=720000  Loss=0.8605
Step=730000  Loss=0.8572
Step=740000  Loss=0.8540
Step=750000  Loss=0.8509
Step=760000  Loss=0.8478
Step=770000  Loss=0.8447
Step=780000  Loss=0.8417
Step=790000  Loss=0.8387
```

```
Step=800000  Loss=0.8358
Step=810000  Loss=0.8329
Step=820000  Loss=0.8301
Step=830000  Loss=0.8273
Step=840000  Loss=0.8246
Step=850000  Loss=0.8219
Step=860000  Loss=0.8192
Step=870000  Loss=0.8166
Step=880000  Loss=0.8140
Step=890000  Loss=0.8115
Step=900000  Loss=0.8089
Step=910000  Loss=0.8065
Step=920000  Loss=0.8040
Step=930000  Loss=0.8016
Step=940000  Loss=0.7993
Step=950000  Loss=0.7970
Step=960000  Loss=0.7947
Step=970000  Loss=0.7924
Step=980000  Loss=0.7902
Step=990000  Loss=0.7880
Step=1000000  Loss=0.7858
Step=1010000  Loss=0.7837
Step=1020000  Loss=0.7816
Step=1030000  Loss=0.7795
Step=1040000  Loss=0.7775
Step=1050000  Loss=0.7755
Step=1060000  Loss=0.7735
Step=1070000  Loss=0.7716
Step=1080000  Loss=0.7696
Step=1090000  Loss=0.7677
Step=1100000  Loss=0.7659
Step=1110000  Loss=0.7640
Step=1120000  Loss=0.7622
Step=1130000  Loss=0.7604
Step=1140000  Loss=0.7587
Step=1150000  Loss=0.7569
Step=1160000  Loss=0.7552
Step=1170000  Loss=0.7535
Step=1180000  Loss=0.7519
Step=1190000  Loss=0.7502
Step=1200000  Loss=0.7486
Step=1210000  Loss=0.7470
Step=1220000  Loss=0.7455
Step=1230000  Loss=0.7439
Step=1240000  Loss=0.7424
Step=1250000  Loss=0.7409
Step=1260000  Loss=0.7394
Step=1270000  Loss=0.7379
Step=1280000  Loss=0.7365
Step=1290000  Loss=0.7351
Step=1300000  Loss=0.7337
Step=1310000  Loss=0.7323
Step=1320000  Loss=0.7309
Step=1330000  Loss=0.7296
Step=1340000  Loss=0.7283
Step=1350000  Loss=0.7270
Step=1360000  Loss=0.7257
Step=1370000  Loss=0.7244
Step=1380000  Loss=0.7231
Step=1390000  Loss=0.7219
Step=1400000  Loss=0.7207
Step=1410000  Loss=0.7195
Step=1420000  Loss=0.7183
Step=1430000  Loss=0.7171
Step=1440000  Loss=0.7160
Step=1450000  Loss=0.7148
```

```
Step=1460000  Loss=0.7137
Step=1470000  Loss=0.7126
Step=1480000  Loss=0.7115
Step=1490000  Loss=0.7104
Step=1500000  Loss=0.7094
Step=1510000  Loss=0.7083
Step=1520000  Loss=0.7073
Step=1530000  Loss=0.7063
Step=1540000  Loss=0.7053
Step=1550000  Loss=0.7043
Step=1560000  Loss=0.7033
Step=1570000  Loss=0.7023
Step=1580000  Loss=0.7014
Step=1590000  Loss=0.7005
Step=1600000  Loss=0.6995
Step=1610000  Loss=0.6986
Step=1620000  Loss=0.6977
Step=1630000  Loss=0.6968
Step=1640000  Loss=0.6959
Step=1650000  Loss=0.6951
Step=1660000  Loss=0.6942
Step=1670000  Loss=0.6934
Step=1680000  Loss=0.6925
Step=1690000  Loss=0.6917
Step=1700000  Loss=0.6909
Step=1710000  Loss=0.6901
Step=1720000  Loss=0.6893
Step=1730000  Loss=0.6885
Step=1740000  Loss=0.6878
Step=1750000  Loss=0.6870
Step=1760000  Loss=0.6863
Step=1770000  Loss=0.6855
Step=1780000  Loss=0.6848
Step=1790000  Loss=0.6841
Step=1800000  Loss=0.6834
Step=1810000  Loss=0.6827
Step=1820000  Loss=0.6820
Step=1830000  Loss=0.6813
Step=1840000  Loss=0.6806
Step=1850000  Loss=0.6799
Step=1860000  Loss=0.6793
Step=1870000  Loss=0.6786
Step=1880000  Loss=0.6780
Step=1890000  Loss=0.6774
Step=1900000  Loss=0.6768
Step=1910000  Loss=0.6761
Step=1920000  Loss=0.6755
Step=1930000  Loss=0.6749
Step=1940000  Loss=0.6743
Step=1950000  Loss=0.6738
Step=1960000  Loss=0.6732
Step=1970000  Loss=0.6726
Step=1980000  Loss=0.6720
Step=1990000  Loss=0.6715
Step=2000000  Loss=0.6709
Training set mean squared error: 0.6709
Training set r-squared scores: 0.4946
Validation set mean squared error: 0.6986
Validation set r-squared scores: 0.5013
Testing set mean squared error: 0.6648
Testing set r-squared scores: 0.4863
***** Results of our linear regression model trained with half_mean_squared loss, alph
a=2.5e-07 and T=100000 on California housing prices dataset *****
Step=10000  Loss=0.6503
Step=20000  Loss=0.6430
Step=30000  Loss=0.6362
```

```
Step=40000  Loss=0.6297
Step=50000  Loss=0.6234
Step=60000  Loss=0.6174
Step=70000  Loss=0.6114
Step=80000  Loss=0.6056
Step=90000  Loss=0.6000
Step=100000  Loss=0.5945
Training set mean squared error: 0.5945
Training set r-squared scores: 0.1044
Validation set mean squared error: 0.6310
Validation set r-squared scores: 0.0991
Testing set mean squared error: 0.5798
Testing set r-squared scores: 0.1039
```