NYCU Pattern Recognition, Homework 1

Ivan K. 0860838

PART 1, CODING

Linear Regression Model - Single Feature

1. (0%) Show the learning rate and epoch you choose.
   Learning rate: 0.002
   Epochs: 110000

```
batch_size = x_train.shape[0]

# Tune the parameters
# Refer to slide page 9
lr = 0.002
epochs = 110000

np.random.seed(2023)
linear_reg = LinearRegression()
linear_reg.fit(x_train, y_train, lr=lr, epochs=epochs, batch_size=batch_size)
```

2. (5%) Show the weights and intercepts of your linear model.
   Intercepts: 1382.08676323
   Weights: 380.14868938

```
print("Intercepts: ", linear_reg.weights[0])
print("Weights: ", linear_reg.weights[1])
```

✓ 0.0s

```
Intercepts:  [1382.08676323]
Weights:  [380.14868938]
```

3. (5%) What's your final training loss (MSE)?
   Training loss: 150013100.16474593

```
print('training loss: ', linear_reg.evaluate(x_train, y_train))
```
✓ 0.0s

```
training loss:  150013100.16474593
```

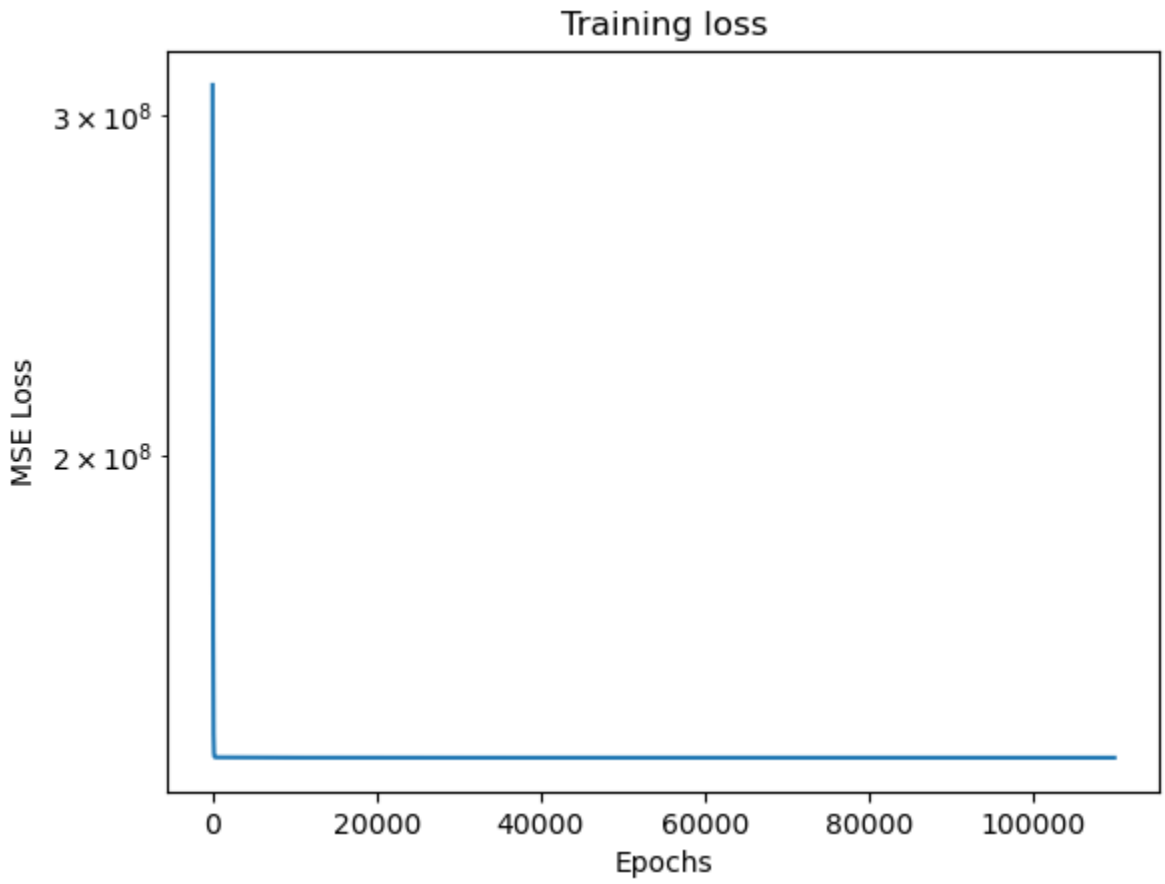
4. (5%) What's the MSE of your validation prediction and validation ground truth?

Validation loss: 152654366.78169277

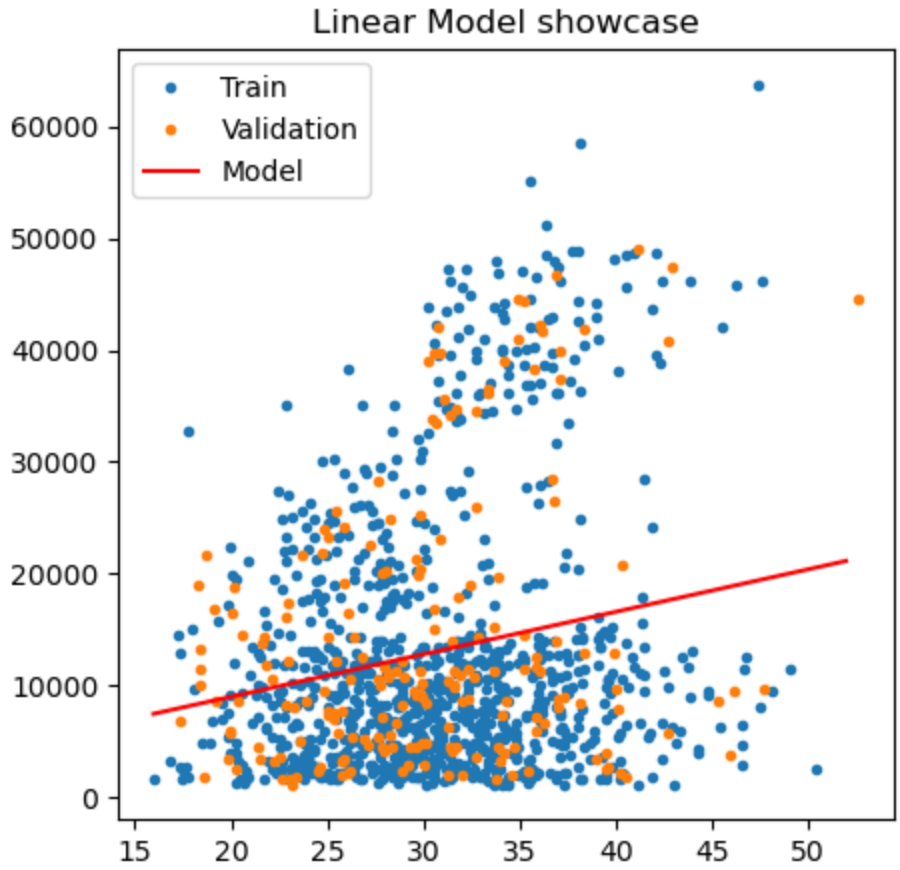
```
print('validation loss: ', linear_reg.evaluate(x_val, y_val))
```
✓ 0.0s

```
validation loss:  152654366.78169277
```

5. (5%) Plot the training curve. (x-axis=epoch, y-axis=loss)



6. (5%) Plot the line (using red) you find with the training data (using blue) and validation data (using orange).



Linear Regression Model - Multiple Features

7. (0%) Show the learning rate and epoch you choose.

    Learning rate: 7e-4

    Epochs: 650000

```
batch_size = x_train.shape[0]

# Tune the parameters
# Refer to slide page 10
lr = 7e-4
epochs = 650000

linear_reg = LinearRegression()
linear_reg.fit(x_train, y_train, lr=lr, epochs=epochs, batch_size=batch_size)
```

8. (10%) Show the weights and intercepts of your linear model.

```
Intercepts:  -11857.0282974
Weights:
 [[  259.85072835]
 [ -383.54711728]
 [  333.33185838]
 [  442.55699935]
 [24032.21979136]
 [ -416.01494405]]
```

```
print("Intercepts: ", linear_reg.weights[0])
print("Weights: \n", linear_reg.weights[1:])
```
✓ 0.0s

```
Intercepts:  [-11857.0282974]
Weights:
 [[  259.85072835]
 [ -383.54711728]
 [  333.33185838]
 [  442.55699935]
 [24032.21979136]
 [ -416.01494405]]
```

9. (5%) What's your final training loss (MSE)?

```
Training loss:  254877566.17408064
```

```
print('training loss: ', linear_reg.evaluate(x_train, y_train))
```
✓ 0.0s

```
training loss:  254877566.17408064
```

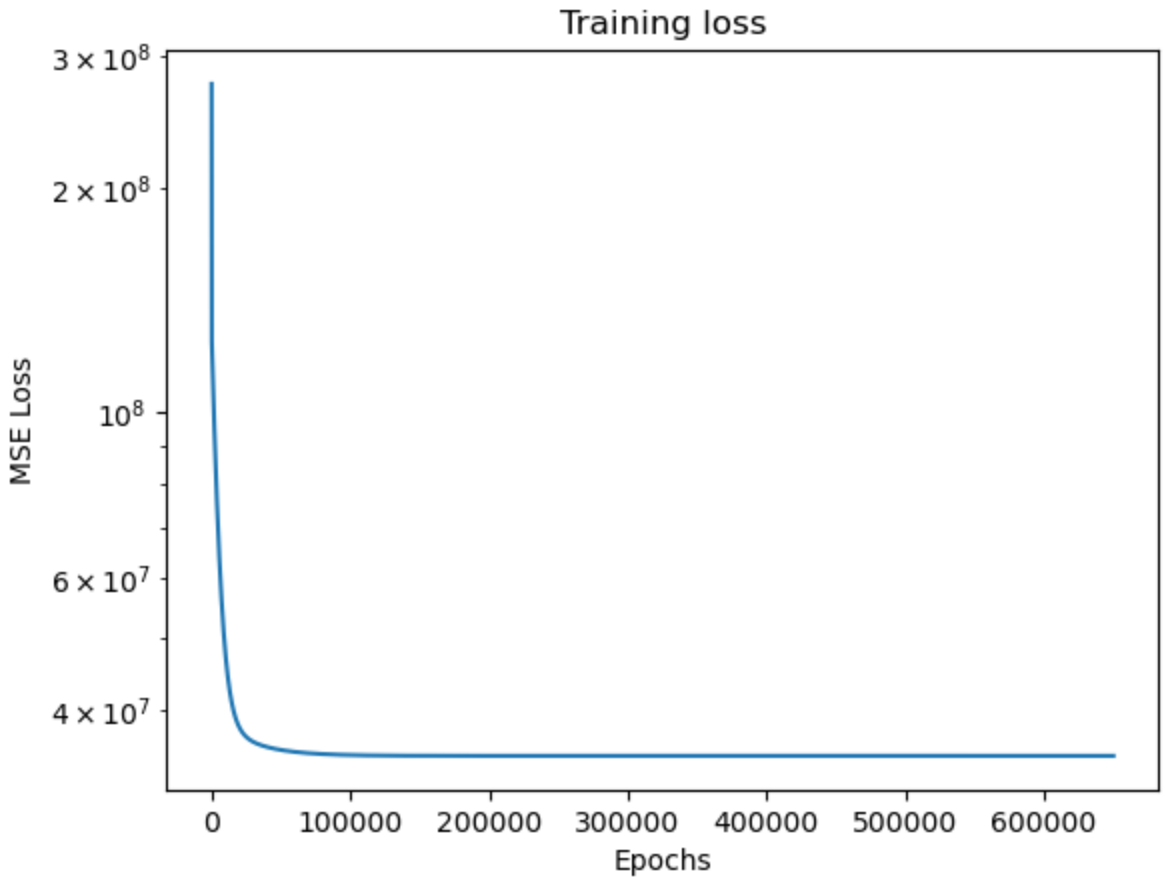
10. (5%) What's the MSE of your validation prediction and validation ground truth?

```
validation loss:  261046250.56485775
```

```
print('validation loss: ', linear_reg.evaluate(x_val, y_val))
```
✓ 0.0s

```
validation loss:  261046250.56485775
```

11. (5%) Plot the training curve. (x-axis=epoch, y-axis=loss)



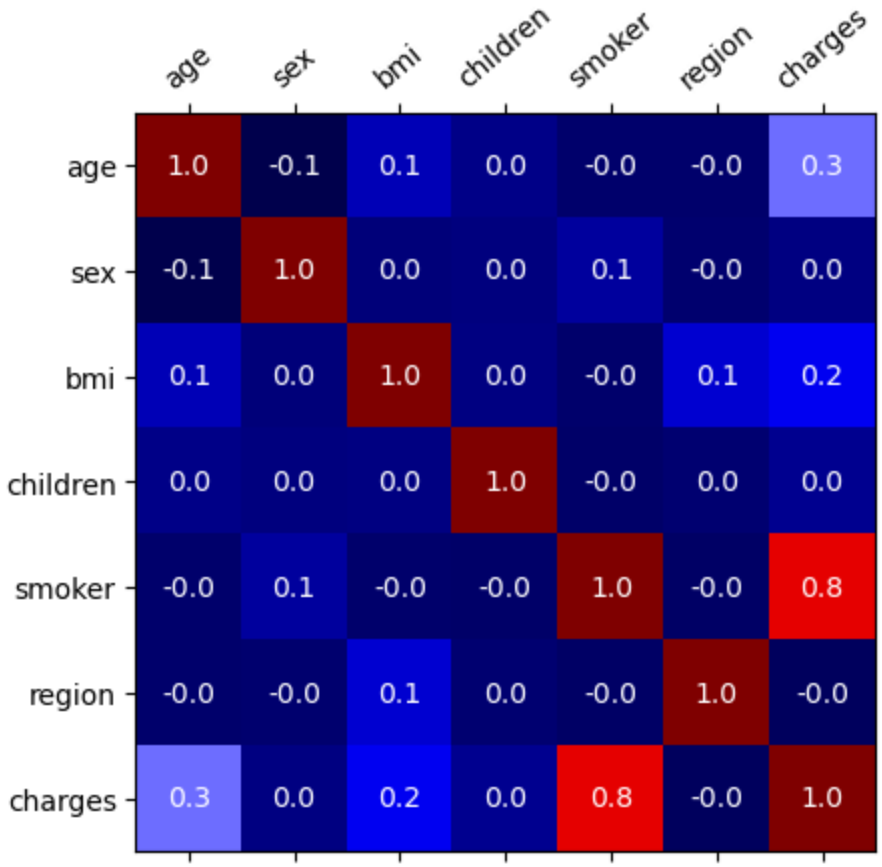
12. (20%) Train your own model and save your final testing predictions in the csv file. Try different learning rates, epochs, batch_size, and do some data analysis to choose the feature you want to use).
learning rate: 5e-8
epoch: 4000000
batch_size: 938 (or `x_train.shape[0]`)
Used features: 9

```
np.random.seed(2023)

lr = 5e-8
epoch = 4000000

linear_model = LinearRegression()

linear_model.fit(x_train, y_train, lr=lr, epochs=epoch, batch_size=x_train.shape[0])
```



Data covariance matrix with respect to predicted features was computed. We can see that features [sex, children, region] have close to 0 correlation to the value we want to predict, so we can drop them.

Next I apply polynomial feature transformation in order to capture interactions between features and extend my feature space. So now I have 3 features I will call them A, B and C, in polynomial combination we will combine them in the next way:

X = [A, B, C, A^2, A*B, B*C, B^2, C*A, C^2]

now we have a 9 dimensional vector that contains features that capture nonlinear relations between data points.

```python
# Compose polynomial features
def polynomial(X):
    X3 = (X[:,0]**2).reshape(-1,1)
    X4 = (X[:,0] * X[:,1]).reshape(-1,1)
    X5 = (X[:,1] * X[:,2]).reshape(-1,1)
    X6 = (X[:,1]**2).reshape(-1,1)
    X7 = (X[:,2] * X[:,0]).reshape(-1,1)
    X8 = (X[:,2]**2).reshape(-1,1)

    X = np.concatenate((X, X3,X4,X5,X6,X7,X8), -1)
    return X

x_train = polynomial(x_train)
x_val = polynomial(x_val)
x_test = polynomial(x_test)
```

Then I just tried to finetune learning rate and epochs until validation and training loss fall below needed threshold. I notes that lower learning rate with higher epoch count leads to better results so I just used that. I didn't noticed any improvement in changing batch size, but in theory I understand that it should help tune noise reduction in gradient signal.

PART 2, QUESTIONS

(7%) 1. What's the difference between Gradient Descent, Mini-Batch Gradient Descent, and Stochastic Gradient Descent?

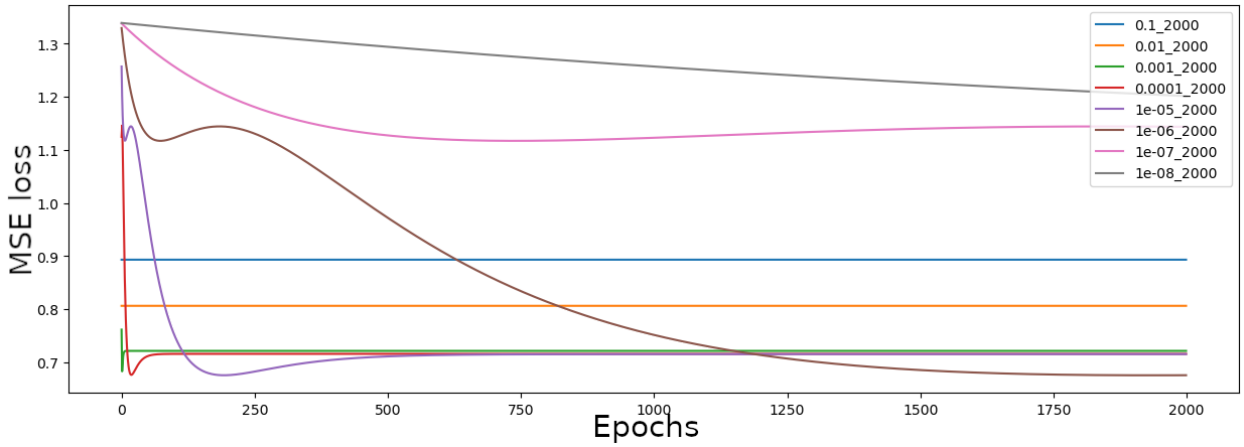
Gradient Descent - optimization algorithm that updates the weights of a model, using derivatives of the error(cost function) with respect to parameters. (Usually way when you use entire dataset called batch Gradient Descent)

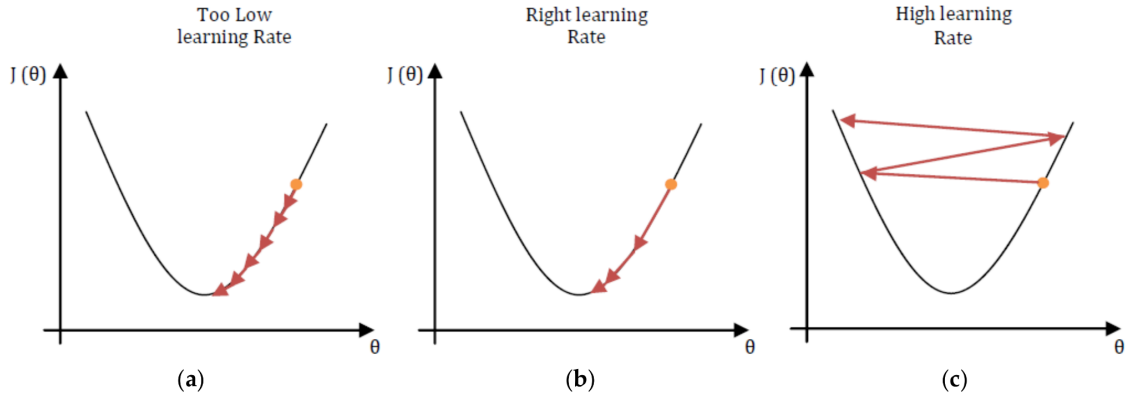
Mini-Batch Gradient Descent - Same algorithm as GD but we have a subset of our dataset as observations to calculate the cost function over and update our weights.

Stochastic Gradient Descent - Same algorithm as GD but we use one datapoint of our dataset as observations to calculate the cost function over and update our weights.

(7%) 2. How do different values of learning rate (too large, too small...) affect the convergence of optimization? Please explain in detail.

As you can see, from the first figure below the learning rate affects the speed of learning and how good our model performs. It's eventually our step in a direction or local/global minima of our optimization manifold.





(8%) 3. Suppose you are given a dataset with two variables, X and Y, and you want to perform linear regression to determine the relationship between these variables. You plot the data and notice that there is a strong nonlinear relationship between X and Y. Can you still use linear regression to analyze this data? Why or why not? Please explain in detail.

You can perform nonlinear data processing, like the kernel method for example, and use linear model. We can project data from lower dimensional space where it's not linearly separable, into higher dimensional space where we can use linear models to separate it there.

(8%) 4. In the coding part of this homework, we can notice that when we use more features in the data, we can usually achieve a lower training loss. Consider two sets of features, A and B, where B is a subset of A. (1) Prove that we can achieve a non-greater training loss when we use the features of set A rather than the features of set B. (2) In what situation will the two training losses be equal?

1) It's about learning best data representation and the amount of relevant entropy that features contain. Set B will have lower relevant entropy (correlation) with predicted data, so its prediction can be only less or equal to prediction from set A. And in our homework we just processed existing information and didn't create new one.
2) When sets are equal, or if entropy of subsets are equal.