NYCU Pattern Recognition, Homework 1

[109550198], [卜銳凱]

Part. 1, Coding (60%):

(10%) Linear Regression Model - Closed-form Solution.

1. (10%) Show the weights and intercepts of your linear model.

```
2024-04-02 03:07:20.996 | INFO     | __main__:main:78 - LR_CF.weights=array([2.8491883 , 1.0188675 , 0.48562739, 0.1937254 ]), LR_CF.intercept=-33.8223
```

(40%) Linear Regression Model - Gradient Descent Solution

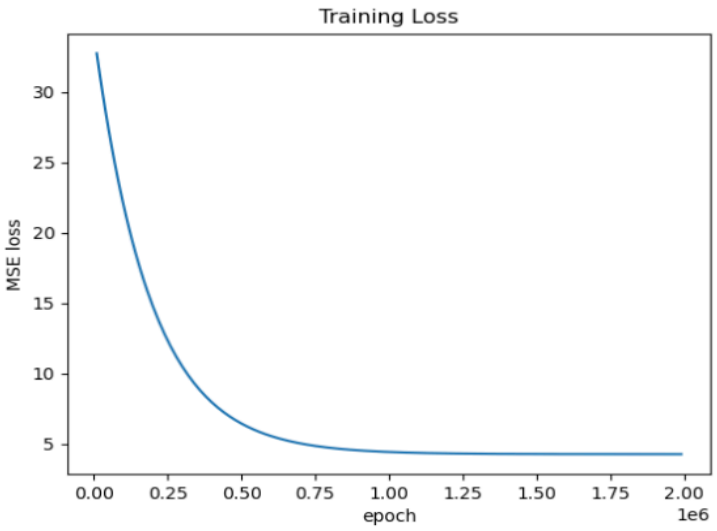
2. (0%) Show the learning rate and epoch (and batch size if you implement mini-batch gradient descent) you choose.

```
LR_GD.fit(train_x, train_y, learning_rate=0.0001, epochs=2000000)
```

3. (10%) Show the weights and intercepts of your linear model.

```
4-02 03:12:13.431 | INFO     | __main__:main:83 - LR_GD.weights=array([2.84575657, 1.01779048, 0.47489125, 0.19126521]), LR_GD.intercept=-33.6441
```

4. (10%) Plot the learning curve. (x-axis=epoch, y-axis=training loss)

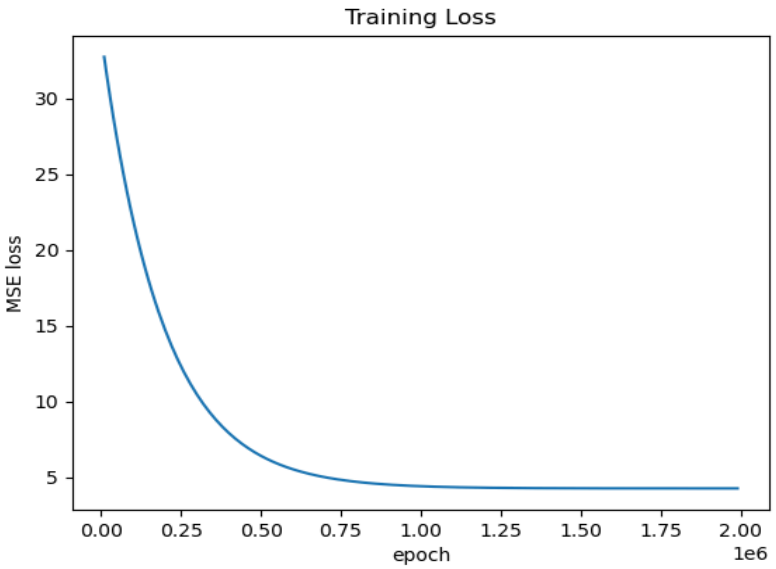


5. (20%) Show your error rate between your closed-form solution and the gradient descent solution.

```
2024-04-02 03:12:13.457 | INFO     | __main__:main:92 - Prediction difference: 47.5794
2024-04-02 03:12:13.458 | INFO     | __main__:main:97 - mse_cf=4.1997, mse_gd=4.1978, Difference: -0.047%
```

6. (Bonus 5 points) Implement the L1 regularization into the gradient descent method and show the weights, intercept, and learning curve.

```
2024-04-03 05:18:47.607 | INFO     | __main__:main:83 - LR_GD.weights=array([2.84870875, 1.01873674, 0.48421907, 0.19341915]), LR_GD.intercept=-33.7996
```


Training Loss
MSE loss
30
25
20
15
10
5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
epoch
1e6

(10%) Code Check and Verification

7. (10%) Lint the code and show the PyTest results.

```
============================= test session starts ==============================
platform darwin -- Python 3.9.13, pytest-7.1.2, pluggy-1.0.0
rootdir: /Users/ralphkedywillensbuteau/Desktop/PR/HW1/release
plugins: anyio-3.5.0
collected 2 items

test_main.py 2024-04-02 03:40:31.974 | INFO     | test_main:test_regression_cf:27 - model.weights=array([[3.]]), model.intercept=array([4.])
.2024-04-02 03:40:32.602 | INFO     | test_main:test_regression_gd:39 - model.weights=array([3.]), model.intercept=3.996349413151626
.

============================== 2 passed in 2.13s ===============================
```

Part. 2, Questions (40%):

1. (10%) Please describe the Vanishing Gradient Problem in detail and provide **at least two solutions** to overcome this problem.

   [Your answer here.]

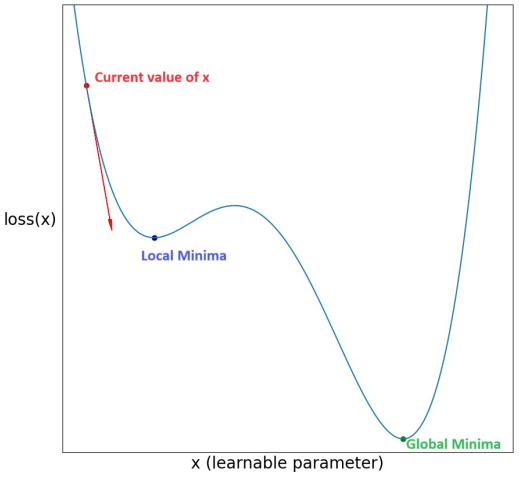
   Vanishing Gradient: when the model is deep, the gradient maybe near 0 which cause the Vanishing Gradient problem.

   two solutions to overcome this problem:

   Initialization Techniques: Proper weight initialization might help to mitigate the vanishing gradient problem. Techniques like Xavier initialization (also known as Glorot initialization) and He initialization set up the weights in such a way that they are better adapted to avoiding vanishing or bursting gradients during training. These strategies are intended to keep gradients at a tolerable scale as they propagate across the network.

   Activation Functions: Choosing appropriate activation functions can help to solve the vanishing gradient problem in neural networks. Activation functions such as ReLU and its derivatives (Leaky ReLU, Parametric ReLU) are less likely to produce vanishing gradients than functions such as sigmoid or tanh. ReLU-based activations allow gradients to flow unmodified for positive values, preventing them from becoming too small during backward propagation and thereby addressing the vanishing gradient issue.

2. (15%) Gradient descent often suffers from the issue of getting stuck at local minima (refer to the figure provided). Please provide **at least two methods** to overcome this problem and discuss how these methods work.



   [Your answer here.]

   Two methods to overcome this problem and discuss how these methods work:

One common method is to employ the stochastic gradient descent (SGD) algorithm. SGD works by randomly sampling a subset of data points at each iteration and updating the model parameters based on the gradient of those data points. This keeps the algorithm from being stuck in a local minimum because it is constantly exploring various parts of the loss landscape.

Another method for avoiding local minima is to employ a momentum algorithm. Momentum is calculated by adding a fraction of the previous gradient to the current gradient. This helps to smooth the algorithm's path and reduces the likelihood of it getting stuck in a local minimum.

3. (15%) What are the basic assumptions of Linear regression between the features and the target? How can techniques help Linear Regression extend beyond these assumptions? Please at least answer one technique.

[Your answer here.]

The basic assumptions of Linear regression between the features and the target:

Linearity: The relationship between the features and the target variable is assumed to be linear. This means that the target variable can be approximated as a linear combination of the features.
Independence: The observations are independent of each other. In other words, the value of one observation does not depend on the values of other observations.
Homoscedasticity: The variance of the errors is constant across all levels of the independent variables. This implies that the spread of the residuals is uniform across the range of predictor variables.
Normality: The errors follow a normal distribution.
No multicollinearity: The independent variables are not highly correlated with each other.

Regularization techniques can enable linear regression to go beyond these assumptions.

- Regularization techniques like Ridge Regression and Lasso Regression can enhance linear regression by imposing penalties on model parameters. These penalties favor simpler models while also mitigating concerns like as multicollinearity and overfitting.

- Ridge Regression involves adding a penalty term to the linear regression cost function that is proportional to the square of the magnitude of the coefficients. This penalty factor helps to reduce the influence of multicollinearity by lowering the coefficients to zero, resulting in more stable estimates.

- Lasso Regression, like Ridge Regression, introduces a penalty term to the linear regression cost function. Lasso, on the other hand, employs the absolute value of the coefficients as the penalty term rather than their square. This causes sparsity in the coefficients, allowing for effective feature selection by setting some coefficients to zero, which can aid in the handling of multicollinearity and model simplification.