NYCU Pattern Recognition, Homework 1

[312552056], [鄭璟翰]

Part. 1, Coding (60%):

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

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

```
2024-04-03 01:19:10.272 | INFO     | __main__:main:87 - LR_CF.weights=array([
2.8491883 , 1.0188675 , 0.48562739, 0.1937254 ]), LR_CF.intercept=-33.8223
```

(40%) Linear Regression Model - Gradient Descent Solution

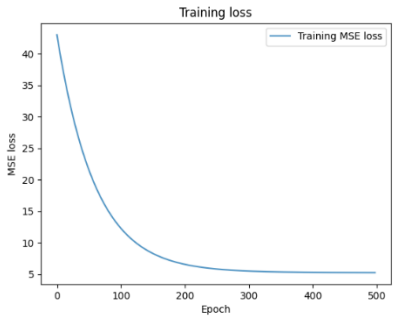
2. (0%) Show the learning rate and epoch (and batch size if you implement mini-batch gradient descent) you choose.

```
37      def fit(self, X, y, learning_rate: float = 1e-4, epochs: int = 500, batch_size: int = 5,
38              L1: bool = False, L1Rate: float = 1e-4):
```

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

```
2024-04-03 01:20:02.493 | INFO     | __main__:main:92 - LR_GD.weights=array([
2.8279515 , 1.0161265 , 0.46783482, 0.18994619]), LR_GD.intercept=-33.3525
```

4. (10%) Plot the learning curve. (x-axis=epoch, y-axis=training loss)
   Because the weights of first epoch is random by variance, not by training, so I don't list the first epochs loss. And same as 6. learning curve.

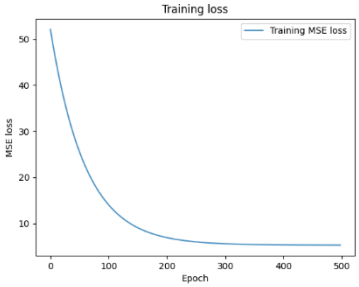


5. (20%) Show your error rate between your closed-form solution and the gradient descent solution.

```
2024-04-03 01:20:02.496 | INFO     | __main__:main:101 - Prediction differenc
e: 142.6457
2024-04-03 01:20:02.497 | INFO     | __main__:main:106 - mse_cf=4.1997, mse_g
d=4.2045. Difference: 0.113%
```

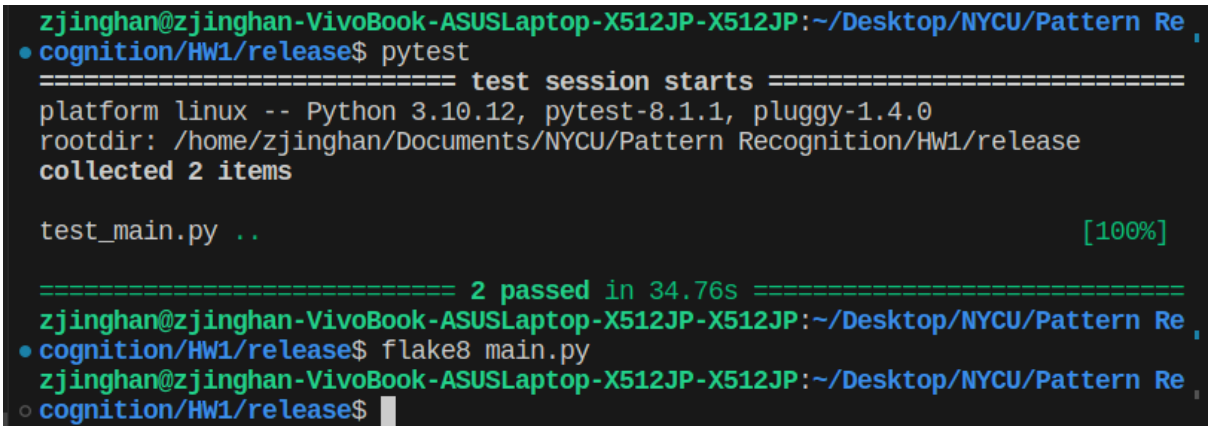
6. (Bonus 5 points) Implement the L1 regularization into the gradient descent method and show the weights, intercept, and learning curve.

```
2024-04-03 01:23:05.770 | INFO     | __main__:main:92 - LR_GD.weights=array([
2.82680549, 1.01579001, 0.46418429, 0.18910397]), LR_GD.intercept=-33.2911
2024-04-03 01:23:05.776 | INFO     | __main__:main:101 - Prediction differenc
e: 158.2817
2024-04-03 01:23:05.779 | INFO     | __main__:main:106 - mse_cf=4.1997, mse_g
d=4.2052. Difference: 0.129%
```



(10%) Code Check and Verification

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



Part. 2, Questions (40%):

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

   Answer:

   The Vanishing Gradient Problem occurs when the gradient, the derivative of the loss function with respect to the weights ($\frac{\partial L}{\partial w}$ in below formula), becomes zero or very close to zero. This results in the weights being updated very little or even not at all. When the weights do not change, the loss computed in the next epoch remains the same, leading to no updates in weights in subsequent epochs.

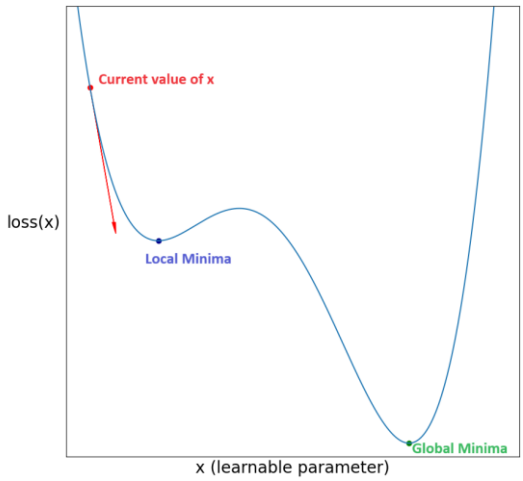
   $$w_{new} = w - \eta \frac{\partial L}{\partial w}$$

   Here are two solutions to avoid the Vanishing Gradient Problem.

   Mini-batch or SGD (Stochastic Gradient Descent): This solution involves dividing the training data into several batches, each containing a batch size of training samples. Unlike traditional Gradient Descent, where weights are updated after processing the entire training data, Mini-batch updates the weights after processing each batch, while SGD updates weights after processing each individual data point (this is batch size equal to 1 in Mini-batch). Because the training data used between two iterations is different, even if weights are not updated in one iteration, the next iteration uses different training data, preventing the loss function from becoming zero.

   Regularization: Another solution is to add a regularization term to the gradient term ($\frac{\partial L}{\partial w}$), such as L1 regularization or L2 regularization. Adding regularization terms helps prevent the gradient term from becoming zero, thereby mitigating the Vanishing Gradient Problem.

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.



Answer:

When processing Gradient Descent, if the result converges to a local minimum, it indicates that under this epoch weights, the gradient of the loss function with respect to w becomes zero, which is the same issue as the Vanishing Gradient Problem mentioned in the previous question. Therefore, the first method here is mini-batch and SGD, described in the previous question. The second method is Momentum, which considers the direction of weight updates. If the update direction is the same as the previous one, it increases the update magnitude; However, if the update direction differs from the previous iteration, it decreases the magnitude of the update. This helps to avoid getting stuck in local minima when oscillating near them.

$$v_t = 0.9v_{t-1} + \eta \frac{\partial L}{\partial w}$$

$$w_{t+1} = w_t - v_t$$

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.

Answer:

The assumptions of linear regression include the following:

1. Each feature should be independent of others features.
2. There should be a linear dependence between each feature and the target.
3. The noise in the target variable should follow independent normal distribution.

In general, the target function is

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + \cdots$$

If we incorporate basis functions into the training function for linear regression

$$y = w_0 + w_1\phi(x)_1 + w_2x\phi(x)_2 + w_3x\phi(x)_3 + \cdots$$

We can try different basis functions, such as Polynomial, Gaussian and Sigmoidal, then there is no need that feature and the target to be linear dependence.