Deep Learning for Image Analysis

DL4IA – Report for Assignment 1

Mujtaba Fadhil Jawad

March 29, 2024

Introduction

In this first assignment, we implement a linear regression model by using gradient descent. First, we compute the necessary equations to enable training a model. Then we proceed by writing the required python code and consequently analyzing the results.

Linear regression with gradient descent

Given the linear regression model:

$$z_i = \sum_{j=1}^{p} w_j x_{ij} + b = W^{\intercal} x_i + b \tag{1}$$

with the cost function

$$J = \frac{1}{n}\sum_{i=1}^{n} L_i \qquad \text{where } L_i = (y_i - z_i)^2 \tag{2}$$

Exercise 1

We need to use the chain rule, it is possible to express the partial derivatives of J with respect to b and w_j in terms of $\frac{\partial J}{\partial z_i}$, $\frac{\partial z_i}{\partial b}$, and $\frac{\partial z_i}{\partial w_j}$

$$\frac{\partial J}{\partial b} = \sum_{i=1}^{n} \frac{\partial J}{\partial z_i}\frac{\partial z_i}{\partial b} = \frac{1}{n}\sum_{i=1}^{n} 2\left(y_i - z_i\right)\cdot 1 = \frac{2}{n}\sum_{i=1}^{n}\left(y_i - z_i\right) \tag{3}$$

$$\frac{\partial J}{\partial w_j} = \sum_{i=1}^{n} \frac{\partial J}{\partial z_i}\frac{\partial z_i}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{n} 2\left(y_i - z_i\right) x_{i,j} = \frac{2}{n}\sum_{i=1}^{n}\left(y_i - z_i\right) x_{i,j} \tag{4}$$

where z_i is given by (1) and x_i,j denotes the j-th component of the input vector x_i.

Exercise 2

Computing the partial derivatives of J with respect to the model parameters:

$$\frac{\partial J}{\partial z_i} = \frac{\partial}{\partial z_i}\left[\frac{1}{n}\sum_{i=1}^{n}\left(y_i - z_i\right)^2\right] = \frac{2}{n}\left(z_i - y_i\right) \tag{5}$$

$$\frac{\partial z_i}{\partial b} = \frac{\partial}{\partial b}\left[\sum_{j=1}^{p} w_j x_{i,j} + b\right] = 1 \tag{6}$$

$$\frac{\partial z_i}{\partial w_j} = \frac{\partial}{\partial w_j}\left[\sum_{j=1}^{p} w_j x_{i,j} + b\right] = x_{i,j} \tag{7}$$

Exercise 4

After loading the data, we proceed with making to models. One is using only the `horepower` feature from the `Auto.csv` dataset to predict the value of `mpg`, and the second model uses all of the features except the `name` column. Furthermore, the dataset values are normalized by subtracting the mean from the value, and dividing by the standard deviation, this is to ensure there is no numerical instability as various features are in different units and scales. Using `learning_rates = [1, 1e-1, 1e-2, 1e-3, 1e-4]` and `num_iterations=1000`, we get the following results:

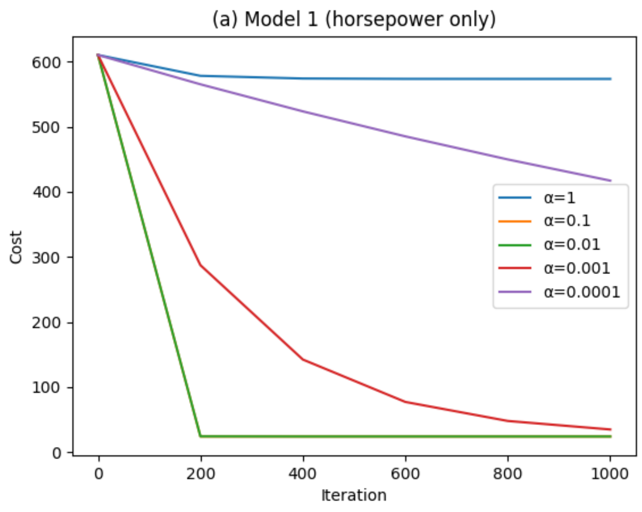

Figure 1: Plot showing how the cost (2), evaluated on the horsepower-only model, is changing with with number of iterations. It can be seen that learning rate *1* is yielding high values of cost and no signs of converging. Very small learning rates (such as 0.0001 in this case) make the model very slow at learning and converging. While we also observer that *lr* values of *0.1* and *0.01* are almost equal in terms of efficiently helping the model learn.

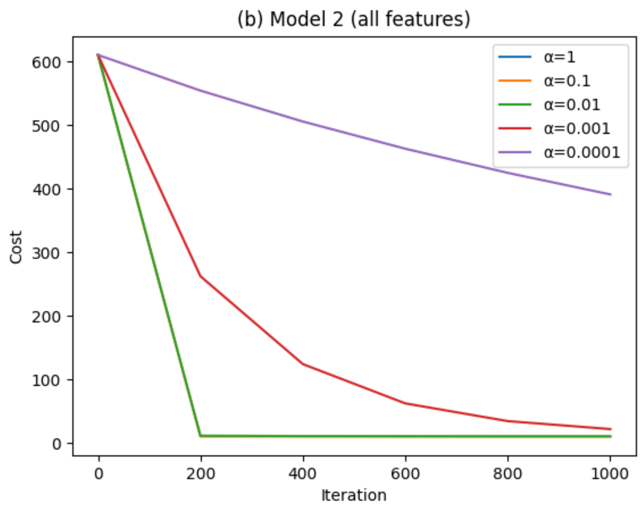

Figure 2: Plot showing how the cost (2), evaluated on the all-features model, is changing with with number of iterations. It can be seen that learning rate *1* is not plotted, as it yields infinity or nan values. Very small learning rates (such as 0.0001 in this case) make the model very slow at learning and converging. While we also observer that *lr* values of *0.1* and *0.01* are almost equal in terms of efficiently helping the model learn.

Now let's have a look at the output of the experiments:

```
All Features Model:
Minimum Cost: 10.847481499970433
Best Learning Rate: 0.1
lr = 0.100000 | Cost after 0: 610.4738
lr = 0.100000 | Cost after 200: 10.8809
lr = 0.100000 | Cost after 400: 10.8493
lr = 0.100000 | Cost after 600: 10.8476
lr = 0.100000 | Cost after 800: 10.8475
lr = 0.100000 | Cost after 1000: 10.8475

w = [[-0.6518016]
     [-0.83976308]
     [ 2.07852947]
     [-5.49827114]
     [ 0.22221673]
     [ 2.76561046]
     [ 1.14850055]], b = 23.445918367346934
```

```
Horsepower Only Model:
Minimum Cost: 23.943662938603108
Best Learning Rate: 0.1
lr = 0.100000 | Cost after 0: 610.4738
lr = 0.100000 | Cost after 200: 23.9437
lr = 0.100000 | Cost after 400: 23.9437
lr = 0.100000 | Cost after 600: 23.9437
lr = 0.100000 | Cost after 800: 23.9437
lr = 0.100000 | Cost after 1000: 23.9437

w = [[-6.07562688]], b = 23.44591836734693
```

Exercise 4.a We observe the final cost for the horsepower-only model and all-features model to be `23.943662938603108` and `10.847481499970433` respectively.

Exercise 4.b

Furthermore, for the **All-features model**, given the vector $\mathbf{w}$ and bias b as follows:

$$\mathbf{w} = \begin{pmatrix} -0.6518 & -0.8398 & 2.0785 & -5.4983 & 0.2222 & 2.7656 & 1.1485 \end{pmatrix}, \quad b = 23.4459$$

The linear regression model equation incorporating these values directly is:

$$z_i = \begin{pmatrix} -0.6518 \\ -0.8398 \\ 2.0785 \\ -5.4983 \\ 0.2222 \\ 2.7656 \\ 1.1485 \end{pmatrix} \mathbf{x}_i + 23.4459$$

where $\mathbf{x}_i$ is the feature vector for the i^{th} observation, represented as a row vector. The highest weight in $\mathbf{w}$ seems to correspond to the '`year`' feature in the dataset. While this is somewhat reasonable, as the more recent car models tend to improve the trend on the predicted value, it is also worth noting the second highest weight belongs to '`horsepower`' feature.

And similarly, the linear regression model equation for the **horsepower-only model** is as follows:

$$z_i = \begin{pmatrix} -6.0756 \end{pmatrix} \mathbf{x}_i + 23.4459$$

Exercise 4.c As discussed above, the models are trained using various values of learning rates. The plots and results of this section are presented previously in Figures 2 and 1.

Exercise 4.d For this section, we omit the normalization of the data and proceed with fitting the model. This produces warnings in the training process as shown below:

```
parameters_all, costs_all, it = train_linear_model(X_all_train, Y_train, num_iterations, 0.1)
    Cost after iteration 0: 610.473827
    Cost after iteration 200: nan
    Cost after iteration 400: nan
    Cost after iteration 600: nan
    Cost after iteration 800: nan
    Cost after iteration 1000: nan
    \A1\utils.py:84: RuntimeWarning: invalid value encountered in scalar subtract
  b = b - learning_rate * db
```

And as observed, in higher iterations, no values for cost are reported. This is due to the fact that larger features are affecting the scale of the model parameters and blowing it out of proportions resulting in unexpected behaviours.

Exercise 4.e Using the horsepower-only model and the best learning rate from the previous experiments, we proceed to plot a Scatter Plot of the data as well as the learned Regression Line. The plot is as follows:

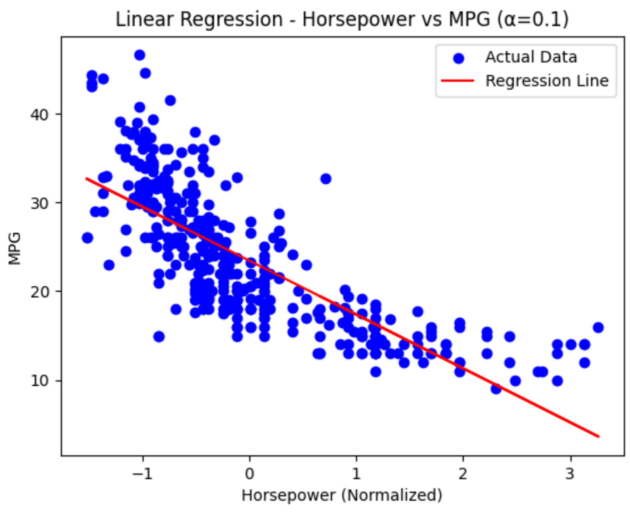

Figure 3: Scatter plot showing actual data from the `Auto.csv` dataset, and the learned regression line plotted on top of the data points. The regression line seems to capture the general trend of the data distribution to some extent. However, as the distribution is non-linear, and especially the fact that only one feature is used to learn the model, the result is sub-optimal. It showcases how one important feature can be crucial to learning the model.