بسمه تعالی

HW1- Deep Neural Networks

Supervisor:

Prof. Johari Majd

Student:

Morteza Bigdeli - 40261662001

Tarbiat Modares University

Spring 2024

Contents

Table if Figures

1. Class Exercises

For $\hat{y} = f_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$ where $\sigma(x) = \frac{1}{1+e^{-x}}$ with binary cross entropy loss: $\mathcal{L}(\hat{y}_i, y_i) = -y_i \ln \hat{y}_i - (1-y_i)\ln(1-\hat{y}_i)$

Prove that gradient is simply: $\nabla_{\mathbf{w}} \mathcal{L}(\hat{y}_i, y_i) = (\hat{y}_i - y_i)\mathbf{x}_i$

Also prove that $\mathcal{L}(\hat{y}_i, y_i)$ is convex wrt. $\mathbf{w}$. So gradient-based algorithm yield global min.

There are two proofs in the lecture that will be represented for this part:

The derivative of cross entropy, considering chain rule is:

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial(-y_i \ln \hat{y}_i) - (1-y_i)\ln(1-\hat{y}_i)}{\partial \hat{y}}$$

$$= \frac{\partial(-y_i \ln \hat{y}_i)}{\partial \hat{y}} + \frac{\partial(-(1-y_i)\ln(1-\hat{y}_i))}{\partial \hat{y}}$$

$$= \frac{-y_i}{\hat{y}_i} + \frac{1-y_i}{1-\hat{y}_i} = \frac{\hat{y}_i - y_i}{\hat{y}_i(1-\hat{y}_i)} \quad \textcircled{1}$$

Since we have:

$$\frac{\partial f}{\partial x} = \frac{\partial}{\partial x}\left(\frac{1}{1+e^{-x}}\right) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{\partial \hat{y}_i}{\partial x} \quad \textcircled{2}$$

$X = [x_1, \text{—} \ x_i]$

which is equal to $= \hat{y}_i(1-\hat{y}_i)$ $\textcircled{3}$

$$2 \Rightarrow \frac{\partial \hat{y}_i}{\partial x} = \hat{y}_i(1-\hat{y}_i)x_i$$

Considering chain rule, we have

$$\frac{\partial L}{\partial x} = \frac{\partial \hat{y}}{\partial x} \cdot \frac{\partial L}{\partial \hat{y}_i} = \hat{y}_i(1-\hat{y}_i)\frac{\hat{y}_i - y_i}{\hat{y}_i(1-\hat{y}_i)}x_i = [\hat{y}_i - y_i]x_i$$

proved!

Proof that the Cross Entropy cost is convex:

To show that the Cross Entropy cost function is convex we can use the second order definition of convexity, by which we must show that the eigenvalues of this cost function's Hessian matrix are all always nonnegative. Studying the Hessian of the cross-entropy, we have:

$$\mathcal{L}(w) = \frac{1}{P}\sum_{p=1}^{p} \mathcal{L}_p(w)$$

$$\nabla^2 \mathcal{L}(w) = \frac{1}{P}\sum_{p=1}^{p} \sigma(x_P^T w)(1 - \sigma(x_P^T w))x_p x_P^T$$

We know that the smallest eigenvalue of any square symmetric matrix is given as the minimum of the Rayleigh quotient, the smallest value taken by

$$z^T \nabla^2 \mathcal{L}(w) z$$

for any unit-length vector z and any possible weight vector w. Substituting in the particular form of the Hessian here, denoting $\sigma_p = \sigma(x_P^T w)(1 - \sigma(x_P^T w)$ for each p for short, we have:

$$z^T \nabla^2 \mathcal{L}(w) z = z^T (\frac{1}{P}\sum_{p=1}^{P} \sigma_p x_p x_p^T) z = \frac{1}{P}$$

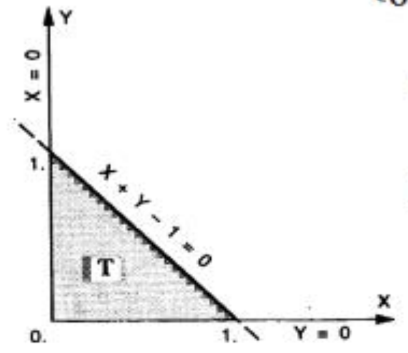
$$\sum_{p=1}^{P} \sigma_p \left(z^T x_p\right)\left(x_p^T z\right) = \frac{1}{P}\sum_{p=1}^{P} \sigma_p (z^T x_p)^2$$

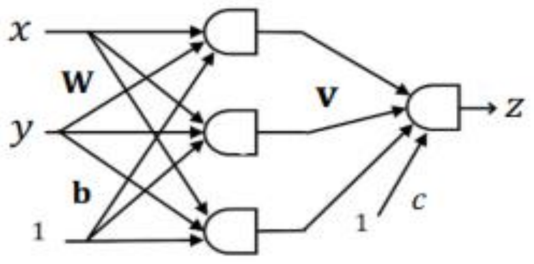
Since it is always the case that, $(z^T x_p)^2 \geq 0$ and $\sigma_p \geq 0$, it follows that the smallest value the above can take is 0, meaning that this is the smallest possible eigenvalue of the logistic cost's Hessian. Since this is the case, the logistic cost must be convex.

2. MLP Engineering

Find the wieghts and biases of the following MLP without training to recognize correctly whether an arbitrary point (x, y) is in the shaded triangular area T or not. (Note that the points on the three dividing boundary lines are considered to be outside area T). Consider all activation functions are hard limiters. The MLP output should be:

$$z = \begin{cases} 1 & \text{if } (x, y) \in T \\ 0 & \text{otherwise} \end{cases}$$





Q2.

Lets write the equations with weights considering the line equation for obtaining the values:

for N① $\rightarrow xw_1 + yw_1 + b_1$

for N② $\rightarrow xw_2 + yw_2 + b_2$

for N③ $\rightarrow xw_3 + yw_3 + b_3$

with line equation we have $\rightarrow \begin{cases} x+y-1=0 \\ \text{if } x=0 \rightarrow y=1 \\ \text{if } y=0 \rightarrow x=1 \end{cases}$

considering these conditions $\rightarrow \begin{cases} xw_1+b_1=0 \\ xw_2+b_2=0 \\ \rightarrow yw_3+b_3=0 \end{cases}$

it can be implied:

$$\begin{cases} w_1=1 & w_2=1 & b_1=-1 \\ w_1=0 & w_2=1 & b_2=0 \\ w_1=1 & w_2=0 & b_3=0 \end{cases}$$

Now Lets Apply hardlim for the next layer to obtain $\underline{\underline{c}}$ value:

Consider one point in gray space like $1/4, 1/4$

Now we have:

① $\rightarrow xw_1 + yw_1 + b_1 \rightarrow 1/4 + 1/4 - 1 = -1/2 \xrightarrow[\text{lim}]{\text{hard}} 0$ $\boxed{k_1}$

② $\rightarrow xw_1 + b_1 \rightarrow 1/4 + 0 \xrightarrow{\text{hardlim}} 1$ $\boxed{k_2}$

$\boxed{3} \rightarrow \cancel{m}\, y w_3 + b_3 \rightarrow 1/4 + 0 \xrightarrow{\text{hardlim}} \boxed{1}$ $\boxed{k_3}$

Now we have:

$k_1 + k_2 + k_3 + C = 2 \Longrightarrow 0 + 1 + 1 + C = 0 \rightarrow C = -2$

For Another Point $\rightarrow 1/4, 1/2$

$\boxed{1} \rightarrow 1/4 + 1/2 - 1 = -1/4 \xrightarrow[\text{lim}]{\text{hard}} 0$

$\boxed{2} \rightarrow 1/4 \xrightarrow{\text{hardlim}} \boxed{1}$

$\boxed{3} \rightarrow 1/2 \xrightarrow{\text{hardlim}} \boxed{1}$

$\cancel{2} \Rightarrow 0 + 1 + 1 + C = 0 \rightarrow C = -2$

3. Computational Graphs and Backpropagation

Consider the following loss function and model with a, b, c, d, m, n as its parameters:

$$\mathcal{L} = \sum_{i=1}^{3} (\hat{y}_i - y_i)^2 \qquad \hat{y}_i = \frac{e^a}{\sqrt{bx_i}} \log(cx_i + d) + m \sin(ne^{x_i})$$

3.1. Solution of a

Introduce intermediate variables $(t_1, t_2, t_3, \ldots)$ such that there is a single mathematical operation per assignment. Reuse intermediate variables whenever possible.

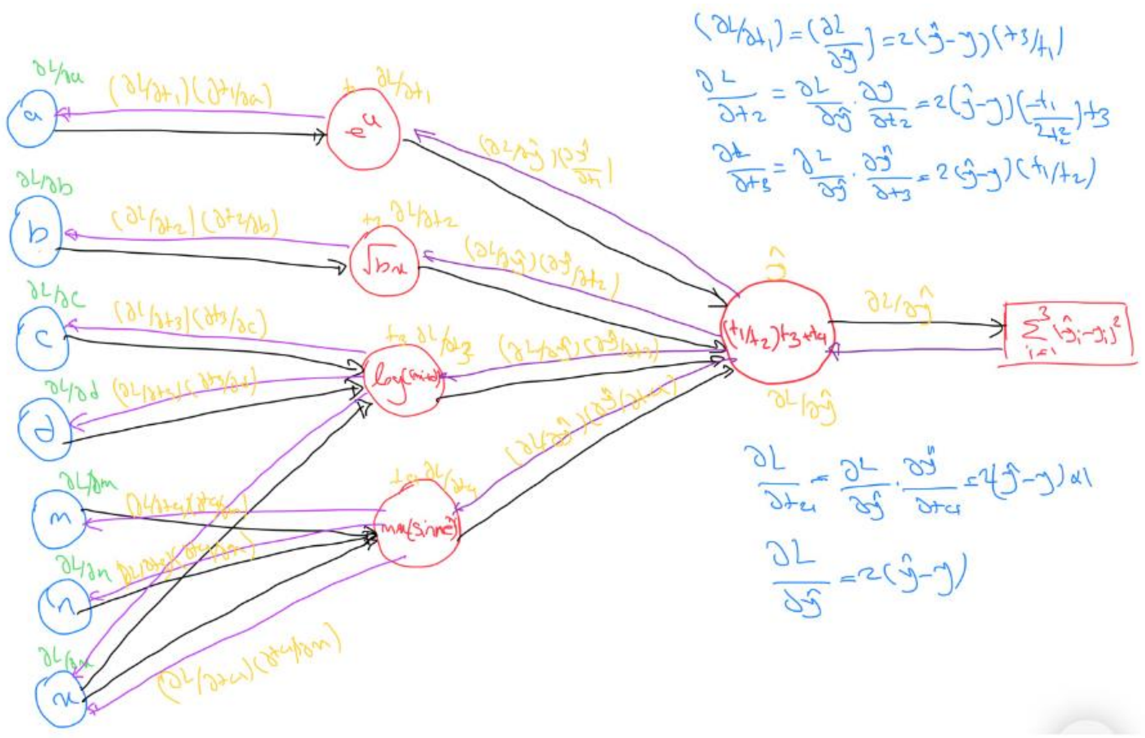
A) To introduce intermediate variables for the given loss function and model with parameters a,b,c,d,m,n, we can break down the calculation into smaller steps with a single mathematical operation per assignment. Here is a possible breakdown using intermediate variables t1,t2,t3, and so on,

t1=e^x t2=$\sqrt{bx}$ t3=log(cx+d) t4=msin(ne^x)

Calculate $\hat{y}$ =(t1/t2)t3+t4

Calculate the loss function L=summation ($\hat{y}$-y)

3.2. Solution of b



$(\partial L/\partial t_1) = (\partial L/\partial \hat{y}) = 2(\hat{y}-y)(t_3/t_1)$
$\frac{\partial L}{\partial t_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial t_2} = 2(\hat{y}-y)(\frac{-t_1}{2t_2})t_3$
$\frac{\partial L}{\partial t_3} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial t_3} = 2(\hat{y}-y)(t_1/t_2)$
$\frac{\partial L}{\partial t_4} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial t_4} = 2(\hat{y}-y)\alpha 1$
$\frac{\partial L}{\partial \hat{y}} = 2(\hat{y}-y)$
a
b
c
d
m
n
x
e^a
$\sqrt{bx}$
log(cx+d)
$m\sin(ne^x)$
$(t_1/t_2)t_3+t_4$
$\sum_{i=1}^{3}(\hat{y}_i-y_i)^2$
$\partial L/\partial a$
$\partial L/\partial b$
$\partial L/\partial c$
$\partial L/\partial d$
$\partial L/\partial m$
$\partial L/\partial n$
$\partial L/\partial x$
$\partial L/\partial t_1$
$\partial L/\partial t_2$
$\partial L/\partial t_3$
$\partial L/\partial t_4$
$\partial L/\partial \hat{y}$
$\hat{y}$
$(\partial L/\partial t_1)(\partial t_1/\partial a)$
$(\partial L/\partial t_2)(\partial t_2/\partial b)$
$(\partial L/\partial t_3)(\partial t_3/\partial c)$
$(\partial L/\partial t_3)(\partial t_3/\partial d)$
$(\partial L/\partial t_4)(\partial t_4/\partial m)$
$(\partial L/\partial t_4)(\partial t_4/\partial n)$
$(\partial L/\partial t_4)(\partial t_4/\partial x)$
$(\partial L/\partial \hat{y})(\partial \hat{y}/\partial t_1)$
$(\partial L/\partial \hat{y})(\partial \hat{y}/\partial t_2)$
$(\partial L/\partial \hat{y})(\partial \hat{y}/\partial t_3)$
$(\partial L/\partial \hat{y})(\partial \hat{y}/\partial t_4)$

3.3. Solution of c and d

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial t_1} \cdot \frac{\partial t_1}{\partial a} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial t_1} \cdot \frac{\partial t_1}{\partial a} = 2(\hat{y} - y)(t_3/t_2)e^{a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial t_2} \cdot \frac{\partial t_2}{\partial b} = \frac{\partial L}{\partial y} \cdot \frac{\partial \hat{y}}{\partial t_2} \cdot \frac{\partial t_2}{\partial b} = 2(\hat{y} - y)\left(\frac{-t_1}{2t_2^2} t_3\right)\left(\frac{n}{2\sqrt{bn}}\right)$$

$$\frac{\partial L}{\partial c} = \frac{\partial L}{\partial t_3} \cdot \frac{\partial t_3}{\partial c} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial t_3} \cdot \frac{\partial t_3}{\partial c} = 2(\hat{y} - y)(t_1/t_2)\left(\frac{n}{(m+d)\ln n}\right)$$

$$\frac{\partial L}{\partial d} = \frac{\partial L}{\partial t_3} \cdot \frac{\partial t_3}{\partial d} = \frac{\partial L}{\partial y} \cdot \frac{\partial \hat{y}}{\partial t_3} \cdot \frac{\partial t_3}{\partial d} = 2(\hat{y} - y)(t_1/t_2)\left(\frac{1}{(m+d)\ln n}\right)$$

$$\frac{\partial L}{\partial m} = \frac{\partial L}{\partial t_4}\left(\frac{\partial t_4}{\partial n}\right) = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial t_4} \cdot \frac{\partial t_4}{\partial n} = 2(\hat{y} - y)\times 1\,(mne^{mi}\cos(ne^{mi}))$$

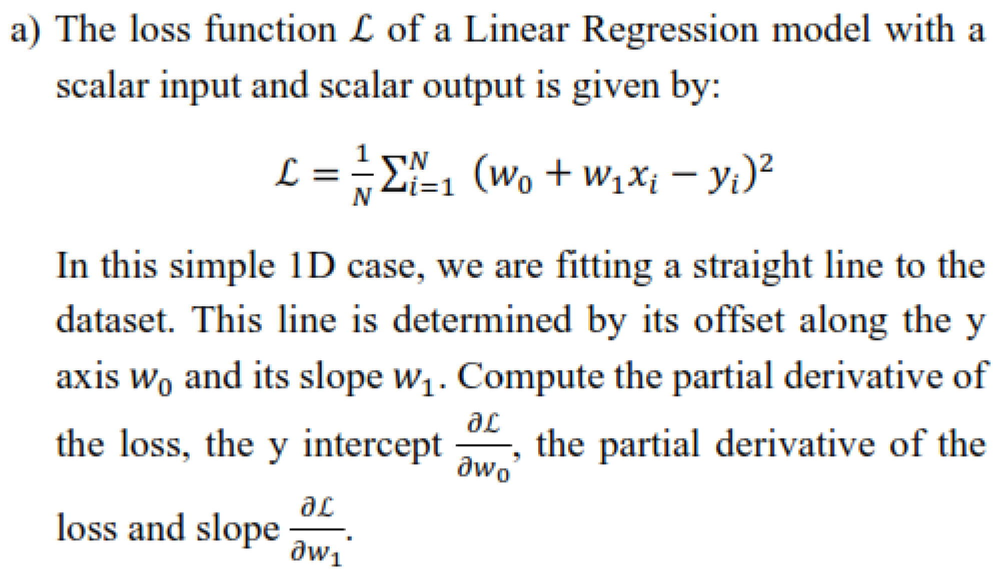
$$\frac{\partial L}{\partial n} = \frac{\partial L}{\partial t_2} \cdot \frac{\partial t_2}{\partial n} + \frac{\partial L}{\partial t_3} \cdot \frac{\partial t_3}{\partial n} + \frac{\partial L}{\partial t_4} \cdot \frac{\partial t_4}{\partial n} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial t_2} \cdot \frac{\partial t_2}{\partial n} +$$
$$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial t_3} \cdot \frac{\partial t_3}{\partial n} + \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial t_4} \cdot \frac{\partial t_4}{\partial n} = 2(\hat{y} - y)\left(\frac{-t_1}{2t_2^2} t_3\right)\left(\frac{b}{2\sqrt{bn}}\right) +$$
$$2(\hat{y} - y)\times t_1/t_2\left(\frac{c}{(m+d)\ln n}\right) + 2(\hat{y} - y)\times 1\left[m\sin(ne^{mi}) + ne^{mi}mn\cos(ne^{mi})\right]$$

4. Linear Regression

There are three parts for this question

The github address for the codes is: https://github.com/Mbigdeli2003/Deep-Learning.git

4.1. Solution of a



a) The loss function $\mathcal{L}$ of a Linear Regression model with a scalar input and scalar output is given by:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} (w_0 + w_1 x_i - y_i)^2$$

In this simple 1D case, we are fitting a straight line to the dataset. This line is determined by its offset along the y axis w_0 and its slope w_1. Compute the partial derivative of the loss, the y intercept $\frac{\partial \mathcal{L}}{\partial w_0}$, the partial derivative of the loss and slope $\frac{\partial \mathcal{L}}{\partial w_1}$.

To compute the partial derivatives of the loss function $\mathcal{L}$ with respect to the y-intercept (w_0) and the slope (w_1) in the context of linear regression, we can start by calculating the derivatives.

Given the loss function:
$\mathcal{L} = 1/N \sum_{i=1}^{xn} (w_0 + w_1 x_i - y_i)^2$

1. Partial derivative of $\mathcal{L}$ with respect to w_0 (y-intercept):
$\partial\mathcal{L}/\partial w_0 = 2/N \sum_{i=1}^{xn} (w_0 + w_1 x_i - y_i)$

2. Partial derivative of $\mathcal{L}$ with respect to w_1 (slope):
$\partial\mathcal{L}/\partial w_1 = 2/N \sum_{i=1}^{xn} (w_0 + w_1 x_i - y_i) * x_i$

4.2. Solution of b

In file HW_4 ipnyb python the solution has been implemented.

Now here is the results for this part:

Fig1 shows the loaded data set for 100*100, the output of the code for the size of data set is:

```
he size of X array= (100,)
The size of Y aray= (100,)
```

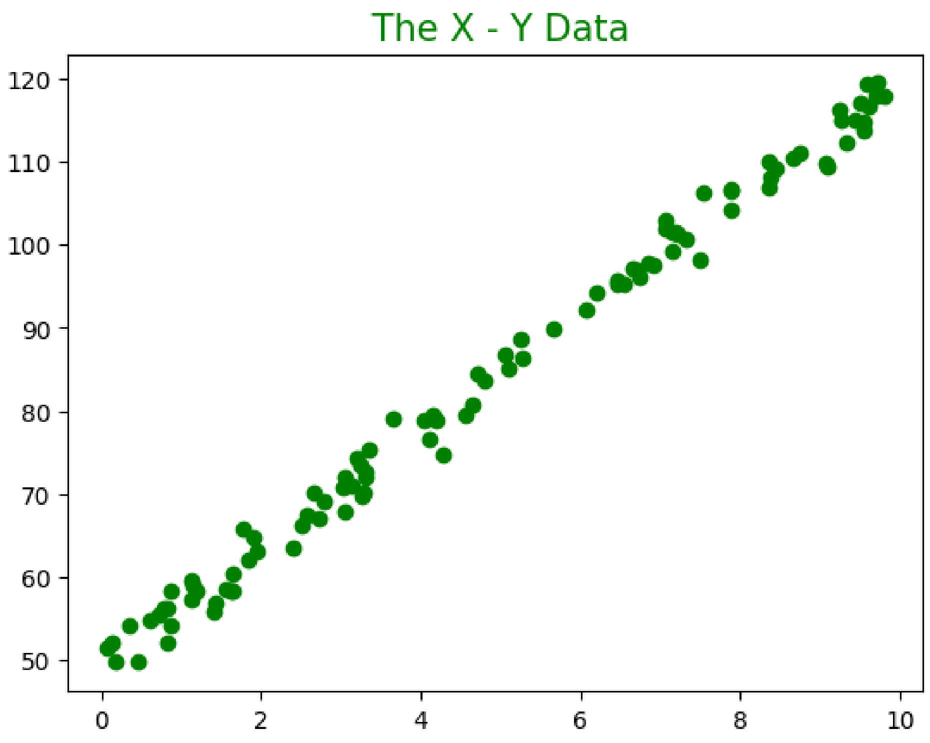


Figure 1. X.Y. data

Now lets work with model regression fit in python for one epoch, Fig.2 shows the result for 100*100 data set, the obtained value for the slope and offset are:

```
slope w1= 7.03528164368494
offset w0= 49.458623084782054
```

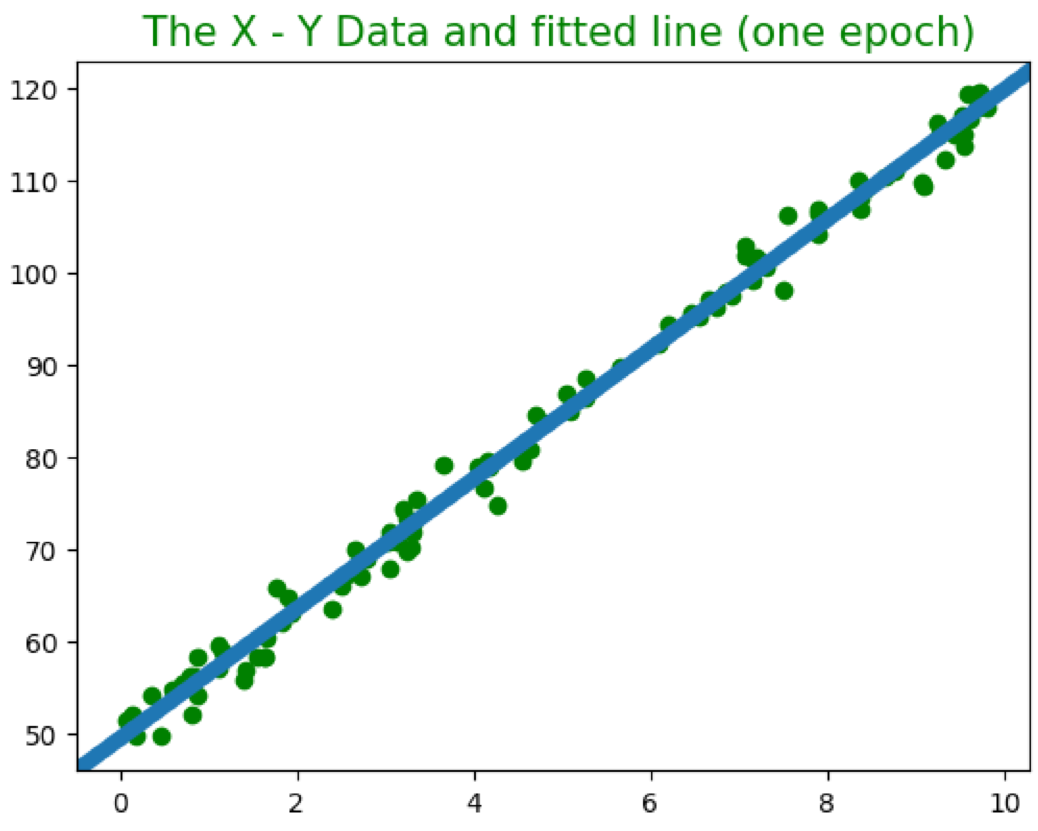


Figure 2. LR fitted line (one epoch)

Using OLS pyhton syntax for this problem has been shown below:

```
                               OLS Regression Results
=======================================================================
================
Dep. Variable:                      y   R-squared (uncentered):
0.907
Model:                            OLS   Adj. R-squared (uncentered):
0.906
Method:                 Least Squares   F-statistic:
963.9
Date:                Wed, 03 Apr 2024   Prob (F-statistic):
7.89e-53
Time:                        16:54:45   Log-Likelihood:
-469.12
No. Observations:                 100   AIC:
940.2
Df Residuals:                      99   BIC:
942.8
Df Model:                           1
Covariance Type:            nonrobust
=======================================================================
=======
                 coef    std err          t      P>|t|      [0.025
0.975]
-----------------------------------------------------------------------
-------
x1            14.3296      0.462     31.046      0.000      13.414
15.245
```

```
==============================================================================
Omnibus:                       49.069   Durbin-Watson:                   1.648
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                7.015
Skew:                          -0.114   Prob(JB):                       0.0300
Kurtosis:                       1.723   Cond. No.                         1.00
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Now, lets split the data and fit with model regression with 80*80, Fig.3 shows the fitted model with the split data, and here is the result for the slope and offset in with this data:

```
slope w1= 7.012410764012327
offset w0= 49.474212653708356
```

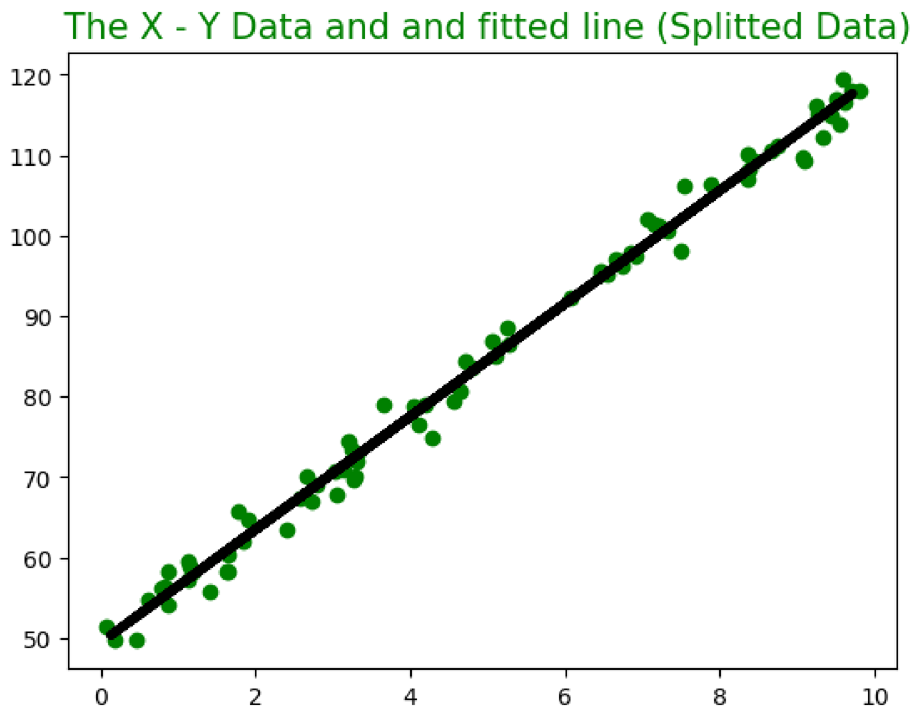


Figure 3. LR fitted line split data (one epoch)

The total loss has been obtained:

```
Loss for one Epoch= 3.639642205210527
```

Now, lets apply the gradient decent algorithm for the data, and fit the line with epochs, and learning rate which has been set 1500, 0.01, respectively for this simple case, the losses during epochs are:

```
Epoch [100/1500], Loss: 206.7655
Epoch [200/1500], Loss: 74.1709
Epoch [300/1500], Loss: 28.1301
Epoch [400/1500], Loss: 12.1435
Epoch [500/1500], Loss: 6.5924
Epoch [600/1500], Loss: 4.6649
Epoch [700/1500], Loss: 3.9957
Epoch [800/1500], Loss: 3.7633
Epoch [900/1500], Loss: 3.6826
Epoch [1000/1500], Loss: 3.6545
Epoch [1100/1500], Loss: 3.6448
Epoch [1200/1500], Loss: 3.6414
Epoch [1300/1500], Loss: 3.6403
Epoch [1400/1500], Loss: 3.6399
Epoch [1500/1500], Loss: 3.6397
```

The resulted loss in epochs show that after about 800, the loss did not decrease a lot during learning procedure, so, it can be implied that about 800 epochs would be enough for this training procedure for this data with applied model.

the Fig.4 shows the total loss for each epoch and all iterations in learning. It is obvious that after about 800 epochs the loss is getting constant during the training procedure. Fig.5 shows the fitted line with applying linear regression which shows the accuracy of the fitted model.

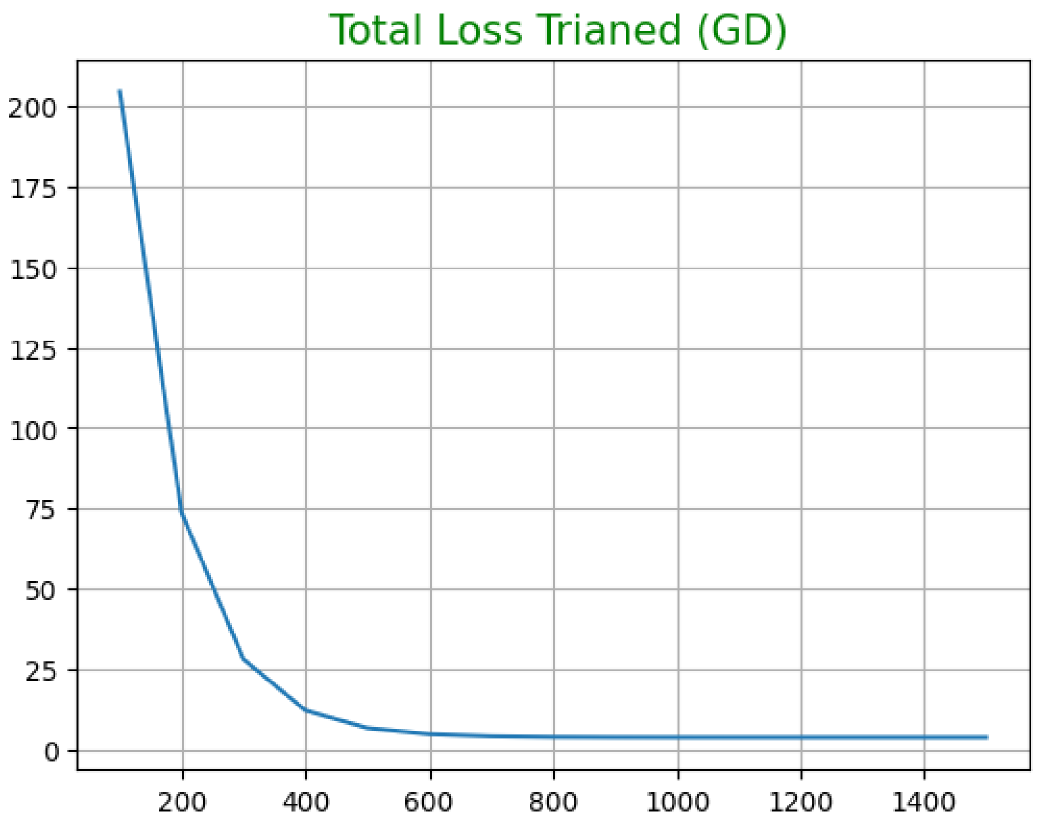


Figure 4. LR total loss (1500 epochs)

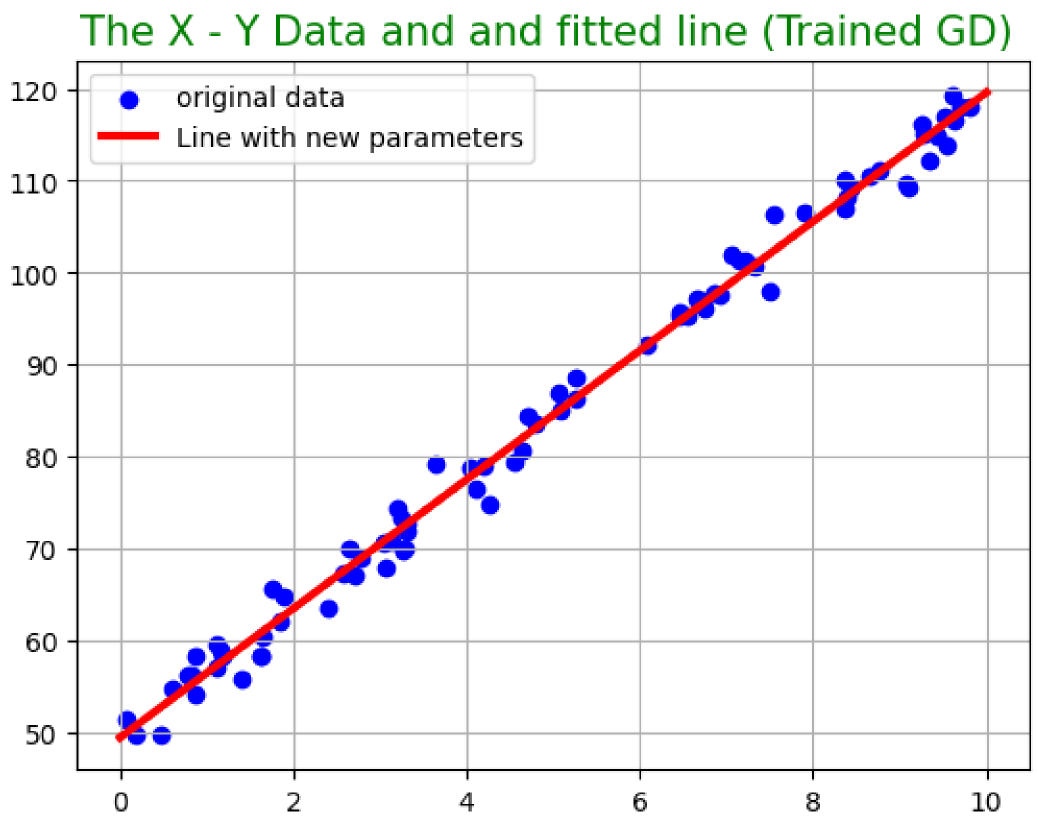


Figure 5. LR fitted line (1500 epochs)

4.3. Solution of c

In this part the closed-form solution to the linear regression problem will be used, the formulation can be expressed as:

RSS $\Rightarrow E(\beta_0, \beta_1) = \beta_0 + \beta_1 x$ OR

$E(b, m) = b + mx$

To find the minimum point of RSS curver derivate E/RSS w.r.t the variables and set the derivative equal to zero.

$$\frac{\partial E(b,m)}{\partial b} = 0 \qquad \frac{\partial E(b,m)}{\partial m} = 0$$

here $m = \beta_1 \quad b = \beta_0$

A) $$\frac{\partial E}{\partial b} = \frac{\partial \left(\sum_1^n (y_i - \hat{y_i})^2 \right)}{\partial b} = 0$$

$$\frac{\partial \left(\sum_1^n (y_i - mx_i - b)^2 \right)}{\partial b} = 0$$

$$\sum_1^n -2 (y_i - mx_i - b) = 0$$

$$\sum_1^n y_i - \sum_1^n mx_i - \Sigma b = 0$$

$$\sum_1^n \frac{y_i}{n} - \frac{\sum_1^n mx_i}{n} - \frac{\Sigma b}{n} = \frac{0}{n}$$

$$\bar{y} - m\bar{x} - \frac{nb}{n} = 0$$

$$\bar{y} - m\bar{x} - b = 0$$

$$\boxed{b = \bar{y} - m\bar{x}}$$

equation for b

$$\frac{\partial E}{\partial m} \Rightarrow \frac{\partial\left(\sum_{1}^{n} (y_i - mx_i - b)^2\right)}{\partial m} = 0$$

$$\Sigma\, 2\,(y_i - mx - \bar{y} + m\bar{x})(-x_i + \bar{x}) = 0$$

$$\Sigma\left[(y_i - \bar{y}) - m(x_i - \bar{x})\right](x_i - \bar{x}) = 0$$

$$\Sigma\left[(y_i - y)(x_i - \bar{x}) - m(x_i - \bar{x})^2\right] = 0$$

$$\Sigma(y_i - \bar{y})(x_i - \bar{x}) = m\,\Sigma(x_i - \bar{x})^2$$

$$\boxed{m = \frac{\Sigma(x_i - \bar{x})(y_i - \bar{y})}{\Sigma(x_i - \bar{x})^2}}$$

equation for finding slope.

Now ~~R~~ replace m and b with β_1 and β_1.

$$\boxed{\beta_1 = \frac{\sum_{1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\Sigma(x_i - \bar{x})^2}}$$

$$\boxed{\beta_0 = \bar{y} - \beta_1 \bar{x}}$$

Fig.6 shows the fitted line by applying the closed form solution for linear regression and the implemented code by pytorch. The obtained loss value is:

```
Loss: 3.639643430709839
```

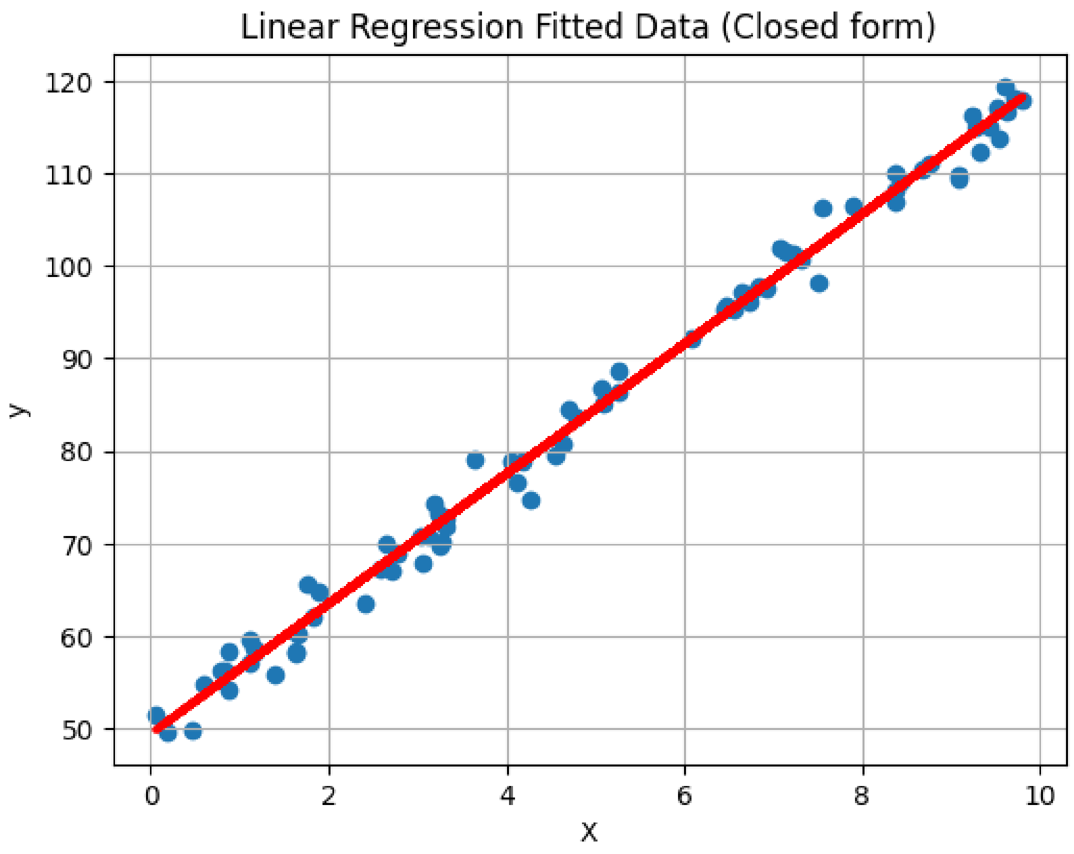
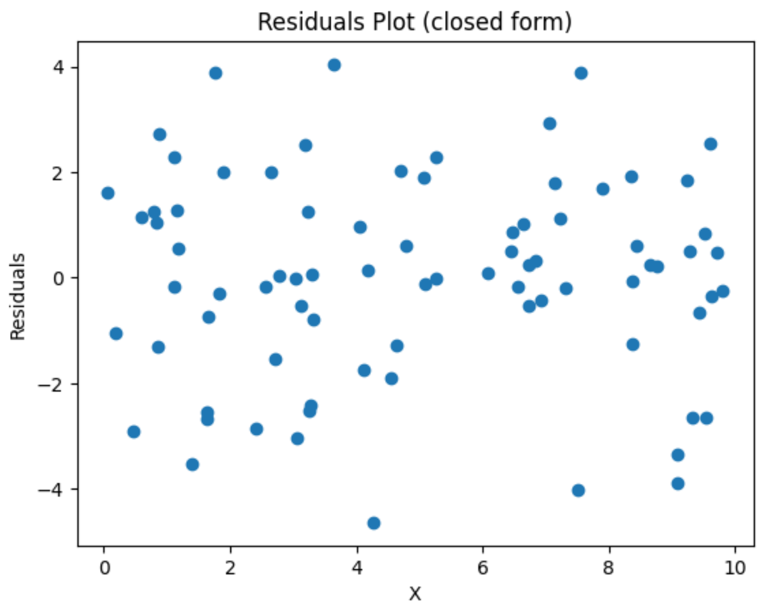


Figure 6. LR closed form fitted line

Fig.7 show the residuals for this training approach:



`

Figure 7. LR closed form residulas

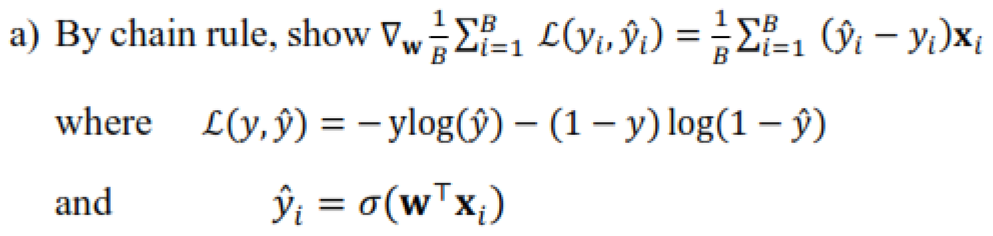
Comparing the results for last two approaches shows that both of them fitted the line perfectly and the final total loss are about same, however, in gradient decent algorithms the training time would be more weights should be updated to fit the line.

5. Logistic Regression

There are three parts for this question that will be answered and the results of implementations will be indicated as follows.

The github address for the codes is: https://github.com/Mbigdeli2003/Deep-Learning.git

5.1. Solution of a



a) By chain rule, show $\nabla_{\mathbf{w}} \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}(y_i, \hat{y}_i) = \frac{1}{B}\sum_{i=1}^{B} (\hat{y}_i - y_i)\mathbf{x}_i$

where $\mathcal{L}(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$

and $\hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$

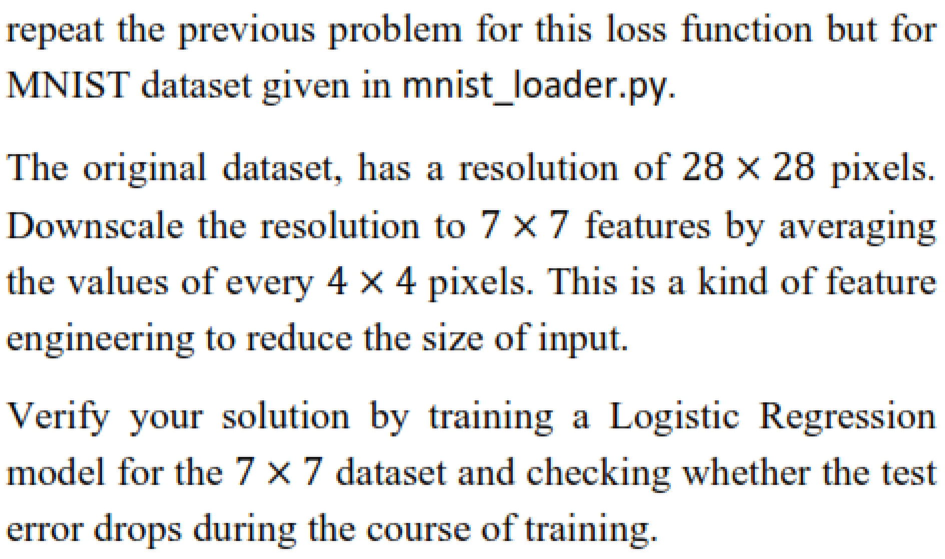
$\nabla \hat{y}i\ \mathcal{L}(yi, \hat{y}i) = \nabla \hat{y}i\ [-yi \log(\hat{y}i) - (1 - yi) \log(1 - \hat{y}i)] = -yi/\hat{y}i + (1 - yi)/(1 - \hat{y}i)$

Next, we apply the chain rule to find the gradient of the loss function $\mathcal{L}(yi, \hat{y}i)$ with respect to **w**:
$\nabla\mathbf{w}\ \mathcal{L}(yi, \hat{y}i) = \nabla\mathbf{w}\ \mathcal{L}(yi, \hat{y}i) * \nabla \hat{y}i\ \mathcal{L}(yi, \hat{y}i) = (\hat{y}i - yi)\mathbf{x}i$

Finally, we can express the overall gradient as:

$\nabla\mathbf{ww}\ 1/B \sum B\ (\hat{y}i - yi)\mathbf{x}i = 1/B \sum B\ (\hat{y}i - yi)\mathbf{x}i$

This completes the derivation of the given equation using the chain rule.



repeat the previous problem for this loss function but for MNIST dataset given in mnist_loader.py.

The original dataset, has a resolution of 28×28 pixels. Downscale the resolution to 7×7 features by averaging the values of every 4×4 pixels. This is a kind of feature engineering to reduce the size of input.

Verify your solution by training a Logistic Regression model for the 7×7 dataset and checking whether the test error drops during the course of training.

The python code for this part is in HW1_5. At this section, first the training with logistic regression will applied on 28*28 pixels, then the reducing size to 7*7 by averaging will be applied to the algorithm. The test result will be compared at last.

Mnist data is handwritten figures, which are in gray shapes, some of this images has been shown in Fig.8.

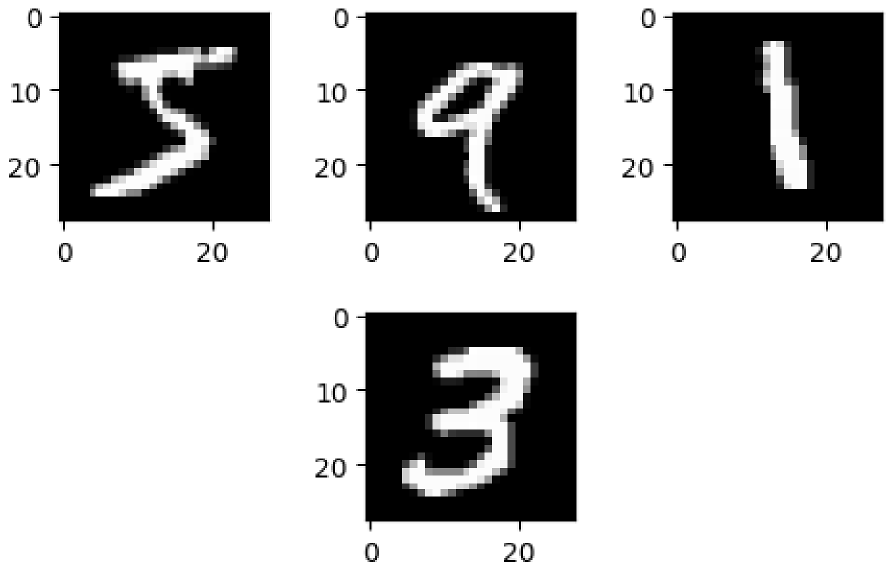


Figure 8. Mnist data

Now, lets train the data with 28*28 pixels with 10 epochs, the learning rate of 0.01, and 10000 iterations. Fig.9 show the loss during iterations, and here is the numerical results during training which indicates that the total loss has been started from about 1.9 and reduced to 1.5 with these assumptions:

```
Epoch [1/10], Step [100/938], Loss: 1.9587
Epoch [1/10], Step [200/938], Loss: 1.8448
Epoch [1/10], Step [300/938], Loss: 1.7918
Epoch [1/10], Step [400/938], Loss: 1.7704
Epoch [1/10], Step [500/938], Loss: 1.7448
Epoch [1/10], Step [600/938], Loss: 1.7481
Epoch [1/10], Step [700/938], Loss: 1.7476
Epoch [1/10], Step [800/938], Loss: 1.7170
Epoch [1/10], Step [900/938], Loss: 1.7287
Epoch [2/10], Step [100/938], Loss: 1.6710
Epoch [2/10], Step [200/938], Loss: 1.7213
Epoch [2/10], Step [300/938], Loss: 1.6719
Epoch [2/10], Step [400/938], Loss: 1.6421
Epoch [2/10], Step [500/938], Loss: 1.6736
Epoch [2/10], Step [600/938], Loss: 1.6605
Epoch [2/10], Step [700/938], Loss: 1.6369
Epoch [2/10], Step [800/938], Loss: 1.6185
Epoch [2/10], Step [900/938], Loss: 1.6394
Epoch [3/10], Step [100/938], Loss: 1.6524
Epoch [3/10], Step [200/938], Loss: 1.6717
Epoch [3/10], Step [300/938], Loss: 1.6451
Epoch [3/10], Step [400/938], Loss: 1.6471
Epoch [3/10], Step [500/938], Loss: 1.6439
Epoch [3/10], Step [600/938], Loss: 1.6390
Epoch [3/10], Step [700/938], Loss: 1.6690
Epoch [3/10], Step [800/938], Loss: 1.6400
Epoch [3/10], Step [900/938], Loss: 1.6418
Epoch [4/10], Step [100/938], Loss: 1.6342
Epoch [4/10], Step [200/938], Loss: 1.6518
Epoch [4/10], Step [300/938], Loss: 1.6359
Epoch [4/10], Step [400/938], Loss: 1.5923
Epoch [4/10], Step [500/938], Loss: 1.6145
Epoch [4/10], Step [600/938], Loss: 1.6372
Epoch [4/10], Step [700/938], Loss: 1.6400
Epoch [4/10], Step [800/938], Loss: 1.6256
Epoch [4/10], Step [900/938], Loss: 1.5784
Epoch [5/10], Step [100/938], Loss: 1.6543
Epoch [5/10], Step [200/938], Loss: 1.5837
Epoch [5/10], Step [300/938], Loss: 1.6282
Epoch [5/10], Step [400/938], Loss: 1.6586
Epoch [5/10], Step [500/938], Loss: 1.5572
Epoch [5/10], Step [600/938], Loss: 1.6441
Epoch [5/10], Step [700/938], Loss: 1.5952
Epoch [5/10], Step [800/938], Loss: 1.6857
Epoch [5/10], Step [900/938], Loss: 1.6277
Epoch [6/10], Step [100/938], Loss: 1.6053
Epoch [6/10], Step [200/938], Loss: 1.6175
Epoch [6/10], Step [300/938], Loss: 1.6402
Epoch [6/10], Step [400/938], Loss: 1.6232
Epoch [6/10], Step [500/938], Loss: 1.5690
Epoch [6/10], Step [600/938], Loss: 1.6163
Epoch [6/10], Step [700/938], Loss: 1.6139
Epoch [6/10], Step [800/938], Loss: 1.6481
Epoch [6/10], Step [900/938], Loss: 1.6166
Epoch [7/10], Step [100/938], Loss: 1.6176
Epoch [7/10], Step [200/938], Loss: 1.6250
Epoch [7/10], Step [300/938], Loss: 1.6347
Epoch [7/10], Step [400/938], Loss: 1.6121
Epoch [7/10], Step [500/938], Loss: 1.6559
Epoch [7/10], Step [600/938], Loss: 1.6086
Epoch [7/10], Step [700/938], Loss: 1.6241
Epoch [7/10], Step [800/938], Loss: 1.5845
Epoch [7/10], Step [900/938], Loss: 1.6492
Epoch [8/10], Step [100/938], Loss: 1.5713
Epoch [8/10], Step [200/938], Loss: 1.6201
Epoch [8/10], Step [300/938], Loss: 1.5830
Epoch [8/10], Step [400/938], Loss: 1.6294
Epoch [8/10], Step [500/938], Loss: 1.5895
Epoch [8/10], Step [600/938], Loss: 1.6012
Epoch [8/10], Step [700/938], Loss: 1.5792
Epoch [8/10], Step [800/938], Loss: 1.6128
Epoch [8/10], Step [900/938], Loss: 1.5790
Epoch [9/10], Step [100/938], Loss: 1.6191
Epoch [9/10], Step [200/938], Loss: 1.5999
Epoch [9/10], Step [300/938], Loss: 1.5663
Epoch [9/10], Step [400/938], Loss: 1.6123
Epoch [9/10], Step [500/938], Loss: 1.6013
Epoch [9/10], Step [600/938], Loss: 1.6087
Epoch [9/10], Step [700/938], Loss: 1.5997
Epoch [9/10], Step [800/938], Loss: 1.5686
Epoch [9/10], Step [900/938], Loss: 1.6033
Epoch [10/10], Step [100/938], Loss: 1.6232
Epoch [10/10], Step [200/938], Loss: 1.6152
Epoch [10/10], Step [300/938], Loss: 1.6291
Epoch [10/10], Step [400/938], Loss: 1.5980
Epoch [10/10], Step [500/938], Loss: 1.5877
Epoch [10/10], Step [600/938], Loss: 1.5792
Epoch [10/10], Step [700/938], Loss: 1.6164
Epoch [10/10], Step [800/938], Loss: 1.5802
Epoch [10/10], Step [900/938], Loss: 1.5953
```

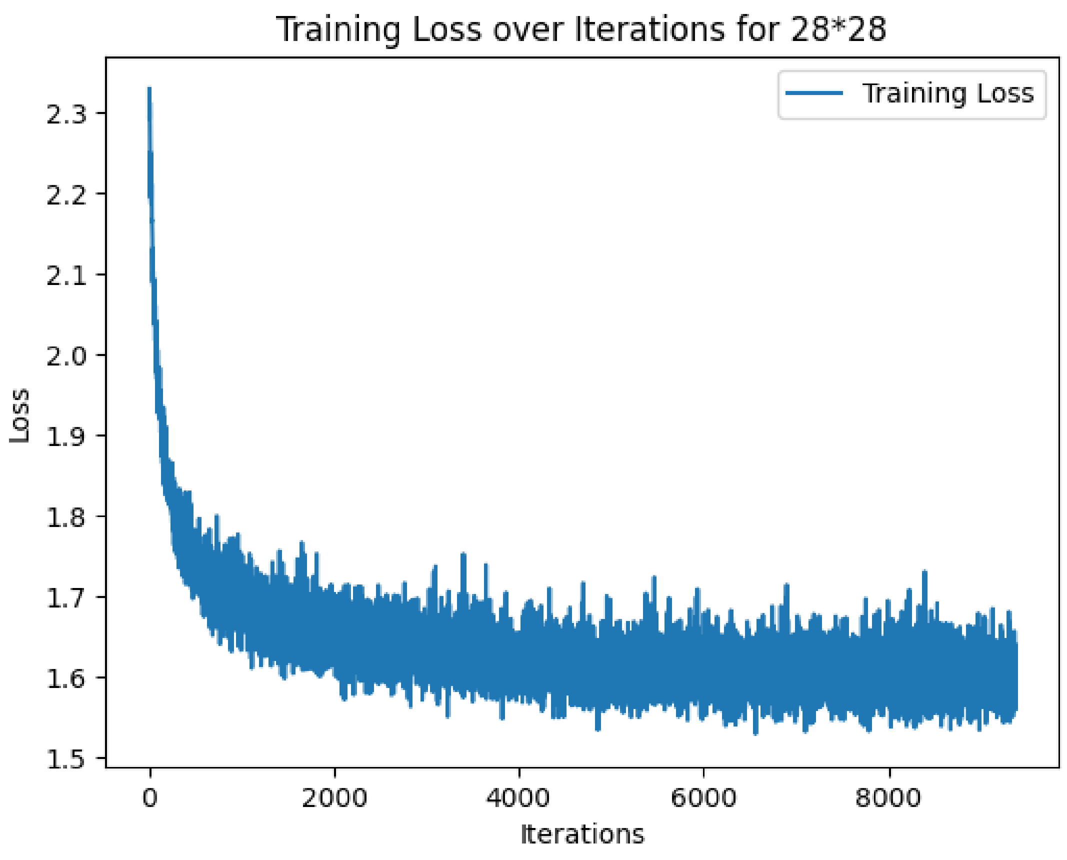


Figure 9. loss over iterations for 28*28 data

The test error for test data set is expressed as the below value:

```
Test Error for 28*28: %10.250000000000004
```

Now, we downscale the data to 7*7 by averaging 4*4 pixels with following syntax in pytorch:

```
downscaled_data = F.avg_pool2d(images_1, kernel_size=4, stride=4)
```

the results of dimension is:

```
Downscaled Data= torch.Size([60000, 7, 7])
```

By applying the considered algorithm, the loss during iterations with the same assumptions as before the output is:

```
Epoch [1/10], Step [100/938], Loss: 2.2793
Epoch [1/10], Step [200/938], Loss: 2.2602
Epoch [1/10], Step [300/938], Loss: 2.2312
Epoch [1/10], Step [400/938], Loss: 2.2102
Epoch [1/10], Step [500/938], Loss: 2.1966
Epoch [1/10], Step [600/938], Loss: 2.1687
Epoch [1/10], Step [700/938], Loss: 2.1483
Epoch [1/10], Step [800/938], Loss: 2.1328
Epoch [1/10], Step [900/938], Loss: 2.1161
Epoch [2/10], Step [100/938], Loss: 2.1123
Epoch [2/10], Step [200/938], Loss: 2.0605
Epoch [2/10], Step [300/938], Loss: 2.0562
Epoch [2/10], Step [400/938], Loss: 2.0664
Epoch [2/10], Step [500/938], Loss: 2.0583
Epoch [2/10], Step [600/938], Loss: 2.0344
Epoch [2/10], Step [700/938], Loss: 2.0282
Epoch [2/10], Step [800/938], Loss: 2.0406
Epoch [2/10], Step [900/938], Loss: 2.0312
Epoch [3/10], Step [100/938], Loss: 1.9744
Epoch [3/10], Step [200/938], Loss: 1.9792
Epoch [3/10], Step [300/938], Loss: 2.0025
Epoch [3/10], Step [400/938], Loss: 1.9993
Epoch [3/10], Step [500/938], Loss: 1.9534
Epoch [3/10], Step [600/938], Loss: 1.9662
Epoch [3/10], Step [700/938], Loss: 1.9533
Epoch [3/10], Step [800/938], Loss: 1.9669
Epoch [3/10], Step [900/938], Loss: 1.9622
Epoch [4/10], Step [100/938], Loss: 1.9393
Epoch [4/10], Step [200/938], Loss: 1.9508
Epoch [4/10], Step [300/938], Loss: 1.9159
Epoch [4/10], Step [400/938], Loss: 1.9372
Epoch [4/10], Step [500/938], Loss: 1.9301
Epoch [4/10], Step [600/938], Loss: 1.8701
Epoch [4/10], Step [700/938], Loss: 1.8840
Epoch [4/10], Step [800/938], Loss: 1.9250
Epoch [4/10], Step [900/938], Loss: 1.9178
Epoch [5/10], Step [100/938], Loss: 1.8870
Epoch [5/10], Step [200/938], Loss: 1.8950
Epoch [5/10], Step [300/938], Loss: 1.8987
Epoch [5/10], Step [400/938], Loss: 1.8933
Epoch [5/10], Step [500/938], Loss: 1.8837
Epoch [5/10], Step [600/938], Loss: 1.8501
Epoch [5/10], Step [700/938], Loss: 1.8979
Epoch [5/10], Step [800/938], Loss: 1.8861
Epoch [5/10], Step [900/938], Loss: 1.8826
Epoch [6/10], Step [100/938], Loss: 1.8619
Epoch [6/10], Step [200/938], Loss: 1.8581
Epoch [6/10], Step [300/938], Loss: 1.8568
Epoch [6/10], Step [400/938], Loss: 1.8963
Epoch [6/10], Step [500/938], Loss: 1.8696
Epoch [6/10], Step [600/938], Loss: 1.8837
Epoch [6/10], Step [700/938], Loss: 1.8336
Epoch [6/10], Step [800/938], Loss: 1.8169
Epoch [6/10], Step [900/938], Loss: 1.8982
Epoch [7/10], Step [100/938], Loss: 1.8339
Epoch [7/10], Step [200/938], Loss: 1.8057
Epoch [7/10], Step [300/938], Loss: 1.8717
Epoch [7/10], Step [400/938], Loss: 1.8237
Epoch [7/10], Step [500/938], Loss: 1.8251
Epoch [7/10], Step [600/938], Loss: 1.8261
Epoch [7/10], Step [700/938], Loss: 1.8043
Epoch [7/10], Step [800/938], Loss: 1.8028
Epoch [7/10], Step [900/938], Loss: 1.8141
Epoch [8/10], Step [100/938], Loss: 1.8202
Epoch [8/10], Step [200/938], Loss: 1.8329
Epoch [8/10], Step [300/938], Loss: 1.7897
Epoch [8/10], Step [400/938], Loss: 1.8533
Epoch [8/10], Step [500/938], Loss: 1.8139
Epoch [8/10], Step [600/938], Loss: 1.7785
Epoch [8/10], Step [700/938], Loss: 1.8312
Epoch [8/10], Step [800/938], Loss: 1.8335
Epoch [8/10], Step [900/938], Loss: 1.8179
Epoch [9/10], Step [100/938], Loss: 1.7942
Epoch [9/10], Step [200/938], Loss: 1.8645
Epoch [9/10], Step [300/938], Loss: 1.8241
Epoch [9/10], Step [400/938], Loss: 1.7923
Epoch [9/10], Step [500/938], Loss: 1.7537
Epoch [9/10], Step [600/938], Loss: 1.8219
Epoch [9/10], Step [700/938], Loss: 1.8439
Epoch [9/10], Step [800/938], Loss: 1.7848
Epoch [9/10], Step [900/938], Loss: 1.8351
Epoch [10/10], Step [100/938], Loss: 1.7771
Epoch [10/10], Step [200/938], Loss: 1.7794
Epoch [10/10], Step [300/938], Loss: 1.7875
Epoch [10/10], Step [400/938], Loss: 1.8248
Epoch [10/10], Step [500/938], Loss: 1.8156
Epoch [10/10], Step [600/938], Loss: 1.8122
Epoch [10/10], Step [700/938], Loss: 1.7948
Epoch [10/10], Step [800/938], Loss: 1.7832
Epoch [10/10], Step [900/938], Loss: 1.7948
```

The output shows that compared to 28*28 pixels training data, the loss values are higher, however, it seems it is being converged to the same vale as before. The Fig.10 shows the loss values during iterations considering downscaled data.

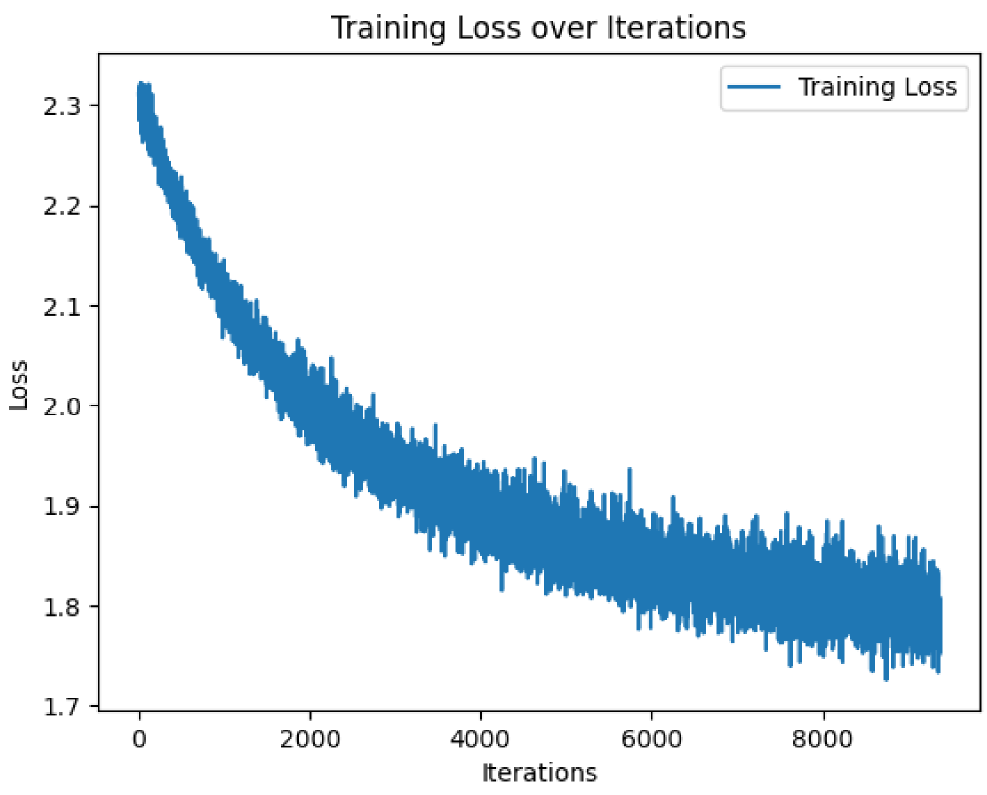


Figure 10. loss over iterations for 7*7 data

The test error for this training output is:

```
Test Error for 7*7: %10.250000000000004
```

Which is not remarkably higher than 28*28 pixels training. Therefore, the applied algorithm can be trained by reduced size techniques and the test error would not be so much different from the more extensive data.

The github address for the codes is: https://github.com/Mbigdeli2003/Deep-Learning.git