Assignment Report for ELL409

Assignment 1

Shubh Goel
2020EE10672
Ayush Kumar
2020EE10481
Vibhor Sengar
2020EE30313

Department of Electrical Engineering

September 19, 2022

Contents

1 Pre-Processing the Data

For **Part 1**, the input features, $\{x_i\}_{i=1}^{i=6}$ are mean normalised using the relation,

$$x_i' = \frac{x_i - \mu_i}{\sigma_i} \tag{1}$$

where μ_i is the mean and σ_i is the standard deviation of x_i over the training data set.

For **Part 2**, the input feature vector of size 784 is scaled down by a factor of 255 to prevent exploding of weights. The training images is split into train data and test data using **self-defined** train-test-split function.

All the weights that are presented in this report are with respect to the transformed input space.
Libraries used: Pandas,Numpy, Matplotlib, Seaborn and argparse. (**No other libraries are used**)

2 Part 1: Multivariate Linear Regression

$$t = \sum_{i=1}^{i=7} w_i x_i + \epsilon \tag{2}$$

Where w_7 is the bias term and $x_7 = 1$.

2.1 Understanding the Data

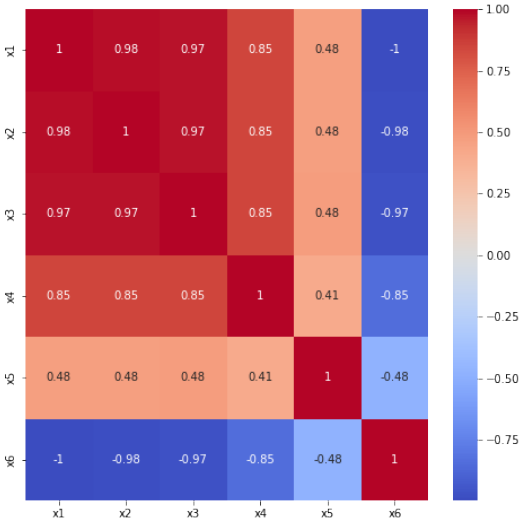

Figure 1: Pairwise Correlation Heatmap

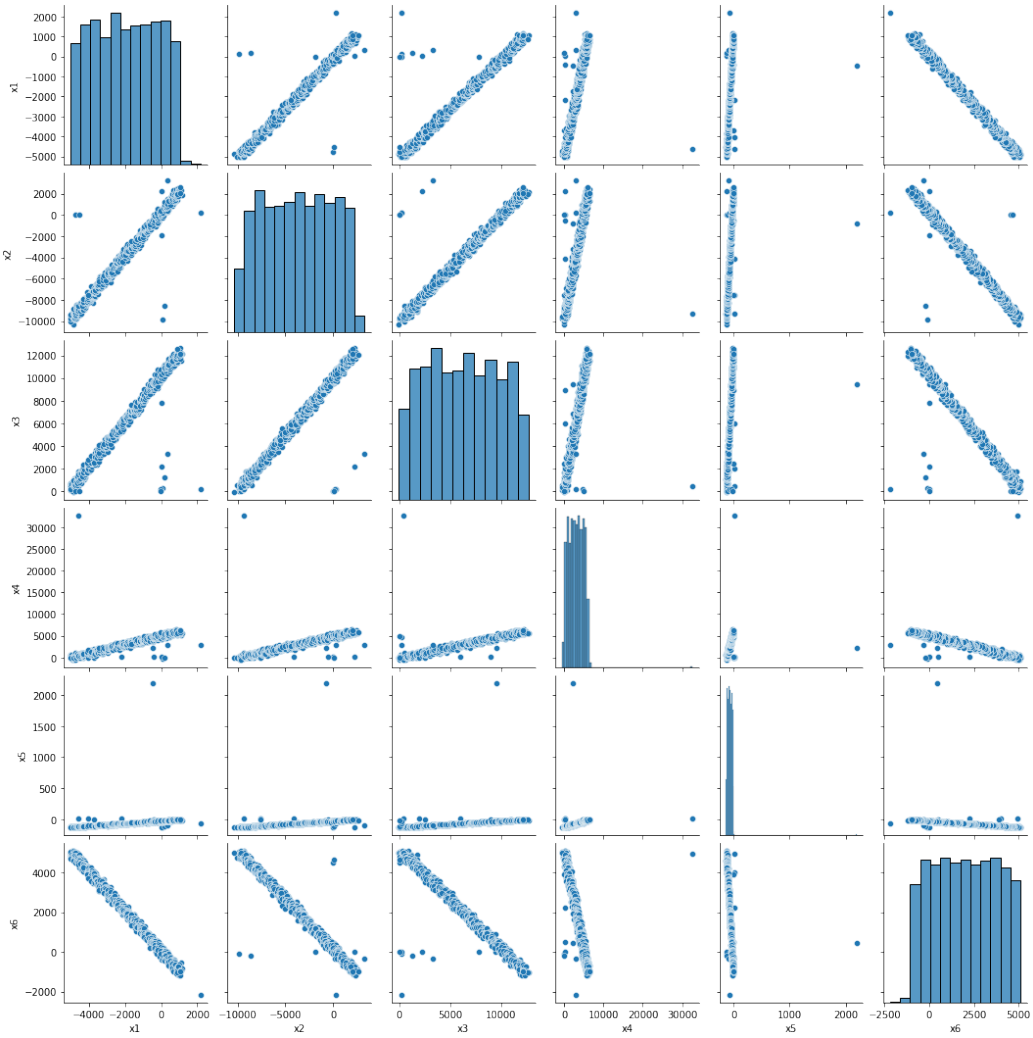

Figure 2: Pairwise Plots

From Figure 1 and Figure 2 we conclude that the features $x_1, x_2, x_3 and x_6$ are highly correlated with each other.

2.2 Effect of Batch Size

2.2.1 Stochastic Gradient Descent

Loss function, E,

$$E_i = \frac{(t_i - \mathbf{w}^T \mathbf{x}_i)^2}{2} \tag{3}$$

The above algorithm was run for 3000 epochs with $\eta = 0.001$. Following are the metrics:-

Algorithm 1 Stochastic Gradient Descent

1: **for** $i = 1, 2, \ldots, numEpochs$ **do**
2: **for** $j = 1, 2, \ldots, N$ **do**
3: $\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)} + \eta(t_j - \mathbf{w}^{(\tau)T}\mathbf{x}_j)\mathbf{x}_j$

- weights(last term is the bias term) = $\{173.751199, 8015.81382, 1741579.28, 196221.881, 0.0739931960, 87128.2142, 3507642.95\}$
- Test Mean Squares Error (MSE) = 0.0001404474867029449
- Test Mean Absolute Error (MAE) = 0.009349058765219525

2.2.2 Batch Gradient Descent

Loss function, E_{batch},

$$E_{batch} = \frac{1}{2k} \sum_{i=1}^{k} \{t_i - \mathbf{w}^T \mathbf{x}_i\}^2 \tag{4}$$

Where k is the batch size.

Algorithm 2 Batch Gradient Descent

1: **for** $i = 1, 2, \ldots, numEpochs$ **do**
2: **for** $j = 1, 1 + k, 1 + 2k \ldots, N$ **do**
3: $\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)} + \frac{\eta}{k} \sum_{l=j}^{j+k} \{t_l - \mathbf{w}^{(\tau)T}\mathbf{x}_l\}\mathbf{x}_l$

The above algorithm was run for 3000 epochs with $\eta = 0.001$ for different batch sizes. Following are the metrics:-

Batch Size	w_1	w_2	w_3	w_4	w_5	w_6
1	173.7511993	8015.813819	1741579.277	196221.881	0.073993196	87128.2142
16	-14023.76157	11073.37413	1740586.277	196340.0095	-7.185344052	75090.12046
64	9657.76793	116424.7222	1551471.445	205536.0716	2027.435321	25384.12342
256	210706.2242	284970.0435	921616.0052	241898.1569	9320.175843	-194995.7627
512	287457.0001	329080.5146	701317.8029	264324.0756	12951.08701	-272702.7376
1200	336608.219	357021.0115	514664.4506	299828.4095	36522.1116	-333213.5177

Table 1: Weights for Batch Gradient Descent

Batch Size	Bias (w_7)	Test MSE	Test MAE	Correlation Coefficient
1	3507642.949	0.000140447	0.009349059	1
16	3507642.419	1405315.229	925.5948526	0.99999882
64	3507701.959	1580037358	13422.12063	0.999100441
256	3508865.502	38029158684	58349.46978	0.972518436
512	3512659.654	62831444668	71825.50954	0.950809041
1200	3333269.7	1.38371E+11	248053.5943	0.925993491

Table 2: Metrics for Batch Gradient Descent

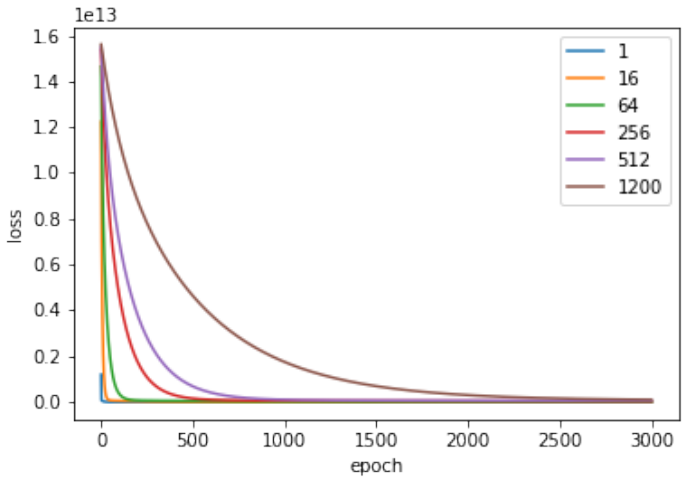

Figure 3: Loss Curves w.r.t different Batch Sizes

We observe that for same number of epochs and learning rate, increasing the batch size lowers the performance of the model. Also, the model reaches the minima of the loss function faster for low batch sizes.

2.3 Effect of Regularization:

2.3.1 Ridge Regression

Loss function, E,

$$E_i = \frac{(t_i - \mathbf{w}^T\mathbf{x}_i)^2}{2} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{5}$$

Algorithm 3 Ridge Stochastic Gradient Descent

1: **for** $i = 1, 2, \ldots, numEpochs$ **do**
2: **for** $j = 1, 2, \ldots, N$ **do**
3: $\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)}(1 - \eta\lambda) + \eta(t_j - \mathbf{w}^{(\tau)T}\mathbf{x}_j)\mathbf{x}_j$

The above algorithm was run for 2000 epochs with $\eta = 0.001$ and $\lambda = 0.00001$. Following are the metrics:-

- weights(last term is the bias term) = $\{171.908176, 8182.98321, 1741192.42, 196244.127, 6.10342863, 86937.3828, 3507607.21\}$

- Test Mean Squares Error (MSE) = 11964.426878570397
- Test Mean Absolute Error (MAE) = 73.58402444090112

2.3.2 Lasso Regression

Loss function, E,

$$E_i = \frac{(t_i - \mathbf{w}^T\mathbf{x}_i)^2}{2} + \frac{\lambda}{2}\sum_{j=1}^{7}|w_j| \tag{6}$$

Algorithm 4 Lasso Stochastic Gradient Descent

1: **for** $i = 1, 2, \ldots, numEpochs$ **do**
2: **for** $j = 1, 2, \ldots, N$ **do**
3: $\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)}(1 - \eta\lambda sgn(\mathbf{w})) + \eta(t_j - \mathbf{w}^{(\tau)T}\mathbf{x}_j)\mathbf{x}_j$

The above algorithm was run for 2000 epochs with $\eta = 0.001$ for different values of λ.

λ	w_1	w_2	w_3	w_4	w_5	w_6
0.00001	166.0472649	8015.611709	1741579.412	196221.947	0.0662556	87120.47506
0.001	165.7870833	8015.579535	1741579.406	196221.948	0.065331222	87120.17808
0.1	139.7434203	8012.363206	1741578.831	196222.0503	0.009345135	87090.46075
1	0.176035775	7976.550257	1741568.844	196221.9073	0.055484122	86905.67689

Table 3: Weights for Lasso Stochastic Gradient Descent

λ	Bias (w_7)	Test MSE	Test MAE
0.00001	3507642.945	0.470345279	0.542783808
0.001	3507642.944	0.504443786	0.561734955
0.1	3507642.839	10.75294464	2.56204714
1	3507641.972	425.3686995	15.10234064

Table 4: Metrics for Lasso Stochastic Gradient Descent

We observe that if λ is sufficiently large, some of the coefficients w_i (w_1 and w_5 for example) are driven to zero, leading to a sparse model.

2.4 Feature Engineering

2.4.1 Feature Selection

We can observe from the sub task 1 that the features x_1,x_2,x_3 and x_6 are highly correlated. So, we ran BGD(Batch Size = 64, learning rate = 0.1, No. of epochs = 10,000) by keeping only one of the above features. Following are the results-
Values of weights when $\{x_3, x_4, x_5\}$ are selected: $\{1668523.48, 190639.831, -559.483728, 3508045.55\}$

Features	Test MSE	Test MAE	Correlation Coeffcient
x_1, x_4, x_5	2.49e+11	162774.919625	0.768447
x_2, x_4, x_5	1.04e+11	144921.710807	0.917523
x_3, x_4, x_5	5.39e+08	6891.236300	0.999687
x_6, x_4, x_5	2.60e+11	163558.825629	0.757532

Table 5: Metrics for Feature Selection

2.4.2 Improvising Loss function

Loss function, E_{batch},

$$E_{batch} = \frac{1}{2k}\sum_{i=1}^{k}(\{t_i - \mathbf{w}^T\mathbf{x}_i\}^2 + \frac{\lambda_1}{2}\mathbf{w}^T\mathbf{w} + \frac{\lambda_2}{2}\sum_{j=1}^{7}|w_j|) \tag{7}$$

Where k is the batch size.

The Improvised BGD algorithm was run for 10000 epochs with $k = 64$,$\eta = 0.1$,$\lambda_1 = 0.00001$ and $\lambda_2 = 0.1$. Following are the metrics:-

- weights(last term is the bias term) = $\{174.030475, 8018.56168, 1741573.03, 196222.260, 0.146144321, 87125.4564, 3507642.41\}$
- Test Mean Squares Error (MSE) = 2.7475217524272595
- Test Mean Absolute Error (MAE) = 1.0117551751167049

2.5 Effect on sample distribution

For each sampling rate, the SGD algorithm was run for 3000 epochs with $\eta = 0.001$. Following are the metrics-

Sampling Rate	Test MSE	Test MAE	Train MSE	Train MAE
0.1	11454303012	32132.63258	671394530.5	20644.07614
0.2	33851.54908	145.69076	24514.25223	120.0897597
0.3	877.3771085	23.52652658	724.8300044	20.90003795
0.4	26.5674703	4.065504524	23.36455465	3.79204111
0.5	2.53083996	1.253410792	2.217059668	1.181059262

Table 6: Metrics for Sampling Rate

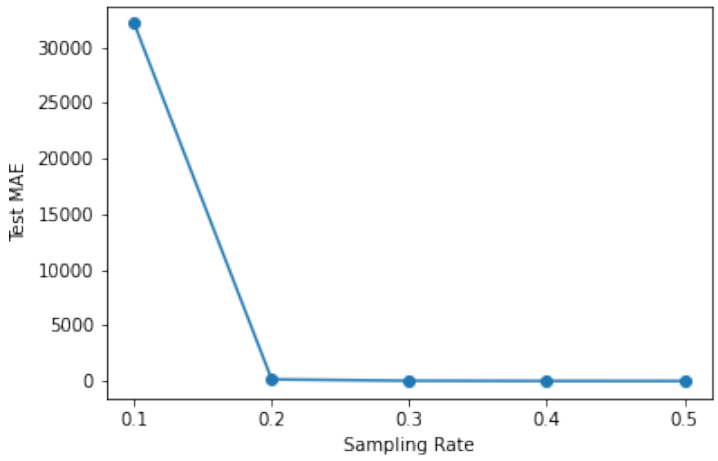

Figure 4: Test MAE vs Sampling Rates

Observations-

- For lower sampling rates, the model is overfitting the training data.
- As the sampling rate increases, overfitting and loss decrease drastically.

2.6 Variance in the noise

Variance in the noise is estimated using Maximum likelihood and Least Squares.

$$t = y(\mathbf{w}, \mathbf{x}) + \epsilon \tag{8}$$

Where ϵ is a zero mean Gaussian random variable with precision (inverse variance) β. For a data set of inputs $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$ and corresponding target values $\mathbf{t} = \{t_1, t_2, \ldots, t_N\}$, the likelihood function can be written as,

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^T\mathbf{x}_n, \beta^{-1}) \tag{9}$$

Maximizing the log of likelihood function w.r.t $\mathbf{w}$ gives us $\mathbf{w}_{ML}$,

$$\mathbf{w}_{ML} = (\mathcal{X}^T\mathcal{X})^{-1}\mathcal{X}^T\mathbf{t} \tag{10}$$

Here, $\mathcal{X}$ is a $N \times D$ matrix (D = Dimension). It's n^{th} row contains the vector x_n. Maximizing the log of likelihood function w.r.t β gives us β_{ML}^{-1} or variance in the noise,

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{i=1}^{N}\{t_n - \mathbf{w}_{ML}^T\mathbf{x}_n\}^2 \tag{11}$$

Using the above strategy we get the value of variance in the noise as $5.4548751265798506e-14$

2.7 Optimal Weights

The optimal learning rate was found to be 0.001 for SGD and 0.1 for BGD and mini-batch Lasso. We ran SGD with learning rates 0.1 and 0.01 as well. However, the algorithm was overshooting in both the cases. We experimented with a bunch of different learning algorithms, number of epochs and batch sizes. The results that we obtain are as follows:-

Batch Size	Epochs	Test MSE	Train MSE
64	10000	3.51E-18	3.58E-19
64	15000	3.64E-18	1.36E-18
64	20000	3.74E-18	1.37E-18
256	10000	1.03E-08	9.07E-09
256	15000	4.33E-16	3.78E-16
256	20000	4.61E-18	1.46E-18
512	10000	0.006322671	0.005560543
512	15000	2.26E-07	1.98E-07
512	20000	8.04E-12	7.08E-12

Table 7: Batch Gradient Descent

Batch Size	Epochs	Test MSE	Train MSE
64	10000	1.83E-11	1.58E-11
64	15000	1.83E-11	1.58E-11
64	20000	1.83E-11	1.58E-11
256	10000	1.06E-08	9.26E-09
256	15000	1.45E-12	1.18E-12
256	20000	1.41E-12	1.14E-12
512	10000	0.006322805	0.005560657
512	15000	2.26E-07	1.99E-07
512	20000	1.36E-11	1.18E-11

Table 8: Mini Batch Lasso Stochastic Gradient Descent

Epochs	Test MSE	Train MSE
3000	0.000175614	0.000156272
5000	1.51E-11	1.35E-11
10000	6.28E-17	6.62E-17

Table 9: Stochastic Gradient Descent

From tables 6,7 and 8 we conclude that the most optimal weights are obtained when we use Batch Gradient Descent with a batch size of 64, the number of epochs equal to 10,000 and the learning rate equal to 0.001.
The weights obtained are $\{173.886504, 8015.81744, 1741579.27, 196221.880, 0.0740764821, 87128.3500, 3507642.95\}$

3 Part 2: Multivariate Logistic Regression

For the following part, we have converted the labels 2,9 to 1,0 for ease of reference in the classification model. We have used the sigmoid activation function here:

$$\sigma(x) = \frac{1}{1 + \mathbf{e}^{-x}} \tag{12}$$

As well as the logarithmic loss function:

$$E_b = \frac{-1}{N} \sum_{i \in b} \{y_i log(\hat{y}_i) - (1 - y_i) log(1 - \hat{y}_i)\} \tag{13}$$

3.1 Effect of Batch Size

3.1.1 Stochastic Gradient Descent

Loss function, E_n,

$$E_n = -\{t_n \log(\hat{y}_n) - (1 - t_n) \log(1 - \hat{y}_n)\} \tag{14}$$

where t_n is the target and $\hat{y}_n (= \sigma(\mathbf{w}^T \mathbf{x}))$ The below algorithm was run for 300 epochs with

Algorithm 5 Stochastic Gradient Descent

1: **for** $i = 1, 2, \ldots, numEpochs$ **do**
2: **for** $j = 1, 2, \ldots, N$ **do**
3: $\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)} + \eta(t_j - \hat{y}_j)\mathbf{x}_j$

$\eta = 0.001$. Following are the metrics:-

- Accuracy = 0.9865602687946241
- Log-Loss (Train data) = 0.018246019485861856
- Log-Loss (Test data) = 0.043903067856299306
- Bias = −0.22923424245224297

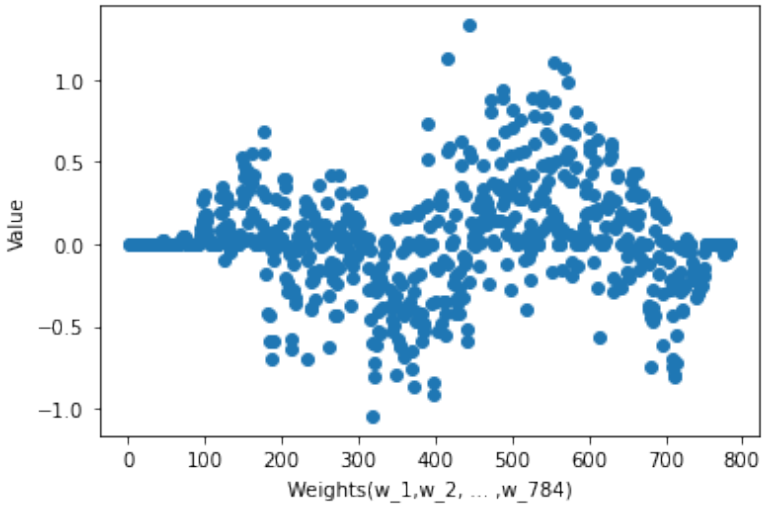

Figure 5: Values of Weights for Stochastic Gradient Descent

3.1.2 Batch Gradient Descent

Loss function, E_{batch},

$$E_{batch} = \frac{-1}{k} \sum_{i \in batch} \{t_i log(\hat{y}_i) - (1 - t_i) log(1 - \hat{y}_i)\} \tag{15}$$

Where, k is the batch size.

Algorithm 6 Batch Gradient Descent

1: **for** $i = 1, 2, \ldots, numEpochs$ **do**
2: **for** $j = 1, 1 + k, 1 + 2k \ldots, N$ **do**
3: $\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)} + \frac{\eta}{k} \sum_{l=j}^{j+k} \{t_l - \hat{y}_l\} \mathbf{x}_l$

The above algorithm was run for 300 epochs with $\eta = 0.001$ for different batch sizes. Following are the metrics:-

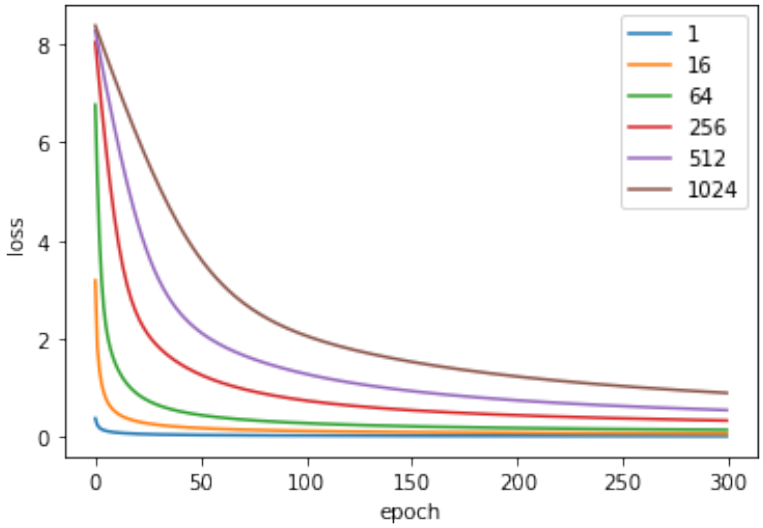

Figure 6: Loss vs Epochs for different batch sizes

Batch Size	Train Loss	Test Loss	Test Accuracy
1	0.022924	0.052230	0.982360
16	0.073688	0.082325	0.976900
64	0.149638	0.144640	0.957581
256	0.340371	0.335718	0.908862
512	0.553059	0.545334	0.859723
1024	0.901166	0.879431	0.782444

There is a stark difference between different batch sizes, which is evident from Figure 6. For the batch size of 1, i.e., taking the whole training data and processing it together, the loss starts from a low value and stabilises after that as the number of epochs increases. On the other hand, for a batch size of 1024, we see that the model starts from a very high value of log and slowly decreases as we increase the number of epochs. This is an intuitive observation, as in the first case, we had worked on the entire data together and hence had the inter-relations between the data right from the beginning, as compared to a large batch size where the inter-relations between batches were gradually built up.

3.2 Effect of Regularization

3.2.1 Ridge Regression

Loss function, E,

$$E_i = -\{t_n \log(\hat{y}_n) - (1 - t_n) \log(1 - \hat{y}_n)\} + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{16}$$

Algorithm 7 Ridge Stochastic Gradient Descent

1: **for** $i = 1, 2, \ldots, numEpochs$ **do**
2: **for** $j = 1, 2, \ldots, N$ **do**
3: $\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)}(1 - \eta\lambda) + \eta(t_j - \hat{y}_j)\mathbf{x}_j$

The above algorithm was run for 300 epochs with $\eta = 0.001$ and $\lambda = 0.01$. Following are the metrics:-

- Accuracy : 0.9802603947921041
- Log-loss (Test) in Ridge : 0.05936809652356804
- Log-loss (Train) in Ridge : 0.052379591199922755
- Bias of RidgeSGD Model: -0.06243293655302035

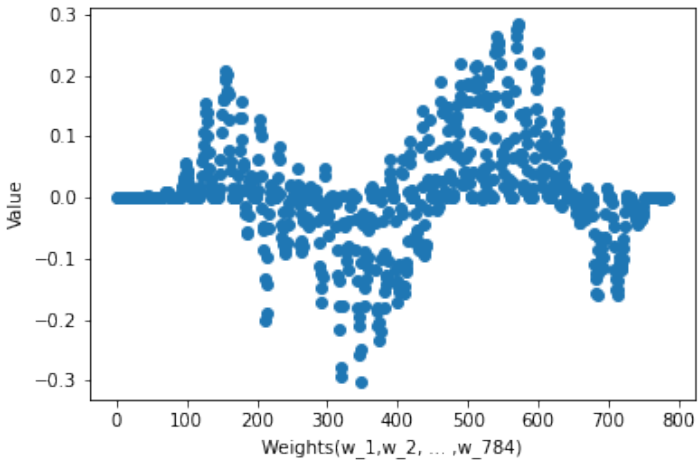

Figure 7: Values of Weights for Ridge Regression

3.2.2 Lasso Regression

Loss function, E,

$$E_i = -\{t_n \log(\hat{y}_n) - (1 - t_n) \log(1 - \hat{y}_n)\} + \frac{\lambda}{2}\sum_{j=1}^{7} |w_j| \tag{17}$$

The above algorithm was run for 300 epochs with $\eta = 0.001$ for $\lambda = 0.001$, 0.01 and 0.1.

Algorithm 8 Lasso Stochastic Gradient Descent

1: **for** $i = 1, 2, \ldots, numEpochs$ **do**
2: **for** $j = 1, 2, \ldots, N$ **do**
3: $\mathbf{w}^{(\tau+1)} := \mathbf{w}^{(\tau)}(1 - \eta\lambda sgn(\mathbf{w})) + \eta(t_j - \hat{y}_j)\mathbf{x}_j$

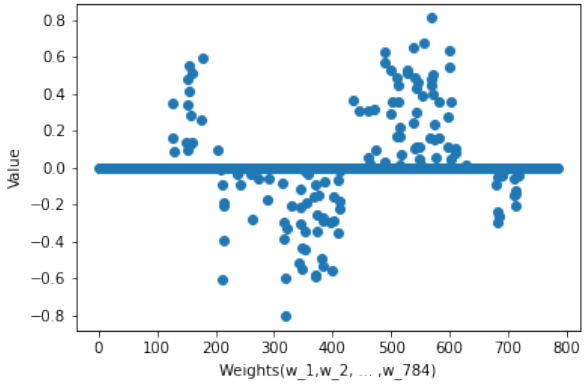

Figure 8: Values of Weights for Lasso Regression with λ= 0.001

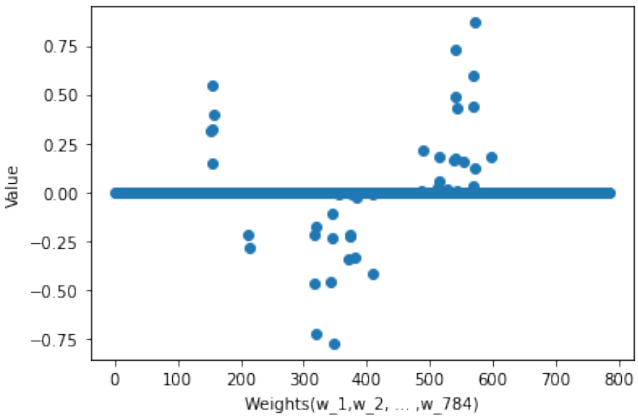

Figure 9: Values of Weights for Lasso Regression with λ= 0.01

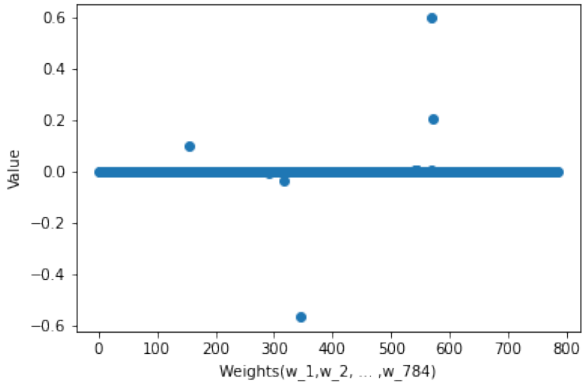

Figure 10: Values of Weights for Lasso Regression with λ= 0.1

We observe that if *lambda* is sufficiently large, some of the coefficients w_i are driven to zero, leading to a sparse model.

3.3 Highest accuracy and Visualization of wrong Classifications

The best accuracy achieved from the different Logistic Regression methods mentioned above are listed as follows:

Method	Accuracy
Stochastic Gradient Descent	0.986560
Batch Gradient Descent (b=1)	0.982360
Ridge Regression($\lambda = 0.001$)	0.984880
Lasso Regression ($\lambda = 0.001$)	0.980680

Hence, the best overall accuracy achieved is that of Ridge Gradient Descent with a value of 0.984880

Some of the samples where the model prediction are going wrong are plotted as follows. It is evident that these cases are outliers that are far away from other samples in their respective labels.

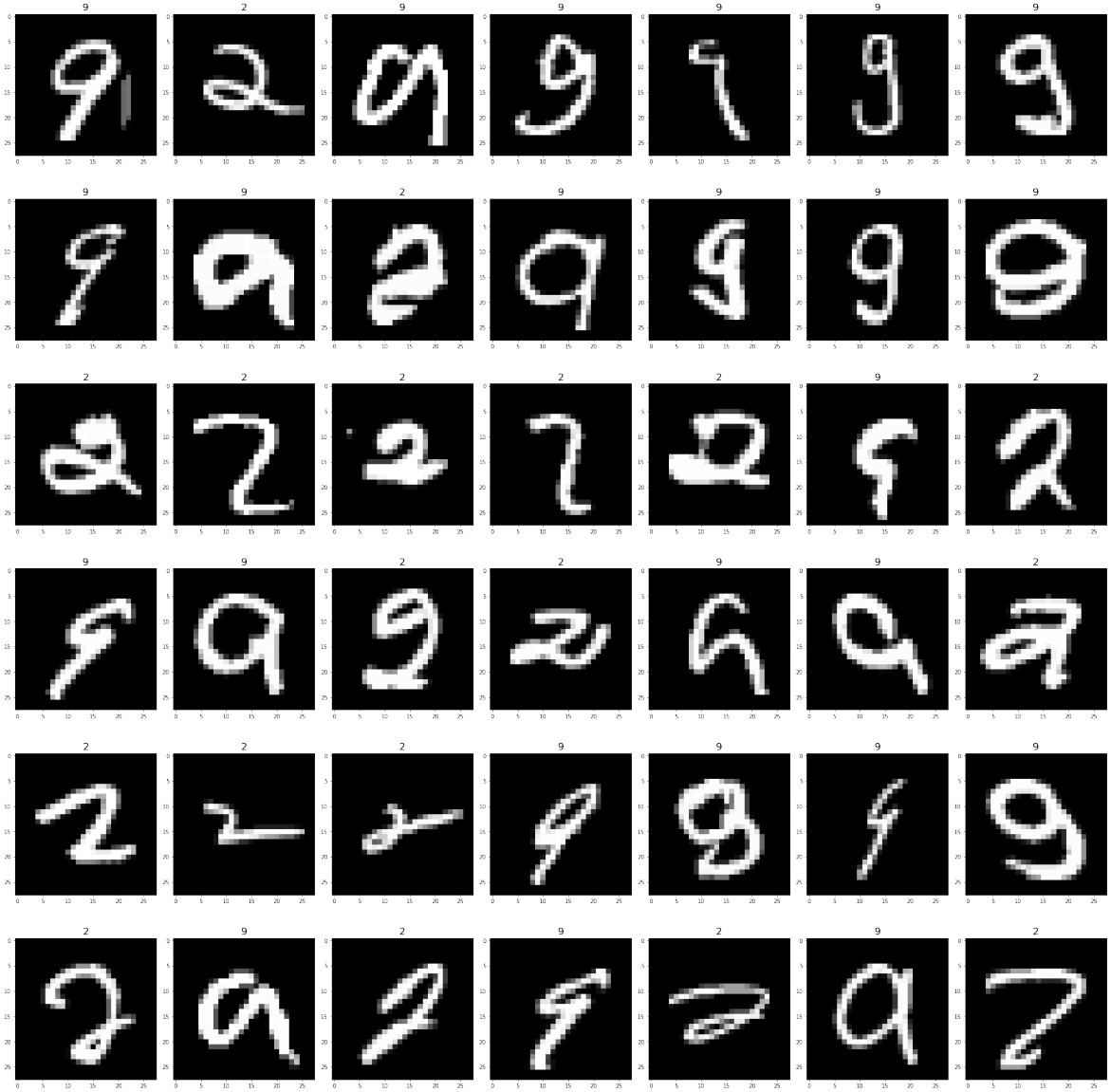

Figure 11: Samples leading to false predictions from the model

3.4 Optimal Weights

The optimal learning rate was found to be 0.001 for SGD. For BGD, we observe that as the ratio of number of iterations vs batch size increases, the accuracy also increases. We experimented with a bunch of different learning algorithms, number of epochs and batch sizes. Out of the various batch sizes we tried, 40 turned out to be the best choice in terms of speed vs number of epochs. Now, we a batch size of 40 and a learning rate of 0.001, we ran our BGD model for different number of epochs. The results that we obtain are as follows:-

Epochs	Train Loss	Test Loss	Test Accuracy
400	0.100990	0.101872	0.970181
480	0.092253	0.095133	0.973121
560	0.085420	0.090182	0.974801
640	0.079862	0.086370	0.975640
720	0.075230	0.083325	0.976900
800	0.071311	0.080823	0.976900

For Ridge Regression, at 900 epochs and 0.001 learning rate, we observed the variation of accuracy with the value of lambda as the following:

Lambda	Test Accuracy	Log-loss
0.01	0.980260	0.059368
0.001	0.984880	0.042451
0.0001	0.987400	0.044145
0.00001	0.986560	0.048177

When the Lasso Regression model was run by varying the number of epochs and the value of lambda, we observed that the log loss error was decreasing slowly with the rise in number of epochs, however the change was much more evident when lambda was changed.

Epochs	Lambda	Train Loss	Test Loss	Test Accuracy
400	0.001	0.0.043569	0.058547	0.980680
600	0.001	0.043392	0.058508	0.979840
400	0.01	0.125316	0.136513	0.964301
600	0.01	0.125303	0.136569	0.964301

Concluding this discussion, we can say that after optimization, the best accuracy we were able to achieve was 0.987400.

4 Part 3: Multi-class Multivariate Logistic Regression

The posterior probability of a class $\mathcal{C}_k$ is given by the softmax function,

$$p(\mathcal{C}_k|\mathbf{x}) = y_k(\mathbf{x}) = \frac{exp(a_k)}{\sum_j exp(a_j)} \tag{18}$$

Where,

$$a_k = \mathbf{w}_k^T \mathbf{x} \tag{19}$$

Here, $\mathbf{x}$ is the feature vector and $\mathbf{w}_k$ is the weights for the k^{th} class.
Let $f(\mathbf{x})$ be the class predicted by the model for feature vector $\mathbf{x}$. Then,

$$f(\mathbf{x}) = argmax_j\{y_j(\mathbf{x})\} \tag{20}$$

We define the likelihood function as,

$$p(\mathbf{T}|\mathbf{w}_1, \ldots, \mathbf{w}_K) = \prod_{n=1}^{N}\prod_{k=1}^{K} y_{nk}^{t_n k} \tag{21}$$

Here, the target vector $\mathbf{t}_n$ is the one-hot vector for the feature vector $\mathbf{x}_n$ belonging to class $\mathcal{C}_k$ having all the elements 0 except the k^{th} element which is 1. $y_{nk} = y_k(\mathbf{x}_n)$, and T is an $N \times K$ matrix of target variables with elements t_{nk}.
Taking negative log of the likelihood function gives the cross entropy loss function,

$$\mathbf{E}(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk}\log(y_n k) \tag{22}$$

Gradient of the error function w.r.t $\mathbf{w}_j$ then will be,

$$\nabla_{\mathbf{w}_j}\mathbf{E}(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{n=1}^{N}(y_{nj} - t_{nj})\mathbf{x}_n \tag{23}$$

Experimentation-

- The train image feature vector of size 784 is scaled down by a scaling factor of 255.
- The train image labels are converted into their respective one - hot vectors using a **self-defined** one-hot encoding function.
- Using Equation 23, BGD algorithm is run with batch size = 64, learning rate = 0.1, and No. of epochs = 100.

metrics-

- Accuracy : 0.9138261521793483
- log-loss(Eq. 22) (Train): 0.25764945
- log-loss(Eq. 22) (Test): 0.31804164

Note: The confusion matrix is made using **self-defined** function. For converting into heat map, the seaborn library is used.

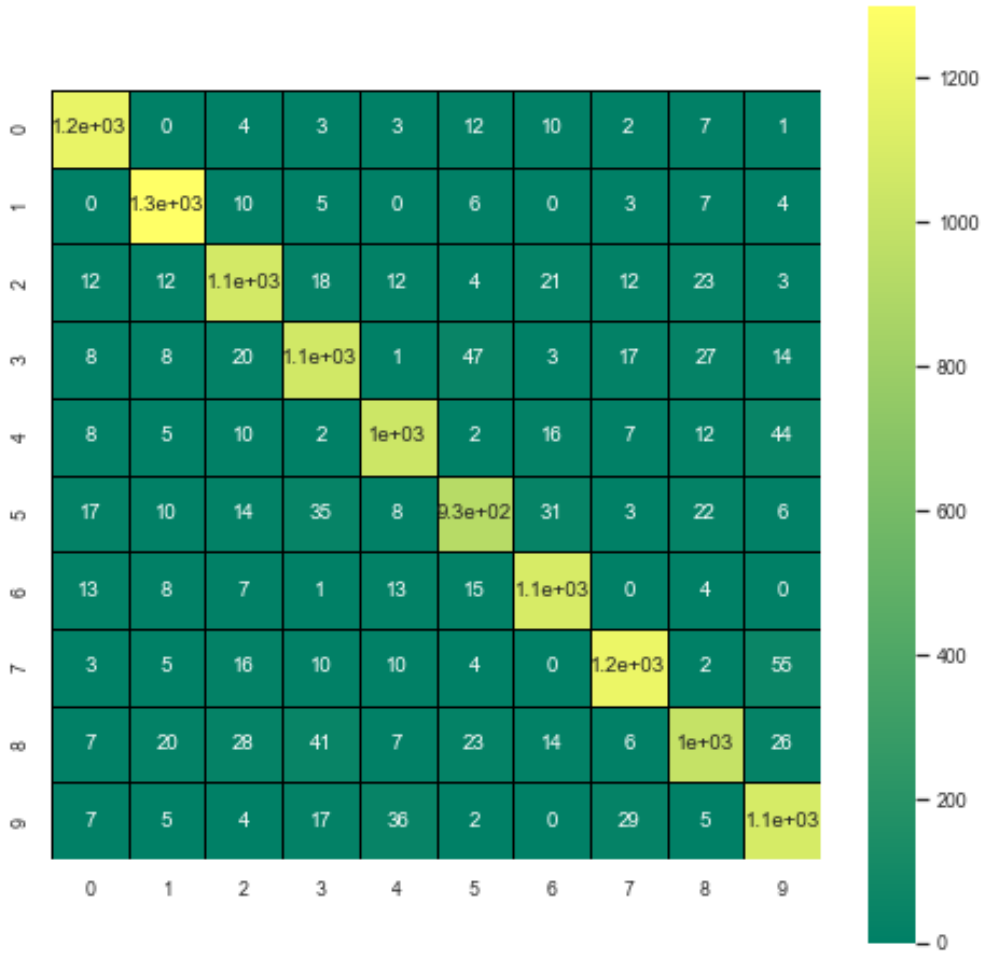

Figure 12: Confusion Matrix Heat Map