CSE 253: Neural Networks

Homework Assignment 1

Logistic and Softmax Regression

Saksham Sharma (PID: A53220021)
Department of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093
sas111@eng.ucsd.edu

Kriti Aggarwal (PID: A53214465)
Department of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093
kriti@eng.ucsd.edu

Abstract

We explore the use of regression models such as Logistic and Softmax for the task of handwritten digit recognition. MNIST is a dataset of handwritten digits, originally comprising of 60000 training examples and 10000 test examples. However, for the experiment we conducted, we sampled 20000 training examples and 2000 test examples. Working with the logistic regression model, we built a binary classification model on '2' and '3' as well as '2' and '8' digits. To handle over-fitting we tried using and analyzing both L1 and L2 regularization, along with early stopping. We used the gradient descent technique for optimizing on the Cost Function, and experimented on different momentum parameters for accelerated convergence. Our approach achieves an error of 2.12% on the test set, and 1.99% on the holdout set, while executing the model at speeds comparable to the fastest algorithms on the dataset. Subsequently, while working with the softmax regression model, we performed 10-way classification on the MNIST dataset. Using the above mentioned enhancements of regularization and momentum, we achieved an accuracy of 89.2% on test set and 93.15% on hold-out set.

1 Derive the gradient for Logistic Regression

Question) We need the gradient of the cost function, Equation 3, with respect to the parameter vector *w*. Show that for the logistic regression cost function, the gradient is:

$$-\frac{\delta E^n(w)}{\delta w_j} = (t^n - y^n)x_j^n$$

Show work.

Answer)

$$-E^n(w) = t^n ln y^n + (1-t^n)\ln(1-y^n) \qquad \text{(given)}$$

On differentiating the above equation w.r.t. w_j , we get:

$$-\frac{\delta E^n(w)}{\delta w_j} = \frac{t^n}{\sigma(w^T x^n)}\sigma(w^T x^n)\big(1-\sigma(w^T x^n)\big)x_j^n - \frac{1-t^n}{1-\sigma(w^T x^n)}\sigma(w^T x^n)\big(1-\sigma(w^T x^n)\big)x_j^n$$

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_j} = [t^n(1-y^n) - (1-t^n)y^n]x_j^n$$

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_j} = [t^n - t^n y^n - y^n + t^n y^n] x_j^n$$

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_j} = (t^n - y^n) x_j^n \qquad \text{Q.E.D.}$$

2 Derive the gradient for Softmax Regression

Question) For softmax regression cost function, Equation 7, show that the gradient is:

$$-\frac{\delta E^n(w)}{\delta w_{jk}} = (t_k^n - y_k^n) x_j^n$$

Show work. **Note:** Here we are departing from Bishop's notation. w_{jk} is the weight from unit j to unit k, not vice-versa.
Hint: Recall your logarithm rules, such as ln(a b) = ln a - ln b. The hardest part here is the derivative of the softmax. Most of the derivations are already done for you in Bishop Chapter 6. You just have to fill in any missing steps.

Answer)

$$-E^n(w) = \textstyle\sum_{k=1}^{c} t_k^n ln y_k^n \qquad \text{(given)}$$

On differentiating it w.r.t. w_{jk}, we get:

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_{jk}} = \frac{\delta}{\delta w_{jk}} [t_k^n ln y_k^n + \sum_{k' \neq k} t_{k'}^n ln y_{k'}^n]$$

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_{jk}} = \frac{t_k^n}{y_k^n} \frac{\delta y_k^n}{\delta w_{jk}} + \sum_{k' \neq k} \frac{t_{k'}^n}{y_{k'}^n} \frac{\delta y_{k'}^n}{\delta w_{jk}}$$

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_{jk}} = \frac{t_k^n}{y_k^n} \frac{[\left(\sum_{k'} \exp\left(a_{k'}^n\right)\right) \exp(a_k^n) x_j^n - (\exp(a_k^n))^2 x_j^n]}{[\sum_{k'} \exp(a_{k'}^n)]^2} + \sum_{k' \neq k} \frac{t_{k'}^n}{y_{k'}^n} \frac{[-(\exp(a_{k'}^n))(\exp(a_k^n)) x_j^n]}{[\sum_{k''} \exp(a_{k''}^n)]^2}$$

(using quotient rule)

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_{jk}} = t_k^n (1 - y_k^n) x_j^n - \sum_{k' \neq k} t_{k'}^n y_k^n x_j^n$$

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_{jk}} = [t_k^n - t_k^n y_k^n - \sum_{k' \neq k} t_{k'}^n y_k^n] x_j^n$$

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_{jk}} = [t_k^n - \sum_{k'=1}^{c} t_{k'}^n y_k^n] x_j^n$$

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_{jk}} = [t_k^n - y_k^n \sum_{k'=1}^{c} t_{k'}^n] x_j^n$$

$$\Rightarrow -\frac{\delta E^n(w)}{\delta w_{jk}} = [t_k^n - y_k^n] x_j^n$$

(because $\sum_{k'=1}^{c} t_{k'}^n = 1$)

Q.E.D.

3 Logistic Regression via Gradient Descent

3.1 Introduction

Logistic regression is a classification technique, but it is called "regression" because it is used to fit a continuous variable: the probability of the category, given the data. In the given problem we have to classify the handwritten digits given in MNIST dataset. In this first task, we have to classify the images for two categories: **2's and 3's**. And then we have to repeat the same procedure for the categories: **2's and 8's**. Logistic regression can be modeled as a single neuron reading in an input vector $(1,x) \in \mathbb{R}^{d+1}$ and parameterized by weight vector $w \in \mathbb{R}^{d+1}$. d is the dimensionality of the input, and we tack on a '1' at the beginning for a bias parameter, w_0. The neuron outputs the probability that x is a member of class C1.

$$P(x \in \mathrm{C}_1|x) = g_w(x) = \frac{1}{1+\exp(-w^T x)} \quad (1)$$

$$P(x \in \mathrm{C}_2|x) = 1 - P(x \in \mathrm{C}_1|x) = 1 - g_w(x) \quad (2)$$

where $g_w(x)$ simply notes that the function g is parameterized by w. Note we identify the output y^n of the "network" for a particular example, x^n, with $g_w(x^n)$, i.e., $y^n = g_w(x)$. The cross entropy loss function for two categories over our training examples which measures how well our hypothesis function g does over the N data points is given as:

$$-E(w) = -\frac{1}{N}\sum_{n=1}^{N}\{t^n ln y^n + (1-t^n)\ln(1-y^n)\} \quad (3)$$

3.2 Methods

We have used logistic regression as a classification model to predict whether a grid of pixel intensities represents '2' or '3'. We initialized the weights randomly with floating point numbers between -0.005 to +0.005. To avoid over-fitting, we used **'Early stopping'** so as to make it better fit the training data with each iteration. We have also added **momentum** in the weight update to help accelerate minimization of the objective function and dampen oscillations. We experimented with different values of momentum parameters and found that mu = 0.9, it yielded the best results.

3.3 Results and Discussion

At the end of the experiment we reached the best results of accuracy = 97.89 on the test set data. and shown by the results below:

lambda: 0.0 (No regularization)
mu: 0.9 (momentum parameter)
Number of epochs :440
Percent Correct Train data : 97.3611111111
Percent Correct Test data : 97.8873239437
Percent Correct Hold data : 98.0198019802
Loss Function Holddata : 28.51073674
Loss Function Test data: 33.92730869
Loss Function Traindata : 276.499822436

Normalization of the input data: Since if the data points are not normalized, it takes a long time for the gradient descent to converge, hence we scaled down the inputs to a range 0 to1. To normalize the input vector, we divide every data point by 255.0.

Initial weights: We initialized the weights as random values and also experimented with using zeroes as the initial weights. We observed that the starting weights as zeroes or random values had little effect on the results.

Initial learning rate(eta): The selection of the initial learning rate is key to the working of gradient descent. If the value of the initial learning rate(eta) is too small, then it takes a large number of iterations for the gradient descent to converge. However, if eta is too large then there is a chance of skipping the solution or we may oscillate around the optimal solution. In our case, we experimented with different initial learning rates.

Table1: Percent correct classification for various initial learning rates

S.no.	**Initial Learning rate**	**No. of Epochs**	**Percent Correct Train**	**Percent Correct Test**	**Percent Correct Hold**
1	1e-05	440	97.3611111111	**97.8873239437**	98.0198019802
2	0.0001	12	95.5555555556	97.1830985915	98.0198019802
3	0.0002	11	95.4722222222	96.7136150235	98.0198019802
4	0.1	999	**100.0**	96.0093896714	96.7821782178
5	0.01	999	100.0	96.2441314554	96.0396039604
6	0.001	999	99.9722222222	96.9483568075	96.2871287129

From the above table, we can clearly see the importance of eta in the convergence and precision results. The values of eta greater or equal to 0.001(0.1,0.01,0.001) did not allow for early stopping and lead to the regression model over fitting. As observed in examples 4,5,6 where the training error is close to 0% while the test set error is close to 4%.

On the other hand, when the values of eta were very small as in example 1(eta=1e-05), the function converged slowly (in 440 epochs) and we reached the best accuracy. Though for middle range values eta in examples 2 and 3(eta), we reached the results very quickly (in 11-12 epochs) and with good enough accuracy (97.18% on test).

Finally, we got the best results for eta value 1e-05 which gave us the accuracy of 97.8873239437% on test set data.

Adaptive Learning rate: We used annealing to decay the learning rate over time.We experimented with various values of meta parameter T in 1/t decay and obtained the following results.

Table 2: Loss function for various values of metaparameter

S.no.	**Metaparameter(T)**	**No. of epochs**	**Loss Function Train**	**Loss Function Test**	**Loss Function Hold**
1.	1000	440	276.499822436	33.92730869	28.51073674
2.	2000	402	276.378536736	33.9183029679	28.5107039059
3.	5000	380	276.46083861	33.9180926658	28.5106403881
4.	10000	374	276.363963925	33.9137294496	28.5106603722

As we are increasing the meta parameter T, we are converging in fewer epochs and value of loss function on the test data is also decreasing.

We also experimented with exponential decay and obtained similar results.

Table 3: Results for 1/t decay and exponential decay

	1/t decay	**Exponential decay**
Meta Parameter(T)	1000	1000
No. of epochs	440	454
Loss function value for training data	276.499822436	276.404597468
Loss function value for testing data	33.92730869	33.9246341004
Loss function value for hold data	28.51073674	28.5107655584

Momentum:

We added a momentum term to add inertia to the motion in the weight update rule.

$$\Delta w_{ij}(t) = \mu_i \delta_i y_j + m \Delta w_{ij}(t-1) \qquad (4)$$

where m is the global parameter determined through trial and error and the parameter μ_i as 1.

Table 4: Percent correct classification for various momentum parameters

S.no.	Momentum parameter(m)	No. of epochs	Percent Correct Train	Percent Correct Test	Percent Correct Hold
1.	0.85	672	97.3888888889	97.8873239437	98.0198019802
2.	0.9	404	97.3611111111	97.8873239437	98.0198019802
3.	0.95	36	95.5277777778	96.9483568075	98.2673267327

The number of epochs in which the function converged dropped with the increasing value of the momentum parameter. But, we got the best results for the momentum parameter 0.9 where the function converged in just 404 epochs. While it took 440 epochs to converge before adding the momentum parameter. If the momentum is very high (0.95), then we can run past the optimal with big strides.

Below are the plots for ‘epochs vs percent correct’ and ‘epochs vs Loss Function’ for the momentum parameter 0.9

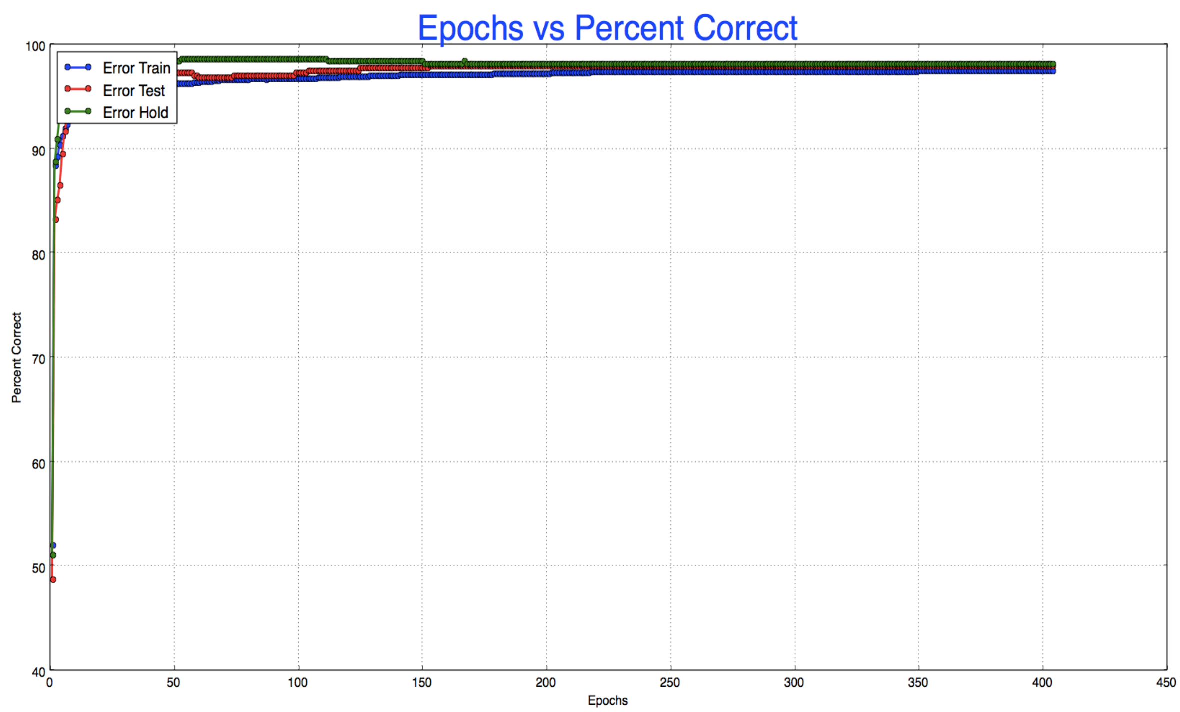

Fig. 3.1 Percent correct classification vs. epochs

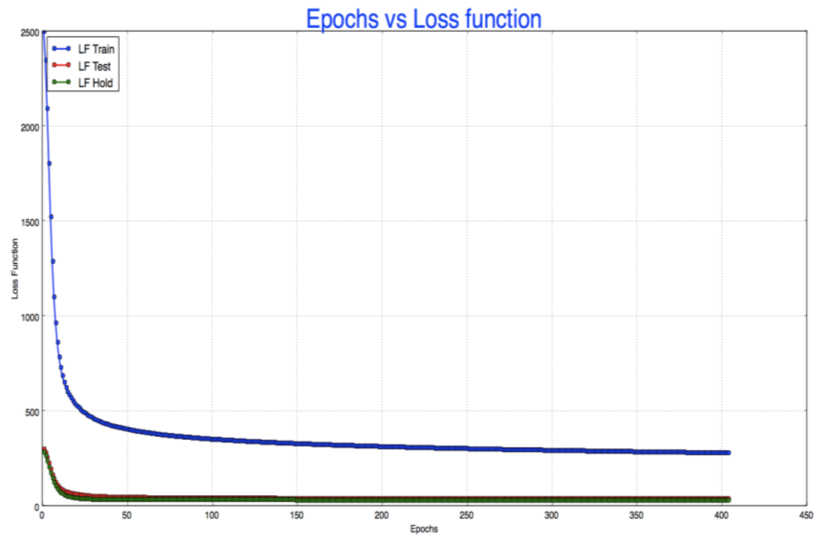

Fig. 3.2 Loss function vs. epochs

a) Here, we want you to Plot the loss function (E) over training for the training set and the hold-out set. The loss function should be computed over the entire training set after the epoch. I.e., after changing the weights, run every pattern through the network and compute the loss.

Testing on the test set is typically not available in the real world (and is considered "cheating" if the test set is available). However, since we aren't in the real world, but are stuck here in academia for the time being, let's check how well the hold-out set actually models the test set by plotting all three on the same plot. (1.5 points)

Plot of epochs vs loss function(E)

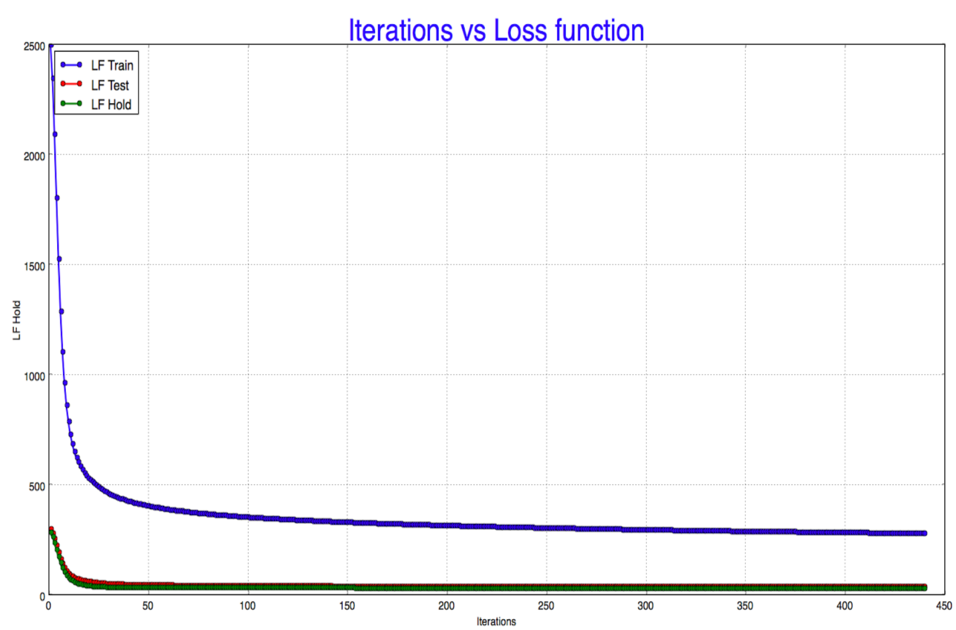

Fig. 3.3 Loss function vs. iterations

lambda: 0.0 (No regularization)
mu: 0.9 (momentum parameter)
Number of epochs/iterations :440
Percent Correct Train data : 97.3611111111
Percent Correct Test data : 97.8873239437
Percent Correct Hold data : 98.0198019802
Loss Function Holddata : 28.51073674
Loss Function Test data: 33.92730869
Loss Function Traindata : 276.499822436

If your classifier learns this task in one or two epochs, take a closer look by plotting these numbers over the entire sets every 1/10th epoch, or even more frequently, so that you can see gradual progress. To do this, you will change the weights after a "minibatch" of 10% of the patterns until you've gone through them all. We're looking for a relatively smooth curve. Does the hold-out set "work" as a good stand-in for the test set? Discuss. (1 point)

From the plot above, we observe there is a strong overlap in the results of the Loss Function vs Iteration for hold and test set data, which indicates that the hold set is a strong mimic or a good stand in to the real world test set data.

(b) Plot the percent correct classification over training for the training set, the hold-out set, and the test set, again on the same plot, again, after each epoch. This can be done simultaneously with computing the loss. Count a classification as correct if the input is a "2" pattern, and the output is >= 0.5, and vice-versa for the other pattern. (Again, if your classifier learns everything in one epoch, check progress more frequently). (1.5 points)

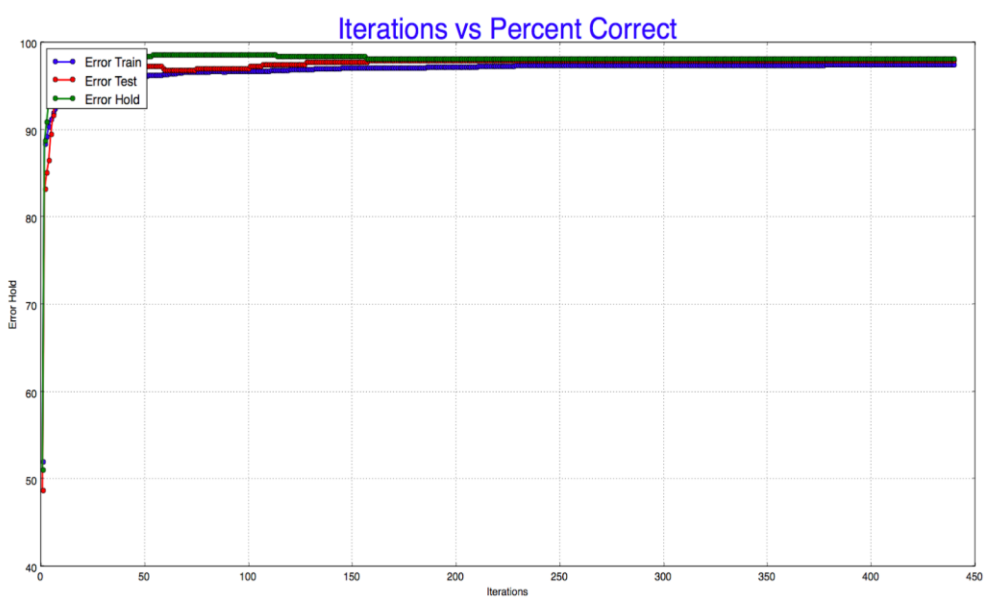

Fig. 3.4 Percent correct classification vs. iterations

(c) Repeat the above exercise for 2's versus 8's. (4 points)

lambda: 0.0 (No regularization)
mu: 0.9 (momentum parameter)

No. of epochs :999
Percent CorrectTrain data:97.6666666667
Percent Correct Test data:97.8102189781
Percent Correct Hold data:96.8
Loss Function Hold:21.5973913741
Loss Function Test:32.7547659859
Loss Function Train:268.068881235

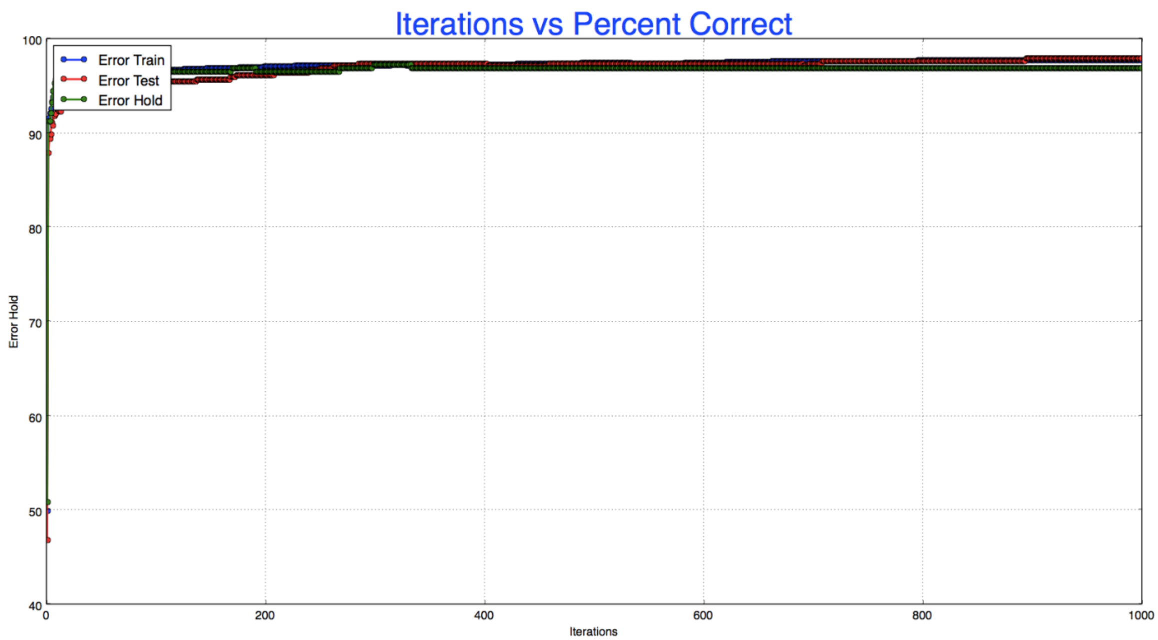

Fig. 3.5 Percent correct classification vs. iterations

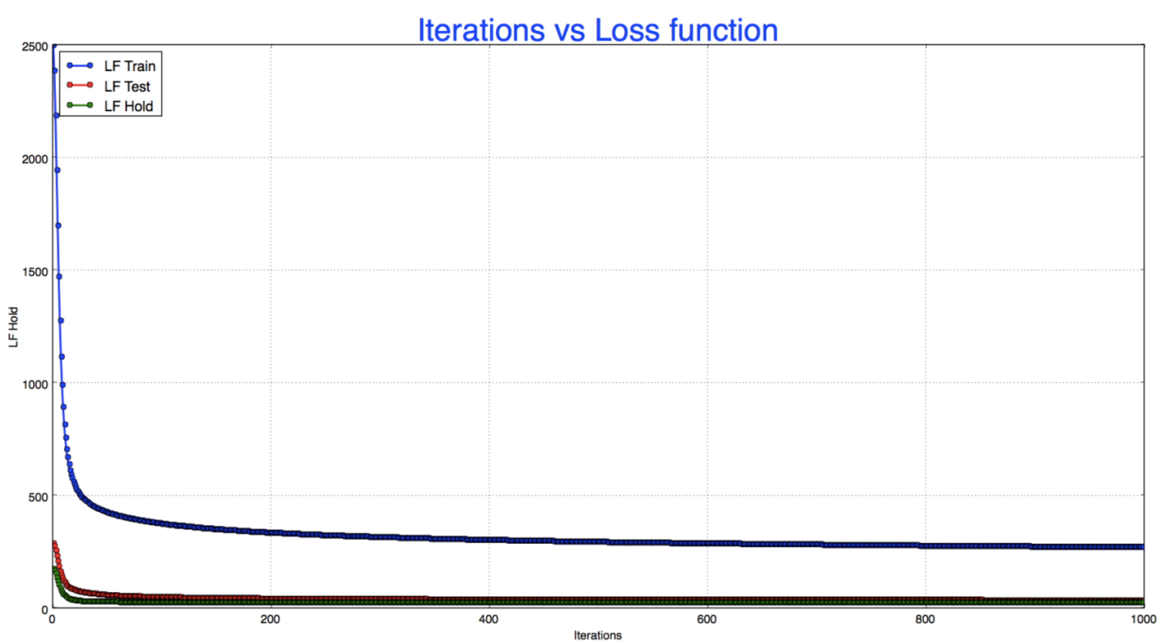

Fig. 3.6 Loss function vs. iterations

As in the above case of 2 vs 3 even in this case, the hold out set has a strong overlap with the test set and hence it's a good stand in for the test data set.

(d) Display the weights (without the bias) as an image for your two classifiers (2 vs. 3 and 2 vs. 8). Plot the difference of the weights between the two classifiers as an image

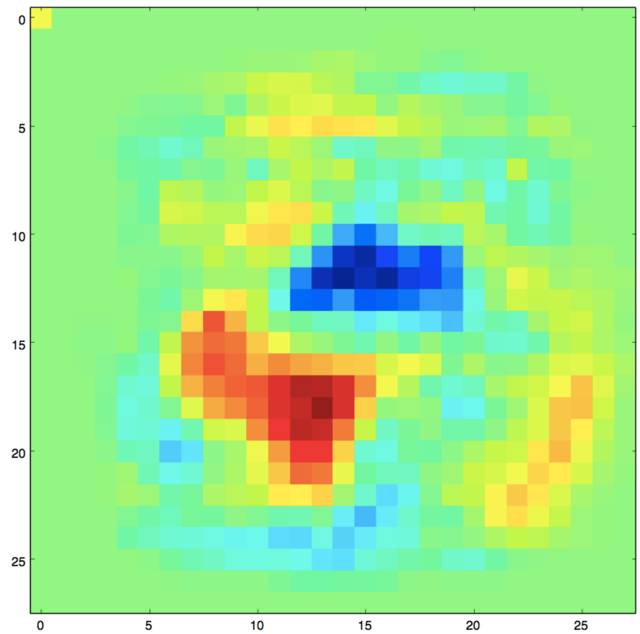

Fig. 3.7 Image of weights (2 vs. 3)

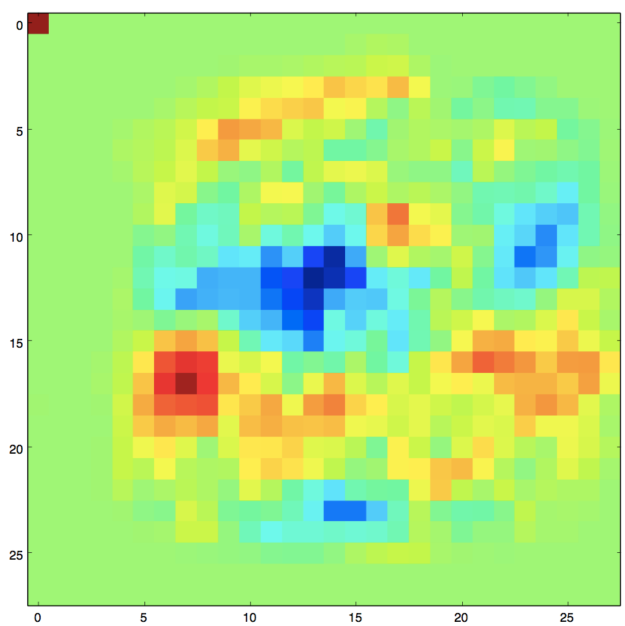

Fig. 3.8 Image of weights (2 vs. 8)

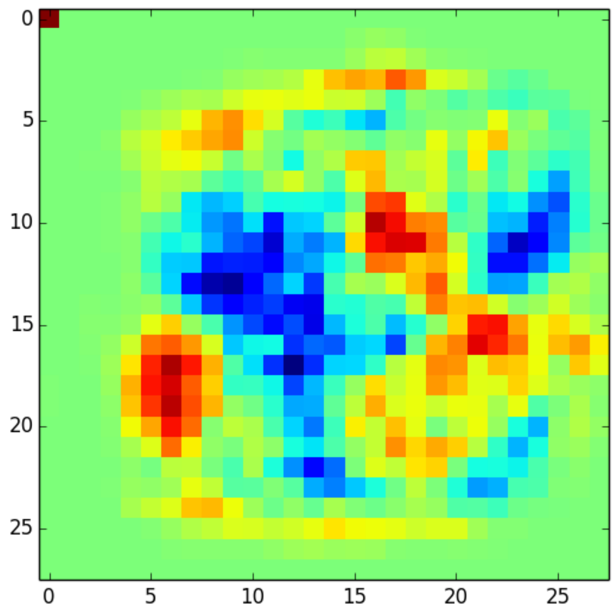

Fig. 3.9 Difference image

From the above figure, it is clear that the difference in the weight vectors of the logistic regression model for classification of ‘2’ and ‘3’, and ‘2’ and ‘8’lead to a weight vector representation for the classification of ‘3’ and ‘8’, as evident from the color distribution in the image above.

4 Regularization

4.1 Introduction

Regularization is a method for improving generalization. It is a way to "smooth" the model - to make it ignore idiosyncratic differences between items. It is basically the idea that we should penalize the model for being too complex. There by aiding in preventing over fitting.

4.2 Methods

We have used two regularization techniques to prevent over fitting, L1 and L2 regularization.

4.2.1 Gradient descent using L1 regularization

L1 regularization is a relatively common form of regularization, where for each weight w, we add the term λ|w| to the objective function. The property of L1 regularization is that it leads the weight vectors to become sparse during optimization.

In this case, the new objective function is given as follow:

$$J(w) = E(w) + \lambda C(w) \qquad (1)$$

$$C(w) = |w| = \sum_{i,j} |w_{i,j}| \qquad (2)$$

where C(w) is the complexity penalty and λ is the strength of regularization.

The derivative of the cost function is given by:

$$\partial C/\partial w = |w_i|/w_i \qquad (3)$$

Since, C(w) is differentiated for a particular value of w, w_i and all the other weights in the summations in (14) becomes zero.

Further simplifying this expression we get

$$\partial C/\partial w = \begin{cases} 1 & if\ x > 0 \\ 0 & if\ x = 0 \\ -1 & if x < 0 \end{cases} \quad (4)$$

Right hand Side of the above expression is essentially the sign function.

Hence, in this case, the update equation for the weights is given by:

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} \quad (5)$$

$$\Delta w^{(t)} = -\eta(\nabla E|_{w^{(t)}} + \lambda\, sign(w^{(t)})) \quad (6)$$

$$sign(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & if\ x = 0 \\ -1 & if x < 0 \end{cases} \quad (7)$$

We have experimented with different values of λ, in order to determine that value of λ which yields the best results.

4.2.2 Gradient descent using L2 regularization

L2 regularization is the most common form of regularization. It can be implemented by penalizing the squared magnitude of all parameters directly in the objective. In L2 regularization we add the term $1/2\lambda w^2$to the objective function, for every weight w in the network. Here, λ is the regularization strength. We have used gradient descent using L2 regularization to perform the classification between 2 and 3 in the MNIST database. In this case, the new objective function is given as follow:

$$J(w) = E(w) + \lambda C(w) \quad (8)$$

$$C(w) = ||w||^2 = \textstyle\sum_{i,j} {w_{i,j}}^2 \quad (9)$$

where C(w) is the complexity penalty and λ is the strength of regularization.

The derivative of the cost function is given by:

$$\partial C/\partial w = 2\lambda w_i$$

Since, C(w) is differentiated for a particular value of w, w_i and all the other weights in the summations in (14) becomes zero.

Hence, in this case, the update equation for the weights is given by the following equation, as the

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} \quad (10)$$

$$\Delta w^{(t)} = -\eta(\nabla E|_{w^{(t)}} + 2\lambda w^{(t)}) \quad (11)$$

We have experimented with different values of λ, in order to determine that value of λ which yields the best results.

4.3 Results and Discussion

(a) Derive the update term for logistic regression for gradient descent in J with respect to w, for L2 and L1regularization. All you have to do is figure out ∂C/∂w in each case. (2 points)

We have derived the expression ∂C/∂w for both L1 and L2 regularization in the methods used section 4.2.1 and 4.2.2.

(b) Implement these two methods, and train logistic regression for just 2 vs. 3, for several different values of λ, e.g., 0.01, 0.001, 0.0001, using early stopping. Plot the percent correct for different values of λ on the same graph over training. What do you observe? (2 points)

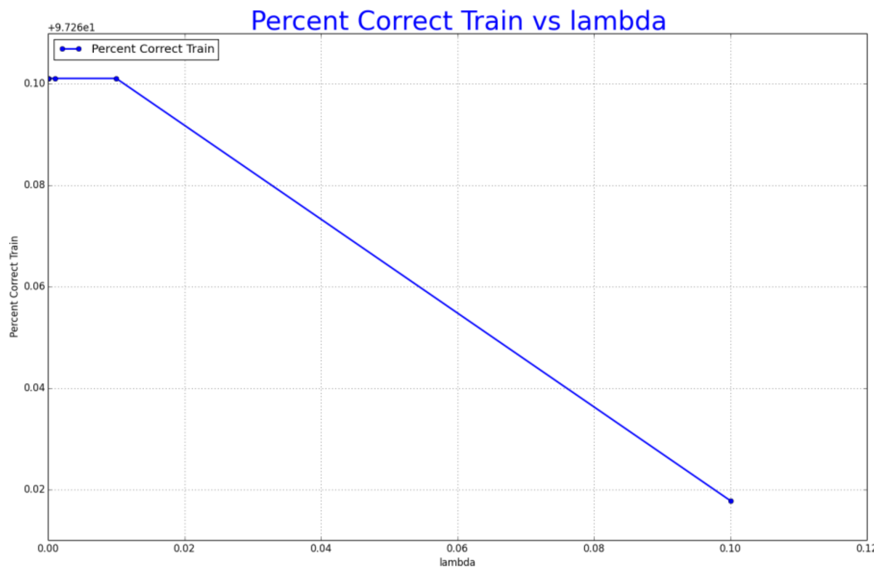

Fig. 4.1 Percent correct classification vs. lambda

We observe that the value of the percent correct decreases with increase in the value of the regularization parameter λ. This is because regularization prevents overfitting of the data by decreasing the weights by an extra factor.

(c) For the same points in training as you use to plot the percent correct, plot the length of the weight vector for each λ as a reality check. Discuss. (2 points)

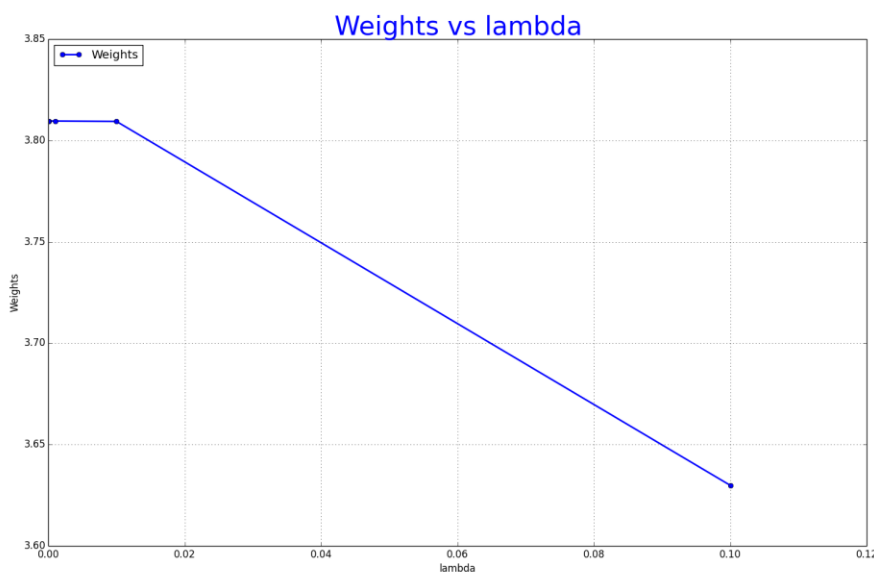

Fig. 4.2 Weights vs. lambda

The plot above shows that the values of weights are decreasing with the increase in the values of the regularization parameter λ. This conforms with our understanding of regularization since it penalizes the weights there by preventing the model to become overly complex.

(d) Make a plot of final test set error (y axis) versus the log of λ on the x-axis. Discuss. (2 points)

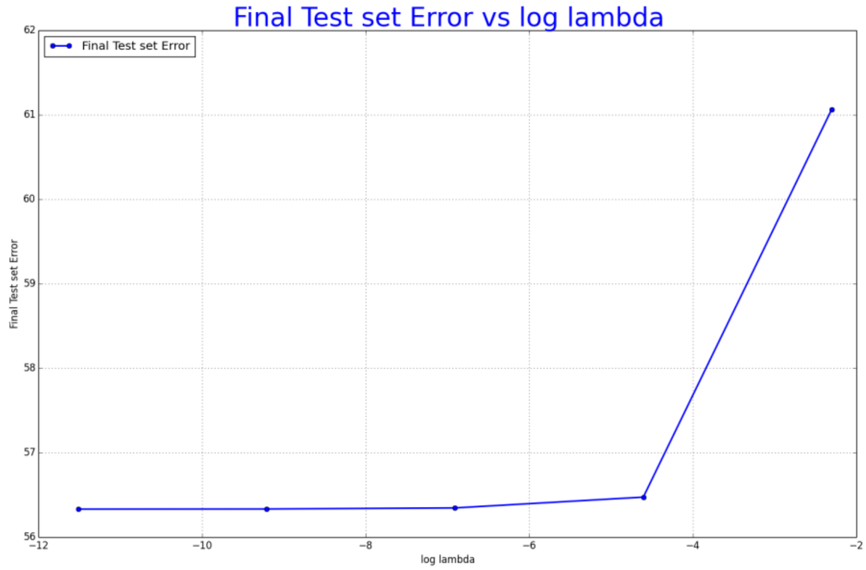

Fig. 4.3 Final test set error vs. lambda

The final test error is constant for low values of λ, after which it increases. The plot shows that the training data is emulating the test data well and so if the model is trained with lower values of λ, it performs better on test data as well.

(e) Plot the weights in each case as an image. What do you observe? (2 points)

λ = 0.1

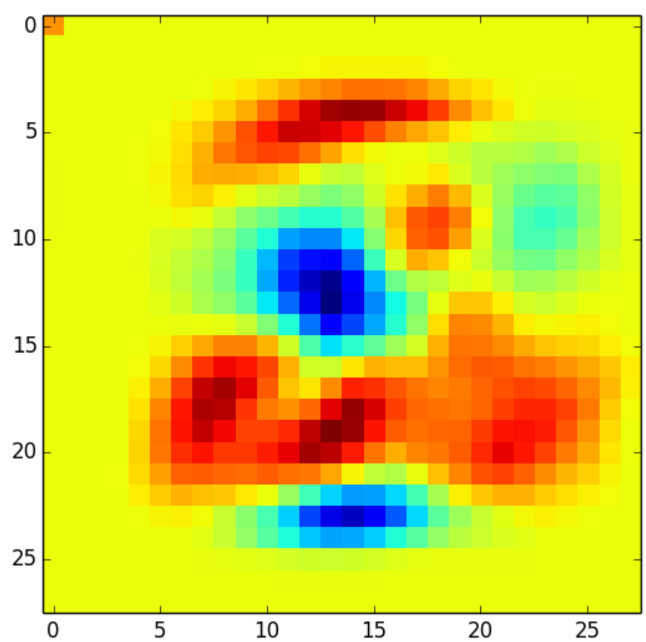

Fig. 4.4 Image of weights for λ = 0.1

λ= 0.01

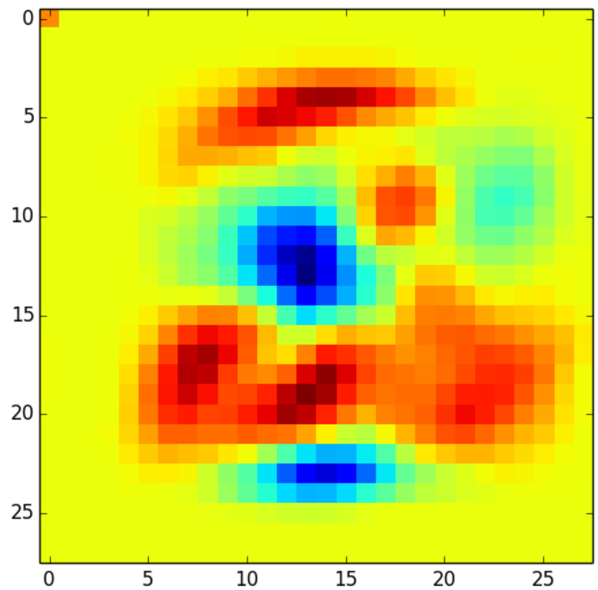

Fig. 4.5 Image of weights for λ= 0.01

λ= 0.001

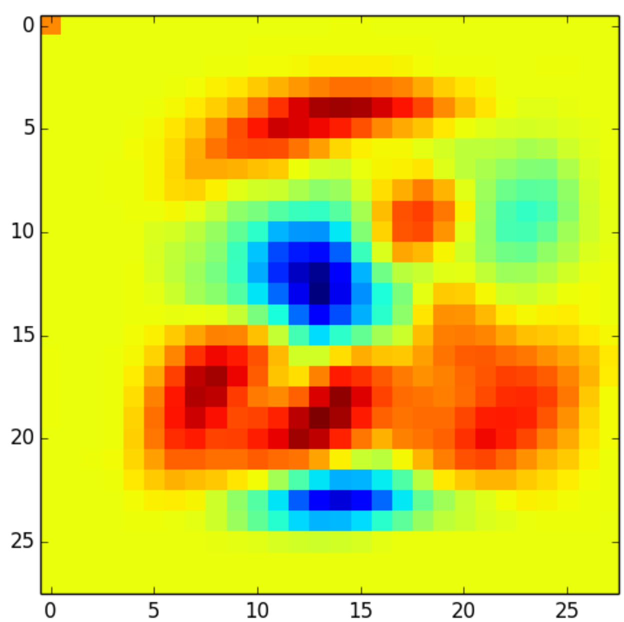

Fig. 4.6 Image of weights for λ= 0.001

λ= 0.0001

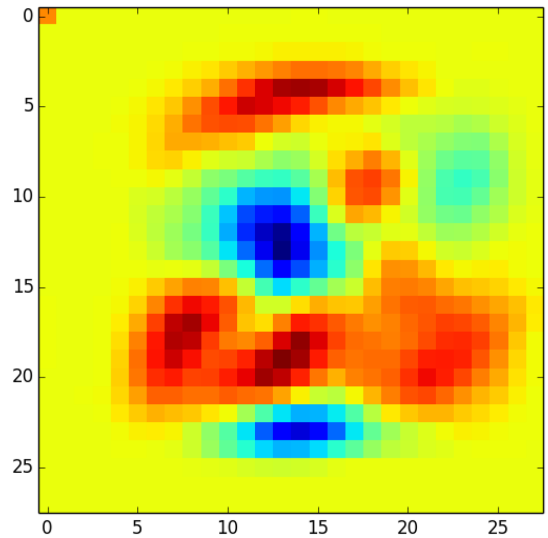

Fig. 4.7 Image of weights for λ= 0.0001

λ= 0.00001

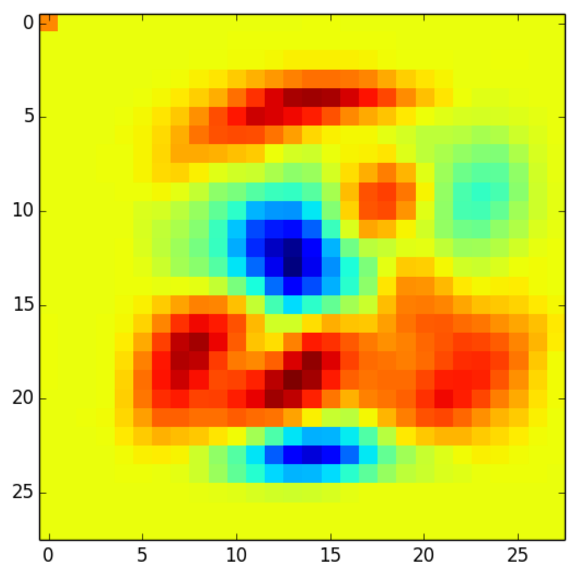

Fig. 4.8 Image of weights for λ= 0.00001

There is not much difference in the weight images for different values of the regularization parameter lambda, because the change in individual weights is minute. But the overall penalization to the weight vectors is significant as the value of lambda increase as can be seen by the plot in part c) between second norm of weights vector and lambda.

5 Softmax Regression via Gradient Descent

5.1 Introduction

Softmax regression is the generalization of logistic regression for multiple (c) classes. Given an input x^n, softmax regression will output a vector y^n, where each element, y_k^n represents the probability that x^n is in class k.

$$y_k^n = \frac{\exp(a_k^n)}{\sum_{k'} \exp(a_{k'}^n)} \quad (1)$$

$$a_k^n = w_k^T x^n \quad (2)$$

Here, a_k^n is called the net input to output unit y_k. Also, each output has its own weight vector w_k. In this model, the objective function to be minimized is called as cross-entropy cost function and is defined as:

$$E = -\frac{1}{N} \sum_n \sum_{k=1}^{c} t_k^n ln\, y_k^n \quad (3)$$

Here, t_k^n is the k^{th} component of target vector t^n for example n and N is the total number of training examples. The factor $\frac{1}{N}$ normalizes the error over different training set sizes.

The learning rule used to minimize the cross-entropy cross function is gradient descent iterative algorithm which is discussed in detail in the next section.

Our task in this problem is to perform 10-way classification of the MNIST database. In particular, we have to classify the handwritten images of the MNIST database into 10 classes, i.e. whether the digit is 0 or 1 or 2 and so on. For this we have to use gradient descent iterative algorithm. The gradient for softmax regression loss has already been derived in Section 2.

5.2 Methods

We have used the following methods to perform 10-way classification of the MNIST database using softmax regression via gradient descent:

5.2.1 Normalization

The first step is to normalize the input data. The normalization of the input data is required because if the data points are not normalized, then it takes a long time for the gradient descent to converge. In order to normalize the data, we have divided the input vector by 255.0 which is basically the maximum pixel intensity in the input image.

5.2.2 Simple gradient descent

In simple gradient descent, the update equation for the weights is given by:

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} \quad (4)$$

$$\Delta w^{(t)} = -\eta \nabla E|_{w^{(t)}} \quad (5)$$

In section 2 we have derived the gradient for softmax regression, which is given as follows:

$$-\frac{\delta E^n(w)}{\delta w_{jk}} = (t_k^n - y_k^n) x_j^n \quad (6)$$

For learning rate η, we have used the following annealing schedule:

$$\eta(t) = \frac{\eta(0)}{1+\frac{t}{T}} \quad (7)$$

where $\eta(0)$ is an initial learning rate, and T is a new metaparameter. We have experimented with different initial learning rates in order to determine that initial learning rate which yields the best results. Also, we have used T = 300 for our experiments.

5.2.3 Gradient descent using L1 regularization

Regularization is a method for improving generalization. It is a way to "smooth" the model - to make it ignore idiosyncratic differences between items. It is basically the idea that we should penalize the model for being too complex. It has already been discussed in detail in section 4.

We have used gradient descent using L1 regularization to perform the 10-way classification of the MNIST database. In this case, the new objective function is given as follow:

$$J(w) = E(w) + \lambda C(w) \tag{8}$$

$$C(w) = |w| = \sum_{i,j} |w_{i,j}| \tag{9}$$

where C(w) is the complexity penalty and λ is the strength of regularization.

In this case, the update equation for the weights is given by:

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} \tag{10}$$

$$\Delta w^{(t)} = -\eta(\nabla E|_{w^{(t)}} + \lambda\, sign(w^{(t)})) \tag{11}$$

$$sign(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & if\ x = 0 \\ -1 & if x < 0 \end{cases} \tag{12}$$

We have experimented with different values of λ, in order to determine that value of λ which yields the best results.

5.2.4 Gradient descent using L2 regularization

Next, we have used gradient descent using L2 regularization to perform the 10-way classification of the MNIST database. In this case, the new objective function is given as follow:

$$J(w) = E(w) + \lambda C(w) \tag{13}$$

$$C(w) = ||w||^2 = \sum_{i,j} {w_{i,j}}^2 \tag{14}$$

where C(w) is the complexity penalty and λ is the strength of regularization.

In this case, the update equation for the weights is given by:

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} \tag{15}$$

$$\Delta w^{(t)} = -\eta(\nabla E|_{w^{(t)}} + 2\lambda w^{(t)}) \tag{16}$$

We have experimented with different values of λ, in order to determine that value of λ which yields the best results.

5.2.5 Gradient descent using momentum

Finally, we used gradient descent using both L2 regularization and momentum to perform the 10-way classification of the MNIST database. By adding a momentum term to the gradient descent formula we can deal with the problem of widely differing eigen values. This adds inertia to the motion through weight space and smoothes out the oscillation [1]. In this case, the update equation for the weights is given by:

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} \tag{17}$$

$$\Delta w^{(t)} = -\eta\left(\nabla E|_{w^{(t)}} + 2\lambda w^{(t)}\right) + \mu \Delta w^{(t-1)} \tag{18}$$

where μ is called the momentum parameter. We have experimented with different values of μ, in order to determine that value of μ which yields the best results.

5.3 Results

5.3.1 Results for simple gradient descent

We have experimented with different initial learning rates in order to determine that initial learning rate which yields the best results. The results are given in Table 1.

Table 1: Results for simple gradient descent

S.no.	1	2	3
Learning rate	0.001	0.0001	0.00001
Loss function (Training set)	0.549181722789590	0.272853839941329	0.414279760141733
Loss function (Hold-out set)	0.758996122284960	0.267391967279090	0.378586647530538
Loss function (Test set)	1.04475747289314	0.369809991871692	0.500436484282204
Percent correct classification (Training set)	92.9333333333333	92.4666666666667	89.2888888888889
Percent correct classification (Hold-out set)	92.6000000000000	93.2500000000000	91.0500000000000
Percent correct classification (Test set)	87.7500000000000	88.8500000000000	86.5000000000000

Plots are as follow:-

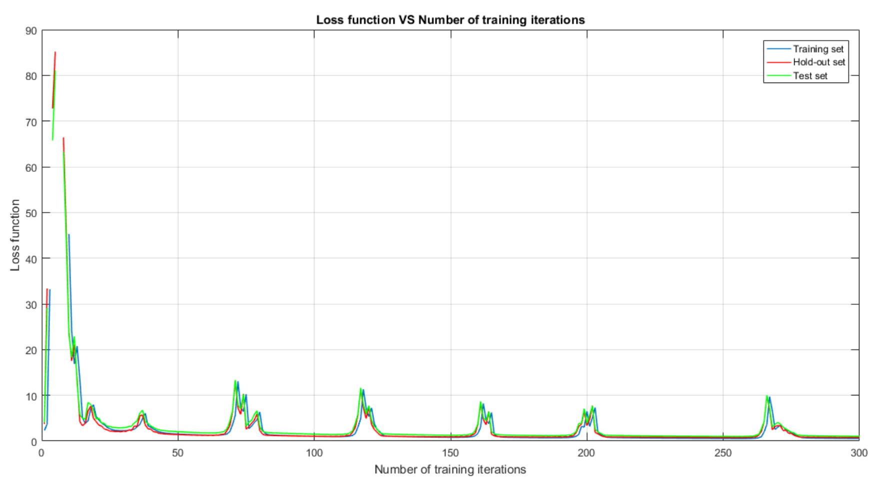

Fig. 5.1 Loss function vs number of training iteration ($\eta = 0.001$)

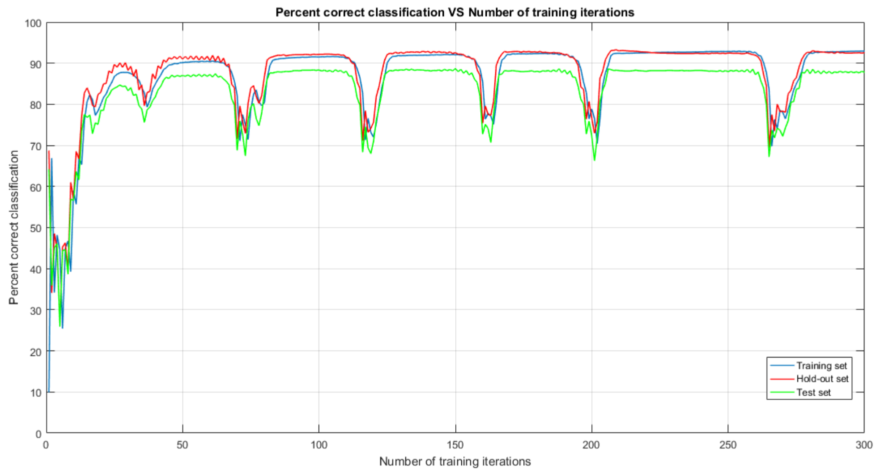

Fig. 5.2 Percent correct classification vs number of training iteration ($\eta = 0.001$)

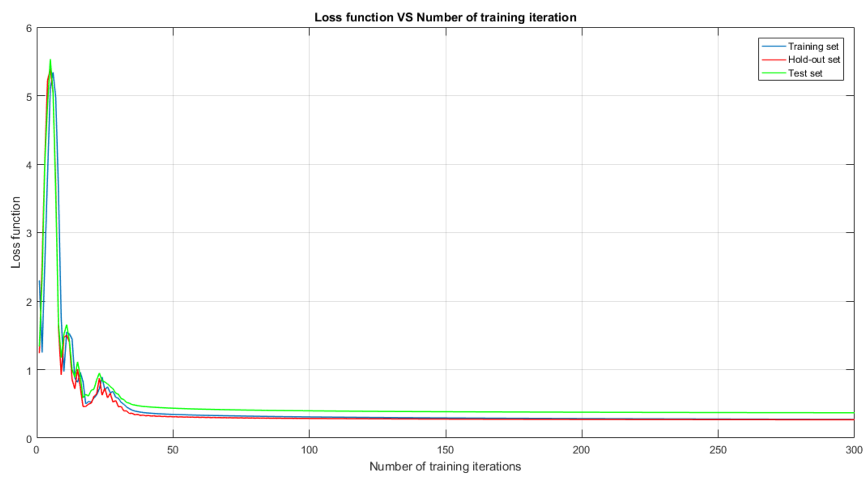

Fig. 5.3 Loss function vs number of training iteration ($\eta = 0.0001$)

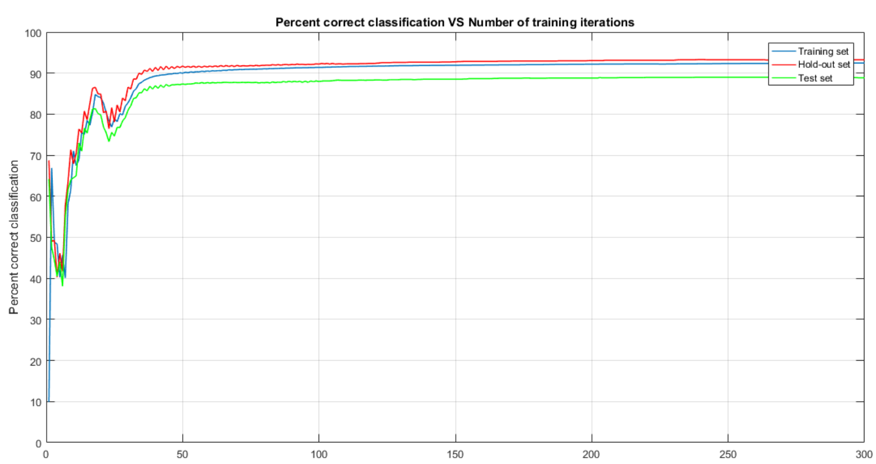

Fig. 5.4 Percent correct classification vs number of training iteration ($\eta = 0.0001$)

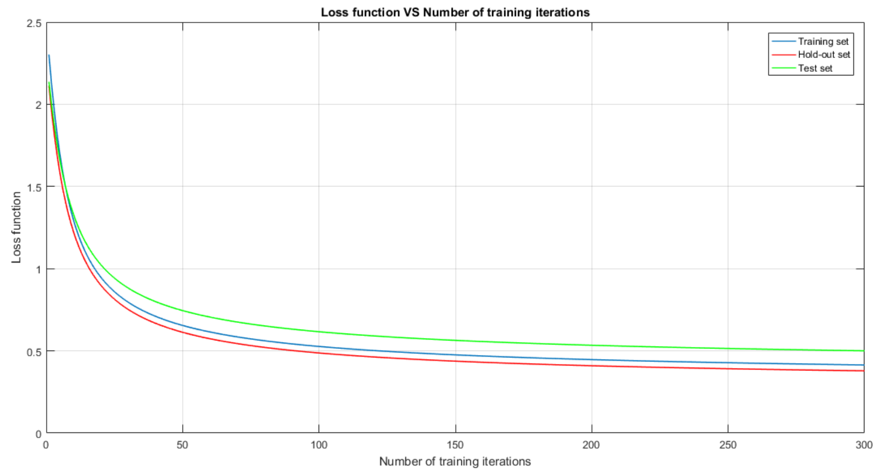

Fig. 5.5 Loss function vs number of training iteration ($\eta = 0.00001$)

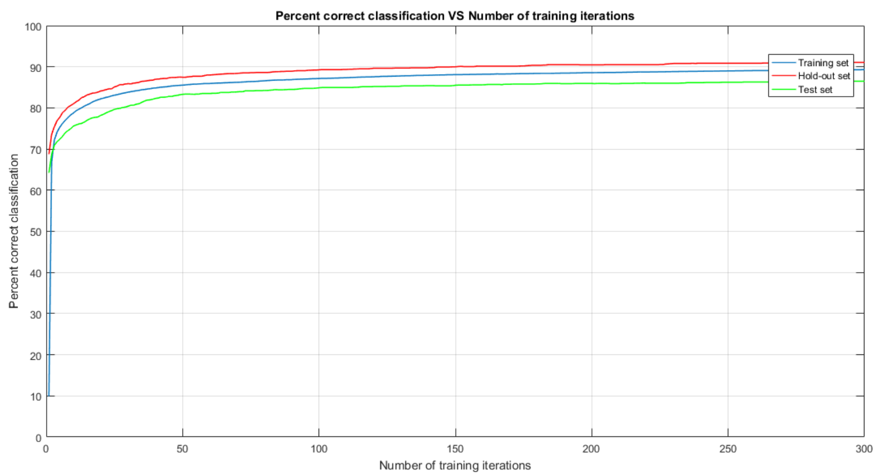

Fig. 5.6 Percent correct classification vs number of training iteration ($\eta = 0.00001$)

5.3.2 Results for gradient descent using L1 regularization

We have experimented with different values of λ, in order to determine that value of λ which yields the best results. The value of λ which yielded the best results was 10^{-7}. The results are given in Table 2.

Table 2: Results for gradient descent using L1 regularization

S.no.	1
$\boldsymbol{\lambda}$	10^{-7}
Loss function (Training set)	0.272910776420959
Loss function (Hold-out set)	0.267448903614924
Loss function (Test set)	0.369866928243499
Percent correct	92.4666666666667

classification (Training set)	
Percent correct classification (Hold-out set)	93.2500000000000
Percent correct classification (Test set)	88.8500000000000

Plots are as follow:-

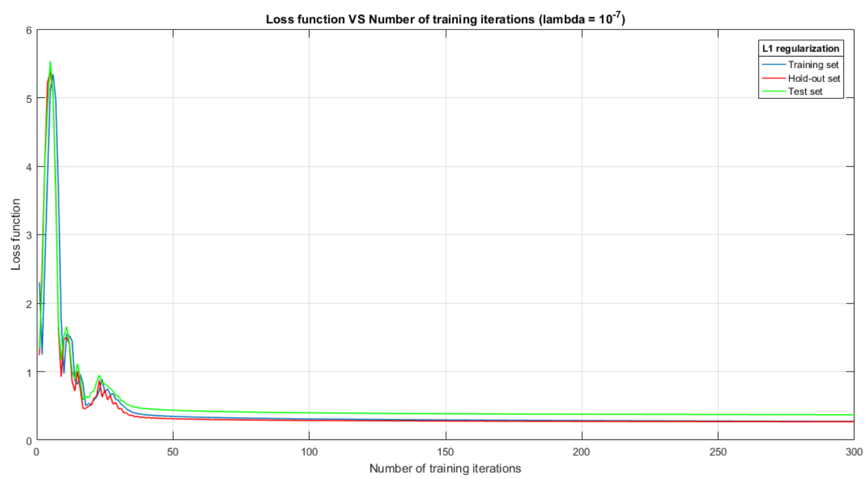

Fig. 5.7 Loss function vs number of training iteration ($\lambda = 10^{-7}$)

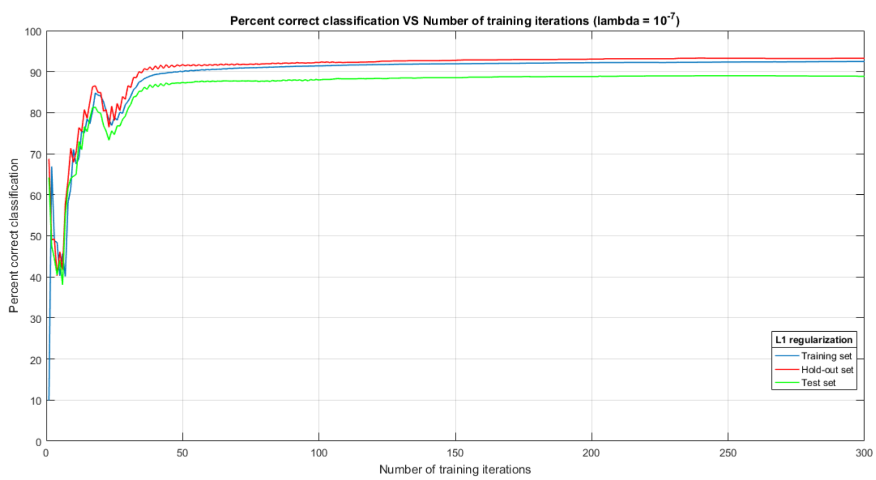

Fig. 5.8 Percent correct classification vs number of training iteration ($\lambda = 10^{-7}$)

5.3.3 Results for gradient descent using L2 regularization

We have experimented with different values of λ, in order to determine that value of λ which yields the best results. The value of λ which yielded the best results was 10^{-7}. The results are given in Table 3.

Table 3: Results for gradient descent using L2 regularization

S.no.	1
$\boldsymbol{\lambda}$	10^{-7}
Loss function (Training set)	0.272865196671867
Loss function (Hold-out set)	0.267403323939262
Loss function (Test set)	0.369821348544536
Percent correct classification (Training set)	92.4666666666667
Percent correct classification (Hold-out set)	93.2500000000000
Percent correct classification (Test set)	88.8500000000000

Plots are as follow:-

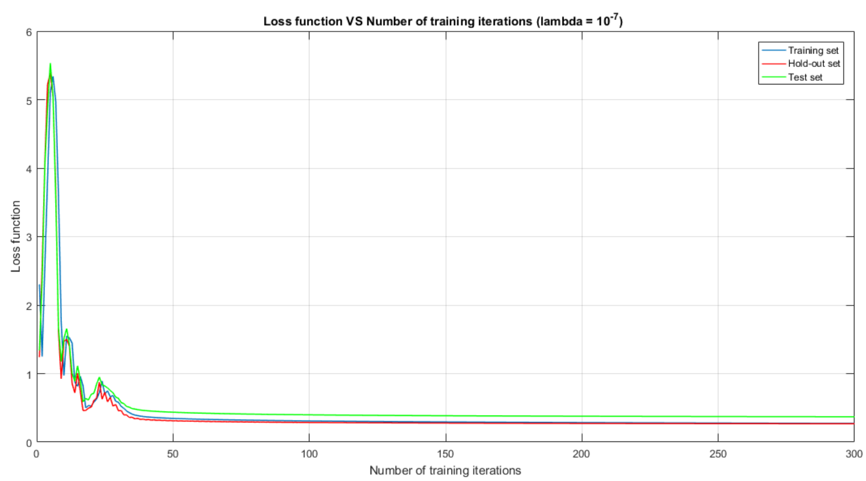

Fig. 5.9 Loss function vs number of training iteration ($\lambda = 10^{-7}$)

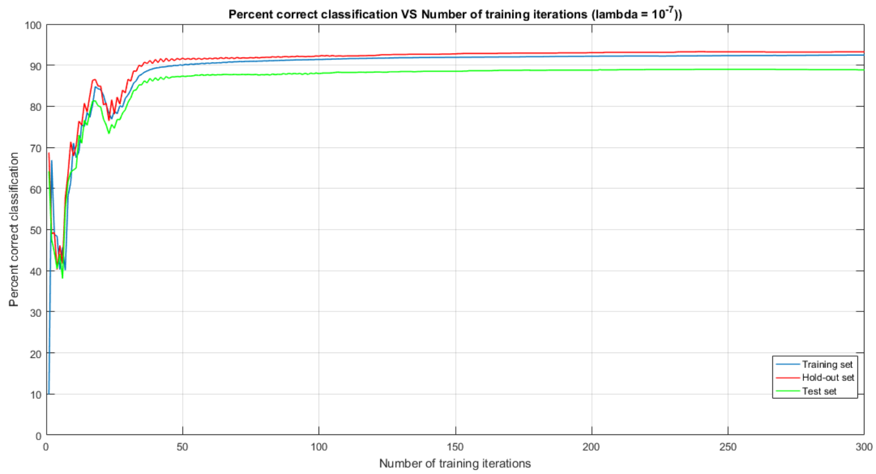

Fig. 5.10 Percent correct classification vs number of training iteration ($\lambda = 10^{-7}$)

5.3.4 Results for gradient descent using momentum

We have experimented with different values of μ, in order to determine that value of μ which yields the best results. The results are given in Table 4.

Table 4: Results for simple gradient descent

S.no.	1	2	3
Momentum parameter, μ	0.85	0.90	0.95
Loss function (Training set)	0.209152680970498	0.197267328841796	0.183423678421895
Loss function (Hold-out set)	0.276596198743607	0.292661207175386	0.360324718121156
Loss function (Test set)	0.371197884443466	0.385847365194169	0.457673258464795
Percent correct classification (Training set)	94.2055555555556	94.4333333333333	94.9777777777778
Percent correct classification (Hold-out set)	93.1500000000000	92.9500000000000	92.0500000000000
Percent correct classification (Test set)	89.2000000000000	88.9500000000000	88.7000000000000

Plots are as follow:-

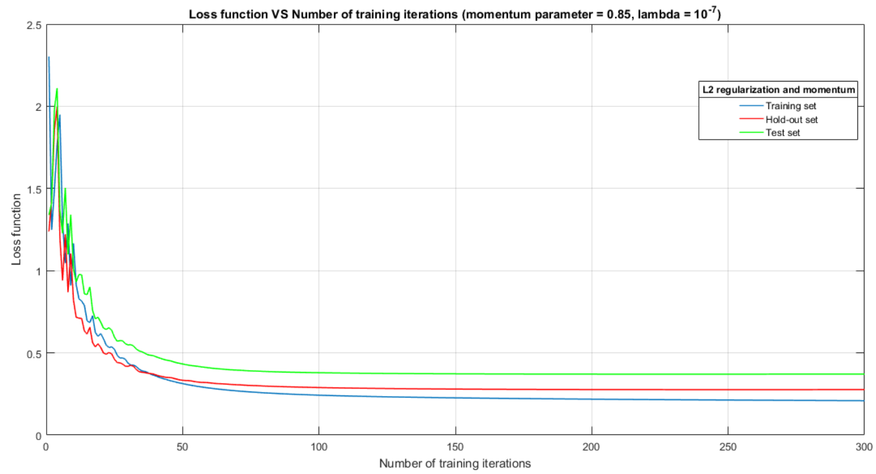

Fig. 5.11 Loss function vs number of training iteration ($\mu = 0.85$)

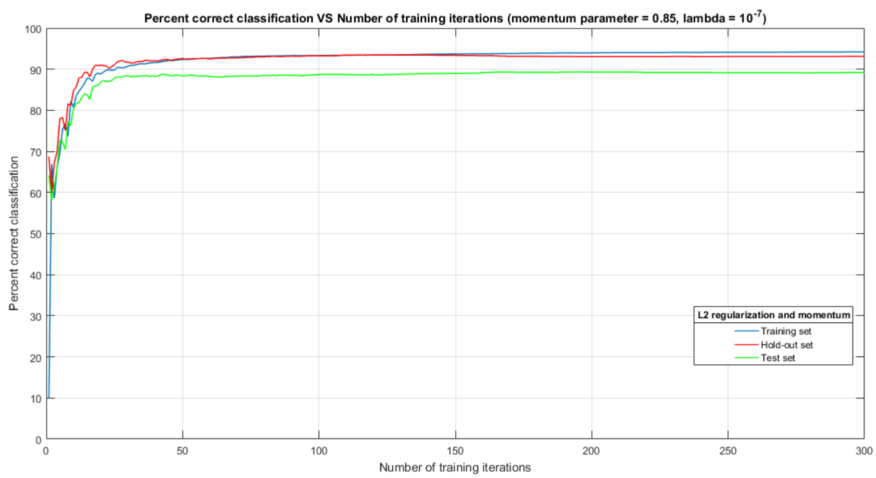

Fig. 5.12 Percent correct classification vs number of training iteration ($\mu = 0.85$)

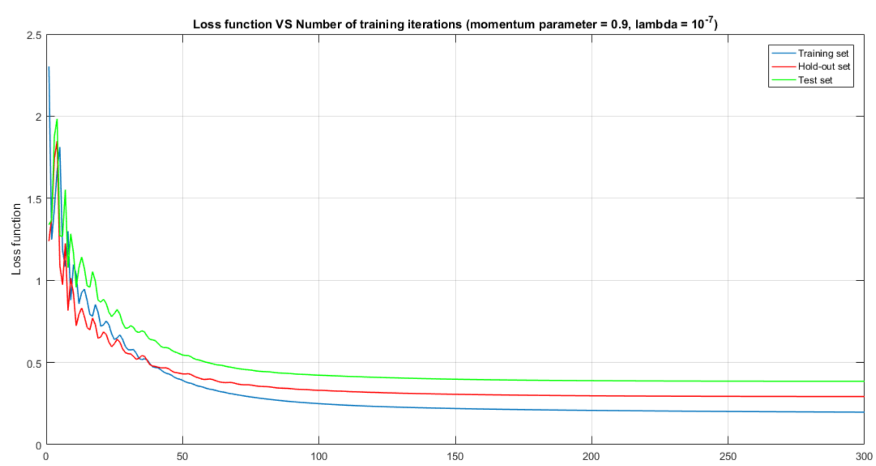

Fig. 5.13 Loss function vs number of training iteration ($\mu = 0.90$)

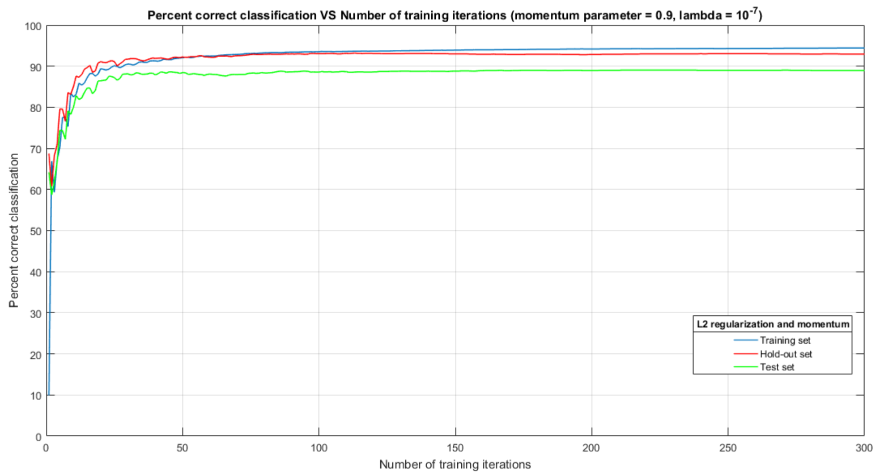

Fig. 5.14 Percent correct classification vs number of training iteration ($\mu = 0.90$)

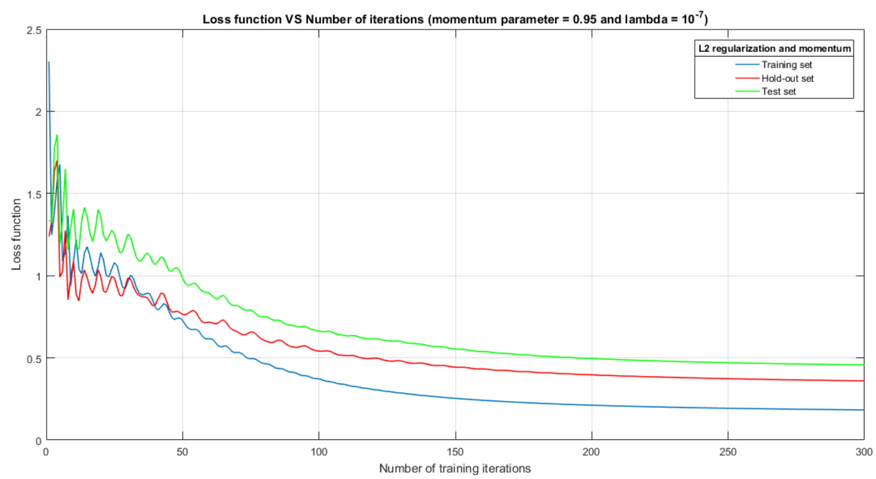

Fig. 5.15 Loss function vs number of training iteration ($\mu = 0.95$)

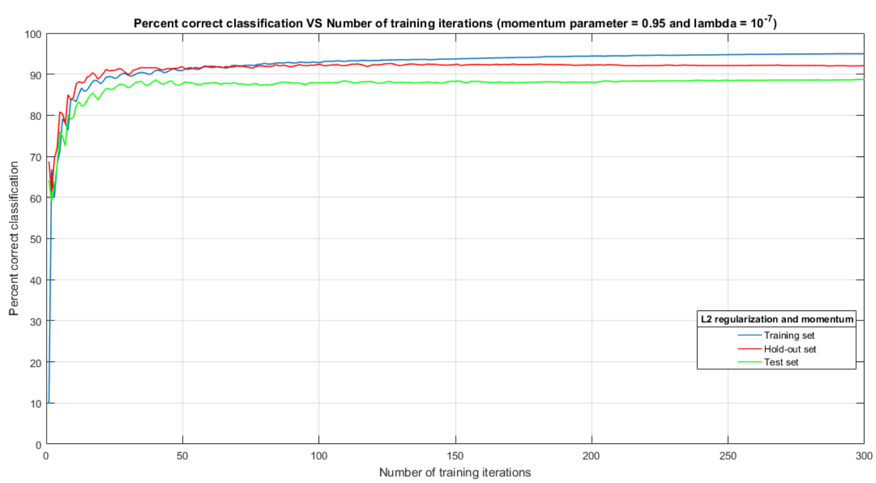

Fig. 5.16 Percent correct classification vs number of training iteration ($\mu = 0.95$)

5.4 Discussion

1) In the first experiment, we varied the value of learning rate η in order to determine that initial learning rate which yields the best results. The value of loss function for the test set was minimum (0.369809991871692) for $\eta = 0.0001$. Therefore, we chose the value of η to be 0.0001 for all of the remaining experiments.

2) In second experiment we used gradient descent with L1 regularization. Also, we varied the value of λ (strength of regularization), in order to determine that value of λ which yields the best results. The value of loss function for the test set was minimum (0.369866928243499) for $\lambda = 10^{-7}$.

3) In third experiment we used gradient descent with L2 regularization. Also, we varied the value of λ (strength of regularization), in order to determine that value of λ which yields the best results. The value of loss function for the test set was minimum (0.369821348544536) for $\lambda = 10^{-7}$.

4) In order to compare L2 vs. L1 regularization, we compared the value of loss function for the test set in both the cases. The loss function for the test set in case of L1 regularization was 0.369866928243499 and in case of L2 regularization was 0.369821348544536. Thus we observed that the performance in case of L2 regularization was slightly better that L1 regularization.

5) In the forth experiment we used gradient descent with both L2 regularization ($\lambda = 10^{-7}$) and momentum. Also, we varied the values of momentum parameter μ in order to determine that value of μ which yields the best results. The value of loss function for the test set was minimum (0.371197884443466) for $\mu = 0.85$.

Questions given in the assignment:

(a) Again, use a hold-out set, and use regularization, for the best value of λ and type of regularization ($L_2 vs: L_1$ from the previous experiment. Check a couple of other values of λ to see if you get better results, and use the best results in your report.

Answer) L2 regularization was slightly better than L1 regularization. The value of λ which gave the best result was equal to 10^{-7}. Please refer to the "Results" section.

(b) Plot the loss function over the number of training iterations for the training, hold-out, and test set. You don't need to make plots based on different values of λ, just report the best result you get, but document which λ you use.

Answer) Plots have been included in the "Results" section, please refer to it.

(c) Also plot the percent correct over the number of training iterations for the training, hold-out and test set for the same λ.

Answer) Plots have been included in the "Results" section, please refer to it.

6 Summary

We performed binary classification on the MNIST dataset using logistic regression achieving an accuracy of 2.12% on the test set, and 1.99% on the holdout set using the parameters $\mu = 0.9$(momentum parameter) and $\lambda = 10^{-6}$(strength of regularization). Also, we performed 10-way classification of the MNIST dataset using softmax regression achieving an accuracy of 89.2% on test set and 93.15% on hold-out set using the parameters $\mu = 0.85$ and $\lambda = 10^{-6}$. From this assignment, we learned the techniques of logistic and softmax regression for performing classification. We also learned the technique of regularization which is a way to improve generalization. Finally, we learned a simple technique of adding a momentum term to the gradient descent formula for dealing with the problem of widely differing eigen values.

7 Contributions

7.1 Saksham Sharma

- Derived the gradient for logistic regression.
- Derived the gradient for softmax regression.
- Completed the "Softmax Regression via Gradient Descent" part using simple gradient descent, L1 regularization, L2 regularization and momentum.

7.2 Kriti Aggarwal

- Completed the "Logistic Regression via Gradient Descent" part using simple gradient descent, L1 regularization, L2 regularization and momentum.

References

[1] Neural Networks for Pattern Recognition, Christopher M. Bishop.