INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

Tuning Hyper-parameters of Logistic Regression and Linear Classifier Model

Shivang Sharma
23M0752
dept. of Computer Science Engineering

Tanisha Chawada
23M1071
dept. of Electrical Engineering

1 INTRODUCTION

Classification is a supervised machine learning method used to predict the discrete label data based on the input data.Logistic regression for binary and multi class classification(linear classifier) are two of the most used machine learning algorithm for classification. These models are based on the concept of probability.Since the outcome is a probability, the dependent variable is bounded between 0 and 1. Here a function is defined on the input or features with the help of which we predict the output.This function can be greater than one but probability is between 0 and 1 so we need some activation function.We used sigmoid function , which maps the predicted value between 0 and 1.the sigmoid function is defined as:

$$g(z) = \frac{1}{1+e^{-z}}$$

where

$$z = w_0 + \sum_{i=1}^{n} w_i x_i$$

In vector form:

$$Z = W^T X$$

Cost function for linear regression is calculated as follows:

$$J(w) = -\frac{1}{m}\sum_{i=1}^{n}(y_i \log(p) + (1-y_i)\log(1-p))$$

where m is the number of entries in data set and p is sigmoid function.

If the value of loss function is high ,then the model reliability is less. So to improve the accuracy of the model we need to minimize the loss function. For this we use gradient descent algorithm. In this algorithm we take gradient of the loss function and subtract it from the weights and then update this value to the weights. We repeat this process to a fixed epochs. Expression for gradient descent:

$$\begin{aligned}\frac{dJ(w)}{dw} &= -\frac{d}{dw}\left(\frac{1}{m}\sum_{i=1}^{n}(y_i \log(p) + (1-y_i)\log(1-p))\right)\\ \frac{dJ(w)}{dw} &= \frac{dJ(w)}{dp}\frac{dp}{dz}\frac{dz}{dw} \qquad (1.1)\end{aligned}$$

Taking inner term and solving this three differently,

$$\begin{aligned}\frac{dJ(w_i)}{dp} &= \frac{-y_i}{p} + \frac{(1-y_i)}{(1-p)}\\ \frac{dp}{dz} &= \frac{e^{-z}}{(1+e^{-z})^2}\end{aligned}$$

We can write it as:

$$\begin{aligned}\frac{dp}{dz} &= (1-p)p \qquad (1.2)\\ \frac{dz}{dw_i} &= x_i \qquad (1.3)\end{aligned}$$

From (1.1),(1.2) and (1.3):

$$\begin{aligned}\frac{dJ(w_i)}{dp} &= \left[\frac{-y_i}{p} + \frac{(1-y_i)}{(1-p}\right](1-p)px_i\\ \frac{dJ(w_i)}{dp} &= \left[\frac{-y_i(1-p)+(1-y_i)p}{(1-p)p}\right](1-p)px_i\\ \frac{dJ(w_i)}{dp} &= \left[-y_i + y_i p) + p - y_i p\right]x_i\\ \frac{dJ(w_i)}{dp} &= (p-y_i)x_i \qquad (1.4)\end{aligned}$$

(1.4) can be written in vector form as:

$$\frac{dJ(W)}{dP} = (P-Y)X \qquad (1.5)$$

Now we can update our weights

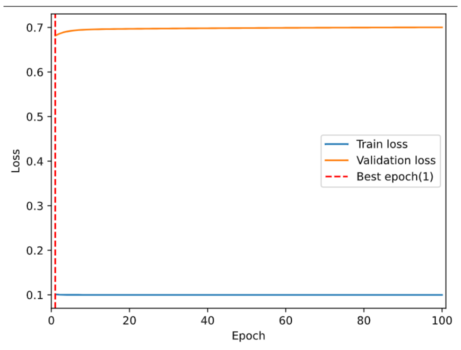
$$W = W - \alpha \frac{dJ(w)}{dP} \tag{1.6}$$

where α is learning rate for the model.

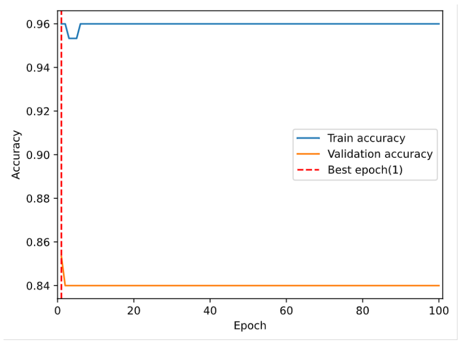
2 Tuning hyper-parameters for binary data set

2.1 Varying learning rate

1. α=0.1,epoch=100,momentum=0:



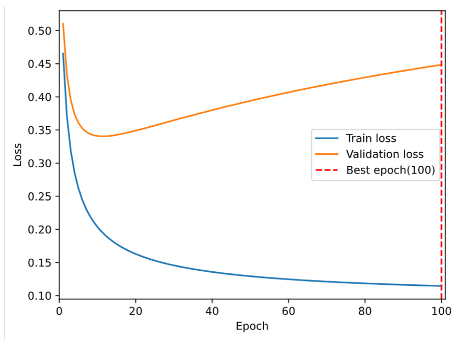
(a) Loss Plot



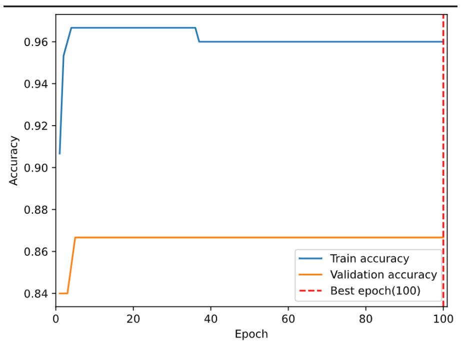
(b) Accuracy Plot

Figure 2.1: train loss=0.1,valid loss=0.7,valid accuracy=84%

2. α=0.01,epoch=100,momentum=0(train loss=0.11,valid loss=0.45,valid accuracy=86.67%):

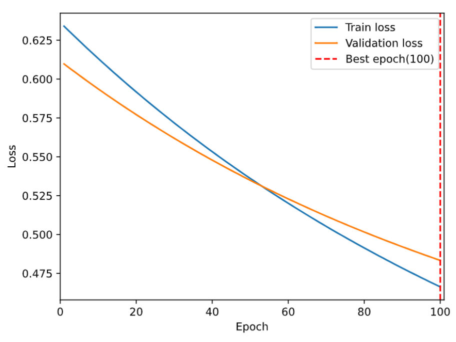


(a) Loss Plot

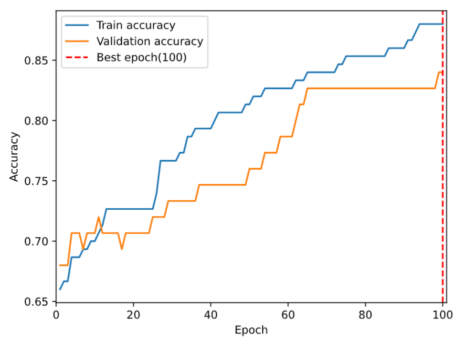


(b) Accuracy Plot

3. α=0.0001,epoch=100,momentum=0:



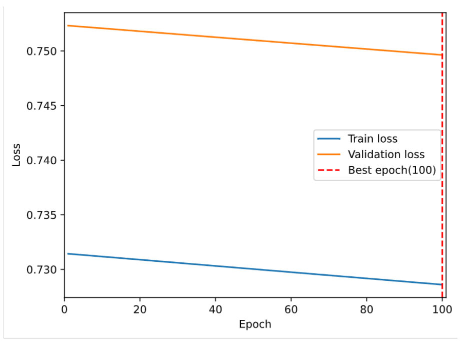
(a) Loss Plot



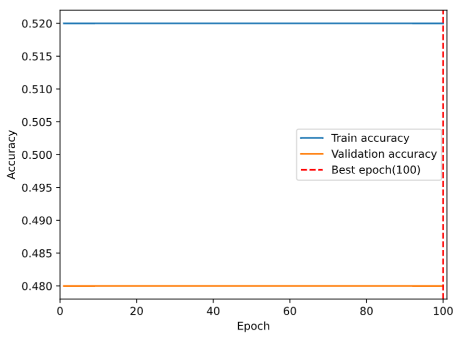
(b) Accuracy Plot

Figure 2.3: train loss=0.47,valid loss=0.48,valid accuracy=84%

4. α=1e-6,epoch=100,momentum=0:



(a) Loss Plot



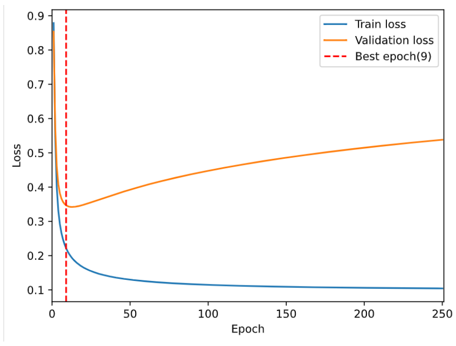
(b) Accuracy Plot

Figure 2.4: train loss=0.75,valid loss=0.48,valid accuracy=48%

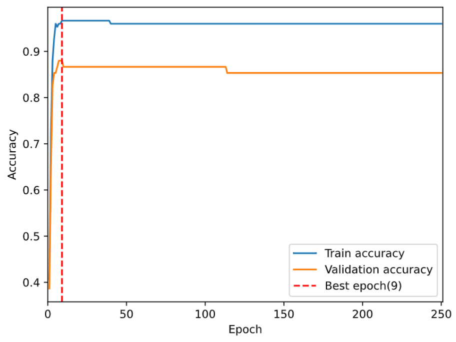
We have seen here that as the learning rate is decreasing ,valid loss also decreasing .We have best accuracy at $\alpha = 0.01 and \alpha = 0.0001$ which is 86.67%. With too small learning rate the process become slow and with too large learning rate the model will not give minimum loss.

2.2 Varying epoch

Varying epoch for $\alpha = 0.01$

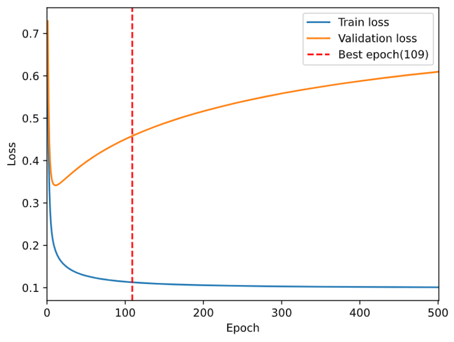


(a) Loss Plot

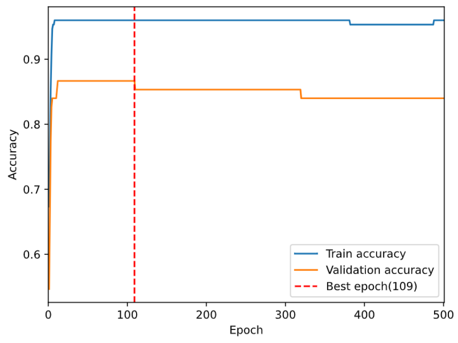


(b) Accuracy Plot

Figure 2.5: epoch==250,train loss=0.10,valid loss=0.54,valid accuracy=88%

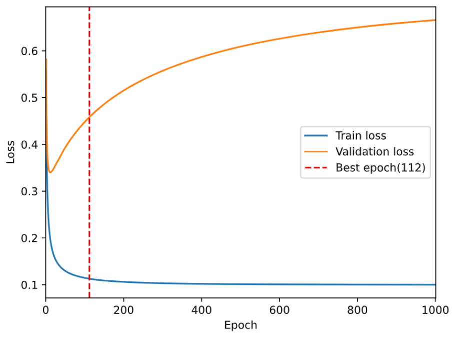


(a) Loss Plot

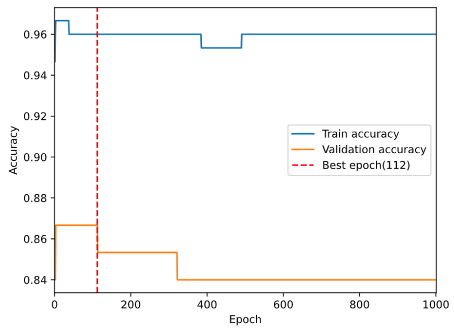


(b) Accuracy Plot

Figure 2.6: epoch=500,train loss=0.10,valid loss=0.61,valid accuracy=86.67%



(a) Loss Plot

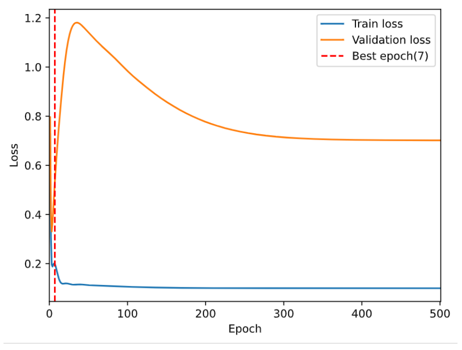


(b) Accuracy Plot

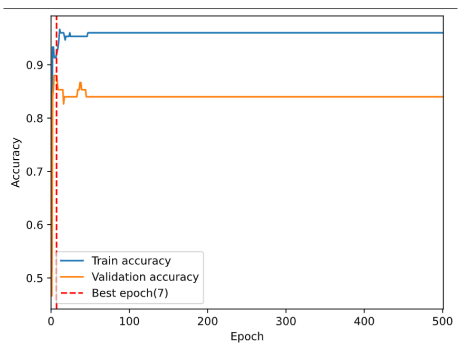
Figure 2.7: epoch=1000,train loss=0.10,valid loss=0.61,valid accuracy=86.67%

2.3 Varying momentum

Taking momentum=0.9,epoch=500 and learning rate=0.01



(a) Loss Plot



(b) Accuracy Plot

Figure 2.8: epoch=500,train loss=0.13,valid loss=0.31,valid accuracy=88%

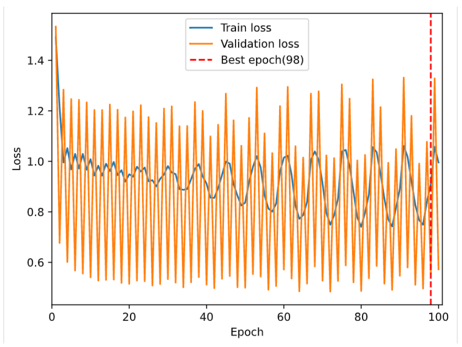
3 Tuning hyper-parameters for iris data set

3.1 Adapting Logistic regression to multi class setting

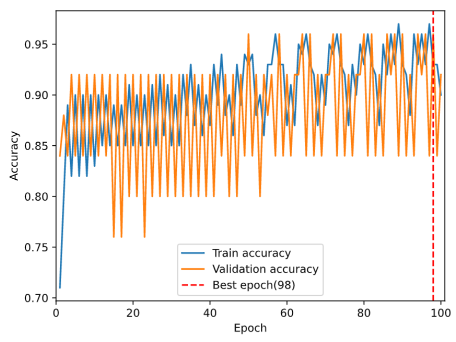
As we know logistic regression deal with classifying the data using linear decision boundary, which implies that the data ,it is classifying should be binary.Hence we have to tweak our multi class data in such a way that logistic regression could be able to classify it.
To achieve the above we have to create "C" linear classifier where "C" is number of classes. These "C" classifier will help us to classify data for each class. This approach is known as one vs all or one vs rest process. We have to process training output vector to one-hot encoding to calculate loss gradient.The weight matrix is of size d x 3 where d is the number of features and 1 added to it.Now for each class we will calculate loss. Similarly for gradient we will calculate gradients for each class decision boundary

3.2 Varying learning rate

1. α=0.1,epoch=100,momentum=0:



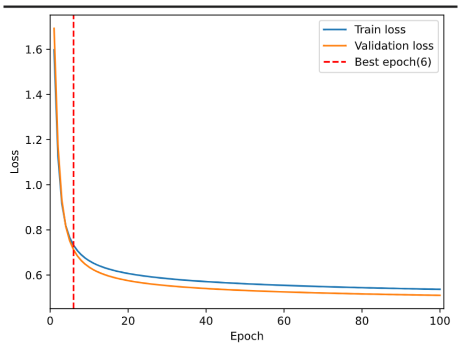
(a) Loss Plot



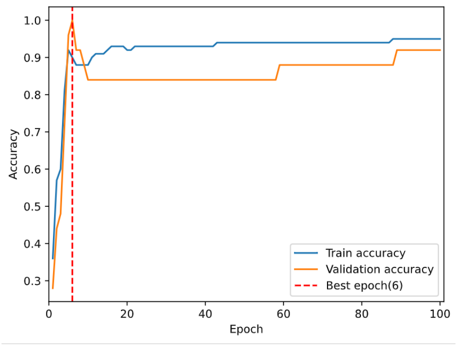
(b) Accuracy Plot

Figure 3.1: train loss=0.54,valid loss=0.92,valid accuracy=92%

2. α=0.01,epoch=100,momentum=0:



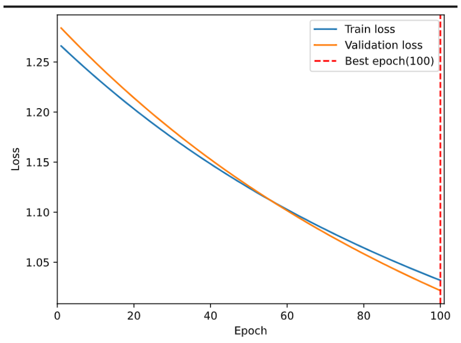
(a) Loss Plot



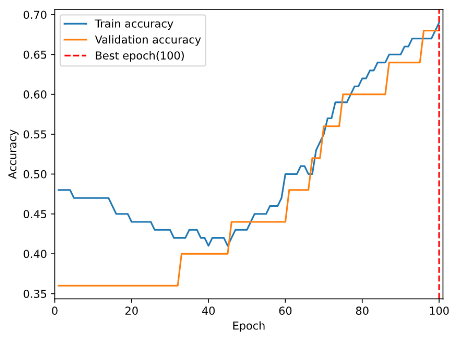
(b) Accuracy Plot

Figure 3.2: train loss=0.58,valid loss=0.57,valid accuracy=96%

3. α=0.0001,epoch=100,momentum=0:



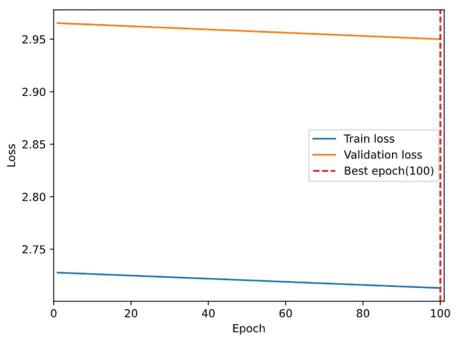
(a) Loss Plot



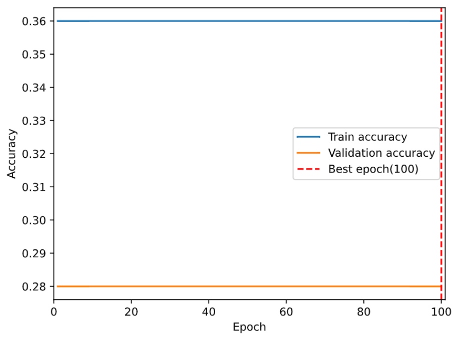
(b) Accuracy Plot

Figure 3.3: train loss=1.02,valid loss=1.03,valid accuracy=68%

4. α=1e-6,epoch=100,momentum=0:



(a) Loss Plot



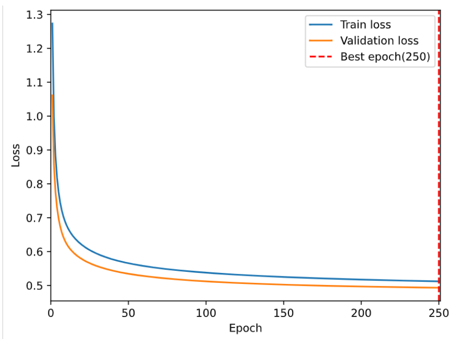
(b) Accuracy Plot

Figure 3.4: train loss=2.71,valid loss=2.95,valid accuracy=28%

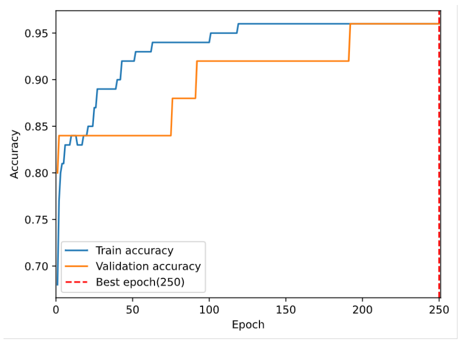
Here the best validate accuracy is for $\alpha = 0.01$ and it decreases with decrease in α.The number of epoch that is sufficient to get best accuracy is 250 .
As we increase the momentum,the number of epoch for the same accuracy is less as momentum accelerates the gradient and helps in achieving the minimum cost function rapidly.

3.3 Varying epoch

1. epoch=250,$\alpha = 0.01$



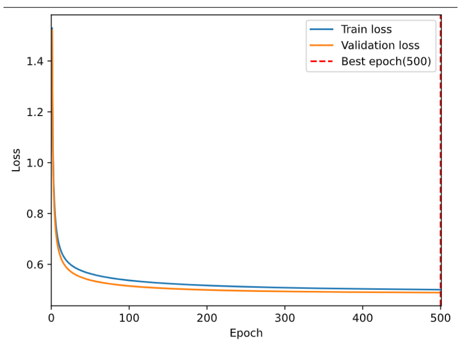
(a) Loss Plot



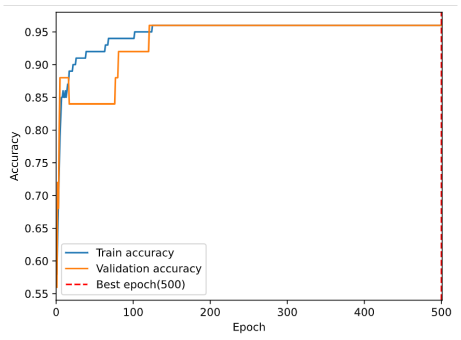
(b) Accuracy Plot

Figure 3.5: epoch=250,train loss=0.51,valid loss=0.49,valid accuracy=96%

2. epoch=500,$\alpha = 0.01$



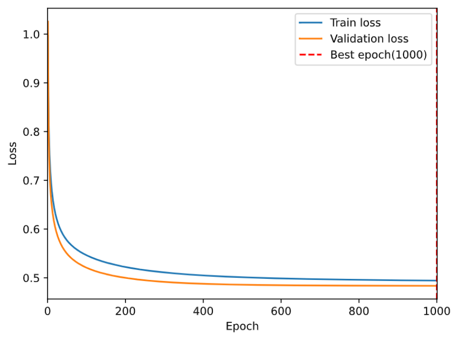
(a) Loss Plot



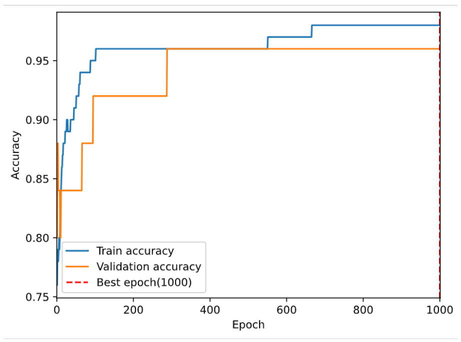
(b) Accuracy Plot

Figure 3.6: epoch=500,train loss=0.51,valid loss=0.49,valid accuracy=96%

3. epoch=1000,$\alpha = 0.01$



(a) Loss Plot

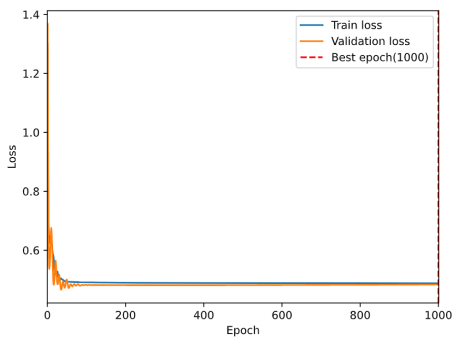


(b) Accuracy Plot

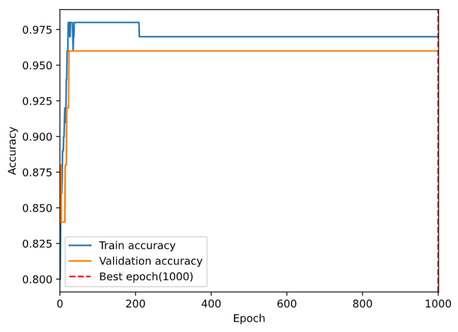
Figure 3.7: epoch=1000,train loss=0.5,valid loss=0.49,valid accuracy=96%

3.4 Varying momentum

Taking momentum=0.9,epoch=1000 and learning rate=0.01



(a) Loss Plot



(b) Accuracy Plot

Figure 3.8: epoch=1000,train loss=0.49,valid loss=0.48,valid accuracy=96%