FOUNDATIONS OF MACHINE LEARNING

HOMEWORK 1 REPORT

Prepared by:

Name	Roll No.	Program
Arijeet	23M0742	MTech CSE
A Asish	23M0759	MTech CSE

Section 1: Logistic Regression

Learning rate:

1e-1

momentum\epochs	100	250	500	1000
0	0.56	0.56	0.56	0.56
0.9	0.56			

1e-2

momentum\epochs	100
0	0.65

1e-4

momentum\epochs	100
0	0.8

1e-6

momentum\epochs	100
0	0.8

Best parameters:

Learning Rate = 1e-4, Epochs= 60, Momentum = 0.9.

Train Loss = 0.67, Valid Loss = 0.68, Valid Accuracy = 0.8.

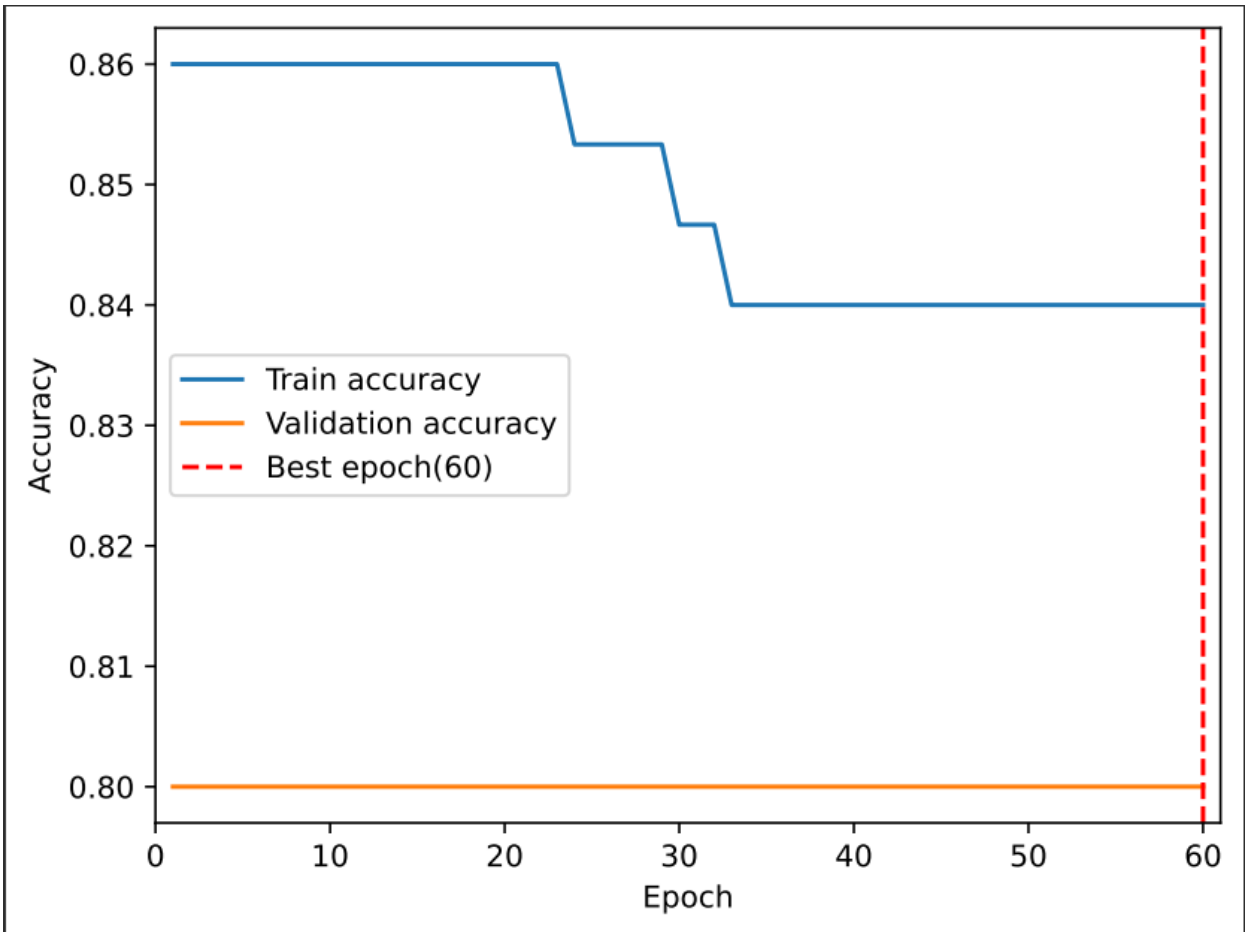
Various values of hyperparameters affected the accuracy and loss values in different ways.

- Learning rate: Having a high learning rate such as 0.1 or 0.01 gave us abysmal 56% and 65% accuracy respectively for 100 epochs. One thing we noticed with high learning rate is that model was oscillating between high and low accuracy rates. This can be attributed to the fact that the gradient descent algorithm must have taken large steps while trying to move towards minima of the loss function, but due to large step values, it crossed the minima several times and couldn't converge. Meanwhile, we also tried the learning rate of 1e-4 which was neither too small nor too large, and it gave us the best accuracy of 80% for 100 epochs.

- Number of epochs: We observed that our model converged within a smaller number of epochs, like within 60 epochs, and a greater number of epochs didn't improve anything, in fact in a case with 0.9 momentum, it decreased the accuracy of our model.

- Momentum: It has been observed that momentum is useful only when we used it with less epochs, it didn't help when epochs were more, and decreased the accuracy further.

So, we observed that the hyperparameters that gave us the best results are learning rate= 1e-4, number of epochs= 60 and momentum of 0.9. The validation accuracy we were able to obtain was 80% with a validation loss of 0.68.



Accuracy Plot

Section 2: Linear Classifier

Learning rate:

1e-1

momentum\epochs	100
0	0.32

1e-2

momentum\epochs	100	250	500	1000
0	0.96	0.96	1.0	1.0
0.9				0.96

1e-4

momentum\epochs	100
0	0.48

1e-6

momentum\epochs	100
0	0.32

Best parameters:

Learning Rate = 1e-2, Epochs = 850, Momentum = 0.9.

Train Loss = 2.10, Valid Loss = 0.67, Valid Accuracy = 1.0.

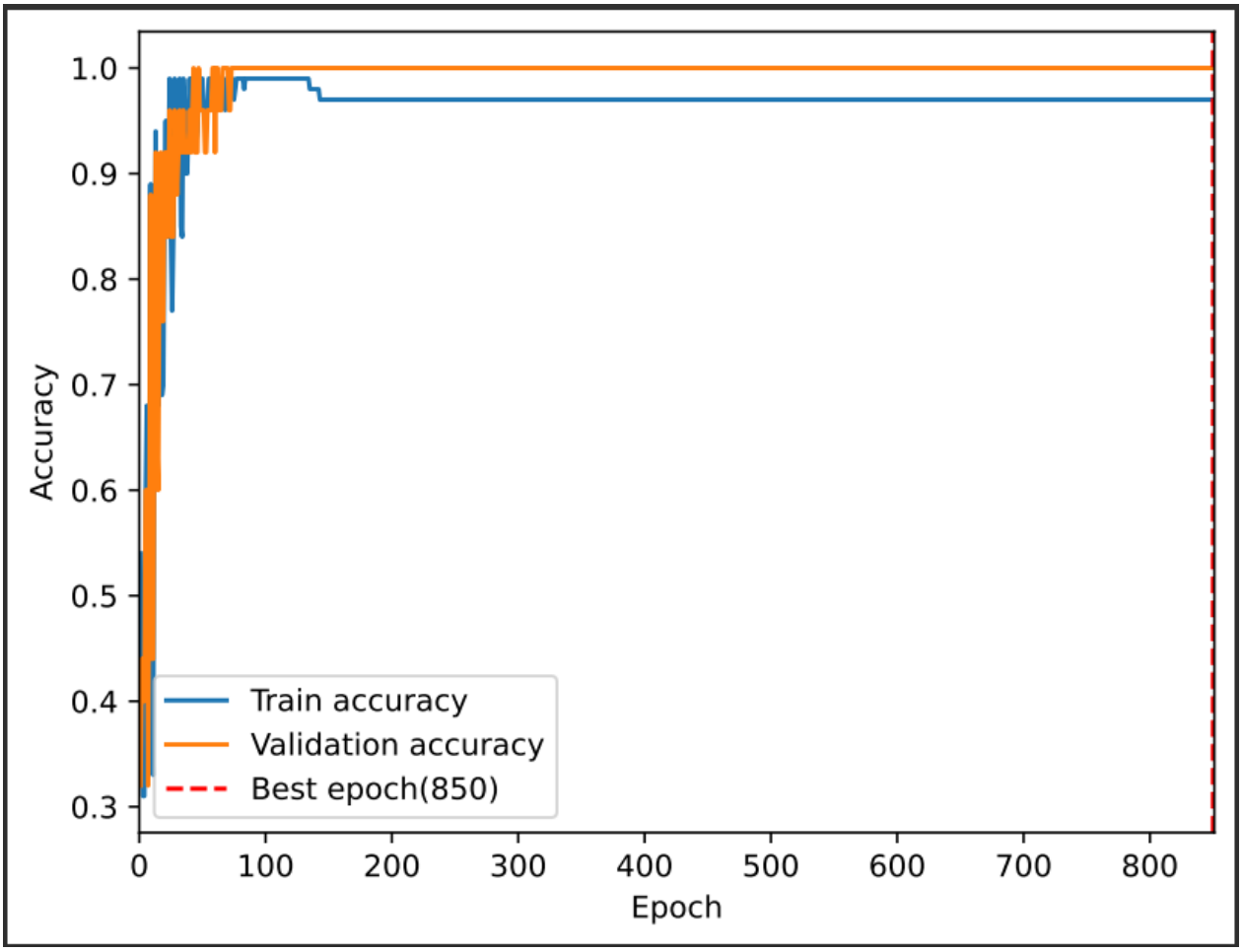
Various values of hyperparameters affected the accuracy and loss values in different ways.

- Learning rate: We observed that having too less learning rates like 1e-4 or 1e-6 gave us very poor accuracy and high loss values. Having a learning rate of 1e-4 gave us an accuracy of only 48% and learning rate 1e-6 gave even worse accuracy of just 32% for 100 epochs. This may be since the gradient descent algorithm made very small incremental changes for moving towards the parameter values which gave lesser loss value, and as the step values were very small, algorithm couldn't move enough to converge. On the other hand, having a high learning rate such as 0.1 also didn't help us either, as it gave an abysmal 32% accuracy for 100 epochs. One thing we noticed with high learning rate is that model was oscillating between high and low accuracy rates. This can be attributed to the fact that the gradient descent algorithm must have taken large steps while trying to move towards minima of the loss function, but due to large step values, it crossed the minima several times and couldn't converge. Meanwhile, we also tried the learning rate of 0.01 which was neither too small nor too large, and it gave us the best accuracy of 100% for 500 epochs.

- Number of epochs: With the increasing number of epochs, we are giving the model more steps and time to let it converge to the minimum loss value. Increasing the number of epochs helps greatly when we have a small learning rate, but too small a learning rate can require a very high number of epochs. With correct learning rate and enough epochs, our model should converge well, as we observed with ours. With learning rate 0.01, 100 epochs gave us 96% accuracy, whereas it increased to 100% when we increased epochs to 500. Increasing epochs after the model has already reached optimal accuracy doesn't help either, as we observed that 500 epochs were sufficient for learning rate of 0.01 and increasing epochs to 1000 gave no better results.

- Momentum: Momentum is useful to help the model escape local minima and converge at global minima if any. We observed that with momentum of 0.9, we were able to obtain the most optimal results that are: accuracy of 100% and validation loss of 0.67 in 850 epochs with a learning rate of 0.01.

So, we observed that the hyperparameters that gave us the best results are learning rate= 0.01, number of epochs= 850 and momentum of 0.9. The validation accuracy we were able to obtain was 100% with a validation loss of 0.67.



Accuracy Plot

Mathematical derivation of the gradient term for logistic regression

A good loss function is that which has the following properties:

- Computationally feasible
- Easy to optimize

Hence, we are using the loss function that satisfies such properties. It is defined as:

loss = - (1/n) * Σ [y * log(g(z)) + (1 - y) * log (1 - g(z))]

loss is a function of parameters w(coefficients) of the model

Here,

n: the number of data examples

y: true binary label (0 or 1)

z: linear combination of features and parameters: z= w^T*x

sigmoid function: g(z) = 1 / (1 + exp(-z))

Then we derive the gradient of sigmoid function g(z) with respect to x: d(g(z))/dz = σ(z) * (1 - g(z))

The derivative of the loss function with respect to w for a single data point (x, y):

d(loss(θ))/dθ = - [y * (1/g(z)) * d(g(z))/dz * x + (1 - y) * (-1/(1 - g(z))) * d(g(z))/dz * x]

= - [y * (1 - g(z)) * x + (1 - y) * g(z) * (-x)]

= - [y * x - y * g(z) * x - g(z) * x + y * g(z) * x]

= - [y - g(z)] * x

Gradient for dataset is average of individual gradients of data points:

∇L(w) = - (1/n) * Σ [(y - g(z)) * x]

Study on iris

In iris we have 3 classes. (Say: c1, c2, c3)

Logistic Regression only helps in classifying 2 classes.

So, instead of creating only one logistic regression model we are going to create 3 logistic regression models.

Logistic Model-1: Classifies class c1 and [c2, c3]

- [c2, c3] together will be considered a separate class

Logistic Model-2: Classifies class c2 and [c3, c1]

- [c3, c1] together will be considered a separate class

Logistic Model-3: Classifies class c3 and [c1, c2]

- [c1, c2] together will be considered a separate class

Now we are going to use all three models for prediction and choose the class with highest predicted probability.

Drawbacks:

- We need to create multiple models whose training and testing is time consuming.
- For 'n' number of classes we need nC2 models to train and test which is not at all feasible for large values of n.