Binary Classification

Classifying the Red-wine Dataset

Kartik Srinivas - ES20BTECH11015

A Club Project

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

June 25, 2022

1 Logistic Regression

Since this is a classification problem, the natural approach would be to apply a regression model , squashed with a Sigmoid , to give the predicted probability of an example being positive or negative. We then find the softmax loss (Binary - case).

1.1 Forward Pass

$$X \in \mathbb{R}^{N \times D} = \text{Input Examples}$$
$$D = \text{Number of Dimensions of Input Vector}$$
$$N = \text{Number of Input Examples}$$
$$w \in \mathbb{R}^{D \times 2}$$
$$y \in \mathbb{R}^{N} = \text{Correct Scores}$$

$$z = w^T X + b \tag{1}$$
$$a = \sigma(z) \tag{2}$$
$$L = -1 * (log(a)y + (1 - a)log(1 - y)) \tag{3}$$

```python
def sigmoid_forward(x, w, b):
    a = sigmoid(np.dot(x, w) + b)
    cache = (x, w, b, a)
    return a, cache

def cross_entropy_loss(a, y):
    y_new = y.reshape(a.shape)
    loss_vector = (-1 *( y_new * np.log(a) + (1 - y_new) * np.log(1 - a) ) )
    loss = np.sum(loss_vector, axis = 0)
    cache = a, loss_vector, y_new
    return loss, cache
```

1.2 Backprop

After the loss we do a standard Backprop, however we can create a layer that does the backward passes of both the BCE and sigmoid layers together, the math is as follows : -

$$\frac{\partial L}{\partial a} = -1 * \left(\frac{y}{a} - \frac{1-y}{1-a}\right) \tag{4}$$
$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial a} * \frac{\partial a}{\partial z} \tag{5}$$
$$\frac{\partial a}{\partial z} = a(1-a) \ldots \text{sigmoid derivative} \tag{6}$$
$$\therefore \frac{\partial L}{\partial z} = a - y \tag{7}$$

$$\boxed{\therefore \frac{\partial L}{\partial w} = X^T \times \frac{\partial L}{\partial z}} \tag{8}$$

```python
def cross_entropy_and_sigmoid_backward(cache, y): # utility function for ease
    x, w, b, a = cache
    dz = a - y
    dw = np.dot(x.T, dz)
    db = np.sum(dz, axis=0)
    return dw, db
```

2 Solver

I have written code for Batch Gradient Descent and have also added the option of training with Momentum:-

Normal Gradient Descent

$$p = p - \alpha \frac{\partial L}{\partial p}$$

Gradient Descent with momentum See :- μ is Just a Friction factor to stop the gradient descent eventually this is a hyper-parameter set to 0.9

$$V_p = \mu V_p - \alpha \frac{\partial L}{\partial p} \tag{9}$$

$$p = p + V_p \tag{10}$$

2.1 Results With Momentum

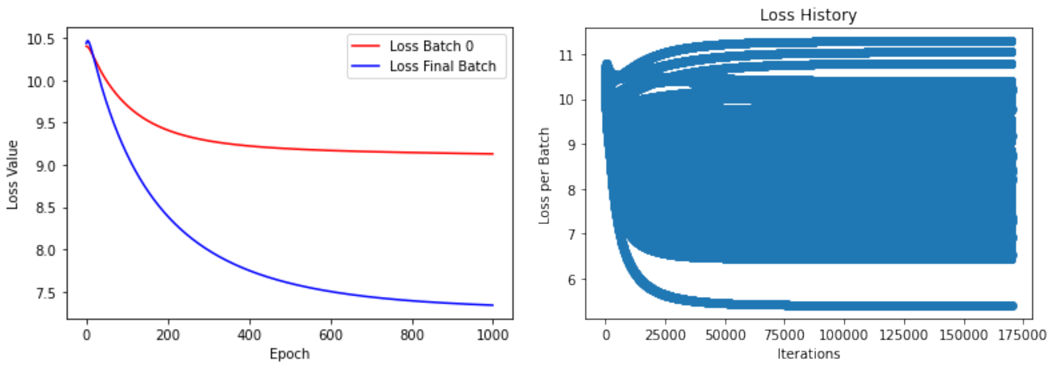

Figure 1: Losses during training $\alpha = 1.7e-4, \mu = 0.9$, batch size = 15

2.2 Results without Momentum for same Learning Rate

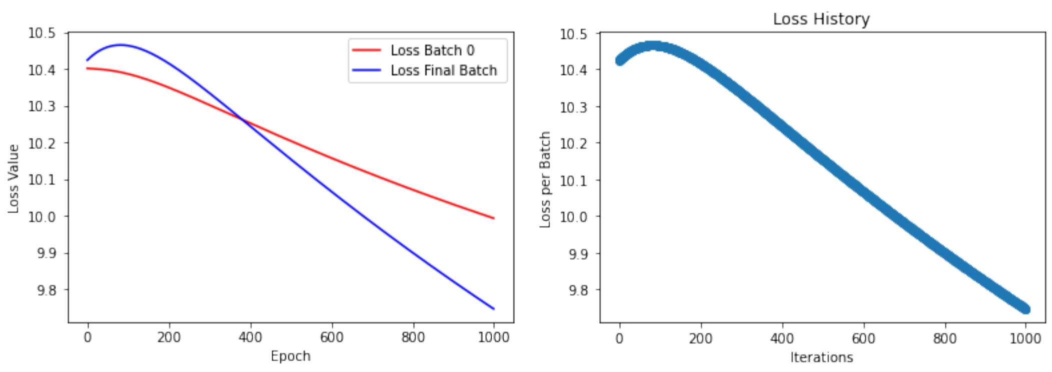

Figure 2: Losses during training

2.3 Accuracy

Train Accuracy with Momentum :- 74.43%
Test Accuracy with Momentum :- 73.53%

3 Neural Network

I have implemented a simple two Layer Net from Scratch.

$$\text{Model is :-} \boxed{Affine - RELU - Affine - Softmax}$$

I learnt the necessary Material from the course I was doing :- CS231N

3.1 Back-prop Through a Softmax

$$L = \frac{1}{N}\sum_{i=1}^{i=N} -\log\left(\frac{e^{S_{ij}}}{\sum_j e^{S_{ij}}}\right) \quad \ldots\ S_j \text{ is correct class Score} \tag{11}$$

$$\tag{12}$$

$$\frac{\partial L}{\partial S_{kl}} = \frac{-1}{N}\sum_{i=1}^{i=N}\left(\frac{\sum_j e^{S_{ij}} - e^{S_{ij}}}{\sum_j e^{S_{ij}}}\right) \ldots \text{if } k,l = i,j \tag{13}$$

$$\boxed{\frac{\partial L}{\partial S_{kl}} = \frac{-1}{N}\sum_{i=1}^{i=N}\left(1 - \frac{e^{S_{ij}}}{\sum_j e^{S_{ij}}}\right) \ldots \text{if } k,l = i,j\, \frac{\partial L}{\partial S_{kl}} = 0 \ldots \text{if } k,l \neq i,j} \tag{14}$$

```
#code inside Models/Layers.py
def softmax_loss(x, y): # this is a numerically stable version of cross entropy loss
    probs = np.exp(x - np.max(x, axis=1, keepdims=True))
    probs /= np.sum(probs, axis=1, keepdims=True)
    N = x.shape[0]
    loss = -np.sum(np.log(probs[np.arange(N), y])) / N
    dx = probs.copy()
    dx[np.arange(N), y] -= 1
    dx /= N
    return loss, dx
```

Rest of the backprps are obvious.

3.2 Training

The training process is still going on, SGD + momentum has been used to train the model.

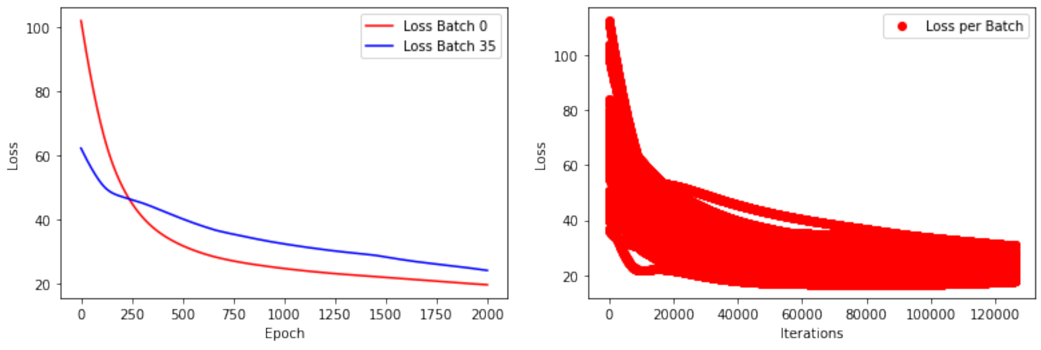

Figure 3: Losses with Momentum

3.3 Accuracy

This model is still a work in progress.
Training Accuracy = 51%
Testing Accuracy = 49.375%