DEEP LEARNING
HW_2
REPORT

INSTRUCTOR: Dr. Sumohana Channappayya
AAYUSH ARORA
EE17BTECH11003

Dataset Information:

All input attributes are integers in the range 0..100.
The last attribute is the class code 0..9

Data Set Characteristics:	Multivariate	**Number of Instances:**	10992	**Area:**	Computer
Attribute Characteristics:	Integer	**Number of Attributes:**	16	**Date Donated**	1998-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	185224

Architecture Used:

Input[16*1]→ Hidden1[50*1] →Batch Normalisation → relu → Hidden2[25*1] → Batch Normalisation → relu → Output Layer → Softmax

OPTIMISATION METHODS:

1. Stochastic Gradient Descent

Observations:

Stochastic Gradient Descent is easy to implement and very efficient. The hyperparmeters required are number of iterations , regularization parameter and learning rate.

Stochastic Gradient Descent is very sensitive to feature scaling.

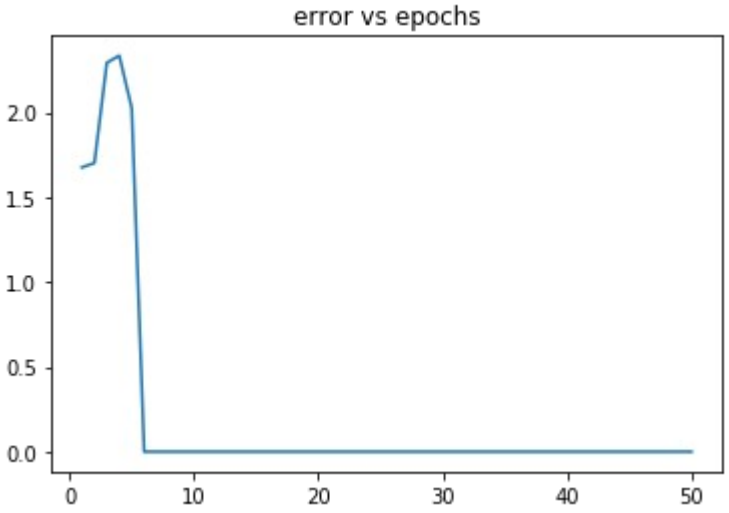
Choosing a proper learning rate can be difficult. A learning rate that is too small leads to slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.

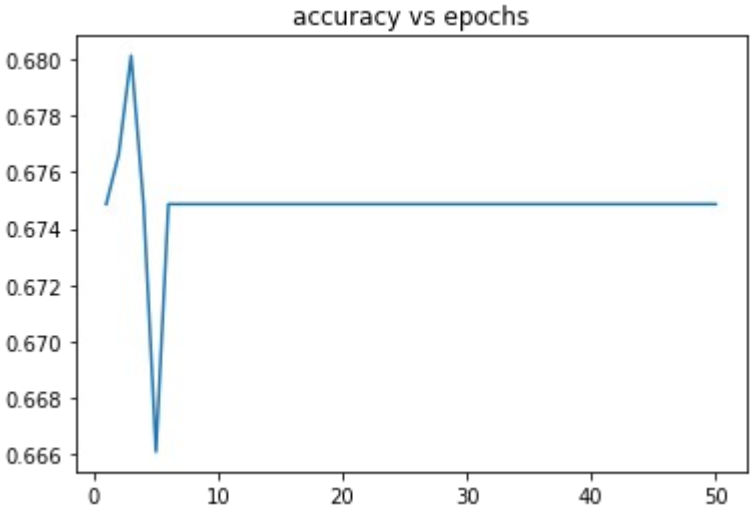
As we need to calculate the gradients for the whole batch to perform just *one* update, batch gradient descent can be very slow

Working:

$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$

GRAPHS:





<u>Results:</u>
learning rate=0.01
accuracy reaches 0.675 in 50 epochs
error tends to zero after 50 epochs

<u>2. Momentum</u>

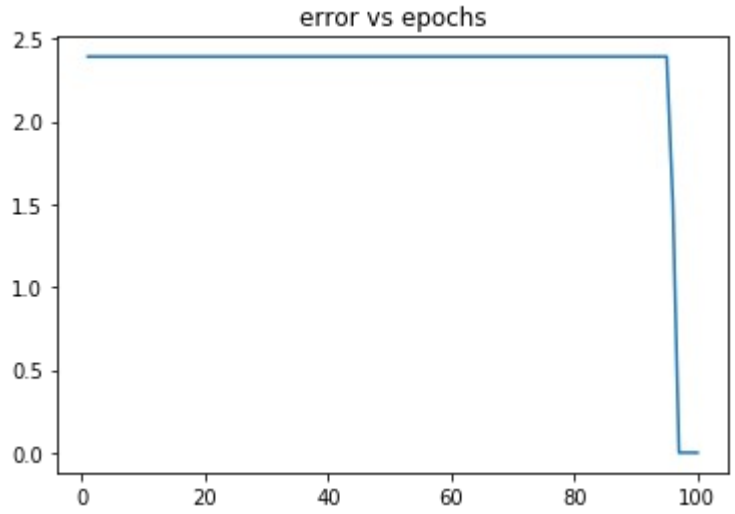
Observations:

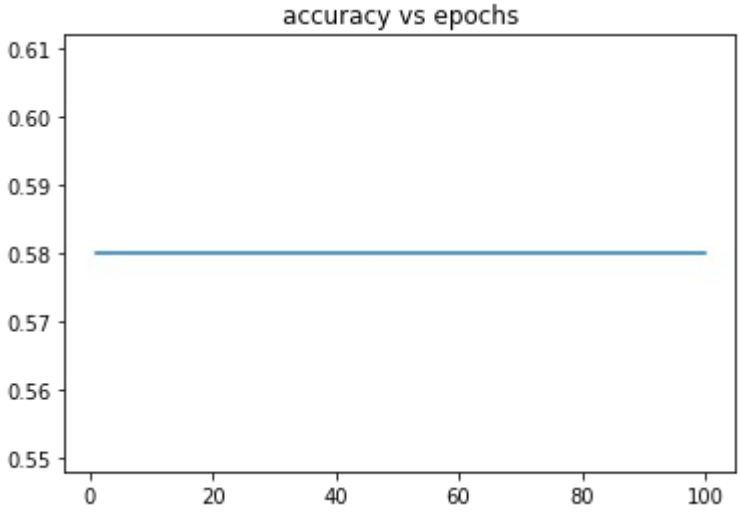
Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations.
The advantage of momentum is that it makes very small change to SGD but provides a big boost to speed of learning.
We need to store the velocity for all the parameters, and use this velocity for making the updates.

Working:

$v_t = \gamma * v_{t-1} + \eta * \nabla_\theta J(\theta)$
$\theta = \theta - v_t$

GRAPHS:





Results:
learning rate=0.001
gamma=0.9
accuracy reaches 0.58in 100 epochs
error tends to 2.5 after 80 epochs and then turns towards zero

3. Nesterov Momentum

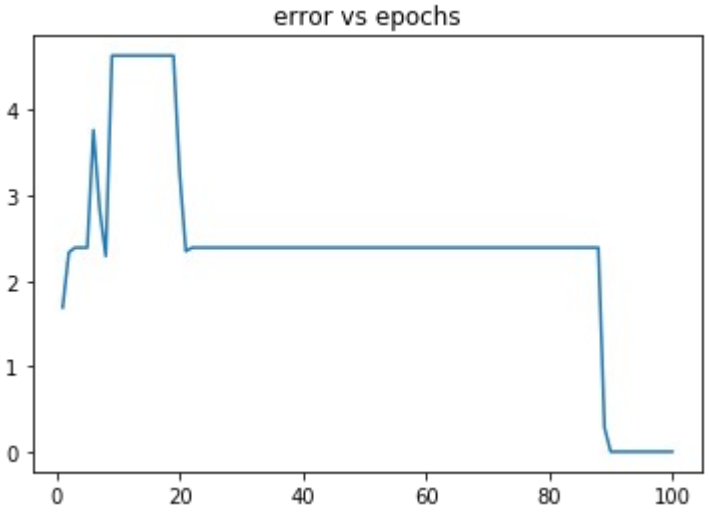
Observations:

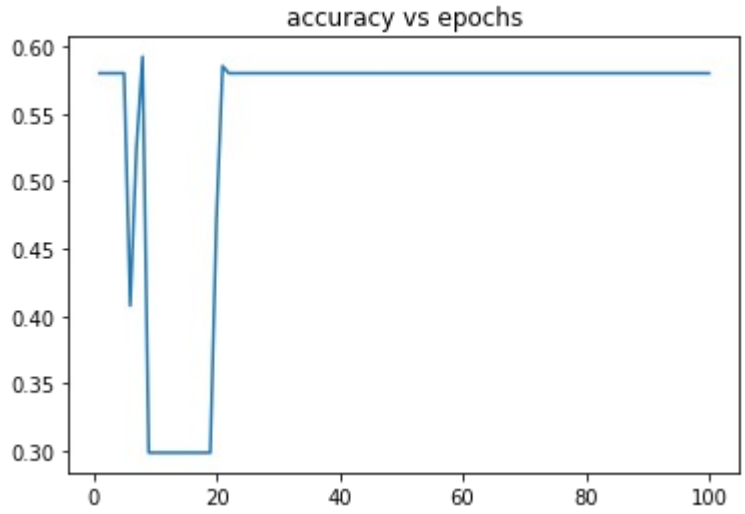
Nesterov accelerated gradient is a way to give our momentum term this kind of prescience. We know that we will use our momentum term $\gamma * v_{t-1}$ to move the parameters θ. Computing $\theta - \gamma * v_{t-1}$ thus gives us an approximation of the next position of the parameters.

Working:

$v_t = \gamma * v_{t-1} + \eta * \nabla_\theta J(\theta - \gamma * v_{t-1})$
$\theta = \theta - v_t$

GRAPHS:





<u>Results:</u>

learning rate=0.001

gamma=0.9

accuracy reaches 0.58 in 50 epochs

4. AdaGrad

Observations:

It adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data.
Adagrad's main benefits is that it eliminates the need to manually tune the learning rate.
Adagrad's main weakness is its accumulation of the squared gradients in the denominator: Since every added term is positive, the accumulated sum keeps growing during training. This in turn causes the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge.

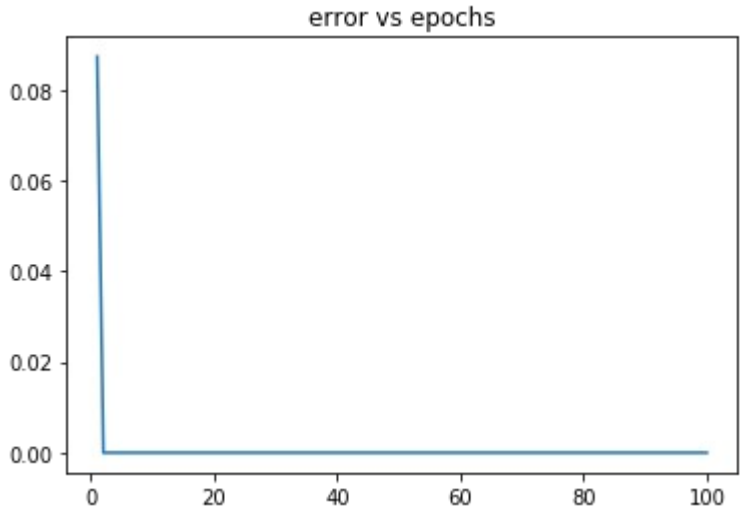
Working:

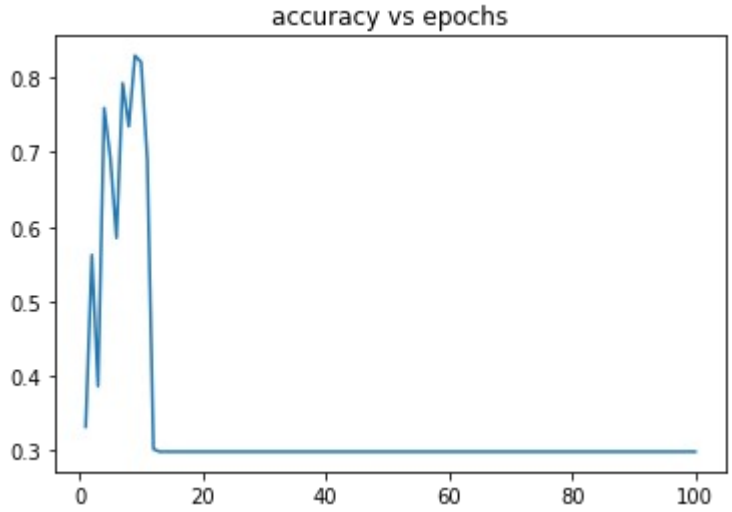
$g_{t,i} = \nabla_\theta J(\theta_{t,i})$

$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\epsilon}} * g_{t,i}$

$\theta_{t+1} = \theta_t - \eta/\sqrt{G_t+\epsilon} \odot g_t$

GRAPHS:





Results:
learning rate=0.001
epsilon=1e-8
accuracy reaches 0.8 in 15 epochs and then decreases to 0.3 till 100 epochs

5. RMSprop

Observations:

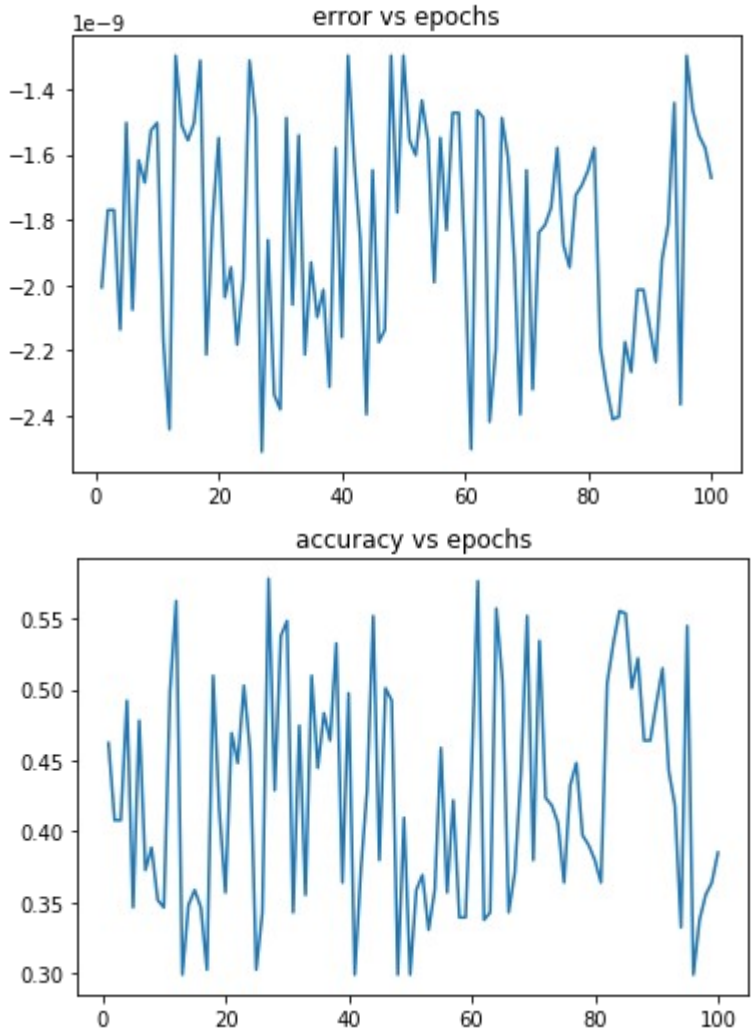
RMSprop is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, RMSprop restricts the window of accumulated past gradients to some fixed size.

Working:

$E[g^2]_t = 0.9*E[g^2]_{t-1} + 0.1*g^2_t$

$\theta_{t+1} = \theta_t - \eta/\sqrt{E[g^2]_t+\epsilon}*g_t$

GRAPHS:



Results:
learning rate=0.01
epsilon=1e-8
beta=0.9
accuracy and error both fluctuate a lot.
Decreasing learning rate and tuning other parameters might help smoothen the curve.

6. Adam

Observations:

Adaptive Moment Estimation is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum.
m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively.

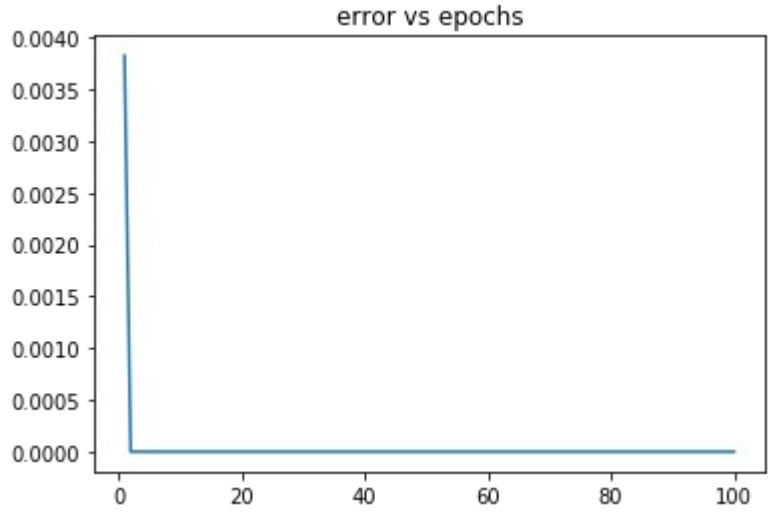
Working:

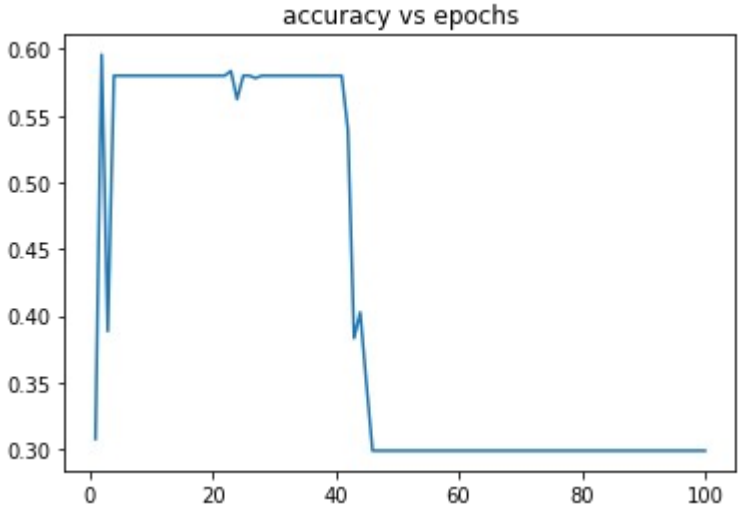
$$m_t = \beta_1 * m_{t-1} + (1-\beta_1) * g_t$$
$$v_t = \beta_2 * v_{t-1} + (1-\beta_2) * g^2_t$$

$m_t\_h = m_t/(1-\beta^t_1)$
$v_t\_h = v_t/(1-\beta^t_2)$

$\theta_{t+1}=\theta_t-\eta/(\sqrt{v_t\_h}+\epsilon)*m_t\_h$

GRAPHS:





Results:

learning rate=0.01
epsilon=1e-8
beta1=0.9
beta2=0.999
accuracy increases till 40^{th} epoch and then decreases steeply.