Home Work 4:

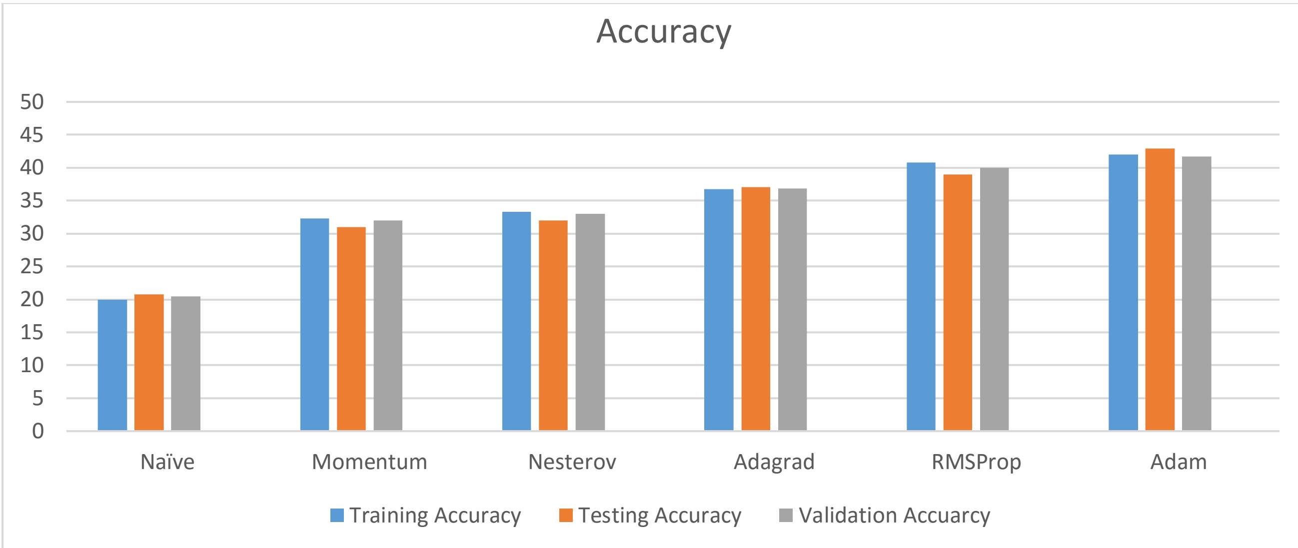
Results

Tabulations:

Update	Naïve	Momentum	Nesterov	AdaGrad	RMSProp	Adam
Loop 0	2.302560867	2.302595339	2.302569961	2.302575305	2.30257827	2.302585205
Loop 100	2.295970858	1.89156783	1.887977726	1.721447587	1.810140755	1.532103872
Loop 200	2.296476312	1.89756581	1.844183583	1.902541848	1.717931217	1.575818739
Loop 300	2.295147342	1.999408491	1.781892118	1.818870056	1.816634685	1.763459347
Loop 400	2.295395484	1.965626443	1.69286362	1.802603348	1.666950595	1.583542374
Loop 500	2.296978998	1.876028544	1.849066847	1.674843616	1.712391654	1.611060405
Loop 600	2.294973187	2.003691006	1.767654288	1.728315423	1.771991308	1.663846837
Loop 700	2.298596359	1.888468521	1.745947052	1.822965662	1.690821058	1.82220571
Loop 800	2.295379388	1.902303381	1.759233819	1.856600457	1.783659299	1.647672486
Loop 900	2.29669784	1.993932728	1.800251133	1.77145822	1.739260831	1.661940727
Loop 1000	2.29497296	1.952166182	1.784524481	1.822247125	1.881608867	1.563723839
Loop 1100	2.295453069	1.905633585	1.856827471	1.781314697	1.814911231	1.617956083
Loop 1200	2.297156317	1.886660916	1.726706561	1.844019364	1.920466441	1.540073917
Loop 1300	2.296055464	1.84796136	1.871714488	1.808413855	1.886326974	1.750189912
Loop 1400	2.295337121	1.93841692	1.741346838	1.781011583	1.762920156	1.645871939

Update	Training Accuracy %	Testing Accuracy %	Validation Accuracy %	Time (s)
Naïve	19.992	20.8	20.47	24.55
Momentum	32.27	31	32.04	28.82
Nesterov	33.29	32	32.97	33.29
AdaGrad	36.78	37.1	36.9	32.7
RMSProp	40.83	39	40	35.1
Adam	42	42.9	41.72	39.59

Graphs:


Accuracy
50
45
40
35
30
25
20
15
10
5
0
Naïve
Momentum
Nesterov
Adagrad
RMSProp
Adam
Training Accuracy
Testing Accuracy
Validation Accuarcy


Loss
2.2
2
1.8
1.6
1.4
1.2
1
Loop 0
Loop 100
Loop 200
Loop 300
Loop 400
Loop 500
Loop 600
Loop 700
Loop 800
Loop 900
Loop 1000
Loop 1100
Loop 1200
Loop 1300
Loop 1400
Naïve
Momentum
Nesterov
Adagrad
RMSProp
Adam

Notes and Observations :

Naïve Update:

The Naïve Update is simplistic in nature and tries to minimize the error Loss by moving in the negative direction of the gradient.

W += -lr* dW

- The naïve update applied to the leaky ReLu model starts with a loss of 2.3025 and proceeds to reduce very slowly to 2.29.
- Observing that the learning rate used in the previous homework (5e^-3) we arrive at a better accuracy in this limited number of 1500 iterations.
- It is clear that the naïve update is not arriving at a respectable loss, with the present learning rate and other tunable parameters, in the limited number of iterations.

Momentum Update:

The Momentum Update uses the gradient to calculate the 'velocity' of the randomly initialized parameters and uses this computed velocity to in turn update the weights.

V = mu * V - lr * dW
W += V

- The momentum update performs significantly better in the accuracy achieved for the testing, training and validation of the same data se than the naïve update for the given.
- We notice that the loss over each iteration decreases quickly down to 1.96 from 2.3025.

Nestrov Update:

The Nestrov Update is a slight variation of the momentum update. The "look ahead" value using momentum is calculated rather than the old value. The loss drastically decreases in loop 100 from 2.3 to ~1.72.

v_prev = v
v = mu * v - lr * dW
w += -mu * v_prev + (1 + mu) * v

- The value oscillates about the range 1.7-1.9
- Then finally the loss settles at the 1.70.
- This tends to provide better results for convex functions.

AdaGrad Update:

AdaGrad is an adaptive learning rate method. The AdaGrad is said to be an 'aggressive' update method, in that with a monotonic learning rate it stops learning after an early stage. The following is the formula followed to achieve an AdaGrad Update:

cache += dW ^ 2
W += -lr * dW / (np.sqrt(cache) + 1e-7

- The loss reduces drastically in this update. Starting at 2.3025 it reduces swiftly to 1.543.
- The loss tends to ‘slow’ down in its swift descent somewhere along the Loop 1100-1300, but by Loop 1400 it has arrived at the low loss of the entire 1500 iterations.
- The losses tend to decrease smoothly without much ado otherwise.
- The descent is swift once the oscillations around the loss of ~1.8 is passed in the Loop 1300.

<u>RMSProp Update:</u>

The RMSProp is a more sophisticated version of the AdaGrad Update. It tries to fix the faults in the AdaGrad by using a moving average of squared gradients to update the weights. The following is the formula used:

cache = decay * cache + (1-decay) * dW^2
#W += -lr*dW/(np.sqrt(cache)+1e-7)

- We notice that the loss tends to decrease in a uniform manner, moving away from a smooth descent just once in through the 1500 iterations.
- At the Iteration 1400 though, the loss increases from the low value of 1.59 to 1.78.

<u>Adam Update :</u>

The Adam Update essentially is defined to be like RMS Prop but with momentum. The following is the formula for Adam:

v = B1 * v + (1 - B1) *dW
cache = B2 * cache + (1 - B2)*(dW ^ 2) # gradient square
vb = v / (1 - B1 ^ t) # Bias corrected gradient
cacheb = cache / (1 - B2 ^ t) # Bias corrected gradient square
W -= lr* vb / [np.sqrt(cacheb) + 1e-7]

- The loss here decreases greatly from 2.3 to 1.6 in the Loop 100 itself.
- The loss tends to oscillate quite a bit before dropping to the best loss of around 1.59 at the final set of iterations.
- It tends to give a better result than the RMSProp under the same conditions.