Advanced Algorithms Project

Analysis of various back propagation algorithms on Neural Networks

JEEVANA R HEGDE – PES1201700633

SAAHIL B JAIN – PES1201700241

Problem Statement:

Analysing various back propagation algorithms on Neural Networks. The approaches we have chosen are:

1. Gradient descent
2. Genetic algorithm
3. ADAprop
4. RMSprop

Abstract

1. *Gradient descent*

   Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient.

2. *Genetic Algorithm - Exploitation and simple hill climbing*

   In this approach genetic algorithms randomly updates weights and it is randomly increased or decreased. The newly chosen weights are used to train the model and it is checked for accuracy. If the accuracy has improved, we update to make these the new weights and delete the old ones else we discard the new ones and revert to old ones. In case of local minima or saturation point(plateau) we jump to a random location and redo the training until a satisfactory accuracy is met.

3. *RMPprop*

   It is an unpublished optimization algorithm designed for neural networks. RMSprop lies in the realm of adaptive learning rate methods. Rprop tries to resolve the problem that gradients may vary widely in magnitudes. The central idea of RMSprop is keep the moving average of the squared gradients for each weight. And then we divide the gradient by square root the mean square. Which is why it's called RMSprop(root mean square).

```
drad_squared = 0
for _ in num_iterations:
    dw = compute_gradients(x, y)
    grad_squared = 0.9 * grads_squared + 0.1 * dx * dx
    w = w - (lr / np.sqrt(grad_squared)) * dw
```

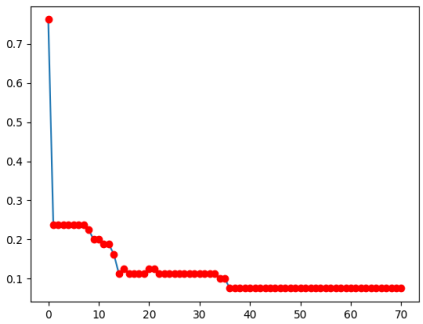
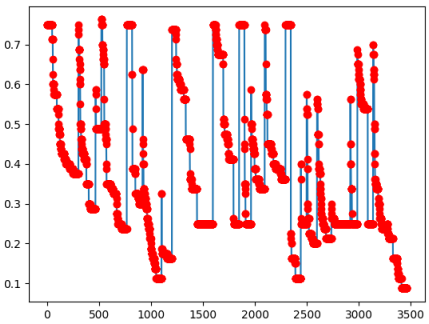
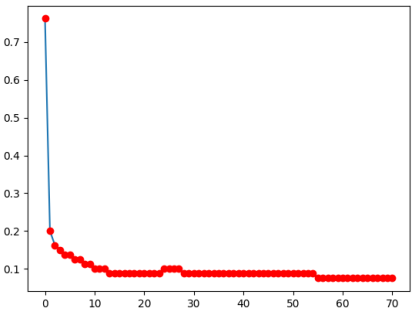
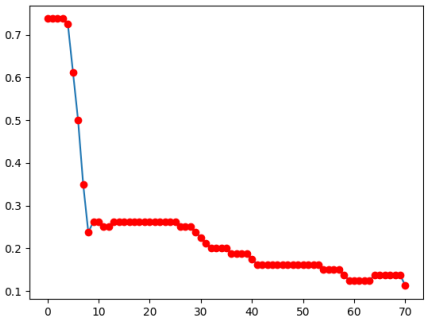
4. ***ADAgrad***

It is adaptive learning rate algorithms that looks a lot like RMSprop. Adagrad adds element-wise scaling of the gradient based on the historical sum of squares in each dimension. This means that we keep a running sum of squared gradients. And then we adapt the learning rate by dividing it by that sum.

```
grads_squared = 0
for _ in num_iterations:
    dw = compute_gradient(x, y)
    grad_squared += dw * dw
    w = w - (lr / np.sqrt(grad_squared)) * dw
```

Time and performance comparisons:

Algorithms	Train accuracy	Test accuracy	Execution time (in secs)
Gradient descent	92.50	90.47	1.0787
Genetic algorithm	93.75	95.24	8.8139
ADAgrad	96.25	80.95	1.4732
RMSprop	91.25	95.23	1.4999

Gradient descent	***Genetic algorithm***
	
ADAgrad	***RMSprop***
	

Conclusion:

1. Gradient descent always has the same and consistent learning curve but usually saturates at low values.
2. Although genetic algorithm takes more iterations, the time taken for each iteration is much lesser.
3. RMSprop performed better than traditional gradient descent although it had a more gradual curve.
4. ADAprop does not perform as well as traditional gradient descent.