CSL712 - Advanced ML

Lab1

Milan Chaudhari(2015csb1010)

The assignment was to implement and analyse logistic regression and a two layered traditional NN with different activation functions, optimization methods for CIFAR-10 dataset.

Note-1: For all the experiments Number of epoch is taken to be 100.
Note-2: The assignment have be implemented in python using tensorflow.

1. Analyse the effect of standardization of input data

The task was to analyse the performance of the model (input x 100 node dense layer x Relu/Sigmoid activation x 10 node dense layer x softmax activation) when the input data is standardized and when only the mean is subtracted from it.

Relu:

	Optimal learning rate	Training accuracy	Testing accuracy
Without std	3x1e-6	66.89%	43.86%
With std	3x1e-5	71.76%	50.95%

Sigmoid:

	Optimal learning rate	Training accuracy	Testing accuracy
Without std	3x1e-5	41.55%	39.59%
With std	3x1e-4	63.83%	40.21%

Observation:
Since the standard dev. in image pixel intensities will be almost equal among all the pixels, in the case of non-standardized data the input will be scaled by the same value for all the pixels, w.r.t. standardized data. So, the gradient will get scaled by a constant except the gradient value for the bias term, which causes a deformation in the curve of loss function.

In the case of Relu, since the gradient of activation function is same for all the previous layer nodes the deformation will be very small and it will go under some fluctuations after some iteration in the case of non-standardization. Which results in a small decay in accuracy.
While in the case of sigmoid activation, the gradient of activation function will be higher for the previous layer nodes which have values nearer to 0 and lower for others, which causes relatively large deformation, which results into more fluctuations and less accuracy.

2. Relationship between batch-size and learning rate

Here a logistic regression model is used, which can be viewed as input x 10 node dense layer x softmax model. The model also had regression applied with it, where *regression constant=0.1*.

BatchSize=200 :

Learning Rate	Training Accuracy	Test Accuracy
1e-7	26.69%	26.94%
1e-6	35.93%	34.75%
1e-5	45.54%	41.08%

BatchSize=100 :

Learning Rate	Training Accuracy	Test Accuracy
1e-7	26.93%	26.25%
1e-6	36.35%	34.98%
1e-5	45.40%	41.05%

Observation:
1. For lower learning rate, lower batch-size works better, as the gradients in the case of smaller batch-size will be higher, due to which the update in the weights will be larger.

2. For a given batch-size, with increasing learning rate, fluctuations gets increased and the loss do not decreases further, as the step size is higher, which causes problem in optimizing after some epoches. For higher batchsize, the learning rate when it starts fluctuating is higher, as we are using greater sample data for the loss function, due to which the variance decreases. So, with increase in batch-size higher learning rate can be used which increases the performance.

3. A two layered NN with Relu activation

The base model for this task was input x 100 node dense layer x relu/leakyRelu/sigmoid/tanh x dense layer(10) x softmax.

Learning rate	Training accuracy	Testing accuracy
1e-3	67.55%	41.29%
1e-4	76.87%	46.97%
1e-5	45.10%	41.89%

Observation:
For higher learning rate(1e-3 here), the loss function starts fluctuating with epoche number and do not get decreased. While lower learning rate(1e-5 here) takes more epoches to get decreased. So there is some medium learning rate which gives maximum accuracy.

4. Comparing different activation functions

For this task the network architecture is the same as that of task 3, with different activation function.

Activation fn	Training accuracy	Testing accuracy	# Dead nodes
ReLU	76.53%	47.55%	63
LeakyReLU	76.41%	47.68%	74
Sigmoid	52.70%	44.71%	89
Tanh	53.80%	46.32%	105

Observation:

1. Order of test accuracy is LeakyRelu > Relu > Tanh > Sigmoid, as relu and leakyrelu are more of linear activation functions having constant gradients while sigmoid and tanh are non-linear, due to which they have larger gradients near the value 0, while smaller for others. Due to this non-uniform gradients it underperforms.
2. LeakyRelu is better as compared to Relu, as Relu converts negative node values to 0, which makes the gradients 0 and the weights do not gets updated after that, due to which the weights just get stucks at some values and do not go it’s optimal value. There for, number of dead-nodes with LeakyRelu is greater than that with Relu, as eventually the values goes to 0.

3. Tanh is better than Sigmoid, as unlike sigmoid, the output of tanh can be -ve as well as +ve, due to which the learning becomes faster(as in the same iteration weights can increases as well as decreases). Number of dead-nodes with Tanh is less than that with Sigmoid, as in tanh(0)=0, which makes the contribution of a node 0 and make it inactive and the weight value gets stuck 0. While the output of sigmoid is always +ve, which forces the weights to update each time.

5. Comparing different optimization algorithms

	Optimal learning rate	Training accuracy	Test accuracy
SGD	1e-4	76.53%	47.55%
AdaGrad	1e-2	76.47%	50.83%
Adam	1e-4	73.17%	49.13%

Observation:

1. In the case of Adagrad, the denominator term in the update of weight becomes quite big which reduces the step size. So the learning rate that was used in SGD/Adam, do not work for AdaGrad. The optimal learning rate in the case of AdaGrad will be high as compare to others.
2. Adam underperforms in this case, as it takes exp. average of gradients, which takes it time to change it’s update orientation when it’s around the optimal value. Adam is good with outliers, but lacks in the case of smooth curve.