Adagrad - An Optimizer for Stochastic Gradient Descent

A. Agnes Lydia[#1], F. Sagayaraj Francis[*2]

[#]*Department of Computer Science and Engineering, Pondicherry Engineering College*

[1]agneslydia@pec.edu

[2]fsfrancis@pec.edu

***Abstract*— Gradient Descent algorithms though popularly used, it still remains to work as a black-box with much of the tuneable hyper-parameters being unexplored. These hyper-parameters employ proximal functions to control the gradient steps which supports for online and adaptive learning. The earlier algorithms required the manual initialization of the hyper-parameters before the training process begins, and it remains static throughout the training. By incorporating Optimizers for existing algorithms, hyper-parameter initialization and updation is automatically done by the algorithm. This article tends to give an insight about these hyper-parameters, its nature and the purpose of adding these hyper-parameters to enhance the performance of Gradient Descent Algorithms.**

***Keywords*— Neural Network, Backpropagation, Gradient Descent, Optimizer, Adagrad, Hyper-parameter**

I. INTRODUCTION

An *Artificial Neural Network* (ANN) is a computational model consisting of several processing units called the neurons. These neurons are associated with weights and biases which constructs the complete structure of the neural network [1]. Neurons receive inputs in the form of weights that is processed through a transfer function and provides an output value. Artificial neural networks are known for the ability to learn complex relationships between the inputs and the outputs of the system and are not restricted by the size and the complexity. Different types of neural networks are designed based on the transfer functions it operates with, the way the network trains, number of layers of neurons and by the way the neurons are interconnected [2]. In the present day's data, it is observed that the sample instances are of very high dimensions, yet only a few instances are with prominent features. In most cases, these infrequently occurring instances are the highly informative features. The existing Stochastic Descent methods follow a static procedural scheme to converge to the optimum value. This may lead to stagnated learning in case of unevenly distributed data. The Adagrad optimizer in contrast, modifies the learning rate adapting to the direction of the descent towards the optimum value. In other words, Adagrad maintains low learning rates for frequently occurring features and high learning rates for less frequently occurring features.

II. BACKPROPAGATION

In a multilayer neural network the sum of products of the input signals are fed into a transfer function to obtain the output from one particular neuron. A neural network is trained with the input values and the target patterns providing the network the ability to learn. In most multilayer neural networks the training algorithm used is *Backpropagation* [3]. The input values are propagated in the forward direction from first layer to the last layer through the network to optimize the neuron during this process. The difference between the received output and the expected output is the *Error value*. The weights of the neurons are then adjusted while propagating backwards from the last layer to the first layer to reduce the error value. The minimization of the error values occur in many training cycles called the *Epoch*. During each epoch, the accuracy of the network increases gradually [4]. The goal of *feature extraction* technique is to measure certain properties and attributes in the original data that distinguish one pattern from another [5]. These features are stored in the form of vectors and fed as input values to the network.

III. GRADIENT DESCENT

Gradient Descent is the most popular algorithm to optimize the neural networks. The objective of this algorithm is to minimize the objective function $J(\theta)$ with θ as the parameters, by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ [6]. Based on the amount of data the algorithm uses to compute the gradient, several variants of Gradient Descent algorithms have been developed. *Batch Gradient Descent* computes the gradient of the cost function with respect to θ for the entire dataset,

$$\theta = \theta - \eta \bullet \nabla_\theta J(\theta) \tag{1}$$

Since the parameter can be updated only once for the entire dataset, this algorithm convergence slowly towards the optimum value and it does not support to train with live datasets. In contrast, *Stochastic Gradient Descent* (SGD) updates the parameter for every training sample,

$$\theta = \theta - \eta \bullet \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)}) \tag{2}$$

Batch Gradient Descent tends to recomputed gradients of similar samples on each parameter update which results in redundancy. Stochastic Gradient Descent overcomes this redundant computation by performing one update at a time. This gives SGD the ability to learn live datasets and converge faster. But SGD greatly fluctuates for features with high variances. To reduce these fluctuations and also maintain the ability of faster convergence and online learning, Mini-Batch Stochastic Gradient Descent was developed,

$$\theta = \theta - \eta \bullet \nabla J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{3}$$

This variant of SGD performs an update for every mini-batch of n training samples, which gives the algorithm the stability to converge without fluctuations. The commonly implementable mini-batch sizes ranges from 50 to 256.

IV. OPTIMIZERS

Though several variants of training algorithms have evolved, the challenge lies in fine-tuning the hyper-parameters to redefine the algorithm to train the network as per the requirements of the dataset. These hyper-parameters are *Optimizers* that enhance the performance of the algorithm. SGD faces difficulties in navigating through surfaces that curves more steeper in one direction than in another direction (*Ravines*). This is handled by adding on the hyper-parameter, *Momentum*, which accelerated the descent in relevant directions and slows down in irrelevant directions.

$$v_t = \gamma \bullet v_{t-1} + \eta \bullet \nabla_{\theta} J(\theta) \tag{4}$$

where $\gamma \bullet v_{t-1}$ is the *momentum term*. But SGD with Momentum tends to get stuck on data that are unevenly distributed datapoints. *Nesterov Accelerated Gradient* (NAG) is a hyper-parameter that gives an insight on the direction in which the global optimum value lies. In this variant, $\theta - \gamma \bullet v_{t-1}$ is computed which gives an idea about the next approximate position of the parameters.

$$v_t = \gamma \bullet v_{t-1} + \eta \bullet \nabla_{\theta} J(\theta - \gamma \bullet v_{t-1}) \tag{5}$$

$$\theta = \theta - v_t \tag{6}$$

Momentum first computes the present gradient and then takes a leap in the direction of the updated accumulated gradient. Whereas NAG first takes a leap in the direction of previous accumulated gradient, measures the gradient and then makes a correction. This form of anticipatory update avoids faster convergence in irrelevant direction and also increases the performance. *Adagrad* is a variant of Stochastic Gradient Descent which changes its learning rate for different parameters, hence the term *Adaptive Gradient Descent Algorithm*. In 2011, Adagrad was published by John Duchi et.al [7]. This algorithm adapts a larger learning rate updates for infrequent and smaller updates for frequent parameters, which makes it suitable for sparse data [8] [9]. Adagrad implements different learning rate for every parameter θ_i at every time step, t. The per parameter update is written as

$$g_{t,i} = \nabla_{\theta_t} J(\theta_{t,i}) \tag{7}$$

The Stochastic Gradient Descent update then changes as,

$$\theta_{t+1,i} = \theta_{t,i} - \eta \bullet g_{t,i} \tag{8}$$

Adagrad modifies the general learning rate η at each time step t for every parameter θ_i based on the past gradients that have been computed for θ_i,

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \varepsilon}} \bullet g_{t,i} \tag{9}$$

in which ε is the quantity added to avoid division by zero, $g_{t,i}$ is the gradient of the loss function with respect to the parameter $\theta(i)$ at time step t, $G_{t,i}$ is a diagonal matrix where each diagonal element i, and i is the sum of the squares of the gradients with respect to θ. There are also other optimizers that provide similar performances to Adagrad like Adadelta, RMSProp and Adam.

V. RESULTS

The above discussed optimizers were implemented on different image datasets like COIL-100 [10], Caltech-101 [11], MNIST [12] and the results exhibit Adagrad providing better results. *COIL-100* is a dataset of 100 different objects captured on a darker background. Each object has 72 instances of images taken in different orientations. *Caltech-101* is a dataset of 101 different objects taken in natural scenes. *MNIST* is dataset of scanned images of handwritten numbers with 60,000 instances.

TABLE.1 : Results of Performances of different Optimizers

DATASETS / OPTIMIZERS	COIL-100		CALTECH-101		MNIST	
	Train_Acc	Val_Acc	Train_Acc	Val_Acc	Train_Acc	Val_Acc
Adagrad	0.9999	0.8352	0.9986	0.9612	0.9329	0.9272
Adadelta	0.9992	0.8352	0.9811	0.9501	0.9320	0.9268
RMSProp	0.9978	0.8300	0.9622	0.9330	0.9294	0.9269
Adam	0.9965	0.8155	0.9610	0.9305	0.9215	0.9211
SGD	0.9731	0.7971	0.9527	0.9211	0.9024	0.9102

VI. CONCLUSION

With the advent of Gradient Descent being used to train any kind of neural network, this work is focused on highlighting the available alternatives and hyper-parameters that can be considered to enhance the performance of the existing algorithms. From the experiments conducted, the Optimizer Adagrad outperforms all the other optimizers and enhances the Stochastic Gradient Descent Algorithm while trained using image datasets of different nature.

REFERENCES

1. S. Haykin, Neural Networks- A Comprehensive Foundation (2nd ed., Pearson Prentice Hall, 2005).
2. Deng, Li, Geoffrey Hinton, and Brian Kingsbury. "New types of deep neural network learning for speech recognition and related applications: An overview." *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013.
3. R. Roja, The Backpropagation Algorithm, Chapter 7: Neural Networks (Springer-Verlag, Berlin, 1996) pp. 151-184.
4. Setiono, Rudy, and Huan Liu. "Neural-network feature selector." *IEEE transactions on neural networks* 8.3 (1997): 654-662.
5. Sharma, Bhavna, and K. Venugopalan. "Comparison of neural network training functions for hematoma classification in brain CT images." IOSR Journal of Computer Engineering 16.1 (2014): 31-35.
6. Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747(2016).
7. Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research* 12.Jul (2011): 2121-2159.
8. Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large Scale Distributed Deep Networks. NIPS 2012: Neural Information Processing Systems, pages 1–11, 2012.
9. Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global Vectors for Word Representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, pages 1532–1543, 2014.
10. Khalid, EL ASNAOUI, et al. "An efficient Descriptor for Image Retrieval: Application to COIL-100 Database."
11. Griffin, Gregory, Alex Holub, and Pietro Perona. "Caltech-256 object category dataset." (2007).
12. Deng, Li. "The MNIST database of handwritten digit images for machine learning research [best of the web]." *IEEE Signal Processing Magazine* 29.6 (2012): 141-142.