ADAPTIVE GRADIENT DESCENT OPTIMIZATION ALGORITHM

Atharva Subhash Gokhale
Student ID:20862183

I. INTRODUCTION

Supervised learning is an area of significant academic and industry interest due to its application in engineering, business, medical sciences and all those domains which demand prediction or classification based on training examples. The primary objective of each supervised learning model is to minimize the error between predictions and reality. Gradient descent optimization algorithm and its successors are widely used to minimize the error function by optimally updating the weights of the learning model. Amongst the basic variants of gradient descent algorithm, the mini-batch Stochastic Gradient Descent (SGD) Algorithmic produces best results and is most widely used. According to [11] SGD picks one instance from the training set at every step and updates the gradient based on the single record. Thus, the algorithm give dynamic insights into the model and is much faster at every iteration which overcomes the drawback of batch gradient descent which recomputes gradient of similar examples before each update. Also, the frequent updates results in oscillations that can assist the algorithm to get out of the local minima of the loss function. On other hand these oscillation leads to overshooting on approaching the local or global minima of the loss function. For large data sets it is practically impossible to consider each data point before each update. Thus, we choose a limited number of data points at a time before each update .This batch of data points is selected randomly from the data set. This random selection introduces noise which leads to fluctuations and complicate the convergence towards the exact minimum as the SGD will keep overshooting. This drawback of mini batch SGD can be compensated by decreasing the learning rate eventually when the algorithm approaches the global minima which leads to accurate convergence for convex and non-convex functions. It is obvious that decreasing the learning rate will lead to painful and slow convergence of the error function. We can adopt few extension algorithms of SGD that accelerate the process of convergence and overcome few drawbacks of SGD.
Other possible solution is to adapt to the optimization algorithms that facilitate dynamic and different learning rate for every dimension of the data set. All the algorithms in the domain of deep neural networks with adaptive learning rate are originally proposed for convex functions and show $O(\sqrt{T})$-type regret bounds. Specifically *ADAGRAD* algorithm proposed by [2] have different learning rate for each feature by analyzing the gradients during the process of optimizing the loss functions. Experimentation of [2] is limited to all convex functions and achieves data dependent regret bounds of $O(\sqrt{T})$. According to [3] there exits few extensions of SGD or online gradient descent which achieves logarithmic regret bound. It is my attempt to modify the original *ADAGRAD* algorithm specifically for strongly convex problems to achieve logarithmic regret bound in accordance with [7]. Later, the use of updated version of *ADAGRAD* to train the deep neural networks for better test accuracy will serve as the potential innovative solution as it is orginally proposed only for stongly convex problems. In this paper I will traverse to the updated *ADAGRAD* optimizer starting from the basic SGD and subsequent extensions to understand the drawback in each version and solution to overcome it.

II. LITERATURE REVIEW

According to [11] stochastic gradient algorithm have problem when it encounters ravines; the area where the surface curves much more steeply in one dimension, which are observed very commonly near local minima. SGD oscillates across the slopes and is resistant to reach local minima. According to [10] there are certain accelerating methods which significantly improve the performance of SGD.

A. *MOMENTUM*

We can use Momentum [9] method to accelerate the process of convergence in the right direction to assist SGD to accelerate the process of convergence for complex non-convex error functions by introducing a momentum constant γ = 0.9 for most conditions. The method proposes to add a constant that assists to average out the oscillations along the short axis along with contribution along the long axis according to [12].

$$\theta^{(t+1)} = \theta^{(t)} - [\gamma v^{(t-1)} + \alpha \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})] \tag{1}$$

B. *NESTEROV'S ACCELERATED GRADIENT ALGORITHM*

Nesterov's Accelerated Gradient (NAG) Algorithm [8] give the momentum term the necessary prescience about where to stop so as to converge. In Momentum we used $\gamma v^{(t-1)}$ to update the parameters. Thus, calculating $\theta - \gamma v^{(t-1)}$ will give approximate next position of the parameter. Therefore, we can look ahead at the position of the updated parameters beforehand and accordingly take the necessary steps.

$$\theta^{(t+1)} = \theta^{(t)} - [\gamma v^{(t-1)} + \alpha \cdot \nabla_\theta J((\theta - \gamma v^{(t-1)}); x^{(i)}; y^{(i)})] \tag{2}$$

According to [1] NAG initially takes a big stride in the direction of the previous accumulated gradient, measures the gradient and then makes a correction, which results in the complete NAG update which improves the performance. Now we are able to adapt the learning rate to the slope of the loss function.

C. *ADAGRAD*

According to [2] we can determine the adaptive learning rate depending upon the frequency of the parameters associated with the data features. We choose low learning rates (i.e. smaller updates) for high frequency features while high learning rate for infrequent features. In other words the algorithm scales the learning rate for each dimension of the network. Let g^t be the gradient at step t, while g_i^t is the partial derivative of the the error function w.r.t each parameter θ_i at step t: $g_i^t = \nabla_\theta J(\theta_i^t)$

$Adagrad$ modifies the learning rate at each step t and for each parameter θ_i based on the past gradient that have been calculated for θ_i:

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\alpha}{\sqrt{\alpha_{(t)} + \epsilon}} \cdot g_i^t \tag{3}$$

where

$$\alpha_{(t)} = \sum_{i=1}^{t} \left(\frac{\partial J}{\partial \theta_i}\right)^2$$

Let us proceed to the next section of the paper where we will discuss the *ADAGRAD* again in detail from the perspective of online convex optimization problem and prove that is show regret bound of $O(\sqrt{T})$ with all necessary mathematical explanation which can be further extended to achieve regret bound of $O(log(T))$ for strongly convex problems.

III. Proposed Solution

A. *NOTATIONS AND MATHEMATICAL PRELIMINARIES*

The content of this document is aligned with [7]. Vectors and scalars are lower case italic letter, $x \in X$. The sequence of vectors is denoted by subscript, $x_t, x_{t+1}, ...$ and entries of each vector by additional subscript, $x_{t,j}$. Let $A \in \mathbf{R}^{dxd}$ be the symmetric positive definite matrix.

$$\langle x, y\rangle_A = \langle x, Ay\rangle = \sum_{i,j=1}^{d} A_{i,j} x_i y_j, \tag{4}$$

$$\|x\|_A = \sqrt{\langle x, x\rangle_A}$$

The Cauchy-Schwartz inequality can be represented as, $\langle x, y\rangle_A = \|x\|_A \|y\|_A$. The element-wise product of two vectors, a and b such that $a, b \in \mathbf{R}^d$ is defined as: $a \odot b = a_i b_i$ for $i = 1, 2, 3, ..., d$ The weighted projection of $z \in \mathbf{R}^d$ on convex set C is given as:

$$P_C^A = arg \min_{x \in c} \|x - z\|_A^2 \tag{5}$$

B. *OPTIMIZATION PROBLEM*

An online convex optimization problem can be represented as repetitive game between the player and his opponent. At iteration t the player chooses an action $x_{(t)}$ from a convex subset C of $\mathbf{R}^{\mathbf{n}}$ and the opponent chooses a convex loss function f_t. The player is suppose to choose $x_{(t)}$ such that the total loss $\sum_{i=1}^{T} f_t(x_t)$ is not much larger than the smallest total loss $\sum_{i=1}^{T} f_t(x)$ i.e. the minima of the loss function. The difference between the total loss and its optimal value for a particular action is termed as *Regret*. The goal of each optimizer is to minimize the loss function by performing well with respect to the optimal decision in hindsight. The Adversarial Regret at time T is given as:

$$R(T) = \sum_{i=1}^{T} f_t(\theta_t) - \min_{x \in c} \sum_{i=1}^{T} f_t(\theta) \tag{6}$$

Our first objective is to prove that the strictly convex functions can have regret to grow logarithmically with *T*, where *T* denotes number of iterations. According to [4] algorithm that achieves logarithmic regret, should necessarily know beforehand the lower bound on the convexity of the loss functions, since this bound is used to determine the step size. In simple words our objective is to reduce the learning rate to order of $(\frac{1}{T})$ instead of in order of $\frac{1}{\sqrt{T}}$ so that the learning rate decreases relatively quicker per iteration to get logarithmic regret bound for the strongly convex problems.

Online Gradient Descent (OGD) [14] is the first method for which the regret grows as $\sqrt{T}$, where *T* is the number of rounds in the game. In other words, the algorithm achieves the optimal regret bound of $O(\sqrt{T})$ for convex optimization problems. OGD is quite similar to stochastic gradient descent as both algorithms picks up one data sample for every update. Still I will redefine the algorithms in terms of online gradient descent algorithm. The update equation for OGD is:

$$\theta^{(t+1)} = P_C(\theta^{(t)} - \alpha_t g^t) \tag{7}$$

where α_t = $\frac{\alpha}{\sqrt{t}}$ is the adaptive learning rate depending upon the number of iterations and g^t is sub-gradient of f_t with respect to θ_t. According to [3] we can achieve logarithmic regret bounds for the online gradient descent algorithm with $\alpha_t = \frac{\alpha}{t}$ for strongly convex problems. A strongly convex function is that for which there strictly exits a quadratic lower bound on growth of the function. For an optimization algorithm to achieve logarithmic regret bound, it is necessary for the algorithm to know the lower bound on convexity of the loss function, as this bound is later used to calculate the learning rate.

C. *ADAGRAD FOR ONLINE CONVEX PROBLEMS*

I will refer to [2] for the outlines of the *ADAGRAD* algorithm but will prefer to use the terms from online convex optimization domain to maintain the uniformity for the further sections. In the problem of online convex optimization, if the opponent is allowed to choose from the set of all possible convex functions on *C*, then *ADAGRAD* will have regret bound of $O(\sqrt{T})$ according to [2].

Algorithm 1: ADAGRAD ALGORITHM

```
Input: $\theta \in C, \delta > 0, v_0 = 0 \in \mathbf{R^d}$
for $t = 1$ to $T$ do
    $g_t \in \partial f_t(\theta_t)$;    Partial derivative of the loss function w.r.t to parameter $\theta_t$ ;
    $v_t = v_{t-1} + (g_t \odot g_t)$ ;
    $A_t = diag(\sqrt{v_t}) + \delta$;    $\delta$ is small positive number to avoid division by 0 ;
    $\theta^{(t+1)} = P_C^A(\theta^{(t)} - \alpha A_t^{-1} g_t)$    Final update equation
end
```

Let $g_{1:T,i} = (g_{1:T,i}, g_{2:T,i}, g_{3:T,i})^T$ where, $g_{t,i}$ is the *i*-th component of gradient $g_t \in \mathbf{R}^d$ of the loss function f_t computed for θ^t. We can confirm that the algorithm is computing the updated weights based on the adaptive learning rate which is different for every feature of the data set. This particular methodology is instrumental in achieving better results for those data sets which have a mixture of dense and sparse data features. Parameters linked to sparse features receive significant updates only when the algorithm encounters a non-zero value. Considering a constant decreasing learning rate, it is possible that the parameters of dense features converge to the minima of the loss function but those of sparse features are still away from the optimal value. Thus, adaptive learning rate specific to each feature is a necessity for accurate convergence. Next, I will mathematically show that the *ADAGRAD* orginally has regret bound of $O(\sqrt{T})$ for convex functions.

Let us consider two assumptions necessary for further computations :

1: If $sup_{t\geq 1} \|g_t\|_2 \leq G$ then we have a constant G_∞ such that, $sup_{t\geq 1} \|g_t\|_\infty \leq G_\infty$

2: If $sup_{t\geq 1} \|\theta_t - \theta^*\|_2 \leq D$ then we have a constant D_∞ such that, $sup_{t\geq 1} \|\theta_t - \theta^*\| \leq D_\infty$ where D denotes the diameter of the underlying convex set C given by:

$$D = \max_{x,y\in C} \|x - y\|_2 \tag{8}$$

Now, in accordance with [2] and considering above two assumptions, let θ_t be sequence of parameters generated by *Algorithm 1*, where $g_t \in \partial f_t(\theta_t)$ and $f_t : C \to \mathbf{R}$ is a random convex function. According to [7] for learning rate $\alpha > 0$, the regret is upper bounded as:

$$R(T) \leq \frac{D_\infty^2}{2\alpha} \sum_{i=1}^{d} \|g_{1:T,i}\|_2 + \alpha \sum_{i=1}^{d} \|g_{1:T,i}\|_2 \tag{9}$$

All positive definite matrices A referred in this paper are diagonal matrices to maintain computational complexity of calculating inner products and norms linear in *d*. The diagonal elements of the matrix A are the squared value of summation of sub-gradient of the loss function with respect to each parameter. Now, the learning rate α is a function of matrix A. Thus, $(A_t)^{-1}$ will be a diagonal matrix with values corresponding to $\dfrac{1}{\sqrt{v_{t,i}} + \delta}$. Then the learning rate α can be defined as:

$$\alpha(A_T^{-1})_{ii} = \frac{\alpha}{\sqrt{\sum_{t=1}^{T} g_{t,i}^2} + \delta} = \frac{\alpha}{\sqrt{T}} \frac{1}{\sqrt{\frac{1}{T}\sum_{t=1}^{T} g_{t,i}^2} + \dfrac{\delta}{\sqrt{T}}} \tag{10}$$

We can confirm that *ADAGRAD* have decaying step size $\frac{\alpha}{\sqrt{t}}$. Along with that we can observe that the the constant positive term added is a running average of squared derivatives plus vanishing damping term according to [7]. Now, to achieve logarithmic regret bound for strongly convex function the adaptive learning rate should be of $O(\frac{1}{T})$ according to [3].

D. ***STRONGLY CONVEX ADAGRAD (SC-ADAGRAD)***

Our primary aim is to achieve logarithmic regret bound for strongly convex problems using *SC-ADAGRAD* algorithm. According to [3] online projected descent uses learning rate of order of $\frac{1}{T}$ to achieve logarithmic regret bounds for strongly convex functions. I attempt to do similar modification in original *ADAGRAD* proposed by [2] in accordance with [7]. Again considering the above mentioned equation for learning rate α, we present the equation for learning rate for *SC-ADAGRAD*.

$$\alpha(A_T^{-1})_{ii} = \frac{\alpha}{\sum_{t=1}^{T} g_{t,i}^2 + \delta} = \frac{\alpha}{T} \frac{1}{\frac{1}{T}\sum_{t=1}^{T} g_{t,i}^2 + \dfrac{\delta}{T}} \tag{11}$$

The denominator of the above equation contains running average of previous gradients and decaying damping factor. Now, the resultant learning rate is of $O(\frac{1}{T})$. The algorithm for *ADAGRAD* is as follows:

Algorithm 2: SC-ADAGRAD ALGORITHM: FOR STRONGLY CONVEX FUNCTIONS

```
Input: $\theta \in C, \delta_0 > 0, v_0 = 0 \in \mathbf{R^d}$
for $t = 1$ to $T$ do
    $g_t \in \partial f_t(\theta_t)$;        Partial derivative of the loss function w.r.t to parameter $\theta_t$ ;
    $v_t = v_{t-1} + (g_t \odot g_t)$;
    $Choose\ 0 < \delta_t \leq \delta_{t-1}$     element wise;
    $A_t = diag(\sqrt{v_t}) + \delta$ ;
    $\theta^{(t+1)} = P_C^A(\theta^{(t)} - \alpha A_t^{-1} g_t)$
end
```

We can observe two major differences in *SC-ADAGRAD* as compared to *ADAGRAD*. The first

and the major one is that the step size is now in order of $O(\frac{1}{T})$ and the damping factor is now function of t. We introduce the constant δ in *ADAGRAD* to avoid the scenario where $g_{t,i}$ may approach zero in the initial iterations and the algorithm encounters division by zero. The value of this constant is considered to be infinitesimally small in *ADAGRAD*. In case of *SC-ADAGRAD*, for inintial iteration, where $g_{t,i}$ is very small, say of order ϵ. Then we update this constant as, $\dfrac{\epsilon}{\epsilon_2 + \delta}$, where δ is a very small number which increases the magnitude of overall constant and introduces instability. This would be the probable reason that according to [2], it was not possible to consider the learning rate of $O(\frac{1}{T})$.
It is much preferred to initialize δ of order of 1 and then allow it to decay as $v_{t,i}$ increases in magnitude. This, adaptive constant can be governed using following equation:

$$\delta_{t,i} = \xi_2 e^{-\xi_1 v_{(t,i)}}, \quad i = 1, 2, 3...., d \tag{12}$$

where $\xi_1 > 0$ and $\xi_2 > 0$. It is also possible for one to consider $\xi_1 = 0$ and $\xi_2 > 0$ which obviously proposes a constant decay factor. From the mathematical analysis given by [7], using decay scheme with $\xi_1 > 0, \xi_2 > 0$ is much preferred as it proves to be adaptive to specific dimension and thus makes the overall algorithm adaptive. Now, with above mentioned decaying scheme and in accordance with [7], the regret bound for *SC-ADAGRAD* is given as follows:

$$\begin{aligned} R(T) \le \frac{dD_\infty^2 \xi_2}{2\alpha} - \frac{\alpha}{2} log(\xi_2 e^{-\xi_1 G_\infty^2}) + \frac{\alpha}{2} \sum_{i=1}^{d} log(\left\|g_{1:T,i}^2\right\| + \\ \xi_2 e^{-\xi_1 \left\|g_{(1:T,i)}\right\|^2} + \frac{\alpha \xi_1 \xi_2}{2log(\xi_1 \xi_2 + 1)} \sum_{i=1}^{d} (1 - e^{-\xi_1 \left\|g_{(1:T,i)}\right\|^2}) \end{aligned} \tag{13}$$

Observing the above equation, it is clear that adaptive learning rate of order of $o(\frac{1}{T})$ achieves logarithmic regret bound. It is possible that, the constant decay factor $\xi_1 = 0$ achieves better regret bounds. According to [7] it is possible that the third term in the equation of regret bound with constant decaying term may become negative. If this particular problem is improved then may be constant decay will achieves better results without any restrictions. I will use the *SC-ADAGRAD* to train the neural networks using Keras as the top level impelementation platform.

IV. Experimental Results

In this section I will experimentally show that the *SC-ADAGRAD* algorithm achieves logarithmic regret bound for a strongly convex loss function for data sets under consideration. Later, I will use the *SC-ADAGRAD* algorithm for training the Convolutional Neural Networks (CNNs) and compare the training loss and test accuracy with *SGD*, *ADAGRAD*, *ADADELTA* [9] and *ADAM* proposed by [5] which is considered as the most state of art technique of optimization. I have used *Keras* as a back-end platform to execute all these algorithms and perform their comparative analysis.

A. ***DATA SETS***

I have used 2 publicly available data sets for comparison of all the optimization algorithms. The first data set is the *MNIST* (Modified National Institute of Standards and Technology database) data set proposed by[6] which is basically a data set of 28 x 28 images of handwritten digits from 0 to 9. The dataset includes 60,000 training samples and 10 target classes. The next data set *Fashion-MNIST* proposed by [13] which is basically a data set of 28 x 28 images of 10 fashion items such as shirts, trousers, sandals, shoes, jackets etc. with 60,000 training samples

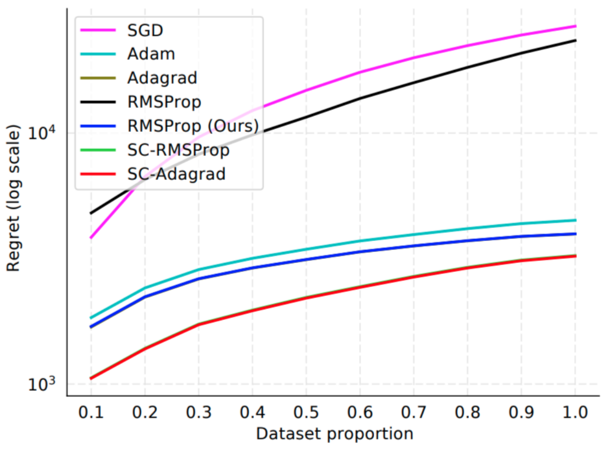

Fig. 1: Regret Vs Data Proportion: MNIST Data set [13]

and 10 target classes. The primary reason for selecting these data sets is that, the MNIST data set includes many sparse dimensions as many pixel values are zero for a handwritten digits. On the other side, the Fashion-MNIST data set includes images of clothes and footwear which have majority non-zero pixels as all the available pixels are used to display the dense image. Also, the subsidiary reason is the computational restrictions to visualize more complex and heavy data sets.

*B. **STRONGLY CONVEX PROBLEM***

According to [4] the key technique of forming a strongly convex function is regularization. Thus, we add a uniformly convex function $\lambda \|x\|^2$ to each loss function $f_t(x)$. Consider a data set with training samples $(x_i, y_i)_{i \in [m]}$ where x_i are the data features, y_i represents the target labels for training samples. m is the number of training samples. Let $y_i \in K$. In this case I will fit a linear model with cross entropy loss and use squared euclidean norm regularization of the weights in the loss function of the deep neural network. Thus, according [7] the modified loss function to be minimized can be defined as:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} log \left(\frac{e^{\theta_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{K} e^{\theta_j^T x_i + b_j}} \right) + \lambda \sum_{k=1}^{K} \|\theta_k\|^2 \qquad (14)$$

All the weights are initialized to zero. The regularization process is fine tuned to achieve best possible accuracy for the test data set. I experimented with few values of the initial learning rate and finally used the best one to plot the regret value. The experiment of regret bound is performed in the online setting where the number of epochs are limited to number of training samples. I will plot the regret value in logarithmic scale versus data proportion. It is clearly observed that the *SC-ADAGRAD* algorithm exhibits best performance as compared to other optimization algorithms considering figure 1.

*C. **CONVOLUTIONAL NEURAL NETWORK***

I have designed a basic convolutional neural network (CNN) with following layers using Keras platform for both the data sets. The network includes 2 convolutional layers, first with 128 filters, second with 64 where the filter size for each layer is 3 x 3 and activation function used is Rectified Linear Unit (ReLu), then the next layer is a MaxPooling layer with window size 3 x 3,

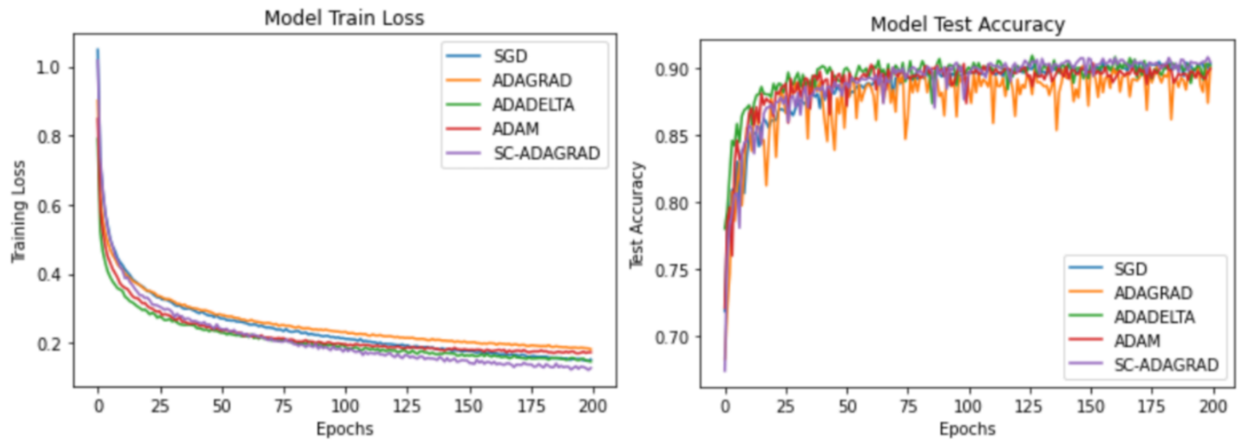

Fig. 2: MNIST Data set [6]: Training Loss and Test Accuracy

followed by a dropout layer with probability 0.5. Then the network includes 2 fully connected dense layers again with ReLu activation function and finally a dense layer with number of units equal to number of target class with softmax activation function. The implementation of all the optimizers is also done using the Keras platform.
Observing the figure 2, we can coclude that the *SC-ADAGRAD* optimization algorithm exhibits better convergence while training the deep neural network for sparse data set. The improvement shown in minimizing the loss function is considerable in comparison to other optimizers. On the other side, the *SC-ADAGRAD* optimizer achieves competitive test accuracy in comparison to all other optimizers. Mostly, importantly it surpasses the performnace of *ADADELTA* and *ADAM* which are considered as the successors of the original *ADAGRAD* algorithm proposed by [2]. Figure 3 shows the performance of *SC-ADAGRAD* for dense data set with majority non-zero feature values. It is observed that, almost all the optimizers exhibit similar convergence while training the neural network. Still, if we consider the details, *SC-ADAGRAD* shows best convergence for 200 epochs. In case of test accuracy, again the performance of all the optimizers is quite similar, yet again *SC-ADAGRAD* achieves that extra mile to deliver the best performance in predicting the target labels for the test data.

TABLE I: Comparison of Optimization algorithms for MNIST Data set

Algorithm	Time Complexity (sec)	Test Accuracy
SGD	36	0.997
ADAGRAD	33	0.991
ADAM	30	0.993
SC-ADAGRAD	29	0.998

V. CONCLUSION

The paper starts with the basic stochastic Gradient Algorithm which is used to optimize the weights of the neural network for accurate prediction of the unseen data. There are certain drawback in *SGD* which are highlighted in the scenario where the application demands for highly accurate results along with optimum time and computational complexity. Thus, I reviewed few extensions of *SGD* such as *Momentum* and *NOG* speeds up the process of convergence

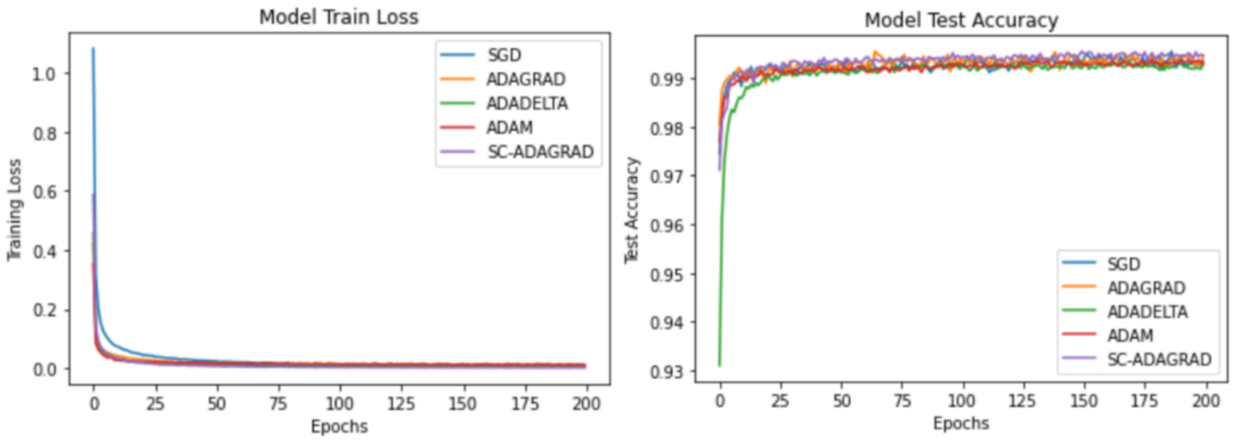

Fig. 3: Fashion MNIST Data set [13]: Training Loss and Test Accuracy

yet lacks to achieve better accuracy. The foundation of this paper is the *ADAGRAD* algorithm which adapts the step size of updating the weight specifically for each feature of the data set and also with respect to each iteration. This algorithm thus provides a big leap in context of accuracy of prediction of the network. Now, to upgrade the performance of the *ADAGRAD* algorithm I suggest to use its counter part used specifically to optimize the strongly convex loss functions. The first objective is to achieve logarithmic regret bound of the strongly convex problem which in turn helps to fix the constraints of the algorithm. In accordance with [7] the updated version *SC-ADAGARD* successfully achieved the logarithmic regret bounds for the data sets under consideration. The process of making the loss function strongly convex in order to achieve the logarithmic regret bound was possible by the process of regularization using the squared euclidean norm. Now, the updated version exhibits regret bound of $O(log(T))$ in comparison to the original *ADAGRAD* algorithm proposed by [2] which could achieve regret bound of $O(\sqrt{T})$. Also the updated *SC-ADAGRAD* outperformed all the earlier version of *SGD* and surprisingly also showed better regret bound than the *ADAM* optimizer which is the state of art model and also overcomes the major drawbacks of *ADAGRAD* which is accumulation of squared gradients in the denominator of the learning rate. Thus, it is obviously attractive to use this optimization algorithm to train the convolutional neural network to achieve better test results.

The *SC-ADAGRAD* algorithm showed promising result as expected even for applications involving deep neural networks. It is observed that the algoeithm improved the training and testing accuracy for the data sets under consideration (*IRIS* and *MNIST*). For considering the factor of dense and sparse data features, I have specifically considered the iris data set which includes relatively dense feature value and the other data set, MNIST which has a mixed properties of all the features. We can partially justify the improvement this improve on the grounds of achieving logarithmic regret bound for strongly convex functions. Although this reason do not serve firm platform for the improvement observed in train and test accuracy in deep neural networks. Also, *ADAM* being the improved version of *ADAGRAD* in general, the *SC-ADAGRAD* achieved better training and test accuracy as compared to *ADAM*. In my future work I would like to mathematically link the concept of strongly convex functions and the loss functions observed in neural networks to establish a theorotical proof which explains the improvement observed by using the *SC-ADAGRAD* algorithm in the domain of neural networks.

References

[1] Y. Bengio, N. Boulanger-lew, R. Pascanu, and U. Montreal, *Advances in optimizing recurrent networks*, in In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on.

[2] J. Duchi, E. Hazan, and Y. Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of Machine Learning Research, 12 (2011), pp. 2121–2159.

[3] E. Hazan, A. Kalai, S. Kale, and A. Agarwal, *Logarithmic regret algorithms for online convex optimization*, in Learning Theory, G. Lugosi and H. U. Simon, eds., Berlin, Heidelberg, 2006, Springer Berlin Heidelberg, pp. 499–513.

[4] E. Hazan, A. Rakhlin, and P. L. Bartlett, *Adaptive online gradient descent*, in Advances in Neural Information Processing Systems 20, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, eds., Curran Associates, Inc., 2008, pp. 65–72.

[5] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, Y. Bengio and Y. LeCun, eds., 2015.

[6] Y. LeCun and C. Cortes, *MNIST handwritten digit database*, (2010).

[7] M. C. Mukkamala and M. Hein, *Variants of rmsprop and adagrad with logarithmic regret bounds*, 2017.

[8] Y. NESTEROV, *A method for unconstrained convex minimization problem with the rate of convergence o(1/k*^{2}*)*$, Doklady ANUSSR, 269 (1983), pp.\ 543--547.$

[9] N. Qian, *On the momentum term in gradient descent learning algorithms.*, Neural Networks, 12 (1999), pp. 145–151.

[10] S. Ruder, *An overview of gradient descent optimization algorithms*. https://ruder.io/optimizing-gradient-descent/index.html#fn7, 1999.

[11] S. Ruder, *An overview of gradient descent optimization algorithms.*, 2016. cite arxiv:1609.04747Comment: Added derivations of AdaMax and Nadam.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Representations by Back-propagating Errors*, Nature, 323 (1986), pp. 533–536.

[13] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms*, (2017).

[14] M. Zinkevich, *Online convex programming and generalized infinitesimal gradient ascent.*, in ICML, T. Fawcett and N. Mishra, eds., AAAI Press, 2003, pp. 928–936.