A Strategic Weight Refinement Maneuver for Convolutional Neural Networks

1st Patrick Sharma
School of Computing,Information and Mathematical Sciences
Faculty of Science, Technology and Environment
The University of the South Pacific
Suva, Fiji
patricksharma610@gmail.com

2nd Adarsh Karan Sharma
School of Computing, Information and Mathematical Sciences
Faculty of Science, Technology and Environment
The University of the South Pacific
Suva, Fiji
adarsh.karan18@gmail.com

3rd Dinesh Kumar
School of Computing,Information and Mathematical Sciences
Faculty of Science, Technology and Environment
The University of the South Pacific
Suva, Fiji
kumar_di@usp.ac.fj

4th Anuraganand Sharma
School of Computing,Information and Mathematical Sciences
Faculty of Science, Technology and Environment
The University of the South Pacific
Suva, Fiji
anuraganand.sharma@usp.ac.fj

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533359

***Abstract*—Stochastic Gradient Descent algorithms (SGD) remain a popular optimizer for deep learning networks and have been increasingly used in applications involving large datasets producing promising results. SGD approximates the gradient on a small subset of training examples, randomly selected in every iteration during network training. This randomness leads to the selection of an inconsistent order of training examples resulting in ambiguous values to solve the cost function. This paper applies Guided Stochastic Gradient Descent (GSGD) – a variant of SGD in deep learning neural networks. GSGD minimizes the training loss and maximizes the classification accuracy by overcoming the inconsistent order of data examples in SGDs. It temporarily bypasses the inconsistent data instances during gradient computation and weight update, leading to better convergence at the rate of $O(\frac{1}{\rho T})$. Previously, GSGD has only been used in the shallow learning networks like the logistic regression. We try to incorporate GSGD in deep learning neural networks like the Convolutional Neural Networks (CNNs) and evaluate the classification accuracy in comparison with the same networks trained with SGDs. We test our approach on benchmark image datasets. Our baseline results show GSGD leads to a better convergence rate and improves classification accuracy by up to 3% of standard CNNs.**

***Index Terms*—Deep Learning, Convolutional Neural Networks, Stochastic Gradient Descent**

I. Introduction

Deep Learning, an area of high interest in the machine learning research community, simulates the human brain mechanism for interpretation of text, voice and visual data. It broadly encompasses three classes of algorithms, namely deep Feed-Forward Networks (FFN), Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN) that have been successfully applied in various domains such as natural language processing, robotics, health care and computer vision [13]. Unlike shallow networks, Deep Learning accounts for the use of multiple layers and hidden neurons, allowing an extensive coverage and higher perceptual level of the data at hand, resulting in a high-level abstraction of raw data or images [9].

This paper focuses on the improvement of the classification accuracy of CNN with a strategic gradient descent approach. CNN is predominantly used for identification and recognition of images and is commonly termed as the computer vision. It is mainly used for optical character recognition, face detection, smile detection, emotion detection, object recognition, etc [10]. Optimizers are one of the major attributes of neural networks, either shallow or deep. Training of such deep neural networks is a challenging, non-convex and high-dimensional optimization problem. Accurate classification or prediction in a neural network is overly reliant on the optimum tuning of weights in the deep layers of the network [13]. Weights describe the strength of the connections between the nodes in a neural network and also signifies the influence of the previous node on the current node. The generalization capability of models largely depends on a set of optimized weight vectors. Therefore, the selection of an appropriate optimizer is important during the network architectural design.

Gradient Descent Algorithms (GDA) have proven to be quite an efficient optimization method and has been effectively used to optimize gradient-based learning systems such as neural networks [6, 14]. There are three main variants of GDA: the Batch Gradient Descent, Stochastic Gradient Descent (SGD) and Mini-Batch Gradient Descent [5]. The utilization of the three variants of GDA depends on the size of the dataset used to compute the gradient of the objective function [5]. SGD proves to be quite popular compared to the other variants as it is quite easy to implement, fast and efficient for problems that have many training samples [16].

A drawback of SGD is that it does not address the poor

978-0-7381-3366-9/21/$31.00 ©2021 IEEE

convergence or low classification accuracy due to inconsistencies in the dataset [16]. Inconsistencies in the dataset are caused by training data samples that cause a net error (or cost) to increase during gradient computation [16]. Inconsistencies are introduced by the algorithm's biased selection of random data instances which causes an increase in the net error during gradient computation for the current network training iteration. These inconsistencies in the dataset lead to high variance resulting in slow convergence with high fluctuations, however, they also help in jumping out of local optimum solutions. Employing SGD algorithms can lead to the problem of non-convergence due to the oscillation of the learning rate in the later training stages [4].

Furthermore, one of the most successful techniques used in helping overcome SGD's weakness in finding global minima is the usage of Momentum [14]. Momentum is a method that helps guide SGD in the right direction and reduces oscillation [3]. It adds a fraction of the updated vector of the previous time stamps to the current updated vector. Some other techniques such as AdaGrad adapts the learning rate according to the value of the existing parameters by controlling its learning rate and using larger learning rates for parameters that appear rarely and smaller learning rates for the parameters that appear quite often [13, 14, 11]. This algorithm is well-suited for dealing with sparse data but unfortunately, its performance deteriorates when the loss function is nonconvex and gradients are dense due to rapid decay of the learning rate, eventually taking it to 0 [11, 17]. AdaDelta resolves this issue of diminishing the learning rate by getting the learning rate for present gradients based on its history rather than taking the entire history of the learning rate like AdaGrad [14, 11]. A similar algorithm, RMSprop solves the issue of AdaGrad's diminishing learning rate and calculates its learning rate with an exponential average of squared gradients [13, 11].

Adaptive Moment Estimation (Adam) is one of the most commonly used optimizers in neural network models, especially for large datasets and high dimensional parameters [11]. It has been derived from two algorithms: RMSprop and AdaGrad [6]. Adam can adapt its learning rate according to its parameters rather than storing exponentially decaying average of past squared gradients like AdaDelta and RMSprop, Adam, similar to momentum, also keeps an exponentially decaying average of past gradients [14, 11].

The focus of this paper is the application of Guided Stochastic Gradient Descent (GSGD) based optimer with CNNs. The GSGD optimer identifies the inconsistencies in the dataset and uses it strategically to produce a better convergence rate [16]. GSGD realizes the significance of inconsistencies on gradient computation and aims to perform gradient computation and weight update only on consistent data. The algorithm hides the inconsistencies present in the training dataset "for a while" considering that it may become consistent over the next few iterations. Noise is never removed permanently but bypassed until they become consistent enough to be included in the gradient computation. GSGD is compatible with all the popular variations of SGD discussed above, however, it has been applied to logistic regression only.

In this paper, we will apply GSGD to a deep learning network, CNN, and address the improvement of the convergence rate and accuracy on benchmark image datasets. We also provide the convergence analysis of the GSGD algorithm.

II. Guided Stochastic Gradient Descent for CNN

The original GSGD algorithm was developed to collect both the consistent and inconsistent datasets in shallow machine learning algorithms such as logistic regression [16]. Inconsistent data instances are simply the data instances within the neighborhood of instance j; which individually performs better, while the average error value $\overline{E}_t$ performs worse than the average error of the previous iteration $\overline{E}_{t-1}$, and vice-versa.

The GSGD for CNN (CNN-GSGD) exclusively identifies and collects the consistent data instances only during the training iterations. It performs the gradient computation on these consistent data and updates the weights of the entire network. These data instances are pushed back into the training data samples after the weights have been updated in anticipation that the remaining inconsistent data instances will gradually become consistent as the model ages. It was reported in [16] that this strategic maneuver of the weights results in improved classification accuracy by approximately 3 % for the benchmark datasets from UCI Library .

The original GSGD uses the entire training dataset as Verification Data to compute the error of the training data instance at every iteration, which is not applicable for deep learning as this would have very high overhead cost of computing, therefore, this research proposes the use of only a portion of the training dataset as verification data. This effectively reduces the performance overhead cost and enables the algorithm to execute more expeditiously. The proposed idea is to hide a random portion of the training dataset at the beginning of every epoch and use it as verification data. The portion is returned to the training data pool at the end of the epoch, and a new portion is selected randomly.

As described earlier, inconsistent data instances are simply data points that lie in the neighborhood (proximity) of a data instance within the neighborhood allocation (ρ) window; and has an average error value $\overline{E}_t$ that is greater than the average error of the previous data instance $\overline{E}_{t-1}$. This has been further detailed in the pseudo-code of the algorithm (see Algorithm 1).

The algorithm initiates execution with gradient computation and weight update with the learning rate η in a typical manner for the first ρ iterations. Average error for the data batch at each iteration ($\overline{E}_t$) is computed, and compared with the average error of the previous data batches ($\overline{E}_{t-1}$, $\overline{E}_{t-2}$,... $\overline{E}_{t-\rho}$). These preceding data instances are re-run

Algorithm 1 Pseudocode for Guided SGD for CNN

while $t = 1 : T$ **do**
  Computer gradient $v(d_i)$
  $W_t = W_t - \eta v(d_i)$
  $\overline{E}_t = approximateAvgError()$
  $\psi = collectConsistentBatches(d_i, d_{i-1}, ...d_{i-\rho})$
  if neighborhood allocation (ρ) reached **then**
    ψ = getConsistentBatches(ψ,$\overline{E}_t$)
    for $i = 1 :\| \psi \|$ **do**
      $W_t = W_t - nv(\psi_i)$
    end for
  end if
end while

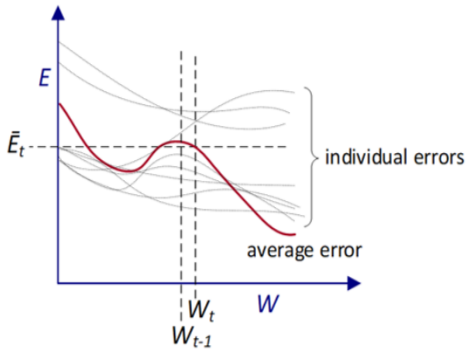

Fig. 1: Behavior of Individual Instances

on the current weights (W_t) to evaluate the consistency of the current data instance with regards to its preceding data instances, and to eventually realize the true behavior of the current data instance (see Fig. 1). At the end of ρ iterations, all data batches performing consistently on $W_{t-\rho}$ to W_t is kept in the consistent image datastore ψ. Finally, the entire weight vectors are updated with the consistent data batches in ψ. The algorithm proposes to process all the data instances (batches) regularly for ρ instances, then redo this for the consistent data instances identified.

Its flowchart is given in Fig. 2 where the algorithm starts with a random selection of a data instance whose gradient is computed to update the weight vector. After ρ iterations, the weight vector is further refined with consistent neighboring data instances. It is important to note that the value of ρ must be chosen wisely, as a very large value would result in the algorithm executing in its original form with typical gradient computation and weight update, and GSGD algorithm having very little to no effect in improving its efficiency. For this paper, the value of ρ was selected with Bayesian Hyperparameter Optimization.

GSGD works with the same gradient computation formula set for the network, and thus, can be easily incorporated with any variant of the commonly used optimization techniques.

The typical Mini-batch gradient descent algorithm in CNN uses B data samples at time t to compute the gradient ∇E_t of the objective function at W_{t-1} as shown in Eq. (1)[6].

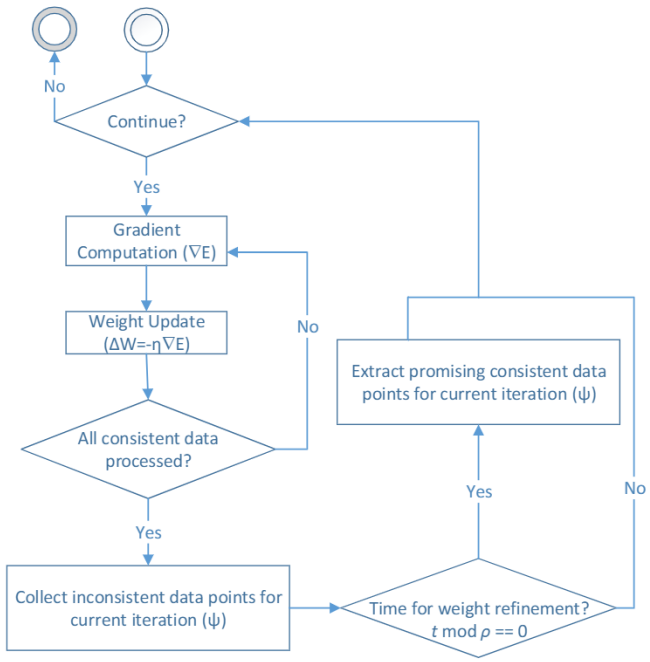

Fig. 2: Flowchart of GSGD

$$\nabla E_t \leftarrow \nabla E_t(W_{t-1}) = \frac{1}{|B|}\Sigma \nabla E_i(W_{t-1}) \tag{1}$$

Given the learning rate $\eta_t > 0$, the current mini-batch computes the independent value W_t as shown in Eq (2) below:

$$W_t \leftarrow W_{t-1} - \eta_t \nabla E_t \tag{2}$$

The cost of computing gradient with mini-batches at each iteration is $O(B)$. It is important to note that when the mini-batch size is 1, the algorithm is the typical SGD, whereas, with the batch size equal to the size of training data, the algorithm is gradient descent. The GSGD algorithm, incorporated with CNN, has the same cost of gradient computation, and continues to execute SGD, performing archetypal gradient computation and weight update, while progressively analyzing and collecting consistent samples for ρ training iterations and places them in ψ as discussed earlier [16]. The ψ dataset is sorted, and at most $\rho/2$ samples are applied for weight refinement and update.

GSGD is now capable of handling big data. As mentioned earlier, the original GSGD was used only with logistic regression and small benchmark datasets. This paper focuses mainly on CNN-GSGD, i.e., incorporating GSGD in CNN as an optimizer and comparing it with other variants of SGD. The code is available at https://github.com/anuraganands.

III. Convergence Analysis for GSGD

Basically, the SGD algorithm is an optimization problem that can be represented as:

$$\hat{W} = \arg\min_{W} \frac{1}{N} \sum_{i=1}^{N} E(d_n, W) \tag{3}$$

where $E(d_n, W)$ is an error or cost function that takes weight vector W and N mini-batches of training examples of size m from $d_1 \dots d_N$. The gradient computation can be shown as $\eta\nu_t(d_1), \eta\nu_t(d_2), \dots, \eta\nu_t(d_i), \dots, \eta\nu_t(d_N)$ where $\nu_t(d_i)$ is a partial gradient function for a mini-batch d_i at iteration t.

GSGD re-computes the gradients for the $\| \psi \| \leq \rho$ consistent mini-batches after ρ iterations, that can be represented as:

$W_{t+1} = W_t - \eta\nu_t(d_{i+1}) - \eta\nu_t(d_{i+2}) - \dots - \eta\nu_t(d_{i+\rho}) = W_t - \eta\sum_{j=1}^{\rho}\nu_t(d_{i+j})$

$= W_t - \eta\rho\bar{\nu_t}$ where $\mathbb{E}(\bar{\nu_t}) = \nabla E(W_t)$. $\nabla E(W_t)$ shows the true gradient for W_t.

To prove the convergence rate for a simplistic case, the error function E is considered to be convex and Lipschitz β-smooth [8, 2] as shown in Eq. 4:

Proof.

$$E(W_{t+1}) \leq E(W_t) + \langle \nabla E(W_t), W_{t+1} - W_t \rangle + \frac{\beta}{2} \| W_{t+1} - W_t \|^2 \quad (4)$$

$$= E(W_+) + \langle \nabla E(W_t), W_t - W_+ \rangle - \eta\rho \langle \nabla E(W_t), (\bar{\nu_t}) \rangle + \frac{\beta^2\eta^2\rho^2}{2} \| \bar{\nu_t} \|^2$$

where

$$W_+ = \arg\min_W E(W)$$

This can be resolved to:

$$\mathbb{E}(E(W_{t+1})) \leq \mathbb{E}(E(W_+)) + \langle \nabla E(W_t), W_t - W_+ \rangle - \eta\rho^2 \| \nabla E(W_t) \|^2 + \frac{\eta\rho}{2} \| \nabla E(W_t) \|^2 + Var(\bar{\nu_t})$$

where Var refers to variance. Here we assume $\beta \leq \frac{1}{\eta\rho}$ and $Var(\bar{\nu_t}) \leq \frac{\sigma^2}{\rho}, \forall t \geq 0$ [12, 1]. Since, $\| \nabla E(W_t) \|^2 = \mathbb{E}(E(\| \bar{\nu_t} \|^2)) - Var(\bar{\nu_t})$; the Cauchy Swartz inequality [2] for the summation of $t = 0 \dots T-1$ results in the following equation:

$$\sum_{t=1}^{T} E(W_t) \leq TE(W_+) + \frac{1}{2\eta\rho}(\| W_0 - W_+ \|^2 - \| W_T - W_+ \|^2) + T\eta\sigma^2$$

rearranging the variables would finally provide the convergence term for GSGD given in Eq. 5 below:

$$\mathbb{E}(E(W_t) - \min_W E(W)) \leq \frac{1}{2\eta\rho T} \left\| \arg\max_W E(W) - \arg\min_W E(W) \right\|^2 + \eta\sigma^2 \quad (5)$$

□

Therefore, the convergence of GSGD is of the order $O(\frac{1}{\rho T} + \sigma^2)$ which is slightly better than SGD's order of $O(\frac{1}{T} + \sigma^2)$. Nevertheless, asymptotically both orders are same.

Dataset-1: Digit	Dataset-2: MNIST
Input image data	Input data image
Conv2D(3,1*numF); stride=1; padding=1;	
ReLU (nonlinearity function)	
Pooling Layer: MaxPooling 2x2; stride=2	
Conv2D(3,2*numF); stride=1; padding=1;	
ReLU (nonlinearity function)	
Pooling Layer: MaxPooling 2x2; stride=2	
Conv2D(3,4*numF); stride=1; padding=1;	
ReLU (nonlinearity function)	
Pooling Layer: MaxPooling 2x2; stride=2	
Flattening (Input Layer of Neural Network)	
Optimizer	
Softmax Layer	

Fig. 3: CNN Architecture

IV. Experiment Setup

In this section, we evaluate our model by comparing it with other optimization models. We start off by introducing the datasets we used, experimental setup, and reporting the analysis of the experimental results.

A. Dataset

To evaluate the effectiveness of the CNN-GSGD, we have performed experiments using the three benchmark datasets: MNIST [9], USPS Digit [4] and CIFAR-10 [7].

MNIST dataset consists of greyscale images of handwritten single digits between 0 and 9. The size of the images are 28 x 28 pixels and comprises of 60000 training set data and 10000 test set data. USPS Digit Dataset has a total of 10000 greyscale images of handwritten digits out of which 7291 images are used as the training dataset and 2007 test set. The image size is 16 x 16 pixels. After performing the experiments on the MNIST and USPS Digit Datasets, we ran experiments on the CIFAR-10 dataset. CIFAR-10 dataset has 32 x 32 pixels color images containing 10 different classes of data. The total number of images in this dataset is 60000, divided into 50000 images of the training set and 10000 of the test set.

B. Network Architecture

The architecture of the CNN used in this research is described in Fig. 3. The feature extraction part comprises of standard convolution and pooling layers. However, the fully connected layers that are normally present in the classifier layer are replaced by the proposed optimizer. Therefore the network learns through different levels of abstraction of the images over the different layers in the network. The network uses softmax to produce final classification results. Matlab's in-built CNN libraries are used to develop the CNN-GSGD model.

C. Training Process

The CNN-GSGD model was trained using GSGD, SGD, ADAM and guided-ADAM (G-ADAM) optimizers on the datasets. A total of 30 successive runs were made for each

TABLE I: Bayesian parameter values for different datasets where SectionDepth indicates the depth of the network, η and ρ are the the learning rate and momentum respectively.

Dataset	SectionDepth	filterSize	η	ρ	Momentum	L2Regularization
Digit (GSGD)	2	3x3	0.00086	10	0.95327	0.00012631
Digit (SGD)	1	2x2	0.00081	-	0.96732	0.00081662
MNIST (GSGD)	1	2x2	0.00099	9	0.92358	0.00137410
MNIST (SGD)	1	2x2	0.00066	-	0.97960	0.00315110
CIFAR-10 (GSGD)	1	2x2	0.00093	8	0.89219	0.0060794
CIFAR-10 (SGD)	1	2x2	0.00025	-	0.9755	0.0000000002
Digit (ADAM)	1	3x3	0.00092	-	0.98133	0.00132240
Digit (G-ADAM)	1	2x2	0.00088	-	0.98952	0.0007137x1
MNIST (ADAM)	1	2x2	0.00099	-	0.98412	0.00004373
MNIST (G-ADAM)	1	3x3	0.00085	7	0.98976	0.00000032
CIFAR-10 (ADAM)	1	3x3	0.00095	-	0.98771	0.0098458
CIFAR-10 (G-ADAM)	1	3x3	0.00098	10	0.98842	0.0000031299

dataset with each combination of optimizers mentioned above. The combinations can be seen in Table. I. To obtain the best hyper-parameters for CNN with GSGD, SGD, G-ADAM and ADAM, the Bayesian optimization algorithm was used. Table. I highlights all the hyper-parameters obtained from training the model on the datasets with the other variants of SGD. All experiments were conducted on a Windows-based desktop PC with an i7-9th Gen processor, 16 GB of RAM and RTX 2060 GPU.

V. Results and Discussion

The networks have been trained on three datasets with GSGD, SGD, G-ADAM and ADAM optimizers. The main objective was to determine how the guided variant performs with the deep learning algorithm. Identical scenarios were used for all the datasets so that a better comparison could be made. A total of 30 runs were executed per dataset for the different variations of SGD and the results were recorded. Fig. 4, Fig. 5 and Fig. 6 gives a comparative visualization between the two variations of the SGD and ADAM used in three different dataset and the results it produces.

The maximum, minimum and mean test accuracy results for each dataset and optimizer are shown in Table II. The bold font indicates better solution. For the MNIST dataset, the results of the SGD are nearer to the GSGD, however, evaluating the overall graphs concludes that the GSGD outperforms SGD even at such high accuracy results yielding a maximum accuracy of 99.85% for GSGD and 99.60% for SGD with a difference of 0.25%. The results for ADAM and G-ADAM are quite close. Even though ADAM yields high accuracy results of 99.85%, G-ADAM gives 99.90% of accuracy and outperforms it by 0.05%.

For the USPS Digit Dataset, GSGD produced a maximum accuracy of 98.85% while SGD produced 98.74%. On the other hand, G-ADAM yielded a maximum accuracy of 99.91% while ADAM produced a maximum accuracy of 98.94%. Evidently, the results show that the guided version of the optimizers performs better. GSGD outperformed SGD by 0.11% while G-ADAM outperformed vanilla ADAM by 0.97%. Even the mean values for each optimizer shows that the guided version of optimizers performed quite well. The figures Fig. 4 and Fig. 5 shows that the guided variants are either quite nearer or above the vanilla variants.

Furthermore, we trained and tested the model with the CIFAR-10 dataset. Despite the overall result being a letdown, the Guided variant still managed to outperform the vanilla variants. GSGD had the maximum test accuracy of 69.66% while SGD had 68.8%. G-ADAM had 69.36% while ADAM had 67.09%. On average, the guided variants for SGD and ADAM were better than the vanillar variants by about 2-3%.

GSGD outperforms its canonical counterpart in limited time constraints. The algorithm acts as a guide to the SGD when dealing with inconsistent datasets, showing notable improvement in classification accuracies of CNN models, and is easily incorporable with other variants of SGD such as Adam, Momentum and RMSprop [16]. The neighborhood allocation value (ρ) obtained from the Bayesian optimizing algorithm ensured efficiency of the algorithm with timely weight refinement, and avoids the consistent samples from becoming inconsistent by being left out for too long before it gets included in the gradient computation. The improvement in classification accuracy comes at a small cost of performance for average error calculation with verification data, while the time complexity remains the same for both the guided and canonical variant of SGD. To keep the overhead performance cost low, only a portion of the training dataset was utilized as verification data to calculate the average error of each batch during training.

A better depiction of the average error would be obtained by utilizing the entire training dataset, however, its overhead cost of performance can be resolved by parallelizing GSGD's average error calculation on multiple processors or executing it on multiple threads.

VI. Conclusion

Guided Stochastic Gradient Descent has proven to be quite useful in terms of improving the accuracy of the CNN for the benchmark datasets: USPS Digit, CIFAR-10 and MNIST datasets. Our experimental results show that GSGD has performed better when incorporated with CNN. Its convergence rate is slightly better than SGD with the order of $O(\frac{1}{\rho T} + \sigma^2)$. ADAM with the guided approach has also shown improvement

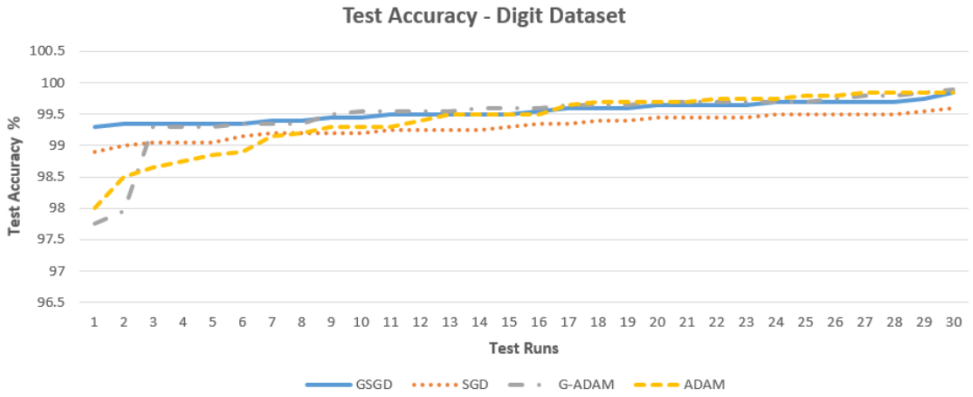

Fig. 4: Accuracy Results of the Digit Dataset with GSGD, SGD, G-ADAM and ADAM

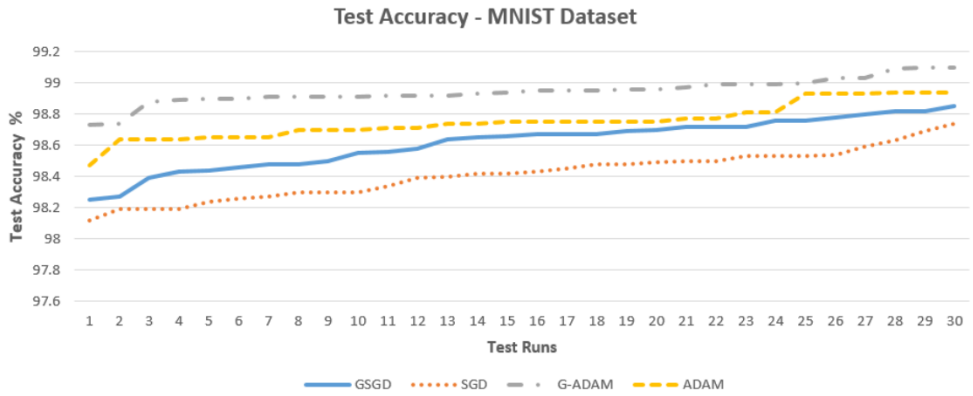

Fig. 5: Accuracy Results of the MNIST Dataset with GSGD, SGD, G-ADAM and ADAM

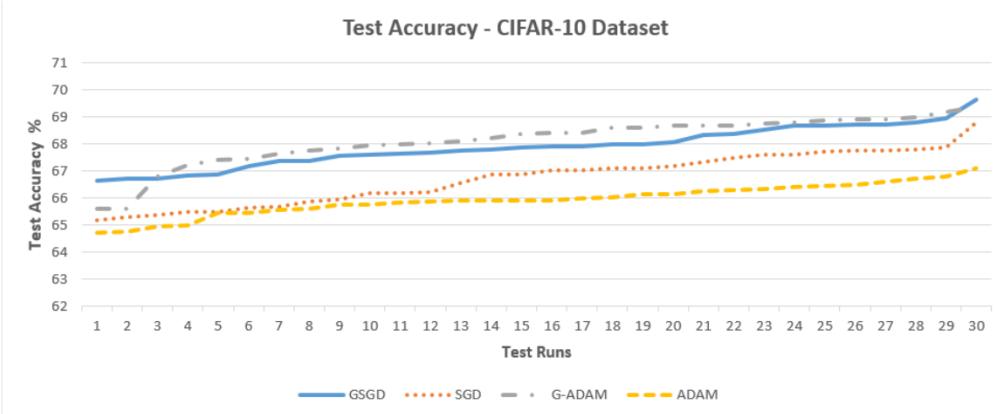

Fig. 6: Accuracy Results of the CIFAR-10 Dataset with GSGD, SGD, G-ADAM and ADAM

TABLE II: Maximum, Minimum and Mean Test Accuracy for each Dataset and Optimizer

Dataset	Maximum	Minimum	Mean
GSGD-DIGIT	**99.85**	**99.30**	**99.55**
SGD – DIGIT	99.60	98.90	99.31
G-ADAM – DIGIT	**99.90**	97.75	**99.48**
ADAM – DIGIT	99.85	**98.00**	99.40
GSGD - MNIST	**98.85**	**98.25**	**98.62**
SGD – MNIST	98.74	98.12	98.41
G-ADAM – MNIST	99.91	**98.73**	**98.95**
ADAM – MNIST	**98.94**	98.47	98.75
GSGD - CIFAR-10	**69.66**	**66.65**	**67.91**
SGD - CIFAR-10	68.80	65.18	66.74
G-ADAM - CIFAR-10	**69.36**	**65.6**	**68.13**
ADAM - CIFAR-10	67.09	64.71	65.94

in the accuracy of CNN at an average of around 1% for MNIST and USPS datasets, and around 3% for CIFAR-10 dataset. These experiments prove the usability of GSGD with large datasets in deep learning networks. The future work would be to apply this guided approach on parallel environment to improve its performance further for deep learning algorithms such as Convolutional Neural Networks and Recurrent Neural Network. Parallel GSGD [15] with shallow network already provides promising results.

REFERENCES

[1] Dan Alistarh, Christopher De Sa, and Nikola Konstantinov. "The Convergence of Stochastic Gradient Descent in Asynchronous Shared Memory". In: *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. PODC '18. event-place: Egham, United Kingdom. New York, NY, USA: ACM, 2018, pp. 169–178. ISBN: 978-1-4503-5795-1. DOI: 10.1145/3212734.3212763. URL: http://doi.acm.org/10.1145/3212734.3212763 (visited on 12/21/2019).

[2] Sébastien Bubeck. "Convex Optimization: Algorithms and Complexity". In: *arXiv:1405.4980 [cs, math, stat]* (Nov. 2015). arXiv: 1405.4980. URL: http://arxiv.org/abs/1405.4980 (visited on 12/21/2019).

[3] Imen Chakroun, Tom Haber, and Thomas J Ashby. "SW-SGD: the sliding window stochastic gradient descent algorithm". In: *Procedia Computer Science* 108 (2017), pp. 2318–2322.

[4] CC Chang and CJ Lin. *LIBSVM data: classification (Multi Class)*. 2006.

[5] EM Dogo et al. "A comparative analysis of gradient descent-based optimization algorithms on convolutional neural networks". In: *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*. IEEE. 2018, pp. 92–99.

[6] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[7] Alex Krizhevsky. *The CIFAR10-dataset*. https://www.cs.toronto.edu/~kriz/cifar.html. [Online; accessed 16-Aug-2016]. 2009. (Visited on 08/16/2016).

[8] Simon Lacoste-Julien, Mark Schmidt, and Francis Bach. "A simpler approach to obtaining an O(1/t) convergence rate for the projected stochastic subgradient method". In: *arXiv:1212.2002 [cs, math, stat]* (Dec. 2012). arXiv: 1212.2002. URL: http://arxiv.org/abs/1212.2002 (visited on 07/11/2020).

[9] Yann LeCun, Corinna Cortes, and Christopher JC Burges. "The MNIST database of handwritten digits, 1998". In: *http://yann.lecun.com/exdb/mnist/* 10.34 (1998), p. 14.

[10] Keiron O'Shea and Ryan Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015).

[11] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151.

[12] Raghu Meka. *CS289ML: Notes on convergence of gradient descent*. github. Dec. 2019. URL: https://raghumeka.github.io/CS289ML/gdnotes.pdf.

[13] SV Reddy, K Thammi Reddy, and V ValliKumari. "Optimization of Deep Learning Using Various Optimizers, Loss Functions and Dropout". In: *Int. J. Recent Technol. Eng* 7 (2018), pp. 448–455.

[14] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[15] Anuraganand Sharma. "Guided parallelized stochastic gradient descent for delay compensation". In: *Applied Soft Computing* 102 (2021), p. 107084.

[16] Anuraganand Sharma. "Guided stochastic gradient descent algorithm for inconsistent datasets". In: *Applied Soft Computing* 73 (2018), pp. 1068–1080.

[17] A Wibowo, PW Wiryawan, and NI Nuqoyati. "Optimization of neural network for cancer microRNA biomarkers classification". In: *Journal of Physics: Conference Series*. Vol. 1217. IOP Publishing. 2019, p. 012124.