Conjugate gradient descent with Barzilia-Borwein step size for Non-convex optimization

Furqan Yaqub Khan

Roll No:- 2011CS05

Department of Computer Science and Engineering

Indian Institute of Technology, Patna

Furqan_2011CS05@iitp.ac.in

Supervisor

Dr. Abyayananda Maiti

Contents

1. **Introduction**
2. **Related Works**

    2.1 2nd Order Optimization Algorithms
    2.2 Variants of Conjugate Gradient Descent

3. **Motivation**
4. **Proposed Algorithms**

    4.1 KAF algorithm

    4.2 KAMF algorithm

5. **Experimentation**

    5.1 CIFAR10 Dataset
    5.2 FMNIST Dataset
    5.3 MNIST Dataset

6. **Comparison and Results**
7. **Future Analysis**
8. **References**

Abstract

Introduction

Neural networks mainly revolve around a minimization of a problem. These problems themselves are divided into two types convex and non-convex. The convex problems have only one local minimum which is itself known as global minimum. It consists only of convex objectives while as Non-convex problems does also have one or more non-convex objectives and also have one or any local minimums[1]. In recent past the advance in this sector are mainly due to powerful minimization algorithms. The available algorithms for minimization problems are classified into two types viz 1^{st} order and 2^{nd} order. The 1st order algorithms are less efficient as compared to 2^{nd} order algorithms but on the other hand 2^{nd} order algorithms are computationally expensive and require calculation of hessian for every iteration. The main aim of my project is to remove the hessian calculation from one of the 2^{nd} order algorithm named conjugate gradient descent for which i proposed two new CGD based algorithms namely KAF and KAMF.

Some of the prominent 2^{nd} order optimization algorithms include Newton method, Quasi-newton method, Levenberg-Marquardt, Hessian-free and Gauss-Newton method with many variants of each but for solving non-convex problems there are certain drawbacks to each of these methods. Newton's method main drawbacks include that the convergence is not guaranteed, It shows quadratic convergence only if the search point close to the minimum solution. It requires high computation cost to compute hessian at each iteration and also requires to store large hessian matrix to compute the updates for each iteration[2]. Quasi-Newton requires descent amount of memory to store gradient and update direction information from previous iteration[3]. Gauss-Newton struggle with indefinite Hessian and may halt the training without making significant training progress[4].

The choice of parameters in Levenberg-Marquardt for regularization is heuristic which is hard to get right that is the initial estimate or seed shall be provided closer to the solution other wise the algorithm struggles to maintain a proper path and diverges[5]. Hessian-Free requires to compute the approximate Hessian with finite differences of gradient evaluation which in turn requires conjugate gradient to compute the optimization[6]. Overall all second order methods require more computational power and space for storage of intermediatory results as compared to first order but are more accurate and solves the problems in lesser epochs. Some of them also deviate from path towards the solution, show lesser stability and as well loop indefinitely.

Conjugate gradient method utilizes direction to replace costly inverse hessian computation and also requires line-search approach to approximate step size therefore only suitable for batch learning[7]. The famous conjugate gradient descent variants are Hestenes-Stifiel, Fletcher-Revees, Polak-Ribiere, and Hagar-Zangar. Hestenes-Stifiel CGD is not able to solve unconstrained non-linear optimization problems, Fletcher-Revees can not solve all unconstrained non-linear optimization problems, Polak-Ribiere iterates indefinitely for non-linear functions and Hagar-Zangar does not show stability. In order to overcome all these drawbacks I proposed two new variants viz KAF CGD and KAMF CGD. KAG CGD uses Barzilai-Borwein type step size. The step size is chosen such that residual of secant equation is minimized and then the updation is done using my given updation rule. In the KAMF CGD a new step size is introduced based on difference between PRP and FR step size.

RELATED WORK

Second order optimization is the advances of first order optimization in neural networks. It provides an additional curvature information of an objective function that adaptively estimate step length of optimization trajectory. It reduces training iterations and achieves faster convergence. Although requires greater computational power than first order optimization techniques but with day by day increase in computing resources it becomes beneficial[8]**.** For solving non-convex optimization problems first order optimization algorithms come with well known deficiencies such as relatively slow convergence sensitivity to the settings of hyperparameters, Stagnation at high training errors, Difficulty in escaping asymmetric valleys, flat regions and saddle points. While on the other hand second order optimization algorithms overcome all of these deficiencies to great extent[9]. Lets analyse some second order methods. Newtons method is a numerical method for solving an equation. It is based on geometry of curve, using the tangent lines to a curve as linear approximator. Thus reducing the approximated distance iteratively and coming closer to solution. It converges rapidly but does not always converge. It also requires function to be differentiable as it is necessary for function to have tangent line. It converges slowly for multiple roots[10]. Any method that replaces exact Jacobian with an approximation is a quasi-newton method. Quasi-newton methods used to either find zeroes or local maxima or minima of function. They can be used if Jacobian or Hessian is unavailable or is too expensive to compute at every iteration. To reduce computational cost quasi-newton method uses BFGS for inverse Hessian computation and also needs descent memory to store gradient and update direction information from previous iteration[3]. It not only shows slower convergence in terms of steps but also lacks precision in inverse hessian computation[11]. The Gauss–Newton algorithm is used to solve non linear least square problems. It is a modification of Newton Method for finding a minimum of a function. Unlike Newton's method, the Gauss–Newton algorithm can only be used to minimize a sum of squared function values, but it has the advantage that second derivatives, which can be challenging to compute, are not required. It shows slower convergence in terms of steps and often gets lost[12]. Levenberg-Marquardt also known as damped least squares method is used to solve non-linear least squares problems. These problems arise especially in least square curve fitting. However LMA finds only local minimum which is not global minimum. LMA interpolates between Gauss-Newton and gradient descent methods. For multiple minima , the algorithm converges to the global minimum only if the initial guess is already somewhat close to the final solution. It needs the heuristic regularization parameter or seed which is heuristic in nature[5]. Hessian Free method solves the linear system using conjugate gradient . CG is an iterative method where each step involves a matrix-vector multiplication. When the number of parameters is large, it is not possible to explicitly compute hessian, but the matrix-vector product can still be computed. From where we can compute approximate Hessian with finite difference of gradient evaluation[6]. The above given table highlights the drawbacks of some famous second order algorithms most of which are not found in conjugate gradient descent. Apart from that they are computationally expensive and require more time for implementation. There is a choice to move towards first order optimization algorithms as they are computationally inexpensive and require less time to implement but they show poor results. Therefore conjugate gradient as an intermediatory algorithm between them is the best choice to work with.

I modified CGD and used Barzilai-Borwein step size along with some other changes. The inspiration for this step size has been taken from Quasi-Newton method. The approximation of Hessian is replaced by scalar step size $ŋ_t$ whose equation is

$$ŋ_{\boldsymbol{t}} = (||\boldsymbol{S_t}||_2^2 \,/\, \boldsymbol{S_t^T Y_t})$$

ŋt is not the actual solution of secant equation rather it is chosen such that the residual of secant equation is minimized and the next step of updation is taken according to below given equation

$$X_{t+1} = X_t - ŋ_t \nabla f(X_t) \quad [13]$$

MOTIVATION

The motivation behind my research is to develop an algorithms mainly for non-convex optimization functions commonly used in neural networks without getting stuck in saddle points, plateaus, asymmetric valleys where other algorithms face difficulty and as well to improve convergence rate and accuracy with being computationally cost effective.

PROPOSED ALGORITHMS

Algorithm1: *KAMF VARIANT OF CONJUGATE GRADIENT*

```
• $r_0 := b - Ax_0$
• if $r_0$ is sufficiently small return $x_0$ as result
• $p_0 := r_0$
• $k := 0$
• $repeat$
•     $\alpha_k := (r_k^T r_k / p_k^T A p_k)$
•     $x_{k+1} := x_k + \alpha_k p_k$
•     $r_{k+1} := r_k + \alpha_k A p_k$
•     if $r_{K+1}$ is sufficiently small, then exit loop
•     $\beta_k := (r_k^T r_{k+1} / r_k^T r_k)$
•     $p_{k+1} := r_{k+1} + \beta_k p_k$
•     $k := k + 1$
• $end\ repeat$
• $return\ x_{k+1}\ as\ result$
```

The equation $\beta_k := (r_k^T r_{k+1} / r_k^T r_k)$ is the only change here which is derived from which is absolute difference of Polak-Ribiere and Fletcher-Revees direction approximators.

Algorithm2: *KAF VARIANT OF CONJUGATE GRADIENT*

```
• $r_0 := b - Ax_0$
• if $r_0$ is sufficiently small return $x_0$ as result
• $p_0 := r_0$
• $k := 0$
• $repeat$
•     $\alpha_k := (r_k^T r_k / p_k^T A p_k)$
•     $x_{k+1} := x_k + \alpha_k p_k$
•     $r_{k+1} := r_k + \alpha_k A p_k$
•     if $r_{K+1}$ is sufficiently small, then exit loop
•     $\beta_k := (||X_k - X_{k-1}||_F)/(1/M(< X_k - X_{k-1}, g_k - g_{k-1} >) + \varepsilon( X_k - X_{k-1}))$
                                                            $0<\varepsilon < 1$
•     $p_{k+1} := (\beta_k p_k - (1 - \beta_k) r_{k+1}) / (1 - \beta_k)$
•     $k := k + 1$
• $end\ repeat$
• $return\ x_{k+1}\ as\ result$
```

The direction approximator $\boldsymbol{\beta_k}$ used here is based on Barzlai-Borwein step size with frobenious of difference of previous position and previous to previous position in the numerator and mean of dot product of absolute difference of previous position and gradient to that of previous to previous position and gradient with addition of episilon times absolute difference between previous position and previous to previous position where episilon ranges between zero and one.

$$\beta_k := (||X_k - X_{k-1}||_F)/(1/M(< X_k - X_{k-1}, g_k - g_{k-1} >) + \varepsilon(X_k - X_{k-1})) \qquad 0<\varepsilon < 1$$

The direction updation is given according to following equation

$$p_{k+1} := (\beta_k p_k - (1 - \beta_k) r_{k+1}) / (1 - \beta_k)$$

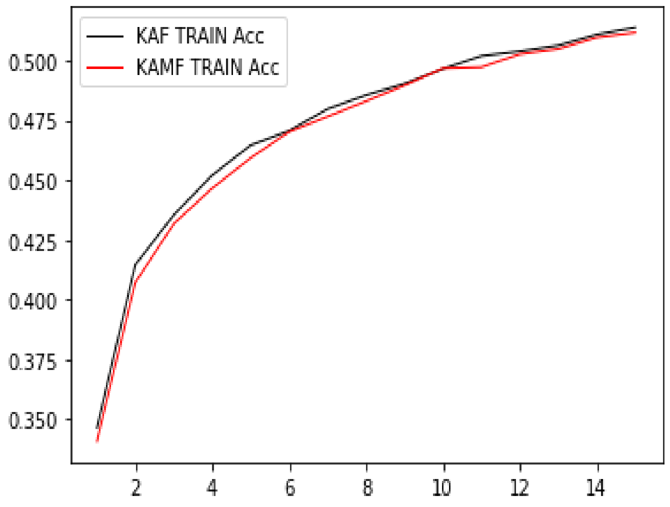
Where p_k previously used direction and r_{k+1} is solution approximator.

EXPERIMENTATION

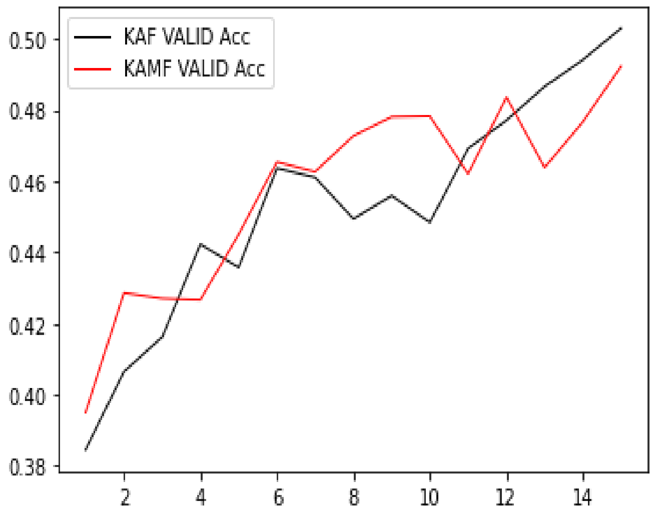
Experiment1: Testing the algorithms with CIFAR10 dataset

CIFAR10: This dataset consists of sixty thousand 32x32x3 pixel color images divided into 10 categories viz aeroplane:0, automobile:1, bird:2, cat:3, deer:4, dog:5, frog:6, horse:7, ship:8, truck:9 divided into 45000 training images, 5000 validation images and 10000 test images.

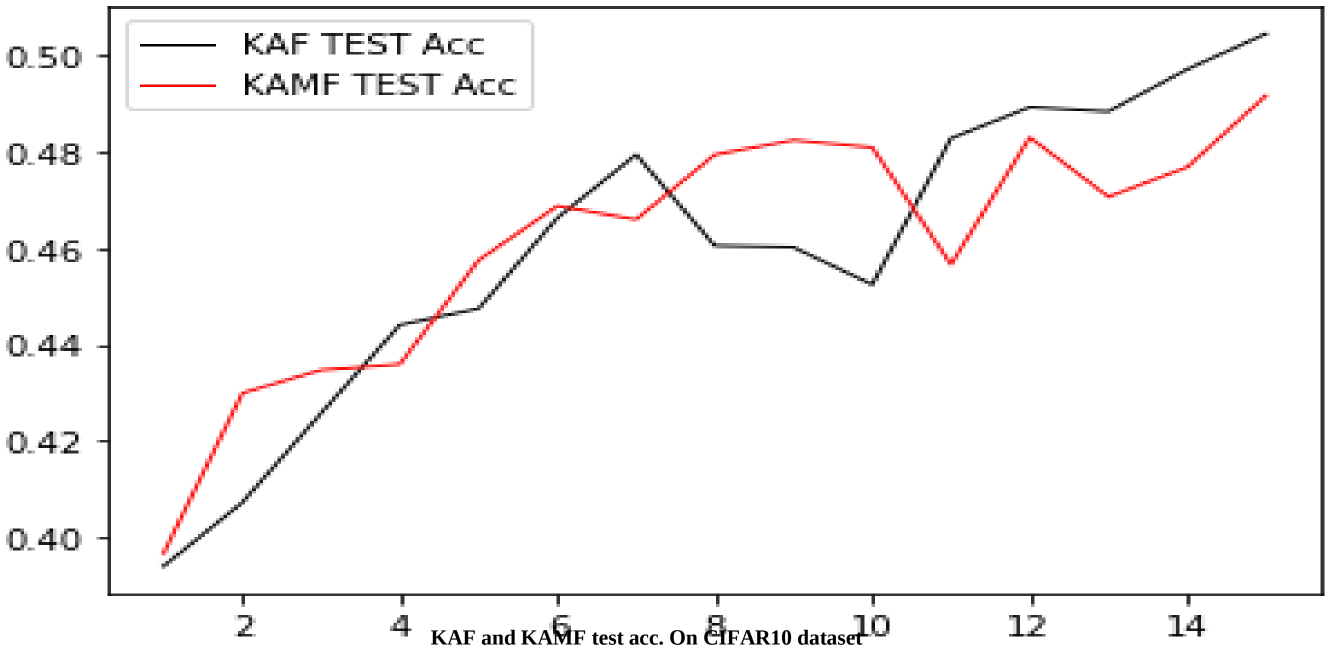
The testing has been done with multilayer perceptron model (MLP) in order to get the better intuition of algorithm. The 32x32x3 pixel color image has been fed to linear input layer of 250 neurons whose outputs are given to linear hidden layer of 100 neurons whose outputs are fed to linear layer of 10 neurons as inputs thus producing the final outputs for 10 classes. The loss function used is mean square error loss function(MSE) with activation function as ReLU and evaluation matrix as categorical cross entropy. Initial step size is taken as 0.1. The algorithms have been tested against four main variants of CGD viz Polak Ribiere, Fletcher Revees, Hagar Zangar and Hesteenes Stiefel for 15 epochs each.



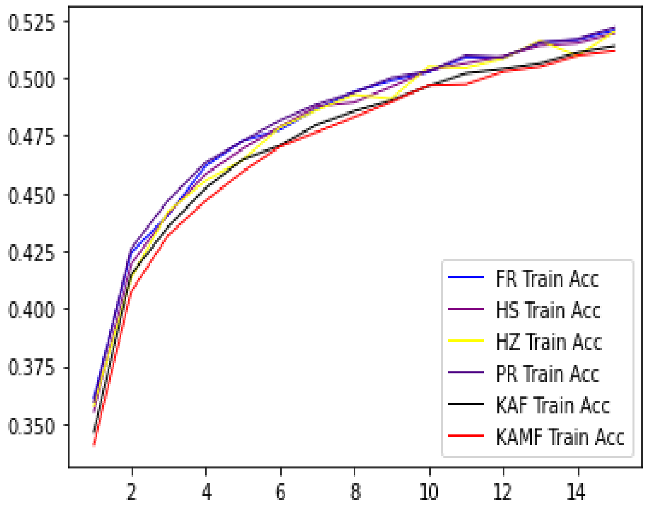
KAF and KAMF training acc. on CIFAR10 dataset



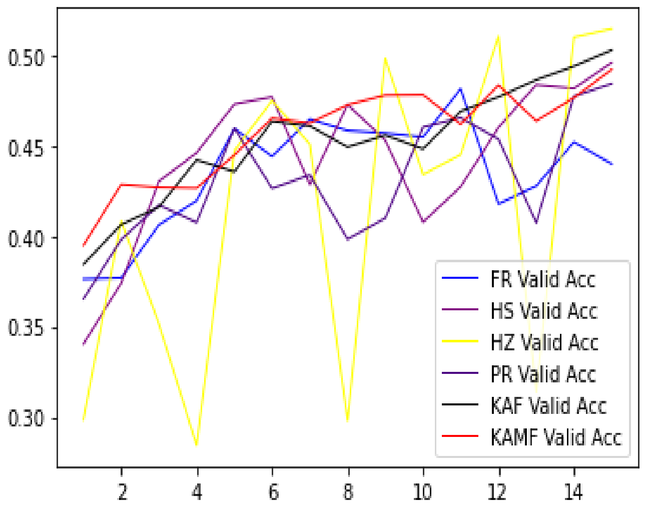
KAF and KAMF validation acc. on CIFAR10 dataset



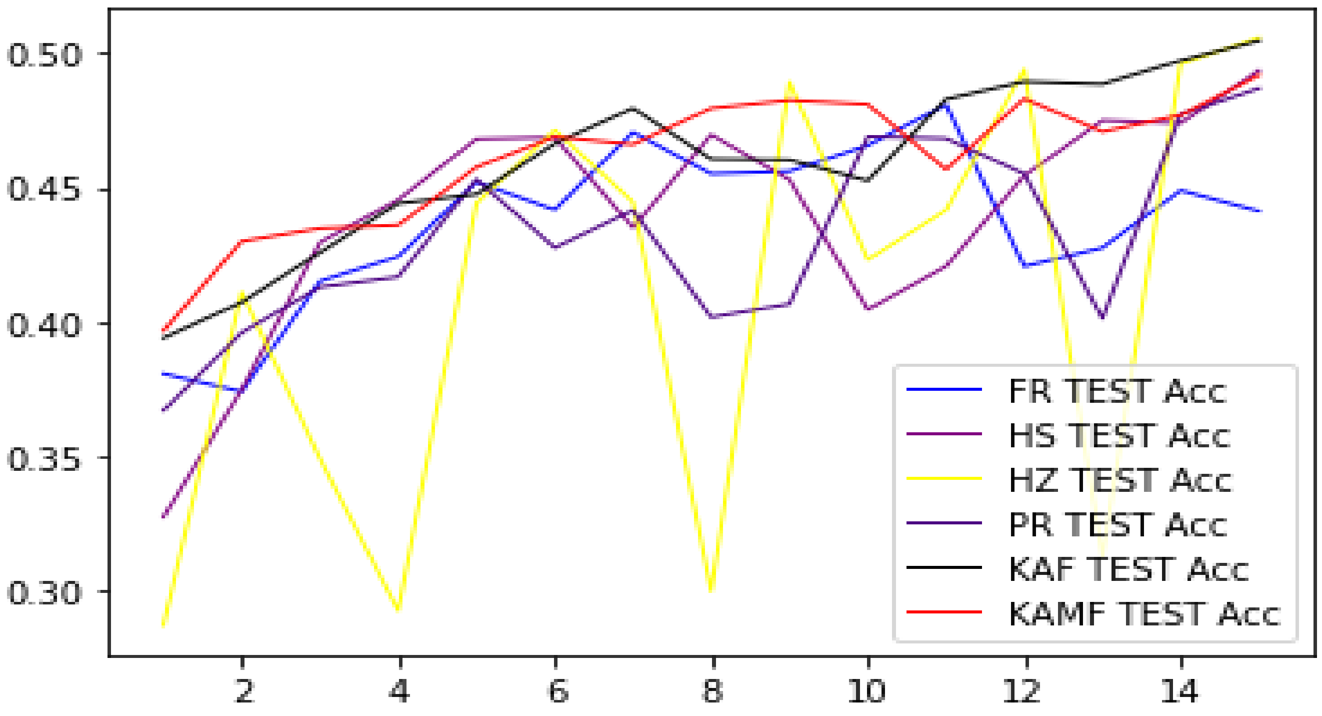
KAF and KAMF test acc. On CIFAR10 dataset



Training acc. of all variants on CIFAR10 dataset



Validation acc. of all variants on CIFAR10 dataset



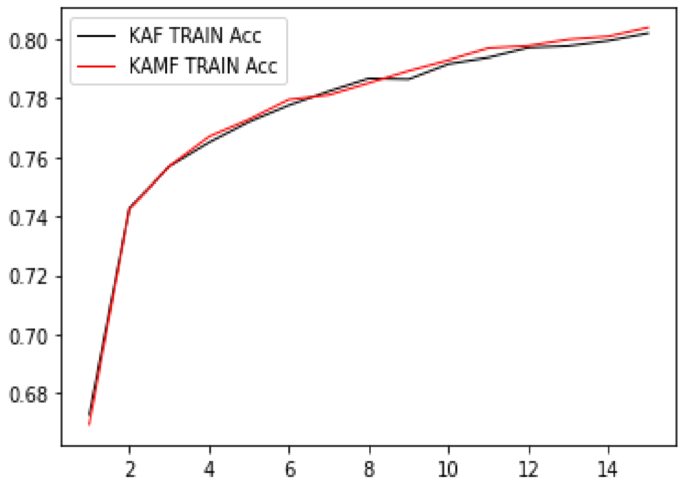
Test acc. of all variants on CIFAR10 dataset

Discussion: The combined training graph indicates that KAF and KAMF train curves are slightly lower below the training curves of other algorithms thus there are very lesser chances of overfitting as compared to others while looking on combined validation graph KAF and KAMF are performing average with greater stability. The test graphs indicate that KAF and KAMF show better test accuracy than any other algorithm out there with Hagar-Zangar as least stable followed by Hestenes-Stiefel with Fletcher-Revees dipping towards end. The Polak-Ribiere is performing average with less stability than KAF and KAMF.

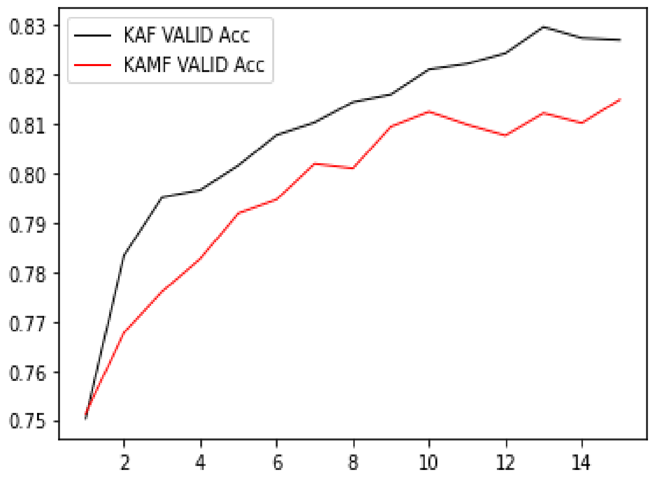
Experiment2: Testing the algorithms with Fashion-MNIST dataset

FMNIST: Fashion-MNIST is a dataset of Zalando's article images consisting of sixty thousand 28x28 greyscale images divided into 10 categories viz T-shirt:0, Trouser:1, Pullover:2, Dress:3, Coat:4, Sandal:5, Shirt:6, Sneaker:7, Bag:8, Ankle-boot:9 divided into 45000 training images, 5000 validation images and 10000 test images.

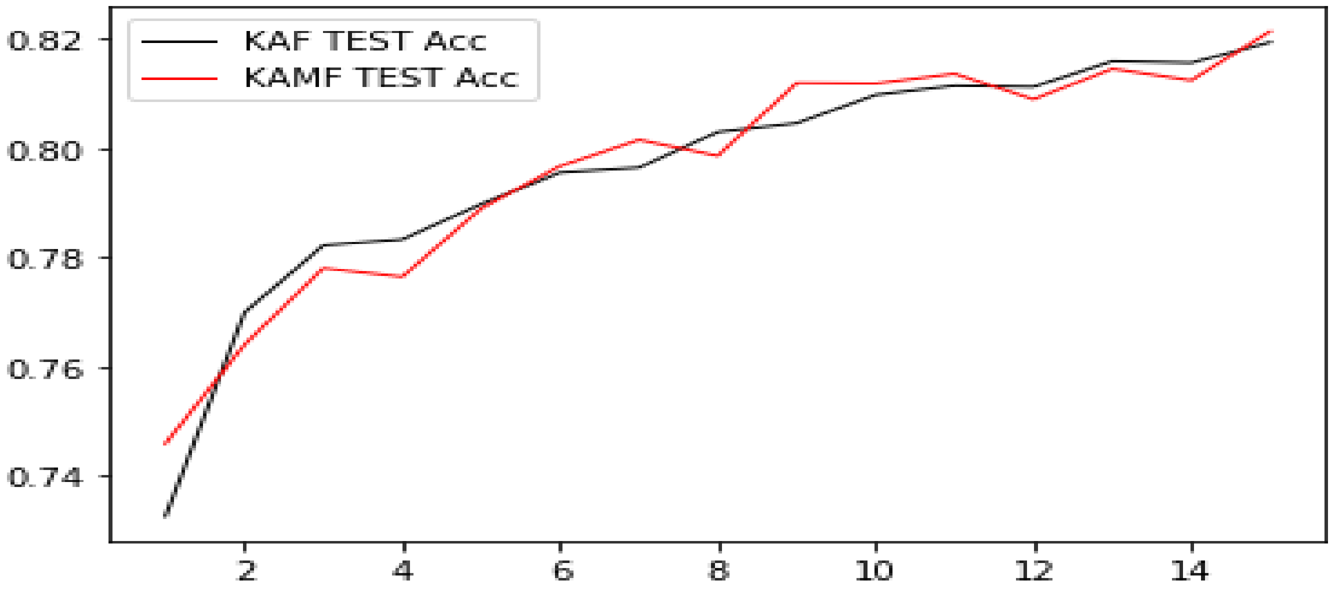
The testing has been done with multilayer perceptron model (MLP) in order to get the better intuition of algorithm. The 28X28 pixel greyscale image has been fed to linear input layer of 250 neurons whose outputs are given to linear hidden layer of 100 neurons whose outputs are fed to linear layer of 10 neurons as inputs thus producing the final outputs for 10 classes. The loss function used is mean square error loss function(MSE) with activation function as ReLU and evaluation matrix as categorical cross entropy. Initial step size is taken as 0.1. The algorithms have been tested against four main variants of CGD viz Polak Ribiere, Fletcher Revees, Hagar Zangar and Hesteenes Stiefel for 15 epochs each.



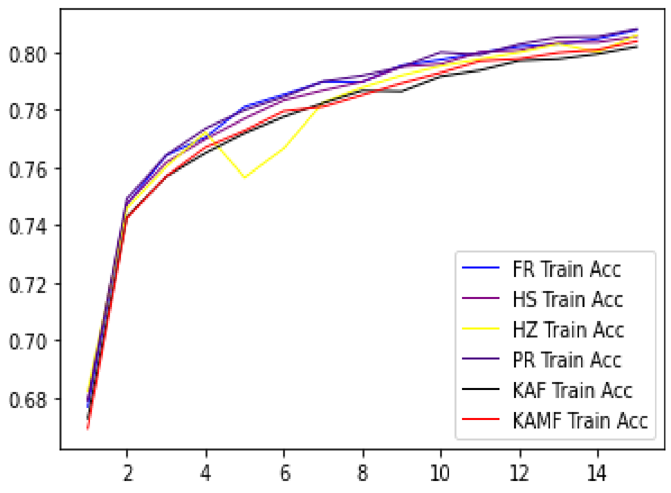
Kaf and KAMF training acc. on FMNIST dataset

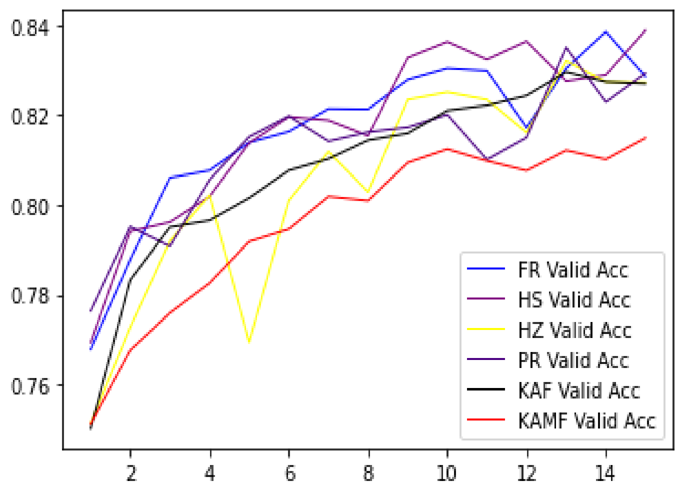


KAF and KAMF validation acc. on FMNIST dataset



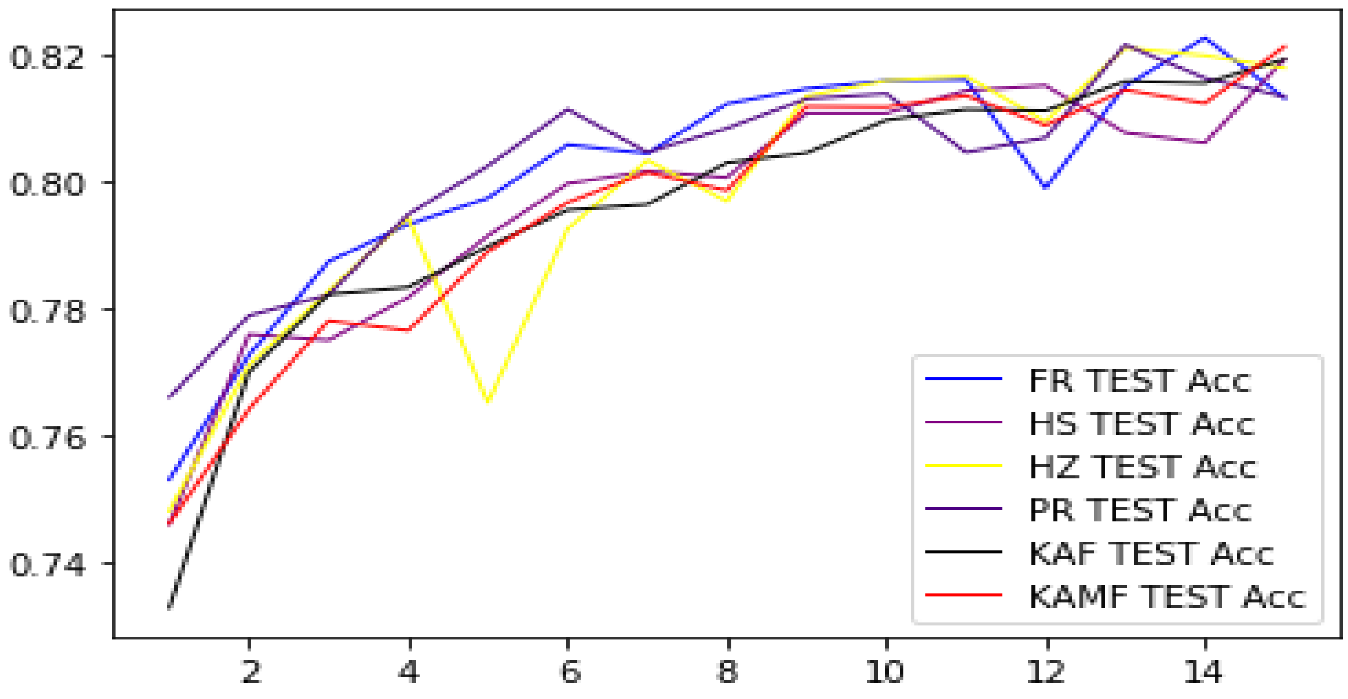
KAF and KAFM testing acc. on FMNIST dataset





Training acc. of all variants on FMNIST dataset

Validation acc. of all variants on FMNIST dataset



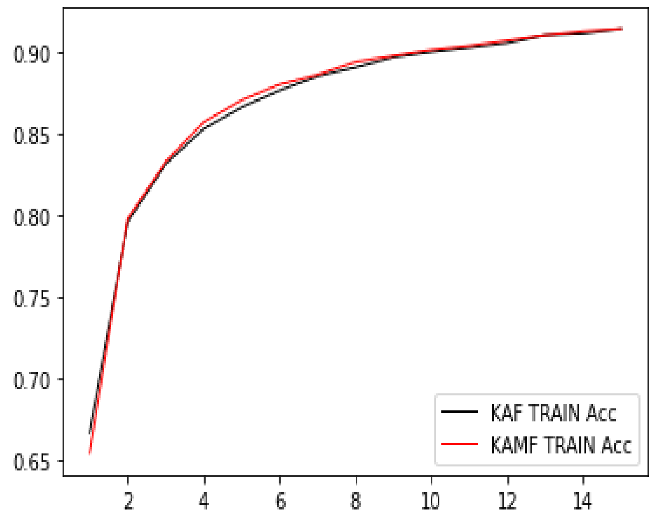
Testing acc. of all variants on FMNIST dataset

Discussion: The training of KAF and KAMF is very much stable and with almost no chance overfitting but in the validation phase the accuracy of KAMF is lesser may be due to the step size which is take as 0.1. During Testing the accuracies of KAF and KAMF are average but increasing towards the end which in turn leads conclusion that with few more epochs the proposed algorithms with overshoot the other.

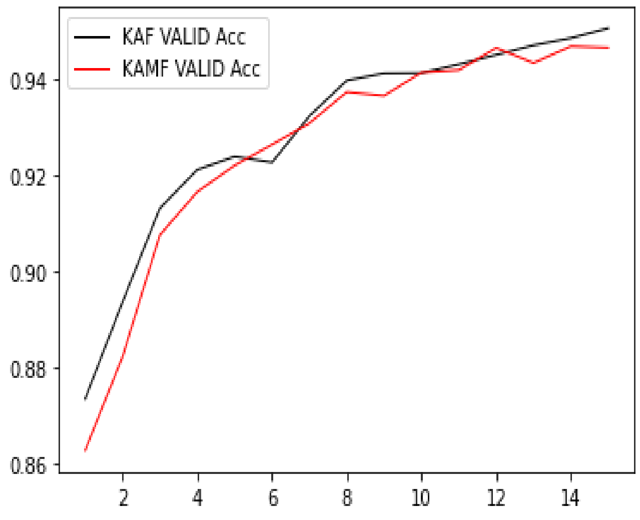
Experiment3: Testing the algorithms with Fashion-MNIST dataset

MNIST: Digit-MNIST or simply MNIST dataset is an acronym that stands for the Modified National Institute of Standards and Technology dataset. It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9 represented by 10 output neurons divided into 45000 training images, 5000 validation images and 10000 test images.

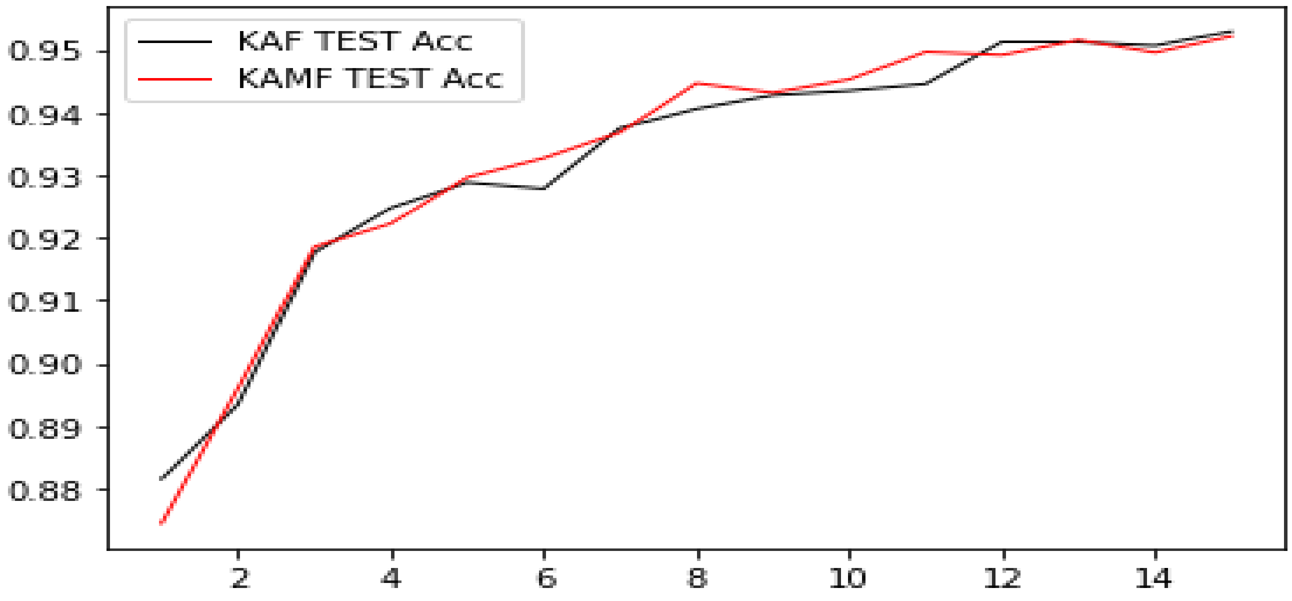
The testing has been done with multilayer perceptron model (MLP) in order to get the better intuition of algorithm. The 28X28 pixel greyscale image has been fed to linear input layer of 250 neurons whose outputs are given to linear hidden layer of 100 neurons whose outputs are fed to linear layer of 10 neurons as inputs thus producing the final outputs for 10 classes. The loss function used is mean square error loss function(MSE) with activation function as ReLU and evaluation matrix as categorical cross entropy. Initial step size is taken as 0.1. The algorithms have been tested against four main variants of CGD viz Polak Ribiere, Fletcher Revees, Hagar Zangar and Hesteenes Stiefel for 15 epochs each.



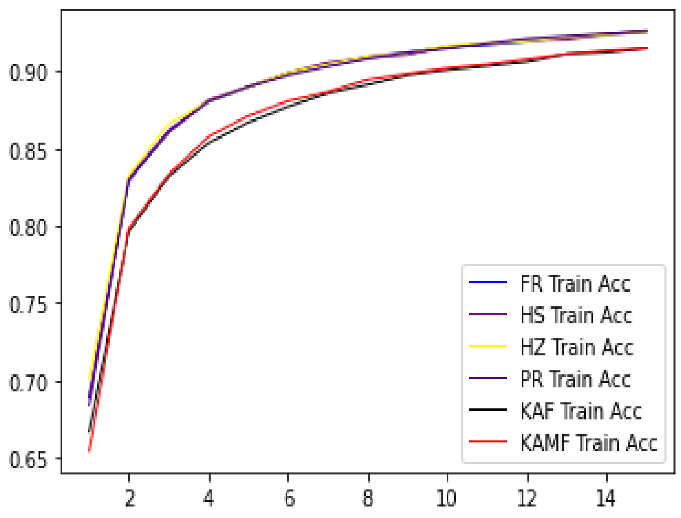
Kaf and KAMF training acc. on MNIST dataset



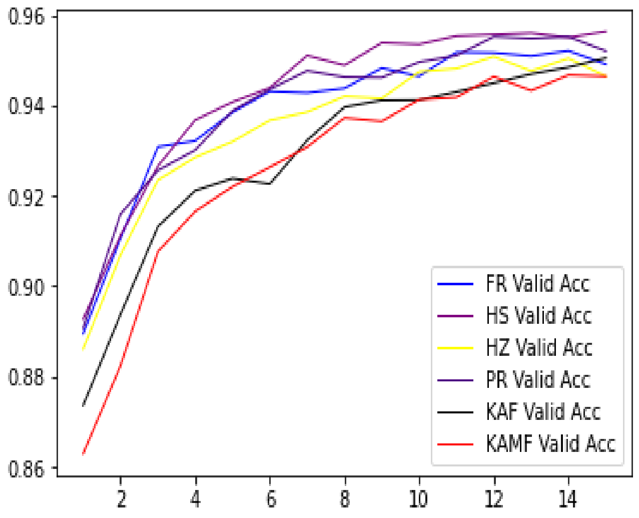
KAF and KAMF validation acc. on MNIST dataset



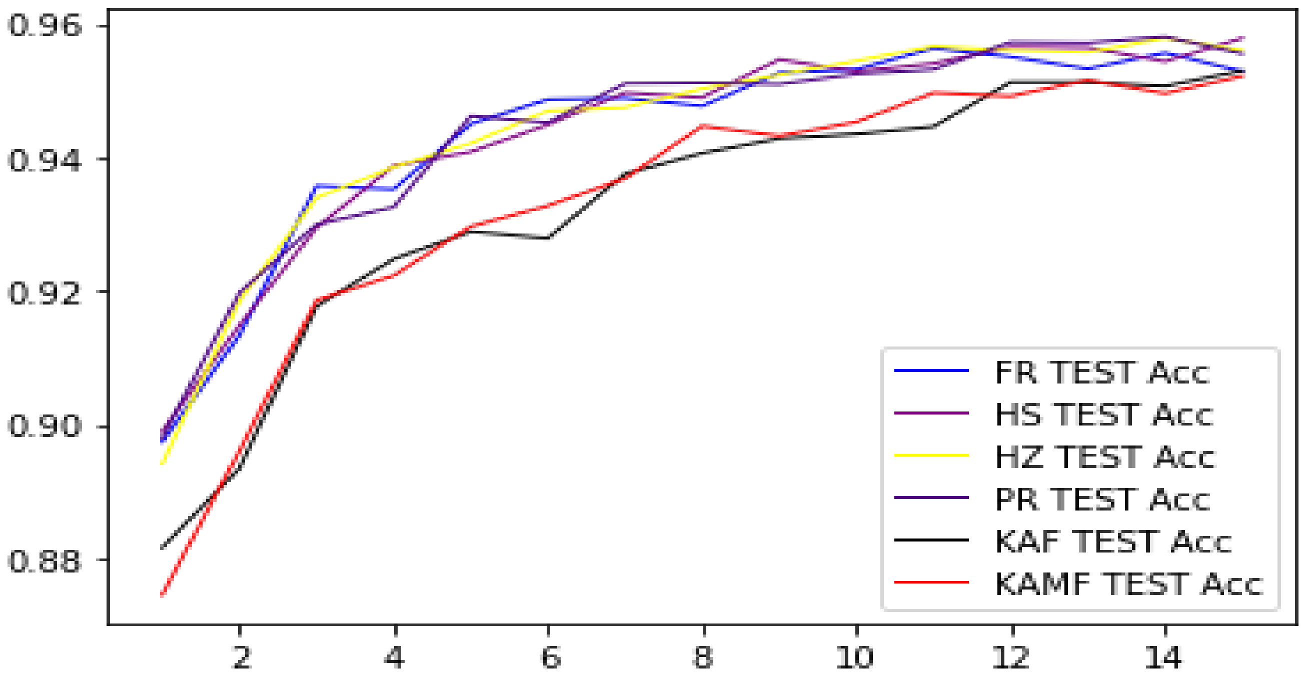
KAF and KAFM testing acc. on FMNIST dataset



Training acc. of all variants on FMNIST dataset

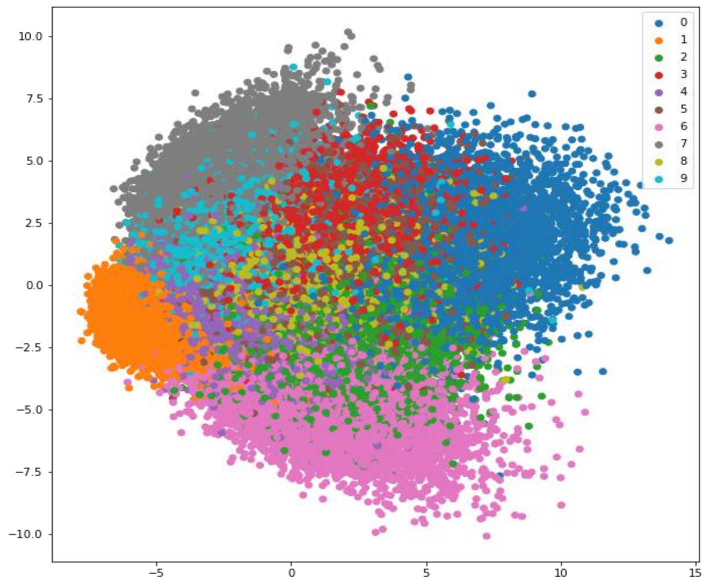


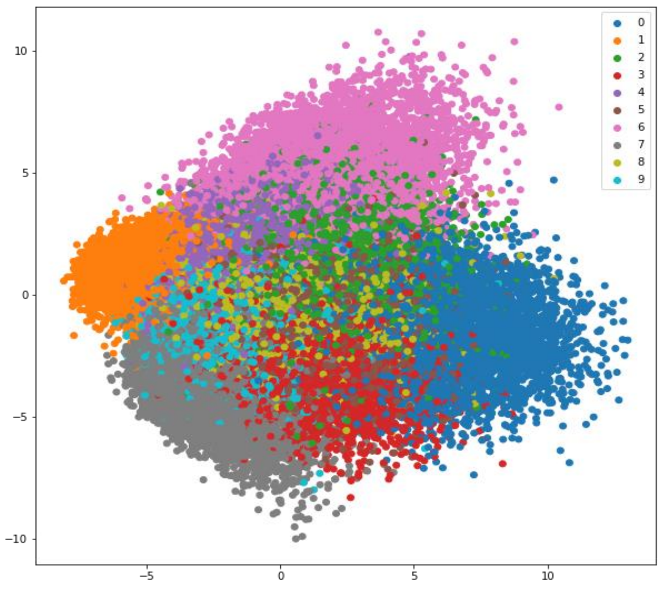
Validation acc. of all variants on FMNIST dataset



Discussion: The training phase is smooth, curvature and steep are better for KAF and KAMF. Although the accuracies are lower for KAF and KAMF but with higher steep they compensate it in the end this means model is learning better with KAF and KAMF than other variants of CGD. Same is the case with validation and testing accuracies that is after 15 epochs accuracies of all the algorithms are almost equivalent.

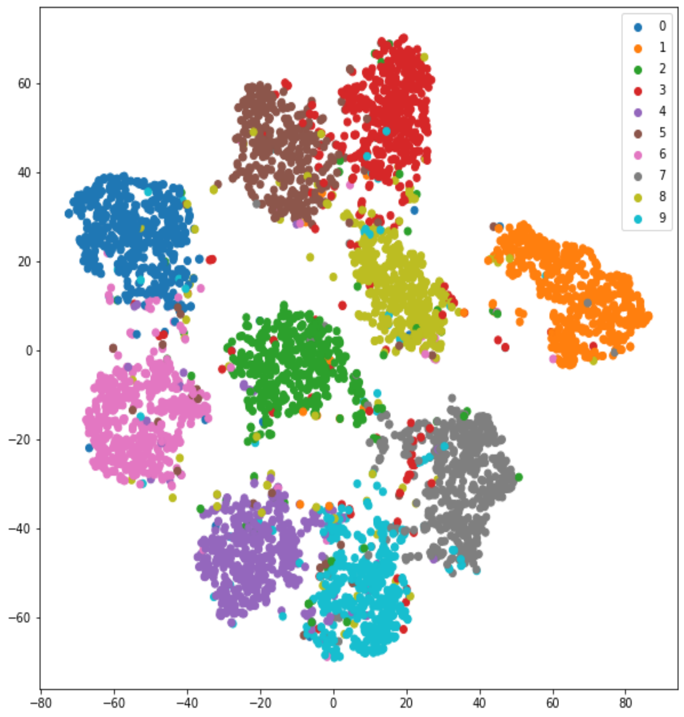
Distribution graphs of MNIST dataset for KAF and KAMF taken from output layer.

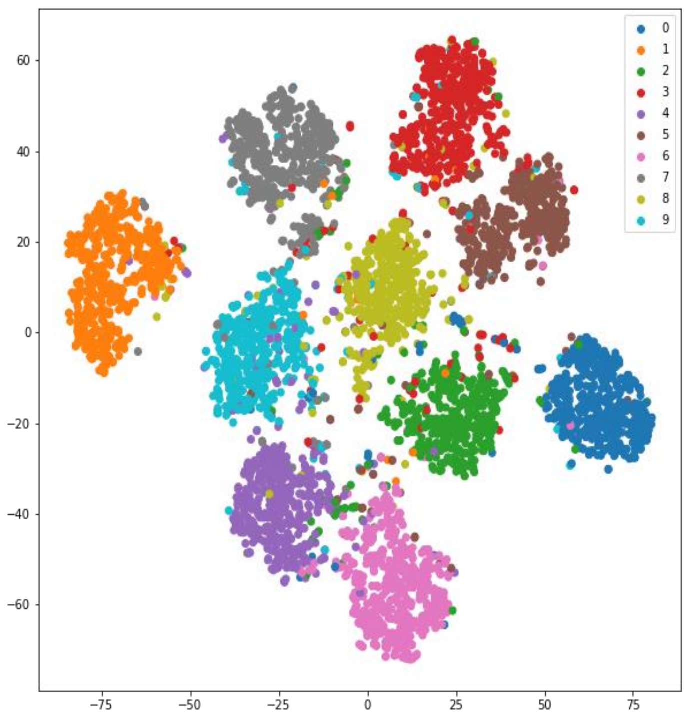




KAF MNIST distribution graph **KAMF MNIST distribution graph**

Clusterings of MNIST data by KAF and KAMF taken from output layer.

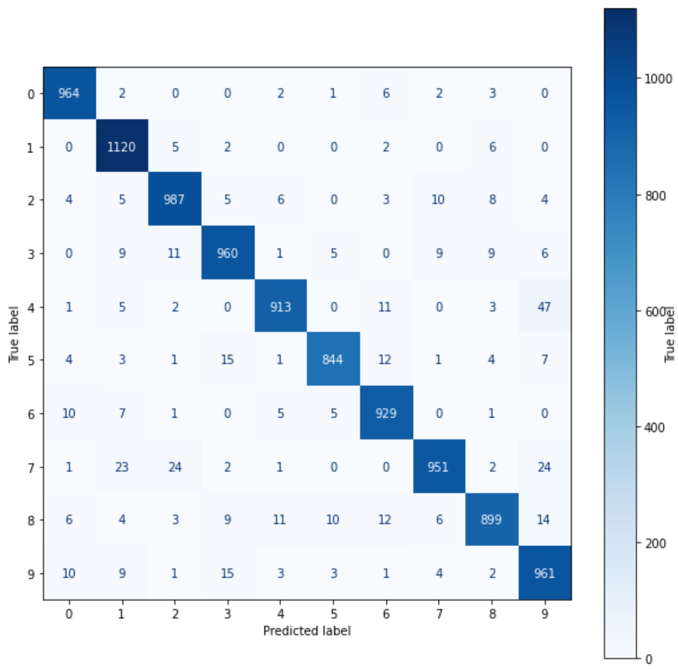




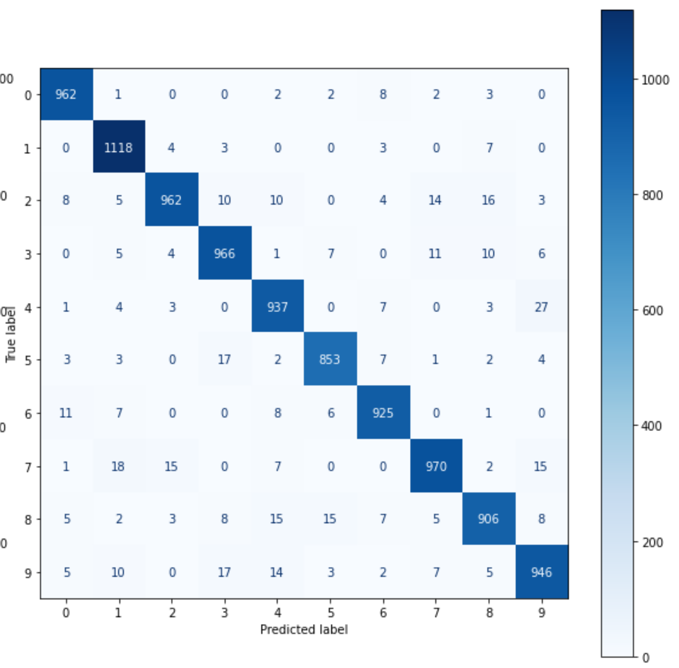
KAF MNIST clustering **KAMF MNIST clustering**

Discussion: All the classes are well clustered with greater inter cluster distance and lesser intra-cluster distance as visible in the images**.**

Confusion matrix's of KAF and KAMF for MNIST dataset



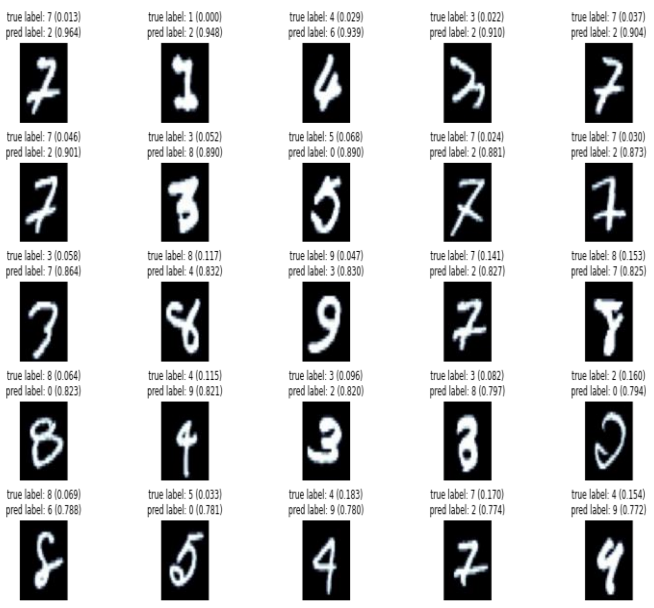
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	964	2	0	0	2	1	6	2	3	0
1	0	1120	5	2	0	0	2	0	6	0
2	4	5	987	5	6	0	3	10	8	4
3	0	9	11	960	1	5	0	9	9	6
4	1	5	2	0	913	0	11	0	3	47
5	4	3	1	15	1	844	12	1	4	7
6	10	7	1	0	5	5	929	0	1	0
7	1	23	24	2	1	0	0	951	2	24
8	6	4	3	9	11	10	12	6	899	14
9	10	9	1	15	3	3	1	4	2	961

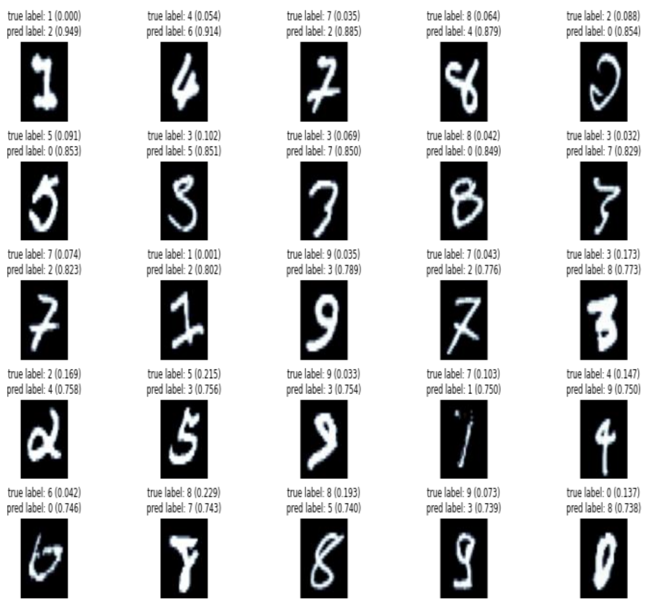


True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	962	1	0	0	2	2	8	2	3	0
1	0	1118	4	3	0	0	3	0	7	0
2	8	5	962	10	10	0	4	14	16	3
3	0	5	4	966	1	7	0	11	10	6
4	1	4	3	0	937	0	7	0	3	27
5	3	3	0	17	2	853	7	1	2	4
6	11	7	0	0	8	6	925	0	1	0
7	1	18	15	0	7	0	0	970	2	15
8	5	2	3	8	15	15	7	5	906	8
9	5	10	0	17	14	3	2	7	5	946

KAF confusion matrix for MNIST **KAMF confusion matrix for MNIST**

KAF and KAMF wrongly classified images


true label: 7 (0.013) pred label: 2 (0.964)
true label: 1 (0.000) pred label: 2 (0.948)
true label: 4 (0.029) pred label: 6 (0.939)
true label: 3 (0.022) pred label: 2 (0.910)
true label: 7 (0.037) pred label: 2 (0.904)
true label: 7 (0.046) pred label: 2 (0.901)
true label: 3 (0.052) pred label: 8 (0.890)
true label: 5 (0.068) pred label: 0 (0.890)
true label: 7 (0.024) pred label: 2 (0.881)
true label: 7 (0.030) pred label: 2 (0.873)
true label: 3 (0.058) pred label: 7 (0.864)
true label: 8 (0.117) pred label: 4 (0.832)
true label: 9 (0.047) pred label: 3 (0.830)
true label: 7 (0.141) pred label: 2 (0.827)
true label: 8 (0.153) pred label: 7 (0.825)
true label: 8 (0.064) pred label: 0 (0.823)
true label: 4 (0.115) pred label: 9 (0.821)
true label: 3 (0.096) pred label: 2 (0.820)
true label: 3 (0.082) pred label: 8 (0.797)
true label: 2 (0.160) pred label: 0 (0.794)
true label: 8 (0.069) pred label: 6 (0.788)
true label: 5 (0.033) pred label: 0 (0.781)
true label: 4 (0.183) pred label: 9 (0.780)
true label: 7 (0.170) pred label: 2 (0.774)
true label: 4 (0.154) pred label: 9 (0.772)


true label: 1 (0.000) pred label: 2 (0.949)
true label: 4 (0.054) pred label: 6 (0.914)
true label: 7 (0.035) pred label: 2 (0.885)
true label: 8 (0.064) pred label: 4 (0.879)
true label: 2 (0.088) pred label: 0 (0.854)
true label: 5 (0.091) pred label: 0 (0.853)
true label: 3 (0.102) pred label: 5 (0.851)
true label: 3 (0.069) pred label: 7 (0.850)
true label: 8 (0.042) pred label: 0 (0.849)
true label: 3 (0.032) pred label: 7 (0.829)
true label: 7 (0.074) pred label: 2 (0.823)
true label: 1 (0.001) pred label: 2 (0.802)
true label: 9 (0.035) pred label: 3 (0.789)
true label: 7 (0.043) pred label: 2 (0.776)
true label: 3 (0.173) pred label: 8 (0.773)
true label: 2 (0.169) pred label: 4 (0.758)
true label: 5 (0.215) pred label: 3 (0.756)
true label: 9 (0.033) pred label: 3 (0.754)
true label: 7 (0.103) pred label: 1 (0.750)
true label: 4 (0.147) pred label: 9 (0.750)
true label: 6 (0.042) pred label: 0 (0.746)
true label: 8 (0.229) pred label: 7 (0.743)
true label: 8 (0.193) pred label: 5 (0.740)
true label: 9 (0.073) pred label: 3 (0.739)
true label: 0 (0.137) pred label: 8 (0.738)

KAF MNIST wrongly classified images **KAMF MNIST wrongly classified images**

Discussion: 1 is the best classified digit for both the algorithms as well in wrongly classified digits, the digits with wrong predicted labels hard represent the correct digit whose label is given.

COMPARISON & RESULTS

Overall the proposed algorithms KAF and KAMF have lower starting points for training accuracies leading to better tolerance to overfitting and also have higher steep which leads to better accuracy. Both the proposed algorithms show better performance with more complex datasets and are much stable for all the scenarios. KAMF is accurate but KAF is computationally cost effective. On the contrary other variants of CGD have higher starting points for training accuracies which in the long run will lead to overfitting and steep is also lower depiction slow learning rate. They slow lesser stability in one or the other experiment. Their accuracy depends on data complexity and usually doesn’t work well with complex datasets.

FUTURE ANALYSIS

In future we have to look for how our proposed algorithms are going to work with different datasets and also provide further theoretical analysis of our algorithms as well need to prove theoretically why our algorithms are working better and how step size is affecting the convergence rate.

REFERENCES

[1] A Choromanska., “Convex and non-convex worlds in machine learning”, Courant Institute of Mathematical Sciences, New York University

[2] Nocedal J and Wright S., “ Numerical Optimization Springer Series in Operations Research and Financial Engineering”. (Springer New York) ISBN 9780387303031, 2006.

[3] Sohl-Dickstein J, Poole B and Ganguli S., “CoRR abs” /1311.2115 (Preprint 1311.2115) URLhttp://arxiv.org/abs/1311.211, 2013

[4] LeCun Y, Bottou L, Orr G B and M ̈uller K R., “Efficient BackProp” (Berlin, Heidelberg: Springer Berlin Heidelberg) pp 9–50 ISBN 978-3-540-49430-0, 1998.

[5] Wilamowski B M and Yu H., IEEE Transactions on Neural Networks 21 930–937 ISSN 1045-922, 2010

[6] Martens J Proceedings of the 27th International Conference on International Conference on Machine Learning 735–742 URL http://dl.acm.org/citation.cfm?id=3104322.3104 , 2010

[7] Møller M F Neural Networks 6 525 – 533 ISSN 0893-6080 URL http://www.sciencedirect.com/science/article/pii/S089360800580056, 1993

[8] Hong Hui Tan and King Hann Lim., “Review of second-order optimization techniques in artificial neural networks backpropagation”, Department of Electrical and Computer Engineering, Curtin University Malaysia, CDT 250, 98009, Miri, Sarawak, Malaysia

[9] Yinyu Ye., “Second Order Optimization Algorithms I” , Department of Management Science and Engineering, Stanford University, Stanford, CA 94305, U.S.A

[10] Wei-Ta Chu “An Introduction to Optimization”, Spring, 2014 Chapter 9 Newton’s Method

[11] Donald Goldfarb, Yi Ren, Achraf Bahamou., “Practical Quasi-Newton Methods for Training Deep Neural Networks”, Department of Industrial Engineering and Operations Research, Columbia University, New York, NY 10027

[12] S Nandy et al., “An Improved Gauss-Newtons Method based Back-propagation Algorithm for Fast Convergence” International Journal of Computer Applications, (0975 – 8887) Volume 39– No.8, February 2012, DETS, Kalyani University, Kalyani, Nadia, West Bengal.

[13] Conghui Tan, Shiqian Ma, Yu-Hong Dai, Yuqiu Qian., “Barzilai-Borwein Step Size for Stochastic Gradient Descent”.

[14] https://en.wikipedia.org/wiki/Conjugate_gradient_method Conjugate gradient method

[15] J. R. Shewchuk., “ An Introduction to the Conjugate Gradient Method Without the Agonizing Pain” August 4, 1994.