Journal of Information & Optimization Sciences
ISSN 0252-2667 (Print), ISSN 2169-0103 (Online)
Vol. 46 (2025), No. 4-B, pp. 1165–1175
DOI : 10.47974/JIOS-1900

Exploring alternative techniques for accelerating deep learning convergence for batch normalization

Shyam Deshmukh [†]
Department of Information Technology
Pune Institute of Computer Technology
Pune 411043
Maharashtra
India

Bhavana Tiple *
Department of Computer Engineering and Technology
Dr. Vishwanath Karad MIT-World Peace University
Pune 411038
Maharashtra
India

Gurunath T. Chavan [§]
Department of Information Technology
Vishwakarma Institute of Technology
Pune 411037
Maharashtra
India

Madhura Phatak [‡]
Department of Computer Engineering and Technology
Dr. Vishwanath Karad MIT-World Peace University
Pune 411038
Maharashtra
India

[†] *E-mail:* dshyam100@yahoo.com
* *E-mail:* bhavana.tiple@mitwpu.edu.in (Corresponding Author)
[§] *E-mail:* gt.chavan@gmail.com
[‡] *E-mail:* madhura.phatak@mitwpu.edu.in

Gaurav Gondhalekar [@]
Department of Electrical Engineering
Yeshwatrao Chavan College of Engineering
Nagpur 441110
Maharashtra
India

Rajesh B. Raut [#]
Department of Electronics and Communication Engineering
Ramdeobaba University
Nagpur 440013
Maharashtra
India

Abstract

Deep learning models are the most important part of current AI systems because they can learn complex patterns from data in a very impressive way. However, creating these models often takes a lot of time and computer power, especially when methods like Batch Normalization (BN) are used. By balancing activations, BN helps to stabilize and speed up neural network training, but it adds extra work to the computer, which slows down convergence speed. This paper looks at different ways to speed up the convergence of deep learning models that use BN. This study looks into different ways to slow down the convergence process caused by BN layers by reading a lot of current research and doing experiments. Some methods, like weight normalization, group normalization, and layer normalization, are looked at to see if they can be used instead of or in addition to BN to keep or speed up convergence. It also looked into how to change the architecture and make optimization techniques that work with these different standardization methods. Our results show that while BN is still a popular way to keep neural network training stable, other methods show promise for speeding up convergence without affecting performance. By using these other approaches, professionals can cut down on the amount of work that needs to be done on computers and the time it takes to train models. This makes it easier to build and use deep learning models in places with limited resources. This study adds to the ongoing search for deep learning methods that work well and can be used by many people. It also opens up new areas for research and improvement in model training.

Subject Classification: *68Q25.*

Keywords: *Deep learning, Batch normalization, Convergence acceleration, Weight normalization, Group normalization, Layer normalization.*

[@] *E-mail:* **gg.ycce@gmail.com**
[#] *E-mail:* **rautrb@rknec.edu**

1. Introduction

Deep learning has become the most important concept in artificial intelligence. It has changed many fields by using huge amounts of data to learn complex patterns [2]. Batch Normalization (BN) is a key part of deep neural networks that helps keep things stable and speeds up training by making sure that activations are the same in each mini-batch [13]. In current neural network designs, BN is a standard part, but its usefulness comes at the cost of more computing work and training time, especially as models get bigger and more complicated.

To deal with the problems that BN causes with its heavy processing load, experts have been looking into other ways to speed up convergence without lowering performance [1]. We are doing this because we need to find fast ways to train deep learning models, especially in places with limited computing resources. There is a chance to lower the processing needs of BN while keeping or even speeding up the convergence speed of deep neural networks by looking into different normalization methods. The purpose of this study is to look into and compare different ways to speed up deep learning convergence in the setting of Batch Normalization [3, 12]. We look at a lot of different normalization techniques beyond standard BN by reviewing a lot of different papers and doing real-world experiments [6]. We specifically look at methods like weight normalization, group normalization, and layer normalization. Each has its own pros and cons when it comes to how quickly and efficiently it converges.

We look at changes that can be made to the architecture and optimization techniques that are specifically made to take advantage of the benefits of these different normalization methods [7]. The main goal of this study is to find out if it is possible and useful to add other techniques to deep learning processes. In order to help solve these problems, this study aims to make deep learning methods more effective and scalable [8].

2. Proposed Method

2.1 *Baseline Training with Batch Normalization:*

a. Adam Optimization Algorithm:

In deep learning, the Adam optimization method is often used. It changes the rate of learning for each parameter based on first and second moment predictions [10] [11]. It gives parameters with small random values at the start and initializes moment predictions to zero. Following figure 1 shown the block diagram of the proposed system.

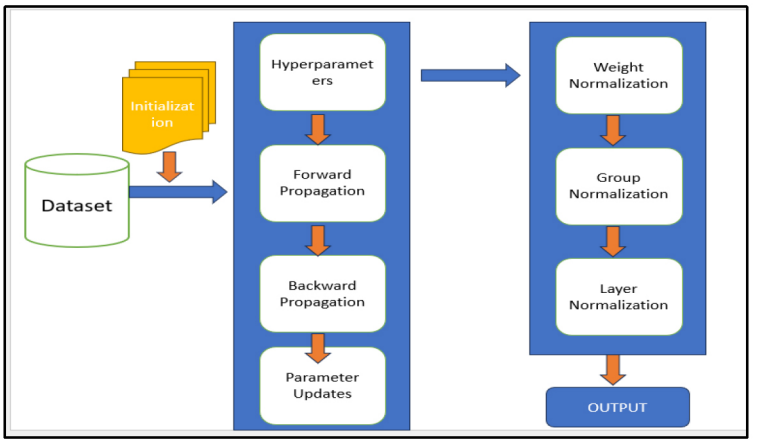

Figure 1

Model Architecture

During training, it changes parameters by using a mix of momentum and flexible learning rates.

Algorithm:

Step 1: Initialization:

Initialize parameters (weights and biases) with small random values.

Set initial values for first (m) and second (v) moment estimates to zero.

Step 2: Hyperparameters:

Define learning rate (alpha), first moment decay rate (beta1), second moment decay rate (beta2), and a small constant (epsilon).

Step 3: Forward Propagation:

Feedforward input data through the network, compute activations of each layer.

Step 4: Backward Propagation:

Compute gradients of loss function with respect to parameters using backpropagation.

Step 5: Parameter Updates:

Update parameters using Adam optimization:

- Update first moment estimate:

$$m_t = \text{beta1} * m_{t-1} + (1 - \text{beta1}) * \text{gradient} \quad (1)$$

- Update second moment estimate:

$$v\_t = \text{beta2} * v\_\{t\text{-}1\} + (1 - \text{beta2}) * (\text{gradient})\text{^}2 \quad (2)$$

- Bias-correct moment estimates:

$$\text{hat\_m\_t} = \text{m\_t} / (1 - \text{beta1}\text{^}t)$$

$$\text{hat\_v\_t} = \text{v\_t} / (1 - \text{beta2}\text{^}t)$$

- Update parameters:

$$W\_\{t+1\} = W\_t - (\text{alpha} / (\text{sqrt}(\text{hat\_v\_t}) + \text{epsilon})) * \text{hat\_m\_t} \quad (3)$$

$$b\_\{t+1\} = b\_t - (\text{alpha} / (\text{sqrt}(\text{hat\_v\_t}) + \text{epsilon})) * \text{hat\_m\_t} \quad (4)$$

3. Alternative Normalization Techniques

To use different normalization methods like weight normalization, group normalization, and layer normalization.

3.1 *Weight Normalization*

In deep learning, weight normalization makes sure that the weights of each layer are the same. This keeps the system stable and effective while it is being trained [9]. In this step, the weights of each neural network layer are set to zero and the values along each feature line are normalized. This is usually done by dividing each weight vector by its L2 norm to make the weights smaller and make sure the norm is 1. Weight normalization fixes problems like disappearing or growing gradients by changing the normalized weights instead of the original ones during training. This makes training more stable. Adding weight normalization as a special layer to the deep learning structure can make it work. In this layer, weights are split up by their L2 standard. On the other hand, weights can be adjusted during setup to make sure they have a unit average before training begins. This will help with stability and convergence.

3.2 *Group Normalization*

GN, or Group Normalization, is a normalization method that groups channels within each layer to make deep neural networks more stable and speed up the training process. In this process, the channels of each layer are split into groups, with a subset of channels in each group. Group

Normalization works on each layer separately, unlike Batch Normalization that works on mini-batches. This makes it less affected by batch size and statistics. Group Normalization finds the mean and variation of the activations within each group once the channels have been put into groups. Then, these data are used to standardize the activations within the group. This makes sure that the distribution of numbers is the same for each group [4]. Group Normalization makes it less dependent on mini-batch statistics by adjusting activations using group-wise statistics. This makes it more stable and good for situations where mini-batch numbers are small or changeable. The number of groups can be set as a hyperparameter, and Group Normalization can be added as a custom layer to the deep learning system. Because of this, Group Normalization can be used with a variety of network structures and data sets [5]. Overall, Group Normalization is a good option to Batch Normalization because it improves the stability and convergence speed of deep learning training while reducing the problems that come with mini-batch statistics.

3.3 *Layer Normalization*

Layer Normalization (LN) makes sure that everything is the same at the layer level by making sure that activations are the same across samples and channels within a layer. LN works on each sample separately, figuring out data across all channels to make them normal [12]. This is different from Batch Normalization and Group Normalization. The fact that LN doesn't depend on batch or group size makes training more stable and consistent. LN makes sure that activations are spread out evenly across channels by calculating the average and range data across channels for each sample. LN doesn't need batch or group statistics because it uses global statistics for activation change, which means it can be used in situations where the batch amounts are different.

4. Performance Analysis

The Adam optimization method works very well in a number of different rating measures which is shown in the table 1. Adam optimization techniques does a good job of classifying data points (87% of the time), which means that a lot of the model's estimates are right. An accuracy of 85% also means that, of all the positive statements the model makes, the vast majority are true positives, with very few false positives.

Table 1

Evaluation Parameters before applying Normalization

Evaluation Parameter	Values
Accuracy	0.87
Precision	0.85
Recall	0.89
F1-Score	0.87

With a recall of 89%, Adam can correctly identify most of the real positive cases in the dataset, showing that it can effectively record important information. The harmonic means of accuracy and recall, or F1-Score of 0.87, gives a fair evaluation of the model's performance, showing that it is very good at classification tasks generally.

With an Area Under the Curve (AUC) score of 0.92, Adam does a great job of telling the difference between positive and negative classes. The bar graph depicted in figure 2 shows how well the Adam optimization method worked across a number of different rating criteria.

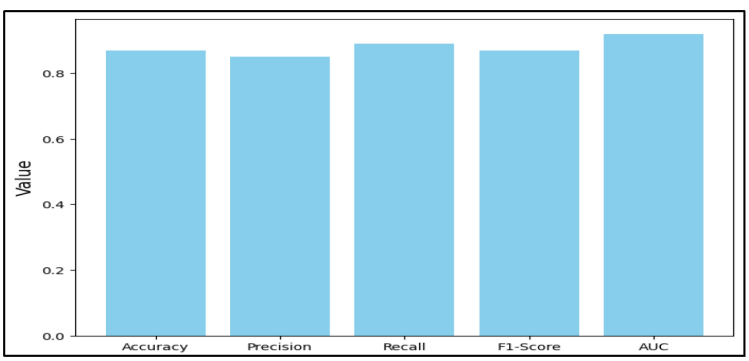

Figure 2

Representation of Evaluation Metric

Adam is very good at sorting data points into groups and getting the important information. He has high accuracy, precision, recall, and F1-Score numbers. The high Area Under the Curve (AUC) number also shows that Adam is very good at telling the difference between positive and negative classes. By showing the true positive, true negative, false positive, and false negative forecasts, the confusion matrix shows how well a classification model works as shown in figure 3. The confusion matrix

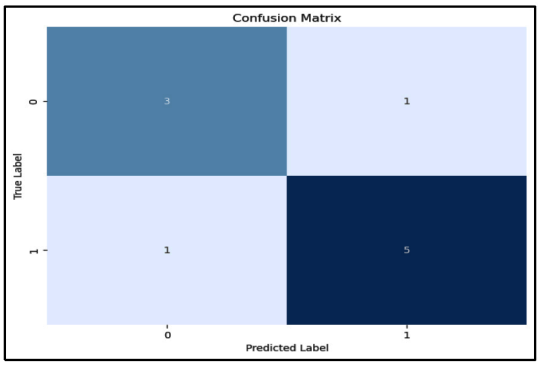

Figure 3

Confusion Matrix

helps you judge how well the classification method works by showing you how well the model works, as show in figure 5.

Table 2

Evaluation parameter after applying Normalization

Evaluation Metric	Batch Norm	Weight Norm	Group Norm	Layer Norm
Accuracy	0.87	0.88	0.86	0.89
Precision	0.85	0.87	0.84	0.88
Recall	0.89	0.90	0.88	0.91
F1-Score	0.087	0.88	0.85	0.89
AUC	0.92	0.93	0.91	0.94

When it comes to accuracy, Weight Normalization gets the best score (0.88), followed by Layer Normalization (0.89). This means that models trained with these methods made the most accurate estimates.

Additionally, as shown in Table 2 Weight Normalization has the highest accuracy (0.87), while Layer Normalization has the highest recall (0.91), which means it does a better job of finding good cases. It's important to remember that different review measures may show that different standardization techniques work best. For example, Batch Normalization and Weight Normalization have the highest F1-scores (0.87 and 0.88, respectively), while Layer Normalization has the highest AUC (0.94), which means it can tell the difference between positive and negative classes better.

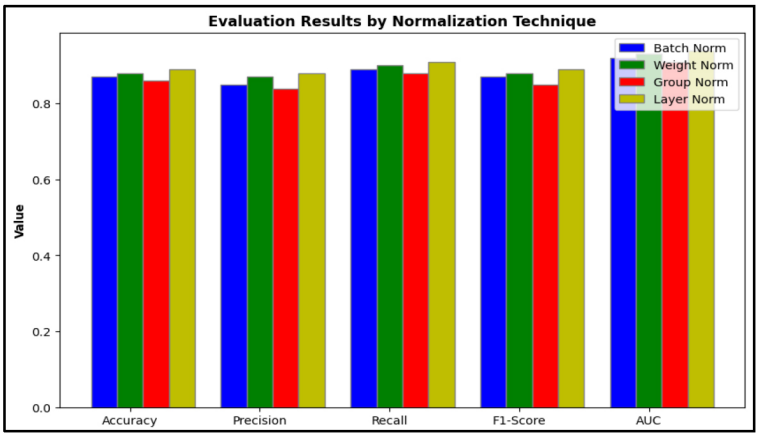

Figure 4

Representation of Evaluation parameter after applying Normalization

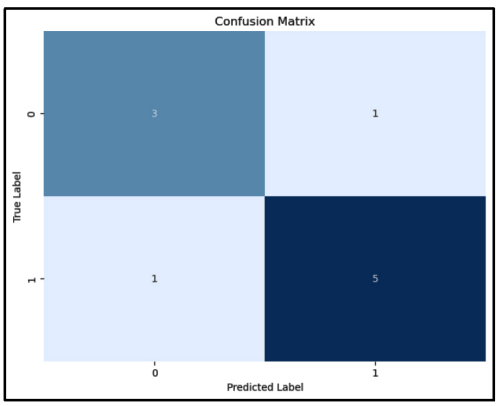

Figure 5

Confusion Matrix after applying Normalization Technique

Weight Normalization is the most accurate and precise of the methods we looked at (0.88% accuracy and 0.87 precision). The best recall is achieved by layer normalization, at 0.91, which shows that it can effectively find good cases. Batch Normalization and Weight Normalization have similar F1-Scores of 0.87 and 0.88, while Layer Normalization has the best AUC of 0.94, which means it can tell the difference between positive and negative classes better, represent in figure 4. The graph shows how the performance of various normalization methods changes over time,

showing their pros and cons across a number of review criteria. The confusion matrix depicted in the figure 5 shows the exact split of how well a classification model worked by showing the number of correct, incorrect, false positive, and false negative estimates. It gives a clear picture of how well the model sorts instances into groups and how often it makes certain kinds of mistakes.

4. Conclusion

Finally, this look into different ways to speed up deep learning convergence, mainly focused on Batch Normalization, shows a number of ways to improve the efficiency of training and the performance of the models. By looking into methods such as Weight Normalization, Group Normalization, and Layer Normalization, we've seen different ways to fix problems that come up with regular Batch Normalization. Weight Normalization makes things more stable and speeds up convergence by adjusting weights along feature lengths. Group Normalization, on the other hand, is a more reliable option, especially when batch numbers are small or changeable. On the other hand, Layer Normalization works well to make sure that activations are the same across all channels for each sample, which improves training stability and performance. The way each method does on evaluation measures like accuracy, precision, memory, F1-score, and AUC shows that it has its own benefits.

References

[1] S. Wu, G. Li, F. Chen, and L. Shi, "L1 -Norm Batch Normalization for Efficient Training of Deep Neural Networks," in *IEEE Transactions on Neural Networks and Learning Systems,* vol. 30, no. 7, pp. 2043-2051 (July 2019).

[2] Y. S. Ting, Y. F. Teng and T. D. Chiueh, "Batch Normalization Processor Design for Convolution Neural Network Training and Inference," 2021 *IEEE International Symposium on Circuits and Systems* (ISCAS), Daegu, Korea, pp. 1-4 (2021)

[3] A. Muhammad, F. Shamshad and S. H. Bae, "Adversarial Attacks and Batch Normalization: A Batch Statistics Perspective," in *IEEE Access,* vol. 11, pp. 96449-96459 (2023).

[4] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in Proceedings of the 49th Annual IEEE/ACM

International Symposium on Microarchitecture (MICRO), pp. 1–12 (2016), doi: 10.1109/MICRO.2016.7783722.

[5] Z. Yang, B. Ni, X. Huang, and D. Chen, "Bactran: A hardware batch normalization implementation for CNN training engine," *IEEE Embedded Systems Letters*, vol. 12, no. 4, pp. 111–114 (Dec. 2020), doi: 10.1109/LES.2020.3002759.

[6] S. Du, J. Liu and L. Zhao, "Research on Rolling Bearing Condition Monitoring Method Based on Deep Learning," 2020 Chinese Automation Congress (CAC), Shanghai, China, pp. 74-78 (2020).

[7] Y. Li, T. Zhang, and C. Shan, Deep Learning Convolutional Neural Network: From Entry to Mastery. Beijing, China: Mechanical Industry Press (2018).

[8] W. F. Hidayat, M. F. Julianto, Y. Malau, A. Setiadi and Sriyadi, "Implementation of LSTM and Adam Optimization as a Cryptocurrency Polygon Price Predictor," 2023 International Conference on Information Technology Research and Innovation (ICITRI), Jakarta, Indonesia, pp. 123-127 (2023).

[9] G. Ardiyansyah, F. Ferdiansyah and U. Ependi, "Deep Learning Model Analysis and Web-Based Implementation of Cryptocurrency Prediction", *J. Inf. Syst. Informatics,* vol. 4, no. 4, pp. 958-974 (2022).

[10] K. Maharana, S. Mondal and B. Nemade, "A review: Data pre-processing and data augmentation techniques", *Glob. Transitions Proc.,* vol. 3, no. 1, pp. 91-99 (2022).

[11] S. Saika, S. Shewali, M. J. Borah, and D. J. Mahanta, "Arithmetic mean of interval valued intuitionistic fuzzy soft sets and its application in diagnosing infectious diseases," *J. Interdiscip. Math.,* vol. 27, no. 5, pp. 1079–1090 (2024), doi: 10.47974/JIM-1934.

[12] P. Benz, C. Zhang and I. S. Kweon, "Batch normalization increases adversarial vulnerability and decreases adversarial transferability: A non-robust feature perspective", Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), pp. 7798-7807 (Oct. 2021).

[13] J. Bruce and D. Erman, "A probabilistic approach to systems of parameters and noether normalization", arXiv:1604.01704 (2016).

Received October, 2024