iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 11 **Issue:** I **Month of publication:** January 2023

DOI: https://doi.org/10.22214/ijraset.2023.48771

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue I Jan 2023- Available at www.ijraset.com

Optimization of Deep Neural Networks Using DAPP (DNN Acceleration Using Ping-Pong) Approach

Mashkoor Ahmad Naik[1], Er. Jasdeep Singh[2]
[1]*Scholar,* [2]*Assistant Professor, Department Of Computer Science and Engineering, RIMT University, Mandi Gobindgarh, Punjab, India*

Abstract: ***Deep Neural Networks (DNNs) are one of the leading classification algorithms. Deep Learning has achieved remarkable milestones such as Google-Net and Alpha-Go. They have shown promising results in pattern recognition for images and text, language translation, sound recognition, and many more. DNN has been widely accepted and also employed for pattern matching and image recognition. All these applications are possible as these networks emulate functioning of human brain and hence name "Neural" Network.However, to provide competent results, these millions of neurons in neural networks needs to be trained. For which billions of operations are to be carried out. Training of these many neurons and with these many operations is a time-consuming affair. Hence choice of network and its parameter play an important role both in providing accurate trained network and time taken for training. If the network is deep and has plethora of neurons, the time taken is considerably high as training works sequentially on batches of dataset using Sequential Back-propagation Algorithm.To accelerate the training there are many hardware solutions like use of GPU, FPGA and ASICs. However because of popularity of DNN there is increase in demand in mobile and IoT platform devices. These are resource constrained devices, where power and size of these device, restricts usage and implementation of deep NN (Neural Network).Simulation of DAPP is done on MNIST and CIFAR-10 datasets using System-C. Additionally, this technique has been adapted for multi-core architectures. The design shows a reduction in time by 38% for 3 layers of CNN and 92% for 10 layers of CNN, while maintaining the accuracy of networks. This generic methodology has been implemented for Vanilla RNN and LSTM networks. An improvement of 38% for Vanilla RNN and 40% for LSTM has been demonstrated by this methodology.***

I. INTRODUCTION

Deep neural networks are state-of-the-art pattern recognition algorithms. The deep layer architecture of neural network allows it to extract and to learn complex and high-dimensional data features creating a discriminative classifier. Like genetic algorithms and simulated annealing, DNNs are based on an analogy with real-world biological/physical processes. They mimic the behavior of human brain neurons. The degree of influence a neuron has on another neuron is reflected by a numerical weight. In simple terms, training a DNN is the process of selecting values for the weights so that the overall neural network produces the desired output for a given input.NNs are used in analyzing huge volume of data to extract patterns and bring new discoveries. In recent past, neural network has successfully taken a giant leap in many problems related to image recognition, speech recognition, automatic machine translation, natural language processing, automatic music composition and self-driving cars. They have outperformed human beings in games such as Alpha-Go. They are used in many applications such as video surveillance, mobile robot vision, and pedestrian detection. [3][4][5].Along with this, huge amount of data are generated from various other domains, like Internet-of-Things and today's tremendous amount of devices able to capture pictures and videos, the potential for DNNs have vastly increased. By making our devices able to recognize its surroundings, there could be a huge amount of potential interesting applications. One other field is biology, here it may be a study of genome of any organism or it may to study of any chemical molecular structure. These have a complex and huge datasets, to learn such complex patterns, layers in DNNs are being increased manifold. This has led to a substantial increase in trainable parameters and computation.

To provide more accurate results, the state-of-the-art DNN requires millions of parameters and billions of operations to process a single image, which represents a computational challenge for general purpose processors. As a result, hardware accelerators such as Graphic Processing Units (GPU) [6] [7], Field Programmable Gate Arrays (FPGA)[8][9], and Application Specific Integrated Circuits (ASIC) [10][11] , have been utilized to improve the throughput of the DNN.

Among these accelerators, GPUs are the most widely used to improve both training and classification process of ConvNet, because of their high throughput and memory bandwidth. However, GPUs consume a considerable amount of power which is another important evaluation metric in the modern digital systems. ASIC design, on the other hand, has achieved high throughput with low power consumption by assigning dedicated resources and customizing memory hierarchy. But the development time and cost is significantly high compared to other solutions. As an alternative, FPGA-based accelerators provide high throughput, low power consumption and reconfigurability at a reasonable price.

A Neural Network is a collection of layers which are densely connected to each other. Each connection has some weights. NN is fed with the dataset. It is fed with one or more inputs along with corresponding weights. It takes the weighted summation of inputs and applies a non-linear function called activation function which are discussed as above. The layers perform operations on the output of previous layer. Thus, NN transforms the original input, layer by layer, to desired output and the category with maximum score is predicted. The error in prediction is propagated back in path to tune weights which results in a trained model. Fine-Tuning is done with respect to training algorithms. Some examples of training algorithm are Gradient Descent, Adam and Momentum Optimizer.

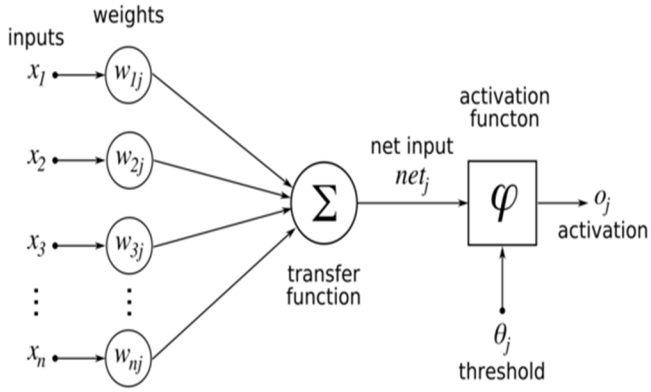


Figure 1.1 Structure of Neuron

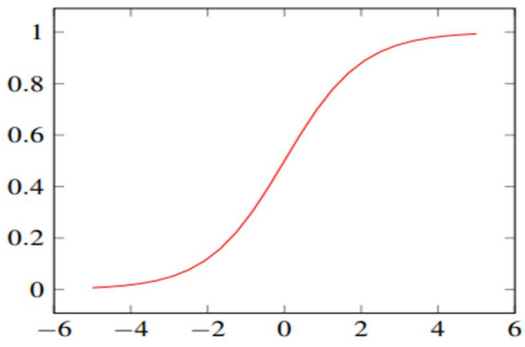
A. Activations

Whenever we see, hear, feel and think something, a synapse (electrical impulse) is fired from one neuron to another in the hierarchy which enables us to learn, remember and memorize things in our daily life. For a particular activity, a specific set of neurons are fired in human brain.

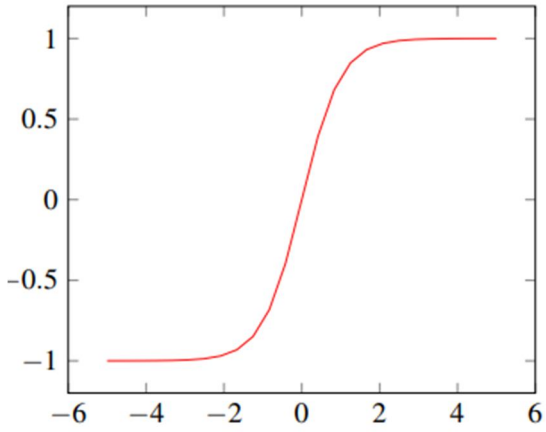
Likewise, Activation functions are important for a Neural Network. They have the role of firing a neuron in neural network. They introduce non-linear properties to the Network. In a neuron, the sum of products of inputs and their corresponding Weights is taken and Activation function f(x) is applied to it to get the output of that layer and feed it as an input to the next layer. A Neural Network without Activation function would simply be a Linear Regression Model, which has limited power and does not performs good most of the times. Also without activation function our Neural network would not be able to learn and model other complicated kinds of data such as images, videos, audio, speech etc.Most popular types of Activation functions are described as follows:

1) *Sigmoid:* It is a activation function of form f (x) = 1/1 + exp(−x) Its Range is between 0 and 1, which generally represents the probability of class as output. But it has major reasons which have made it fall out of popularity - Vanishing gradient problem. Secondly, its output isn't zero centered. It makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder.

(a) Sigmoid



(b) Tanh



(c) Relu

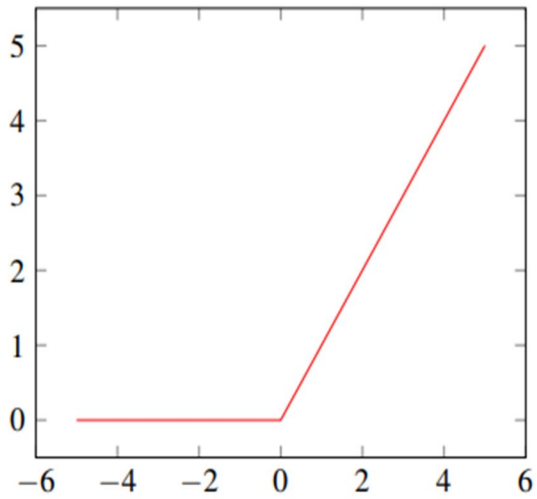


Figure1.2ActivationFunctions

2) *Tanh:* It's mathematical formula is f (x) = (1 − exp(−2x))/(1 + exp(−2x)). Now it's output is zero centered because its range in between -1 to 1 i.e -1 < output < 1. Hence optimization is easier in this method hence in practice it is always preferred over Sigmoid function. But still it suffers from Vanishing gradient problem.
3) *Relu:* Rectified Linear Unit (Relu) is R(x) = max(0, x) i.e if x <0 , R(x) = 0 and if x>= 0 , R(x) = x. It avoids and rectifies vanishing gradient problem. Almost all deep learning Models use Relu these days. But limitation of Relu is that, it can only be used within Hidden layers of a Neural Network Model.

B. Layers

Convolution Layer has a set of kernels and bias. The features extracted by convolution layer are stored in kernels. First they are initialized with some random numbers. During back-propagation, the gradients are calculated and then the kernels are updated. Convolution Layers are generally followed by Pooling Layer. Pooling Layer is used to scale down its input by sub-sampling. It creates "summaries" of each sub-region. Dropout Layer is used to prevent the model from over-fitting. The last part of NN is Dense Layer which takes the output of previous layers and converts them into scores. The highest score value is predicted by NN.

C. Optimizers

Optimization is the process of finding the set of parameters W that minimizes the loss function. The following Optimizers have been used in this report.

1) *Stochastic Gradient descent (SGD):* is one of the most popular optimizer. It computes the gradient of the cost function w.r.t. to the parameters θ for the entire training dataset:

$$\theta=\theta-\eta.\delta_\theta J(\theta) \quad (1.1)$$

2) *Momentum Optimizer:* The high variance oscillations in SGD makes it hard to reach convergence, so a technique called Momentum was invented which accelerates SGD by navigating along the relevant direction and softens the oscillations in irrelevant directions. In other words all it does is adds a fraction "γ"of the update vector of the past step to the current update vector.

$$V(t) = \gamma.V(t-1) + \eta.\delta\theta J(\theta) \quad (1.2)$$

and finally we update parameters by $\theta = \theta - V(t)$.

3) *Adam Optimizer:* Adam stands for Adaptive Moment Estimation. Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. It keeps the tract of two things, momentum and velocity**.**

$$\begin{aligned} lrt &= learningrate * \frac{sqrt(1-\beta 2^t)}{(1-\beta 1^t)}) \\ mt &= \beta 1 * mt-1+(1-\beta)*g \\ vt &= \beta 2 * vt-1+(1-\beta 2)*g*g \\ \theta &= \theta - \frac{lrt*mt}{sqrt(vt)+epsilon} \end{aligned} \quad (1.3)$$

D. Convolutional Neural Networks

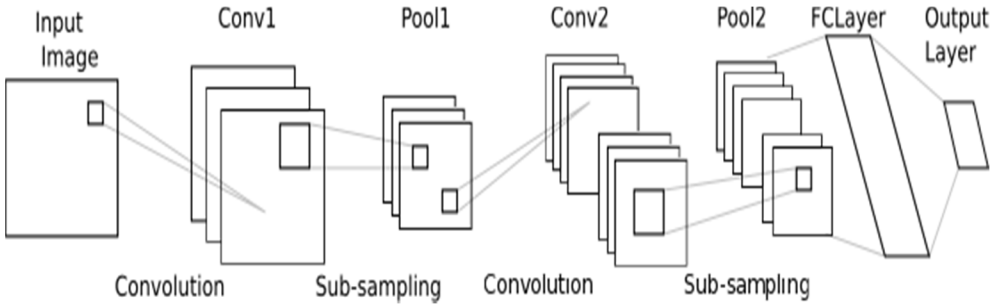


Figure1.3An example of Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of neural network in which convolution operation is performed between the fed input and kernels at a particular layer. It is followed by a non-linear activation unit, whose output becomes the input for next layer. An example of convolution network is shown in Figure-3.1.

From the Latin convolvere, "to convolve" means to roll together. Mathematical, convolution is the integral measuring how much two functions overlap as one passes over the other. Convolution is a way of mixing two functions by multiplying them. With image analysis, one function is the input image being analyzed, and the second, function is known as the filter, because it picks up a signal or feature in the image. The two functions relate through multiplication.

II. RESEARCH OBJECTIVES

The goal of the research is to design, develop, analyze and evaluate software implementation technique of deep neural networks, with the aim of achieving considerably better speed and energy efficiency than those can be realized with standard implementation technique. Thescopeofthisresearchworkisasfollows:

1) Proposing a novel design to implement a generic algorithm which can be applied todifferent types of neural network like CNN, Vanilla RNN and LSTM.
2) Emulate the implementation both in software and hardware so that fare comparison can be made to observe improvement if any. Validating the concept by using different datasets and standard neural network model

III. RESEARCH METHODOLOGY

This provides a discussion of the methodology used in this work. It begins with explaining the functioning of traditional Sequential Back propagation Algorithm. Here we will see the bottleneck in this approach and how to overcome this bottle neck. We then bring the concept of module through which we can exploit efficient parallelism.

Next we cover the algorithm which implements proposed techniques in SystemC and multi processor platform before we show why this technique works and can be used in a generic way. It elaborates the experimental setup used to implement naive and DAPP architecture. It also covers various model, its structure and parametric configuration considered. It covers details of dataset and libraries used in this work and provide mathematical analysis of expected results.

A. Unrolling of Backward Path

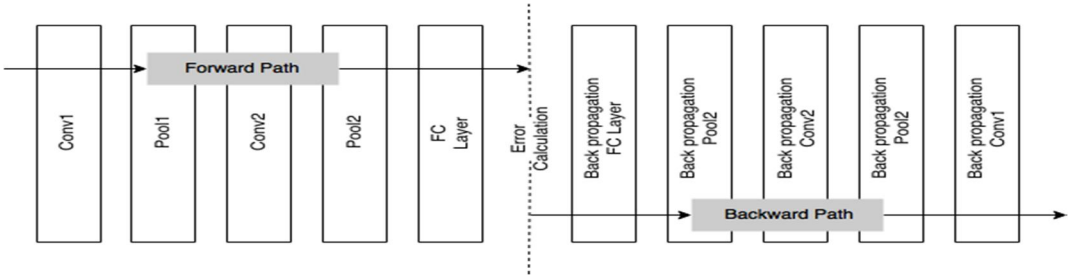


Figure3.3UnrollingtheBackwardPath

The stalls in Sequential Back-propagation Approach can be removed by unrolling the pipeline and allowing all the layers to work simultaneously on every clock cycle. Figure-3.3 depicts the visualization of unrolled pipeline.

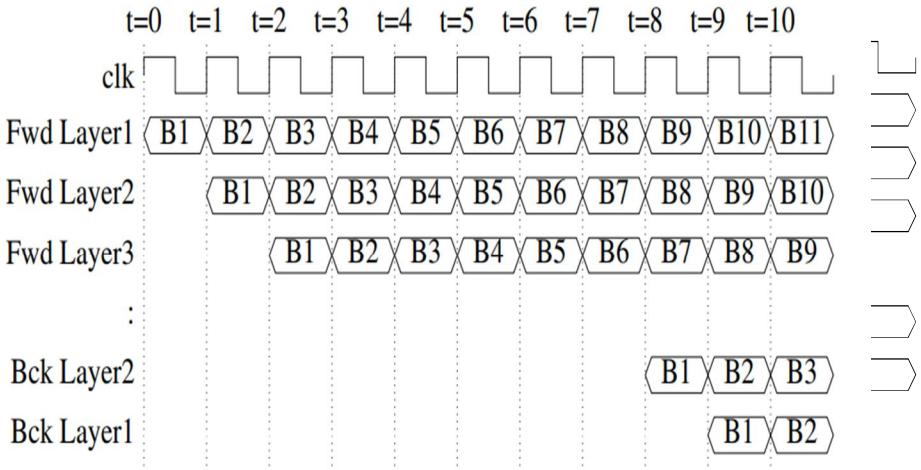


Figure 3.4 Timing Diagram for Unrolled Pipeline for MNIST (CNN)

Here by unrolling of pipeline we mean connectcorrect sequence of backward path at end of forward path as depicted in Figure-3.4. Now the length of the pipeline is doubled then what it was earlier. At beginning, the pipeline is empty and random weights are initialized in global memory. These weights are fetched by layers of forward path for processing. Every clock cycle, the batches move forward in pipeline and new batch is fed to pipeline at layer 1. These batches use initialized weights till the pipeline is filled. Once the pipeline is filled, all the gradients corresponding to first batch are computed. The gradients corresponding to batch B1 are used to update the global weights. Updated global weights are used by batch entering the pipeline in the next clock cycle. Post this, global weights are updates on every clock cycle.

Unlike sequential back-propagation, this technique does not use updated weights of immediate previous batch but delayed by few cycles which is proportional to the length of pipeline. Thus, introducing delay in update of weights, allows the layers to work in parallel on consecutive set of batches.

The timing diagram for unrolled architecture is shown in Figure-3.4. From clock cycles 0 to 9, the layers work with initialized weights. After 9th clock cycle, the weights are updated and the next batch enters the pipeline, i.e Batch 11 (B11) is processed using the updated weights of B1. In next clock cycle, B12 works using weights updated till B2.

This can begeneralized as ithbatch works with weights updated till (i − 10)th batch. If each worker, thread or computational unit, works on independent layer, then the whole process will become in-efficient. This is because, layers like pooling and dense have much less number of computations to perform as compared to Convolutional Layer. Acc to Alex[24]Convolutions take 80% of computations in ConvNets.

Table 3.1 Grouping Layers

GroupNo	LayerOps
1	Conv 1 FWD Pooling 1 FWD
2	Conv 2 FWD Pooling 2 FWD
3	DenseFWD Error Computation DenseBCK
4	Pooling2BCK Conv2BCK
5	Pooling1BCK Conv1BCK

B. Grouping Layers

We group the layers as shown in table 3.1. At first glance, one might presume that the grouping is uniform wrt the number of computations. But, the grouping discussed here is not the very efficient. This grouping is shown for easy understanding. We describe the complete process of efficient groups in Section 4.6.

C. Pre-DAPP Design

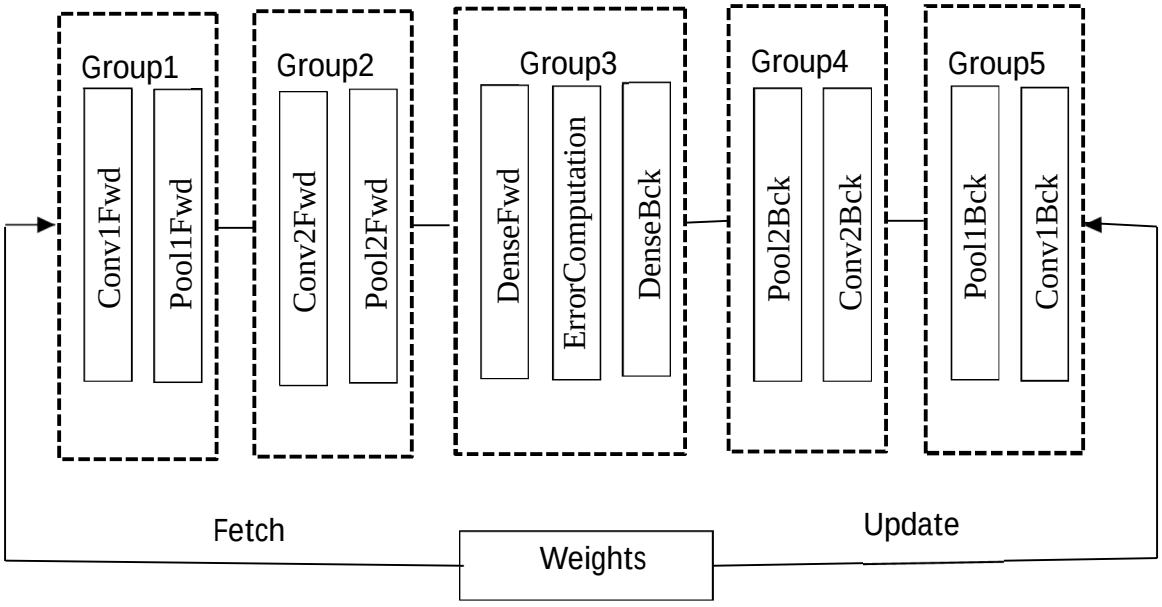


Figure3.5ADeepNeuralNetworkforCNN.Thenetworkhasbeendividedintomodules

The groups are used to exploit parallelism in the training phase using global weights. They are independent of each other. Thus, each group works on consecutive batch synchronously and simultaneously.

We define a tuple that depicts the relation between batch and group. The tuple will have m entries, m being the total numbers of groups. The index of tuple represents the group number and value at that index represents the batch on which the group is working. For example, in tuple (p, q, r, s,. . .) , first group is working on batch p, second on q and third group on batch r and group four on batchs.At time-step t, let the tuple state be (i, i-1, i-2, i-3, i-4).

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue I Jan 2023- Available at www.ijraset.com

Once all the groups finish computation on their respective batch, global weights are updated by gradients of $(i-4)^{th}$ batch. On next time-step, the tuple state is updated as (i+1, i, i-1, i-2, i-3), where $(i+1)^{th}$ batch works with weights taken from global weights which are updated till $(i-4)^{th}$ batch.

IV. RESULTS AND DISCUSSION

In this section, we will analyze the proposed design and study its performance with respect to naive implementation. To do so we assume following things:

1) There are m number of modules in a model.
2) Each module takes 1 clock cycle to complete its function. This assumption of 1 cycle is the maximum time taken of all m modules. Although all m modules should take equal time, practically the difference between any 2 values of m is negligible. Here we consider the one which is high of all m values.
3) Given any data set and number of images that data set have, we will be dividing the into n number of batches.

A. Theoretical Comparison

Here we carry out mathematical analysis with above assumption and process carried out of both Sequential BP and DAPP. Sequential Back-propagation. As bought out earlier in methodology, only one batch can be processed in pipeline at any given time. The next batch cannot be processed by pipeline till gradients are updated from previous batch is obtained. Due to this reason all stages in pipeline are idle, except one. Therefore, feedback updates for ith batch are done after every m clock cycles. The i+1th batch is fed after m clock cycles with weights updated till ith batch. Therefore,

No. of cycles to process one batch = m

Total no. of clock cycles required to = no. of batches * no. of cycles per batch process complete dataset= n * m

1) *Pre–DAPP Architecture:* In this implementation, different modules are working on different batches. Initially it takes m clock cycles to fill up the pipeline. The ith batch is fed in tth clock cycle. In the same cycle, updates are done w.r.t $(i-(m-1))$th batch in same stage. For next clock cycle, $(i+1)$th batch is fed and updates of $(i-(m-1))$th batch are obtained. Thus,
   At every cycle, no. of batch completed = 1
   Total no. of clock cycles to process = Total no. of batches= n
   Total no. of clock cycles to fill pipeline = m
   Total no. of clock cycles to process complete dataset = m + n
2) *DAPP Architecture:* In two pipeline architecture, different modules are working on different batches. Two pipeline are offset by m number of batchs. Initially it takes m clock cycles to fill up the one pipeline. Lets say ith batch is fed in tth clock cycle. In the same cycle, updates are done w.r.t $(i-(2*m-1))$th batch in same stage. This is happening for both pipelines, ie For next clock cycle, $(i+1)$th batch is fed and updates of $(i-(2*m-1))$th batch are obtained. Thus,
   At every cycle, no. of batch completed = 2
   Total no. of clock cycles to process = Half of Total no. of batches = n/2
   Total no. of clock cycles to fill pipeline = m
   Total no. of clock cycles to process complete dataset = m + n/2
3) *Summary:* Theoretically, the ideal speedup using DAPP technique with 2 pipeline is (m*n/(m + n/2)) times. Figure-6.2A shows the comparison of time for normal and DAPP approach. Since the purpose of this part of experiment is to demonstrate the speed up, we use number of clock cycles taken by model as unit of measurement. This is only used for theoretical comparison. In MNIST (CNN), the time is theoretically decreased by 66% with 3 modules.

B. Comparison for Simulation in SystemC

Simulation of DAPP for MNIST (CNN) shows time reduction of 60.4%. The no. of clock cycles given are 60,000 for sequential back-propagation and 6,006 for DAPP.

C. Comparison for Multi-core

This section provides comparison for Sequential Back-propagation Algorithm and DAPP design in Multi-core environment. Both the algorithms expoit the multi-cores.

Sequential Back Results of various networks which have been implemented using DAPP methodology in multicore are shown in Table-4.1.

Table 4.1 Acceleration using DAPP

Model	Dataset	Accuracy(Normal)	Accuracy(DAPP)	DecreaseinTime(%)
CNN	MNIST	96.75	96.61	52.3%
CNN	CIFAR	86.97	86.54	92.95%
VanillaRNN	MNIST	97.04	97.02	39.73%
LSTM	MNIST	96.4	96.7	38.45%

Test Accuracy graph for various models and two methods are shown in Figure 4.1. All these graphs bring out the difference in the accuracy between sequential and DAPP methodology. X-axis is number of iterations and Y-axis is accuracy scaled between 0 to 1. In MNIST (CNN), the batch size is 1, the update of weights are delayed by 3 batches,i.e 3 images. This has negligible effect throughout the training process. This can be seen from Graph 1 of Figure 4.1 as the two models are overlapping. In MNIST (RNN), the batch size is 100. With 3 modules, the weights are delayed by 3 batches, i.e 300 images which has significant effect in starting phase of training. As the number of iterations increase, both the models converge and depict same behavior. For MNIST (LSTM), the batch size is 50. At initial phase of training of LSTM, unlike RNN, the difference between Sequential and DAPP is less. Thus, in DAPP design, batch size has considerable effect at initial phase of training. As the iterations increase, the effect becomes negligible.Figure-4.2 shows the comparison of theoretical and empirical speed-up achieved in DAPP. It can be concluded from Figure 4.2 that empirical results follow the same pattern of the theoretical results

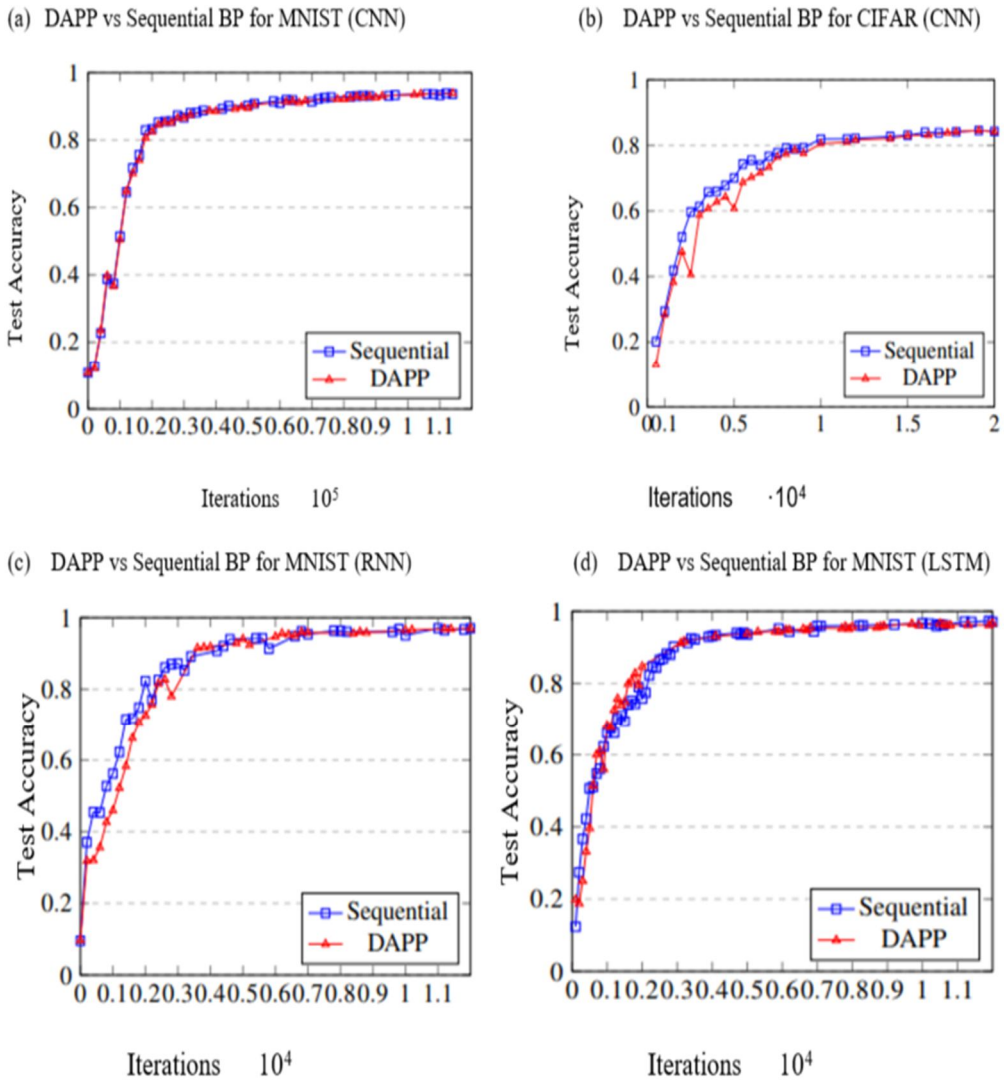


Figure 4.1 Accuracy vs. Iteration for Sequential BP and DAPP

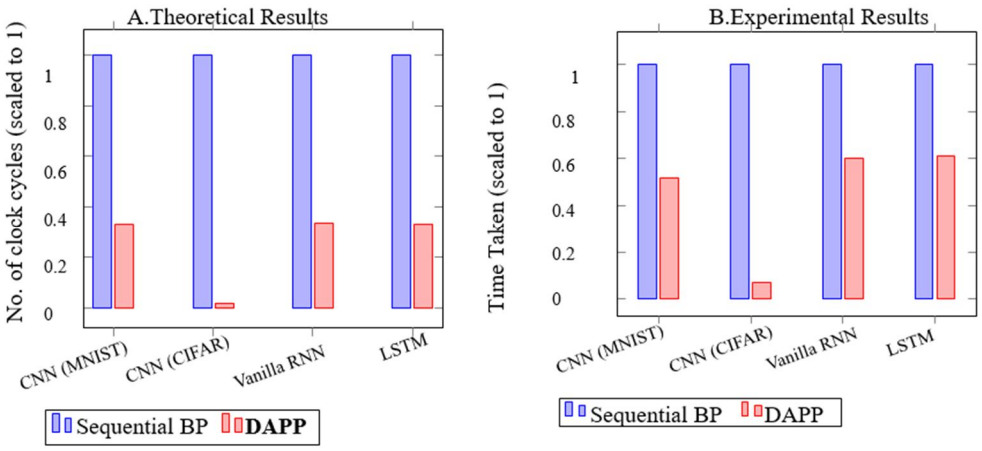


Figure 4.2 Speed-up Achieved

V. CONCLUSION

1) To conclude, we introduce a practical design flow for deep learning where have proposed a synchronous technique called DAPP, which accelerate the learning of Neural Network. When introducing the design flow we show how this speed up DNN training limitations of parallel Synchronous SGD has been removed by eliminating the need for global memory.

2) DAPP design can be implemented in for both BP and BPTT algorithm making this generic implementation. Although we have referred proof that this works for both, we demonstrate this, by implementing DAPP on different CNN model which has different depth along with Vanilla RNN and LSTM networks.

3) This approach carries the data in such a way that parallel workers keep gradients in local memory which requires less bandwidth and minimizes the delay caused by contention and communication. To address a few pitfalls of using synchronous technique some optimization steps are suggested along with algorithm which needs to be followed for a better result.

4) When evaluating the performance of DAPP technique, it is compared with the naive implementation of respective algorithms. Experimental results show a reduction in time by 40% for LeNet which is smaller network and 92% on Network-in-Network architecture of CNNs wherein it has 10 layers. For vanilla RNN, and LSTM class of algorithm on multi-core platform. 38% decrease in time for training basic one layer of Vanilla RNN and 40% for one layer of LSTM. Although we have not implemented this for a more deeper network like GooleNet [35], ResNet [36] we believe that this can further work on a more deeper network. In closing, we believe that we have answered our objective question by showing that DAPP can be used as practical acceleration platform for deep learning. We believe that this work is valuable in acting as a guide to future researchers who wish to continue efforts in this field and encourage the future development.

VI. FUTURE WORK

1) DAPP can further be inspected closely to improvise its implementation so as to patch its outcome of experimental performance with theoretical expectation.

2) Currently, DAPP has been implemented on Multi-core Xeon Processor and simulated using System-C. Implementation of DAPP on FPGA to make it energy and power efficiency can be exploited.

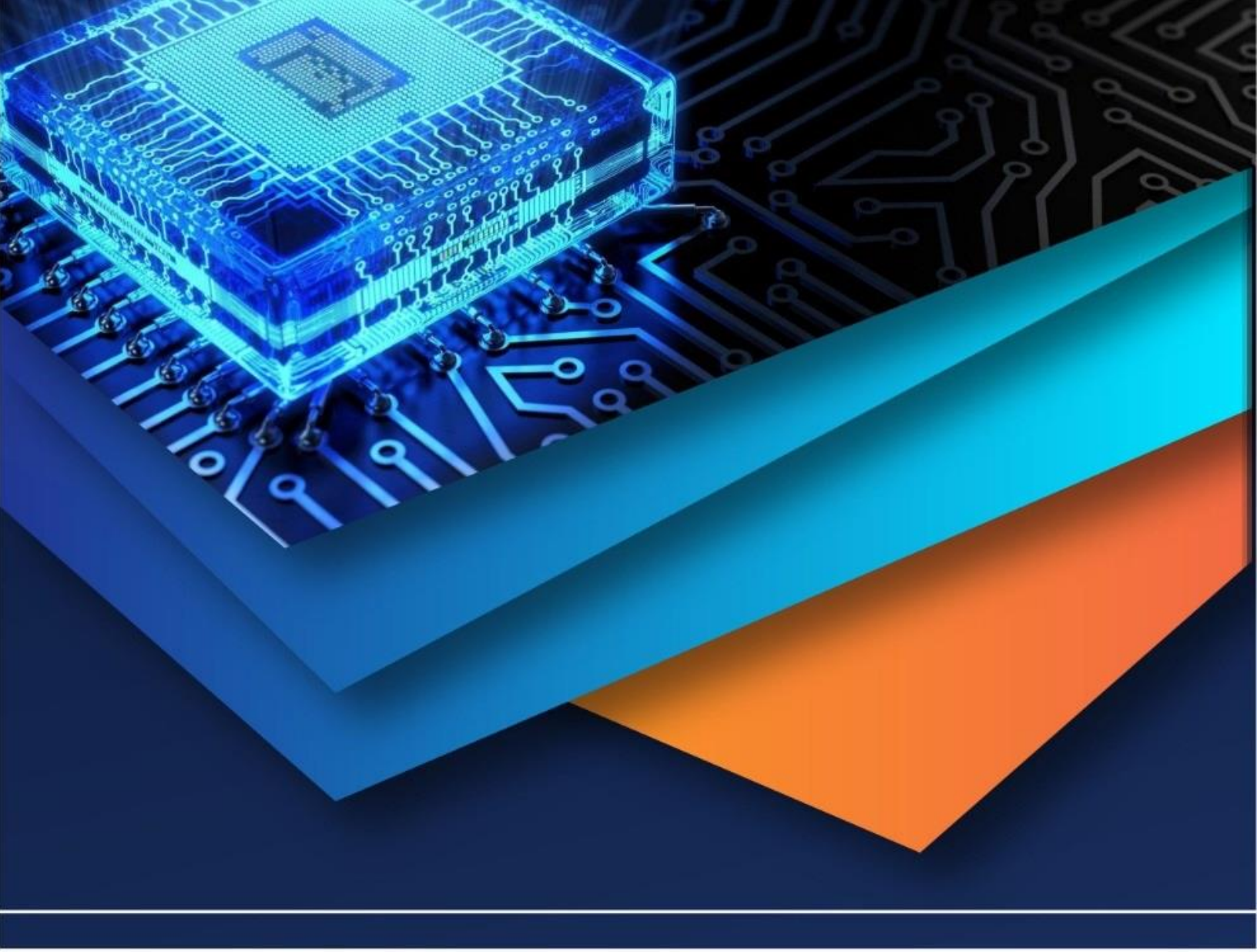
3) The multi- core implementation of DAPP can be incorporated with Tensorflow library.

4) To further accelerate the training process, the proposed approach can be incorporated with techniques like approximate computing, and quantization.

REFERENCES

[1] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.
[2] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research)
[3] SamanBashbaghi, Eric Granger, Robert Sabourin, and Mostafa Parchami.Deeplearning architectures for face recognition in video surveillance. CoRR, abs/1802.09990,2018.
[4] JahanzaibShabbir and TariqueAnwer. A survey of deep learning techniques for mobile robot applications. CoRR,abs/1803.07608,2018.
[5] Denis Tomè, Federico Monti, Luca Baroffio, Luca Bondi, Marco Tagliasacchi, andStefano Tubaro. Deep convolutional neural networks for pedestrian detection. CoRR,abs/1510.03608,2015.
[6] Thomas Paine, HaiinJin, Jianchao Yang, Zhe Lin, and Thomas S. Huang.GPUasynchronous stochastic gradient descent to speed up neural network training. CoRR,abs/1312.6186,2013.

[7] Alexander Guzhva, Sergey Dolenko, and Igor Persiantsev. Multifold acceleration ofneural network computations using GPU. In Artificial Neural Networks – ICANN 2009,pages373–380.SpringerBerlinHeidelberg,2009.
[8] KaiyuanGuo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang.A survey off pga based neural network accelerator,2017.
[9] JiantaoQiu, Jie Wang, Song Yao, KaiyuanGuo, Boxun Li, Erjin Zhou, Jincheng Yu,Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. Going deeper with embedded fpga platformfor convolutional neural network.InProceedingsofthe2016ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA'16,pages26–35,NewYork,NY,USA,2016.ACM.
[10] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, andDebbie Marr.Accelerating binarized neural networks: Comparison of FPGA, CPU,GPU, and ASIC. In 2016 International Conference on Field-Programmable Technology(FPT).IEEE,dec2016.
[11] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, Asit Mishra, Srivatsan Krishnan, and Debbie Marr. Accelerating recurrent neuralnetworksinanalyticsservers:Comparisonof FPGA, CPU, GPU, and ASIC.In 2016 26th International Conference on FieldProgrammableLogicandApplications(FPL).IEEE,aug2016.
[12] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter.A downsampled variant of image netasanalternativet othecifar datasets,2017.
[13] YoungwooYoo and Se-Young Oh.Fast training of convolutional neural networkclassifiers through extreme learning machines.In2016InternationalJointConferenceonNeuralNetworks(IJCNN).IEEE,jul2016.
[14] Yijin Guan, Zhihang Yuan, Guangyu Sun, and Jason Cong. FPGA-based acceleratorfor long short-term memory recurrent neural networks. In 2017 22nd Asia and SouthPacificDesignAutomationConference(ASP-DAC).IEEE,jan2017.
[15] SrimatChakradhar, MuruganSankaradas, VenkataJakkula, and Srihari Cadambi. Adynamically configurable coprocessor for convolutional neural networks. In Proceed-ings of the 37th Annual International Symposium on Computer Architecture, ISCA '10,pages247–257,NewYork,NY,USA,2010.ACM.
[16] OmryYadan, Keith Adams, YanivTaigman, and Marc'AurelioRanzato. Multi-gputrainingofconvnets,2013.
[17] Wojciech Marian Czarnecki, GrzegorzS´wirszcz, Max Jaderberg, Simon Osindero,OriolVinyals, and Koray Kavuk cuoglu. Understanding synthetic gradients and decoupledneuralinterfaces,2017.
[18] Lei Jimmy Ba and Rich Caurana.Do deep nets really need to be deep?CoRR,abs/1312.6184,2013.
[19] A.Nøkland.DirectFeedbackAlignmentProvidesLearninginDeepNeuralNetworks.ArXive-prints,September2016.
[20] Tim Salimans and Diederik P. Kingma.Weight normalization: A simple reparameteri-zationto accelerate training of deep neural networks. CoRR,abs/1602.07868.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY