Deep Learning on GPUs with Limited Precision

Harsh Patel[1], Tanmay Pradhan[1]
hpatel108@hawk.iit.edu,tpradhan1@hawk.iit.edu

[1]Department of Computer Science, Illinois Institute of Technology, Chicago IL, USA

***Abstract*— Deep learning is a subfield of machine learning that deals with algorithms inspired by structure and function of the brain called artificial neural networks. A deep neural network is a subset of artificial neural network. Deep neural networks have hidden layers between the input and output layers. The extra layers enable composition of features from lower layers, potentially modeling complex data. In recent years, researchers have found running these networks on GPUs would give greater performance. Due to this deep learning on GPUs is a booming research topic in machine learning field. Recent research suggest that neural networks implemented on GPUs have identical performance even if we vary precision. Nvidia has started manufacturing specialized hardware for GPUs with limited precision. In this project we implemented a deep neural network on GPUs and analyzing their performance by varying the neural networks data type as well as their learning models by comparing their precision and recall statistics.**

Keywords-Neural Network,Deep Learning,CUDA,GPU

1.INTRODUCTION

Neural network (NN) is one of the object recognition methods including character recognition.NN research has been done since the 1940's, and based on learning methods such as pre-training and RBM recently proposed and generation model The learning of multilayered NN became possible.The machine learning methodology using this multilayered NN is called Deep Learning.Since it is used for search engines and speech recognition etc. In the future automatic operation of cars It is also expected to be applied to improve efficiency of work of technological and industrial robots etc. However, Deep Learning requires enormous learning, securing a large amount of learning data and the length of learning time are problematic For this reason, in Deep Learning, GPUs that can process a large amount of data in parallel have been used, but in recent years, in order to reduce power consumption and latency, It is progressing also dedicated hardware of research that, GPU,FPGA hardware with nearly half the power consumption at the same level of recognition performance and speed has been realized.

1.1 Outline of Neural Network and Deep Learning

NN is a model imitating the mechanism of the neural circuit of the animal's brain, has been undergoing research since the occurred due to the discovery of the propagation method.However, this boom ended in the latter half of the 1990s due to the following three reasons: First, learning by the error back propagation method can be done well with three layers of NN However, if more layers are added, it can not be learned well and it learns excessively. Secondly, how to select the number of layers and the number of units is difficult compared to other machine learning, Three, the computer's ability at the time handled realistic problems.It can be said that it was not able to handle the network of the scale that can be obtained.Then, by proposing the learning method called pre-training in 2006, it becomes possible to learn multilayered NN (Deep Learning), and in 2012, In the ILSVRC, the Deep Learning method won the victory with a great difference in the conventional method.NN structure has various kinds such as multilayer perceptron, convolution neural network, etc. In this chapter, it is easy to explain the structure and learning of the neural network explain.

1.2 Neuron Model

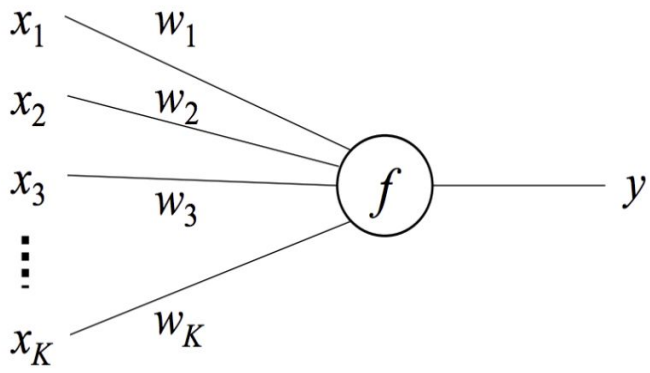


Figure 1. Neuron model

The neuron model is a unit imitating a neuron which is a neural circuit of the brain. As shown in Fig.1, the activation function is applied to the sum of the sum of the input and the weight and the bias and the output

$$y = f(\Sigma\ w_i x_i + b) \quad (1)$$

The activation function includes,

Logistic function $f(x) =$ $1/1+\exp(-x)$ (2)

Hyperbolic tangent function f(x)=tanh(x) (3)

Truncation linear function f(x)=max(0,x) (4)

In recent years,censored linear functions are generally used because learning can proceed efficiently.

1.3 Multi-Layer Perceptron

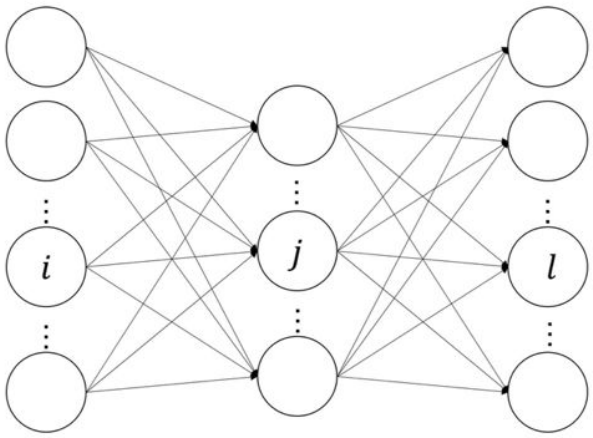


Figure 2:MultiLayer Perceptron

There are several kinds of multilayered neural networks depending on the configuration.In this research MLP and CNN are used.MLP is a nonlinear multi class classifier, and as shown in Figure 2, units are hierarchically arranged and all Unit is composed of an input layer, an intermediate layer, and an output layer, and the ith output hki of the k-th layer is represented by a vector

$$h_i^k = f(b_i^k + \boldsymbol{w}_i^{kT}\boldsymbol{h}^{k-1}) \quad (5)$$

In the output layer, softmax function

$$p_i = softmax_i(w_i x_i + b_i) = \frac{\exp(w_i x_i + b_i)}{\sum_j \exp(w_j x_j + b_j)} \quad (6)$$

The output is the probability that the input belongs to that class, and the class of the unit giving the maximum value in the output layer is the input class.

1.4 Convolutional Neural Network Convolutional Neural network

The CNN has the structure of the entire binding layer after repeating the convolution layer and the pooling layer.The feature quantity is automatically extracted by repeating the convolution layer and the pooling layer.In the convolution layer, the weight filter is convoluted with the input image In the pooling layer, we discard extra information by lowering the resolution of the obtained feature map, then we connect to the whole coupling layer and use the softmax function in the output layer.

1.4.1 Convolution Layer

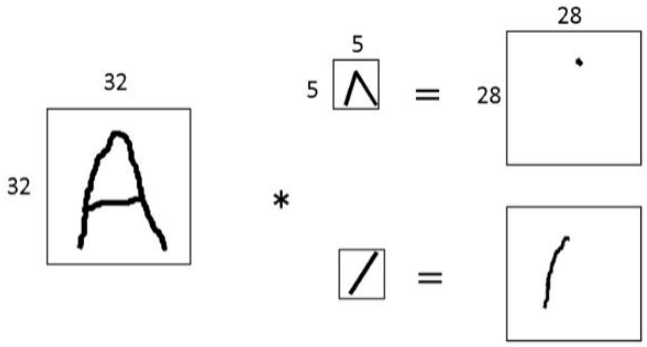


Figure 3:Convolution processing example

In the convolution layer, as shown in Figure 3, the process of convolving the filter with the input image is performed to obtain an output image called a feature map.When the image and filter size are $n_x \times n_y$,$n_w \times n_w$ respectively, the feature map is

$$n' = n_x - n_w + 1, n'_y = n_y - n_w + 1$$

1.4.2 Pooling layer

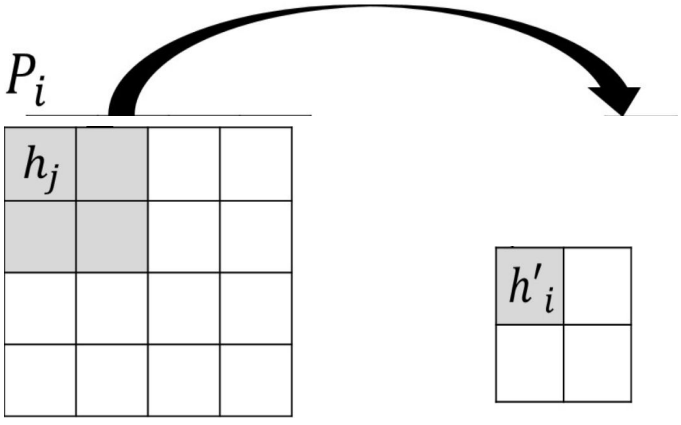


Figure 4:Pooling processing example

Pooling is to discard extra information in recognition from extracted features and convert it into a new expression that keeps the information necessary for recognition.In this case, as shown in Figure 4, by reducing the resolution of the image, There are several kinds of pooling, and average pooling, max pooling, Lp pooling, etc. exist.

$$h'_i = \frac{1}{|P_i|} \sum_{j \in P_i} h_j \qquad (7)$$

Max Pooling $$h'_i = \max_{j \in P_i} h_j \qquad (8)$$

Lp Pooling $$h'_i = \left(\frac{1}{|P_i|} \sum_{j \in P_i} h_j^P\right)^{\frac{1}{P}} \qquad (9)$$

Average pooling is the difference between the data in the small region Pi in k - 1 layer.
The average is taken as the value of neuron i in k layer.Max pooling is the maximum value of data in small region Pi in k - 1 layer as the value of neuron i in k layer As an intermediate method between average pooling and max pooling, there is the Lp pooling of equation (9).

2.Deep Learning

Deep learning is a machine learning method using multi layered NNs.Deep Learning learning is a method of adjusting the parameters (weights, biases) of units and approximating the output when inputting training data to the desired output The difference between the desired output and the actual output is the cross entropy

$$C = \Sigma di \log pi \qquad (10)$$

Where di is the teacher data and only one takes 1 and everything else becomes 0. This C is learned by adjusting the parameter and adjusting it.It is called the gradient descent method Algorithm is used.

2.1 Gradient descent method

The gradient descent method is an algorithm that searches for a local minimum from the slope of a function.The gradient related to the parameter of cost C is calculated as shown in the following formula and parameters are continuously updated.

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

$$\Delta w_{ij} = -\epsilon \frac{\partial C}{\partial w_{ij}} \qquad (11)$$

ε is called the learning coefficient and determines the width of learning.When it is too large, divergence occurs,If it is too small, the number of iterations increases and the progress of learning slows down.

2.2 Error back propagation method

In the gradient descent method, it is necessary to calculate the gradient ∂C/∂wij for the parameter of cost C.C is defined for the final output of NN as in equation (10), Figure 5.

The gradient for this output layer weight wij is

$$\frac{\partial C}{\partial w_{ij}} = (p_i - d_i) h_j \qquad (12)$$

The output of each layer is a function of the output of the immediately preceding layer, and each.The parameter becomes a nest of activation function like f (f (f ·)) and it is difficult to calculate. Therefore, error back propagation method is used. As shown in Figure 2.6, the three layers Indexes of the units are denoted by l, i, j in order from the top, for the sake of simplicity the bias is omitted and the input to unit i.

$$x_i = \sum_j w_{ij} h_j \qquad (13)$$

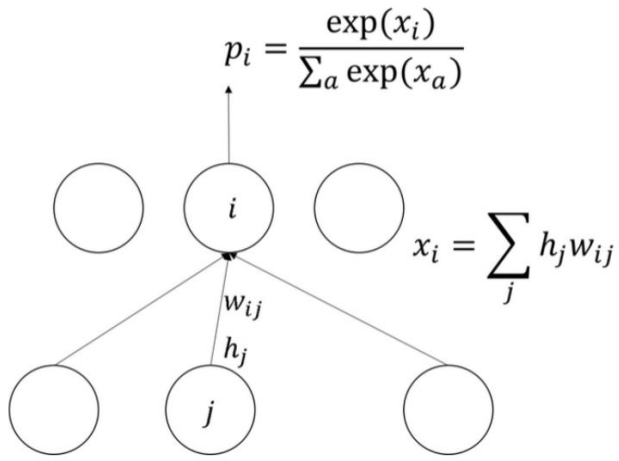


Figure 5:Calculated at the output layer of backpropagation

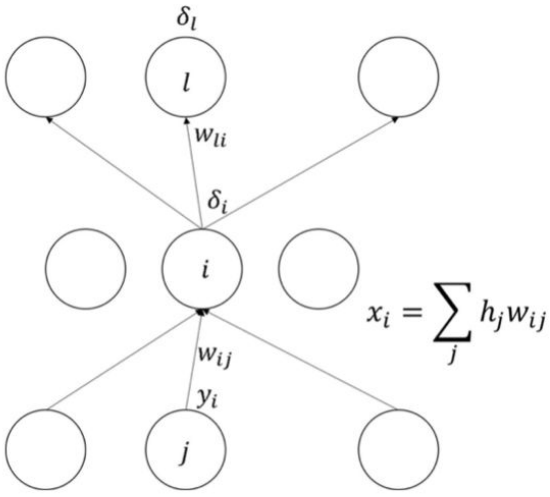


Figure 6:Calculation of an intermediate layer of backpropagation

In this case, in Figure 2.6, the gradient ∂C/∂wij is calculated by the chain sequence method of differentiation.
From the law

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial x_i}\frac{\partial x_i}{\partial w_{ij}} \quad (14)$$

Here, the first term on this right side is denoted by $\delta \equiv \partial C/\partial x_i$, and the second term is $x = \Sigma_j w_{ij} h_j$
From the relationship

$$\frac{\partial x_i}{\partial w_{ij}} = h_j \quad (15)$$

The input x to the unit l of the top layer is $x = \Sigma_j w_{ij} h_j$
Also given by $\Sigma_j w_i f(x)$, C is a function of each x (1 = 1, 2, ...)
It can also be seen that it is a function of δi

$$\delta_i = \frac{\partial C}{\partial x_i} = \sum_l \frac{\partial C}{\partial x_l}\frac{\partial x_l}{\partial x_i} \quad (16)$$

The first term on this right side is denoted by $\delta \equiv \partial C/\partial x$. The second term is $x = \Sigma_j w_i f(x)$
From the relationship

$$\frac{\partial x_l}{\partial x_i} = f'(x_i) w_{li} \quad (17)$$

As a summary of the above calculation

$$\delta_i = f'(x_i) \sum_l \delta_l w_{li} \quad (18)$$

Given the δl of the top layer of Figure 6 from this equation, δi can be calculated. If the uppermost layer in Figure 6 is the output layer,Δl of knit can be calculated by $\delta l = p_l - d_l$ Refix index of equation (18).When used in order from the output layer to the input layer, δi is calculated for all layers.If δi is obtained, from Equation (14), the desired gradient ∂C/∂wij is

$$\frac{\partial C}{\partial w_{ij}} = \delta_i h_j \quad (19)$$

Considering δi as an error, the above calculation corresponds to calculating the error δi from the output layer to the input layer in the reverse direction, which is called the error back propagation method.

3.Related Work

As GPU manufacturers are increasingly making GPUs that have specific hardware to compute fixed points and floating point calculations, there are a lot of papers available that study neural networks like Karol Gregor *et al.,* 2015, Gupta *et al*., 2015, Baboulin M. *et al*., 2009. As we have implemented the neural network using Theano and Pylearn2, there is a plethora of material which study them like Goodfellow *et al*., 2013, Bastien F. *et al.*, 2011. Out of all these works Goodfellow *et al* is a detailed explanation about Pylearn2 library and all of the components of the library like training models, datasets etc. Gupta *et al*. talks more about limited precision arithmetic and its effect on neural network and is a base for our paper. Goodfellow *et al.,* 2014 discusses image recognition using the same convoluted neural networks. The same paper also trains and test their neural network on SVHN dataset, which acts as our inspiration for conducting experiments on SVHN dataset. Ciresan *et al.,* 2012 talks about using neural network for image and pattern recognition and uses the same datasets which we have used in this paper.

4 Proposed Solution

We implemented a simple four layer neural network using Theano and Pylearn2 that can be interfaced with various datasets. We have tested and trained this neural network on MNIST, CIFAR 10 and SVHN datasets. Information of each dataset can be found in the evaluation section.

4.1 Neural Network Implementation

NN depends on the idea of the workings of the human cerebrum. There are a wide range of sorts of NN, with the more well known being a multilayer perceptron (MLP), learning vector quantization, outspread premise capacity, Hopfiled, and self-arranging map.The current examination concentrates on actualizing the test phase of the MLP utilizing a CUDA. The MLP comprises of one input layer, one output layer, and at least one than hidden layer.The mechanism for general computation for adjacent layers consists of two steps a) inner product operation between weights and input vectors of each layer (logistic regressions) and b) activation function (tanh(dot(input,W) + b) (20) where W is the weight matrix). This general operation is continually performed from the first hidden layer to the output layer. These operations can be easily applied to another neural network, MLP is more appropriate for implementation on GPUs Since another NN is additionally figured by the general operation, the internal creation operation and enact work, this operation can be effortlessly connected to another NN. In addition, since numerous inward item operations can be supplanted with a framework duplication, the MLP is more proper for CUDA usage. In that capacity, the calculation per-layer can be composed as takes after. where M is the quantity of hubs in the present layer, N is the quantity of hubs in the lower layer, and xij is the ith include estimation of the jth input vector. The outcome Rij is the yield of the ith yield hub for the jth input vector. Here, the subscript 0 implies the predisposition term, and this is to make one framework duplication without the summation term in Eq 1. While actualizing the NN utilizing CUDA, all information highlight information for the NN can't be moved into the recollections of the GPU, because of the restricted recollections of the GPU. In this way, the proposed engineering isolates the entire procedure into two sections. The initial segment is to make a reasonable size of highlight

information for the memory of the GPU, and can likewise incorporate an element extraction advance to remove highlights for the NN.

$$\mathcal{W} = \begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1N} \\ w_{20} & w_{21} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M0} & w_{M1} & \cdots & w_{MN} \end{bmatrix} = \begin{bmatrix} \mathcal{W}_1 \\ \mathcal{W}_2 \\ \vdots \\ \mathcal{W}_M \end{bmatrix},$$

$$\mathcal{X} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_{11} & x_{12} & \cdots & x_{1L} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{NL} \end{bmatrix} = [\mathcal{X}_1 \quad \mathcal{X}_2 \quad \cdots \quad \mathcal{X}_L],$$

$$M = \mathcal{W} \times \mathcal{X} = \begin{bmatrix} \mathcal{W}_1 \cdot \mathcal{X}_1 & \mathcal{W}_1 \cdot \mathcal{X}_2 & \cdots & \mathcal{W}_1 \cdot \mathcal{X}_N \\ \mathcal{W}_2 \cdot \mathcal{X}_1 & \mathcal{W}_2 \cdot \mathcal{X}_2 & \cdots & \mathcal{W}_2 \cdot \mathcal{X}_N \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{W}_M \cdot \mathcal{X}_1 & \mathcal{W}_M \cdot \mathcal{X}_2 & \cdots & \mathcal{W}_M \cdot \mathcal{X}_N \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1L} \\ m_{21} & m_{22} & \cdots & m_{2L} \\ \vdots & \vdots & \ddots & \vdots \\ m_{M1} & m_{M2} & \cdots & m_{ML} \end{bmatrix}.$$

$$R = \text{sigmoid}(M) = \begin{bmatrix} 1+e^{-m_{11}} & 1+e^{-m_{12}} & \cdots & 1+e^{-m_{1L}} \\ 1+e^{-m_{21}} & 1+e^{-m_{22}} & \cdots & 1+e^{-m_{2L}} \\ \vdots & \vdots & \ddots & \vdots \\ 1+e^{-m_{M1}} & 1+e^{-m_{M2}} & \cdots & 1+e^{-m_{ML}} \end{bmatrix}$$

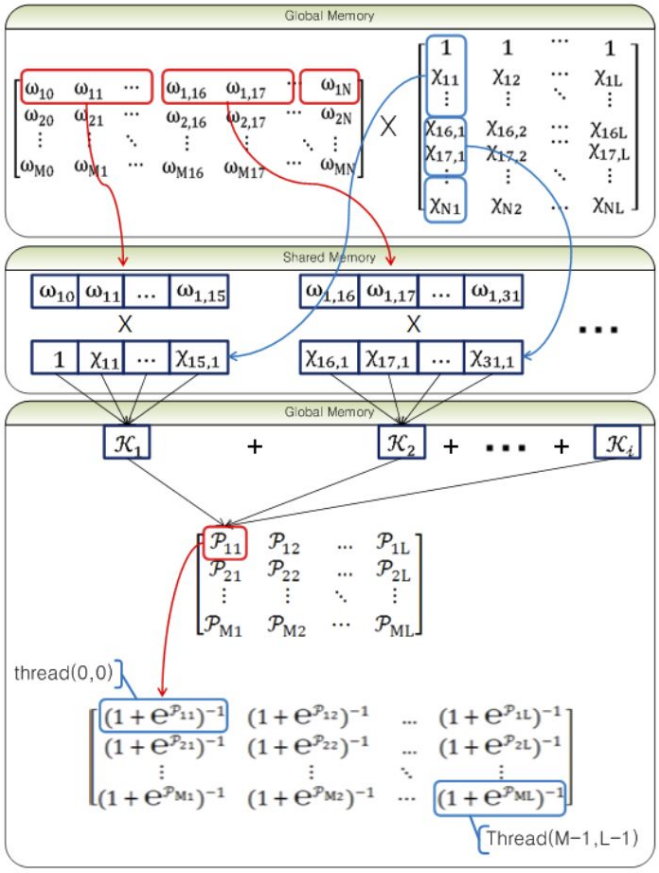
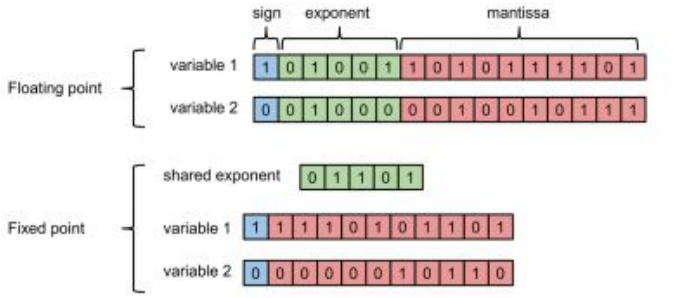


Figure 7:Operations of NN using CUDA

Fig 7 indicates lattice increase and calculation of sigmoid capacity utilizing CUDA. Since the CUDA can adequately figure framework duplication by utilizing shared recollections. By and large operation in GPU, around 400-600 cycles are required to survey the worldwide recollections, yet in memory condition of CUDA, just 4 cycles is required to get to the mutual recollections. In this manner, the mutual recollections of the CUDA help to successfully figure the operations. The sigmoid capacity can be performed in parallel by designating string equivalent to the quantity of components of network and afterward register the operation in each string autonomously.

4.2 Floating Point

Floating points are used to represent real values. They have a sign, a exponent and a mantissa. The exponent gives the float point formats a wide range and the mantissa gives good precision. Floats use 32 bits for internal calculation. Similar to floating point we have Half Floating Point (HFLP) which use 16 bits for internal calculations.



4.3 Fixed Point

Fixed point formats consists of a signed mantissa and a global scaling that is shared between all variables. The scaling factors can be seen as a position of radix point. It is usually fixed hence the name Fixed Point.. Fixed point relies on integers for internal computations and it is hardware wise cheaper as the exponent is fixed and shared between all variables.

4.4 Dynamic Fixed Point

In NN activation functions, gradients, parameters have different ranges and these ranges slowly change with training and testing the NN, hence fixed point format is not less suitable for deep learning neural networks. The dynamic fixed point format is a modified version of fixed point format in which there are various scaling factors instead of a single global one. In dynamic fixed point, each number is represented as follows: (−1)s · 2 −f l · PB−2 i=0 2 i · xi . Here B denotes the bit-width, s the sign bit, fl is the fractional length, and x the mantissa bits. The intermediate values in a network have different ranges. Therefore it is desirable to assign fixed point numbers into groups with constant fl, such that the number of bits allocated to the fractional part is constant within that group. Each network layer is split into two groups: one for the layer outputs, one for the layer weights. This allows to better cover the dynamic range of both layer outputs and weights, as weights are normally significantly smaller. On the hardware

side, it is possible to realize dynamic fixed point arithmetic using bit shifters. Dynamic fixed point format is costlier than fixed point but gives higher precision and is more desirable for neural networks.

5. Evaluation

This section describes the evaluation parameters, systems used to test our NN implementation and comparisons of the effect of changing precision on NN output. The table below shows minimum test error we got on different datasets for varying precision.

Dataset	64bit	32bit	16bit	8bit
MNIST	0.56%	0.58%	0.6%	0.58%
CIFAR10	13.88%	13.98%	14.26%	14.44%
SVHN	3.12%	2.98%	3.5%	3.95%

5.1 Hardware

We have used ChameleonCloud's baremetal nodes having P100 GPUs to evaluate our network. These nodes have 2 Intel(R) Xeon(R) CPU E5-2670 v3 CPU with 2.30 GHz of clock speed each and a total 48 threads. They 128GB of system RAM and have TESLA's P100 GPUs.

5.2 Operating System

We have used Ubuntu 16.1 with CUDA version 8 to execute GPU dependent code. We have used Python 2.7 to code the NN and also all other libraries are written in python. As python is the most widely used language for machine learning and also there are many open source libraries available related to neural network implementation, hence python was an ideal choice for us to implement the NN. Various other libraries were used which are listed below.

5.3 Libraries

This section gives a brief description of all the libraries used to run our code.

Theano

Theano is numerical computation library coded in python. Theano uses python package NumPy like syntax to store values and this code is compiled to run efficiently on both CPU and GPU. We edit .theanorc file in home directory to run the same code on both CPU and GPU. Theano is an open source library developed primarily by machine learning group at University of Montreal.

Pylearn2

Pylearn2 is a machine learning research library developed by LISA at Universit´e de Montr´eal. The goal of the library is to facilitate machine learning research. This means that the library has a focus on flexibility and extensibility, in order to make sure that nearly any research idea is feasible to implement in the library. The target user base is machine learning researchers. Pylearn2 is also an open source library. An important advantage of using Pylearn2 is that most of code used for pre processing and training the models for the datasets we have used here is already present as a part of the library. This reduced our workload very much.

NVidia CUDA Toolkit

Nvidia CUDA Toolkit development tool kit required to run high performance GPU accelerated applications like ours. The toolkit comprises of C/C++ compiler called as nvcc, GPU accelerated libraries, debugging tools and a run time library to deploy our applications.

Cython

Cython is a static compiler for Python which makes writing C extensions for python easy. As GPU accelerated programs can only be implemented using CUDA and CUDA in turn uses C/C++ we can easily reference C functions in our python code by using cython compiler.

5.4 System Parameters

We had to set some paths to set environment variables for CUDA to work. We are also required to set paths for datasets for Pylearn2 library. The following commands will set CUDA paths:

```
export PATH=/usr/local/cuda-8.0/bin${PATH:+:${PATH}}
export
LD_LIBRARY_PATH=/usr/local/cuda-8.0/lib64${LD_LIBR
ARY_PATH:+:${LD_LIBRARY_PATH}}
```

Following commands will set paths for Pylearn2 library dataset:

```
export
PYLEARN2_DATA_PATH=/home/cc/pylearn2/pylearn2/data
sets
export SVHN_LOCAL_PATH=/home/temp
```

We have to also set some parameters when we call our code. The general way of calling is given as follows:

```
python final.py [dataset] [precision] [initial range]
[propagations bit-width] [parameters update bit width] [ranges
update frequency] [maximum overflow rate] [no. of epochs]
```

Dataset: There are three options here MNIST, CIFAR10 or SVHN.

Precision: We can select Floating Point (FLP), Half Floating Point (HFLP), Fixed Point (FXP), Dynamic Fixed Point (DFXP).

Initial Range: It is used only for FXP and DFXP. It is the initial position of the radix point for fixed points. This is to be set at 5.

Propagations and Parameters update bit width: These are used only for FXP and DFXP. These are bit widths of respective propagations and parameters update.

Range Frequency: It is only used for DFXP. It is the number of batches between two range updates.

Maximum Overflow Rate: Used only for DFXP. It is the maximum overflow that can be tolerated before modifying the range.
Number of epochs: Number of epochs we train to find scaling factor.
Examples:

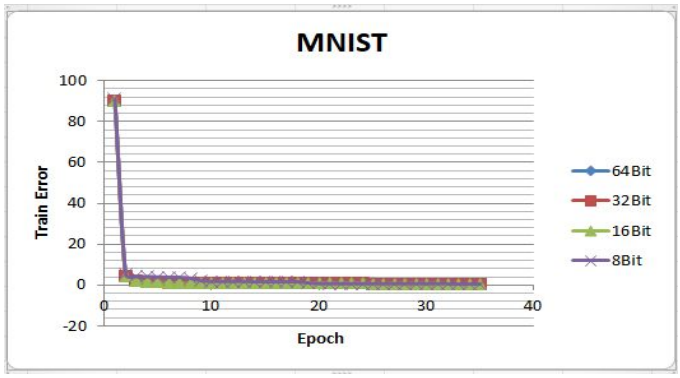
```
python final.py MNIST FLP
python final.py MNIST HFLP
python final.py CIFAR10 DFXP 5 9 11 100 0.001 2
python final.py SVHN FXP 5 19 19
```

5.5 Datasets

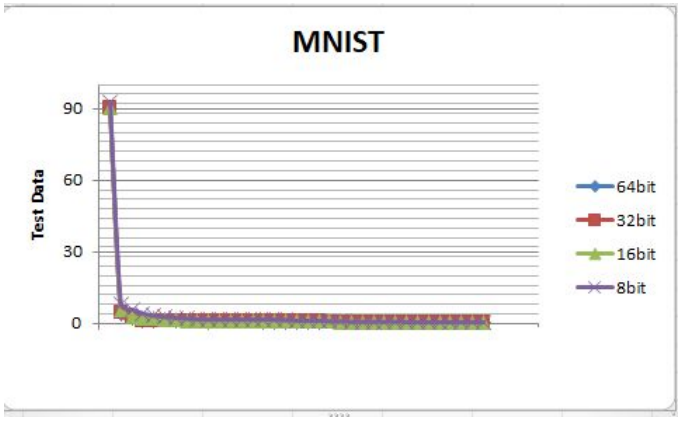
This section gives a brief description of each dataset on which we have evaluated our NN. We plot graphs for each dataset by varying precision levels and compare with each other to conclude if varying precision has an effect on performance of the neural network.

MNIST

The MNIST dataset comprises of 60,000 training images and 10,000 test images, each image is 28 x 28 pixels containing a digit from 0 to 9. The pixel values are normalized to lie in the [0,1] range. No other form of pre-processing is done on the dataset. We train this dataset on our NN which has 4 layers and having a batch size of 128. For training this network we adopt a decreasing learning rate and set it to 0.02 for epoch 1. When running on FLP we achieve a best test error of 0.58%. We train and test the dataset for other precision types too.



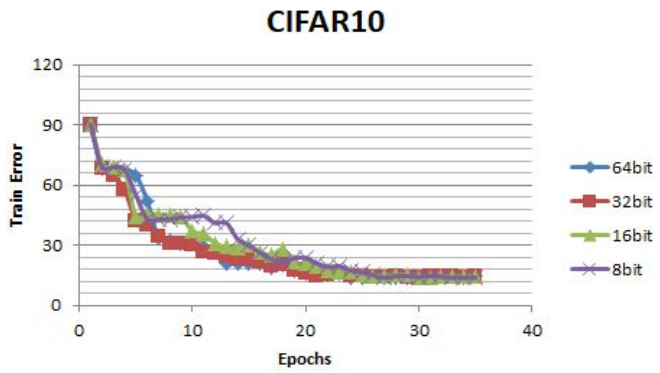
The above graph compares training error of MNIST dataset with varying precision.



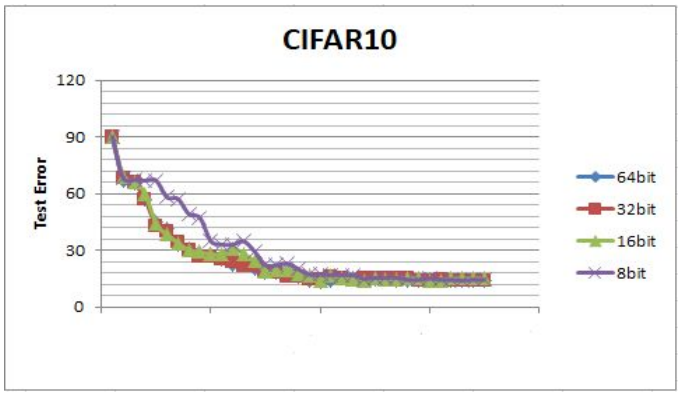
The above graph shows us the gradually falling test error for various precision

CIFAR10

CIFAR10 dataset consists of a training set of 50,000 RGB images of size 32x32 pixels. The images are divided into 10 classes each containing 5,000 images. The test set has 10,000 images. We scale the image RGB values to [0,1] range and do not perform any other preprocessing on it. We feed this dataset to our NN which consist of 4 layers having a batch size of 128. For training we adopt a decreasing learning rate and set it 0.02 for epoch 1. When running on FLP we achieve a best test error of 13.98%. We also train and test this dataset for other precision types.



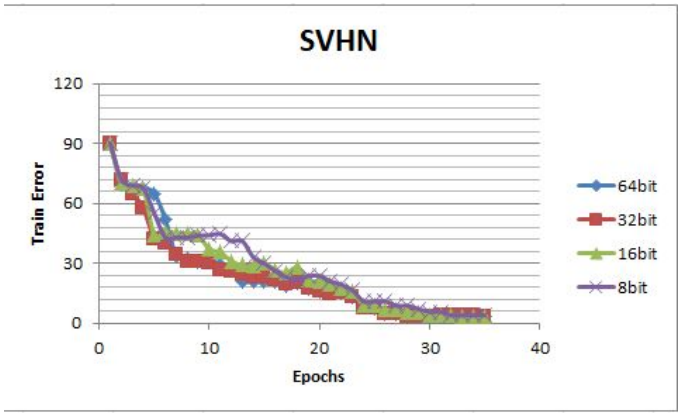
The graph above gives training error for each epoch for varying precision carried on CIFAR10 dataset.



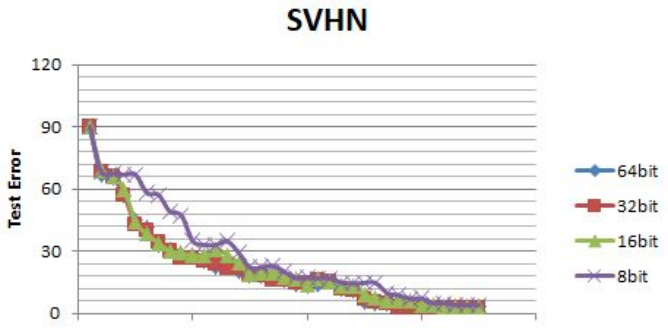
The above graph shows gradually decreasing error rate for various precision

SVHN

SVHN is a real world image dataset used for machine learning research and object recognition algorithms with minimal requirement on data preprocessing and formatting. It has 10 classes with 1 class for each digit. It consists of 73,257 digits for training 26,032 digits for testing and has an additional 5,31,131 digits that can be used as extra training data. SVHN is obtained from house number in Google Street View images.



The above graph gives train error for SVHN dataset



The above graphs gives us the test error for SVHN dataset for various precision

6. Conclusion

After carrying out these experiments and comparing results of three datasets with varying precision we can conclude that neural networks can be trained and tested with almost the same efficiency with lower precision. There is no significant improvement in performance of neural networks when using high precision. We can also conclude that out of all the formats Dynamic Fixed Point (DFXP) is the most suitable format for training neural networks on GPUs.

7. Acknowledgement

We thank the developers of Theano, Pylearn2 which were the backbone of our experiments. We also thank Prof Ioan Raicu for teaching us this wonderful course and for giving us resource to run our experiments on ChameleonCloud. Both partners contributed equally in each phase of the project right from development of NN to testing it on ChameleonCloud, making reports and preparing the presentation.

8. References

[1] K.S Kyong and K. Jung. "GPU Implementation of Neural Network", Pattern Recognition, Vol. 37, Issue 6, pp. 1311-1314, 2004.

[2] K. Moreland and E. Angel. "The FFT on a GPU", Proceedings of SIGGRAPH Conference on Graphics Hardware, pp. 112-119, 2003. [3] J. Mairal, R. Keriven, and A. Chariot. "Fast and Efficient Dense Variational Stereo on GPU", Proceedings of International Symposium on 3D Data Processing, Visualization, and Transmission, pp. 97-704, 2006.

[4] R. Yang and G. Welch. "Fast Image Segmentation and Smoothing using Commodity Graphics hardware", Journal of Graphics Tools, Vol. 17, issue 4, pp. 91-100, 2002.

[5] J. Fung and S. Man. "OpenVIDIA: Parallel GPU Computer Vision", Proceedings of ACM International Conference on Multimedia, pp. 849-852, 2005.

[6]An, G. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8(3):643–674, 1996.

[7]Audhkhasi, K., Osoba, O., and Kosko, B. Noise benefits in backpropagation and deep bidirectional pre-training. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pp. 1–8. IEEE, 2013.

[8]Baboulin, M., Buttari, A., Dongarra, J., Kurzak, J., Langou, J., Langou, J., Luszczek, P., and Tomov, S. Accelerating scientific computations with mixed precision algorithms. *Computer Physics Communications*, 180(12): 2526–2533, 2009.

[9]Bishop, C. M. Training with noise is equivalent to Tikhonov regularization. *Neural computation*, 7(1):108– 116, 1995.

[10]Bottou, L. and Bousquet, O. The tradeoffs of large scale learning. In *NIPS*, volume 4, pp. 2, 2007.

[11]Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., Li, L., Chen, T., Xu, Z., Sun, N., et al. Dadiannao: A machine-learning supercomputer. In *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pp. 609–622. IEEE, 2014.

[12]Chilimbi, T., Suzue, Y., Apacible, J., and Kalyanaraman, K. Project Adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pp. 571–582, Broomfield, CO, October 2014.

[13]Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12): 3207–3220, 2010.

[14]Coates, A., Huval, B., Wang, T., Wu, D., Catanzaro, B., and Andrew, N. Deep learning with COTS HPC systems. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 1337–1345, 2013.

[15]Courbariaux, M., Bengio, Y., and David, J.-P. Low precision arithmetic for deep learning. *arXiv preprint arXiv:1412.7024*, 2014.

[16]Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Senior, A., Tucker, P., Yang, K., Le, Q. V., et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pp. 1223–1231, 2012.

[17]Farabet, C., Martini, B., Corda, B., Akselrod, P., Culurciello, E., and LeCun, Y. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on, pp. 109– 116. IEEE, 2011.*

[18]*Gokhale, V., Jin, J., Dundar, A., Martini, B., and Culurciello, E. A 240 G-ops/s mobile coprocessor for deep neural networks. In Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on, pp. 696–701. IEEE, 2014.*

[19]Hammerstrom, D. A VLSI architecture for high-performance, low-cost, on-chip learning. In Neural Networks, 1990., 1990 IJCNN International Joint Conference on, pp. 537–544. IEEE, 1990.

[20]Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.

[21]Ho ¨hfeld, M. and Fahlman, S. E. Probabilistic rounding in neural network learning with limited precision.Neuro-computing, 4(6):291–299, 1992.

[22]Holt, J. and Hwang, J.-N. Finite precision error analysis of neural network hardware implementations. Computers, IEEE Transactions on, 42(3):281–290, 1993.

[23]Iwata, A., Yoshida, Y., Matsuda, S., Sato, Y., and Suzumura, N. An artificial neural network accelerator using general purpose 24 bit floating point digital signal processors. In Neural Networks, 1989. IJCNN., International Joint Conference on, pp. 171–175. IEEE, 1989.

[24]Kim, J., Hwang, K., and Sung, W. X1000 real-time phoneme recognition VLSI using feed-forward deep neural networks. In Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on, pp. 7510–7514. IEEE, 2014.

[25]Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems, pp. 1097–1105, 2012.

[26]Kung, H. Why systolic architectures? Computer, 15(1): 37–46, Jan 1982. doi: 10.1109/MC.1982.1653825.

[27]LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient- based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[28]Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., Jackson, B. L., Imam, N.,Guo, C., Nakamura, Y., et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. Science, 345(6197):668–673, 2014.

[29]Murray, A. F. and Edwards, P. J. Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training. Neural Networks, IEEE Transactions on, 5(5):792–802, 1994.

[30]Recht, B., Re, C., Wright, S., and Niu, F. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in Neural Information Processing Systems, pp. 693–701, 2011.

[31]Vanhoucke, V., Senior, A., and Mao, M. Z. Improving the speed of neural networks on CPUs. In Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop, 2011.

[32]Wu, R., Yan, S., Shan, Y., Dang, Q., and Sun, G. Deep image: Scaling up image recognition. arXiv preprint arXiv:1501.02876, 2015.

[33] D. Hammerstrom, "A VLSI architecture for high-performance, low cost, on-chip learning," in Proc. Int. Joint Conf. Neural Networks, San Diego, CA, June 1990, pp. II:537-543.

[34]J. L. Holt and J. N. Hwang, "Finite precision error analysis for neural network hardware implementation," in Proc. Int. Joint Conf. Neural Networks, Seattle, WA, July 1991, pp. I:519-526.

[35]J. N. Hwang, J. A. Vlontzos, and S. Y. Kung, "A systolic neural network architecture for hidden Markov models," IEEE Trans. Acoust., Speech, Signal Processing, vol. 37, pp. 1967-1979, Dec. 1989.

[36] J. N. Hwang and P. S. Lewis, "From nonlinear optimization to neural network learning," in Proc. 24th Asilomar Conf. Signals, Syst., & Comput., Pacific Grove, CA, Nov. 1990, pp. 985-98

[37]T. E. Baker, "Implementation limits for artificial neural networks," Master thesis, Dep. Comput. Sci. and Eng., Oregon Graduate Institute of Science and Technology, 1990.

[38]P. S. Lewis and J. N. Hwang, "Recursive least squares learning algorithms for neural networks," in Proc. SPIE's Int. Symp. Opt. and Optoelectron. Appl. Sci. and Eng., San Diego, CA, July 1990, pp. 28-39.

[39] J. L. Holt and T. E. Baker, "Back propagation simulations using limited precision calculations," in Proc. Int. Joint Conf. Neural Networks, Seattle, WA, July 1991, pp. II: 121-126.

[40] Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlun, Ronak Singhal, and Pradeep Dubey (2010) Debunking the 100× GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU, Proceedings of the 37th annual international symposium on Computer architecture, ISCA'10, ACM.

[41] Noriyuki Fujimoto (2008) Faster Matrix-Vector Multiplication on GeForce 8800GTX, Proceedings of the 22nd IEEE International Parallel and Distributed Processing Symposium (IPDPS), LSPP-402, pp. 1–8.

[42] Volodymyr Mnih (2009) CUDAMat: a CUDA-based matrix class for Python, Technical Report UTML TR 2009-004, Department of Computer Science, University of Toronto.

[43] Rajat Raina, Anand Madhavan, and Andrew Y. Ng (2009) Large-scale deep unsupervised learning using graphics processors, Proceedings of the 26th Annual International Conference on Machine Learning, ICML'09, ACM.

[44] Kyoung-Su Oh and Keechul Jung (2004) GPU implementation of neural networks Pattern Recognition, 37(6):1311-1314.

[45] Honghoon Jang, Anjin Park, and Keechul Jung (2008) Neural Network Implementation using CUDA and OpenMP, Proceedings of the 2008 Digital Image Computing: Techniques and Applications, pp 155–161.

[46] Navdeep Jaitly, Patrick Nguyen, and Vincent Vanhoucke (2012) Application of Pretrained Deep Neural Networks to Large Vocabulary Speech Recognition, Submitted to ICASSP'12