Deep learning on GPUs with Limited Precision

Navneet Goel
A20405197
ngoel2@hawk.iit.edu

Chitrarth Singh
A20387080
csingh7@hawk.iit.edu

Pradyot Mayank
A20405826
fpradyotmayank@hawk.iit.edu

ABSTRACT

Numerical operations on GPUs are challenging in terms of accuracy, precision and performance making it essential to choose the best representation for data.
We propose to create a model with limited precision data to train Deep Neural Networks on two benchmark datasets: MNIST and CIFAR. In particular, we will be using floating point data formats to enhance the performance and reduce the memory usage of neural networks running on GPUs. Based on the related work we assume that the deep neural networks with limited precision data formats is sufficient to train them with little to no error in accuracy.

BACKGROUND INFORMATION

High Precision Data formats consume relatively higher amount of memory and yield mediocre performance as opposed to limited precision data format.
The half precision (FP16) Format is not new to GPUs. In fact, FP16 has been supported as a storage format for many years on NVIDIA GPUs, mostly used for reduced precision floating point texture storage and filtering and other special-purpose operations. The Pascal GPU architecture implements general-purpose, IEEE 754 FP16 arithmetic. High performance FP16 is supported at full speed on Tesla P100 (GP100), and at lower throughput (similar to double precision) on other Pascal GPUs (GP102, GP104, and GP106).

PROBLEM STATEMENT

Applications today being data-intensive, generating huge datasets which requires faster computing and greater accuracy. CPUs and GPUs are exhausting in their potential to be at par in processing this kind of data. High Precision Data Formats, as we already know, consumes significantly huge amount of memory space to achieve higher accuracy with great precision which reduces the performance of GPUs and system throughput overall.e.g.Neural network with supervised learning are rather meant to be more efficient than power consuming.

RELATED WORK

Research paper by Yoshua *et al.*(2015) talk about training deep neural networks using low precision multiplication and achieving optimized memory usage on general purpose hardware. Jordan *et al.*

adapted limited precision calculations to train deep neural network for back propagation algorithms. Gupta *et al* used 16- bit fixed point format to train neural networks on FPGA gaining significant amount of efficiency and computation throughput.

PROPOSED SOLUTION

Exponential growth of data sets inhibits challenges in the processing, analysis and archival of data in today's real world applications, to match up with the increasing amount of data we need to upsurge the computation of our systems. One way is to reduce the high precision data to low precision which has a significant result on the performance without compromising with the accuracy of the result.

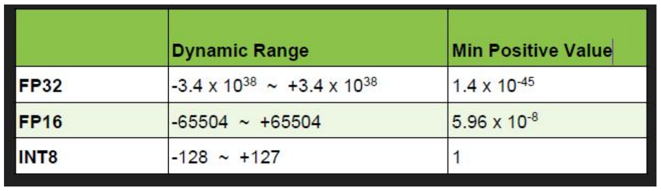
We propose to implement a deep neural network model using TensorFlow libraries on GPUs. GPUs through their multi-core architecture provides enhanced throughput to train models/neural nets in deep neural network when compared to CPUs. TensorFlow libraries automatically discovers and uses GPUs and multiple cores.Also, it supports seamless quantization of limited precision data types.

Our goal from designing to implementation and testing of model iterates through several steps. (Fig.1)

As we began with the model, first and most imperative step would be selecting the dataset which leads us to benchmark datasets: MNIST and CIFAR10. Both the datasets holds adequate amount of data to train and test our model. Using the parameters/features of the datasets our model will be calibrated to be trained. Next we develop a hypothesis function and we will find a way of measuring how well it fits into the data. Now we need to estimate the parameters that where the Gradient Descent comes into the picture.
We will know we have succeeded when our cost function is at the very bottom of the pit i.e. when its value is the minimum.
Now, the neural network/nets learns from the training data and calculates the weights on the basis of given parameters that enables it to map the input data to precise outcome. Training requires iterative forward and backward propagation through the network as the accuracy is increased and errors are minimized with respect to the network weights. The error reduction will be done through the implementation Gradient Descent algorithm. Our model will be trained and accuracy is validated against the data which were not included during the training of the model to estimate the performance of the model for real world applications.
The next step includes *inference*–through which we will be using our trained model to make predictions from new data. During this step, we will be implementing our trained model in Chameleon testbed.Our
Often several models are trained and accuracy is validated against data not seen during training in order to estimate real-world performance.
Our choice of programming language will be Python and will be using PyCharm, or Jupyter notebook to implement DNN(Deep Neural Networks).
Execution time and power are not cheap and are critical in real time applications. Given the amount of compute required we would like to optimize our inference in favor of time and power without hurting the accuracy. So, we will be moving to a lower precision representation of parameters and activation at inference time. Lower precision would need lesser cycle for memory fetch compared to higher bits. There is also one other advantage in the form of a new hardware instruction (DP4A).
We also provide a set of routines that allow users to train and evaluate complete deep neural networks without the need to write parallel code manually.

- Floating Point Format
    - We will use floating format which consist of sign, exponent and mantissa. In our experiments we set single floating point format as base format and will compare other

limited precision format as it will show little to no impact on the training of neural networks.



	Dynamic Range	Min Positive Value
FP32	$-3.4 \times 10^{38} \sim +3.4 \times 10^{38}$	1.4×10^{-45}
FP16	$-65504 \sim +65504$	5.96×10^{-8}
INT8	$-128 \sim +127$	1

During the process of training, parameters and activation are represented in FP32. FP32's high precision fares well for training as every training step corrects the parameters by small amount. DL algorithms are in general resilient to noise. During the course of learning the model, in general, tries to selectively preserve the features necessary for prediction. So, throwing out unnecessary stuff is kind of built into the process. So we hope that the noise introduced by low precision representation is thrown away by the model (again, very layman terms here). As far as I know there is no rigorous mathematical proof which says low precision should work as good as FP32 during inference.

The high level implementation of our algorithm will go as-

1. Calibrate dataset with the model.
2. Run inference in FP32 on dataset.
3. Collect required statistics.
4. Quantize FP32 weights→ limited precision(e.g. INT8)

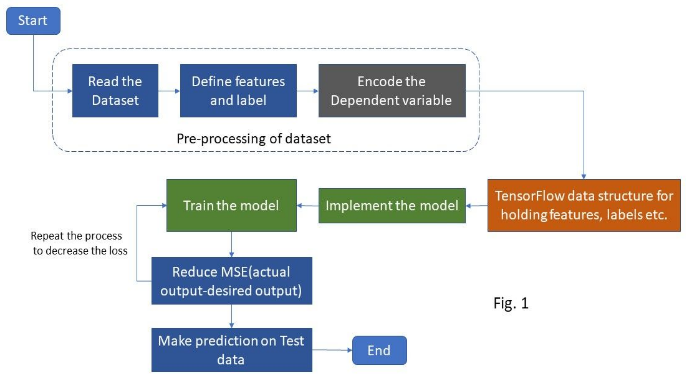

Start
Read the Dataset
Define features and label
Encode the Dependent variable
Pre-processing of dataset
TensorFlow data structure for holding features, labels etc.
Implement the model
Train the model
Repeat the process to decrease the loss
Reduce MSE(actual output-desired output)
Make prediction on Test data
End

Fig. 1

EVALUATION

We will evaluate the Memory usage, Performance across a wide range of input sizes, Accuracy of our deep neural model with limited precision formats on real datasets both training and testing. These evaluations will done on GPUs in the Chameleon testbed on a single node with one or two GPUs.

TIMELINE (WITH WEEKLY GOALS)

Week	Tasks
1	Draft the Model
2	Prototype of the model
3	Project Midterm Progress Report
4	Calibrating model with the dataset
5	Build training and test framework

6	Run test framework
7	Project final presentation
8	Project Final Reports

DELIVERABLES

- Final report in PDF format
- Final powerpoint presentation
- Source code of the implementation
- Research Paper

CONCLUSION

During implementation of our model we will get to learn the following

- How to implement deep neural networks efficiently.
- Very low precision multipliers are sufficient for training deep neural networks.
- Using a higher precision for the parameters during the implementation helps for better understanding of the model.
- Our work can be exploited to optimize memory usage on general-purpose hardware(e.g chameleon testbed with more than 1 GPU .)

Based on our observations we will show that:

- Limited precision floating formats is sufficient for training deep neural network with little to no degradation in the accuracy of the final result.
- Achieve faster computation on GPUs using these limited precision data types.
- Minimize loss of information when quantizing trained model weights to limited precision formats and during limited precision computation of activations.

REFERENCES:

Presley, R. K. and Haggard, R. L. (1994). *A fixed point implementation of the backpropagation learning algorithm*. In Southeastcon'94. Creative Technology Transfer-A Global Affair., Proceedings of the 1994 IEEE, pages 136–138. IEEE.

Holt, J. L. and Baker, T. E. (1991). *Back propagation simulations using limited precision calculations. In Neural Networks*, 1991., IJCNN-91-Seattle International Joint Conference on , volume 2, pages 121–126. IEEE
Courbariaux, M., Bengio, Y., David, J.P.: *Training deep neural networks with low precision multiplications.* arXiv preprint.

Gupta, Suyog, Agrawal, Ankur, Gopalakrishnan, Kailash, and Narayanan, Pritish. *Deep learning with limited numerical precision*. arXiv preprint arXiv:1502.02551 ,2015.

Devblogs nvidia: *Mixed-Precision Programming with CUDA 8*
Tensorflow: *Quantizing Neural Networks with TensorFlow*

Mid Term Project Progress Report

- **Proposed Solution:**
    - We have implemented a Deep Neural Network model using Tensorflow library in Python. Our model performs the numerical operation on limited precision data formats, particularly floating-point data formats, on two standard datasets: **MNIST** and **CIFAR10.**

      (**#Ref: https://www.cs.toronto.edu/~kriz/cifar.html; http://yann.lecun.com/exdb/mnist**).

        - **MNIST** database of handwritten digits of 28x28 pixels which has a training set of **60,000** examples, and a test set of **10,000** examples. It is a subset of a larger set available from **NIST**. The digits have been size-normalized and centered in a fixed-size image
        - The **CIFAR-10** dataset consists of **60000** 32x32 colour images in 10 classes, with 6000 images per class. There are **50000** training images and **10000** test images.

  We have divided our model into several modules/tasks which are as follows:

    1. Compute the Gradient of the cost function using back propagation algorithm to get the maximum accuracy and minimum loss.----**Navneet Goel**
    2. Quantize the FP32 weights to FP16 using stochastic rounding to run the limited precision floating point data formats on GPUs .------ **Chitrarth Singh**
    3. Softmax function at final layer which will squash the outputs of each unit to be between 0 and 1. But it also divides each output such that the total sum of the outputs will be equal to 1.----- **Pradyot Mayank**
    4. For activation of neuron we are using sigmoid function. The neuron will be activated when the output from the final layer will cross the threshold limit.----**Navneet, Pradyot, Chitrarth**

    - Initially we tried to quantize the weights from FP32 to FP16 by using data type FP16 but the result we got was not precise. Moreover, there was a noticeable difference between the output we got using the FP32 data formats and FP16 data formats. We realized later we need to round off the FP32 using stochastic rounding function.

- **Evaluation:**

  In the coming weeks we will run more simulations while increasing the epochs every time to achieve the maximum accuracy and minimum loss using tensorflow in python. Our final step will be to implement our model and run the datasets MNIST and CIFAR-10 with limited precision value into chameleon testbed using CUDA. Below is the graph we got for epochs vs accuracy for FP32 datasets while testing on CPU (i5-6200U @ 2.30 GHz 2.40 GHz with 2 cores).

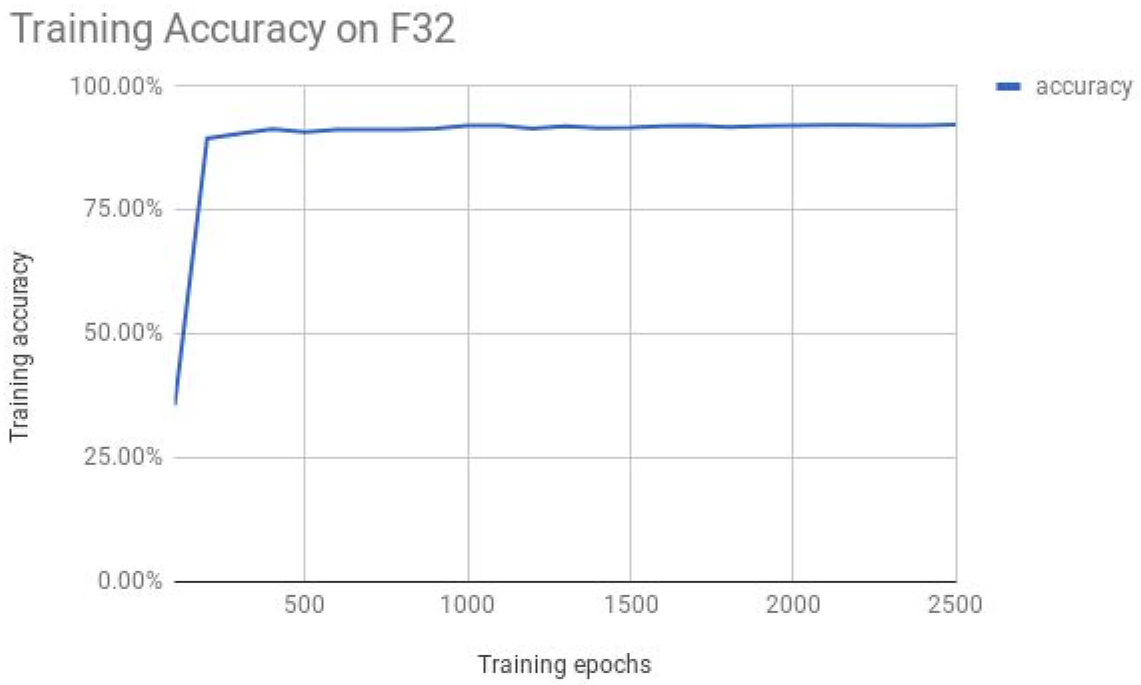


FIgure 1. MNIST dataset using Linear regression. The graph shows increase in accuracy with increase in epochs. The graph shows 92.3% accuracy for MNIST dataset on F32 data type.

Our algorithm will focus on quantizing FP32 weights to limited precision (e.g.: FP16) with the evaluation of memory usage and performance across a wide range of input sizes. Scope of our work will cover the implementation of Deep Neural Network model on chameleon testbed with more than one GPU, successfully optimizing the model to achieve faster computation, prevent degradation in the accuracy of the final result and minimize loss of information while quantizing trained model weights to limited precision. Since DNN(Deep Neural Networks) requires more training sets to learn better, we will be trying to run on more datasets if possible, so we are running behind schedule by 1-2 days. We hope to be on track by this week.