uDNN: Micro Deep Neural Networks with Variable Precision Computations

Milap Jhumkhawala
A20409159

Raphael Cohen
A20432518

Abstract

Deep learning algorithms are playing an important role in computer vision and image recognition tasks. Their applications range from self-driving cars to intelligent voice assistants. Deep learning algorithms are approximations of the target functions(actual properties), thus they cannot be truly accurate but accurate enough. From the engineering point of view, most important metrics of a DNN are speed of training and prediction, accuracy, performance and energy consumption. Recent trends in deep learning involves exploring lower precision operations for increasing speed, usability and energy efficiency without losing too much quality of prediction. If computing the model using lower precision gives us similar accuracy, that might save us a lot of training time, energy and can be compiled on various lower end processors and GPUs, thus saving dedicated resources in the process.

Background information

Deep learning models are a complicated network of layers of neurons. The more complex a problem gets, deeper the model needs to be in order to perform better, this also means more amount of computation is required. The current deep learning frameworks are more like a black box, they do not really tell how they work. They only expose a few parameters we can tweak for our implementation. That is why we need a simpler and more customizable, transparent program to work with.
The purpose of the project is to evaluate how does using variable precision(8-bit, 16-bit, 32-bit and 64-bit) on different kind of neural network model affect the quality, the performance and the speed the of model.

Problem statement

Most of popular the deep neural network frameworks used usually focus on optimizing the model itself and not the underlying driver functions, like changing the precision or modifying the runtime environment or instruction set architecture of the hardware. Since these make up for the foundation of any processor based computation, they are kind of left unexplored in terms of how it might affect the neural network performance. Thus a question that arises is “Are we computing more than required”?

Related work

1. [1] In this paper they evaluate the increase in compute density with change in precision. They estimate till what point determine till what point we can approximate the precision without losing performance.
2. [2] [7] In these papers they suggest that using different precision at different layers linear increase in performance with right combination of precision.
3. [3] [4] These papers talk about how using fixed point instead of floating point without much loss of accuracy.
4. [4] There are many Nvidia GPUs that have support for lower precision. The performance statistics(FLOPS) are also provided
5. [5] This paper talks about optimizing the parallel routines for optimal compilation and runtime for the deep learning models.

Proposed solution

We want to measure the impact of variable precision on both, the speed and the precision of the neural network.
To do so, we need to build our own neural network in order to be able to configure it as we want. We have classified possible areas in which we can focus our research to get valuable results and perform comparison. The three main aspects we will focus on to optimize the results:

- ➔ Software Optimization
- ➔ Model Optimization
- ➔ Hardware Optimization

Development Environment/ Language : C/C++, Python, CUDA
Compiler : gcc for C, TVM: Open Deep Learning Compiler Stack for Python
Performance Standard: Tensorflow, MXNet, Keras

We decided to work with C/C++ as it is a low level language and fast therefore it allows extreme flexibility and customization. We plant to use Python to run Tensorflow, MXNet or Keras on various datasets to compare the result with our neural network implementation to verify whether or not our results are consistent and coherent. Algorithms will have to be written in different versions to support different CPU and GPU architectures.

These are the areas we will be trying to target for each of the three main aspects:

Software Optimization(Driver Function Modification)

1. Effect on accuracy of DNN model with respect to various precision(quarter, single, half and double).
2. Effect of change in precision on speed with which the model can be trained. Provided the hardware remains the same even before changing the precision.
3. Data loss the number of bits is reduce. This might be a tradeoff for faster training time. How does that impact the quality of the model and its capability to discern patterns?.

Model Optimization(Neural Net Modification)

1. Reaction of different DNN algorithms to change in the precision.
2. Reaction of different kind of datasets(images, text etc) to change in precision. Is there a specific combination of dataset type, precision and algorithm that can yield super fast training time with high accuracy?
3. Usage of a particular type of precision for the neural network layers.
4. Effect of adding more layers in the neural network(8 -bit or 16-bit or 32-bit or 64-bit layers), on the training time positively or negatively One of the proposed work suggest that using different precision at different layers linear increase in performance with right combination of precision.
5. Effect of the optimization algorithms on the accuracy of the model and how each of them behave when we change the number of bits(Stochastic gradient descent, AdaGrad, RMSprop, Adam) (https://towardsdatascience.com/neural-network-optimization-algorithms-1a44c282f61d)
6. Vanishing Gradient: The gradient can vanish if it is too small, leading to an incapacity for the neural network to learn anything. How worse can it get with smaller precision? (https://ayearofai.com/rohan-4-the-vanishing-gradient-problem-ec68f76ffb9b)
7. Impact of variable precision on the activations functions
8. Using weights that connect neurons, of different precision.

Hardware Optimization(CPU/ GPU)

1. Effect of changing the hardware architecture on the performance of the model, for example using the VNNI instruction set architecture.
2. Parallelising and multi-threading the computation of training of data set.
3. Threading of implementation of model generated.

Proposed Approach

➔ We will start by implementing our own neural network in C/C++ and make it work with a simple dataset such as the MNIST(handwriting). In order to validate our results, we will compare them with a similar model built with Tensorflow.

➔ Once everything is working fine we will be able to start the experimentations and benchmarks. First of all, we are going to perform small tweaks on the network such as changing the variable's precision to commonly used types like 8/16/32/64 bits floats and benchmark our results based on performance and speed metrics.

➔ By then, we will be able to clearly see some behavioural pattern when changing the number of bits. In order the have more in depths results we want to write a library that will allow us to choose more exotic types and precisions such as the complete range between 1 bit and 64 bits.

➔ We will also try the precision combination with fixed point/ floating point or dynamic fixed point weights.

➔ But then, how do we know if the results we have are not just simply related to the kind of network we have or even the dataset itself ? To answer this question, we will proceed to the next step which is: building confidence in our results. We are going to try those same previous experiments on new datasets and reworked models. It is fundamental to test a large range of settings for each of our models to locate the source of what disrupts our results. We are going to test parameters such as activations functions, optimizations algorithms (Adam, gradient descent etc) and hyperparameters such as the number of layers and the number of neurons per layer.

➔ As we will be working on multi threaded machines we need to explore hardware optimization. We will be testing different sets of CPUs/GPUs instructions as well as various numbers of threads and cores. Again, the goal is to study how a change in precision can impact the performances and the speed, so every results will be recorded and added to the benchmark to compare them.

➔ Lastly, if we have some time left we would like to experiment a kind of systematic approach on how to find the best hyperparameters and parameters for our neural networks. We may start looking in the direction of genetic algorithms or simulated annealing as optimization algorithms.

➔ Another idea would be to customize our own compiler to make it use the best instructions possible for a given algorithm/model. One of the example is the TVM compiler stack for Python.

Evaluation

For each experiments we are going to conduct, we will benchmark several useful metrics to compare the results we get.

Model metrics:

- ➔ Accuracy
- ➔ Recall
- ➔ Confusion matrix
- ➔ Kappa's coefficient

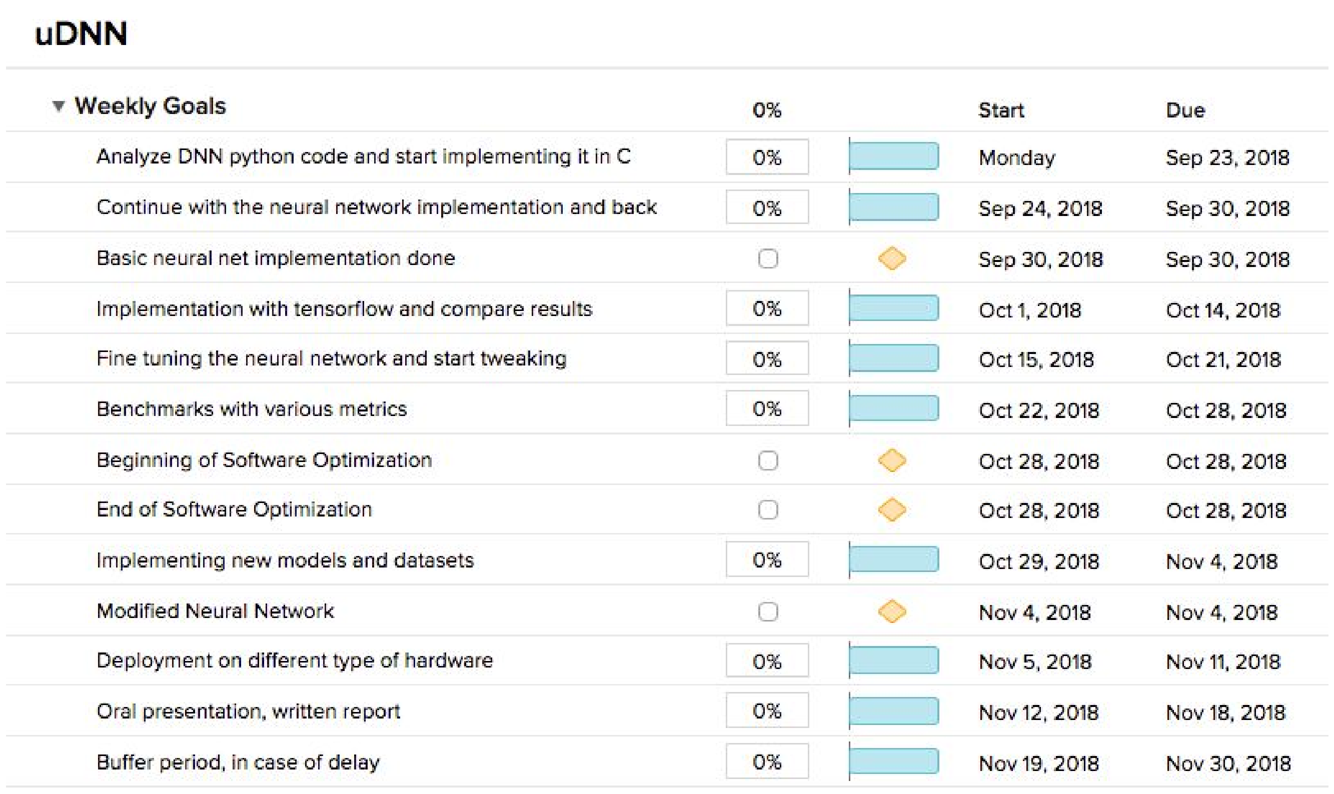
Hardware metrics:

- ➔ Number of cores/threads
- ➔ Computing power
- ➔ RAM, cache

Software metrics:

- ➔ Training speed
- ➔ Evaluation speed
- ➔ Operation's precision (Number of bits used)
- ➔ Data loss (e.g: How to round numbers to get minimal loss)

Timeline with weekly goals



uDNN

▾ Weekly Goals	0%		Start	Due
Analyze DNN python code and start implementing it in C	0%	▭	Monday	Sep 23, 2018
Continue with the neural network implementation and back	0%	▭	Sep 24, 2018	Sep 30, 2018
Basic neural net implementation done	☐	◆	Sep 30, 2018	Sep 30, 2018
Implementation with tensorflow and compare results	0%	▭	Oct 1, 2018	Oct 14, 2018
Fine tuning the neural network and start tweaking	0%	▭	Oct 15, 2018	Oct 21, 2018
Benchmarks with various metrics	0%	▭	Oct 22, 2018	Oct 28, 2018
Beginning of Software Optimization	☐	◆	Oct 28, 2018	Oct 28, 2018
End of Software Optimization	☐	◆	Oct 28, 2018	Oct 28, 2018
Implementing new models and datasets	0%	▭	Oct 29, 2018	Nov 4, 2018
Modified Neural Network	☐	◆	Nov 4, 2018	Nov 4, 2018
Deployment on different type of hardware	0%	▭	Nov 5, 2018	Nov 11, 2018
Oral presentation, written report	0%	▭	Nov 12, 2018	Nov 18, 2018
Buffer period, in case of delay	0%	▭	Nov 19, 2018	Nov 30, 2018

Deliverables

- ➔ Basic deep neural network implementation.
- ➔ Modified neural network working on a dataset and various benchmarks with respect to the metrics mentioned in Evaluation section.
- ➔ Hardware modifications and multi-threading.
- ➔ Final comparison results with Tensorflow.
- ➔ Bonus : Compiler customization and optimization algorithm.

Conclusion

This project will help us understand deep learning neural network in depth, since we will be implementing our own neural network from scratch. DNN are very complex and a breakthrough in field of artificial intelligence, but without optimal compilation, runtime environment and hardware they can be very time and resource hogging. But in this project we will learn to optimize the neural network to run as fast as possible and without much resources at the same time maintaining great accuracy. If we are able to get comparable results on various problems with our own DNN implementation with industry popular frameworks like Tensorflow, with right combination of all the optimizations listed in Proposed Solution, then we will evaluate our project to be a success.

References

[1] Srivatsan Krishnan , Piotr Ratusziak , Chris Johnson , Duncan Moss , and Suchit Subhaschandra, “Accelerator Templates and Runtime Support for Variable Precision CNN”, Intel Corporation University of Sydney.
https://parasol.tamu.edu/pact17/Sri-architecture-templates-runtime_final.pdf

[2] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos, “Stripes: Bit-serial deep neural network computing,” in Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on. IEEE, 2016, pp. 1–12.
http://delivery.acm.org/10.1145/3200000/3195661/a19-judd.pdf?ip=104.194.117.240&id=3195661&acc=ACTIVE%20SERVICE&key=50864D773CC43BF0%2E50864D773CC43BF0%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1536363972_1ef09c328d5ee53689c6700a2efd3b96

[3] Philipp Gysel, Mohammad Motamedi & Soheil Ghiasi, “HARDWARE-ORIENTED APPROXIMATION OF CONVOLUTIONAL NEURAL NETWORKS”, Department of Electrical and Computer Engineering University of California, Davis.
https://arxiv.org/pdf/1604.03168.pdf

[4] “Nvidia tesla p4 inferencing accelerator,” https://www.nvidia.com/enus/data-center/tesla-v100/, accessed: 2017-08-30.

[5] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, “cuDNN: Efficient Primitives for Deep Learning”, NVIDIA, Santa Clara, USA.
https://arxiv.org/pdf/1410.0759.pdf

[6] Matthieu Courbariaux & Jean-Pierre David, Ecole Polytechnique de Montreal and Yoshua Bengio, Universite de Montreal, CIFAR Senior Fellow, “TRAINING DEEP NEURAL NETWORKS WITH LOW PRECISION MULTIPLICATIONS”.

https://arxiv.org/pdf/1412.7024.pdf

[7] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, “Deep Learning with Limited Numerical Precision”, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598.
https://arxiv.org/pdf/1502.02551.pdf

[8] Jürgen Schmidhuber, “Deep learning in neural networks: An overview”.
https://ac.els-cdn.com/S0893608014002135/1-s2.0-S0893608014002135-main.pdf?_tid=993b0404-b445-4dab-8a39-c36039a79e15&acdnat=1536369787_d08e44d4e1640e32bcd7bed6840dcfe0

[9] Optimizing Tensorflow for low power and resource devices like the mobile phone and raspberry pi.
https://www.tensorflow.org/mobile/optimizing

[10] A visualization of how deep neural network work by Google.
https://playground.tensorflow.org/#activation=linear&regularization=L2&batchSize=10&dataset=circle&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=4&seed=0.74185&showTestData=false&discretize=false&percTrainData=50&x=true&y=true&xTimesY=false&xSquared=false&ySquared=false&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero=false&hideText=false

[11] A deep learning overview suggested by Google.
http://www.deeplearningbook.org/