ApproxTrain Framework for Multiplier Generation

John Clayton Sanford   
*School of Electrical, Computer and Biomedical Engineering, Southern Illinois University, Carbondale, U.S.A.*john.sanford@siu.edu

*Abstract* — Deep Neural Networks (DNNs) utilize very many multiplication and addition operations to compute estimation. As DNNs do not rely on high levels of precision to function, efforts have been made to reduce the time to train and test DNNs by simulating the operation of DNNs with approximate multipliers. This paper aims to introduce a framework for utilizing the library ApproxTrain to generate multipliers with customizable variance and evaluate the impact of variance on different stages of DNNs.

Keywords — Approximate computing, DNN, DNN accelerators, ApproxTrain

# Introduction

Deep Neural Networks (DNNs) and AI in general have been becoming increasingly prevalent in modern life. Convolutional Neural Networks (CNNs) have become especially prevalent in daily life, from the high-precision recognition networks driving self-driving vehicles, to lower stake networks acting as filters on cell phone cameras recognizing and modifying facial features for photos taken and added to social media sites.

As these DNNs become more prevalent, they also become more complex. This complexity has costs associated with it, namely in the time to train and validate these models, as well as the size of hardware dedicated to generating and running these networks.

As DNNs are generally the result of training a network with random weights and biases, precision is generally not the most prominent concern of the project, as accuracy is typically much more desirable. Because of this, DNNs inherently have a high tolerance for error in computation [1]. In an attempt to utilize this tolerance for error, developers can tradeoff the precision of the circuits that handle the mathematical operations inside the network for greater speed and smaller footprints.

However, this tolerance is not a guarantee that a neural network will work with any variance in the mathematical operations. Bounds of tolerance may exist where these networks function with low loss to accuracy, and variances in the computation that drives the neural network beyond these bounds may negatively impact the network beyond all practical use. Because of this, it is useful to be able to simulate and test neural networks with different variances in the hardware to find the impacts of the variance in the arithmetic.

# Related work

There are several ongoing projects working on the approximation of arithmetic in training and verifying DNNs, most notably are ApproxTrain (featured in this paper), AdaPT, and EvoApproxLib.

## ApproxTrain

ApproxTrain is the library that this framework utilizes. ApproxTrain aims to introduce error into neural networks via the implementation of approximate multipliers. ApproxTrain takes the c/c++ code of an approximate multiplier, generates a lookup table, and then uses python to call these c-based multipliers. ApproxTrain modifies TensorFlow tools for CNN simulation to utilize the approximate lookup tables and multipliers instead of the normal more precise floating-point operations that are natively run in python and c-based languages.

ApproxTrain has been utilized in several other papers determining the effects of approximate multipliers on the testing of deep neural networks.

## AdaPT

AdaPT is similar to ApproxTrain, although there is significantly less documentation around the library. AdaPT also generates lookup tables from c-based code to implement as approximate multipliers while evaluating DNN models. AdaPT also has a simple process, it sets up how it is going to train a DNN, trains it, and then retrains the DNN with the approximate multiplier. AdaPT differs from ApproxTrain in the retraining part; although ApproxTrain can retrain a DNN, AdaPT always retrains a DNN. AdaPT also seems to be unable to simply evaluate a trained DNN, instead taking an architecture for a DNN, training it, and then evaluating.

## EvoApproxLib

EvoApproxLib simulates the hardware itself instead of just emulating the mathematical behavior of approximation. Since the hardware is simulated, there is significantly more control and more information from the simulations. However, EvoApproxLib seems to also have increased latency compared to the other mentioned tools for introducing error into the

# Methodology

The general methodology of this framework is quite simple. Using the native C++ std::normal\_distribution class template, it is possible to generate a “multiplier” that uses randomness to generate error with a specified variance. This multiplier is then used to train and evaluate a CNN in different layers.

*A. Generating Error*

The average of the normal distribution entity is the float value that is the multiplication of the two input floats, and the variance is a user-specified variance. For every multiplication operation that calls the generated multiplier there will be an error injected into the network that the parameters can control. This is useful in the process of designing circuits to function in neural networks, as if a multiplier with a specific variance can be designed, it can be simulated with this software without having to use any form of hardware elaboration or mathematics.

*B. Implementation*

The ApproxTrain library generates a lookup table (LUT) based on the variance of the multiplier, and after a lookup table is generated, the testing and training of the DNN can begin. The framework generates a simple CNN with four layers. The first two layers of the CNN operate using the TensorFlow Conv2D. The last two layers operate using the TensorFlow Dense. In between every layer the network pools. After running through the training and testing of the model, the training and testing is repeated with specific layers using approximate multipliers instead of the native multipliers, and finally the entire network is run using approximate multipliers.

# Evaluation

This framework does seem to generate a multiplier, although the generation of the lookup table is questionable. Since there is no guarantee that the multiplier will behave as the lookup table models, this step should most likely be skipped. However, the ApproxTrain library needs to generate a lookup table to function, and as such this step is required.

There is sometimes considerable slowdown from this random multiplier during training. For reasons unknown, the slowdown during training and testing was not consistent throughout all of the attempts of running the framework, and no correlation between variance or any other variable could be found. This slowdown could increase the time to train the model by orders of magnitude, which is undesirable.

The overall results were predictable and uninteresting. As the variance increases in the random generation, so too does the error. Moreover, error applied earlier in the DNN causes a higher loss than error applied later in the DNN, as early error can propagate through more of the network and become magnified.

# Conclusion

This framework has some value in the process of studying the impacts of approximation on neural networks. However, there is a lot to be left desired by this solution. An ideal framework would allow for more customization of the approximate multiplier. EvoApproxLib has significantly more control that could be desirable for this purpose, such as maximum variances (which the gaussian distribution has no control of), error probability (the current method will always yield the same probability of error), and more importantly is deterministic instead of random. Although randomness is not uncommon in neural networks, every introduction of randomness is another source that the network can function improperly.

Moreover, control over the network would be much more desirable. Limiting to the same form of CNN every iteration is far from useful for any practical means, especially when deeper networks will have significantly more impact from the error. This framework provides no way to load a neural network and simulate approximate multiplications, which was a core criteria of this project.

##### Acknowledgment

John Sanford would like to both thank and apologize to Professor Iraklis Anagnostopoulos. Thank you for going out of the way to ensure that I was able to work with ApproxTrain, and apologies for the lackluster paper and functionality missing from the designed framework.

##### References

1. O. Spantidi and I. Anagnostopoulos, "How much is too much error? Analyzing the impact of approximate multipliers on DNNs," 2022 23rd International Symposium on Quality Electronic Design (ISQED), Santa Clara, CA, USA, 2022, pp. 1-6.
2. G. Tasoulas, G. Zervakis, I. Anagnostopoulos, H. Amrouch and J. Henkel, "Weight-Oriented Approximation for Energy-Efficient Neural Network Inference Accelerators," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4670-4683
3. J. Gong, H. Saadat, H. Gamaarachchi, H. Javaid, X. S. Hu, S/ Parameswaran R. Nicole, “ApproxTrain: Fast Simulation of Approximate Multipliers for DNN Training and Inference,” 2022
4. D. Danopoulos, G. Zervakis, K. Siozios, D. Soudris, J. Henkel, “AdaPT: Fast Emulation of Approximate DNN Accelerators in PyTorch” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 6, pp. 2074-2078