A Comparative Analysis of Neural Networks Implemented with Reduced Numerical Precision and Approximations

Vitor Ferreira Torres[1], **Frank Sill Torres**[2]

[1] Graduate Program in Electrical Engineering
Federal University of Minas Gerais
Av. Antonio Carlos 6627, 31270-901,
Belo Horizonte, MG, Brazil

[2]Department of Electronic Engineering

{vtorres,franksill}@ufmg.br

Abstract. *Machine Learning belongs to a class of computational problems that are resilient to noise and approximations. This is recently being explored in efficient implementations of these algorithms which aim to reduce resource consumption (like memory and energy) while retaining the application performance or trading off some acceptable degradation for the benefits. This paper compares precise and approximated implementations of Artificial Neural Networks, trained with the same architecture and parameters and applied to well known benchmark problems. Results show that direct conversion to more efficient and approximated structures does not always work flawlessly when simplifications are applied not only to the network but also to the training calculations.*

1. Introduction

Machine Learning (ML) applications have become ubiquitous and their implementation is realized in very distinct hardware levels: from embedded platforms, with power, processing and storage limitations to large clusters or tightly coupled parallel processing units. Both levels of hardware platforms are relevant [Wu et al. 2011, Reed and Dongarra 2015] for the ML field and can benefit from optimizations in the final implementations.

Artificial Neural Networks (ANN), an established method to handle ML problems, have been closely related to hardware since their early proposals. ANNs not only were inspired by biological models but were also presented in efficient hardware implementations which became known as "Neuromorphic Computing" [Mead 1990]. These early proposals have not become the standard implementations, largely due to the fast evolution in digital hardware computing performance. The popularity of other methods (e.g. Support Vector Machines and Random Forests) may also have contributed to a reduced interest ANNs in general. After a period without a considerable progress, recent years have seen a few important commercial examples of platforms optimized for the computations used in ANNs: Qualcomm (Zeroth Processor - 2013), IBM [Merolla et al. 2014] (TrueNorth - 2014), NVIDIA (Tesla P100 - 2016), Google [Jouppi et al. 2017] (Tensor Processing Unit - 2016). Intel has also joined this trend [Nervana 2017]. Besides providing tailored architectures for ANNs, a common resource in these hardware solutions is the availability of reduced numerical precision calculations.

Approximate Computing (AC) studies the impact of the lack of hardware reliability, deterministic or not, on applications and its potential for improving efficiency

(e.g., in terms of power consumption or chip area). Related to problems that became a concern when integrated digital circuits reached a level of miniaturization and speed at which reliability started to be an important concern, AC [1] appeared as a technique not only to be aware of these problems but to explore the simplifications to obtain efficiency gains [Agrawal et al. 2016]. Some recent examples of AC used to improve ML efficiency will be presented in Section 2.

The analysis presented in this work intends to compare reference (precise) implementations of ANNs to approximated equivalent networks that require less memory but perform calculations in a less precise manner. Simpler math operations also require less complex hardware or reduced software libraries for low-end processors. This comparison is performed under the same conditions (network structure, activation functions, training algorithm, adjustable parameters and initial weights) to focus the analysis on the approximations themselves.

2. Recent Work

The biological inspiration [2] of ANNs [Rosenblatt 1958, McCulloch and Pitts 1943] is clearly an intrinsically analog model. The first breakthrough [Williams and Hinton 1986] for their practical implementation provided a way to determine the neuron connection weights by iteratively back-propagating their adjustments from the output to the input layer. These changes were calculated based on the output error of each training example, and applied according the activation derivatives. Although the original model is inherently resilient to noise, this training process must be handle with more care.

Benchmark datasets handled with deep neural networks are tested with three different number representations (floating point, fixed point and dynamic fixed point) and analyzed in [Courbariaux et al. 2014]. Authors come to the conclusion that simple fixed point is considered harmful because activations, parameters and gradients have very different ranges, which may vary during training. The dynamic fixed point implementation was found to be equivalent to the reference floating point, even when used during training.

Energy savings from 34% to 51% in ANNs are reported in [Zhang et al. 2015] by applying approximations in data representation, computation and memory accesses. An iterative optimization heuristics to select candidate neurons and adjust their approximation is presented. The experimental results are obtained by simulation of the real hardware implemented in a 45nm CMOS technology and show less than 5% of quality loss.

In [Gupta et al. 2015] the authors train deep networks with 16 *bits* fixed-point number representations and stochastic rounding. This proposal presents similar results to 32 *bits* floating-point reference implementations in terms of classification accuracy. It is also mentioned that a mixed-precision approach, with higher precision fixed-point operations at the end of the training process, is also a promising strategy.

The paper [Lin and Talathi 2016] reviews recent literature showing that stochastic rounding is a good strategy to improve stability of the training process in deep neural networks under limited numeric precision. Three proposals are presented: low precision weights and full precision activations, fine-tuning only the top layers and a bottom-to-top

[1] "deterministic designs that produce imprecise results." [Han and Orshansky 2013]

[2] neurons with several inputs and their respective weighted sum, which determines the output activation

iterative fine-tuning scheme which applies approximations progressively. With similar results between the proposals, the best performance was achieved with 8 *bits* weights and 16 *bits* activations, which, if reduced to 8 *bits*, cause only a 2% drop in accuracy.

The analysis presented in [Hashemi et al. 2016] is based on a broad range of numerical representations applied to ANNs in both inputs and network parameters: floating point, fixed point, powers of two multiplications and binary weight nets. The compromise between accuracy and hardware implementation metrics is analyzed and results show a wide range of approximation parameters with negligible degradation in performance.

3. Methodology

The purpose of the following comparison is to perform the training and inference processes of ANNs in different well known datasets and evaluate the performance of approximated implementations. There are many examples in the literature of considerable approximations in the inference phase of the ANNs, but not as many applied during the training process, and even less to the training calculations themselves. Also rare are theoretical analyses of the effect of approximations in each operation on the final result. Another aspect not frequently studied is how to automatically select where and how to approximate, while guaranteeing acceptable results.

The datasets were chosen to allow the use of ANNs with similar topologies: multilayer, fully connected, without recursion or convolution. Even without reaching state-of-the-art results, the focus was not to extract the maximal performance from the baseline, which is the reference implementation with double precision floating point representation and standard math library functions, but to compare its performance with the approximated versions. The type of application was also a factor kept constant, hence all selected benchmarks were classification tasks with supervised training. Data ranges were linearly scaled when needed, using the limits of the training set.

The "Fast Artificial Neural Network Library" (FANN [Nissen 2012]) was used as a basis for the implementation, but was heavily modified. One of the main changes was the integration of part of the Berkeley SoftFloat Library [Hauser 2017], to perform half-precision (16 *bits*) standard [IEEE 2008] floating point operations. All code and modifications were implemented in ANSI C, without architecture specific assembly optimizations, and compiled with GCC 5.4. Compiler optimizations which improved the training time when using the approximated implementations (like aggressive "inlining") were enabled but no accelerations to the native floating point operations were activated.

Using a standardized floating point format has the advantage to make this investigation relevant for the growing amount of available systems (compilers and hardware) supporting this representation. Data types used in the entire implementation were split in the following groups, selectable at compile time:

Group 1 variables used for external interaction (dataset loading and saving, ANN persistence, real time statistics etc) fixed in hardware native floating points

Group 2 variables used directly by the neurons (weights, steepness, neuron inputs and outputs) and the operations involving only these variables (except activation functions and derivative functions, which belong to **Group 3**)

Group 3 all other variables involved in the forward and backward phases of the ANN execution (including all parameters and operations of the training algorithms)

Analyzing the variable groups that are relevant for the training process, in the **Group 2** were included the most important for memory usage (space and traffic) in the forward phase in large networks and the ones that are frequently used with them (to reduce the need for conversions). Activation functions were not included in this group because their relevance in resource usage decreases with the network size and complexity: the more connections each neuron has, more relevant the sum of products in the inputs will be when compared to the activation. With this reasoning and the irrelevant variables in **Group 1**, the **Group 3** was already defined.

Figure 1 shows graphical representations of two piecewise linear approximations of mathematical functions frequently used in ANN. On the left, the Hyperbolic Tangent ('tanh') reference is compared with the approximated version (implemented in FANN and shown as "approx."), while the relative error is shown in the other 'y' axis. The Logistic function approximation has a very similar aspect since it is only a shifted and scaled version of the Hyperbolic Tangent. In the plot on the right, the exponential approximation [Schraudolph 1999] is also compared with the exact reference, but the difference is barely visible (even if the y axis is shown in logarithmic scale). Relative errors are also presented in the same graph and the observed pattern repeats itself in the whole useful range for 16 *bits* floating points. This approximation is used for the *Softmax* [Dunne and Campbell 1997] outputs and other activation functions (like the Hyperbolic Tangent and Logistic, when not using the direct piecewise linear versions).

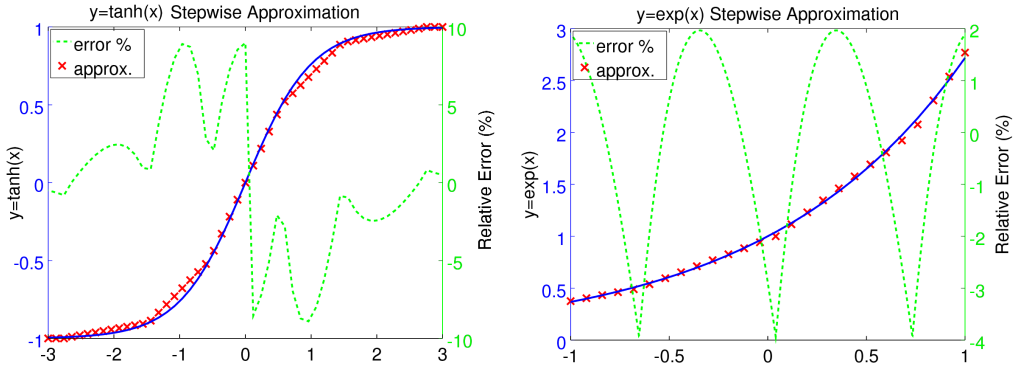

Figure 1. Approximations of Activation Functions and the Exponential

$$Logistic(x) = \frac{1}{1 + e^{-x}} \qquad (1)$$

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \qquad (2)$$

$$Softmax(\vec{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \qquad (3)$$

Equations 1 and 2 show the activation function definitions which were approximated by linear piecewise segments. Equation 3 shows the definition of the *Softmax* function, used for multi-class datasets implemented with mutually exclusive outputs. The actual implementation subtracts the maximum x_i from the exponents for better stability.

Regarding the numerical representation for the variable groups 2 and 3, three fixed implementations were selected for the tests. The baseline (already detailed), an intermediate approximation level (used only with the MNIST database, which will be described later) and the least precise one, used in all databases, as follows:

Group 2 modified half precision floating points, flushing subnormal numbers to zero and saturating maximum numbers (which is a simple modification that saves processing time or circuit/software complexity to handle these exceptions)

Group 3 standard half precision floating points, including the stepwise linear activation and derivative functions and the approximate exponential function for *Softmax* output calculation (using a readily available, but efficient, format for the group that requires more precision)

It should also be noted that conversions from groups 2 and 3 are actually not needed, since their hardware representations (exponent and significant values) are the same. Flushing subnormals to zero and replacing infinites for the maximum value are simple operations.

Weight initialization was also a factor kept constant among all tests and datasets. A normalized [Glorot and Bengio 2010] pseudo-random method was applied to every layer and each random seed was repeated once in the baseline and approximated runs. This assured that, for every baseline execution, there was a respective approximated trial starting from the same random weights (apart from the reduced numerical representations). Since all trials were trained with the same order of examples using the same algorithms and parameters, this procedure allowed a more direct comparison of the approximations effects in the training epochs.

4. Datasets

Table 1 summarizes the main characteristics of the classification datasets selected for this study. There is a mixture of following parameters: set size, input complexity and number os classes (mutually exclusive). Class imbalance also varies and will be analyzed later.

Table 1. Characteristics of the benchmark datasets

Dataset	Inputs	Outputs	Training Set Size	Testing Set Size
MNIST	784	10	60000	10000
Breast Cancer	30	1	455	114
Thyroid	21	3	3600	3600
Soybean	82	19	342	341

For each dataset the initial network architecture was chosen from one literature example and adjusted to be adequate for the intended comparison. This tuning, using the baseline implementation, searched for the highest accuracy on the training dataset (even with overfitting) and just checked if the selected configuration would perform adequately (convergence to a stable result) in the approximated implementation.

In order to obtain ANN configurations with enough capacity to detect possible regularization effects of the approximated methods, the training adjustments have evaluated the training accuracy only with the baseline. To be consistent with this search for an increase in the generalization capability, no regularization technique (e.g. early stop, weight decay, random noise, Dropout) was enabled and, based on the trial runs, a maximum number of epochs was fixed for interruption.

Regarding the training methods, this study preferred standard back-propagation in batch modes (mini-batch for the largest datasets and full-batch for the two others) over adaptive, more robust to parameter changes and faster learning algorithms (like iRPROP [Igel and Hüsken 2000], rmsprop [Tieleman and Hinton 2012] or Adam [Kingma and Ba 2014]). Not even incremental SGD with a momentum factor was thoroughly tested in the approximated implementations, hence not used in the trials. These better training algorithms also add more operations to the basic back-propagation method, which, if not behaved consistently, would also affect the methods based on it.

This methodology for architecture definition and training parameters adjustments resulted in the following configurations:

MNIST two Logistic hidden layers (the first with 50 neurons and the second with 200), *Softmax* output layer, mini-batch training (with batch size = 100).

Breast Cancer two TahH hidden layers (the first with 32 neurons and the second with 8), Logistic output layer, full batch training, data input was linearly scaled to the $[-1.0, 1.0]$ range, using the training dataset limits to calculate the offset and scaling factor, which were applied to the test dataset.

Thyroid two Logistic hidden layers (the first with 40 neurons and the second with 20), *Softmax* output layer and mini-batch training, with a batch size of 50.

Soybean two Logistic hidden layers (the first with 40 neurons and the second with 20), *Softmax* output layer and full batch training.

5. Results and Analysis

The following sections compare the training process of each dataset in approximated implementations to the baseline version, which uses double precision floating points and standard math libraries. Both the training and test accuracy are presented since the latter was not used for any decision during the process.

5.1. MNIST

As mentioned in section 3, the trials with this dataset included an intermediate approximation level. It was used as a first attempt for the approximations, which as then succeeded by the final implementation, already detailed.

This intermediate method was implemented as follows: all data and mathematical operations (groups 2 and 3) were implemented with standard half precision floating point except activation and derivative functions (realized with standard single precision floating points and math library functions). This implementation is documented as "approx. 1" while the other, identical to the one used in the remaining datasets, appears as "approx. 2" in the graphics.

Figure 2 shows in the graph on the left how train and test accuracy evolved on average on all the baseline runs. Approximate trials are not presented in the same graph due to the small differences between them and the baseline. For the same reason, the 95% confidence intervals (0.01% for the last training epoch and 0.05% for the last testing epoch) are not plotted. To make the small performance difference more visible, the graph on the right shows how much better, on average, the approximated versions perform compared to the baseline (a negative value means worse accuracy).

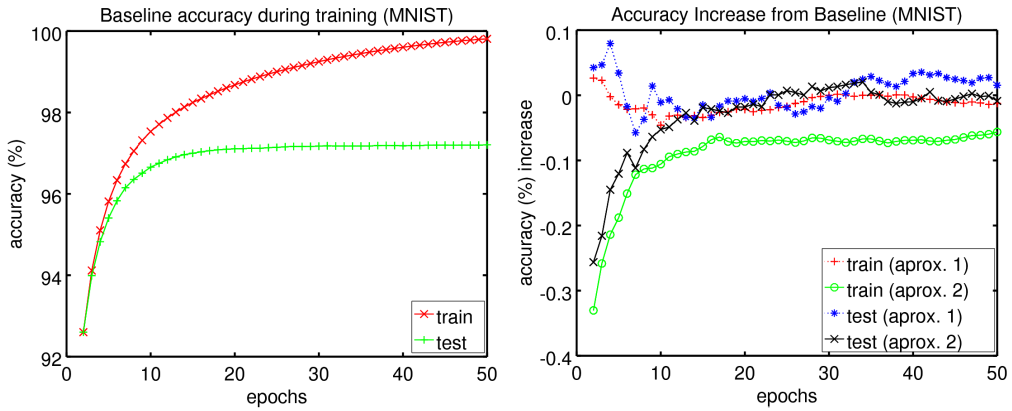

Figure 2. Accuracy Analysis of MNIST dataset

Even with very similar performances observed, the variability of the learning process was analyzed to check if the narrow confidence intervals were hiding less stable results. Figure 3 shows the standard deviation in accuracy along the training process, clearly showing that towards the end of the learning process all implementations converge to similar values of variability among successive independent runs.

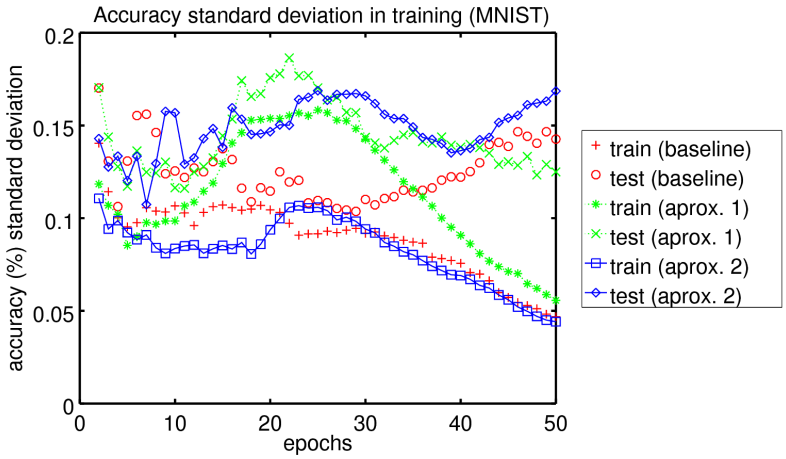

Figure 3. Accuracy Variation during training (MNIST)

Although sometimes considered a "*toy*" dataset, since the examples are preprocessed in order to reduce the spacial variability found in this type of problems, the presented results allow interesting conclusions. As the ANN proceeded to a clear overfit to the training data, the test performance did not decrease and this behavior was repeated on the approximated versions. Since the test performance difference was not significant, no evidence was found of generalization improvement due do the added noise. This noise was also not enough to make the stability of the convergence considerably worse.

5.2. Breast Cancer (Wisconsin)

This single class classification dataset differs significantly from MNIST. The total amount of examples (569) is two orders of magnitude lower, minimizing the regularization effect found in problems with a large amount of training data. There is also a slight class imbalance: $\approx 41\%$ of the training cases are positive and this number drops to $\approx 23\%$ on the

test set. For this reason, to avoid a benefit to the negative class, the reported accuracy is actually the average of the True Positive Ratio (TPR) and the True Negative Ratio (TNR).

Since the dataset classes are encoded with a single output (with values 0 and 1), the *Softmax* was not used in the output layer. A limited Inverse Hyperbolic Tangent error function was attempted to increase the penalty for larger errors, but the result was not robust in the approximated version. The interesting behavior observed was that in roughly 10% of the trials, after the ANN reached a state of learning stabilization, the weights suddenly saturated (within a single epoch). For this reason, a simple linear error function was used, still achieving better results than with the baseline.

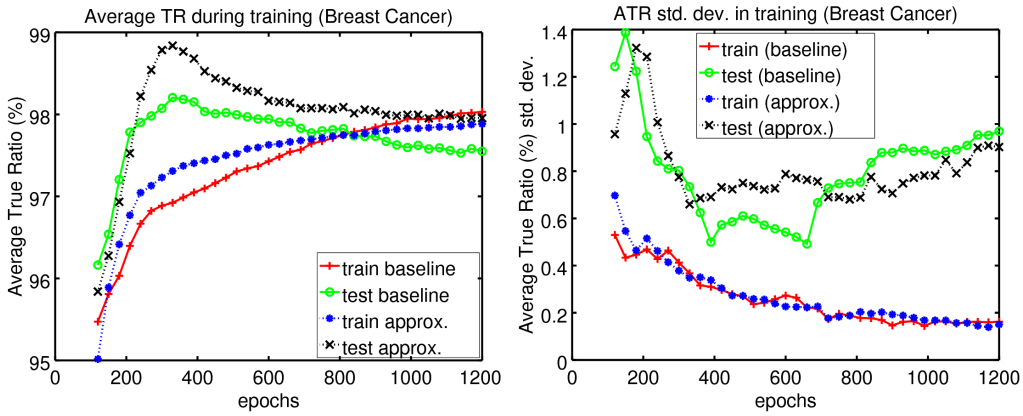

Figure 4. Accuracy Analysis of Breast Cancer dataset

Figure 4 shows in the graph on the left how train and test accuracy evolved on average on all the baseline and approximated runs. A minor but consistent advantage of the approximated version over the baseline is visible. The smallest 95% testing accuracy confidence intervals ($\approx 0.14\%$) coincide with the epoch range where the approximate version reaches its best results. This can be seen in the graph on the right where test performance variability (standard deviation) increases while overfitting proceeds (decreasing also the variability of the training process, which reaches a confidence interval as low as 0.03%). It could be argued whether the small accuracy gain is relevant, specially for a testing dataset where a single positive case missed represents a drop of 0.88% in the TPR, but the consistent differences found encourage further studies on the subject.

5.3. Thyroid

This dataset magnifies a problem found on the Breast Cancer problem: there are three classes but one of them represents more than 92% of the examples. Although the dataset has almost ten times more training data compared to the previous one, this imbalance poses problems to the batch training method used in this study. Overall training accuracy surpasses 99% within the first training epoch, but this number is dominated by the majority class. For this reason, the accuracies reported in the results that follow are the averages of the two minority class accuracies.

The setup chosen for this dataset showed good performance on the baseline but not on the approximated version. It was previously defined in the methodology that no adjustments would be performed to improve performance on the second trial group, otherwise

a direct comparison would not be possible. An instability similar to what was found with the Inverse Hyperbolic Tangent error function in the Breast Cancer dataset was observed.

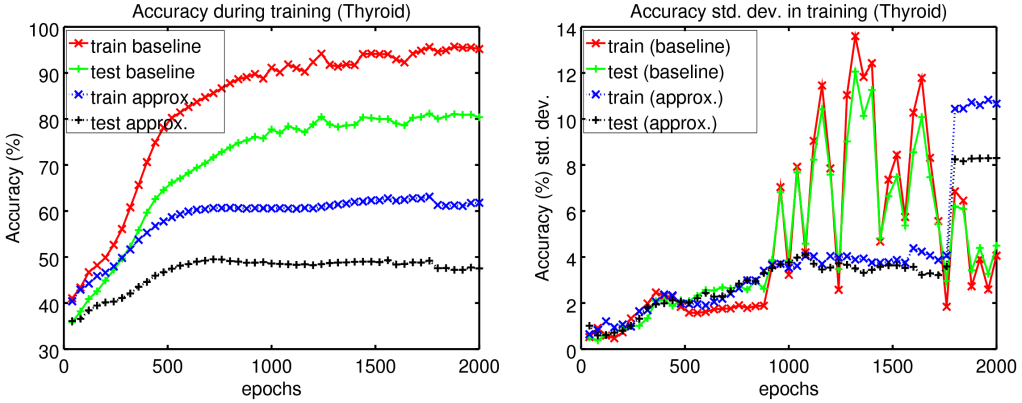

Figure 5. Accuracy Analysis of Thyroid dataset

Figure 5 on the left clearly shows that the accuracy of the approximated implementation is considerable inferior to the baseline. What is not clear is that if the learning rate could be improved in the approximated version, it would show the same variations that appeared in the baseline. The standard deviation plot on the right depicts a smoother approximate training behavior, up to the point when it is affected by the occasional instabilities already mentioned. It is worth pointing out that these issues could easily pass unnoticed if global accuracies had been used. This destructive behavior over the minority classes in the learning process is a known problem in unbalanced datasets.

5.4. Soybean

With this dataset there is a decrease in the set size when compared to the Breast Cancer but the number of classes increases to 19. There is also some class imbalance (the smaller ones have less than 10 examples), but due to the number of classes, the overall accuracy is used in the analysis.

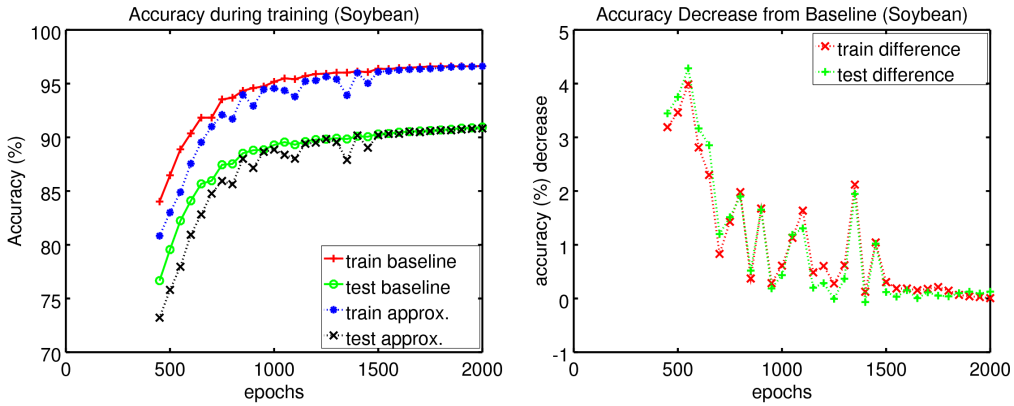

Figure 6. Accuracy Analysis of Soybean dataset

Figure 6 shows in the graph on the left how train and test accuracy evolved on average on all the baseline and approximated runs. The approximated version starts with a minor disadvantage but at the end of the selected training period the difference to the

baseline becomes irrelevant considering the 95% confidence intervals ($\approx 0.21\%$ for testing and $\approx 0.07\%$ for training). The graph on the right depicts how the disadvantage consistently drops towards zero. The pattern is similar to what was observed with the MNIST dataset, without considerable difference from baseline and with no clear overfitting to the training data.

6. Conclusions and Future Work

This comparative study showed some effects that numeric precision and approximations have on the training performance of ANNs. The presented results reinforce the importance of a careful robustness analysis of this ML method when implemented in GPUs (Graphics Processing Units) or other optimized hardware with simpler math operations.

For each dataset a specific analysis of the main results was provided. In two of the examples no evidence was found to show relevant differences between the baseline and approximated versions. In one of them the approximated version behaved consistently better regarding the accuracy and in the other a large disadvantage was observed, although the approximated implementation presented a smother learning curve.

A next step in this study will be the definition of an architecture selection and parameter optimization method, which would be applied to both implementations using the training data. This could result in different ANNs and training algorithms for the same datasets (or adjustments tuned for the specific problem) and the comparison of the two implementations could show more interesting differences: faster learning times, better resilience to overfitting and a final result with less variability.

Since the main goal of the approximations is the reduction of resources, it would not be acceptable that a complex quality monitoring system had to be added to the system. There is, however, a good opportunity to optimization if a light-weight process could dynamically adjust some approximations and trim down the complexity of the ANN (or even increase it, if the model was found to have insufficient learning capacity).

Finally, an investigation of faster training algorithms could show more vulnerabilities of the approximated approach, demanding more care during the training process to avoid instabilities. These methods will also present new opportunities for approximated arithmetic operations not used in this study. A method that finds superior results when only the best candidates are compared to the ones provided by other method may not be adequate if, on average, the quality of the results is inferior.

References

Agrawal, A., Choi, J., Gopalakrishnan, K., Gupta, S., Nair, R., Oh, J., Prener, D. A., Shukla, S., Srinivasan, V., and Sura, Z. (2016). Approximate computing: Challenges and opportunities. In *Rebooting Computing (ICRC), IEEE International Conference on*, pages 1–8. IEEE.

Courbariaux, M., Bengio, Y., and David, J.-P. (2014). Training deep neural networks with low precision multiplications. *arXiv:1412.7024 (Workshop contribution at ICLR 2015)*.

Dunne, R. A. and Campbell, N. A. (1997). On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne, 181*, volume 185.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). Deep learning with limited numerical precision. In *ICML*, pages 1737–1746.

Han, J. and Orshansky, M. (2013). Approximate computing: An emerging paradigm for energy-efficient design. In *Test Symposium (ETS), 2013 18th IEEE European*, pages 1–6. IEEE.

Hashemi, S., Anthony, N., Tann, H., Bahar, R., and Reda, S. (2016). Understanding the impact of precision quantization on the accuracy and energy of neural networks. *arXiv:1612.03940 (Accepted for conference proceedings in DATE17)*.

Hauser, J. (2017). Berkeley softfloat. `http://www.jhauser.us/arithmetic/SoftFloat.html`. Accessed: 2017-06-26.

IEEE (2008). *IEEE Standard for Floating-Point Arithmetic*. IEEE Std 754-2008.

Igel, C. and Hüsken, M. (2000). Improving the rprop learning algorithm. In *Proceedings of the second international ICSC symposium on neural computation (NC 2000)*, volume 2000, pages 115–121. ICSC Academic Press.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. (2017). In-datacenter performance analysis of a tensor processing unit. *arXiv:1704.04760 (To appear at the 44th International Symposium on Computer Architecture (ISCA), Toronto, Canada, June 26, 2017.)*.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lin, D. D. and Talathi, S. S. (2016). Overcoming challenges in fixed point training of deep convolutional networks. *arXiv:1607.02241 (As "Fixed Point Quantization of Deep Convolutional Networks" in Proceedings of the 33 rd International Conference on Machine Learning, New York, NY, USA, 2016. JMLR)*.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Mead, C. (1990). Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636.

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. (2014). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673.

Nervana (2017). Intel nervana hardware. `https://www.intelnervana.com/`. Accessed: 2017-06-26.

Nissen, S. (2012). Fast artificial neural network library. `http://leenissen.dk/fann/wp`. Accessed: 2017-06-26.

Reed, D. A. and Dongarra, J. (2015). Exascale computing and big data. *Communications of the ACM*, 58(7):56–68.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Schraudolph, N. N. (1999). A fast, compact approximation of the exponential function. *Neural Computation*, 11(4):853–862.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

Williams, D. and Hinton, G. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–538.

Wu, G., Talwar, S., Johnsson, K., Himayat, N., and Johnson, K. D. (2011). M2m: From mobile to embedded internet. *IEEE Communications Magazine*, 49(4).

Zhang, Q., Wang, T., Tian, Y., Yuan, F., and Xu, Q. (2015). Approxann: an approximate computing framework for artificial neural network. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 701–706. EDA Consortium.