Exploring Floating-Point Trade-Offs in ML

Rocco Salvia
School of Computing, University of Utah, USA
rocco@cs.utah.edu

Zvonimir Rakamarić
School of Computing, University of Utah, USA
zvonimir@cs.utah.edu

Abstract

Perceptron and Support Vector Machine (SVM) algorithms are two well-known and widely used linear predictors. They compute a hypothesis function using supervised learning to predict labels of unknown future samples. Both training and testing procedures are typically implemented using double precision floating-points to minimize the error, which often results in overly conservative implementations that waste runtime and/or energy. In this work, we empirically analyze the impact of floating-point precision on these predictors. We assess whether the precision of reading the dataset, training, or testing is the most critical for the overall accuracy. Our analysis in particular focuses on very small floating-point bit-widths (i.e., only several bits of precision), and compares these against the well-known and widely used single and double precision types.

1 Introduction

Floating-point representation was invented to model real numbers in computers. Representing real numbers as a finite sequence of bits naturally introduces approximation, which leads to errors in computations [7]. Hence, developers are often conservative about floating-point precision, and use the maximum precision provided by their target platform. However, executing floating-point operations can be a major contributor to runtime and energy consumption. For example, Tagliavini et al. [12] run a set of floating-point applications on a microcontroller, and show that 30% of the energy consumption was due to floating-point computations and another 20% due to moving operands between memory and registers. Hence, leveraging lower precisions can lead to significant savings, and the impact of low-cost floating-point precisions on machine learning predictors is an active research area.

Several recent papers explore the effects of floating- and fixed-point reduction on neural networks [8, 9, 14]. Gupta et al.[8] show that neural networks can be trained using low precision fixed-points, with a minimal sacrifice of quality. Zhou et al. [14] propose innovative methods to train convolutional neural networks with low bit-width weights, activations, and gradients. Hubara et al.[9] introduce a method to train quantized neural networks using low precision arithmetic. Others studied the effects of reduced precision on the SVM algorithm [2, 10]. As us, Lesser et al. [10] apply quantization on mantissas. Unlike us, they focus only on the classification process, and do not independently study reading, computing, and testing. Furthermore, they show that using any precision in the $[15, 52]$ range leads to comparable accuracy, while our results indicate that even lower precisions are adequate for many datasets; note that we do differ in kernel implementations (Gaussian vs Linear).

We study the effects of floating-point precision on the accuracy of the Perceptron and SVM machine learning algorithms. We focus on single-layer neural networks since they are easy to control and employ a convex optimization problem that converges to the optimum in reals. Moreover, in multi-layer neural networks the relation between quantization and accuracy might be affected by other factors orthogonal to our exploration. We implemented the algorithms using multiple-precision numerical libraries, which allows us to independently vary the precision of reading the dataset, computing the model, and testing of the computed model. We empirically evaluated the precision/accuracy trade-offs on several datasets, with a particular focus on very small floating-point bit-widths. Our results and conclusions serve as a guideline for making floating-point precision choices when implementing these algorithms.

2 Methodology

The format adopted for floating-point representation consists of the sign of the number, significand (mantissa), and exponent. This format is part of the IEEE 754 standard [1], which includes, among others, two well-known and popular representations: single precision (1-bit sign, 23-bit mantissa, 8-bit exponent) and double precision (1-bit sign, 52-bit mantissa, 11-bit exponent). However, it has been shown that using representations other than these default ones can be beneficial in many applications to improve runtime and energy efficiency [12], which is what we explore in this work.

Perceptron [11] is a mistake-driven algorithm: once a sample is misclassified the predictor is updated. The update step involves the weight vector as the main contributor to the classification process. Particularly interesting in terms of floating-point analysis is the average variant of Perceptron, which maintains an average predictor weighted on wrong predictions, and it needs a wider dynamic range for this computation. SVM [4, 13] is more restrictive than Perceptron since it aims for a safe margin between the linear separator and the closest samples. SVM is trained using a stochastic sub-gradient descent method, whose goal is to maximize the safe margin while minimizing the number of violating samples.

We implemented Perceptron (P), Average Perceptron (AP), and SVM algorithms using multiple-precision numerical libraries MPFR [6] and FlexFloat [12]. This

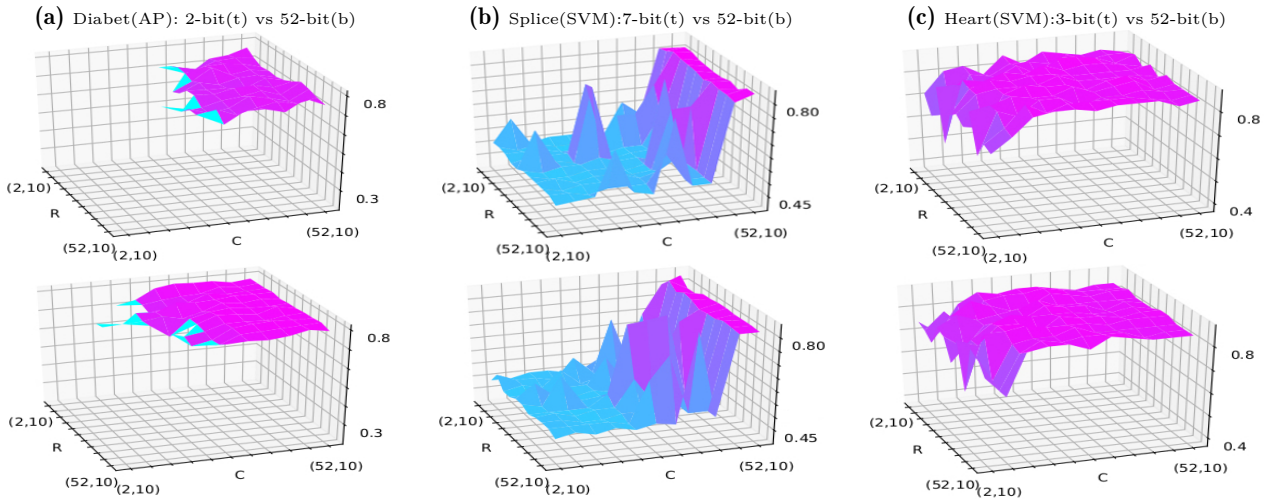

Figure 1. Accuracy for various precision regimes. Graphs compare two chosen testing precisions (top and bottom) on different datasets. Axis R denotes reading precision, C computing precision, and the third axis shows the accuracy. For example, $(2, 10)$ represents a 2-bit mantissa and 10-bit exponent.

allows us to specify how many bits are assigned to mantissa and exponent of each floating-point instruction. We mainly focus on varying the mantissa bits, since exponent does not play a major role in influencing the accuracy once it is large enough that exceptions (e.g., overflow, underflow) do not appear.

We analyze the accuracy impact of precision of the three main learning algorithm stages: reading the dataset, training, and testing. Storing and reading the dataset in low precision potentially leads to savings in memory traffic. Once the dataset is read at the desired floating-point precision, the training procedure of an algorithm executes with the specified computing precision. Finally, the generated model is passed to the testing procedure, executed with the specified testing precision. We measure the accuracy of the model by performing a 4-fold cross-validation on each dataset.

3 Results

We used the following datasets for our empirical evaluation: *fourclass* [3] and *heart*, *diabet*, *ionosphere*, *splice* [5]. They range in size between 270–1000 samples and 2–60 features. Before training, we scale each dataset to the $[-1, 1]$ interval, and detect a good hyper-parameters configuration for each algorithm. We vary the number of mantissa bits for reading, training, and testing from $\{2, 3, \ldots, 9, 10, 23, 52\}$. We made the source code and detailed results publicly available at https://github.com/soarlab/FPML. Fig. 1 shows several representative and interesting graphs we generated, and in the rest of this section we discuss key observations.

The precision used to store and read the dataset can be substantially reduced with respect to single or double precisions. This is encouraging since detecting a minimal threshold value for the reading precision can significantly reduce memory traffic. In all our datasets, reducing the reading precision does not have a noticeable impact on the final accuracy — samples can be stored even using only 3 bits for mantissa. However, the relation between quantization and accuracy depends on the dataset, and precision has to be fine-tuned for the given dataset.

The computing precision is the most critical and sensitive parameter. For example, Fig. 1(b) shows that in case of the *Splice* dataset, SVM has to be trained at the highest (double) precision to achieve good accuracy. To summarize, out of 10 analysis runs (5 AP and 5 SVM) only 30% of the time training has to be performed at double precision, while in the rest 10 bits is enough. (Note that we excluded Perceptron from the summary because it is hard to observe a precision threshold due to its erratic behavior in terms of accuracy.)

The testing precision can often be substantially reduced without having a major impact on the accuracy. For our datasets, using 9 bits mantissa for testing is always enough to reach the same accuracy as double precision. More precisely, 60% of the time 6 bits is enough, while 30% of the time 4 bits is enough. Finally, in one case using only 2 bits for testing achieves similar accuracy as double precision. Figs. 1(a,c) show how testing with just 2 and 3 bits produces accuracy similar to using double (52 bits). This is particularly important for deployment scenarios where training is performed on large machines potentially using double precision, but the learned models are deployed to for example mobile platforms using very low precision to conserve energy.

Acknowledgements We thank Annie Cherkaev and Vivek Srikumar for insightful discussions. This work was supported in part by NSF CCF 1552975.

References

[1] 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008* (2008), 1–70. https://doi.org/10.1109/IEEESTD.2008.4610935

[2] Davide Anguita, Andrea Boni, and Sandro Ridella. 2003. A digital architecture for support vector machines: theory, algorithm, and FPGA implementation. *IEEE Transactions on Neural Networks* 14, 5 (Sept 2003), 993–1009. https://doi.org/10.1109/TNN.2003.816033

[3] Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A Library for Support Vector Machines. *ACM Trans. Intell. Syst. Technol.* (2011), 27:1–27:27. https://doi.org/10.1145/1961189.1961199

[4] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (Sept. 1995), 273–297. https://doi.org/10.1023/A:1022627411411

[5] Dua Dheeru and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. (2017). http://archive.ics.uci.edu/ml

[6] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. 2007. MPFR: A Multiple-precision Binary Floating-point Library with Correct Rounding. *ACM Trans. Math. Softw.* 33, 2, Article 13 (June 2007). https://doi.org/10.1145/1236463.1236468

[7] David Goldberg. 1991. What Every Computer Scientist Should Know About Floating-point Arithmetic. *ACM Comput. Surv.* 23, 1 (March 1991), 5–48. https://doi.org/10.1145/103162.103163

[8] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15)*. 1737–1746. http://dl.acm.org/citation.cfm?id=3045118.3045303

[9] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Y Bengio. 2016. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *CoRR* (2016). http://arxiv.org/abs/1609.07061

[10] Bernd Lesser, Manfred MÃijcke, and Wilfried N. Gansterer. 2011. Effects of Reduced Precision on Floating-Point SVM Classification Accuracy. *Procedia Computer Science* 4 (2011), 508 – 517. https://doi.org/10.1016/j.procs.2011.04.053 Proceedings of the International Conference on Computational Science, ICCS 2011.

[11] Frank Rosenblatt. 1958. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review* (1958), 65–386.

[12] Giuseppe Tagliavini, Stefan Mach, Davide Rossi, Andrea Marongiu, and Luca Benini. 2017. A Transprecision Floating-Point Platform for Ultra-Low Power Computing. *CoRR* (2017). http://arxiv.org/abs/1711.10374

[13] Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA.

[14] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *CoRR* abs/1606.06160 (2016). http://arxiv.org/abs/1606.06160