Neural Network Precision Tuning With Stochastic Arithmetic

Basile Lewandowski

conducted under the supervision of Pr. Stef GRAILLAT

Abstract

With the increasing use of neural networks on embedded systems comes the need for low resource programs. The solution for more straightforward networks is often to lower the precision of their neurons parameters. Smaller weights however implies a less precise output in the end. The stake is then to find the right trade-off between accuracy and resources consumption. The approach presented here consists of determining the right precision among IEEE754 standards (half/single/double) through auto-tuning and dynamical analysis.

Keywords: round-off error, auto-tuning, dynamic analysis, numerical accuracy

1 Introduction

Neural Networks have become widely used, including in embedded systems and critical applications. The resources available on such systems are still however limited, especially in terms of computing power or memory. On the other hand, neural networks have proven themselves very sensitive to variation in their parameters, and may be very large in some cases. Our goal is therefore to provide a way to find a network's lowest size for a given output precision.
To do so, we are here using PROMISE (PRecision OptiMISE), a tool to auto-tune the precision of floating-point variables in numerical codes (cf. [5]). We will be balancing the parameters of our networks between the half, single and double precision floating-point digits (binary16, binary32 and binary64 as defined in IEEE754 [1]), and the round-off error will be measured through the CADNA (Control of Accuracy and Debugging for Numerical Applications) software.
This article is focused on simple feed-forward networks, as a proof of concept. We consider that our networks will be executed on CPU and should not use an extensive amount of memory, as it is often the case on embedded systems.
Precedent works have setup some alternative tools, such as [8] ; some even working on neural networks : for instance the solution proposed in [6] uses linear constraints to find the precision of each neurons. Our work is here based on stochastic arithmetic, an approach to provide confidence intervals for numerical results, as detailed in [3, 7].
We will first introduce our method and then present some experimental results on function approximation and image classification networks.

2 Design and setup

The fully connected neural networks we consider here are defined by a number of neurons N distributed on one or several layers L. Each neuron N_i is defined by a weight vector W_i and a bias b_i. In the end, the network is an affine transformation f made from the composition of the transformation of each layer and their activation function h_i. Let $x := (x_1 \dots x_d)$ be the input of the network we have :

$$f(x) := h_1 \begin{pmatrix} W_1 y + b_1 \\ \vdots \\ W_d y + b_d \end{pmatrix} \circ \quad \dots \quad \circ\, h_n \begin{pmatrix} \hat{W}_1 x + \hat{b}_1 \\ \vdots \\ \hat{W}_d x + \hat{b}_d \end{pmatrix}$$

Thus, the whole network is a sequence of n layers, which corresponds to the composition of n affine functions $f = L_1 \circ L_2 \circ \dots \circ L_n$. The activation function is a function often non-linear and monotonous, examples presented here will be : the hyperbolic tangent `tanh`, the rectified linear unit `ReLU`, the normalized exponential function `softmax`.

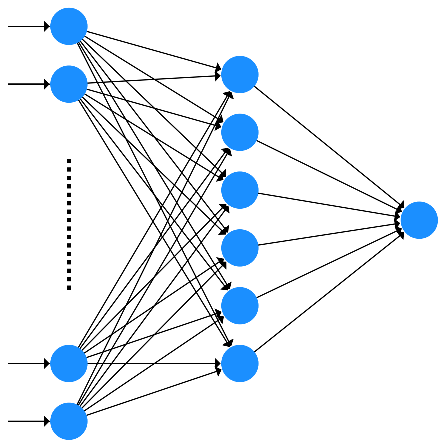
As explained in [4], PROMISE precision auto-tuning operates recursively : the first step consists of getting an default result with all variables to the maximum precision, to determine what the most accurate output would be. Second step is then to lower the precision of some variables and compare the output thus obtained with the default one. Step 2 is repeated on a trial and error fashion until the right accuracy is met for every variable analysed.
For the sake of convenience, we will be using `TensorFlow Keras` to design the network we will test. The translation from Python to C++ (which is used on PROMISE) is done by saving the network data in HDF format and then parsing it to be natively readable on C (see [2] for further details). The main goal is then to create a program file which can be analysed by PROMISE.
To do so, we split the network data so that each neuron has its own variable, to which we couple a PROMISE optimization type. Computing the output is then done using the transformation and activation functions as detailed above. An example of the final result, ready to be processed on PROMISE, and its translation is given below :

```
    __PR_L0N0__ l0n0_w = 0.028140 ... ;             float l0n0_w = 0.028140 ... ;
    __PR_L0N1__ l0n1_w = -0.01891 ... ;             double l0n1_w = -0.01891 ... ;
    __PR_L0N2__ l0n2_w = 0.116603 ... ;             double l0n2_w = 0.116603 ... ;
    __PR_L0N3__ l0n3_w = 0.471030 ... ;             half_float::half l0n3_w = 0.471030 ... ;
    __PR_L0N4__ l0n4_w = 0.220481 ... ;             float l0n4_w = 0.220481 ... ;
```

3 Results

3.1 Function approximation

First network experimented is an interpolation network to approximate the sine function. Though function interpolating through neural networks has few use cases in practice, it is a good way to measure how changing the network precision impacts the network accuracy. It is made of three fully interconnected layers with respectively 20, 6 and one neurons, as displayed in fig.1, and the activation function is `tanh` in every layer.



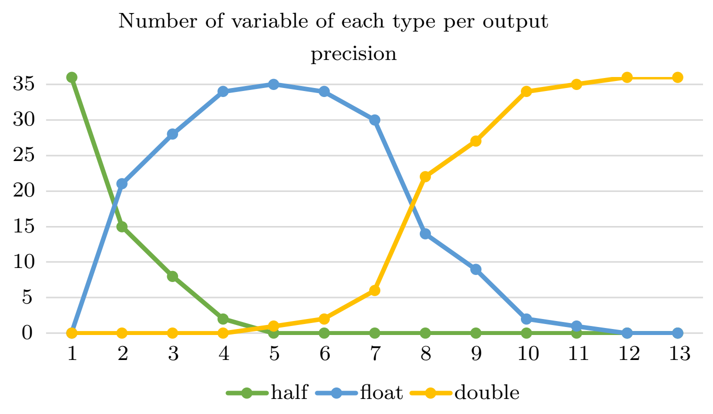
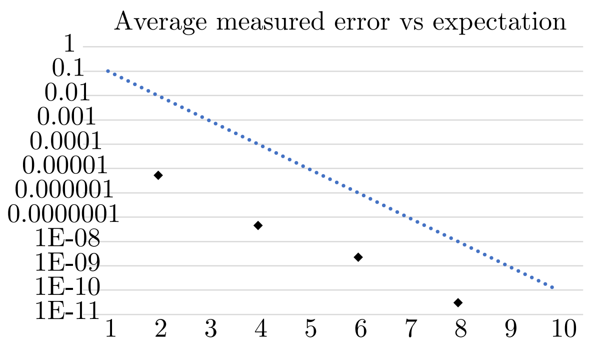


Figure 1: Model design and optimization results

Optimization behaves as expected, as we can see in fig.2 that the error measured on the network output is always below the number of significant digits specified as parameter. Concerning memory savings, it seems that optimization is more efficient on small significant digits (ie. broader approximation), as the amount of memory saved on highly precise results is lower in those cases.



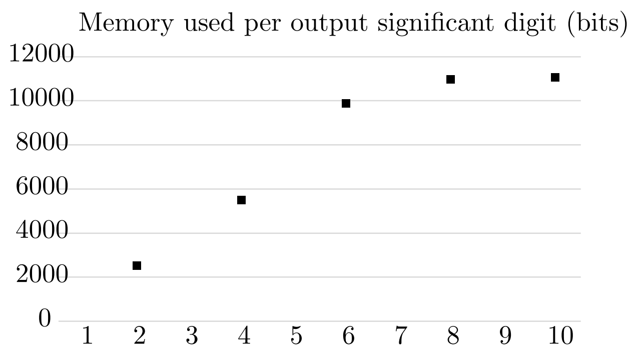


Figure 2: Network running performance

3.2 Image classification

The second network we introduce is a feed-forward image recognition network. We will be using the MNIST database, which contains a total of seventy thousand images of numbers from 0 to 9. The input is a 784 long double table containing the grey-scale value of each pixel for a 28*28 image. The network we use is made of two layers : the first one of 512 neurons activated by `ReLU` and the second of 10 neurons activated by `softmax`. Final classification is obtained through the maximum value of the output, with .97 accuracy by default.

The results of the optimization process are somewhat limited in this case. We can observe

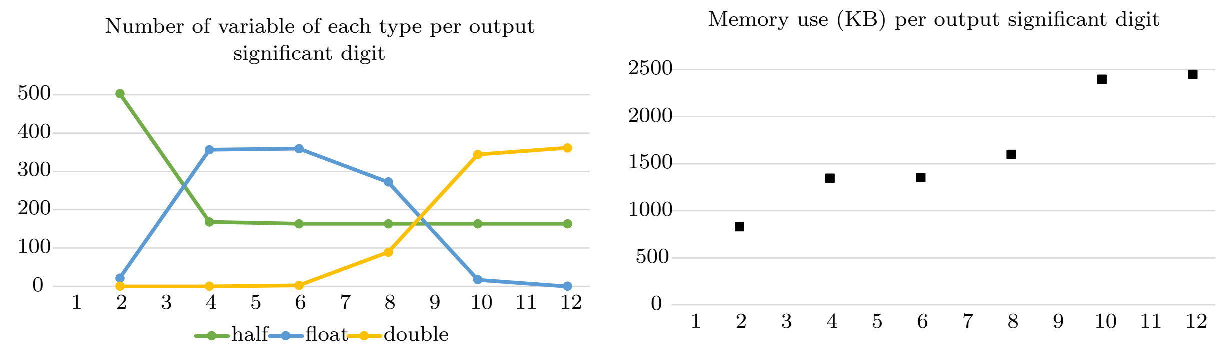


Figure 3: Network optimization results

at first that the distribution of variables in each case and the memory use behaves as expected (aside from half precision type, as we will discuss later). However, the accuracy of the final prediction seems not to be impacted by the precision of the network. Indeed, the accuracy measured remains the same no matter the precision of the output.

We tested the influence of the input value on which the network was optimized, and our experimentation show that there is no impact of this input value on the final accuracy. Three input configurations were tested : all pixels to the same value, average value on the set for each pixel, and a sample image ; both produced similar results in terms of prediction accuracy. In terms of type distribution, there was a small variation between the three cases, but nothing relevant in terms of memory saving. We can extrapolate that some neurons are related to pixels the value of which does not matter in the final results, and those pixels vary with the shape of the input. That would explain why there is a threshold on the half type in our experiment.

4 Conclusion

We presented in this article the result of optimizing a feed-forward neural network thanks to a stochastic arithmetic based tool. The first network, dedicated to function interpolation, has shown positive results, as the tool is able to find the right trade-off between memory use and output accuracy. The second network results are however more limited, as we were not able to find a correlation between memory use and output precision that is relevant to the final accuracy. We can formulate several hypothesis to explain this result : either the precision required for the output of the network to meet a lower accuracy is below the

ones we tested (ie. even with all parameters to the minimum precision, network's accuracy is not impacted), or the value of the output is not the right variable to analyse in the optimization process.
Further experimentation could be conducted on more complex networks, such as recurrent or deep networks. We have also considered the support of convolutional networks but we were not able to complete an implementation. Additional work could as well focus on more convenient ways to test a network and generate a neat implementation once it is optimized.

Acknowledgments

As stated previously, this work is based on two tools developed for floating-point precision tweaking. Resources can be fount online at `promise.lip6.fr` and `cadna.lip6.fr`.The source code of programs used in this paper is available online at `github.com/BasileLewan/NN-optimization-on-PROMISE`

References

[1] IEEE Standard for Floating-Point Arithmetic. Tech. rep., IEEE, 2019.

[2] Conlin, R., Erickson, K., Abbate, J., and Kolemen, E. Keras2c: A library for converting Keras neural networks to real-time compatible C. *Engineering Applications of Artificial Intelligence 100* (Apr. 2021), 104182.

[3] Eberhart, P., Brajard, J., Fortin, P., and Jézéquel, F. High performance numerical validation using stochastic arithmetic. *Reliable Computing 21* (2015), 35–52.

[4] Graillat, S., Jézéquel, F., Picot, R., Févotte, F., and Lathuilière, B. Auto-tuning for floating-point precision with Discrete Stochastic Arithmetic. *Journal of computational science 36* (2019), 101017. Publisher: Elsevier.

[5] Graillat, S., Jézéquel, F., Wang, S., and Zhu, Y. Stochastic Arithmetic in Multiprecision. *Mathematics in Computer Science 5*, 4 (Dec. 2011), 359–375.

[6] Ioualalen, A., and Martel, M. Neural Network Precision Tuning. In *Quantitative Evaluation of Systems, 16th International Conference, QEST 2019, Glasgow, UK, September 10-12, 2019, Proceedings* (2019), D. Parker and V. Wolf, Eds., vol. 11785 of *Lecture Notes in Computer Science*, Springer, pp. 129–143.

[7] Rene, A., Lamotte, J.-L., and Markov, S. Stochastic Arithmetic, theory and experiments. *Serdica Journal of Computing 4* (2010).

[8] Rubio-González, C., Nguyen, C., Mehne, B., Sen, K., Demmel, J., Kahan, W., Iancu, C., Lavrijsen, W., Bailey, D. H., and Hough, D. Floating-point precision tuning using blame analysis. In *Proceedings of the 38th International Conference on Software Engineering* (Austin Texas, May 2016), ACM, pp. 1074–1085.