How Bounded Precision in Computing of Weights Affects the AI Computation Results, and What Is the Optimal Allocation of Weight Precisions

Christoph Q. Lauter[1], Martine Ceberio[1], Marcelo Frias[1],
Esteban Rangel[2], Christopher J. Knight[2],
Eric Petit[3], and Vladik Kreinovich[1]

[1]University of Texas at El Paso, El Paso TX 79968, USA,
{cqlauter,mceberio,mfrias4,vladik}@utep.edu
[2]Argonne National Laboratory, Lemont, IL 60439, USA,
{erangel,knightc}@anl.gov
[3]Intel Corporation, Portland, OR 97124, USA,
eric.petit@intel.com

1. Need to solve systems of partial differential equations

- Real-world physical systems are usually described by systems of partial differential equations.
- So, to analyze and to control such systems, we need to solve the corresponding systems of equations, i.e.:
  - to transform the appropriate input x (e.g., the initial conditions and/or boundary conditions)
  - into the desired result y – e.g., the state of the system at some future moment of time.
- This is how we predict weather, this is how we predict how a building will react to an earthquake, etc.

2. Traditional way of solving such systems, its limitation, and how AI can help

- There are many numerical techniques for solving these equations.
- These techniques are usually very time-consuming.
- As a result, even on modern high-performance computers:
  - these computations may take hours or even days,
  - and the resulting accuracy is often lower than desired.
- In the last decades, very effective machine learning techniques have been developed, the most widely used is deep learning.
- In this technique, to compute the desired dependence $y = f(x)$ between the input x and the output y, we:
  - find many cases $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$ in which we know both the input $x^{(k)}$ and the corresponding output $y^{(k)} = f\left(x^{(k)}\right)$, and
  - train a neural network so that for the results $f_{NN}(x)$ generated by this network, $f_{NN}\left(x^{(k)}\right) \approx f\left(x^{(k)}\right)$ for all k.

3. Traditional way of solving such systems, its limitation, and how AI can help (cont-d)

- Once the training is over and we get the weights w_i for which the desired approximate equalities hold:
  - we then "freeze" (= fix) these weights and
  - use the resulting neural network to compute $y = f(x)$ based on x.
- In other words, for each input x, we return $f_{NN}(x)$ as an estimate for the desired value $f(x)$.
- The main advantage of this technique is that:
  - while the training of a neural network may take a significant amount of time,
  - e.g., the training for the Large Language Models like ChatGPT took years,
  - once the network is trained, it produces its results very fast.

4. Traditional way of solving such systems, its limitation, and how AI can help (cont-d)

- Machine learning has been successfully applied:
  - to many practical problems.
  - in particular, to problems requiring solution of systems of partial differential equations.
- As a result, we have the following new method of solving the corresponding problems.
- First, we use traditional numerical methods to solve several cases of a given system.
- This way, we find the outputs $y^{(1)}, \ldots, y^{(N)}$ corresponding to several inputs $x^{(1)}, \ldots, x^{(N)}$.

5. Traditional way of solving such systems, its limitation, and how AI can help (cont-d)

- Then, we train a neural network on the resulting pairs

$$(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)}).$$

- After that:
  - given any input x,
  - we apply the trained neural network to this input and
  - return the result $f_{NN}(x)$ of applying this neural network as the solution $y = f(x)$ to the given system of equations.

6. Remaining challenge

- Experience has shown that:
  - to get reliable results when solving systems of partial differential equations,
  - it is often necessary to use double precision (or sometime even higher precision) in all computations.
- Because of this:
  - when the practitioners train the neural network to solve these equations,
  - they often use at least 32-bit precision to calculate all the weights.
- In many cases, the resulting network produces the desired result $y = f(x)$ and produces it fast.
- However, there a remaining challenge,
- Storing 32 or more bits of information for each weight means that overall, we use a large portion of memory and use a lot of energy.

7. Remaining challenge (cont-d)

- A general experience of machine learning has shown that we do not need such high precision for all the weights:
  - some small weights can be just reduced to 0, and
  - for some weights, we can use their 8-bit approximations and still get the desired accuracy of the final result y.
- It is desirable to use lower bounded precision for computing weights – while still maintaining the desired accuracy of the final result.

8. How can we predict the result of bounding weight precision? How can we find the optimal precisions?

- Rigorous tools for analyzing the effect of floating-point precision on neural network inference have been developed.
- These tools use interval and affine arithmetic to provide guaranteed error bounds for a given uniform precision level.
- However, determining the optimal *per-weight* precision allocation remains largely done on a trial-and-error basis; we:
  - try some combinations of bounded precisions and
  - see whether the accuracy remains good.
- This is a very time-consuming process, and the results are often far from impressive.

9. How can we predict the result of bounding weight precision? How can we find the optimal precisions (cont-d)

- It is therefore desirable to be able:
  - to predict how bounded precisions will affect the accuracy of the computation result
  - without having to test the neural network every time, and
  - to find the optimal precisions.
- In the talk, we provide answers to both these questions.
- We provide an algorithm that:
  - given proposed reduced precisions,
  - estimates the effect of these reduced precisions on the accuracy of the computation result.

10. How can we predict the result of bounding weight precision? How can we find the optimal precisions (cont-d)

- We also provide an algorithm that:
  - given the desired accuracy of the result,
  - produces the optimal weight precisions,
  - i.e., weight precisions that minimize the overall number of bits and thus,
  - weigh precisions that minimize the use of resources such as memory and energy consumption.
- Of course, what we will describe are *general* algorithms that provide the proof of the concept.
- As usual with algorithms, a lot of additional work is needed to make these algorithms practically efficient and easy to use.
- What we show is that this is all doable.

11. How accuracy of y depends on the weight's precision: general analysis

- Each value y in the numerical computation result depends on all the weights $w_1, \ldots, w_n$: $y = f(w_1, \ldots, w_n)$.
- When we use k_i-bit precision to represent the i-th weight, we thus replace the original weight w_i with a rounded value $\widetilde{w}_i = w_i + \Delta w_i$.
- Here, Δw_i takes values from the interval $\left[-2^{-(k_i+1)}, 2^{-(k_i+1)}\right]$.
- Usually, the value Δw_i is uniformly distributed on this interval, and the values Δw_i corresponding to different i are independent.
- In this case, for each i, the mean value of Δw_i is 0, and the standard deviation is equal to
$$\sigma_i = \frac{1}{\sqrt{3}} \cdot 2^{-(k_i+1)}.$$
- When we use the modified weights instead of the original weights, we get the value
$$\widetilde{y} = f\left(\widetilde{w}_1, \ldots, \widetilde{w}_n\right) = f(w_1 + \Delta w_1, \ldots, w_n + \Delta w_n).$$

12. How accuracy of y depends on the weight's precision: general analysis (cont-d)

- This value differs from the original value $y = f(w_1, \ldots, w_n)$ by the difference

$$\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y = f(w_1 + \Delta w_1, \ldots, w_n + \Delta w_n) - f(w_1, \ldots, w_n).$$

- The values Δw_i are much smaller than the weights themselves.
- So, we can safely ignore terms that are quadratic or higher order with respect to Δw_i.
- Thus, we can expand the above expression in Taylor series in terms of Δw_i and ignore quadratic and higher order terms in this expansion.
- As a result, we get a linear expression

$$\Delta y = \frac{\partial f}{\partial w_1} \cdot \Delta w_1 + \ldots + \frac{\partial f}{\partial w_n} \cdot \Delta w_n.$$

- All the values Δw_i are independent and have mean 0.

13. How accuracy of y depends on the weight's precision: general analysis (cont-d)

- So, the mean value of this expression is 0, and the variance V of Δy is equal to:

$$V = E\left[(\Delta y)^2\right] = \left(\frac{\partial f}{\partial w_1}\right)^2 \cdot \sigma_1^2 + \ldots + \left(\frac{\partial f}{\partial w_n}\right)^2 \cdot \sigma_n^2.$$

- The value Δy is the sum of a large number of small independent random variables.
- So by the Central Limit Theorem, we can safely assume that the value Δy is normally distributed.
- Tthe distribution is normal and its mean is 0.

14. How accuracy of y depends on the weight's precision: general analysis (cont-d)

- So, to get bounds on Δy with certain confidence, we can have the bound $k_0 \cdot \sqrt{V}$, where k_0 depends on the desired confidence level:
  - for confidence level 95% we can take $k_0 = 2$;
  - for confidence level 99.9%, we can take $k_0 = 3$;
  - for confidence level $1 - 10^{-8}$, we can take $k_0 = 6$, etc.
- The above formula describes the variance corresponding to a single input x.
- To get the variance σ^2 corresponding to all possible inputs, we should take the mean value $E_x[V]$ of the expression V over all the inputs x:

$$\sigma^2 = E_x[V] = E_x\left[\left(\frac{\partial f}{\partial w_1}\right)^2\right] \cdot \sigma_1^2 + \ldots + E_x\left[\left(\frac{\partial f}{\partial w_n}\right)^2\right] \cdot \sigma_n^2.$$

15. Where can we get the partial derivatives and their expected values?

- To use the above formulas to predict the effect of limited weigh precision on the overall accuracy, we need to know:
  - the values of the corresponding partial derivatives with respect to w_i, and
  - the expected values of their squares.
- Where can we get this information?
- To see where we can get these weights, let us recall that the weights in a neural network are determined by backpropagation.
- On each training step, each weight changes according to the following gradient descent formula:

$$w_i \to w_i - \alpha \cdot \frac{\partial J}{\partial w_i}.$$

16. Where can we get the partial derivatives and their expected values (cont-d)

- Here J is the objective function, e.g.,
$$J = \sum_k \left(y_{NN}\left(x^{(k)}\right) - y^{(k)} \right)^2.$$
- Thus, in the process of training, we already have the computed values of the partial derivative of J with respect to w_i.
- The value J depends on the result y of computations in a known way.
- So, by the chain rule, we have
$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial w_i}.$$
- Hhere the derivative of J with respect to y is easy to compute.
- For example, for the least squares objective function, we have
$$\frac{\partial J}{\partial y} = 2 \cdot \left(y - y^{(k)} \right).$$

17. Where can we get the partial derivatives and their expected values (cont-d)

- Thus, based on the known derivatives of J, we can use the above formula to find the desired derivatives of y:

$$\frac{\partial y}{\partial w_i} = \left(\frac{\partial J}{\partial y}\right)^{-1} \cdot \frac{\partial J}{\partial w_i}.$$

- Of course, the actual values of the derivatives depend on the values of the weights and thus, change from cycle to cycle.
- We need to only take into account the partial derivatives at the last cycle(s) of training, when the weights are close to the final ones.

18. Comment

- We note that the quantities $V_i = E_x \left[\left(\frac{\partial y}{\partial w_i} \right)^2 \right]$:
  - are proportional to the diagonal elements of the Gauss-Newton approximation to the Hessian of the loss function;
  - these terms have been used in the neural network pruning literature to estimate the impact of removing individual weights.

19. How can we find the optimal weight precisions?

- We want to minimize the overall number of bits.
- As we have mentioned, when we use k_i bits, then

$$\sigma_i = \frac{1}{\sqrt{3}} \cdot 2^{-(k_i+1)}.$$

- By taking binary (base-2) logarithm of both sides, we get

$$\log_2(\sigma_i) = -\frac{1}{2} \cdot \log_2(3) - (k_i + 1), \text{ so}$$

$$k_i = -\log_2(\sigma_i) - \frac{1}{2} \cdot \log_2(3) - 1 \text{ and}$$

$$k_i = -\log_2(\sigma_i) + b \text{ for constant } b = -\frac{1}{2} \cdot \log_2(3) - 1.$$

- So $\sum_i k_i = -\sum_i \log_2(\sigma_i) + n \cdot b$.

20. How can we find the optimal weight precisions (cont-d)

- Where a function attains its minimum does not change if we:
  - add a constant to a minimized function and/or
  - multiplying the minimized function by a positive constant.
- Thus, minimizing the sum of the numbers of bits is equivalent to minimizing the sum of the $-\log_2(\sigma_i)$ terms.
- Our constraint is that:
  - either the average approximation error σ^2 should not exceed some given value σ_0^2,
  - or the confident upper bound $k_0 \cdot \sigma$ should not exceed some given value Δ_0 – which is equivalent to $\sigma^2 \le \sigma_0^2 \stackrel{\text{def}}{=} \dfrac{\Delta_0^2}{k_0^2}$.
- In both cases, we can use the Lagrange multiplier approach.

21. How can we find the optimal weight precisions (cont-d)

- Then, the constraint optimization problem reduces to the problem of minimizing the expression:

$$-\sum_i \log_2(\sigma_i) + \lambda \cdot \sum_i V_i \cdot \sigma_i^2, \text{ where } V_i \stackrel{\text{def}}{=} E_x\left[\left(\frac{\partial y}{\partial w_i}\right)^2\right].$$

- Here, λ is the Lagrange multiplier that needs to be determined by the condition $\sigma^2 = \sigma_0^2$.
- Differentiating the above expression with respect to an unknown σ_i, we conclude that $-\frac{1}{\ln(2) \cdot \sigma_i} + 2 \cdot \lambda \cdot V_i \cdot \sigma_i = 0$, i.e., that:

$$\sigma_i^2 = \frac{c}{V_i}, \text{ where } c \stackrel{\text{def}}{=} \frac{1}{2 \cdot \ln(2) \cdot \lambda}.$$

- Here, $V_i \cdot \sigma_i^2 = c$.

22. How can we find the optimal weight precisions (cont-d)

- Thus, the condition that $\sigma^2 = \sigma_0^2$ becomes

$$\sigma^2 = \sum_i V_i \cdot \sigma_i^2 = n \cdot c = \sigma_0^2.$$

- So $c = \dfrac{\sigma_0^2}{n}$; thus, $\sigma_i = \sqrt{\dfrac{c}{V_i}} = \dfrac{\sigma_0}{\sqrt{n \cdot V_i}}$, and:

$$\log_2(\sigma_i) = \log_2(\sigma_0) - \frac{1}{2} \cdot \log_2(n) - \frac{1}{2} \cdot \log_2(V_i).$$

- So, the optimal precision k_i for each weight w_i is

$$k_i = -\log_2(\sigma_0) + \frac{1}{2} \cdot \log_2(n) + \frac{1}{2} \cdot \log_2(V_i) - \frac{1}{2} \cdot \log_2(3) - 1.$$

23. Remark

- At first glance, the $+\frac{1}{2} \cdot \log_2(n)$ term may seem to suggest that larger networks require *more* bits per weight.
- However, it is well known empirically that larger models are more amenable to quantization.
- Our formula is consistent with this observation:
  - in a larger network, the sensitivity V_i of any individual weight is expected to be smaller,
  - since the output depends on more weights.
- If the decrease in V_i with n is sufficiently fast, the $\log_2(V_i)$ term dominates and the net required precision per weight decreases.
- The precise relationship between V_i and n depends on the network architecture and is an interesting direction for future investigation.
- Now, we are ready to describe the resulting algorithms.

24. 1st algorithm: estimating the effect of different weight precisions on the accuracy of the computation result

- We have a trained neural network, and we have the precisions k_i that we plan to assign to each neuron:
  - the value $\widetilde{y}$ computed with thus bounded precisions will be somewhat different from
  - the value y returned by the neural networks in which each neuron has the original high precision.
- We would like to estimate:
  - either the mean squared value σ^2 of the inaccuracy $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$ caused by these bounded precisions
  - or the upper bound Δ on the resulting inaccuracy – that will be maintained with a given confidence level.

25. 1st algorithm: estimating the effect of different weight precisions on the accuracy of the computation result (cont-d)

- To perform these estimations:
  - we need, during the last cycle(s) of the neural network training, to record the partial derivatives $\frac{\partial J}{\partial w_i}$ of the objective function J;
  - these values that are computed and used during the backpropagation computations;
  - then, we need to use these values – and the easy-to-compute values $\frac{\partial J}{\partial y}$ – to compute the values $\frac{\partial y}{\partial w_i} = \left(\frac{\partial J}{\partial y}\right)^{-1} \cdot \frac{\partial J}{\partial w_i}$;
  - finally, for each i, we need to take the arithmetic average V_i of the squares of all the values of this derivative.
- What if this was not done during the original training?
- Then, we need to perform at least one cycle of additional training with the available pairs $\left(x^{(k)}, y^{(k)}\right)$.

26. 1st algorithm: estimating the effect of different weight precisions on the accuracy of the computation result (cont-d)

- We will thus collect the corresponding values of the partial derivatives; hhen:
  - the expected mean squared value σ^2 is equal to

$$\sigma^2 = \sum_i V_i \cdot \sigma_i^2, \text{ where } \sigma_i = \frac{1}{\sqrt{3}} \cdot 2^{-(k_i+1)};$$

  - the upper bound Δ can be computed as $k_0 \cdot \sigma$.
- Here the coefficient k_0 depends on the desired level of confidence; e.g.:
  - $k_0 = 2$ for confidence 0.95,
  - $k_0 = 3$ for confidence 0.999, and
  - $k_0 = 6$ for confidence $1 - 10^{-8}$.

27. 2nd algorithm: finding optimal weight precisions for a given accuracy of the computation result

- In this case, we are given:
  - either the upper bound σ_0^2 on the mean squared value σ^2 of the inaccuracy $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$ caused by these bounded precisions
  - or the upper bound Δ_0 on the resulting inaccuracy that will be achieved with a given degree of confidence.
- Out of all the weight precisions k_i that guarantee the desired accuracy, we need to find the values that:
  - minimize the overall number of bits, and thus,
  - minimize the use of resources such as memory and energy.
- In this case also, we first need to make sure that during the training, we compute the corresponding values V_i.
- If we are given the bound Δ_0, then we need to compute the equivalent bound $\sigma_0 = \Delta_0/k_0$.

28. 2nd algorithm: finding optimal weight precisions for a given accuracy of the computation result (cont-d)

- Here k_0 depends on the desired level of confidence.
- Then, we compute the optimal precisions k_i by using the above formula:

$$k_i = -\log_2(\sigma_0) + \frac{1}{2} \cdot \log_2(n) + \frac{1}{2} \cdot \log_2(V_i) - \frac{1}{2} \cdot \log_2(3) - 1.$$

29. Comment

1) If for some small weights, their values with accuracy k_i becomes 0, this means that the corresponding connection is not needed.

- If for some neuron, all the weights coming out of this neuron are small, this means that we do not need this neuron at all.

2) The simplest version of the above problem is when:

  - we cannot change the precisions,
  - but we can ignore some weights, i.e., we can make some weights equal to 0.

- In this setting, we can ask the two similar questions:

  - How can we estimate the effect of this weight elimination on the result of the computations?
  - For a given accuracy of the result, which weights should we eliminate to minimize the use of resources?

30. Comment (cont-d)

- In this case, if denote by D, the set of indices of deleted weights, then the difference $\Delta y = \widetilde{y} - y$ takes the form

$$\Delta y = -\sum_{i \in D} \frac{\partial y}{\partial w_i} \cdot w_i.$$

- It makes sense to assume that the derivatives are kind of randomly distributed, and that they are independent.
- In this case, the mean squared value σ^2 of Δy is equal to

$$\sigma^2 = \sum_{i \in D} V_i \cdot w_i^2.$$

- This formula provides the answer to the first question.
- In the second question, the number of saved bits is proportional:
  - to the number of the deleted weights,
  - i.e., to the number of weights in the set D.

31. Comment (cont-d)

- What if we keep a weight i with a larger value of the product $V_i \cdot w_i^2$ and delete a weight j with the smaller value $V_j \cdot w_j^2$?
- Then, by swapping these two weights, we can improve the situation.
- Thus, to come up with the optimal arrangement, we need to sort all the weights in the increasing order of the product $V_i \cdot w_i^2$.
- Then, we:
  - delete the ones with the smallest product $V_i \cdot w_i^2$ and
  - continue deleting the weights one by one until the sum of the corresponding small products reaches the required value σ_0^2.

32. Comment (cont-d)

- This pruning criterion is closely related:
  - to the *Optimal Brain Damage* approach, which uses the diagonal of the Hessian of the loss function to rank weights for elimination
  - and to the *Optimal Brain Surgeon*, which extends this to off-diagonal terms.
- The above mixed-precision allocation problem can be seen as a generalization of this pruning setting, where:
  - instead of a binary keep/remove decision,
  - each weight is assigned a variable number of bits.

33. Acknowledgments

This work was supported by:

- the AT&T Fellowship in Information Technology,
- the Institute for Risk and Reliability, Leibniz Universitaet Hannover, Germany,
- the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) Focus Program SPP 100+ 2388, Grant Nr. 501624329,
- the European Union under the project ROBOPROX (No. CZ.02.01.01/00/22 008/0004590),
- the Center of Excellence in Econometrics, Faculty of Economics, Chiang Mai University, Thailand,
- the Ho Chi Minh City University of Banking, Vietnam, and
- Thang Long University, Hanoi, Vietnam.