Optimization of neural network computations by reducing the precision of numbers

Medric Bruel Djeafea Sonwa[*]

[*]medric49@gmail.com
[+]Grenoble INP - Ensimag, 38000 Grenoble, France

ABSTRACT

The current generation of neural networks requires a lot of resources in terms of storage and computing time. Due to the high number of parameters needed in this networks and the high precision of numbers (integers and floating point numbers) representing the weights and bias, the computations take a lot of time. it's hard to imagine integrating this generation of neural networks in embedded systems, such as drones or microcontrollers that do not have a high power. Having noted this problem, new research has looked into the possibility of reducing the number of bits on which neural networks parameters are represented in order to reduce computing time and memory requirements. Thus, in this paper we present the formulations mathematics of two of them: the binarization which consists in reducing weights and outputs in the binary set $\{-1,+1\}$, and the ternarization which consists in reducing weights into the set $\{-1,0,+1\}$. We explain in this paper how to implement these types of neural networks, efficient in terms of memory and computation time, based on research and open source projects related to this issue. We also present a learning algorithm adapted to these neural networks. And finally, we're redefining the arithmetic operations to fit the operands of the set $\{-1,+1\}$, so that they can be used in simplified systems that operate in binary.

1 Introduction

The use of neural networks to solve complex problems has particularly increased in recent years. This increase has led to an evolution in the architectures of neural networks in order to make them more efficient. These neural networks are built from mathematical entities called neurons. A neuron being an object which from input data produces an output result, using a function called the activation function[1]. The calculation of the final result of these neural networks is done through numerous machine operations which, in general, have a non-negligible cost in time and energy. Since current neural networks (NN) are built on the basis of floating-point numbers, the arithmetic operations compatible with these numbers are all the more costly in time and energy[2]. Depending on the number of operations performed, this time becomes enormous and it is more than necessary to reduce it. Moreover, it becomes impossible to import these networks built in embedded systems.

A new method of implementing neural networks was then thought, it is the binarization of weights and biases of neurons and possibly of the results of activation functions[3]. The idea that we want to explore in this paper is the possibility of optimizing computational operations in neurons by reducing the number of representation bits of neural network parameters. In this paper we make our contribution through the following points:

- We present binary neural networks and we explain the mathematical notion behind binary neural networks, while presenting the problems behind the parameters of these binary neurons. Thus, we explain the process of binarization of the parameters and outputs, of our network, in the set $\{-1,+1\}$.
- Taking into account the mathematical modeling and the constraints related to the computation of the error in this binary context, we present a learning algorithm for binary neural networks sufficiently robust to allow a good optimization of weights and biases.
- We extend this optimization study to ternary neural networks. We study the mathematical model that will allow the implementation of neural networks whose weights and outputs belong to the set $\{-1,0,+1\}$.

- We present a method for learning ternary networks, based on a defined mathematical model and facing the problem of ternarization of weights.
- We reformulate addition and multiplication operations for use in binary neural networks. These operations are presented in such a way that they can be used in lightweight systems that work on 1-bit representable numbers. In our case, these numbers will be the weights and outputs of our networks.

We present two inspiring projects on which this study is based and which more or less implements the concepts discussed: the project *brevitas*[4] and the project *qnn-inference-examples*[5].

2 Principle of binary neural networks

The first approach for the optimization of computations in neural networks is the binarization of weights and biases of neurons, and the binarization of their outputs. In other words, weights and biases can only have 2 possible values, which limits their representation to one bit. Neuron outputs can also be transformed so that they are represented on a single bit. This leads to a loss of information, so depending on the precision we wish to have and the execution time we can tolerate, we can determine whether or not we also wish to convert the outputs of the neurons on a certain number of bits.

Knowing that weights and biases have been reduced to a number of bits, as well as the input data, versions of arithmetic operations (addition, subtraction, etc.) can be designed specifically for operations with a limited number of values[6]. The goal is to speed up the execution speed of these operations. Indeed, an addition operation designed for 2-bit operands will have a lower cost than an operation designed for 8-bit operands.

2.1 Mathematical formulation

When training neural networks, the real values of the weights (including bias) are not limited, it is necessary to convert these values to 2 possible values -1 or 1. Let us note w the real value of the weight of a neuron, and w^b the conversion on the set $\{-1,+1\}$ of this weight.

The first function that retrieves w^b is the *deterministic binarization*[3]:

$$w^b = \begin{cases} +1 \ if \ w \geq t_w \\ -1 \ if \ w < t_w \end{cases}$$

Where t_w is a threshold value that is a function of w, which is either fixed or variable and whose value can vary during the learning process of the neuron. It is common to set t_w to 0 independent of w.

Another method of determining w_b is the *stochastic binarization*[3]:

$$w^b = \begin{cases} +1 \ with \ prob. \ \sigma(w) \\ -1 \ with \ prob. \ 1-\sigma(w) \end{cases}$$

Where σ is the *sigmoid* function given by:

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

The *determistic binarization* is preferable in our context because it is easy to implement and requires less machine calculation. However, with this method, the problem of threshold choice arises. Although we can directly set it to 0, we can also find an optimal value for it by calculating the error committed on its choice. This error is :

$$E_w(t_w) = (w - w^b)^2 = (w - (-1)^{1_{w<t_w}})^2$$

And so, an optimal choice of t_w is then :

$$t_w^* = \underset{t_w \in]-\varepsilon,\varepsilon[}{\arg\min} E_w(t_w)$$

Where *epsilon* is a positive number close to zero.

Let's consider the i^{th} layer of an NN, whose weight matrix is W_i and the bias vector is b_i. Let's consider the input x, we have : $z = W_i x + b_i$.

The final real-valued vector of this layer is : $a = activation(z)$

Where $activation$ is a the activation function[7] of this layer. It can be $sigmoid$, $tanh$, $ReLU$ etc.

So, to get the value, in the set $-1, +1$, of a, a binarization is applied to that result:

$$a^b = \begin{cases} +1 \ if \ a \geq 0 \\ -1 \ if \ a < 0 \end{cases}$$

Another practice would be to replace the activation function by binarization, our expression would become :

$$a^b = a = \begin{cases} +1 \ if \ z \geq 0 \\ -1 \ if \ z < 0 \end{cases}$$

2.2 Learning process

The learning of a binarized NN is done by considering the particular characteristic of the derivative of the loss functions[8].

Let us note by $Loss()$ the loss function that we will use, $Binarize()$ the binarization function that will be used to translate the weights and outputs into the $\{-1, +1\}$ set. Let us note by J the value of the error given by the loss function. And finally, given x any parameter, Δ_x designates the partial derivative[9] of the error with respect to x, thus $\Delta_x = \frac{\partial J}{\partial x}$.

Thus, a proposal for a learning algorithm for a binary network, inspired by the one provided in the paper *Binarized Neural Networks*[3], is this:

Algorithm 1: Algorithm for learning a binary neural network

```
for $k = 1$ to $L$ do
    $W_k^b \leftarrow Binarize(W_k)$
    $a_k \leftarrow W_k^b a_{k-1}^b$
    if $k < L$ then
        $a_k^b \leftarrow Binarize(a_k)$
    end
end
$J \leftarrow Loss(a_L, y)$
$\Delta_{a_L} \leftarrow \frac{\partial J}{\partial a_L}$
for $k = L$ to $1$ do
    if $k < L$ then
        $\Delta_{a_k} \leftarrow \Delta_{a_k^b} * 1_{|a_k| \leq 1}$
    end
    $\Delta_{a_{k-1}^b} \leftarrow W_k^b \Delta_{a_k}$
    $\Delta_{W_k^b} \leftarrow a_{k-1}^b \Delta_{a_k}^T$
end
for $k = 1$ to $L$ do
    $W_k \leftarrow W_k - \lambda * \Delta_{W_k^b}$
end
```

The learning algorithm does not really change from the one known by everyone. It is in fact a double learning, in which the neural network has improved its real parameters, and its binary parameters, and the neural network has improved its binary parameters.

When the learning process is complete, what we get are parameters $W_k^b \forall \ k = 1 \ to \ L$. These weights (and biases) constitute our new lightened neural network, because they are represented on 1 bit. We can then adopt the formatting:

$$w^b = \begin{cases} 1_2 \; if \; w^b = 1 \\ 0_2 \; if \; w^b = -1 \end{cases}$$

Where x_2 is a number written in base 2. For example 10_2, 11010_2, 100101_2 etc. Since our parameters will now be translated into base 2, we will be defining new operations for addition, subtraction, etc.

3 Ternary neural network approach

The use of binary neural networks greatly reduces the time and complexity of the calculations, but in exchange decreases the accuracy of the neural networks. Thus, to improve this, while keeping the complexity of operations reduced, The principle is that the weights and biases of neurons are no longer represented on 1 bit at values of -1 or 1, but rather on 2 bits and with values in the set $\{-1, 0, +1\}$. As before, arithmetic operations constructed for operands with values in the $\{-1, 0, +1\}$ set are necessary to ensure speed of execution.

3.1 Mathematical formulation

The approach of the ternarization of the parameters is similar to that of the binarization. However, it is necessary to to take into account the thresholds. Let's consider a parameter w of any neuron, the process to ternarize it is the following[10] :

$$w^b = \begin{cases} -1 \; if \; w < t_{w,1} \\ 0 \; if \; t_{w,1} \leq w \leq t_{w,2} \\ +1 \; if \; t_{w,2} < w \end{cases}$$

Where $t_{w,1}$ and $t_{w,2}$ are the two thresholds that indicate a change in value.

3.2 Learning Process

The learning algorithm of a ternary neural network is similar to that of a BNN. The difference is the way to modify of weights and outputs. Indeed, a $Ternarize()$ function will be applied, instead of $Binarize()$. But there is a problem, what thresholds will we use? The answer to this question is complex, because a wrong choice of thresholds would lead to a big loss of information. Contrary to BNNs which have only one threshold and generally set at 0, the thresholds of BNNs must vary. These are moreover subject to this constraint : $\begin{cases} t_{w,1} \leq 0 \\ t_{w,2} \geq 0 \end{cases}$

A method used to calculate a suitable value for the thresholds would be to go through the error calculation on the approximation of the weight w by w^b[11]. Thus, the error made on the approximation of w by w^b is:

$$E_w(t_{w,1}, t_{w,2}) = (w - w^b)^2 = (w - Ternarize(w, t_{w,1}, t_{w,2}))^2$$

And so the new optimal thresholds $t^*_{w,1}$ and $t^*_{w,2}$ resulting from this error are given by :

$$t^*_{w,1}, t^*_{w,2} = \underset{t_{w,1} \leq 0, t_{w,2} \geq 0}{\arg\min} \; E_w(t_{w,1}, t_{w,2})$$

4 Adaptation of arithmetic operations

As previously mentioned, in order to optimize the CPU operation time, it is necessary to define new mathematical operations for our sets $\{-1, +1\}$ and $\{-1, 0, +1\}$. Once our neural network has finished learning and has been implemented in an embedded system, the neural network now considers its parameters as binary or ternary. In the case of BNN, let's recall that the representation adopted by the weights :

$$w^b = \begin{cases} 1_2 \; if \; w^b = 1 \\ 0_2 \; if \; w^b = -1 \end{cases}$$

This representation allows us to define the arithmetic operations $\bar{+}$ (representing $+$) and $\bar{*}$ (representing $*$) in the following way:

Logical operation	Mathematical translation
$1_2 \bar{+} 1_2 = 1_2$	$1+1=2\approx 1$
$1_2 \bar{+} 0_2 = 0_2 \bar{+} 1_2 = 1_2$	$1+(-1)=0\approx 1$
$0_2 \bar{+} 0_2 = 0_2$	$(-1)+(-1)=-2\approx -1$

Logical operation	Mathematical translation
$1_2 \bar{*} 1_2 = 1_2$	$1*1=1$
$1_2 \bar{*} 0_2 = 0_2 \bar{*} 1_2 = 0_2$	$1*(-1)=-1$
$0_2 \bar{*} 0_2 = 1_2$	$(-1)*(-1)=1$

It follows from these tables that, the operation $\bar{+}$ is actually the logical *or*, and the operation $\bar{*}$ is the logical *xnor*. So we have:

$$\begin{cases} x \bar{+} y = x \text{ or } y \\ x \bar{*} y = x \text{ xnor } y \end{cases}$$

The operations $\bar{+}$ and $\bar{*}$, being defined, can be used as addition and multiplication operations in our embedded systems, the weights and outputs now being 1-bit numbers.

Thus, let's consider a neuron implemented in a device running with 1-bit numbers. The weight vector of the neuron is $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ and the bias is b.

The result of the activation for an input $a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$ by this neuron is given by :

$$z = a_1 \bar{*} w_1 \bar{+} a_2 \bar{*} w_2 \bar{+} b = (a_1 \text{ xnor } w_1) \text{ or } (a_2 \text{ xnor } w_2) \text{ or } b$$

5 Conclusion

In this paper, we have studied the mathematical model behind binary and ternary neural networks and the problems related to these new architectures. This allowed us to approach an algorithm that would allow the learning of these types of networks, whose weights and outputs are restricted values, all this with the sole aim of optimizing the storage and computation time of these neural networks. Current research shows that this problem is not in vain, and deserves more attention. Indeed, this new generation of neural networks would undoubtedly be a very good way to improve the performance of our embedded systems, our chips and microcontrollers or any other equipment that does not have a high computing power. It is a non-negligible way to give intelligence to all our equipment.

References

1. Abraham, A. Artificial neural networks. *Handb. measuring system design* 901–903 (2005).

2. Goldberg, D. What every computer scientist should know about floating-point arithmetic. *ACM Comput. Surv. (CSUR)* **23**, 5–48 (1991).

3. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R. & Bengio, Y. Binarized neural networks. In *Advances in neural information processing systems*, 4107–4115 (2016).

4. Xilinx. Brevitas: quantization-aware training in pytorch. https://xilinx.github.io/brevitas/.

5. Jupyter notebook examples on image classification with quantized neural networks. https://github.com/maltanar/qnn-inference-examples.

6. MacSorley, O. L. High-speed arithmetic in binary computers. *Proc. IRE* **49**, 67–91 (1961).

7. Karlik, B. & Olgac, A. V. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *Int. J. Artif. Intell. Expert. Syst.* **1**, 111–122 (2011).

8. Bottou, L. Stochastic gradient learning in neural networks. *Proc. Neuro-Nimes* **91**, 12 (1991).

9. Speelpenning, B. Compiling fast partial derivatives of functions given by algorithms. Tech. Rep., Illinois Univ., Urbana (USA). Dept. of Computer Science (1980).

10. Alemdar, H., Leroy, V., Prost-Boucle, A. & Pétrot, F. Ternary neural networks for resource-efficient ai applications. In *2017 International Joint Conference on Neural Networks (IJCNN)*, 2547–2554 (IEEE, 2017).

11. Mellempudi, N. *et al.* Ternary neural networks with fine-grained quantization. *arXiv preprint arXiv:1705.01462* (2017).