HOKKAIDO UNIVERSITY

Title	Smart hardware architecture with random weight elimination and weight balancing algorithms
Author(s)	Ali, Emiliano J.; Amemiya, Yoshiki; Akai-Kasaya, Megumi et al.
Citation	Nonlinear theory and its applications, IEICE, 13(2), 336-342 https://doi.org/10.1587/nolta.13.336
Issue Date	2022
Doc URL	https://hdl.handle.net/2115/85561
Rights	Copyright ©2022 The Institute of Electronics, Information and Communication Engineers
Type	journal article
File Information	13_336.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

NOLTA, IEICE

Paper

Smart hardware architecture with random weight elimination and weight balancing algorithms

Emiliano J. Ali [1 a)], *Yoshiki Amemiya* [1], *Megumi Akai-Kasaya* [1,2], *and Tetsuya Asai* [1]

[1] *Graduate School of Information Science and Technology, Hokkaido University Kita 14, Nishi 9, Kita-ku, Sapporo, Hokkaido 060-0814, Japan*

[2] *Graduate School of Science, Department of Chemistry, Osaka University 1-1 Machikaneyama, Toyonaka, Osaka 560-0043, Japan*

[a)] *ali.emiliano.3c@ist.hokudai.ac.jp*

Received October 18, 2021; Revised December 18, 2021; Published April 1, 2022

Abstract: Reducing the number of connections in hardware artificial neural networks, as compared with their software counterparts, can result in a drastic reduction in costs, because the reduction translates into utilizing fewer devices. This paper presents the demonstration of a method, by using simulations, to halve the amount of weights in a network while minimizing the accuracy loss. Additionally, the appropriate considerations for translating these simulation results to hardware networks are also detailed.

Key Words: hardware neural networks, random weight elimination, unsupervised learning, smart architecture, weight balancing.

1. Introduction

With recent advancements in artificial neural networks, the increasing complexity of the challenges at hand results in a notable increase in the number of neurons and layers involved in the networks. Although this might not be a significant issue when software networks are considered, because even though an increased number of neurons and weights affect the processing speed, the cost does not increase proportionally. However, when considering analog neural networks that must be implemented in hardware, the concern grows.

In hardware neural networks, the weight values are usually represented using conductance values that are strictly positive (e.g., in crossbar arrays with sense amplifier configurations,such as the schematic illustrated in **Fig.1-A**).Therefore, we are required to implement differential weights resulting in the duplication of weights of their software counterpart.

On this basis, determining a way to reduce the number of connections in a network while trying to maintain the highest processing accuracy possible is potentially key to producing economically and computationally efficient hardware artificial neural networks.

This study addressed this problem by adopting two algorithms named *Weight Elimination* and

Nonlinear Theory and Its Applications, IEICE, vol. 13, no. 2, pp. 336–342 ©IEICE 2022 DOI: 10.1587/nolta.13.336

Weight Balancing,resulting in, what we called, a *Smart Hardware Architecture* [1] [2] represented in **Fig.1-B**. This can effectively halve the weights in a hardware network with minimal losses in processing accuracy. Here, an unsupervised learning algorithm with a well-known dataset utilizing a simple perceptron architecture, and a supervised Deep-Belief Network (DBN) configuration, are tested.

Additionally, because the calculations are performed in software simulations, the considerations required to translate this architecture into hardware are also introduced and implemented in the simulation to represent the system accurately. Notably, the influence on the amplifier error due to the addition of a conductance (g_0 in **Fig.1**) used for biasing the operational amplifier input.

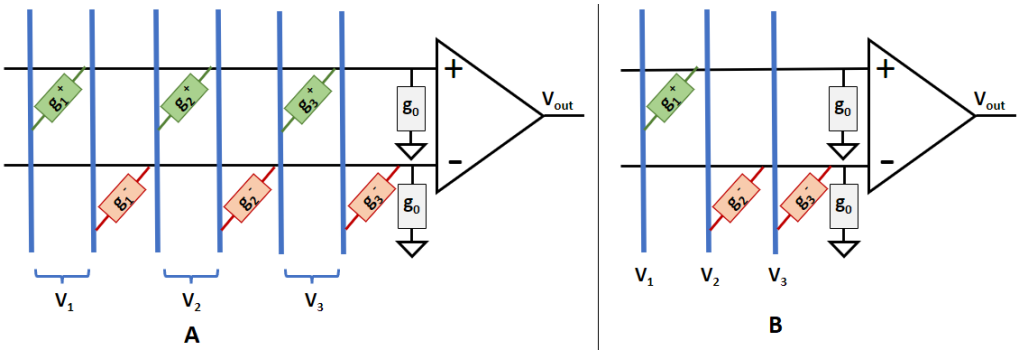

Fig. 1. Schematics of hardware ANN pre and post weight elimination.

2. Methodology

To indicate how the reduction in the number of connections is performed, two individual algorithms must be explained. Firstly, the *Weight Balancing Algorithm* (WB) is detailed; the purpose of this algorithm is to create an equivalent of hardware differential weights into software. Subsequently, the *Random Weight Elimination* (RWE) algorithm is shown; this is the main method that enables the reduction of the number of weights in the network. Additionally, we describe how the amplifier error is affected by the g_0 conductance and how to determine a value that minimizes this error while maintaining the power consumption as low as possible.

2.1 Weight Balancing

When a neural network is implemented in hardware, the weights are represented as the difference between two positive conductance values. However, the weights of software neural networks are a mere number with no physical significance; they are not comprised of a positive and negative value combination.

In this regard, in order to compare a normal learning algorithm with no weight elimination and another algorithm that only deals with either the positive or the negative component of the weight, we are required to find to determine how to divide the software weight numbers into positive and negative splits.

Therefore, the *Weight Balancing* algorithm was developed as a solution. In this algorithm, we create an auxiliary matrix (called **Rating matrix** onward) that represents the magnitude of the shift of each individual weight value in relation to the combined weight. This matrix will be set with random values within a range for our case since our starting point is a software neural network, but in principle this matrix can be designed according to the initial network's weight values (if the nature of the network is known beforehand).

In the case of hardware neural networks this matrix is simply the representation of the balance between the positive and negative weight values. For example, if we have a weight of **10**, this can be represented by an infinite number of possibilities. The number could be represented as the combination of a positive weight of **11** and a negative weight of **1** (which would be represented with a rating of 0.1), or a positive weight of **1000** and a negative weight of **990** (with a rating of 99). In both cases,

the total weight is still 10; but when eliminating either of the weights, the shift in the network is different when going from 10 to 1000 or to -1. An example of a rating matrix is illustrated in **Fig.2**.

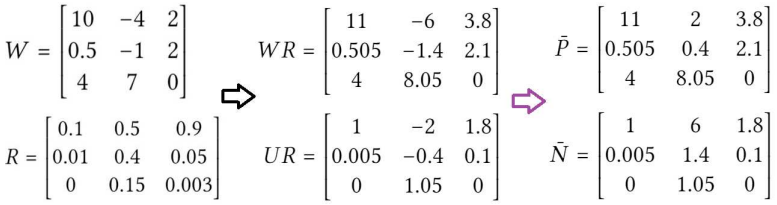

Fig. 2. Example of Rating matrix construction and weight separation.

After the matrix is constructed, the following operations are performed on the weight values:

1. The original weight value w_{ij} is multiplied by $1+r_{ij}$ where r_{ij} is the rating for the ith/jth element. This new value is called wr_{ij} and is stored in a new matrix called **WR**.So, $wr_{ij} = w_{ij} * (1 + r_{ij})$

2. Another matrix is created with the counterpart of this newly rated matrix, multiplying w_{ij} by r_{ij}. This value is called ur_{ij} and is stored in a matrix called **UR**.This is then: $ur_{ij} = w_{ij} * r_{ij}$

3. The matrices **WR** and **UR** are split into their positive and negative values, yielding four matrices,refered to as **P**,**N**,**p** and **n**, corresponding to the positive and negative values of **WR** and the positive and negative values of **UR** respectively.

4. By subtracting **n** from **P**; and **N** from **p**, we obtain the total positive and negative matrices, noted as $\bar{P}$ and $\bar{N}$: $\bar{P} = P - n$ and $\bar{N} = p - N$. Subsequently, we are left with the final result: $W = \bar{P} - \bar{N}$

After this process we end up with two matrices, as in the example illustrated in **Fig.2**; we have strictly positive matrix and strictly negative matrices that, when added, result in the original weight matrix that the process started with.

2.2 Random Weight Elimination algorithm

The *Random Weight Elimination* (RWE) algorithm is, as its name reflects, the elimination of either the positive or negative component of each weight in a random manner after WB. In a software network, this does virtually nothing, but in a hardware network, since **each weight** is constructed via the difference of two conductance values, the RWE reduces the complexity and size of the network by eliminating one of the components of the differential weight (halving the amount of devices needed to construct a network).

This process is remarkably simple; we start with the matrices resulting from the WB algorithm, and before performing any learning in the network, we eliminate one of the components for each weight (either positive or negative), and then subtract the matrices to yield a new starting weight matrix.

However, solely doing this would not suffice; it is required to perform a restriction during the learning process. This restriction does not allow the initial weights to decrease in absolute value to mimic a physical conductance, implying that , for example, if a weight solely has its negative part, it cannot become more positive because its positive component has been eliminated.

Even though this last assumption can be discussed depending on the device (since, for example, memristors can reduce their conductance [3]), in our case, the target hardware device is based on material electropolymerization [4], thus making it impossible to reduce the conductance once it has been increased.

The combination of these two processes, applying RWE to the network pre-learning and the restriction of weights during the machine learning operations, yields a software network that can mimic a hardware network with half of its components, and which its constructing devices' conductance can not be reduced.

2.3 RWE dependances

The RWE algorithm results are mainly influenced by two variables, the **network size** and the **rating matrix composition** (for the case of non-decremental conductance devices). For bigger networks, there is a smaller accuracy drop between the original and that with eliminated weights. The hypothesis is that bigger networks allow for additional weights to compensate the aforementioned strictly positive or negative restriction.

For the rating matrix, maximum ratings smaller than 1% result in virtually no partition and thus hold no significance for the algorithm, but high ratings result in a comparably high dispersion between the original and post-elimination weight values, hindering the learning accuracy. To set the optimal values for the rating matrix (in our case an optimal range for the random values), an analysis of the network must be performed. This implies that **before** using RWE, simulations to determine an optimal rating matrix must be done according to the network/dataset. Once the rating matrix is designed, the RWE algorithm can be applied. It is important to mention again that this holds true only for simulation of *non-decremental conductance networks* like the ones mentioned in Section 2.2. If a network with **variable conductance** (where the value can increase or decrease within a range) is simulated, **the rating matrix does not affect the learning significantly** since the absolute value of the weights can decrease due to the nature of the device.

2.4 Amplifier error

Since our focus is on hardware neural networks, we must consider the addition of the aforementioned resistor (denoted as g_0 in **Fig.1**). The addition of this conductance results in an error due to the amplifier "miscalculating" the output, implying that the expected output and the real amplifier output differ.

Utilizing the sense amplifier, we expect to obtain a digital high when the weighted sum of the positive input is higher than the weighted sum of the negative input, and a digital low when the opposite occurs. Thus, the expected equation that the amplifier should implement is $V_o = H\left[\sum_{i=1}^{n} V_i\left(g_{i+} - g_{i-}\right)\right]$

Where V_o is the output voltage, V_i is the ith input voltage, g_{i+} is the positive component of the ith weight, g_{i-} is the negative component of the ith weight and H is the transfer function of the amplifier. With the addition of the auxiliary conductance, instead of implementing this function, the amplifier performs the following computation:

$$V_o = H\left\{\left[\frac{1}{g_0 + \sum_{i=1}^{n} g_{i+}}\right]\sum_{i=1}^{n} V_i g_{i+} - \left[\frac{1}{g_0 + \sum_{i=1}^{n} g_{i-}}\right]\sum_{i=1}^{n} V_i g_{i-}\right\} \tag{1}$$

By comparing the outputs of these two equations when varying the input voltages and the values of g_0, we can measure the error induced in the sense amplifier due to the addition of the resistor.

However, before this calculation, it is important to detect how much amplifier error is acceptable for our RWE algorithm. Thereafter, the appropriate value of g_0 can be set. For this purpose,an artificial error was introduced during the learning process using the RWE algorithm. F or example, if the error is set to 0.5, every neuron in the network has a 50% of yielding an incorrect output (opposite to the expected output).

In a network with binary units, a 50% error chance in every unit is virtually random; therefore, an analysis was performed for an error rate between 0 and 0.5. The results of these analyses are illustrated in **Fig.3**.

In our case, an error of 0.0025 was determined to be the maximum admissible, this is due to higher error rates yielding low-quality images as a result. It is worth mentioning that this analysis was performed on the MNIST dataset; however, it should be performed on the desired dataset to check how much error is admissible in that case.

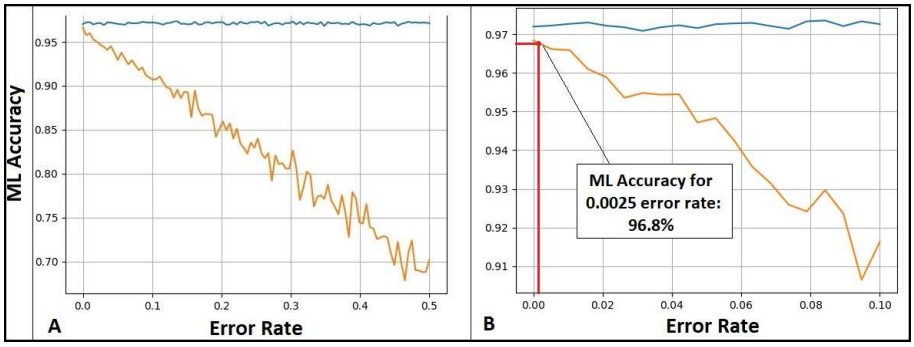

Fig. 3. Curves for Machine Learning accuracy vs amplifier error rate. In blue, accuracy without amplifier error, in orange, accuracy with error introduced. A)Machine Learning accuracy variance in relation to the error rate in the amplifier, B)Zoom in for the same curve of A.

2.4.1 Determination of G0

After deciding the admissible error rate, we are required to find a way to control this error rate. To this end, we refer to equation 1 and determine a correlation between the value of g_0 and the amplifier error (this would be the error of an individual neuron).

Since this conductance value depends directly on the range of the conductances involved in the network, the analysis was performed between various conductance ranges, and a conclusion was drawn afterwards. The results are presented in **Fig.4**.

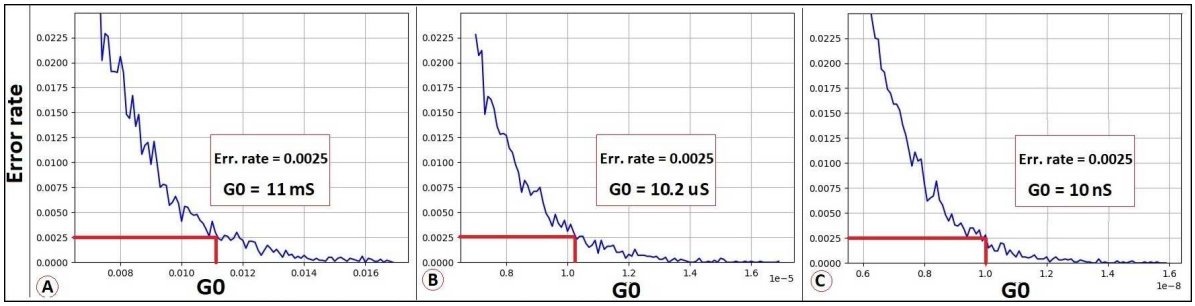

Fig. 4. Curves for g_0 determination. A) Curve for maximum conductance in the network = 100 μS, B) Curve for maximum conductance in the network = 100 nS, C) Curve for maximum conductance in the network = 100 pS

In this figure we can observe that for all conductance ranges, the optimal g_0 to achieve the desired error rate can be estimated as 100 times the maximum conductance value. If we have a system for which the maximum conductance can be expressed as G_{MAX}, the optimal g_0 to achieve a 0.25% amplifier error is $g_0 \approx 100\, G_{MAX}$

If a smaller error rate is desired, a larger conductance can be selected at the cost of increasing the power consumption.

3. Results

3.1 Autoencoder results

To test the effectiveness of the algorithm, we start by testing the reconstruction accuracy between a simple autoencoder with no RWE and an autoencoder with RWE. The autoencoders are sized *784 x 3136* and trained with the MNIST digits dataset, and the outputs of each network can be observed in **Fig.5-A**. We can observe that with half of the weights in the network, the reconstruction accuracy slightly decreased from 97.39% to 96.98%, which is an acceptable result since these autoencoders will be utilized mainly for pre-training deeper networks.

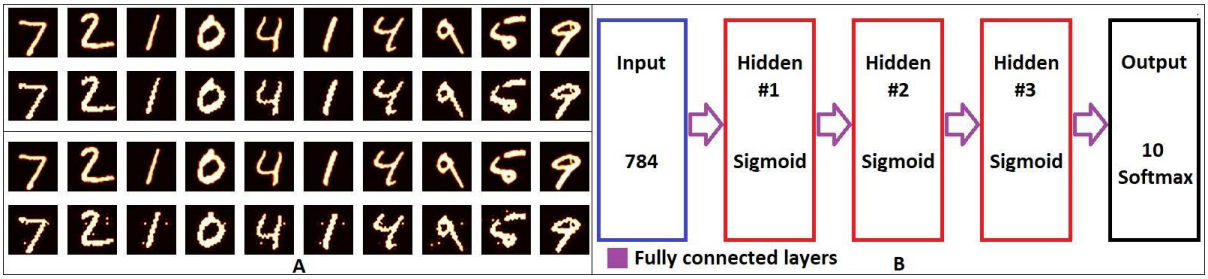

Fig. 5. A) MNIST digit reconstruction example output, normal autoencoder (up) and autoencoder with RWE (down). B) General schematic of the networks trained using supervised MLP and DBN

3.2 Deep Belief Network results

In this section, the objective is to utilize autoencoders to provide the initial weights for a Deep Belief Network (DBN) [5]. In such a network, each layer is pre-trained with RWE, and the final weight values are utilized as the initial weights of the DBN which is then subjected to supervised learning. We can observe the basic schematic of the trained DBNs in **Fig.5-B**. This study trained the networks using different parameters for optimization and layer size. The results of the different tests are summarized in **Table I**.

Table I. Learning accuracy results for DBN compared to MLP for several structures, GD represents a Gradient Descent Optimizer while A represents an Adam Optimizer.A)784x1568x784x784x10, B)784x784x784x784x10, C)784x784x600x300x10, D)784x500x300x200x10

Structure	Network type	GD Accuracy	A Accuracy	Structure	Network type	GD Accuracy	A Accuracy
A	MLP	11.78%	89.8%	B	MLP	11.75%	80.08%
A	DBN	66.46%	97.60%	B	DBN	65.94%	97.37%
Structure	**Network type**	**GD Accuracy**	**A Accuracy**	**Structure**	**Network type**	**GD Accuracy**	**A Accuracy**
C	MLP	11.35%	68.86%	D	MLP	11.23%	20.01%
C	DBN	66.05%	97.27%	D	DBN	65.62%	96.96%

4. Conclusions

In conclusion, the Weight Balancing and Random Weight Elimination algorithms were explained and tested with two different problems, an autoencoder reconstruction problem and a DBN supervised learning problem with successful results.

In the autoencoder case, a slight decrease in reconstruction accuracy was observed; however, this drop was diminished proportionally to the size of the network. This implies that when the number of connections involved in the network was higher, it was less sensitive to the RWE algorithm. On this basis, this method could be promising when considering already big networks in need of space reduction (in the case of hardware).

Regarding the DBN results, because this data-set had already been studied in detail, there are several solutions with higher accuracy than those achieved in these tests, but the important result was the **improvement** observed when applied to deeper networks. In our tests, when utilizing a relatively deep network, the accuracy for a normal fully connected MLP was observed to be approximately 11% when using a Gradient Descent Optimizer and improved to approximately 66% when using the DBN pre-trained with RWE. Regarding the Adam Optimizer, the accuracy of the MLP was highly dependent on the number of neurons. However, with the DBN, we observed accuracies closer to 97% even with a substantially reduced number of weights in the network. This result is the most important because it indicated a good result for reducing the number of connections in a deep fully-connected network.

Although it is required to test this algorithm with more datasets to check how much of an improvement could be achieved as a general result, we believe that it is at least promising for reducing the size and cost of hardware neural networks by halving (or even further reduction) the weights in deep neural networks.

References

[1] Asai T., "Fixed-Weighting-Code Learning Device" International Patent WO/2018/168293 issued September 20, 2018.

[2] Hida I., Ueyoshi K., Takamaeda-Yamazaki S., Ikebe M., Motomura M., and Asai T., "Sign-invariant unsupervised learning facilitates weighted-sum computation in analog neural-network devices," *Proc. of NOLTA 2017*, pp. 82–82, December 2017.

[3] T. Prodromakis and C. Toumazou, "A review on memristive devices and applications," *Proc. of 17th IEEE International Conference on Electronics, Circuits and Systems 2010*, pp. 934–937, December 2010. DOI: 10.1109/ICECS.2010.5724666.

[4] Akai-Kasaya M., Hagiwara N., Hikita W., Okada M., Sugito Y., Kuwahara Y., and Asai T., "Evolving conductive polymer neural networks on wetware," *Japanese Journal of Applied Physics* vol . 59, no. 6, May 2020. DOI:10.35848/1347-4065/ab8e06.

[5] Pinaya, W., Gadelha, A., Doyle, O. et al, "Using deep belief network modelling to characterize differences in brain morphometry in schizophrenia," *Scientific Reports* vol. 6, no. 1, pp. 1–9 , December 2016. DOI: 10.1038/srep38897