C-018

Fault-Tolerant MLP Learnings using Deep Learning Framework

Astriwindusari† Tadayoshita Horita† Masakazu Akiba†

Abstract

One of the author proposed a learning algorithm, called "FTL-algo", which make 3-layered perceptrons multiple-weight-and-neuron-fault tolerant, developed its CUDA C code, and gave the result using GPU. The FTL-algo is an ordinary back propagation algorithm with some modification. However, the FTL-algo using Deep Learning Frameworks have not been found.

The purpose of this study is to evaluate in time and accuracy of learning when the FTL-algo calculation is done using Deep Learning Frameworks to compare with previous result.

Index Terms – Multilayer Perceptron, Fault tolerance, GPGPU, Deep Learning Framework

1. Introduction

Artificial neural network is a computer signal processing system that based on human biological nervous system. It is composed of a large number of simple processing elements, called artificial neurons, and they are interconnected by direct links called connection weights. Artificial neurons called nodes and weights are cooperating to perform parallel distributed processing, in order to solve a desired computational task [1]. The benefits of artificial neural network is their ability to adapt connection weights in time to improve performance based on current result and continue learning in certain condition where training data is limited [2].

The development on research of artificial neural network have attracted intensive interest and elevating significant renewed growth in the field of artificial intelligence over the last two decades, e.g., deep learning models based on a feedforward deep network or multilayer perceptron [3].

Artificial neural networks are acquire other intrinsic features of biological system such as their tolerance against imprecision, uncertainty, and fault [3]. Based from neurobiological studies, the human brain is able to tolerate a small amount of synapse or neuron faults, or even use noise as a source of computation [4]. Nervous system are complex, highly massive parallel information processing architecture made of seemingly imperfect and slow, but exceptionally adaptive and power efficient components that carry out information processing functions [5][6]. Moreover, brains have the capability to relearn by growth of new neurons and/or neural connections and/or retraining of the existing neural architecture [7].

Derived from these observations, it is claimed that the majority of neural network models, abstracted from biological ones, has a potential ability of fault tolerance. In 1995, Phatak et. al. discussed the fault-tolerance through replication [8]. By the benchmark test for Sonar nets[9], they showed that more than 99% of all possible single weight faults, which are stuck at +W, 0 , or –W, are tolerated without any additional redundancy, but complete (100%) fault tolerance is not achieved even at 6 extra replications. Furthermore, Nijhuis et al. showed that fault-tolerance behavior is not self-evident, but it must be activated by an appropriate learning scheme[10]. Since then, many ideas to make multilayer neural network fault-tolerant have been studied until now. These studies evaluated the fault-tolerance from the viewpoint of stuck-at faults of weights or outputs of neurons, but not simultaneous faults. Beside above studies, Takanami et al. proposed a learning algorithm, which makes multilayer neural networks fault-tolerant to multiple weight faults, with their values being arbitrary in an interval which is specified by two extreme values [11]. Furthermore, Horita et. al proposed two extended back-propagation algorithms, hereinafter referred to as "FTL-algo" which make 3-layered perceptrons multiple-weight-and-neuron-fault tolerant [1]. FTL-algo is based on back propagation algorithm with some modification, which make the output errors in learning phase smaller than those in practical use. The abilities of fault-tolerance of the perceptrons in practical use are analyzed in the relationships between the output errors in learning phase and in practical use that shows the perceptrons have complete (100%) fault-tolerance to multiple weight-and-neuron faults in practical use. In addition, Horita et al. proposed the acceleration, by developing its CUDA C code and using GPU [12].

† Polytechnic University (PTU) Japan

In this paper, we proposed the acceleration of the previous FTL-algo using Deep Learning Frameworks. In sect 2 the multilayer neural network and the back-propagation algorithm are described. In sect 3, the fault model is mentioned. In sect 4, Deep Learning Frameworks in this research are shortly explained. In sect 5, the Method are used in research is described including FTL-algo which is the modification of back propagation. In sect 6, the simulation and result in are shown. Finally, in sect 7 the paper is concluded. The paper is the extension of [12].

2. 3-Layered Perceptron

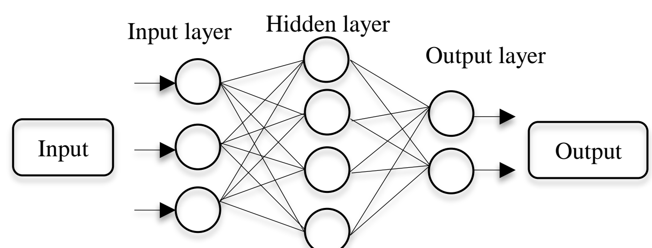

Fig. 1. 3-layered Perceptron

Copyright © 2019 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

Figure 1 shows a simple 3-layered perceptron. The first and the last layer are called input and output layers, and one layer between them is called hidden layer. Each neuron in a layer connected to all neurons in the adjacent layers through uni-directional links (synaptic weights). In this paper, we deal with only MLP which have one hidden layer. The output of each neuron (o_i) is given by

$$o_i = f\ (X_i) \tag{1}$$

$$X_i = \sum_{j=0}^{N_{pre}} w_{ij} \cdot u_j \tag{2}$$

where w_{ij} is the value of sypnatic weights from the *j*-th neuron in the preceding layer to the *i*-th neuron, $N_{pre}$is the number of the neurons in the preceding layer connected to the *i*-th neuron, u_j is the output of the *j*-th neuron in the preceding layer, w_{i0} is the synaptic weight connected to the input $u_0 = 1$ corresponding to the threshold, X_i is called "inner potential" of the neuron, and *f* is the activation function (the sigmoid function) of a neuron defined by

$$f\left(x : T_{emp}\right) = \frac{1}{1+\exp(-x/T_{emp})} \tag{3}$$

where T_{emp} is a constant called "temperature".

The learning process called "back-propagation algorithm" is based on a steepest-descendant gradient rule. Let O be a set that indices neurons in the output layers, and P be a set that indices the learning input examples. The change of each weight for the *p*-th learning input example (named$\Delta\omega_{ij}^p$) is done as follows:

$$\Delta\omega_{ij}^p = -\eta \cdot \frac{\partial E_p}{\partial w_{ij}} \tag{4}$$

$$E_p = \frac{\sum_{i\in O}\left(t_i^p - o_i^p\right)^2}{2} \tag{5}$$

where Eq.(5) $t_i^p = 0$ or 1 is the learning output example of the *i*-th neuron in the output layer for the *p*-th learning input example ($i \in O$ and $p \in P$), o_i^p is the output of the *i*-th neuron in the output layer for the *p*-th learning input example, and ηis a parameter of a positive real number.

From the above, $\Delta\omega_{ij}^p$ is derived as follows.

$$\Delta\omega_{ij}^p = \eta \cdot \delta_i^p \cdot h_j^p \tag{6}$$

$$\delta_i^p = \left(t_i^p - o_i^p\right) \cdot f'\left(s_i^p\right) \tag{7}$$

where h_j^p is the output of the *j*-th neuron in the hidden layer, and s_i^p is the inner potential of the *i*-th neuron in the output layer, for the *p*-th learning input example.

In addition, $\Delta\widehat{\omega}_{jk}^p$, which is the modification for the weight value $\widehat{\omega}_{jk}$ between the *j*-th neuron in the hidden layer, and the *k*-th neuron in the input layer for the *p*-th example, is derived as follows.

$$\Delta\widehat{\omega}_{jk}^p = \eta \cdot \sigma_j^p \cdot i_k^p \tag{8}$$

$$\sigma_j^p = \left(\sum \delta_h^p \cdot \omega_{hj}\right) \cdot f'\left(x_j^p\right) \tag{9}$$

where i_k^p is the *k*-th neuron output in the input layer, and x_j^p is the inner potential of the *j*-th neuron in the hidden layer, for the *p*-th learning input example.

In fact, concerning the modifications of ω_{ij} and $\widehat{\omega}_{jk}$, first, $\Delta\omega_{ij} = \sum_{p=1}^{N_p} \Delta\omega_{ij}^p$ and $\Delta\widehat{\omega}_{jk} = \sum_{p=1}^{N_p} \Delta\widehat{\omega}_{jk}^p$ are calculated, and then the modification is done by replacing ω_{ij} with$\omega_{ij} + \Delta\omega_{ij}$, where $N_p = |P|$ and |a set| is the number of elements in the set. Then, the weight modification is repeated until the condition defined by eq. (10) is satisfied, or the number of modification times reaches a constant number, called the number of maximum weight modification times.

$$\max_{p\in P, i\in O} \left(t_i^p - o_i^p\right)^2 < e_o^2 \tag{10}$$

where e_o is called the output error in a learning phase. If an MLP obtain by learning satisfies this condition when the number of weight modification times is less than the maximum one, the learning is said to have finished successfully and the MLP obtained is called to be "successful" in a learning phase. Otherwise, the learning is called to be "failed".

3. Fault Model

The three assumption concerning faults in the elements of MLPs, which are shown in Fig.2, are described as follows.

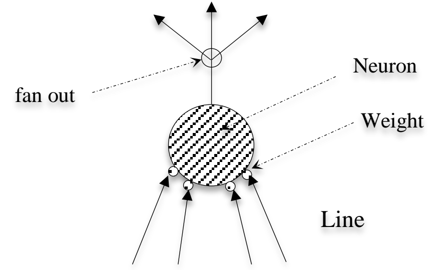

Fig. 2. Elements in MLPs

Assumption 1: (the range of faults)
Neuron also weights in hidden and output layers may be faulty except in the input layers.

Assumption 2: (the value of a weight)
The value of each weights is assumed finite for the convenience of analysis, in the range -1 to 1 even when is faulty.

Assumption 3: (the output value of a neuron)
The output value of each neuron is assumed to be in the range by its activation function from 0 to 1 even when it is faulty.

4. Deep Learning Framework

In recent years, the developments of Deep Learning are growing very fast. Nowadays, to make people easier in developing Deep Learning project there are Deep Learning Frameworks. There are so many Deep Learning Frameworks now, but in this paper we use two types of frameworks which are Keras backend Tensorflow and Pytorch based on the development, the user friendly for beginner, and speed. The purpose of this study is to evaluate in time and accuracy of learning phase when the FTL-algo calculation is done using these frameworks.

Copyright © 2019 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

The function and performance of each Deep Learning Frameworks are shown in Fig. 3.

	Languages	Tutorials and training materials	Architecture: easy-to-use and modular front end	Speed
Theano	Python, C++	++	+	++
Tensor-Flow	Python	+++	+++	++
Torch	Lua, Python (new)	+	++	+++
Caffe	C++	+	+	+
MXNet	R, Python, Julia, Scala	++	++	++

Fig. 3 The benefit of each Deep Learning Frameworks [14]

5. Method

FTL-Algo principle is the same like BP algorithm with some modification as follows.

1) The sigmoid function f_o of a neuron in the output layer is $f_o(x) = 1/\left(1 + exp(-x/T)\right)$, and $T = N_{lm}/\ln 9$, where N_{lm} is a parameters of positive integer, related to the degree of fault-tolerance.
2) The sigmoid function f_h of a neuron in the hidden layer is $f_h(x) = 1/\left(1 + exp(-x)\right)$.
3) If the value of a weight to be modified is greater(less) than 1(-1), it is set to 1(-1) to make it in the range from -1 to 1 which is called" $W_{|1|}$-process".

The process for the calculation of Eq. (6) or (8) and its update for each weight, including the $W_{|1|}$ -process, is called "backward calculation".

6. Simulations and Result

6.1. Simulations environment

The simulations in this paper are executed with deep learning environment as shown in Table 1.

In our research, we run multiple Deep Learning Frameworks using Python's virtual environment (virtualenv) on one PC under Linux environment. In Python's virtual environment, it is possible to construct an environment separated by module, package, and Python version [12]. The dependency among various libraries are different for each deep learning framework.

Table 1. Linux Environment Setting

OS	Ubuntu 16.04 LTS
CPU	AMD A10 5800K
GPU	NVIDIA GeForce GTX 1070
GPGPU Platform	CUDA 9.0 / cuDNN 7.0
Programming Language	Python 2.7.12
Deep Learning Frameworks	Tensorflow 1.8.0/ Keras 2.1.16 Pytorch 0.4.1

6.2. Result

Before making customization in Keras backend Tensorflow and Pytorch, we have to understand the structure for each Deep Learning Frameworks and its functions. The difference between each Deep Learning Frameworks is shown in Fig.4 and Fig. 5 below.

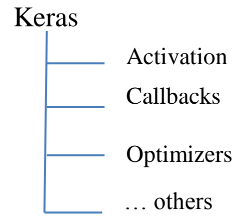

Fig. 4. Keras backend Tensorflow framework structure

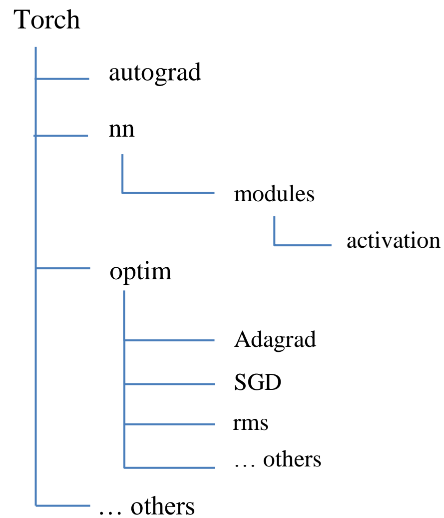

Fig. 5 Pytorch deep framework structure

Until this time the process have not been finished yet, so the research progress described as follows

1) The environment for running program with each deep learning framework (Keras backend Tensorflow and Pytorch) has been created with Virtualenv [15].
2) The custom in sigmoid activation function as shown in Eq. 3 calculation has been created
3) MNIST dataset are still using The MNIST database [16].
4) Weight outputs in csv file format have been done however the weight output in each layer has not been succeeded.
5) Early stopping in program with Eq. (10) condition has been implemented.
6) "$W_{|1|}$-process" has not been included in the program.
7) Since this paper is the extension from [12], so adjusting dataset with previous research (MNIST dataset set 1, set 2, and set 3) has not been succeeded.
8) After this code is succeeded in Tensorflow, the same code will be running in Pytorch

Copyright © 2019 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

7. Conclusion

After some simulations, we discover that the first thing to do deep learning is hardware preparations and knowing your GPU compute capability is important.

Also, Deep Learning Frameworks are full package that had different structure in each type. Making modification on the package Deep Learning Frameworks library are difficult and took a lot of time for beginners. In deep learning framework Keras backend Tensorflow, there is special ability that called auto differentiation which is the differentiation in activation function that are automatically be done. This special ability give us benefit when we want to make custom in activation function that are based from activation function library that comes with Tensorflow package because we don't have to define the new custom activation function derivative.

References

[1] T. Horita, I. Takanami, and M. Mori. "Learning Algorithm Which Make Multilayer Neural Networks Multiple-Weight-and-Neuron-Fault Tolerant", IEICE Transactions on Information and Systems Vol.E91-D, No.4, pp.1168-1175 (2008).
[2] Richard P.Lippmann, "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine Vol. 4 No. 2, pp.4-22 (1987)
[3] Cesar Torres-Huitzil and Bernard Girau, "Fault and Error Tolerance in Neural Networks : A Review", IEEE Access, Vol. 5, pp.17322-17341 (2017).
[4] S. Srinivasan and C. F. Stevens, ``Robustness and Fault Tolerance Make Brains Harder to Study," BMC Biol., vol. 9, pp. 46-48, (2011)
[5] T. Sejnowksi and T. Delbruck, ``The Language of The Brain," Sci. Amer.,vol. 307, pp. 54-59, (2012).
[6] W. Maass, ``To Spike or not to Spike: That is The Question," Proc. IEEE,vol. 103, no. 12, pp. 2219-2224, (2015).
[7] P. Chandra and Y. Singh, ``Fault Tolerance Of Feedforward Articial Neural Networks a Framework Of Study," in Proc. Int. Joint Conf. Neural Netw.,vol. 1., pp. 489-494, (2003).
[8] D.S. Phatak and I.Koren, "Compete And Partial Fault Tolerance Of Feedforward Neural Nets," IEEE Trans. Neural Netw., vol.6, No. 2, pp.446-456, (1995).
[9] S.E. Fahlman, et al., "Neural Nets Learning Algorithms and Benchmarks Database," maintained by S.E. Fahlman et al. at the Computer Science Dept., Carnegie Mellon University.
[10] J. Nijhuis, B. Hoefflinger, A. van Schaik, and L. Spaanenburg, "Limits to The Fault-Tolerance of A Feedforward Neural Network with Learning," FTCS, pp.228-235, (1990).
[11] I.Takanami and Y.Oyama, "A Novel Learning Algorithm which Makes Multilayer Neural Networks Multiple-Weight-Fault-Tolerant", IEICE Trans. Inf. & Syst., vol.E86-D, no.12, pp.2536-2543, (2003).
[12] T.Horita, I.Takanami, M.Akiba, N. Terauchi, T. Kanno, "A GPGPU-based Acceleration of Fault-Tolerant MLP Learnings, Proc. IEEE 8th International Symposium on MCSoC, (2014)".
[13] "仮想環境" homepage, https://www.python.jp/ install/ windows /virtualenv.html
[14] " Getting Started with Deep Learning" homepage, https://www.svds.com /getting-started-deep-learning (2017).
[15] "Virtualenv" homepage, https://virtualenv.pypa.io/en/latest/
[16] "The MNIST Database of handwritten digits" homepage http://yann.lecun.com/exdb/mnist/./

Copyright © 2019 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.