Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Volume 13, Issue 03 March 2024

A Shadow Cost of Jacobian Accumulation in Computation Graph of Backpropagation

Ruo Ando
National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan

Yoshihisa Fukuhara, Yoshiyasu Takefuji
Musashino University
3-3-3 Ariake, Koto-Ku, Tokyo, Japan

***Abstract*—Backpropagation applies backward automatic differentiation on the computational graph. Theoretically, it is assumed that the computation time per processing unit remains unchanged as the number of epochs increases. This paper introduces a novel phenomenon: as the number of batch processes increases, the time to complete the backward automatic differentiation increases. The proposed method implements error backpropagation by implementing recursive calls to the backpropagation function through class derivation. It was shown that the execution time of the processing unit of this recursive execution gradually increases over the course of the learning process. This paper also discusses the possible causes of this hidden computational cost.**

***Keywords*—Back propagation, reverse mode diffrentiation, gradual increase of elapsed time.component**

I. INTRODUCTION

Reverse mode differentiation is currently employed in deep learning frameworks. The implementation of backward automatic differentiation used in error back propagation makes extensive use of the chain rule. The chain rule was invented by Leibniz in the 17th century [1]. Engineering applications of the chain rule began in the 1960s, mainly in the field of control [2]. Based on the results of these studies, BP was formulated in 1986 [3]. One of the main themes of error backpropagation in the 21st century is the gradient vanishing problem. In 2006, methods to solve the gradient vanishing problem, such as prior learning, were proposed [4]. This paper tests the following hypotheses:

Hypothesis: The computation graph does not change regardless of the number of batch processes or epochs, so the execution time per processing unit does not change.
This paper implements backpropagation by recursive function calls on a computational graph to test the above hypothesis.

II. RELATED WORK

Backpropagation is a special case of a broad class of techniques called reverse mode automatic differentiation. The task of reducing computational complexity by simplifying the computational graphs generated during reverse mode automatic differentiation belongs to the NP-complete problem [5]. Theano [6] and Tensorflow[7] use heuristics to simplify the iterative computational graph while matching known patterns Some studies use quadratic partial derivatives to calculate the gradient.

III. METHODOLOGY

The proposed method measures the execution time of back propagation on an epoch or batch basis.

A. Fromulation

First of all, the loss fuction is defined as follows:

$$E_p = \frac{1}{2}\sum_{j=1} N(Z_j^{(L)} - t_j)^2 \tag{1}$$

The derivative of the output layer is as follows:

$$\frac{\partial E_p}{\partial U^{(L)}} = \sum_c \frac{\partial Z_c^{(L)}}{\partial U^{(L)}} \frac{\partial E_p}{\partial Z_c^{(L)}} = \sum_c t_c (Z_j^{(L)} - t_j) = Z_j^{(L)} - t_j \tag{2}$$

where:

$$\frac{\partial Z_c^{(L)}}{\partial E} = -\sum_c t_c log' Z_c^{(L)} \; - \sum_c t_c \frac{1}{Z_c^{(L)}} \tag{3}$$

Simply put,

$$\Delta^{(L)} = \frac{\partial E_p}{\partial U^{(L)}} = Z^{(L)} - T \tag{4}$$

The differential of the intermediate layer is as follows:

$$\frac{\partial E_p}{\partial W_{ji}^{(L)}} = \frac{\partial E_p}{\partial U_j^{(L)}} \frac{\partial U_j^{(L)}}{\partial W_{ji}(L)} = \delta_j^{(l)} Z_i^{(l-1)} \tag{5}$$

Where:

$$\delta_j^{(l)} = \frac{\partial U_j^{(l)}}{\partial E_p} \tag{6}$$

Simply put,

$$\Delta^{(l)} = (W^{(l+1)T} \Delta\ (l+1)) \odot f^{(l)}(U^{(l)}) \tag{7}$$

$$\frac{\partial E_p}{\partial W^{(l)}} = \Delta^{(l)} Z^{(l-1)T} \tag{8}$$

Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Volume 13, Issue 03 March 2024

```
1  void Variable::backward(Variable *v) {
2      if (v == NULL) {
3          return;
4              }
5      if (v->creator != NULL) {
6          if (v->last_opt != NULL && v->opt == *v->last_opt){
7              *v->is_last_backward = true;
8          }
9          if (v->forward_count >0)
10           v->forward_count--;
11         if (v->is_last_backward != NULL && *v->is_last_backward == false)
12             return;
13         if (v->forward_count != 0) return;
14         v->creator->backward(v->grad);
15
16         for (int i = 0; i< v->creator->inputs.size(); i++) {
17             PVariable nv = v->creator->inputs[i];
18             if (nv->isGetGrad) {
19                 this->backward(nv.get());
20             }
21         }
22     }
23     else{
24     }
25 }
```

Listing 1. Core routine of backpropagation.

B. Implementation

We define four classes: model. graph. function and variable. Figure 1 depicts the program control of forward and backward propagation by the member function (*creator) of the variable class. In forward propagation, the program generates a new function class. In backward propagation, the program traces the function pointer (*creator) through the argument of the function. Note that *creator points to function 1, of which input is variable 1.

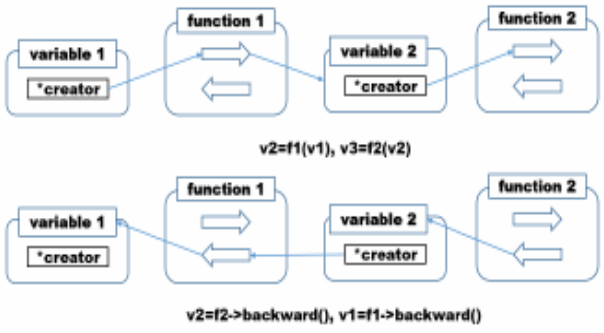

Figure 1.

Figure 2 depicts tree view of proposal method which is same as Figure 1. In Figure 2, program run down as a new variable is generated in each function code. Also, as shown in this figure, some functions have two arguments, while others have one argument. Each function has a return value of variable class. When a function is called, the variable class of the return value is generated, and the function from which it was generated is specified by "this" and registered. This makes it possible to refer to which function generated each variable class at the time of backpropagation. In backpropagation, the backpropagation function is first called from the variable class.

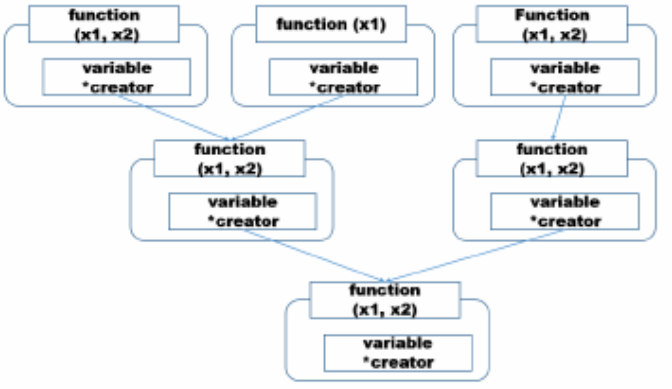

Figure 2.

There are two back propagation invocation points: line 14 and line 19. At line 16 of Listing 1, the backpropagation function of the function class that generated this variable class is called. After this call, the backpropagation function is executed for each variable, which is an argument of the backpropagate action function of the completed function class. This process is performed in a loop starting at line 16.

IV. EXPERMENTAL RESULTS

In the experiment, we use a workstation with Intel(R) Xeon(R) CPU E5-2620 v4 (2.10GHz) and 251G RAM. The evaluation experiment uses a basic multilayer perceptron for recognizing MNIST dataset. The snippet of our program is shown in Listing 2.

Figure 3 plots the execution time measured for each batch process. CPU idle time is measured by batch unit. The experiment was conducted with a sample size of 18,000 divided into 50 batches. The units on the y-axis are milliseconds (execution time), and x depicts the number of batches processed. The overall trend shows a slight increase, oscillating between a minimum of 7 milliseconds and a maximum of 14 milliseconds.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Volume 13, Issue 03 March 2024

```
model("g1", new Linear(n_size, i_size));
model("g_relu1", new ReLU());
model("g_drop1", new Dropout(dropout_p));
model("g2", new Linear(n_size, n_size));
model("g_relu2", new ReLU());
model("g_drop2", new Dropout(dropout_p));
model("g3", new Linear(o_size, n_size));
model("g_softmax_cross_entoropy",
    new SoftmaxCrossEntropy());
model("g_softmax", new Softmax());
```

Listing 2.

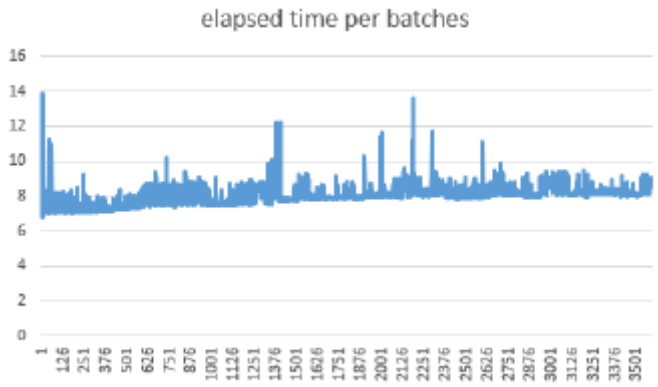

Figure 3. Elapsed time (sec) per batches.

Figure 4 plots the execution time per epoch. Here, too, the overall trend is upward, oscillating between 16000 and 25000 ms. Finally, we find that the processing time increased by a factor of 1.6 during the 30 epochs.

Figure 4. Elapsed time (sec) per epoches.

Although the overall number of repetitions is the same, the rate of increase is higher for the epoch measurement time compared to the batch measurement. This is because the number of iterations for the batch is 3600, while the number of iterations for the epoch is 30. In other words, the total processing time for 120 batches per epoch is totaled for each epoch. In addition, the variation in processing time per batch is very large. In the comparison between Figures 3 and 4, it can be seen that Figure 3 has more spikes in execution time due to external factors, but the trend as a whole is increasing. From this, it can be seen that the increase in the trend of execution time due to the increase in the number of epochs in Figure 4 is not caused by the sum of the external disturbances in Figure 3. The conclusion is that as the number of epochs increases, there is a shadow cost, where the processing content is the same, but the processing time increases.

V. CONCLUSION

In this paper, we have done the shadow cost of processing the computational graph of error back propagation. The hypothesis stated at the beginning of this paper was as follows.

Hypothesis: The computation graph does not change regardless of the number of batch processes or epochs, so the execution time per processing unit does not change.
In fact, it was shown that even if the computational graph does not change during the learning process, the computation time per processing unit time gradually increases. This is quite difficult to formulate theoretically into a mathematical equation. The proposed method implements a forward propagation network using C++ object orientation and function pointers. In the evaluation experiments, the execution time for each epoch and number of batch processes was measured in milliseconds. The measurement results showed that the average processing time per epoch increased by 7 to 8 milliseconds through 30 trials. The processing time per batch increased, on average, by 16,000 to 24,000 milliseconds through 3,600 trials. Experimental results reveal that there is a shadow cast to the processing time of backpropagation, which increases with each successive epoch and batch process. One possible factor for shadow cost is the delay in memory release timing during class derivation and pointer processing. For further work, we will improve the debugging method to validate freeing memory through backward propagation.

REFERENCES

[1] Rodriguez, Omar Hernandez and Lopez Fernandez, Jorge M. (2010) “A semiotic reflection on the didactics of the Chain rule,” The Mathematics Enthusiast: Vol. 7 : No. 2, Art. 10.
[2] Stuart E Dreyfus, "Dynamic programming and the calculus of variations", J. Math. Anal. and Appl., 1 (No. 2) (1960), pp. 228-239
[3] Uwe Naumann:Optimal Jacobian accumulation is NP-complete. Math. Program. 112(2): 427-441 (2008)
[4] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. (2010).Theano: a CPU and GPU math expression compiler.In Proceedings of the Python for Scientific Computing Conference (SciPy).
[5] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zheng: TensorFlow: A System for Large-Scale Machine Learning. OSDI 2016: 265-283
[6] Ronald J. Williams, David Zipser: A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. Neural Comput. 1(2): 270-280 (1989)
[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. I, D. E. Rumelhart and J. L. McClelland (Eds.) Cambridge, MA: MIT Press, pp. 318-362, 1986
[8] Yurii E. Nesterov: Primal-dual subgradient methods for convex problems. Math. Program. 120(1): 221-259 (2009)