# Leapfrogging for parallelism in deep neural networks

## Yatin Saraiya

847 Moana Court, Palo Alto 94306, CA, USA

#### Abstract

We present a technique, which we term leapfrogging, to parallelize back-propagation in deep neural networks. We show that this technique yields a savings of 1 - 1/k of a dominant term in backpropagation, where k is the number of threads (or gpus).

Keywords:

Neural net, parallel, optimization, backpropagation

#### 1. Introduction

One pass over a neural network consists of 2 phases, forward and backward propagation. Each phase consists of computations applied at each layer of the neural net, in sequence. There are three dominant subcomputations at each level, all matrix computations: of z,  $\delta$  and  $\nabla w^1$ . We present an algorithm, which we call leapfrogging, to parallelize the computation of  $\nabla w$ . The relative speedup in this computation is 1-1/k, where k is the number of threads used. Our approach seems to be different from existing approaches, such as pipelining ([1]) and striping ([2]).

#### 2. Computations in one pass

We will use the treatment and notation of [3] in this paper. Consider a neural network with L layers numbered  $1, \ldots, L$ , in which each of the hidden layers has N neurons. The metrics below apply to the hidden layers, although all equations are generally valid.

Email address: yatinsaraiya12@gmail.com (Yatin Saraiya)

<sup>&</sup>lt;sup>1</sup>We will use the notation of [3] in this paper.

We will use the following notation. Let  $w^l$  be the matrix of weights at the lth layer. It has dimension  $N \times N$ . Let  $z^l$  be the vector of weighted inputs to the lth layer. It has dimension  $N \times 1$ . Let  $d^l$  be the vector of activations. It has size  $N \times 1$ . Let  $d^l$  be the vector of biases at the dth layer. It has dimension d is at the dth layer. It is of dimension d is the vector of errors at the dth layer. It is of dimension d is at the cost with respect to the activations at the dth layer. Its dimension is d is the vector of one pass proceeds as follows, where d is the vector of inputs.

$$a^1 = x^1 \tag{1}$$

$$z^l = w^l \dot{a}^{l-1} \tag{2}$$

$$a^l = \sigma(z^l) \tag{3}$$

The above equations define the forward pass. The following equations apply to backpropagation.

$$\delta^L = \nabla^L_a C \odot \sigma'(z^L) \tag{4}$$

$$\delta^{l} = ((w^{l+1})^{T} \delta^{l+1}) \odot \sigma'(z^{l}) \tag{5}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{6}$$

We will use  $\nabla_b^l$  to denote the vector of  $\frac{\partial C}{\partial b_j^l}$ .

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{7}$$

We will use  $\nabla_w^l$  to denote the matrix of  $\frac{\partial C}{\partial w_{jk}^l}$ .

The dominant computations are Equation 2, Equation 5 and Equation 7.

#### 3. Leapfrogging

The essence of leapfrogging is to create a number of threads, say k, so that each thread computes equations 6 and 7 at intervals of size k such that the threads are interleaved. Let the threads be numbered  $0, 1, \ldots k-1$ , and assume that all quantities have been computed for the last k layers. Then for any j, the computation by thread numbered j will compute all quantities

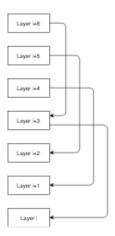


Figure 1: Leapfrogging

except for equations 6 and 7, and compute these at levels denoted by mk+j for any m such that mk+j < L-k-1. That is, each thread will compute these equations at only 1/kth of the layers. Figure 1 shows the picture, where the number of threads k is set to 3.

Algorithm 1 describes the process.

#### Theorem 1 (Correctness of Algorithm 1). Algorithm 1 is correct.

*Proof.* The parent thread computes Equation 4 at layer L, and equations 5, 6 and 7 for each level j such that j > L - k. Each child thread computes Equation 5 at every level j such that j <= L - k. Furthermore, each child thread numbered j computes equations 6 and 7 at levels L - 1 - j - km, where m >= 0, and puts the results in shared memory. Thus these equations are computed at every layer by some child thread.

#### 4. Analysis

Our analysis addresses the relative speedup of the entire forward and backward pass. More precisely, let f be the total computational cost at any one layer, and let  $f_1$ ,  $f_2$  and  $f_3$  be the cost of evaluating equations 2, 5 and 7 respectively, sequentially. Even with the synchronization cost, it is clear that these three values are dominant, so we write

$$f = f_1 + f_2 + f_3 \tag{8}$$

## **Algorithm 1** Backpropagation with threads

```
1: procedure Backpropagation(k)
                                                    \triangleright One backward pass. k is the
    number of threads to use.
        Apply Equation 4 to obtain \delta^L
 2:
        for i = L - 1, L - 2, \dots, L - k + 1 do
 3:
            Apply equations 5, 6 and 7 at layer i
 4:
            Save \nabla_b^i and \nabla_w^i to shared memory
 5:
        end for
 6:
        Construct k threads t_0, t_1, \ldots, t_{k-1}
 7:
        for j=0,1,\ldots,k-1 do Thread(j,\,k,\,\delta^{L-j})
 8:
        end for
 9:
        for j = 0, 1, ..., k-1 do join t_i \triangleright Wait for all threads to complete
10:
        end for
11:
12: end procedure
13: procedure Thread (j, k, \delta) \triangleright Backward pass with thread t and offset j
14:
        while j < L - k - 1 do
            i \leftarrow 0
15:
            while i < k \text{ do}
16:
                l \leftarrow L - 1 - j - i
17:
                Apply Equation 5 at layer l
18:
                if i == k-1 then
19:
                    Apply Equations 6 and 7 at layer l
20:
                    Save \nabla_b^l and \nabla_w^l to shared memory
21:
                end if
22:
                i \leftarrow i + 1
23:
            end while
24:
25:
            j \leftarrow j + k
        end while
26:
27: end procedure
```

Let f' be the cost of Algorithm 3. It is given by

$$f' = f_1 + f_2 + f_3/k \tag{9}$$

where k is the number of threads. The relative speedup is then given by

$$(f - f')/f = (1 - 1/k)f_3/f \tag{10}$$

Hence the relative speedup for a complete forward pass and backward pass is 1-1/k times the ratio of  $f_3$  to f, which we will assume is approximately constant for each hidden layer. The quantity 1-1/k rapidly approaches 1 as k increases, Formally, let  $\epsilon$  be the desired speedup 1-1/k. Assume we wish to find the smallest k such that  $1-1/k > 1-\epsilon$ . Then simple algebraic manipulations show that we need to set the number of threads  $k = \lceil 1/\epsilon \rceil$ . However, the absolute speedup depends on the magnitude of  $f_3$ .

#### 5. Compliance

Conflict of interest. There are no conflicts of interest on the part of the author.

- [1] A. Petrowski, G. Dreyfus and C. Girault, Performance analysis of a pipelined backpropagation parallel algorithm, IEEE Transactions on Neural Networks, Volume 4, Issue 6, (1993)
- [2] Karel Vesely, Lukas Burget and Frantisek Grezl, Parallel Training of Neural Networks for Speech Recognition, Proc. Interspeech, Makuhari (2010).
- [3] Michael A. Nielsen, Neural Networks and Deep Learning. Determination Press (2015)