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 backpropagation 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 $\nabla w$. 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, ..., $L$, 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)

\footnote{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 $l$th layer. It has dimension $N \times N$. Let $z^l$ be the vector of weighted inputs to the $l$th layer. It has dimension $N \times 1$. Let $a^l$ be the vector of activations. It has size $N \times 1$. Let $b^l$ be the vector of biases at the $l$th layer. It has dimension $N \times 1$. Let $C$ be the cost function for the network. Let $\delta^l$ be the vector of errors at the $l$th layer. It is of dimension $N \times 1$ for each hidden layer. Let $\nabla^l_a$ denote the vector of the partial derivatives of the cost with respect to the activations at the $l$th layer. Its dimension is $N \times 1$. Let $\sigma$ be the sigmoid function, and $\sigma'$ be the derivative. Then the computation of one pass proceeds as follows, where $x^1$ is the vector of inputs.

\begin{align*}
a^1 &= x^1 \quad \text{(1)} \\
z^l &= w^l a^{l-1} \quad \text{(2)} \\
a^l &= \sigma(z^l) \quad \text{(3)}
\end{align*}

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

\begin{align*}
\delta^L &= \nabla^L_a C \odot \sigma'(z^L) \quad \text{(4)} \\
\delta^l &= (((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad \text{(5)} \\
\frac{\partial C}{\partial b^l_j} &= \delta^l_j \quad \text{(6)}
\end{align*}

We will use $\nabla^l_b$ to denote the vector of $\frac{\partial C}{\partial b^l_j}$.

\begin{equation}
\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \quad \text{(7)}
\end{equation}

We will use $\nabla^l_w$ to denote the matrix of $\frac{\partial C}{\partial w^l_{jk}}$.

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
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 \( 1/k \)th 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 \( j \leq L - k \). Furthermore, each child thread numbered \( j \) computes equations 6 and 7 at levels \( L - 1 - j - km \), where \( m \geq 0 \), and puts the results in shared memory. Thus these equations are computed at every layer by some child thread. \( \square \)

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

\[
f = f_1 + f_2 + f_3
\]  
(8)
Algorithm 1 Backpropagation with threads
1: \textbf{procedure} \textsc{Backpropagation}(k) \Comment{One backward pass. $k$ is the number of threads to use.}
2: \hspace{1em} Apply Equation 4 to obtain $\delta^L$
3: \hspace{1em} \textbf{for} $i = L - 1, L - 2, \ldots, L - k + 1$ \textbf{do}
4: \hspace{2em} Apply equations 5, 6 and 7 at layer $i$
5: \hspace{2em} Save $\nabla^i_b$ and $\nabla^i_w$ to shared memory
6: \hspace{1em} \textbf{end for}
7: \hspace{1em} Construct $k$ threads $t_0, t_1, \ldots, t_{k-1}$
8: \hspace{1em} \textbf{for} $j = 0, 1, \ldots, k - 1$ \textbf{do} \textsc{Thread}($j, k, \delta^{L-j}$)
9: \hspace{2em} \textbf{end for}
10: \hspace{1em} \textbf{for} $j = 0, 1, \ldots, k - 1$ \textbf{do} join $t_j$ \Comment{Wait for all threads to complete}
11: \hspace{1em} \textbf{end for}
12: \textbf{end procedure}
13: \textbf{procedure} \textsc{Thread}($j, k, \delta$) \Comment{Backward pass with thread $t$ and offset $j$}
14: \hspace{1em} \textbf{while} $j < L - k - 1$ \textbf{do}
15: \hspace{2em} $i \leftarrow 0$
16: \hspace{2em} \hspace{1em} \textbf{while} $i < k$ \textbf{do}
17: \hspace{3em} $l \leftarrow L - 1 - j - i$
18: \hspace{3em} \hspace{1em} Apply Equation 5 at layer $l$
19: \hspace{3em} \hspace{1em} \textbf{if} $i == k - 1$ \textbf{then}
20: \hspace{4em} \hspace{1em} Apply Equations 6 and 7 at layer $l$
21: \hspace{4em} \hspace{1em} Save $\nabla^l_b$ and $\nabla^l_w$ to shared memory
22: \hspace{3em} \hspace{1em} \textbf{end if}
23: \hspace{3em} $i \leftarrow i + 1$
24: \hspace{2em} \hspace{1em} \textbf{end while}
25: \hspace{2em} $j \leftarrow j + k$
26: \hspace{1em} \textbf{end while}
27: \textbf{end procedure}
Let $f'$ be the cost of Algorithm 3. It is given by

$$f' = f_1 + f_2 + f_3/k$$  \hspace{1cm} (9)$$

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

$$\frac{f - f'}{f} = (1 - 1/k)\frac{f_3}{f}$$  \hspace{1cm} (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)