支持 OpenCL 的 GPU 加速人工神经网络训练①

祝伟华，付先珺

(重庆大学 软件工程学院，重庆 400044)

摘 要：人工神经网络训练所包含的运算量随着网络中神经元的数量增多而加大，对于神经元较多的网络训练很耗时。提高人工神经网络训练速度的一个方法是对训练算法优化以减少计算量。由于人工神经网络训练算法包含大量的矩阵和向量运算，如果把优化的算法用运行在 GPU 上的 OpenCL C 语言实现，则训练速度相比传统基于 CPU 计算的实现会提高很多。从硬件的并行计算能力着手，以 RPROP 算法为例，对其运行在 GPU 上的 OpenCL C 语言实现作一些研究。

关键词：加速；人工神经网络；RPROP；OpenCL；CPU

Accelerating of Artificial Neural Network Training by GPU with OpenCL Support

ZHU Wei-Hua, FU Xian-Jun

(School of Software Engineering, Chongqing University, Chongqing, 400044, China)

Abstract: The computation quantity in artificial neural network training will get more and more with the increase of neurons quantity, it is time-consuming for training a neural network with too many neurons. A method that accelerates artificial neural network training is to optimize the training algorithm, so as to reduce the computation quantity. Since there is too much matrix and vector computation in artificial neural network training algorithm, the optimized training algorithm implemented by OpenCL C language on GPU, compared to the conventional CPU-based implementation, the training speed will be increased a lot. Based on parallel computing ability of hardware, accelerating of artificial neural network training by GPU with OpenCL Support is researched in this paper.

Key words: accelerating; artificial neural network; RPROP; OpenCL; GPU

1 概述

人工神经网络的运算量不但与网络的规模有关，还与网络各层权值和偏置值的初始值的选取、学习模型的选取和算法等有关。为了减少运算量，很多改进算法被提出来。例如 BP 算法，由于可能陷入局部极值、最佳学习率的选取不当等会对训练的速度和结果造成影响，一种改进方法（例如 RPROP 算法）是通过不断调整学习率来改变算法的收敛速度，避免发散，让神经网络的训练更快更佳[1]。如果把改进的算法用基于 GPU 运算的 OpenCL C 语言实现将比传统的基于 CPU 运算的实现节约更多时间。本文在 64 位 Windows7 操作系统的环境下，用开源程序 Encog 2.3 for C#中以 XOR 运算为训练模型的 RPROP 算法为例，对其运行在 ATI Radeon HD 5700 Series GPU 上的 OpenCL C 语言实现方法做一些研究，并与运在 Intel(R)Core(TM) i5 CPU 上的源程序训练时间做比较。

2 RPROP算法原理

以下是 Encog Project 中关于 RPROP 算法的原理。

以 x-y-z 二层网络为例（输入、中间层和输出层神经元数量分别为 x，y 和 z）。设输入层向量、理想输出向量和实际输出向量分别用 I、O_{ideal} 和 O_{actual} 表示；各层权值矩阵、偏置值纵向量、输出纵向量和激活函数分别用 W_i、B_i、A_i 和 $f_i()$表示；各层权值偏差矩阵、

① 收稿时间:2010-10-20;收到修改稿时间:2010-12-04

偏置值偏差纵向量、输出偏差纵向量和激活函数的微分函数分别用 Wd_i、Bd_i、Ad_i 和 $f_i'()$表示；其中 i 表示层顺序，如果 $f_i()$不可微，则 $f_i'()$表示常数为 1 的函数； [j]表示矩阵的第 j 个横向量或向量的第 j 个标量；N<i>表示第 i 层神经元数量。用 * 表示两相同长度的向量相乘，其结果是一向量，结果中的标量元素值是两相乘向量对应位置的标量元素相乘。

按经验，对 Wi 和 Bi 的每一个标量元素，应选择较小的随机值作为的初始值并设置最初学习率（设为 aj），各标量元素的变化加速因子（一个大于 1 的浮点数，设为 increase）、减速因子（一个在 0 到 1 之间的浮点数,设为 decrease）和变化的上下限(设为 u 和 l)都设置为统一的常量。下面是一次训练算法的基本原理。

首先，对每一个学习模型都作以下操作：

① 前馈计算：I=模型的输入向量；Oideal=模型的输出向量；所以有 $A_2=f_2[W_2f_1(W_1I+B_1)+B_2]$。

② 反向传播计算各层权值和偏置值偏差，并与其它所有模型在该步骤计算出的偏差求代数和：

a. $O_{actual}= A_2$；

b. $Ad_2= f_2'(O_{actual}) * (O_{ideal} - O_{actual})$；

c. $Bd_i =Ad_i$；

d. $Wd_i[j]=(A_{i-1}\times Ad_i[j])^T$，其中 1<=j<=N<i>；

e. $Ad_i^T=[\sum_{j=1}^{N<i+1>}(W_{i+1}[j]\times Ad_{i+1}[j])] * f_i'(A_i)$；

然后，把每个训练模型所计算出相同层的 Wd_i 和 Bd_i 累加得到本次训练的 Wd_{inew} 和 Bd_{inew}，与上次训练计算出的 Wd_{ilast} 和 Bd_{ilast} 相比较来调整相应层的权值矩阵和偏置值向量（对第一次循环，Wd_{ilast} 和 Bd_{ilast} 各元素均为 0）。其原理是分别比较 Wd_{inew} 和 Bd_{inew} 与 Wd_{ilast} 和 Bd_{ilast} 对应层的对应标量元素，设为 d_{new} 和 d_{last}，作为对应层的权值矩阵和偏置值中对应标量元素的改变量，设为 change：

① 如果 $d_{new}\times dl_{ast}=0$，则 change=0；$d_{last}=d_{new}$；

② 如果 $d_{new}\times d_{last}>0$，则 $a_j= min(a_j*increase,u)$；change=Sign（d_{new}）*a_j； $d_{last}= d_{new}$；其中 Sign（x）根据 x 是零、正或是负值分别返回 0，1 或-1；

③ 如果 $d_{new}\times d_{last}<0$，则 $a_j= max(a_j*decrease,l)$；change=0；$d_{last}=0$；

从弹性算法的原理看来，相对于传统的反向传播算法，它不但通过适当地改变学习率来提高网络的学习速度和防止训练形成局部最小，并且在运算量也减少了。

3 运行在GPU上的OpenCL C语言实现RPROP算法的研究

本文的程序和测试数据是用支持 1.0 版《OpenCL 规范》的 ATI Stream SDK v2.1 完成的。目前支持 1.1 版《OpenCL 规范》的 ATI Stream SDK v2.2 已经发布。

OpenCL 的执行模型可以分为两部分，一部分是在宿主机上执行的主程序，另一部分是在支持 OpenCL 的备上执行的内核程序。在本文的程序里，主程序在 CPU 上执行，内核在 GPU 上执行[2]。因为我的计算机只有一个支持 OpenCL 的 GPU，所以在初始化 OpenCL 时只建立一个命令队列为佳。由于算法的实现是在内核中，所以本文只讨论单 GPU 上内核对于算法的实现。关于 OpenCL 介绍和主程序的实现可参考 Khronos 发布的《The OpenCL Specification》。

3.1 前馈算法的实现举例

由第 2 部分看出，RPROP 算法中仅在前馈计算中涉及矩阵运算，且是矩阵与向量相乘，其他步骤都是向量运算，所以算法的每一步都应由一维索引空间中的工作项执行。

下面以 x-y-z 二层网络前馈计算的第一层计算 $A_1=f_1(W_1I+B_1)$为例，来说明内核实现的方法。因为矩阵 W_1 的大小是 y 行 x 列，向量 I 长为 x，B_1 和 A_1 长为 y，所以执行内核所需的工作项数量为 y,索引空间为一维。工作项执行内核时，第 i 个工作项所计算的是 $A_1[i]$，其值是 $f_1(W_1[i]I+B_1[i])$，内核实现代码为图 1 所示[3,4]。

```
// 神经网络一层前馈计算
// W,I,B,A,x分别表示该层的
// 权重矩阵，输入向量，偏置值向量，输出向量，输入向量长度
__kernel void FeedForward(__global float* W,
                          __global float* I,
                          __global float* B,
                          __global float* A,
                          int x)
{
    // i表示工作项的索引，其值从0到y，
    // 一个工作项执行一个内核，y个工作项同时执行，完成一层前馈计算
    int i=get_global_id(0);
    float sum=0;
    // 矩阵-向量乘法
    for(int k=0; k<x;k++)
    {
        sum+=W[I*x+k]*I[k];
    }
    // 加偏置值
    sum+=B[i];
    // 设f1为sigmoid函数
    A[i]=1.0/(1.0+native_exp(sum*(-1)));
}
```

图 1　第一层前馈计算内核举例

3.2 多个内核实现 RPROP 算法

图 1 是对网络第一层前馈计算的 OpenCL C 语言实现；反向传播计算各层权值和偏置值的偏差中每一

步计算都是基于向量计算，可按 3.1 的方法对其每一步用一个内核实现。一次训练中迭代完所有训练模型后调整各层权值矩阵和偏置值向量时，由于只是改变矩阵和向量中的元素，可以把 m×n 的矩阵看作长为 m×n 的向量来处理，可按 3.1 的方法对每层的权值和偏置值调整都分别用一个内核实现。由于很多步都依赖其前面步骤的结果，所以在主程序中应把每个内核应依次进入顺序执行的命令队列并执行，即可完成一次训练。若完成训练需要 t 次，由主程序循环地把一次训练的所有内核依次入队命令队列，这样若一次训练要执行 n 个内核，则完成训练需要执行 n×t 个内核。由于各层之间的权值矩阵和偏置值向量的规模大小不同，所以每次训练中不同的内核就可能需要不同数量的工作项来执行。

3.3 单个内核实现 RPROP 算法[5]

表 1 是对 2-3-1 网络用 OpenCL C 语言多个内核实现的训练时间与源程序的训练时间。

表 1 2-3-1 网络多内核实现与源程序的训练时间

实现方式	OpenCL 多内核实现 单 GPU 运行		源程序	
训练次数	1000	100	1000	100
耗时（秒）	8.9363425	0.8867170	0.5767116	0.8867170

表 1 的数据显示，多个内核实现与用源程序的训练速度相差很大，还达不到理想训练速度。这是由于多次执行多个内核，增加了 CPU 与 GPU 之间数据传输的时间。

根据数据传输优化原则[6]：数据传输优化要避免无谓的数据传输，并在传输数据量不变情况下提高在 GPU 上计算量的比重。考虑把整个算法在一个内核中实现：

① 在内核代码中用 for 语句实现训练次数循环和每个学习模型迭代；

② 由于目前 OpenCL 规范没有使工作组之间动态同步的阻断函数，所以执行该内核的所有工作项必须在同一工作组中实现同步，在完成算法的某一步后，用同步函数（barrier）使所有工作项同步，即只有一个工作组执行内核；

③ 对于 ATI Radeon HD 5700 Series GPU，由于其 wave 的大小为 64，所以工作组的工作项数为 64 的整数倍时性能更好；

④ 由于每一步所需的工作项数量不同，应通过 if 语句判断工作项的索引是否小于该步骤所需的工作项数量，来确定该工作项是否执行该步；

对给定 GPU，其执行内核的工作组中所能存放的工作项的最大数目是确定的，所以这种实现能利用的工作项是有限的，这是对该方法的一个限制。

3.4 内存访问优化[6]

由于内核的执行效率很大程度上依赖于全局内存，影响性能的主要方面包括访存模式和访存次数，内存访问优化的方法如下。

① 由于最大工作项数量的限制，同一工作项需要多次访问同一内存对象的不同地址，为了很好地利用数据读取的局部性，有效地提升内存读取速度，各工作项必须按顺序以间隔为 1 循环读取内存对象的各地址。该读取模式如图 2 所示。

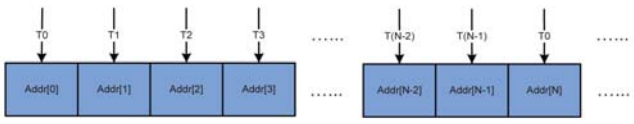

图 2 工作项读取内存对象最佳模式

② 应尽可能使用局部内存减少内存访问时间。由于训练的目的是根据学习模型调整权值和偏置值，所以只有训练次数、学习模型数据、加减速因子和权重、偏置值、学习率的初始值与变化范围必须作为内核的全局变量参数，其它变量应设置为局部变量。需要访存多次的全局变量，应设置局部变量保存其数据来进行操作，以减少对全局变量的访问。

对给定 GPU，其局部内存区的大小是一定的，所以局部内存的使用是有限的，这是该方法的一个限制。

3.5 优化后的测试结果与分析

表 2 是对 2-3-1 网络用优化后的单内核以不同大小的工作组实现的训练时间。

表 2 2-3-1 网络单内核以不同工作组实现的训练时间

实现类型和训练次数	OpenCL 单内核实现单 GPU 运行 训练 1000 次		
工作组大小	64	128	256
耗时（秒）	0.0958330	0.0988771	0.1044600

由表 1 和表 2 比较，经过优化后的内核训练速度大约是源程序的 5 倍。由表 2 得知，对于 2-3-1 网络，64 个工作项的工作组性能更优，这是由于相比更大的工作组，该工作组对于 2-3-1 网络训练各步骤所浪费的工作项更少。

表 3 显示了不同大小的工作组对 2-300-1 网络的使用 RPROP 算法的训练时间，由于 256 个工作项更满足 2-300-1 网络的需要，所以其性能要优于其他两个工作组。但对于各层神经元数量相差很大的网络，运算时浪费的工作项会很多，所以训练的速度大约只有源程序的 2 倍。

表 3　2-300-1 网络单内核实现与源程序的训练时间

实现类型和训练次数	OpenCL单内核实现单GPU运行训练1000次			源程序 训练1000次
工作组大小	64	128	256	/
耗时（秒）	1.0349206	0.9766208	0.9562168	2.3481872

4　结论

由表 1、表 2 和表 3 来看，在人工神经网络各层神经元数量相差不很大的情况下，其训练算法在支持 OpenCL 的 GPU 上运算比在 CPU 上的效率高数倍。但也存在一些限制，由于没有工作组之间的同步，为了减少数据传输消耗的时间，只能使用一个工作组来计算，而指定的 GPU 其执行内核的工作组中所能存放的工作项的最大数目是确定的，所以不能使用更多的工作项来进行计算；由于网络各层神经元数量不一定是 GPU 的 wave 大小的整数倍，可能导致同一 wave 中有的工作项不能被充分利用；由于给定 GPU 的工作组内存大小有限，对全局内存的访问不一定能够完全优化。

参考文献

1 Hagan MT, Demuth HB, Beale MH. 神经网络设计.北京:机械工业出版社,2005.197–244.

2 Wang B, Zhu L, Jia KB, Zheng J. Accelerated Cone Beam CT Reconstruction Based on OpenCL. Image Analysis and Signal Processing (IASP), 2010 International Conference on. 2010.291–295.

3 The OpenCL Specificatin Version:1.0. Khronos OpenCL Working Group. 2008.

4 The OpenCL Specificatin Version:1.1. Khronos OpenCL Working Group. 2010.

5 Programming Guide-ATI Stream Computing OpenCLTM. Advanced Micro Devices,Inc.

6 跨平台的多核与众核编程讲义—OpenCL 的方式.AMD 上海研发研发中心.

(上接第 235 页)

程序运行并拿到控制权，然后完成保护软件的任务[3]。壳分为两种：保护壳与压缩壳。压缩壳的主要功能在于如何优化算法，增加压缩比，进而减少程序的体积，保护能力相对较弱；保护壳则运用各种加密算法和先进的加密技术来保护程序，使得解密变得异常困难，甚至无法脱壳解密。常见的保护壳有 ASProtect、Armadillo、EncryptPE 等，但有加壳就有脱壳，软件开发者最好根据自己的需要，建立模型，设计自己的加壳保护方案。

4　结语

随着软件代码分析技术的不断提升，只要有足够的时间和精力，代码保护技术总是能够被破解的，它只能够增加软件被逆向分析的复杂度，在一定程度上延缓对程序的分析与破解。但是如果一种保护技术的强度强到足以让破解者在软件的生命周期内无法实现完全破解，这种保护技术就可以被认为是成功的。因此，将多种保护技术结合起来将是一种有效的选择。

参考文献

1 Business software alliance.URL:http://www.bsa.org.

2 段钢.加密与解密.第 3 版.北京:电子工业出版社, 2008.241.

3 武新华,张慧娟,李秋菊.加密解密全攻略.第 2 版.北京:中国铁道出版社, 2008.203.