南開大學
Nankai University

计算机学院

并行程序设计期末开题报告

ANN-bp 的并行优化

姓名：林语盈

学号：2012174

专业：计算机科学与技术

2022 年 3 月 28 日

摘要

本文首先介绍了 ANN 反向传播算法的定义。接下来，对此问题的历史并行研究进行了比较和分析。然后，说明了期末作业拟采取的研究方案。最后，对 5 次编程作业的研究计划与安排进行简要规划。

关键字: ANN-bp; Parallel

目录

1 问题定义 **2**

1.1 ANN 模型结构 2

1.2 ANN-bp 算法流程 2

1.2.1 总体框架 2

1.2.2 前向传播 3

1.2.3 反向传播与参数更新 4

1.2.4 结束条件 4

2 历史研究综述 **4**

2.1 向量化架构与多线程架构 4

2.2 多进程、多机架构 5

2.3 GPU 相关架构 5

3 期末拟采取的研究方案 **5**

4 五次编程作业的研究计划 **6**

4.1 SIMD 6

4.2 pthread 6

4.3 MPI 和 openMP 6

4.4 CUDA 6

1 问题定义

ANN，Artificial Network，人工神经网络，泛指由大量的处理单元 (神经元) 互相连接而形成的复杂网络结构，是对人脑组织结构和运行机制的某种抽象、简化和模拟。BP，Back Propagation，反向传播，是 ANN 网络的计算算法。

ANN 可以有很多应用场景，在期末大作业中，拟将其运用于简单的特征分类实际问题中，如根据花期等特征对植物的分类，环境污染照片的分类，或基于已知词嵌入的文本情感分类等。

- 问题输入: 训练样本及其标签数据矩阵
- 问题输出: 模型的参数矩阵，当输入新的训练样本，可用以计算其预测值。

1.1 ANN 模型结构

ANN 的模型结构，如图1.1所示

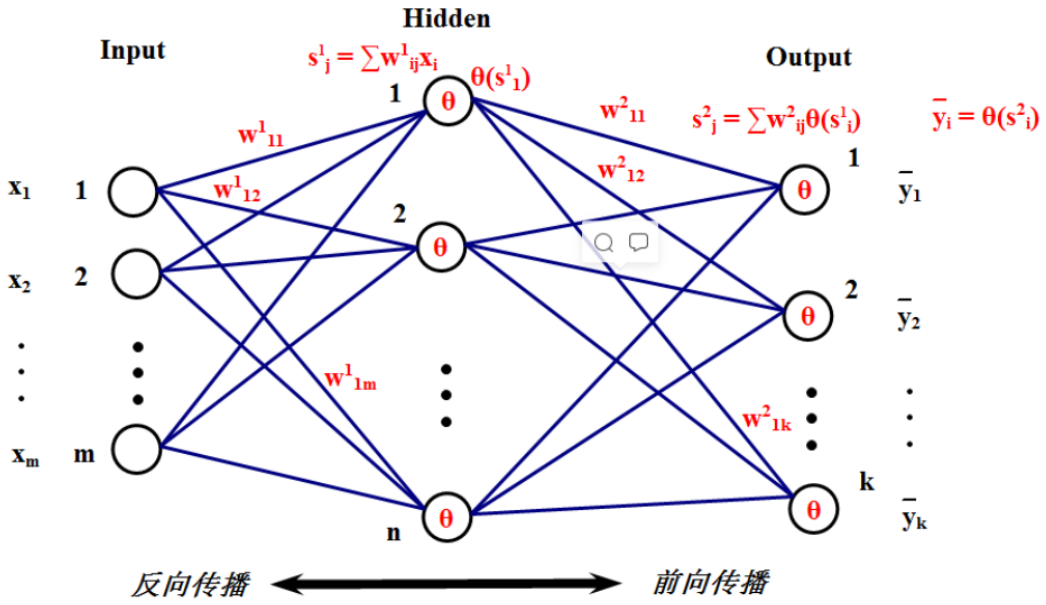


图 1.1: ANN 模型结构

其中，输入层维度为 m，输出层维度为 k，为了简化，假设仅有一个隐藏层，维度为 n。模型的参数分为两部分，W_1 为一个 m*n 的权值矩阵，用于计算输入层到隐藏层，W_2 为一个 n*k 的权值矩阵，用于计算隐藏层到输出层。此外，需要维护一个大小相同的导数矩阵，用于更新权值。

1.2 ANN-bp 算法流程

1.2.1 总体框架

Algorithm 1 神经网络总体框架

Input: 样本输入特征矩阵 X，样本输出标签矩阵 Y

Output: 模型内部参数 W_1、W_1

1: **function** PREDICT($Face$)

初始化模型参数
while 未到达结束条件 **do**:
    前向传播
    反向传播
    更新参数矩阵
end while
end function

1.2.2 前向传播

前向传播即由输入样本矩阵计算出对应的输出标签矩阵，其中每一层的计算即参数矩阵相乘，再加之偏置。

$$\begin{aligned} \boldsymbol{z}^{(l)} &= W^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)} \\ \boldsymbol{x}^{(l)} &= f\left(\boldsymbol{z}^{(l)}\right) \end{aligned} \tag{1}$$

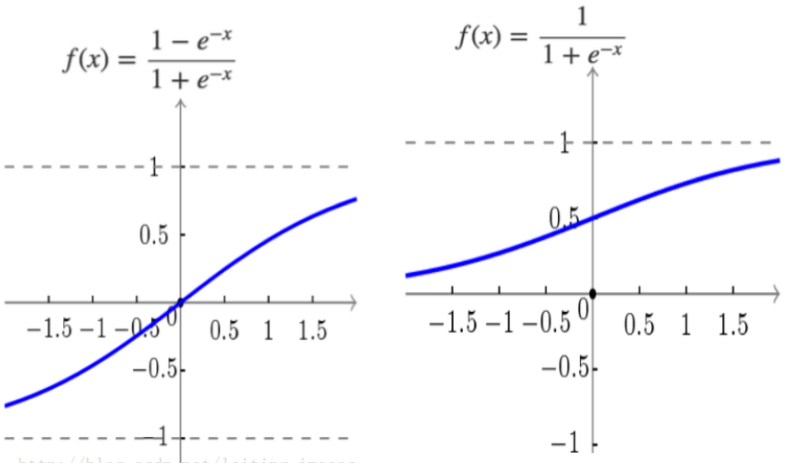
Algorithm 2 前向传播

function PREDICT($Face$)
    $hiddenLayer \leftarrow W_1 \times X + bias$
    $outputLayer \leftarrow f(W_2 * hiddenLayer + bias)$
    return $outputLayer$
end function

其中的 f 函数, 即激活函数，可能有多种选择，它的加入是为增加非线性性。常见的激活函数包括 sigmoid(如图2(a)) 和 tanh(如图2(b))。

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{2}$$

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$



(a) sigmoid (b) tanh

图 1.2: 激活函数

1.2.3 反向传播与参数更新

对于一个训练数据 $\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right)$，首先计算其代价函数:

$$\begin{aligned} E_{(i)} &= \frac{1}{2}\left\|\boldsymbol{y}^{(i)} - \boldsymbol{o}^{(i)}\right\| \\ &= \frac{1}{2}\sum_{k=1}^{n_L}\left(y_k^{(i)} - o_k^{(i)}\right)^2 \end{aligned} \tag{4}$$

所有数据的平均代价函数为:

$$E_{total} = \frac{1}{N}\sum_{i=1}^{N} E_{(i)} \tag{5}$$

采用梯度下降法更新参数:

$$\begin{aligned} W^{(l)} &= W^{(l)} - \mu\frac{\partial E_{\text{total}}}{\partial W^{(l)}} \\ &= W^{(l)} - \frac{\mu}{N}\sum_{i=1}^{N}\frac{\partial E_{(i)}}{\partial W^{(l)}} \\ \boldsymbol{b}^{(l)} &= \boldsymbol{b}^{(l)} - \mu\frac{\partial E_{\text{total}}}{\partial \boldsymbol{b}^{(l)}} \\ &= \boldsymbol{b}^{(l)} - \frac{\mu}{N}\sum_{i=1}^{N}\frac{\partial E_{(i)}}{\boldsymbol{b}^{(l)}} \end{aligned} \tag{6}$$

现计算其中的偏导数，记 $\delta_i^{(l)} \equiv \frac{\partial E}{\partial z_i^{(l)}}$，根据求导的链式法则，用向量形式可表示为:

$$\begin{aligned} \delta_i^{(L)} &= -\left(y_i - a_i^{(L)}\right) f'\left(z_i^{(L)}\right) && (1 \le i \le n_L) \\ \frac{\partial E}{\partial w_{ij}^{(L)}} &= \delta_i^{(L)} a_j^{(L-1)} && (1 \le i \le n_L, 1 \le j \le n_{L-1}) \end{aligned} \tag{7}$$

其中的 L 为神经网络的层数，为了简化，程序中取 L=2

Algorithm 3 反向传播

```
function BACK PROPAGATION(Face)
    计算代价函数 E
    计算导数矩阵 δ_i^(l)
    更新参数矩阵 W、bias 向量
end function
```

1.2.4 结束条件

模型迭代的结束条件可由代价函数的阈值决定，也可设置迭代次数上限。

2 历史研究综述

2.1 向量化架构与多线程架构

1992 年，Wojtek Przytula[13] 等人基于图论方法，提出了将神经网络工作映射到网格连接 SIMD 阵列的系统方法。该方法用稀疏矩阵向量运算来表示多层网络模型，映射适用的计算机类别包括大多数实验性和商用网状连接 SIMD 处理器阵列。

2.2 多进程、多机架构

1995 年，Malluhi 等人提出了一种在超立方体大规模并行机上映射人工神经网络（ANN）的技术[6]。提出了附加树网格（MAT），用于快速实现 ANN，给出了一个将 MAT 结构嵌入超立方体拓扑的递归过程。这一过程被用作在 hy percube 系统上高效映射 ANN 计算的基础。此外，还考虑了带反向传播的多层前馈（FFBP）和 opfield ANN 模型。设 n 是最大层的大小，实了 O（logn）时间的算法，允许多个输入模式的流水线，从而进一步提高了性能。

2004 年，Udo Seiffert[9] 进行了大规模并行计算机硬件和多层感知器与反向传播并行计算机硬件综述，发现 MPI 是并行计算机硬件的一个很好的折衷方案。2008 年，G.Dahl et al.[2] 等人构造了八位奇偶校验、模式并行训练（PPT）和 FANN 工具 +MPI 库，PPT，8 节点版本（65 秒 vs 689）实现了 10.6 的最佳加速。2008 年，Lyle N. Long et al.[5] 等人实现了识别字符集和 SPANN[面向对象（C++）+MPI 库]。SPANN 可以扩展到训练神经元，并允许使用数十亿个权重。它用于小型串行计算机，并允许人工神经网络在一台极为并行的计算机上进行训练，从而显著降低了通信成本。2009 年，R. Rabenseifner et at[7] 实现了集群系统上并行编程、MPI 通信和 OpenMP 的分层硬件设计。2010 年，V. Turchenko et al.[11] 使用 OpenMP 和 MPICH2 进行并行批处理模式培训和并行编程,MPI–Allreduce。2011 年，H.Jin et al.[4] 进行了多区域 NAS 并行基准测试和 NAS 并行基准测试（NPB-MZ）。2011 年，A.Dobnikar et al.[8] 等人实现了基于 Pthreads 与 OpenMP 的并行批训练和顺序并行反向传播学习算法。通过从可用的内核创建尽可能多的服务器线程，可以实现最佳的加速，而使用 OpenMP 并行化训练算法的效果最佳。但是，如果使用的线程多于内核，则从一个线程切换到另一个线程所需的时间非常长，因此并行化的效率相当低。2016 年，Chanthini, P and Shyamala,K[1] 对 ANN 的 MPI 并行进行了总结和展望。

2.3 GPU 相关架构

2010 年，Sierra-Canto 等人 [10] 在 NVIDIA 开发的并行计算体系结构 CUDA 上实现反向传播算法。使用 CUBLAS（基本线性代数子程序库（BLAS）的 CUDA 实现）简化了过程, 使用两个标准基准数据集对该实现进行了测试，并行训练算法的运行速度是顺序训练算法的 63 倍。

在此之后,针对不同的特定任务,人们在 ANN 的基础上进行了特定的优化。包括 Altaf AhmadHuqqani[3] 等人人脸识别神经网络的多核和 GPU 并行化,Junliang Wang 等人 [12] 大数据驱动晶圆生产计划周期时间并行预测等。

3 期末拟采取的研究方案

实现 ANN 反向传播及预测的全部过程，并结合缓存、流水线优化、多线程、多处理器、GPU 进行性能上的不断优化。主要尝试方向包括：不同计算顺序的尝试、反向传播过程中的不同的任务划分、不同粒度的划分等，具有一定的可行性。

同时，计划将其运用于实际问题的求解，在保证分类问题正确性相当的基础上，对比分析性能上的改进。拟将其运用于简单的特征分类实际问题中，即基于已知词嵌入的文本情感分类等。由于 ANN 本身可能不足以学习词语的复杂含义，直接使用已经训练好的词嵌入向量句子作为输入，并使用情感倾向（0、1、-1）作为标签。

4 五次编程作业的研究计划

4.1 SIMD

拟结合 Cache 优化，计算过程中隐层的数据进行复用，对 ANN 反向传播的矩阵运算进行向量化的并行优化。

4.2 pthread

针对反向传播的计算过程，使用多线程进行性能优化。

4.3 MPI 和 openMP

进一步使用多处理器进行性能优化。

4.4 CUDA

ANN 计算任务尤其适合 GPU 的结构，拟使用 CUDA 进行 ANN 的实现。

参考文献

[1] P Chanthini and K Shyamala. A survey on parallelization of neural network using mpi and open mp. *Indian Journal of Science and Technology*, 9(19), 2016.

[2] George Dahl, Alan McAvinney, Tia Newhall, et al. Parallelizing neural network training for cluster systems. In *Proceedings of the IASTED international conference on parallel and distributed computing and networks*, pages 220–225. ACTA Press, 2008.

[3] Altaf Ahmad Huqqani, Erich Schikuta, Sicen Ye, and Peng Chen. Multicore and gpu parallelization of neural networks for face recognition. *Procedia Computer Science*, 18:349–358, 2013.

[4] Haoqiang Jin, Dennis Jespersen, Piyush Mehrotra, Rupak Biswas, Lei Huang, and Barbara Chapman. High performance computing using mpi and openmp on multi-core parallel systems. *Parallel Computing*, 37(9):562–575, 2011.

[5] Lyle N Long and Ankur Gupta. Scalable massively parallel artificial neural networks. *Journal of Aerospace Computing, Information, and Communication*, 5(1):3–15, 2008.

[6] Qutaibah M. Malluhi, Magdy A. Bayoumi, and TRN Rao. Efficient mapping of anns on hypercube massively parallel machines. *IEEE Transactions on Computers*, 44(6):769–779, 1995.

[7] Rolf Rabenseifner, Georg Hager, and Gabriele Jost. Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. In *2009 17th Euromicro international conference on parallel, distributed and network-based processing*, pages 427–436. IEEE, 2009.

[8] Olena Schuessler and Diego Loyola. Parallel training of artificial neural networks using multithreaded and multicore cpus. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 70–79. Springer, 2011.

[9] Udo Seiffert. Artificial neural networks on massively parallel computer hardware. *Neurocomputing*, 57:135–150, 2004.

[10] Xavier Sierra-Canto, Francisco Madera-Ramirez, and Victor Uc-Cetina. Parallel training of a back-propagation neural network using cuda. In *2010 Ninth International Conference on Machine Learning and Applications*, pages 307–312. IEEE, 2010.

[11] Volodymyr Turchenko, Lucio Grandinetti, George Bosilca, and Jack J Dongarra. Improvement of parallelization efficiency of batch pattern bp training algorithm using open mpi. *Procedia Computer Science*, 1(1):525–533, 2010.

[12] Junliang Wang, Jungang Yang, Jie Zhang, Xiaoxi Wang, and Wenjun Zhang. Big data driven cycle time parallel prediction for production planning in wafer manufacturing. *Enterprise information systems*, 12(6):714–732, 2018.

[13] K Wojtek Przytula, Viktor K Prasanna, and Wei-Ming Lin. Parallel implementation of neural networks. *Journal of VLSI signal processing systems for signal, image and video technology*, 4(2):111–123, 1992.