南開大學
Nankai University

计算机学院

并行程序设计期末开题报告

并行加速的深度学习算法

姓名：张铭徐 张惠程

学号：2113615 2112241

专业：计算机科学与技术

2023 年 3 月 22 日

目录

1 摘要

在信息时代，各行各业与计算机结合越来越紧密，在此趋势之下，对计算机性能和算力的要求也更加严苛。尤其是在人工智能蓬勃发展的今天，更是如此。机器学习，深度学习作为全新的人工智能统计方法已经在各种任务中取得了领先于传统方法的成绩。在深度学习的神经网络中，动辄几百万上亿的参数数量也让训练神经网络的成本大大提高。在这种情况下，采用一些高性能计算的方法能够有效缓解这一点。本组的期末实验将应用一些并行编程与算法设计方法来加快神经网络的训练时间。

关键字：并行计算 机器学习 深度学习

2 问题提出

机器学习是当代人工智能技术的基石，成为了当今所有人工智能工具不可缺少的方法之一。机器学习的前身可以理解是统计学，通过学习训练集上数据的分布模式来预测测试集上数据的分布。但通常情况下机器学习算法对于算力的要求极高。以计算机视觉为例，在计算机视觉中，有一个公开的数据集，名为“Image Net”，其由超过 1400 万个带有标记的图片组成，共包含了常见的 2 万余种物体类别。如果开发者开发出了一个算法，想要在 Image Net 上得以应用，则至少需要在一台 8 卡的机器上运行两周时间，仅电费的花销接近万元！所以，在设计机器学习算法时，考虑计算机本身的特性应用并行的方法进行计算可以提高程序运行的效率，减少运行时间进而解决训练成本。此外，近期极为热门的 ChatGPT 也是一个大型模型，其内部所需要的参数大约为 1750 亿！并且训练一轮的成本就接近 400 万美元！

从上面的例子我们可以看出，目前比较火热的 AI 工具无一例外全都是大型的模型，训练成本极高！目前算法的研究人员和 CPU，GPU 的生产厂商已然意识到这个问题的存在，开始着手从底层的算法角度优化机器学习算法。下面我们将总结截至目前并行计算在机器学习中的应用。

3 相关工作与研究现状

3.1 Gorila 优化方法

强化学习是经典的机器学习方法，可以通过试错的方法进行学习，通过学习过后的结果获得奖赏，学习的目的为获得最多的奖赏。在博弈论，信息论，自动控制等领域得到了较为广泛的应用。在早期的训练往往会出现各种各样的问题，包括环境因素的不确定性，随机种子以及算力的制约。在 2015 年 DeepMind 提出了 Gorila[6] Reinforcement Learning Architecture, GRA)，将分布式和并行计算引入了强化学习中。其工作原理如下图3.1所示：

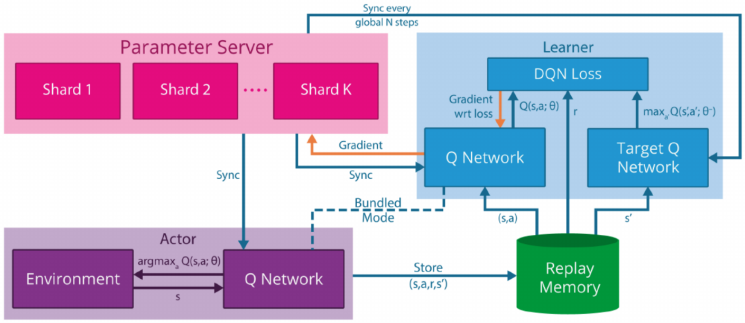

图 3.1: GRA 工作原理

在这篇论文中，提出了若干的优化方法，其中与并行优化相关的有以下几个：

- Experience Replay: 经验回放可以使用多线程的方式并行处理，即多个线程同时存储智能体的经验和从中随机抽样，从而提高数据处理的效率。
- Prioritized Experience Replay: 经验回放的优先级排序可以使用并行计算来加速处理，即多个线程同时对经验的优先级进行计算和排序。
- Trust Region Policy Optimization: TRPO 中的策略更新可以使用多个并行的智能体来计算，每个智能体使用不同的随机种子和参数初始化方式，从而提高优化的效率。
- Proximal Policy Optimization: PPO 中可以使用多个并行的智能体进行数据采样和策略更新，每个智能体使用不同的随机种子和参数初始化方式，从而提高优化的效率。

这篇论文采用了并行优化策略实现了 GRA 框架，使用该框架可以在多种强化学习任务中实现更快的学习速度和更好的性能，包括 Atari 游戏和基于连续动作空间的游戏。此外，该论文还证明了该框架可以扩展到具有非常大的动作空间的游戏，并且可以实现与人类专业玩家相似的表现。最终，该框架被证明是一种有效的强化学习方法，可以用于处理复杂的实际问题。这篇论文具有以下的优点：

- 可以利用多核 CPU 或 GPU 实现高效并行计算.
- 可以通过使用经验池来有效地利用历史数据，提高数据的利用率；
- 可以自适应地调整学习率和其他超参数，以适应不同任务和不同计算资源的需求。

3.2 A3C 优化

在强化学习中，一共有两种方法，一种是 policy-based，另一种是 value-based。2016 年 DeepMind 继 DQN 后在论文《Asynchronous Methods for Deep Reinforcement Learning》[5] 中又将 DNN 引入了 AC 算法，提出了 A3C(Asynchronous Advantage Actor-Critic) 算法，使强化学习可以用于连续动作空间，成为了当时的 SOTA。为了充分地利用多核并行，架构中有多个 worker 在各自的线程中。每个 worker 都独立地进行环境交互与模型训练。

在这篇论文中，采用了以下几种并行优化策略：

- Asynchronous Advantage Actor-Critic（A3C）：使用多个并发的智能体，每个智能体使用自己的经验进行学习，然后将不同智能体的经验汇总起来更新模型参数。
- Asynchronous One-Step Q-Learning（AQ）：使用多个并发的智能体，每个智能体使用一个步长的 Q-learning 算法更新模型参数。
- Asynchronous N-Step Q-Learning（NAF）：使用多个并发的智能体，每个智能体使用一个 N 步的 Q-learning 算法更新模型参数，其中 N 是一个可调节的参数
- 文中还提出了一些与并行计算相关的优化方案，包括将多个智能体的经验合并到一个缓冲区中，减少内存占用；使用非阻塞的异步更新方法，减少训练时间；采用 RMSProp 算法调整学习率，提高训练的效率等等。

算法的实现使用了有 16 核 CPU 的单台机器跑 16 个 agent，花费 4 天学习 Atari 游戏。该方法有以下的优点：

- 异步更新算法可以有效地并行化训练过程，从而大幅提高训练速度和效率。
- A3C 算法具有较好的泛化性能，可以在多个不同的游戏环境中实现较好的学习效果。
- A3C 算法采用 Actor-Critic 网络结构，能够更加高效地学习，同时能够避免其他异步更新算法中可能出现的问题。

3.3 GA3C 优化方法

在前面，我们介绍了 A3C 方法，充分利用了 CPU 的多核并行进行计算，显著性的提高了算法执行的效率，但是显然，上面的方法没有有效的对 GPU 这一运算单元进行有效的利用，于是 NVIDIA 在 2017 年提出了 A3C 算法的的 CPU/GPU 混合版本 [1]，称为 GA3C（GPU Advantage Actor Critic)。和 A3C 算法一样，它也是单机器的算法。A3C 算法使用 CPU 而非 GPU 主要是因为强化学习的序列化特性。训练数据是在学习的过程中产生的，又没有用重复使用内存空间，因此训练和推理的批次都很小。这样用 GPU 在训练的大部分时间中都是空闲的，导致利用率很低。GA3C 中将学习部分统一放到 GPU，将收集数据环境交互部分放到 CPU，会更加高效。A3C 在 16 core CPU 上用 16 个 agent 跑 4 天学习一个 Atari 游戏，而 GA3C 用约一天完成收敛。

3.4 DPPO 优化算法

2017 年时 OpenAI 基于 TRPO 的思想进行改进提出了 PPO 算法，成为了 SOTA，时至今日，仍然在一些问题上得到了广泛应用。2017 年 DeepMind 的论文《Emergence of Locomotion Behaviours in Rich Environment》[2] 尝试将它应用到大规模问题。我们知道，在运动控制场景中如果要学习复杂的行为，设计回报函数会是一个头疼的问题。而这个工作研究通过丰富的环境来帮助复杂行为的学习。它想证明的是通过在多样化的环境中学习，只要简单的回报就可以学习到鲁棒的策略。另外从易到难，循序渐进（curriculum training）学习可以加快学习速度。随之而来的，为了能在丰富的环境中训练，就需要有可伸缩性强的训练架构支撑。于是这篇文章提出一种 policy gradient 的变体，称为 DPPO（Distributed PPO)。它适合大型分布式的高维连续控制优化问题。该架构中数据采集和梯度的计算被分布式地放到 worker 上。参数放于 parameter server 中。这些 worker 在每步梯度计算中会同步这些参数。文中还通过实验比较了同步和异步方式，发现同步地计算梯度均值和更新有更好的结果。

3.5 Accelerated Methods for Deep Reinforcement Learning

传统深度强化学习在训练过程中是存在效率问题的，即当智能体的动作和环境之间的反馈信号具有高度关联性时，传统的基于梯度的优化算法很难高效地优化神经网络的参数。而通过并行计算优化方法可以提高训练效率，减少训练时间，并且还可以在大规模数据集上进行训练。2018 年 Berkeley 论文《Accelerated Methods for Deep Reinforcement Learning》[7] 研究如何在 CPU+GPU 的单台计算机（8 GPU 和 40 核 CPU 的 DGX-1）上优化现有的深度强化学习算法。

本文构建了一个并行化的统一框架。其中所有的神经网络计算（推理和训练）放在 GPU 上进行加速。该框架既适用于 PG 也适用于 Q-learning 方法。基于该框架，文中展示了 A2C, PPO, DQN, Categorical DQN 和 Rainbow 算法的多 GPU 版本。其原理如下图所示3.2。论文基于此动机提出了两种并行算法优化策略：异步优化和多 GPU 并行化。其中作用分别如下：

- 异步优化是通过将多个智能体同时并行运行在不同的 CPU 核心上来进行优化。
- 多 GPU 并行化则是通过将神经网络的不同层分配到不同的 GPU 上来进行优化。

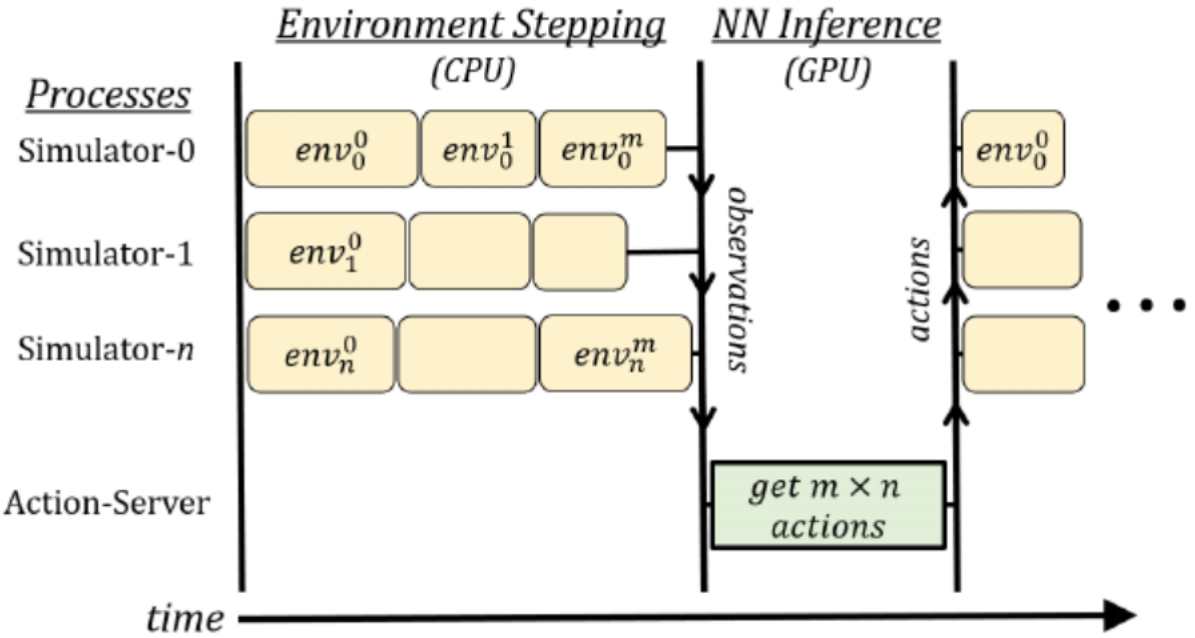

图 3.2: Accelerated Methods for Deep Reinforcement Learning 原理图

本文主要优化点之一在于同步采样以利用 GPU 善于处理大 batch 任务的特点。多个 simulator 以同步方式并行跑在 CPU 核上。它们得到的经验会以 batch 方式给到 GPU 上做推理。由于是单机内，可以用共享内存作为动作产生与 simulator 进程间的通信方式。同步的采样会有 straggler effect，而且当并行进程数量多时更明显，本文用在每个进程中层叠多个独立的 simulator 实例来缓解该问题。每个进程内序列化地执行这些 simulator。如果 simulation 和 inference 的 workload 差不多时，为了防止空闲，可以分两个组。当一组等待动作时，另一组可以执行环境。这样 GPU 交替地服务两个组，从而提高硬件利用率。在同步多 GPU 优化中，所有 GPU 会维护相同的参数值。每个 GPU 使用本地收集的样本计算梯度，并通过 all-reduce 聚合所有 GPU 上的梯度并更新本地参数。在异步多 GPU 优化中，每个 GPU 先本地计算梯度，然后从中央的参数存储（CPU 上）拉最新的参数，并用本地计算的梯度更新它，再将更新的参数写回中央的参数存储中。

但是使用异步优化和多 GPU 并行化也存在一些局限性。例如，异步优化可能会导致不同智能体之间出现竞争，进而导致训练不稳定；多 GPU 并行化需要较高的硬件要求和更复杂的软件实现。

实验结果表明，使用异步优化和多 GPU 并行化的算法可以显著提高深度强化学习的训练效率和性能。例如，在 Atari 游戏中，使用异步优化的 A3C 算法在单个 CPU 核心上的训练时间大约是使用基于梯度的方法的算法的 1/7，同时取得了更好的游戏得分；而使用多 GPU 并行化的 IMPALA 算法可以在不牺牲准确性的前提下，将训练时间减少到使用单个 GPU 的算法的 1/3。因此，这些并行计算优化方法为深度强化学习的研究和应用提供了一种高效的解决方案。

3.6 总结

总的来说，当今人工智能算法日新月异，无论是技术手段还是全新的框架，都在不断的快速更新和迭代，在这个过程中，算法所需要的性能和效率就显得格外重要。更快的速度就意味着可以更快的完成训练集上的训练任务，这也就意味着可以更快的对算法的进行改进更新，更快的研究出一些成果，

提升竞争力。而在所有的提高性能的办法中，并行化是减少训练所需要时间，合理化，充分化应用硬件性能并可以极大降低成本最为行之有效的策略。除去性能的提升，并行化所带给机器学习的，还有整体学习策略的加强，并行操作可以使得训练数据多样化，使得算法的泛化能力更强。目前主流的机器学习框架和方法中都有着并行计算的因素在其中，对传统的机器学习算法进行修改和优化，有助于提高算法的执行效率。

4 研究内容及研究方案

在第三节所讲述的五篇文献都在强化学习工作上做出了一些成绩，作者提出的框架都可以有效的提高算法的执行效率以及减少神经网络训练所需要的时间，极大程度上节约了训练成本。在我们的期末研究中，上面的研究工作可以作为并行优化的思想参考。

由于算力限制以及知识等因素的制约，我们期望于期末研究内容偏于小型化。从最基础的神经网络入手进行优化。目前的深度学习的所采取的，大多都是 numpy 数组做矩阵运算，我们便打算从这里入手，以 MLP(多层感知机)，较为经典的 CNN(卷积神经网络) 为主要的研究背景，详细的探究如何采取并行策略优化，包括但不限于 SIMD，OpenMP，MPI，GPU 编程，SSE 等方法加速神经网络的收敛速度。并通过 profiling 剖析在相同数据集下算法的执行时间及神经网络的收敛速度，以及 CPI，CPU 时钟数等参数来判断并行算法是否针对于原算法有着性能的提高。下面是对研究内容的背景介绍。

4.1 MLP 介绍

神经网络是近年提出的一种全新的数学模型，是计算机科学家们通过模拟人类神经元之间信息传递过程建立的模型，其主要的思路如下图4.3所示。当前神经元按权重接受到前面神经元给定的信息，经过处理后通过激活函数传递给下一个神经元。激活函数便类似于人类神经系统中的冲动阈值，只有当冲动达到一定值时，才能使神经元兴奋，在计算机中仍然是类似的。激活函数可以选取非线性的函数，这使得我们的神经网络可以拟合非线性函数。多层感知机 (MLP) 可以以任意精度逼近任意函数，此理论使应用 MLP 做回归任务成为可能。

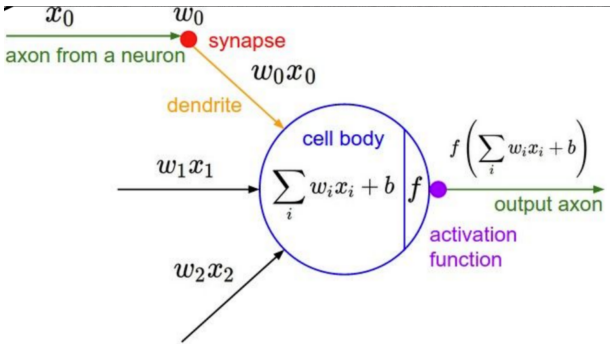

图 4.3: 神经网络原理图

事实上，在机器学习中，MLP 的作用便是以任意精度逼近任意函数，此理论称之为万能近似定理 [4]，定理描述如下：万能近似定理是深度学习最根本的理论依据。它声明了在给定网络具有足够多的隐藏单元的条件下，配备一个线性输出层和一个带有任何“挤压”性质的激活函数 (如 logistic sigmoid 激活函数) 的隐藏层的前馈神经网络，能够以任何想要的误差量近似任何从一个有限维度的空间映射到到另一个有限维度空间的 Borel 可测的函数。正是由于该定理的存在，才使得人工智能后续的一切操作成为了可能。

4.2 CNN 介绍

在计算机视觉领域中，在卷积神经网络 (CNN)[8] 出现之前，其在 Image Net 挑战赛上对图像分类问题的分类正确率一直不尽如人意，自从 2012 年 Alex Net[3] 发布之后，对图像的分类能力得到了跨越式的进步，在此之后，对于图像分类的能力一度超越了经过训练的人类。而 Alex Net 的主要思想内核便是 CNN，采用了卷积的方式提取图像的特征。

CNN 之所以起名叫做卷积神经网络，是因为其神经网络结构中带有卷积层；在数学中两个函数的卷积，本质上是先将一个函数翻转，然后不断的滑动并叠加。在卷积神经网络中，也是同样的思想：卷积层通过卷积核不断的在输入的数据上滑动，并且计算当前位置的权重值，最后进行叠加。整体卷积操作的模式框架如下图4.4所示。神经网络中的卷积，与全连接层一样，可以进行多次操作，也即进行多次卷积操作。但是需要注意的是，我们每次卷积操作过后，都需要加一个激活函数或者叫非线性层。

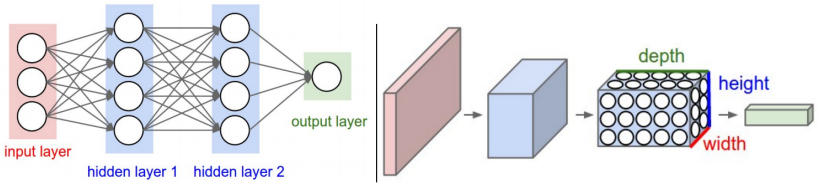

图 4.4: 卷积操作示意图

事实上，卷积操作的本质是提取图像的局部特征，Alex Net 之所以可以在图像分类任务上取得令人惊艳的成绩，就是因为加入卷积之后对于图像局部的特性可以学习的更加清晰。下面将介绍卷积层的具体操作：卷积层中的卷积定义与数学上的卷积极为相似，通过卷积核不断的在输入的数据上滑动，每次计算出相应位置的权重并叠加。卷积层的参数由一组可学习的卷积核（Filter）组成，每一个这种 Filter 在空间上的尺寸都很小，但是它必须要在深度上与输入数据保证同型，例如原本的数据规模为 32*32*3，其中 3 为 RGB 颜色空间的通道数，那么 Filter 的尺寸就必须为 $n * m * 3$，n 和 m 可以任意指定，但是第三维度必须为 3。在 Filter 不断的在输入数据上滑动的过程中，我们需要计算对应位置的点积，最终则会产生一个二维的激活图；在这个图中，每个位置对应的数字可以看做 Filter 在对应位置的相应，或者换句话说，卷积层在不断的提取对应位置的特征。当然，需要注意的一点是，卷积层中可以有很多个独立的 Filter，多个 Filter 的工作模式如下图4.5所示。这些 Filter 都具有独立的参数，即有着独立的映射关系，每一个 Filter 所产生的二维激活图将会在深度方向不断叠加产生最终的输出层，也就是输出层的深度（第三维度）就是 Filter 的数量。

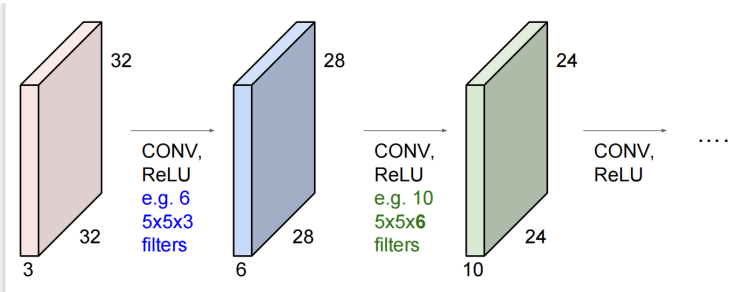

图 4.5: 多次卷积原理图

CNN 之所以可以取得分类问题的桂冠，除去卷积层外，池化层也是必不可少的操作。我们知道，机器学习以及深度学习中，很大程度上需要防止我们训练的分类器出现过拟合的现象。而想要在图像分类等问题上取得良好的效果，防止过拟合是极为必要的。在 CNN 中，为防止过拟合，减少训练所需

要的参数，缩小数据规模，CNN 中引入了池化层：积神经网络重要的层级之一，池化层（Pooling）所起到的主要作用是对输入样本进行 downsamping。

与卷积层相同，池化层同样也需要有相应尺寸大小的 Filter，尺寸大小可以任意指定；目前最常用的池化 Filter 是 2×2 的 Filter，池化 Filter 的规模以及每次池化的步长决定了缩小数据规模的程度，例如刚才所说 2×2 的 Filter，在步长为 2 的情况下就可以缩小 0.75 的数据规模；假设原本是 4×4×n 的输入数据，那么在经过池化层后，就变成了 2×2×n 的输出数据，显然，这缩小了 0.75 的参数。当然，在每个 Filter 内部也依旧有对应的映射关系，和卷积层一样；池化层 Filter 也需要同对应位置的输入数据进行运算，不过不同的是，一般而言池化层的映射关系往往直接指定，而不是通过数据不断的更新迭代。常见的 pooling 有 Max Pooling，Average Pooling 等。在这里我们用 Max Pooling 举例子，如下图4.6所示:

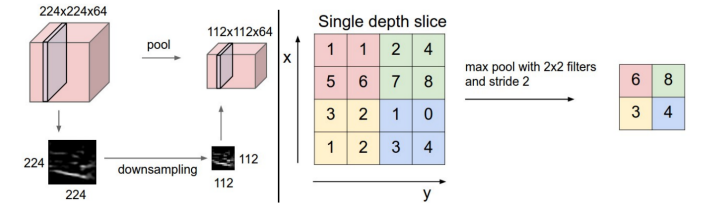

图 4.6: 最大池化示意图

在这里，我们是采用了 2×2 的 Filter，步长（stride）为 2，并且 Filter 内部是取最大值的关系，在上图的左侧，体现的是经过池化层之前以及之后的数据规模，在右侧，体现的是 max Pooling 所做的操作，即在对应位置取最大值。从上面的例子不难看出，池化层也是对数据进行局部采样，得到局部的特征。池化层很显然，也有相应的问题，舍弃了部分的数据可能导致我们的模型欠拟合等问题，但是这些都是可以通过训练轮数的增加等解决的问题。

4.3 研究内容小结

上文详细介绍了研究内容相关知识和背景资料，从资料中，可以发现，我们可以考虑延续原本的算法策略，而仅仅改变其中的实现方法，例如以第一次实验 (体系结构实验) 为例，在进行矩阵运算时，我们可以顺从 C++ 的二维数组的存储形式，采用顺序读取的方法进行优化。通过这些小 tricks，我们期望于可以提升神经网络的训练速度与收敛状态。

5 与日常实验结合的研究计划

5.1 从 SIMD 角度优化

我们知道，CPU 的计算都被抽象成了一系列的 CPU 指令集，有些指令集负责从内存加载数据到寄存器，有些指令集则从寄存器读取数据执行具体的计算操作，然后再由另外一些指令集把寄存器中的数据更新回内存。不同的 CPU 架构和指令集能操作的数据大小是不同的，从 16bit，32bit 到 64bit。在同样的计算精度下，能操作的数据量越大就代表了计算的“吞吐”越大，也就表示更快的计算速度。这正是 SIMD 的初衷。

Single Instruction Multiple Data(SIMD) 指的就是 CPU 在硬件上，支持一个指令读写一个向量 (128bit)，更重要的是，可以对两个向量同时执行计算且计算是可分割的。也就是说可以把 128bit 看成

4 个 32bit 的 float 分别对 4 个 float 执行同样的计算。举个例子，如果我们要计算四对数的和，如下图5.7所示：

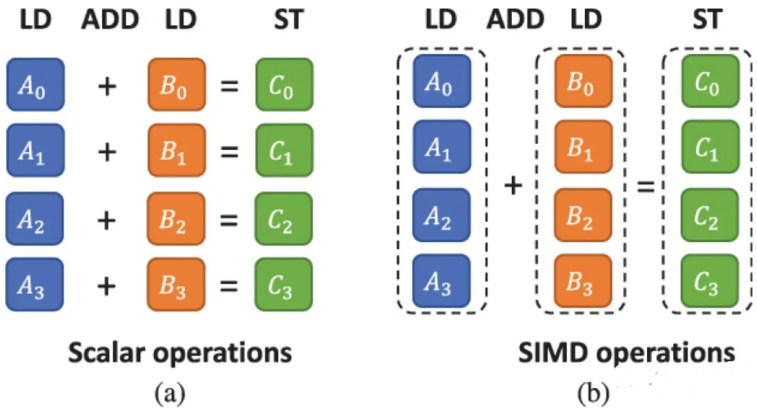

图 5.7: SIMD 加速示意

在没有用 SIMD 时，我们首先需要八次 LD 操作用来从内存把数据加载到寄存器；然后用四次 ADD 操作执行加法计算；最后再用四次 ST 操作，把计算结果存储到内存中，故共需要 16 次 CPU 指令操作。而如果用 SIMD，我们首先需要两次 LD 操作分别把把四个数据加载到寄存器；然后执行一次 ADD 操作完成加法计算；最后用一次 ST 操作把数据更新回内存，这样只需要 4 次 CPU 指令操作。相较于第一种方法性能提升了 4 倍！从上述例子可以看出，SIMD 在可以有效的提升算法执行的效率。

在预计的 SIMD 实验中，我们考虑从 SIMD 角度入手，SIMD 一方面可以在读取的数据中并行化，另一方面还可以在进行运算时达到并行操作的目的。那么在深度学习中，我们可以考虑用 SIMD 加速一些矩阵运算以优化整个算法，在这个实验部分，我们将考虑加速 MLP 多层感知机中，当前神经元接受前面神经元的数据并按照相应权重累加的代码。

5.2 从 Pthread 或 OpenMP 角度优化

OpenMP 是一种用于共享内存并行系统的多线程程序设计方案，支持的编程语言包括 C、C++ 和 Fortran。OpenMP 提供了对并行算法的高层抽象描述，特别适合在多核 CPU 机器上的并行程序设计。编译器根据程序中添加的 pragma 指令，自动将程序并行处理，使用 OpenMP 降低了并行编程的难度和复杂度。当编译器不支持 OpenMP 时，程序会退化成普通（串行）程序。程序中已有的 OpenMP 指令不会影响程序的正常编译运行。

我们在朴素算法中，我们所做的操作往往是在单个核心 CPU 上进行运算，并没有令多个 CPU 协同工作，共同对多个元素进行操作，那么我们便可以通过 OpenMP 对机器学习进行优化，优化的目标可以针对于卷积神经网络中的卷积层进行 OpenMP 编程优化。

5.3 从 MPI 角度优化

MPI 并行编程是一种用于分布式内存系统的编程模型，可以将计算任务分布到多个处理器或计算节点上，并协调它们之间的通信和同步。这种编程模型可以用于优化神经网络的训练和推理过程。

在神经网络的训练过程中，通常需要计算大量的矩阵乘法和向量操作。这些操作可以通过 MPI 并行编程分发到多个处理器或计算节点上进行计算。此外，MPI 也提供了一些用于分布式同步和通信的 API，可以帮助优化神经网络训练的并行化。

在神经网络的推理过程中，也可以使用 MPI 并行编程将输入数据分布到多个处理器或计算节点上进行处理，以提高推理速度。我们在这里，不妨选取参数更新进行 MPI 优化：神经网络的训练过程通常涉及到参数的更新，例如梯度下降。梯度下降需要更新映射矩阵的参数，更新参数的过程可以通过 MPI 并行编程分发到多个处理器或计算节点上进行计算。

5.4 从 GPU 角度优化

我们考虑，如果 CPU 核心数成百上千，那么在 SIMD 等优化策略的加速之下，整体深度学习的收敛速度将会相较于现在提速近百倍！但是很显然，对于民用 CPU 来说，不可能有那么多的核心数，人们无法在有限的成本下创造出核心数更多的 CPU。此时人们发现，原本用来做图形渲染的专用硬件，GPU(Graphic Processing Unit)，非常适合用于并行计算。其原因有以下几点：

- 首先，GPU 在渲染画面时需要同时渲染数以百万记的图形，故 GPU 硬件设计时就利用多种技术优化并行计算以提高渲染的效率；
- GPU 作为专用的硬件，只负责并行计算和并行渲染这一类任务，并不承担复杂的逻辑控制和分时切换等任务，可更专注于计算本身，通过专用指令集和流水线等技术，优化计算效率和吞吐；
- 最后，GPU 作为一类外插设备，在尺寸、功率、散热、兼容性等方面的限制远远小于 CPU，让 GPU 可以有更大的缓存、显存和带宽，可以放置更多的计算核心，而这进一步体现到成本上的优势。

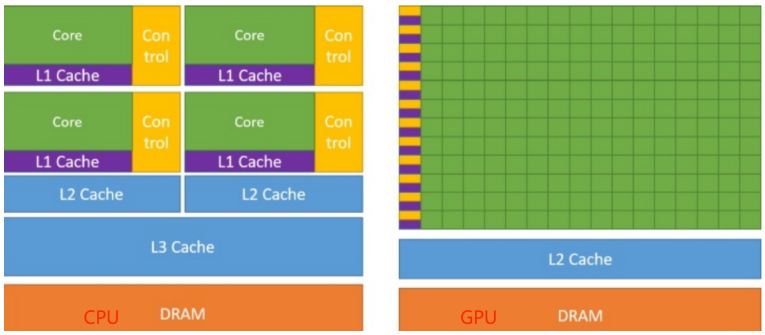

图 5.8: CPU 与 GPU 区别

上图5.8是 CPU 和 GPU 的一个对比示意图。由于 CPU 除了计算任务外，还需要负责大量的逻辑控制，分时调度，兼容多种指令集等等“包袱”，真正用来计算的部分其实很少。而 GPU 则不然，GPU 可以在更大的自由度下，放置更多的计算核心。其实从最本质的角度来讲，GPU 之所以适合并行计算场景，是因为 GPU 使用的是 SIMT（Single Instruction Multiple Threads）模型。SIMT 可以看成是 SIMD（Single Instruction Multiple Data）模型的一种增强。而在本次实验中，我们考虑利用 CUDA 技术来加速 CNN 中的池化操作。

6 小组分工

本次作业采取两人合作的方式完成，所完成的任务量应较为大型，期末研究选取机器学习中的并行优化这一比较大型的研究背景入手，并考虑在落点于具体的实际算法，从 MLP，CNN 等常见且较为经典的神经网络入手进行优化，其中，两人的分工如下所示：

- 张铭徐负责任务如下：
  - 分析任务，调研相关论文文献。
  - 根据文献思想动机考虑实验入手点及预期效果。
  - 根据入手点和思想写出对应代码并予以正确性测试。
- 张惠程负责任务如下：
  - 辅助整理论文文献，总结出论文主要动机及思想。
  - 在通过正确性测试后，进行 profiling 分析。
  - 将 profiling 数据可视化，绘制出对应图表。

论文部分由两人共同完成。

7 总结

总的来说，我们期末实验考虑采用多种优化策略对原本深度学习的部分代码，对深度学习的每一个部分进行 C++ 的复现，并应用 profiling 对原本算法和优化算法进行分析，得到优化算法和原始算法之间的差异。实验试图应用性能剖析工具来分析这些差异的原因，并给出对应的说明。期末的实验将会在平时各个实验的基础上继续加以改进，从其他方面进一步进行优化。希望本学期实验可以完全成功！

7.1 链接

与本学期并行相关的实验代码，实验数据全都会放在以下的仓库中：并行实验项目仓库

参考文献

[1] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256*, 2016.

[2] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, SM Eslami, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[3] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[4] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[5] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[6] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[7] Adam Stooke and Pieter Abbeel. Accelerated methods for deep reinforcement learning. *arXiv preprint arXiv:1803.02811*, 2018.

[8] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.