An Introduction to Modern Deep Learning Systems and Frameworks

Lecture 1 for Advanced Deep Learning Systems

Aaron Zhao, Imperial College London, a.zhao@imperial.ac.uk

Table of contents

Introduction

Introduction - Myself

My name is Aaron, and my research looks at the intersections between algorithm, hardware and security in Deep Learning (DL) Systems.

My email is a.zhao@imperial.ac.uk, and my office is 903.

Introduction - People

The course is also supported by the following people:

- Cheng Zhang (GTA)
- Pedro Gimenese (GTA)
- Mingzhu Shen (GTA)
- Jianyi Cheng (Visiting researcher)

Introduction - Where to find stuff?

Everything is online, you simply need a single url

- The course webpage (`https://aaron-zhao123.github.io/teaching/adls`)
- Bugs should be reported on Github Issues (if it is lab related)
- There are also in-person lab sessions for questions
- Email me (a.zhao@imperial.ac.uk)

Introduction - Why this course?

Learning pure DL theories and model architectures is superficial

You need to understand:

- What is actually running in hardware when you do "torch.matmul"?
- How do these ML/DL frameworks (eg. Tensorflow, Pytorch) work?
- How to deploy your model on different backends?
    - eg. CPUs, GPUs, *etc.*
- How to make them run better or faster? What are the software and hardware tricks for these optimisations?

Hopefully you will have an idea of the answers of these questions.

Introduction - Assessments

The course and the coursework is designed around an in-house compilation framework that we have built, named MASE.

- Practicals (total 30%)
  - Practical 1: includes lab 1 and 2
  - Practical 2: includes lab 3 and 4
  - Answer all the questions in practicals in a single '.md' file.
  - Higher marks would be awarded with more extensions answered/finished.
  - A lab oral with a GTA to show your progress on the practicals (30%).
- A team project (2 people) to pick or self-propose a topic, implement and evaluate the idea on MASE (70%).

This course contains both software and hardware aspects, this is a CHALLENGING course, and you will be expected to write a lot of code!

The lab orals and assessments for MSc ADIC (more hardware focused) and MSc AML (more software focused) students are different. MEngs, feel free to pick your path.

Introduction - Assessments - Lab detail

The purpose of the labs is for you to learn what is involved in a DL framework, we have four lab sessions supported by GTAs:

- Lab 1: Getting familiar with MASE (Practical 1)
- Lab 2: Treating optimization as Passes (Practical 1)
- Lab 3: Automated Machine Learning a Pass search (Practical 2)
- Lab 4 (Software): Network Architecture Search (Practical 2)
- Lab 4 (Hardware): Implement an integer matrix multiplication (Practical 2)
- Questions about MASE functionalities, apart from asking the GTAs in Lab sessions, can also be queried through Github Issues.

Introduction - Assessments - Lab oral

The labs are intended to assist you in navigating and comprehending a substantial codebase. To ensure your understanding, we will facilitate a one-on-one oral (each 15-min) lab session with you that aims to:

- You finish it individually.
- Verify that you have grasped the material thoroughly.
- Ensure that you are engaging with the content rather than merely replicating commands or code.
- Show us your code with your comments.

Introduction - Assessments - Team Project

You will need to find a partner to make up a two-person team. If you cannot find a partner yourself, this is fine, we will find you one.

- A roadmap meeting with a GTA (not assessed).
  - We expect to see an outline of functionalities that you would like to implement, and an arrangement of who is doing what.
  - This is to provide you feedback with your planned work.
- Coursework submission as a pull request on Github (assessed).
  - Your code (correctness and quality)
  - Your augmented testing script (coverage)
  - A short Readme file and a few commands to verify it works (clarity)
- A show me your code session (assessed).
- Documentation on the functionalities you have implemented and also an 8-page report per group (assessed).

Introduction - Assessments - Team Project

- We will provide a list of projects, each project would be assigned with a GTA superviso, each project would be assigned with a GTA supervisor.
- We allow max 3 groups on a single project, first come first serve.
- You can also propose your own group project.
- Your outline of functionalities should describe clearly who is doing what, this would affect your individual mark.
- If your project is merged to the upstream, we will add your name to our annual tech report.

Introduction - Assessments - Team project

With the support of MASE, you can do many fancy stuff, we encourage you to self-propose a team project.

- MASE has a fairly strict CI, ths means when you push on the PR, there is auto-checking, you need to pass all the tests, obviously.
- Our GTAs may ask questions on your code in the PR.
- In this way, I expect you to learn how to work on an open-source project.

The framework is completely open-source, and valuable team projects will be integrated by our team in the upstream, so others can see your contributions on Github. If your code is integrated, the report would be included in an annual technical report that we plan to write on the toolchain.

Introduction - My assumptions

I assume you know

- Basic terminal commands (*eg.* cp, mkdir ...).
- Basic version control knowledge (*eg.* git, github ...)
- Python and Object Orientated Programming
- System Verilog (for hardware-stream students)

If you never heard of these words, and are not willing to pick them up yourself through Google/Youtube/ChatGPT, think twice before you take this course.

Jianyi has prepared an instructional on basic git usage:
`https://jianyicheng-research.notion.site/Git-Tutorial-516864ab8fa04242ad520652744b931f`

Dates and Structures

Important Dates

- Team project sign-up/proposal starts (29th Jan)
- Team project sign-up ends (9th Feb)
- Practical submission (5th Feb)
- Midterm lab oral date (TBC, but around 5th Feb)
- Show me your code session (TBC, but around 18th Mar)
- Report submission and pull request finalization (28th Mar)

Structures - Lectures

You have in total 8 lectures.

The lectures do not necessarily cover all aspects of MASE. The lectures aim to deliver theoretical knowledge, background and an overview of recent progresses in the field of Deep Learning Systems.

I believe you learn coding by doing it (that's why labs exist), and you do good research/engineering by understanding the field and code it up (that's why there are lectures and team project).

The list of team projects would be centered around lectures' contents.

Structures - Labs

You have in total 10-hour labs.

In the first 2 lab sessions, we will cover Lab1 - Lab4.

The other lab sessions are for you to consult with the GTAs for your team projects.

Additional independent study and lab hours are needed.

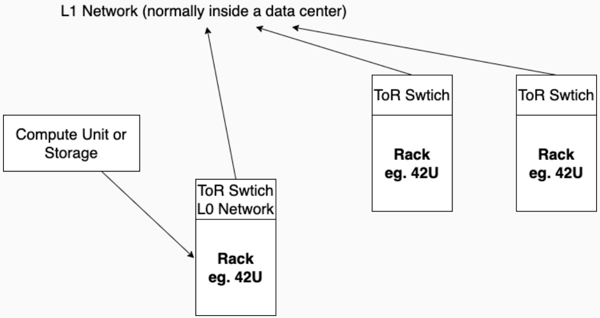
Dates - Lectures and Labs

- Week 2 (15th Jan): Lecture 1 and 2
- Week 3 (22nd Jan): Lab 1
- Week 4 (29th Jan): Lab 2, team project sign-up starts
- Week 5 (5th Feb): Lecture 3 and 4, team project sign-up ddl, Mid-term oral
- Week 6 (12th Feb): Lecture 5 and 6
- Week 7 (19th Feb): Lecture 7 and 8
- Week 8 (26th Feb): Lab 3
- Week 9 (4th Mar): Lab 4
- Week 10 (11th Mar): Lab 5
- Week 11 (18th Mar): Show me your code

An Introduction to Modern Deep Learning Systems

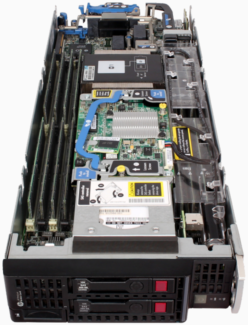
The Limitations on Large-scale Systems

- The absence of shared memory
- Synchronization issues
- Communication latency and network congestions



Blade Servers in a Rack

- Compute hardware (CPU and GPU)
- RAM
- Disks



The Limitations on Hardware

- The end (or slow-down) of Moore's law
- Wire delays
- Power/Cooling is a key limitation (dark silicon)

The slowdown of Moore's Law

The Moore's law states that the number of transistors in an integrated circuit (IC) doubles about every two years.

Fun fact: Gordon Moore is the co-founder of Intel, so this served as the company strategy for a great number of years.

The slowdown of Moore’s Law

- Small feature size is very hard!
- More transistors do not translate directly to performance.
- Parallel hardware is the way to go but instruction level of parallelism (ILP) is limited.

Many vendors are now looking at new technologies such as 3D die stacking and other methods to sustain this growth.

Wire delays

As feature size shrinks, wires become slow relative to logic.

Performance would then be bounded by the amount of data reachable in a single clock cycle, not by the number of transistors manufactured on a chip or the clock frequency of these transistors.

depending upon the processor architecture, cooling technology, and application workloads.

Dark silicon

Dark silicon is the amount of circuitry of an integrated circuit that cannot be powered-on at the nominal operating voltage for a given thermal design power (TDP) constraint.

At 8 nm technology nodes, it is estimated that the amount of dark silicon reaches up to 50–80%

The general trend in hardware computation

Heterogeneous Computing: divide the workload and build specialized hardware for that workload.

In this course, we will look at specialized hardware for ML.

Trends in ML model designs

Concurrently, it seems like it is going from domain-specific ML models to general (or foundation) models.

- The rise of foundation models (eg. GPTs, CLIP).
- The idea of handling multi-modality inputs (eg. Google Pathway).

We are also designing systems that adapts to different use scenarios:

- Federated learning, because of privacy concerns (or not).
- On-device learning.
- the list really goes on

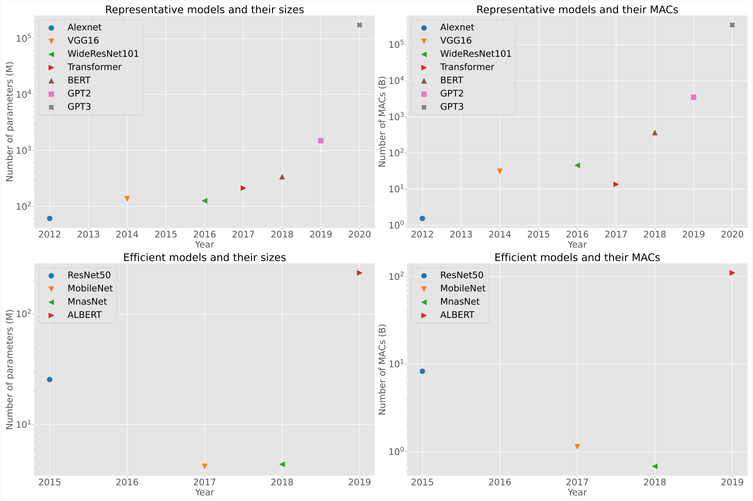
Two key deployment scenarios

We consider mainly two systems: the cloud system and the edge system:

- Difference in scale
- Difference in budgeting (power, speed, reliability, input rate ...)
- Difference in optimization techniques

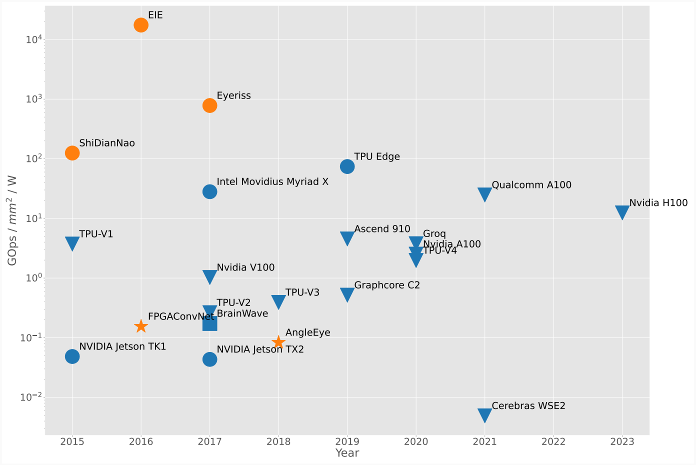
Why ML is a very challenging workload?

- Models are getting larger and larger
- We are developing efficient models, but it requires time



Why ML is a very challenging workload?

- Hardware scaling is a lot slower!
- Basically just Moore's law...

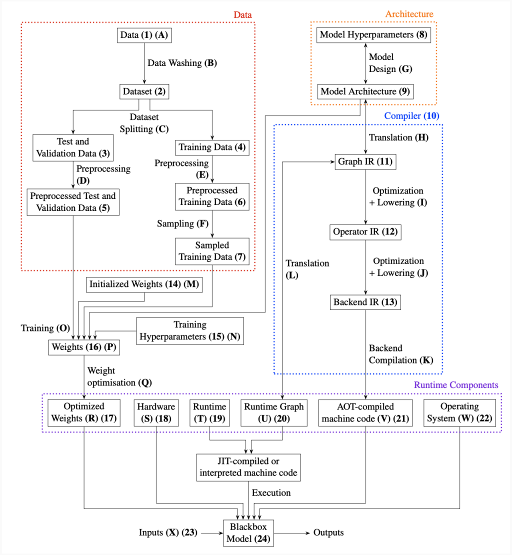


How should we deal with it?

- Software level optimizations (eg. pruning, quantization), typically offers you 2 to 50$\times$ performance gains.
- Compiler level optimizations (eg. loop tiling, loop unrolling)
    - How can we better map the workload to existing hardware, typically offers another 2 to 5$\times$ performance gain.
- Hardware level optimizations (eg. PE design, memory system design), typically 2 to 5$\times$.

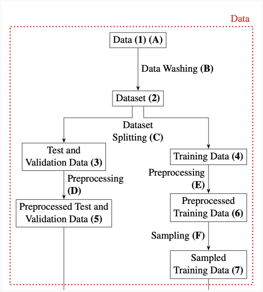
We will explore these optimizations in this course.

An Overview of an ML System



https://mlbackdoors.soc.srcf.net/

ML System: Data



A: The original data

B: The process of removing useless datapoints, outliers, poorly labeled data, and so on.

C: The process of splitting the entire dataset into training data and test/validation data.

ML System: Data

D: The preprocessing of the test/validation dataset, e.g. random rotation and color jittering.

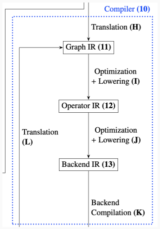
E: The preprocessing of the training dataset, e.g. random rotation and color jittering.

F: The sampling of the training dataset, e.g. to separate it into batches for stochastic gradient descent.

From the system perspective, data may come from sensors (edge systems) or from networked devices (server systems).

Is there enough preprocessing power and bandwidth for receiving the data so we do not starve the computing units?

ML System: Compiler

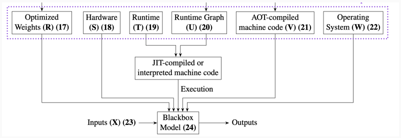


Graph IR is a high-level IR of an ML model. Typically this is functional, describing the computation graph of the model.

Operator IR is a lower-level IR that is closer to machine code, often including explicit parallelism and memory allocation.

Backend IR is the language used by the backend(s) that the ML compiler uses. For example, the CUDA language is a Backend IR for CUDA, LLVM IR is a Backend IR for LLVM, and so on. The ML compiler might use multiple backends, for example if both CPU and GPU can be utilized.

ML System: Hardware


Optimized Weights (R) (17)
Hardware (S) (18)
Runtime (T) (19)
Runtime Graph (U) (20)
AOT-compiled machine code (V) (21)
Operating System (W) (22)
JIT-compiled or interpreted machine code
Execution
Inputs (X) (23)
Blackbox Model (24)
Outputs

If Just In Time (JIT) Compilation is used, we use a Runtime (T) that interprets or JIT-compile the Graph IR (V) to run the model.

- Interpreter: Reads your source code or some IR (bytecode) of it, and executes it directly (normally 1-to-1 mapping).
- JIT compiler: Reads your source code, or more typically some IR (bytecode) of it, compiles that on the fly and executes native code (segments to segments).

If Ahead-of-Time (AoT) Compilation is used, machine code (V) is directly executed on the target hardware.

Summary

- Dates and assessments schedules.
- Limitations on current distributed systems and hardware.
- Trends in ML models and why they are challenging.
- A high-level overview of current ML systems.