Machine Learning Frameworks

Luo Mai

University of Edinburgh
Many ML applications are emerging

Deep neural networks
• Better accuracy
• High computation cost
• Gradient-based training

Diverse applications
• Natural language processing
• Deep reinforcement learning
• Graph neural networks
• ...
Massive computational power is available

**Heterogeneous processors**
- CPUs, GPUs and TPUs
- 10 – 100x acceleration

**Global data centres**
- Easy access to PB-scale data
- 100,000s machines

Three key factors that drive AI booming: Algorithms, Hardware, Data
## ML frameworks: A new category of system software

<table>
<thead>
<tr>
<th></th>
<th>Neural Networks</th>
<th>Automatic Differentiation</th>
<th>Un/semi-structured data management</th>
<th>Training &amp; Inference</th>
<th>Heterogenous Processors</th>
<th>Distributed Execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural network libraries (Theano, Caffe)</td>
<td>✔️</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>Data parallel systems (Spark, Giraph)</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td>ML framework (PyTorch, TensorFlow)</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
System architectures of ML frameworks

**Design Goals**

1. Programming abstraction: Supporting ML in different applications
2. Execution engine: Enable gradient-based computation & parallelise computation
3. Hardware runtime: utilise all heterogeneous processors

**Architecture**

- High-level Front-end Languages (Python)
  - Data Processing
  - Model Library
  - Optimiser Library
  - Model Deployment
- Computational Graph
- Backend Runtime
  - CPU
  - GPU
  - TPU
ML framework programming abstraction

Typical ML Workflow

Dataset → Data pre-processing API → Neural Network API → SGD Optimiser API → Iterative training API → Profiling & Debugging API → Model Serving API

An unified programming abstraction for different ML applications (DNNs, GNNs, DRLs, ...)
Questions?
Expressing ML programs as computational graphs

- labels
- bias
- weights
- features

**MatMul**

**Add**

**Loss**

- Explicit in TensorFlow 1
- Implicit in TensorFlow 2, PyTorch
How to compute gradients automatically?

**Automatic differentiation through the chain rule**: Gradient function takes its primitive function (i) inputs and (ii) output as parameters along with the (iii) gradient of the function outputs with respect to the final outputs.
An automatic differentiation example

labels
bias
weights
features

ggrad: labels

ggrad: bias

ggrad: weights

ggrad: features

Loss

Add

MatMul

ggrad: labels

ggrad: bias

ggrad: weights

ggrad: features

Initial gradient

1.0
Discovering parallelism for better performance

```python
def model(x):
    y1 = op1(x)
    y2 = op2(x)
    return op3(y1, y2)
```

---

**Equivalent Graph**

**Imperative Execution**

**Parallel Execution**

Processor (Multi-core)

Faster execution
Questions?
Frontend and backend languages

**Front-end language: Python**
- Simple and flexible
- Poor performance
- Global Interpreter Lock (GIL)

**Back-end language: C/C++**
- Hardware-friendly
- Excellent performance

---

**Equivalent Back-end Graph**

Python implementation:

```
Data -> op1 -> op2 -> op3
```

C/C++ implementation:

```
Data

Kernel 1 -> Kernel 2 -> Kernel 3
```
Offloading sub-graphs to heterogeneous processors

**Problem:** Frequently launching C++ kernels (e.g., system calls) in Python has large performance overhead

Discovering Sub-graph
- **User annotation** to discover sub-graphs: `@tf.function` (TensorFlow 2), `@jit.script` (PyTorch)
- **Just-in-Time (JIT) compilation:** `@jit.trace` (PyTorch)
Using heterogenous processors

Operators in ML models have execution kernels for CPUs and GPUs

Asynchronous kernel execution
Summary

Benefits

- Simple and flexible frontend
- Full life-cycle support
- Unified expression of computation
- Automatic differentiation
- Enabling backend execution: parallelism, offloading, ...
- kernel dispatchers
- Supporting different processors

ML Systems Architecture

- High-level Front-end Languages (Python)
- Data Processing
- Model Library
- Optimiser Library
- Model Deployment
- Computation Graph
- Backend Runtime
- CPUs
- GPUs
- TPUs
- FPGAs
Reading

• Optional reading
  • [Deep learning with PyTorch in 60 minutes](#)
  • [TensorFlow white paper](#)
  • [PyTorch white paper](#)
Questions?