MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems

Authors
Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, Zheng Zhang

Presented By
Oladotun Aluko

Introduction

- Machine Learning Library to ease the development of ML algorithms, especially Deep Neural Networks
- Computation and Memory Efficient and runs on various heterogeneous systems
- Increasing Scale and Complexity of Machine Learning Algorithms
- Rise of Structural and Computational Complexity

Machine Learning System Considerations

1. Programming Paradigms
- Declarative Programming or Imperative Programming
2. Code Execution Model
- Concrete Execution or Delayed Execution

Machine Learning System Considerations

Declarative or Imperative Programming

- Specifying the computation to be performed or specifying how the computation will be performed

Machine Learning System Considerations

Concrete Execution ot Delayed Execution

- Execution can be concrete, where the result is returned right away on the same thread, or delayed, where the statements are gathered and transformed into a data flow graph as an intermediate representation first, before released to available devices

System Design Approach for MXNet

- The result of combining different paradigms and execution models resulted in MXNet(or “mix-net”)
- The intention is to blend advantages of different approaches. Declarative programming offers clear boundary on the global computation graph, discovering more optimization opportunity, whereas imperative programs offers more flexibility
- In the context of deep learning, declarative programming is useful in specifying the computation structure in neural network configurations, while imperative programming are more natural for parameter updates and interactive debugging
- Embedded in multiple host languages known as the Frontend Languages
- Execution fused into a single backend engine and provides a communication API through the backend

MXNet Programming Interface

1. Symbol: Declarative Symbolic Expression

- Declare a computation graph. Symbols are composited by operators, such as simple operations
- An operator can take several input variables, produce more than one output variables, and have internal state variables.
- A variable can be either free, which we can bind with value later, or an output of another symbol.

MXNet Programming Interface

1. Symbol: Declarative Symbolic Expression

E.g C = A + B; D = C + 1

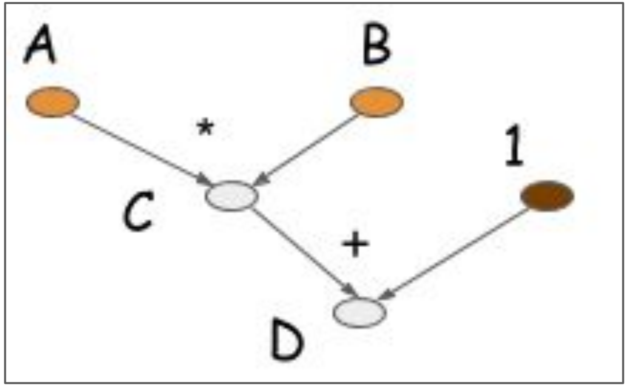

Figure 1: Symbolic Expression

MXNet Programming Interface

2. NDArray: Imperative Tensor Computation

- MXNet offers the NDArray library with imperative tensor computation
- Defines the core data structure for all mathematical computations.

MXNet Programming Interface

3. KVStore: Data Synchronisation over Devices

- The KVStore is a distributed key-value store for data synchronization over multiple devices.
- It supports two primitives: push a key-value pair from a device to the store, and pull the value on a key from the store

Other Modules provided by MXNet

- MXNet ships with tools to pack arbitrary sized examples into a single compact file to facilitate both sequential and random seek.
- Data prefetching and pre-processing are multi-threaded, reducing overheads due to possible remote file store reads and/or image decoding and transformation

MXNet Implementation

1. Computation Graph
2. Dependency Engine
3. Data Communication

MXNet Implementation

1. Computation Graph

- Before computation evaluation, MXNet computes the graph to optimize the efficiency and allocate memory to internal variables
- Graph Optimisation and Memory Allocation
- Ideal strategy for Memory Allocation has $O(n^2)$ time complexity

MXNet Implementation

2. Dependency Engine

- Source units are registered to the engine with a unique tag
- Operations performed, such as a matrix operation or data communication, are then pushed into the engine with specifying the required resource tags
- The engine continuously schedules the pushed operations for execution if dependencies are resolved
- Multiple computation resources such as CPUs, GPUs, and the memory/PCI ebuses, the engine uses multiple threads to schedule the operations for better resource utilization and parallelization

MXNet Implementation

3. Data Communication

- The dependency engine is used to schedule the KVStore operations and manage the data consistency. The strategy not only makes the data synchronization works seamless with computation, and also greatly simplifies the implementation.
- Adopting a two-level server structure:
    - Level-1 server manages the data synchronization between the devices within a single machine
    - Level-2 server manages inter-machine synchronization

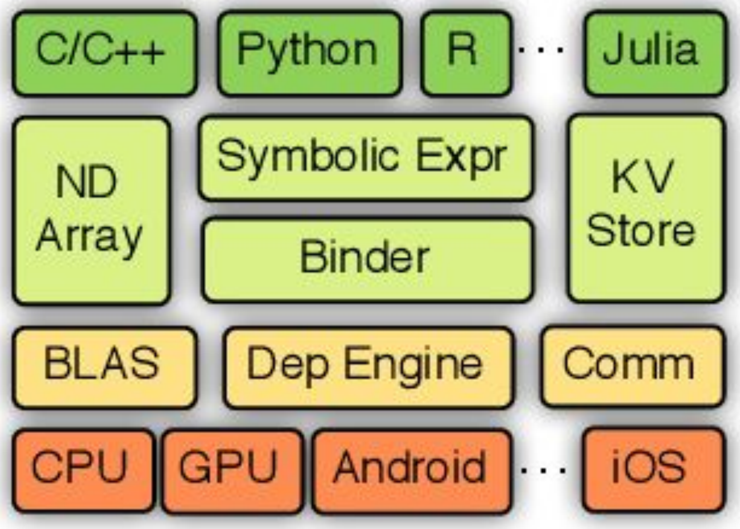

Figure 2: MXNet Overview

Evaluation

1. Raw Performance
2. Memory Footprint
3. Scalability

Evaluation

1. Raw Performance

- Based on convnet-benchmarks. Systems are compiled with CUDA 7.5 and CUDNN 3 except for TensorFlow, which only supports CUDA 7.0 and CUDNN 2. Experiments run on a single Nvidia GTX 980 card

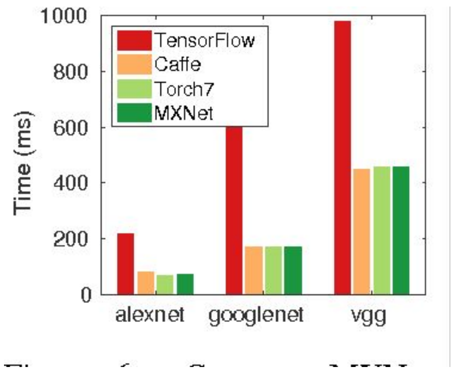

Figure 3: Comparison Graph

Evaluation

2. Memory Footprint

- Both “inplace” and “co-share” can effective reduce the memory footprint. Combing them leads to a 2x reduction for all networks during model training, and further improves to 4x for model prediction

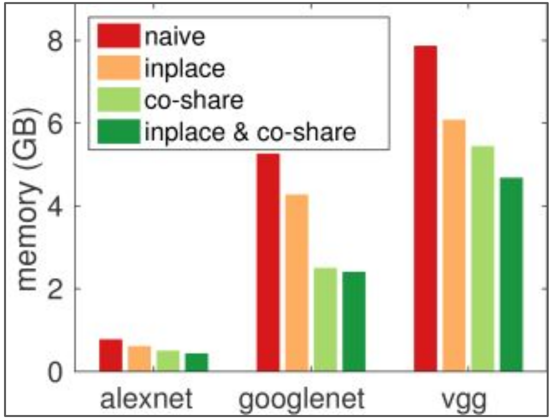

Figure 4: MXNet Memory Footprint

Evaluation

3. Scalability

- Experiment trained googlenet with batch normalization on Amazon EC2 instance, with four Nvidia GK104 GPUs and 10G Ethernet.

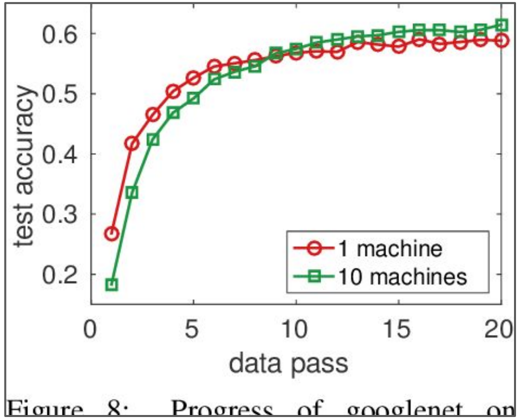

Figure 5: Progress on Multiple Systems

Conclusion

MXNet is a machine learning library combining symbolic expression with tensor computation to maximize efficiency and flexibility

It is lightweight and embeds in multiple host languages, and can be run in a distributed setting

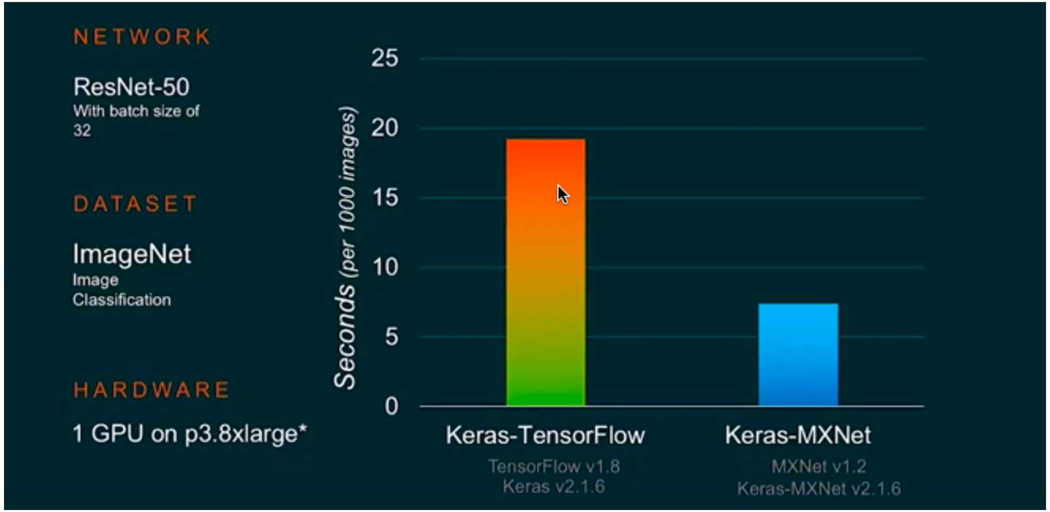

Figure 5: MXNet Eco System

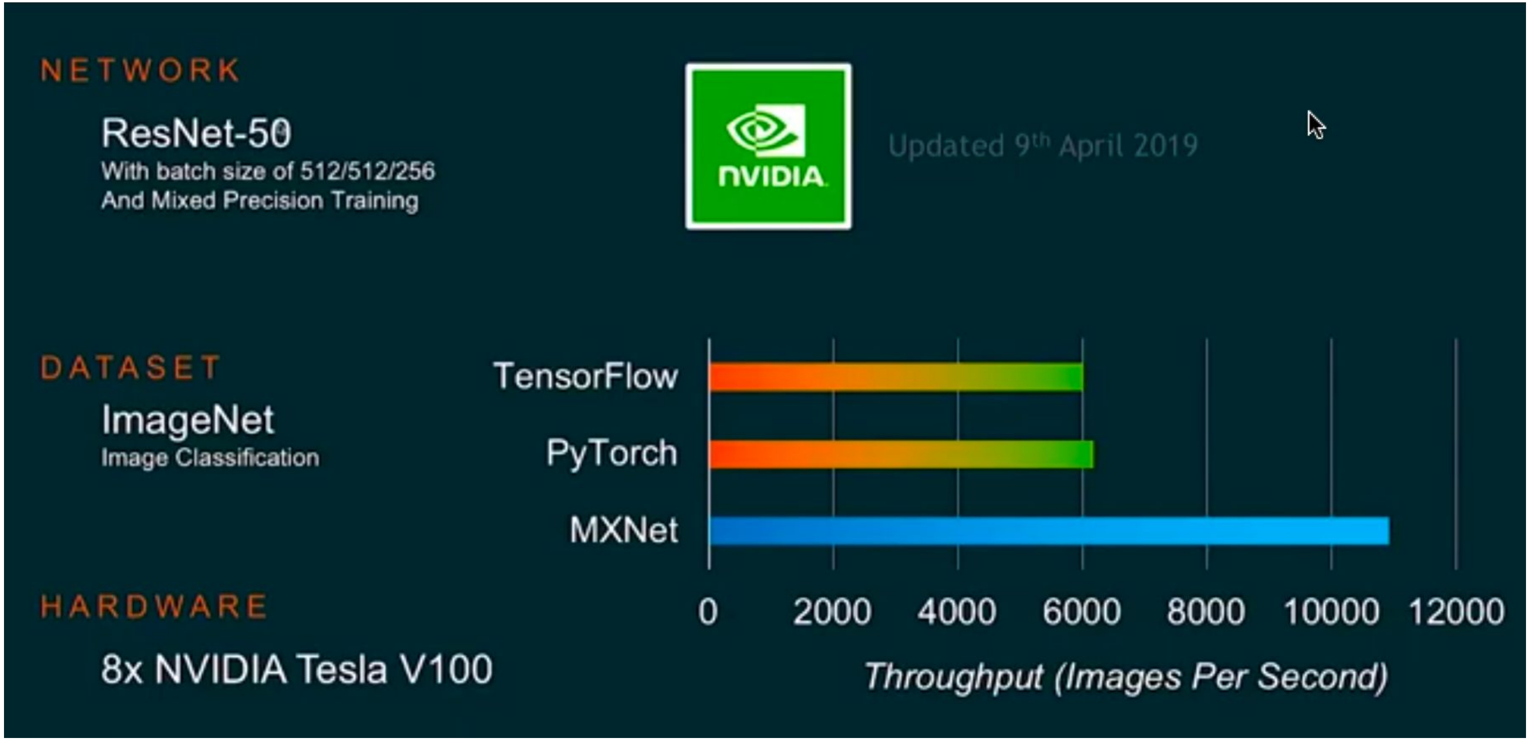

Figure 6: MXNet Nvidia Benchmark

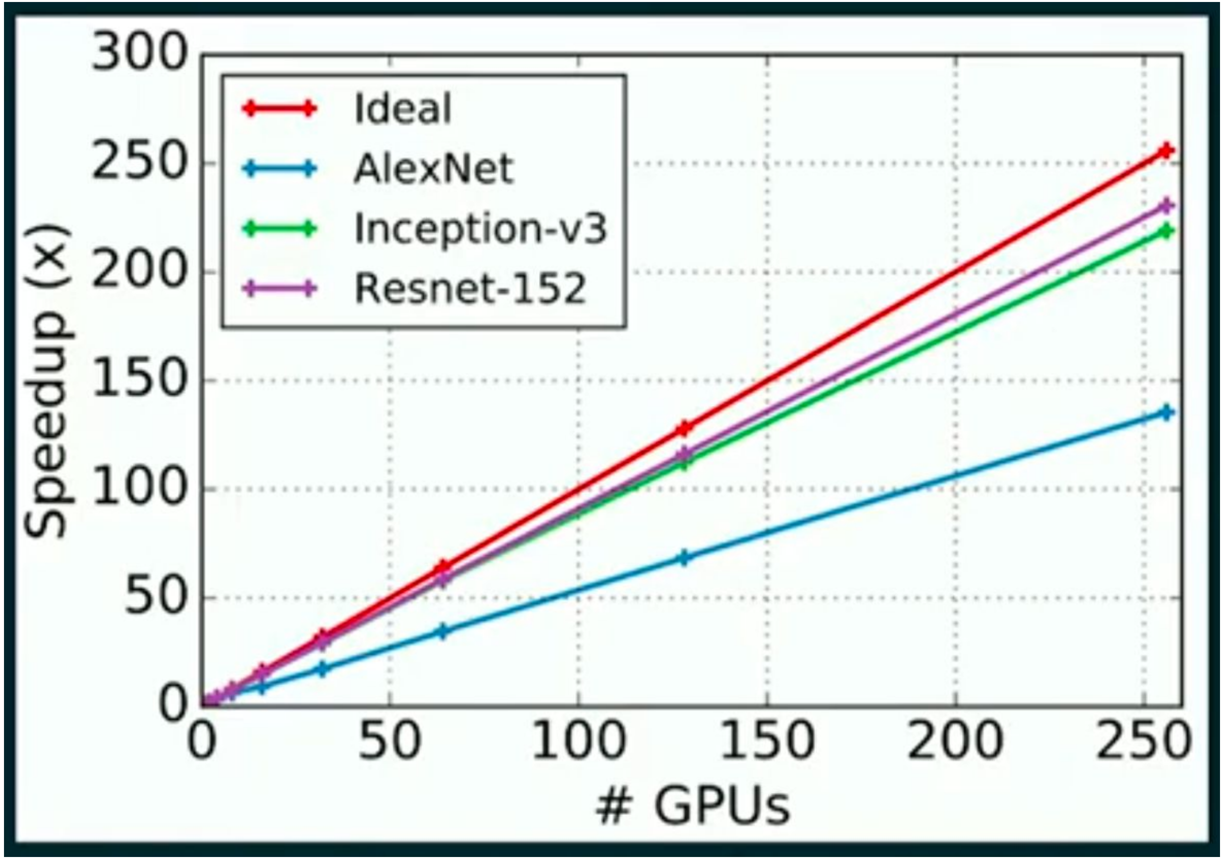

Figure 7: MXNet Distributed Training Results

Thank You :-)