# CSE 598: Assignment 2 TensorFlow vs MXNet

Sreenivasan Ramesh
sreenivasan.ramesh@asu.edu

## 1. Why TensorFlow vs MXNet?

I chose these two systems because these are two most widely used deep learning frameworks in the industry and have support for production-ready deployments. They also have out of the box support for serving models, batch serving, support for mobile, and allow for dynamic model changes with little or no downtime. As we saw in the first lecture, lower latency is imperative in production, and these two frameworks provide excellent support for production ready models. I chose MXNet over the more popular PyTorch as PyTorch is still not widely adopted in industry, and features such as support for quantization and mobile are still experimental as of PyTorch 1.4.

### TensorFlow Overview

Tensors is a deep learning framework that uses a declarative programming paradigm and uses a dataflow graph to represent computations or algorithms. Machine learning models are represented by a directed graph, where nodes perform operations or kernels (optimized computation), and tensors flow along the edges between nodes. Using a computational graph model allows for reverse mode auto differentiation, and the graph can be partitioned into multiple subgraphs which can be executed in parallel on multiple devices. These subgraphs that execute simultaneously communicate through shared variables that stores model parameters (hence no need for parameter server for non-distributed tasks) and queues which allow stateful operations.

On a cluster, we have a set of named processes (*tasks*) that run on different devices (CPU cores/GPU/TPU). A subset of these tasks called *PS tasks*, perform roles similar to that of a parameter server, and in a distributed setting, training parameters are sharded across GPU. But since a PS tasks can run arbitrary graphs; they are more flexible than parameter servers and allows for more experimenting with different architectures.

### Execution Model

A TensorFlow client creates a session and defines a graph. The client then requests the master for execution of the graph, and specifies a feed dictionary of input tensors, and provide a list of edges to fetch the output tensors from. A distributed master places these graph operations on workers devices based on a simple placer, device specific kernel operations, and user constraints, and it orchestrates the execution of the graph. Once the operations are placed, the graph is pruned and partitioned into multiple per-device subgraphs. A dataflow executor present for each task, handles requests from the master and executes kernels that are a part of the local subgraph, executing them in parallel whenever possible. Since there is no concept of the usual parameter servers, tensors flow between subgraphs using Send and Recv kernels which are placed at device edges.

### Dynamic Flow Control

TensorFlow defers execution of the graph, till the entire computation graph is available, so that it can execute an optimized pruned version of the computational graph on accelerated devices. This deferred execution allows us to add conditional and iterative constructs to the dataflow itself, enabling support for advanced models such as Recurrent Neural Networks. This dynamic flow control has an added advantage of supporting auto differentiation, which gets distributed across devices, to compute the gradients parallelly.

## 2. Benefits of TensorFlow

- Dataflow graphs – Using dataflow graphs allow for more optimization.
    - Parallelism – Non overlapping subgraphs can be run in parallel.
    - Distributed Computing – Subgraphs can be placed on different devices across a cluster enabling distributed computing.
    - Device independence – computational graphs are host scripting language independent. They can be checkpointed and restored using the backend C++ engine.
- Distributed Training – TensorFlow supports both Data Parallelism (when data is distributed across machines) and Model Parallelism (where the model is split across machines).
- Control Flow – TensorFlow allows for dynamic control flow, allowing deferred execution, this enabling simpler implementations of advanced models such as RNNs and LSTMs.
- Quantization and Mixed Precision – TensorFlow supports quantization and mixed precision, allowing for faster training on larger networks.
- Accelerator Support – TensorFlow is currently the only framework to have ASIC support via Tensor Processing Units on the Google Cloud Platform.

## 3. Drawbacks of TensorFlow

- Declarative Programming – The declarative nature of TensorFlow makes it much more difficult for debugging and testing new ideas.
- GPU Memory Utilization – By default TensorFlow pre allocates the entire GPU memory, and TensorFlow uses more memory compared to other frameworks such as MXNet [9][15].

## MXNet Overview

MXNet features a mix of imperative and declarative programming paradigms in order to try and mitigate the disadvantages of both paradigms. An imperative approach although more straightforward and easier to experiment with, creates a "*concrete*" execution environment, where statements are executed in the order in which they are specified and are very restrictive from an optimization and parallelization point of view. Declarative approaches on the other hand allow for async operations which allow for more parallelism.

### Hybrid of Imperative and Symbolic Programming

The declarative paradigm in MXNet is supported via *symbols*, which are used to declare the computational graph. Symbols consists of operators (synonymous with TensorFlow kernels) which take input variables, have output variables and have internal state variables. The imperative paradigm is supported via NDArray, which are used to represent and perform tensor computation. NDArray are similar to NumPy's multidimensional array, but provide support for asynchronous computation, lazy evaluation and automatic differentiation.
MXNet provides a hybrid way of programming, where we can program models imperatively using the *HybridSequential* class, and then convert this to a symbolic graph representation using the *hybridize* function. Once converted, MXNet will use the computational graph to achieve better performance.

### Key Value Store for Data Synchronization

MXNet implements KVStore, which acts as a distributed key value store for data synchronization across devices, enabling multi-device training. It is similar to a parameter server, but a Runtime Dependency Engine is used to manage data consistency. The other differentiating factor is that it has two levels of data consistency – one at the machine level, and another at the cluster level, and both levels can use a different consistency model.

# 4. Benefits of MXNet

- Hybrid Programming Paradigm – MXNet supports a hybrid style, allowing to use an imperative style during testing and debugging, and use a hybridize function, to automatically convert the code to a symbolic graph representation for optimization. This symbolic representation makes use of dataflow graphs which allow for better parallel and distributed performance.
- Performance – MXNet performs much better than most other frameworks. It is on par with or slightly better in performance when compared to TensorFlow [7].
- Horizontal Scaling – MXNet achieves almost linear scaling (85% scaling efficiency) on large distributed systems [8] and performs much better than TensorFlow [7][9].

# 5. Drawbacks of MXNet

- Model Parallelism – There is no distributed model parallelism. MXNet supports model parallelism across devices on a single machine only because of how the KVStore is designed [10].
- Flow Control – MXNet does not currently have support for dynamic flow control, and the host language statements much be used imperatively, or underlying modules need to be modified. This makes development of advanced models that require flow control (RNN, LSTM) difficult to implement.
- Sparse Gradient Updates – training very large distributed models efficiently require sparse gradient updates, which the KVStore does not support [1].

## 6. Similarities between TensorFlow and MXNet

TensorFlow and MXNet share a lot of similarities. Both frameworks are written in C++ and

- Declarative programming and dataflow graphs – Both provide support for declarative programming paradigm. This allows both frameworks to represent computations as dataflow graphs.
- Training – They both support distributed, and parallel training, and have support for synchronous and asynchronous training.
- Data Parallelism – Both TensorFlow and MXNet provide data parallelism.
- Both have support for profiling, making it easier to detect inefficiencies in code design.
- Industry readiness – both frameworks provide features such as fault tolerance when training very large models, mobile support, easy deployments, quantization for approximation, batch serving which are geared toward industry use.

## 7. Differences between TensorFlow and MXNet

- Parameter Servers – TensorFlow expands on the concept of parameter servers and use what are called PS tasks; Send and Recv kernels are used for communication between devices. MXNet meanwhile use KVStore (at device and cluster levels) which acts like a parameter server.
- Programming Paradigm – MXNet supports declarative, imperative and a hybrid style of programming. TensorFlow (1.4) only supports a declarative paradigm.
- Flow Control – TensorFlow natively provides operators such as *Cond, Switch, Merge, Enter, Exit*, etc. which provide dynamic flow control. MXNet has no native support for flow control, and host language control flow must be used [11], which comes at the cost of performance.
- CPU utilization: In TensorFlow, each CPU core is treated as a single device, whereas MXNet treats all the CPUs on a machine as a single device [12].
- Memory Utilization: MXNet is heavily optimized for memory usage, and its memory allocation algorithm employs techniques such as mirroring [13], Memonger [14] and in-place or standard memory sharing across operations, which reduces the memory utilization.
- Model Parallelism – While MXNet uses less GPU memory compared to TensorFlow, it only supports model parallelism across devices on a single machine and would be infeasible if model sizes are too large.
- Low Level Tensor Operators – TensorFlow provides support for more low-level tensor operators compared to MXNet, allowing developers for more experimentation.

## 8. Learning Outcomes

Writing this survey, gave me a glimpse into how these systems are designed and I learned the rationale in designing such systems and the tradeoffs which occur, such as the modification to the parameter server architecture and why they are required.

Another learning point is the differences and disadvantages between these two frameworks, and also discovering what features to look for when selecting a framework for distributed deep learning. For example, I now know that I would prefer TensorFlow to MXNet when I want to use model parallelism, MXNet when I want to use a high performant imperative paradigm, or TensorFlow when I want to experiment with new optimization techniques which may require use of flow control and low level tensor operations.

I also got enough knowledge to start experimenting and playing around with distributed approaches for training and serving deep learning models. By going through other resources and repositories listed below, I also learned about good and bad practices to consider while trying to design and optimize such distributed systems.

## 9. References


[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P. "*Tensorflow: A system for large-scale machine learning*". In OSDI '16.

[2] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. *"Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems"*. arXiv:1512.01274, 2015a.

[3] Abrahams, A. "*TensorFlow White Paper Notes*", May 3 2017. Available at https://github.com /samjabrahams/tensorflow-white-paper-notes. [Accessed: January 31, 2020]

[4] Lamberta, B. "TensorFlow Architecture", January 7, 2020. Available at https://github.com/tensorflow /docs/blob/master/site/en/r1/guide/extend/architecture.md. [Accessed: January 31, 2020]

[5] The Apache Software Foundation Apache MXNet, "*MXNet System Architecture"*. Available at https://mxnet.apache.org/api/architecture/overview. [Accessed: January 31, 2020]

[6] The Apache Software Foundation Apache MXNet, "*A Hybrid of Imperative and Symbolic Programming*". Available at https://beta.mxnet.io/guide/modules/gluon/hybridize.html. [Accessed: January
31, 2020]

[7] Karmanov, I. "*DeepLearningFrameworks*", June 19, 2018. Available at https://github.com/ilkarman /DeepLearningFrameworks. [Accessed: January 31, 2020]

8] Werner, V., "*MXNet - Deep Learning Framework of Choice at AWS*", November 22, 2016. Available at https://www.allthingsdistributed.com/2016/11/mxnet-default-framework-deep-learning-aws.html. [Accessed: January 30, 2020]

[9] Synced Review, "TensorFlow, PyTorch or MXNet? A comprehensive evaluation on NLP & CV tasks with Titan RTX", April 23, 2019. Available at https://syncedreview.com/2019/04/23/tensorflow-pytorchor-mxnet-a-comprehensive-evaluation-on-nlp-cv-tasks-with-titan-rtx. [Accessed: January 30, 2020]

[10] The Apache Software Foundation Apache MXNet, "*KVStore for Distributed Model Parallel FC*". Available at https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=89066589. [Accessed: January 31, 2020]

[11] Da Zheng, "*Optimize dynamic neural network models with control flow operators*", July 26 2018, Available at https://cwiki.apache.org/confluence/display/MXNET/Optimize+dynamic+neural+network+ models+with+control+flow+operators. [Accessed: January 29, 2020]

[12] The Apache Software Foundation Apache MXNet, "*Some Tips for Improving MXNet Performance - Intel CPU*". Availabe at https://mxnet.apache.org/api/faq/perf#intel-cpu. [Accessed: January 30, 2020]

[13] The Apache Software Foundation Apache MXNet, "*Environment Variables - Memonger*". Availabe at https://mxnet.apache.org/api/faq/env_var#memonger. [Accessed: January 30, 2020]

[14] Chen, T., Xu, B., Zhang, C., and Guestrin, C. "*Training deep nets with sublinear memory cost*". arXiv preprint arXiv:1604.06174, 2016.

[15] Shi, S., Wang, Q., Pengfei, Q., Xiaowen, C. "*Benchmarking State-of-the-Art Deep Learning Software Tools*". arXiv eprint arXiv:1608.07249