Md Majid Jahangir | SBU ID - 112077145
AMS 560 - Big Data Systems, Algorithms and Networks

Assignment 4 (Paper Reading)
TensorFlow: A System for Large-Scale Machine Learning

Motivation : With extensive research going on in the domain of machine learning, it has led to invention of more advanced and sophisticated novel machine learning models, and due to the availability of large datasets in this web era and software systems that could exploit the heavy computational resources, it had become imperative to have a distributed system for training neural networks to support model representation and training at such a large scale. Even though Google had such a system at their disposal, called DistBelief, it lacked some major flexibilities which developers would often demand, namely in the areas of defining new layers, refining the training algorithms and defining new training algorithms (discussed in details in below section). Also, DistBelief was implemented with the idea of a single platform which was more suited towards training neural networks on huge dataset and in a large scale environment. It was really difficult to scale it down to other environments such as production training, or on a mobile device, or on an online service and would call for implementing new systems satisfying requirements of different platforms. Keeping all these in mind, Google inferred that a new distributed system or framework was required that would provide a single programming model and a runtime system for all the environments, hence, this paper [1] proposed a new system called Tensorflow which would provide a simple dataflow based programming abstraction.

Prior Solutions: One of the prior solutions to Tensorflow is its predecessor DistBelief. DistBelief was a distributed neural networks training system by Google and used a parameter server architecture. Such an architecture consisted of stateless worker nodes that would do the computation and a stateful parameter server that kept track of model parameters. It would define a neural network as a directed acyclic graph (DAG) of layers (a composition of mathematical operators) that would culminate into a loss function (scalar function that quantifies difference between prediction and true values). DistBelief would use worker nodes to compute gradients for each model parameters independently via backpropagation and would write the updates to parameter servers. However, DistBelief lacked 3 flexibilities which TensorFlow sought to provide solution to -

1. DistBelief used C++, a less familiar language amongst machine learning developers, to develop new layers. This was a great impediment for developers who wanted to experiment developing new layer architectures.
2. Refining training algorithms to optimise and accelerate convergence using stochastic gradient descent (SGD) method would involve modifying the implementation of parameter server. Hence, DistBelief was not ideal for such optimizations.
3. Defining new advanced training algorithms such as that of recurrent neural networks , which contains loops; or adversarial networks where two networks are trained alternately, was difficult with parameter server architecture of DistBelief, because parameter server architecture was suited more towards feed-forward neural networks.

Besides, some of the other key limitations of DistBelief had been discussed in Motivation section which motivated Google Brain to develop Tensorflow to address such key issues.
Other prior solutions are Theano, whose programming model is the closest to Tensorflow. Theano would allow the developers to also express or implement a model as a dataflow graph of primitive mathematical operators and would generate an efficient compiled code to train the model. Also, Caffe, another distributed framework, which again enables developers to train neural networks on multicore GPUs and CPUs, whose programming model is closest to DistBelief; and similarly just like DistBelief it was hard to add new layers architectures or optimizers but easier to define models using existing layers. Torch, another similar solution,

unlike all the prior solutions discussed in this section along with Tensorflow, provides a very powerful imperative programming model for computation and training. It enables advanced users to optimise the performance by allowing fine-grained control in the execution order and memory utilization. However, Torch lacks the dataflow graph as a representation of model in small scale or production training environment.

Key Ideas of Proposed Solution: The key ideas of the proposed solution are -

1. Tensorflow uses a single dataflow graph to represent all the computations and state including the operators, parameters, parameter update rules and input processing in a machine learning algorithm.
2. It differs from batch data flow systems in two aspects -
    a. It supports multiple concurrent executions on overlapping subgraphs of the entire graph.
    b. Each vertices may contain a mutable state which can be shared between parallel executions of sub graphs.
3. Dataflow with mutable state allows to make in-place updates to parameters in parallel training steps quickly therefore enabling developers to experiment with optimizations and other parallelization schemes.
4. In a dataflow graph, each vertex will represent a unit of local computation (operations) and each edge (tensors) represents the output , or input to a vertex. A tensor is a multi-dimensional array of primitive data. Therefore, this representation of computation and communication makes it very easy to partition such computations across several machines in a cluster and to execute such independent computations in parallel
5. Operations in the dataflow graph takes one or more tensors as input and produces one or more tensors as output. An operation called Variable, which doesn't take any input and owns a mutable buffer which is used to store shared parameters of the model being trained. This allows, for a quick transfer of an updated shared parameter to a parallel executions and thus allowing such optimizations to improve the performance. Another such operator is queue operator, a simplest queue is FIFOQueue, which owns an internal queue of tensors and allows concurrent access in FIFO order. There are several other queue operators supported such as priority queue, etc.
6. Tensorflow also support dynamic control flow structures through their operations named Switch and Merge to implement conditional or iterative operations which are quite useful in advanced machine learning algorithms such as recurrent neural networks.
7. Tensorflow also supports for partial and concurrent execution of the graph. The API provided by Tensorflow to execute a graph allows the client to specify which subgraphs should be executed, whether zero or more edges to be fed as input tensors and zero or more edges to fetch the output tensors. Each invocation of API is known as a step and Tensorflow allows execution of multiple concurrent steps on the same graph. Through such concurrent executions, subgraphs and computations are able to share shared parameters as and when they get updated by any parallel computations.
8. Distributed execution - The dataflow model enables distributed execution by allowing the same program to be deployed on a cluster of GPUs or CPUs for training, a cluster of TPUs for serving and a mobile device for inference. Each such operation or computation lives on a particular device which is responsible for executing a kernel for each operation assigned to it. Tensorflow runtime will place computations on devices selected by a placement algorithm. Once the computations have been placed and the subgraph been executed, it will partition the operations into per-device subgraph.

Performance: Tensorflow was evaluated to measure the system performance metrics in 4 different aspects -

1. Single machine benchmark - It was evaluated using Chintala's benchmark of convolutional models against 3 other single machine frameworks namely Caffe, Neon and Torch. It was observed that Tensorflow achieved shorter step time in training than Caffe in all the models; same performance as that of Torch because of usage of same

cuDNN library; slower than Neon in 3 of the models because Neon uses a hand optimized convolutional kernel implemented in assembly language.

2. Synchronous replica benchmark - To test the performance of their coordination protocol and implementation, they evaluated the number of null training steps (a worker fetches shared parameters from 16 PS tasks, performs a trivial computation and updates parameters) per second for varying model size and varying numbers of synchronous workers. The median step time for a model of 100 MB increased from 147 ms with one worker to 613 ms with 100 workers; whereas for 1 GB model, the step time increased from 1.01 s with one worker to 7.16 s with 100 workers. This shows their limitation of scaling with synchronous machines.
3. Image classification - For this task, they compared Tensorflow with MXNet using asynchronous SGD. Training throughput improved to 2300 images per second as the number of workers were increased to 200 and on addition of workers the step time increased because of more contention on PS tasks.
4. Language modeling - Tensorflow was evaluated in training throughput, measured in words per second, for varying numbers of PS and worker tasks and two softmax implementation. By adding more PS tasks, the throughput increased because of distributed parallelism model of Tensorflow, but eventually the throughput saturates.

What is the difference between TensorFlow and DistBelief? [Key Question]: DistBelief and its architecture has been covered extensively in “Prior Solutions” section, using its architecture and design from that section, noting down some key differences -

1. Tensorflow is more flexible than the conventional parameter server architecture, DistBelief. Tensorflow is deployed on a cluster as a set of tasks that can communicate over network and there is no such things as a parameter server. A subset of these tasks called ‘PS tasks’ assumes the role of a parameter server; while the others are worker tasks. The flexibility comes from the fact that a PS task is capable to run arbitrary Tensorflow graphs.
2. DistBelief was designed to be run on only one single platform and that was a large distributed cluster of multicore servers and it was very difficult to scale it down to other environments such as mobile device or online service. Using DistBelief for each platform required the developers to create separate systems that would satisfy its requirement. However, Tensorflow provides a single programming model and runtime system and a simple dataflow based abstraction for all platforms and environments.
3. DistBelief comprised very few complex pre-defined layers and additionally for efficiency the layers were implemented as C++ classes, a less familiar programming language among machine learning developers. This would hinder the researcher to develop or add new layer architectures and experiment with optimizations. Also, quite often, in order to optimize and refine training algorithms involved modifying parameter server implementation. However, Tensorflow represents each mathematical operators as nodes in dataflow graph which makes it easier for users to add new layers using a high-level scripting interface and since each layer is built out of simple operators, it becomes easy to optimize many algorithms. Also, since mutable state and the operations updating the state is represented as nodes in graph, experimenting with updates rules also becomes easy which is quite difficult to achieve in DistBelief without modifications to its underlying implementation.
4. Because of the architecture of DistBelief explained in Prior Solutions section and its principle of fixed execution pattern, it is quite difficult to implement advanced new algorithms or models such as RNNs containing loops or adversarial networks or reinforcement models. However, with Tensorflow and its unified dataflow graph to represent computation and state, it is possible to implement such advanced models.

How do differentiation and optimization work in TensorFlow? [Key Question]: Tensorflow provides a user level library that performs a differentiation operation on symbolic expression of a loss function to produce a new symbolic expression representing gradients. The differentiation

algorithm does breadth first search in order to find all the backward paths from target operation, such as a loss function, to a set of parameters and sums up the partial gradients contributed by each path. Tensorflow extended the SGD algorithm to differentiate conditional and iterative sub-computations by adding nodes to graph that is helpful in recording the control flow decisions in forward pass and replaying such decisions in reverse during backward pass. However, in order to curb accumulation of huge amount of intermediate steps in memory during differentiation of iterative computations over a long sequence , Tensorflow developed techniques for managing limited GPU memory on such computations. Through such differentiation operations, Tensorflow developers had been able to implement optimizations such as batch normalization, gradient clipping, etc, to make training faster.
Besides, there are several optimization algorithms which computes new values at each training step for the parameters which can be implemented in Tensorflow using a single write operation, however, there are many more advanced optimization algorithm which are very difficult to express as a single write operation such as Momentum algorithm, and Tensorflow allows such algorithms to be represented easily using Variable operations and primitive mathematical operations without modifying the system.

How does synchronization work in TensorFlow? [Key Question]: Even though Tensorflow was originally designed for asynchronous training, it has already started to implement and experiment with several synchronous methods. During the training of a model, the dataflow graph enables users to change the scheme of reading and update of parameters through three major synchronization alternatives. First, in asynchronous training, each of the worker will read the current parameter values during the start of each step and will apply its gradient to the parameter values at the end of the step ensuring high utilization. However, each step uses a stale parameter value which makes the training less effective. Second, Tensorflow implements the synchronous model of training using queues, where a blocking queue is used to coordinate all executions. The queue ensures that all the workers will read the same parameter values and also there is a per-variable queue which accumulates all the gradient updates from all workers and apply them atomically. However, with this model of synchronization, often stragglers limit the overall throughput as the queue accumulates all the updates from all workers before applying them. To mitigate this issue, a third model of synchronization was introduced by implementing backup workers that are similar to backup tasks of MapReduce. However Tensorflow backup workers run proactively by aggregating first m of n updates, unlike MapReduce which runs in a reactive model after detecting stragglers. This particular model exploits the fact that SGD samples training dataset random during each step, hence each worker processes a different random batch, thus if a batch is ignored, it does not create much issue, but increases the overall throughput by 10%.

Strength :

1. Tensorflow ensures high scalability for machine learning models that are both large as well as small scale and in heterogeneous environment. Therefore, it is being widely adopted and used across teams at Google and outside of Google as well for large scale training and inference.
2. Tensorflow has been designed to be implemented on several backend architecture such as ASICs known as Tensor Processing Units (TPUs), GPUs, multicore CPUs etc. Therefore, because of this application developers have a flexibility in mapping nodes of a dataflow graph across several machines in a cluster.
3. Tensorflow has a massive support from its developers' community because of massive adoption by users and because of this there has been a lot of packages available which lets developers develop applications for difficult machine learning tasks such as voice recognition, question answering, language translations, etc.
4. It supports both large scale training and inference by efficiently using thousands of GPU servers for quick and fast training, and it also runs these trained models for inference on various platforms from distributed clusters of machines in a datacenter to local mobile devices.

5. Tensorflow unifies computation and state management in a single programming model unlike their prior system, DistBelief, enabling application developers to experiment with parallel optimizations and parallelization schemes such as offloading computational tasks to the servers to reduce network traffic. It uses a unified dataflow graph to represent both computations in algorithm and state on which the algorithm is operating. Therefore, by doing this, they have combined the high level programming models of a dataflow and low level efficiency of parameter servers.
6. Tensorflow has also built several coordination protocols and have observed remarkable results using synchronous replication and have therefore opposed the common belief that asynchronous replication is required for scalable machine learning tasks.
7. Unlike traditional batch dataflow system which mandates the input data to be immutable thus making an update of a machine learning model a very expensive operation, Tensorflow represents mutable data along with the operations that operate on that mutable data as nodes in the dataflow graph. This enables the developers to experiment with different update rules and to also allow them to compose multiple layers using a very high level programming interface. Thus, having such layers with predefined gradients makes it easy to apply several optimization algorithms and to also differentiate between the models easily.

Weakness :

1. Tensorflow is not supported on GPUs other than that of NVIDIA's GPUs and the only fully supported programming language is Python which makes the application and usage of it to a limited audience who uses NVIDIA's GPUs or knows just Python as a programming language. Though there are supports for other programming languages such as Java, Go, Swift, etc, however, the support is not full and also without any guarantee of API backward compatibility which makes it difficult to adopt in programming languages other than Python.
2. Tensorflow has not defined default optimization policies to achieve remarkable performance for all types of users, their current implementation and design only enables very advanced and power users to achieve such high performance.
3. Tensorflow has not been designed for applications that require strong consistency.
4. For algorithms like deep reinforcement learning, Tensorflow has a huge limitation in terms of static dataflow graph. In deep reinforcement learning algorithms, structure of the computations becomes known dynamically and therefore the use of a static dataflow graph does not help in such category of algorithms, unlike other tools which provides support for it like PyTorch.
5. There is no support for symbolic loops in Tensorflow which comes quite handy in variable length sequences, unlike other tools such as Theano.

Follow Up Works :

1. Accelerated Machine Learning Using TensorFlow and SYCL on OpenCL Devices [4] - Existing machine learning frameworks mostly target and support NVIDIA CUDA GPUs and there has been very little research in exploring other target devices other than NVIDIA CUDA GPUs using open standards such as OpenCL. This paper proposes and explains that how machine learning applications can harness OpenCL power using open standards and proposes that by using SYCL, Tensorflow can be extended to include operations running on OpenCL devices.
2. TensorLayer: A Versatile Library for Efficient Deep Learning Development [5] - With emergence in usage of deep learning techniques, development of a practical deep learning system has become arduous and complex requiring labor-intensive tasks to construct neural networks, management of large training data, etc. To facilitate such process of developing deep learning system, this paper proposes TensorLayer, a python based deep learning library. TensorLayer provides high-level modules which abstracts

sophisticated operations towards neuron layers, training data and neural models. It has transparent module interfaces allowing developers to embed low-level controls inside the engine.

3. The tensorflow partitioning and scheduling problem: it's the critical path! [6] - TensorFlow imposes iterative calculations on large graphs that need to be partitioned on heterogeneous devices such as CPUs, GPUs, and TPUs. However, such partitioning can not be viewed in isolation,also, each device has to perform local scheduling to select the next graph vertex to be executed. This paper proposes several heuristic strategies to solve partitioning and scheduling issues in Tensorflow, which both are NP-complete but have to be solved in combination in order to minimize overall execution time of an iteration. The paper simulates the performance of the proposed strategies in heterogeneous environments with communication-intensive workloads, common to TensorFlow and states that the best partitioning and scheduling heuristics are the ones that focus on minimizing the execution time of the critical path in the graph.

References :

1. TensorFlow: A System for Large-Scale Machine Learning - Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng, Google Brain
2. The morning paper - https://blog.acolyer.org/2016/12/16/tensorflow-a-system-for-large-scale-machine-learning/?fbclid=IwAR2GvGHUt_uo0g0CVduT8Cl7sjIsc4mGygyxhpm3e3zkPJpYHqZWzfjwO8k
3. http://web.eecs.umich.edu/~mozafari/fall2018/eecs584/reviews/summaries/summary34.html?fbclid=IwAR0Lr2Sxk63_Y3jP4pGV0WTKbdrY7Z3tyJAQZD6dVSfRJLQX0wXqh28KCko
4. Accelerated Machine Learning Using TensorFlow and SYCL on OpenCL Devices : Mehdi Goli, Luke Iwanski, and Andrew Richards - https://dl.acm.org/citation.cfm?id=3078160
5. TensorLayer: A Versatile Library for Efficient Deep Learning Development: Hao Dong, Akara Supratak, Luo Mai, Fangde Liu, Axel Oehmichen, Simiao Yu, Yike Guo - https://dl.acm.org/citation.cfm?id=3129391
6. The tensorflow partitioning and scheduling problem: it's the critical path!: Ruben Mayer, Christian Mayer, Larissa Laich - https://dl.acm.org/citation.cfm?id=3154843