TENSORFLOW : A SYSTEM FOR LARGE-SCALE MACHINE LEARNING

VISWANATH SUBRAMANIAN RAMESH (G01061924)

INTRODUCTION

- TensorFlow is a machine learning system that operates at large scale and in heterogeneous environments.
- Supports large-scale training and inference.
- Runs on wide-range of devices from distributed clusters in datacenters to running locally on mobile devices.
- Supports experimentation and system-level optimizations
- Uses unified dataflow graph to represent computation and state.

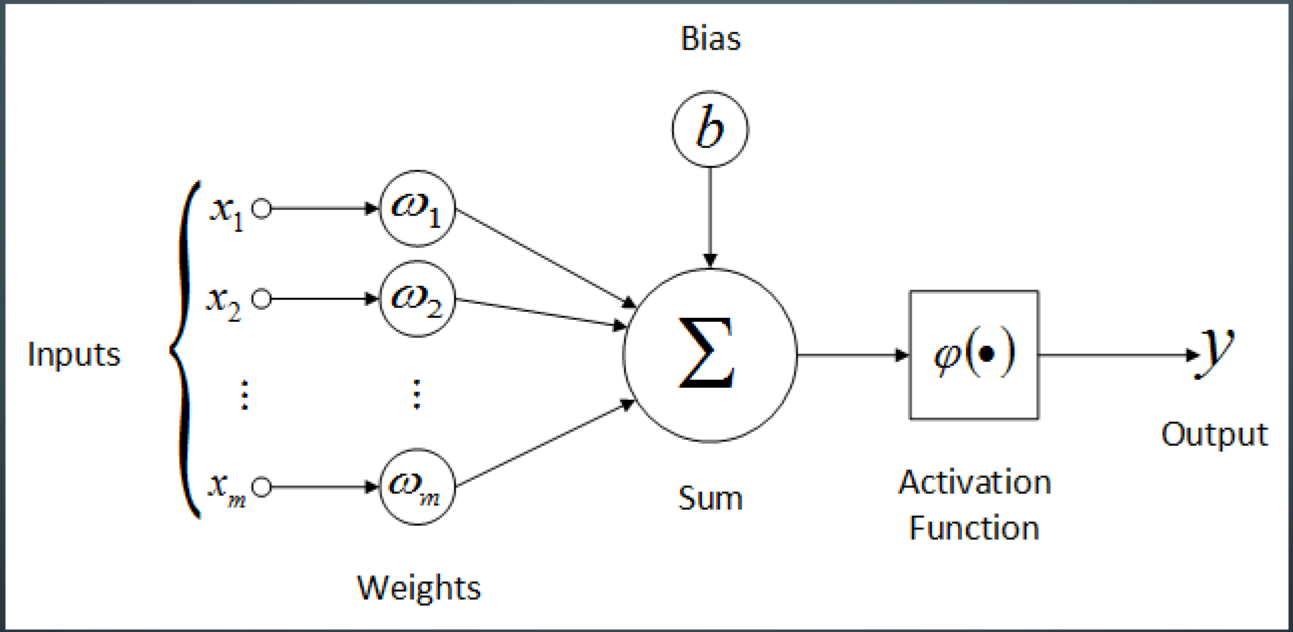
BACKGROUND & MOTIVATION

- DistBelief
- Design principles
- Related Work

DISTBELIEF

- Uses parameter server architecture.
- Stateless worker process performs bulk of computation while training.
- Stateful parameter server process maintains the version of the model parameters.
- Layers and Neural Network.

NEURAL NETWORKS



- Weight matrix and bias(aka parameters) are written to the parameter server which combine it with the current state.

ISSUES WITH DISTBELIEF

- Defining new layers : Language(C++) familiarity.
- Refining training algorithms : Requires modifying parameter server implementation.
- Incompatibility with advances models.
- Inability to scale-down to work with mobile/smaller deployment.

DESIGN PRINCIPLES

- High-level scripting interface(Python API)
- Optimizations without modifying core system.
- Dataflow graph with operators as nodes.
  - Create new layers using the high-level interface
- 2 phase execution :
  - Define NN as graph with placeholder for input
  - Execute optimized version of the program on available devices

- Common abstraction for heterogeneous accelerators
    - Runs on CPUs, GPUs and TPUs(Tensor Processing Units) for ML
    - Methods required :
        - Issuing kernel for execution
        - Memory allocation for inputs and outputs
        - Buffer transfer from and to host memory.
    - Tasks
        - Process that communicate over network
        - Contain devices and export execution API
        - PS tasks
        - Worker tasks

RELATED WORK

- Single-machine frameworks :
  - Single system with CPU and GPU
  - Caffe, DistBelief : Difficulty in adding new layers
  - Theano* : Dataflow graph, efficient code generation for training.
  - Torch
    - Fine-grained control over execution and memory utilization.
    - Lacks dataflow graph for portability over to small-scale usage(experiment, training and deployment).

- Batch dataflow systems
  - MapReduce for ML
  - DryadLINQ adds high-level query and supports advanced algorithms.
  - Dandelion adds code generation for GPUs and FPGAs.
  - Spark adds caching. Better for iterative ML algorithms.
- Primary Limitation : Immutable data and deterministic computation means expensive updating of model.

- Parameter Servers
  - Servers manage and parallel workers update
  - Project Adam : Efficient training of convolutional NNs.
  - Innovations in consistency, fault tolerance, rescaling and performance.
  - MXNet*
    - Uses dataflow graph
    - Parameter server scales training across machines
    - Key-value store interface supports aggregating updates from devices
    - Requires core system modification to support sparse gradient updates

EXECUTION MODEL

- Single dataflow graph to represent computations and state.
- Communications between subcomputations are explicit. Easier for partition and parallel execution.
- Supports multiple concurrent executions on overlapping subgraphs
- Mutable state in vertices that can be shared during execution. Easier in-place updates and propagation.

DATAFLOW GRAPH ELEMENTS

- Tensors
  - N-Dimensional arrays of primitive types
  - Represent inputs and outputs of operations
  - Dense at lowest level(non-zero values).
  - TF provides APIs for sparse to dense and back for usage.

- Operations
  - Vertices representing computations
  - Takes tensors as inputs and produces output tensors.
  - Has a named “type” and compile-time attributes.

- Stateful : Variables
  - Contains mutable states read and/or written upon execution
  - Takes no inputs; produces reference handle ‘r’.
  - *Eg: Read* takes ‘r’ as input and returns value(State[r]) as dense tensor

- Stateful : Queues
  - Supports advanced coordination
  - Eg: *FIFOQueue* – Has internal queue of tensors and allows concurrent access in the said order. Produces reference handle upon execution like variables.

PARTIAL AND CONCURRENT EXECUTION

- Client specifies the subgraph for execution along with edges to be fed as inputs and to fetch outputs.
- API invocations are referred to as steps. TF supports multiple concurrent steps on the same graph.
- Model updates based on the multiple execution instances parallelly.

DISTRIBUTED EXECUTION

- TF runtime places operations on devices based on constraints.
- The placement algorithm computes the feasible device, sets of operations to be collocated etc.
- Users can specify this manually to boost performance
- Send : provide input to tensor based on rendezvous key for the value.
- Recv : Waits for the output value to be available for rend. Key before producing it.
- Session : Maintains mapping to caches graphs for reuse.

EXTENSIBILITY CASE STUDIES

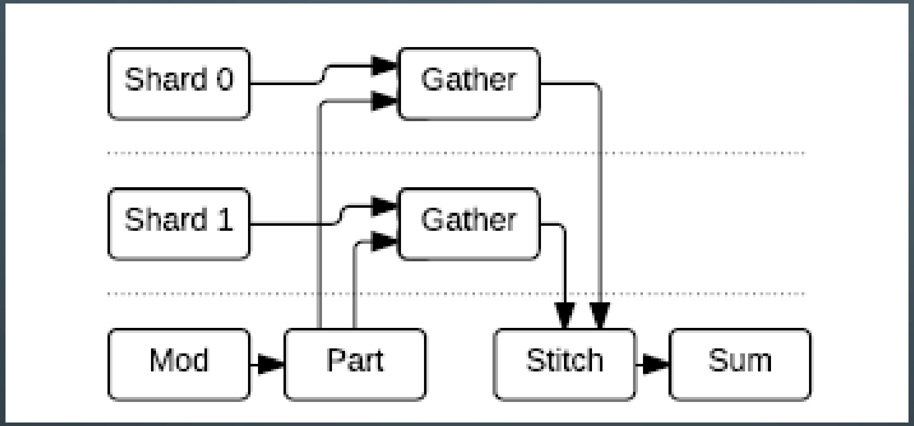
- Differentiation and optimization
- Training very large models
- Fault tolerance
- Synchronous replica coordination

DIFFERENTIATION AND OPTIMIZATION

- Includes user-level library to perform differentiation based on the different inputs. Also uses different techniques to manage the available limited memory.

- Availability of multiple optimizations without requirement to modify the underlying system. Eg. : Write operations that update the values after each computational step.

TRAINING VERY LARGE MODELS



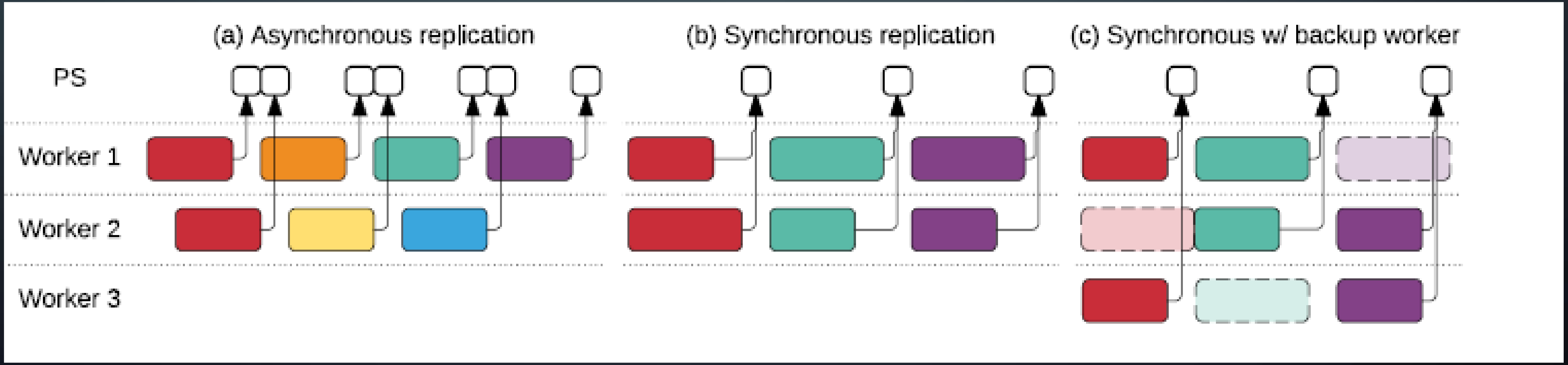
- Implement sparse embedding layers.
- TF generates this graph.
- Gather : Extracts set of rows from tensor and TF co-locates operation with variable.
- Part : Divides incoming indices into variable-sized tensors that contain indices for each shard.
- Stitch : Reassembles partial results into single result tensor.
- Offload arbitrary computation onto devices that have the shared parameters.

FAULT TOLERANCE

- Likely failures or pre-emption. But not very often.
- User-level checkpointing.
  - Save : Writes tensors to checkpoint file.
  - Restore : Reads tensors from checkpoint file
- No consistency checkpoints in library.
- Flexibility for users to specify keep/update checkpoints based on requirement.

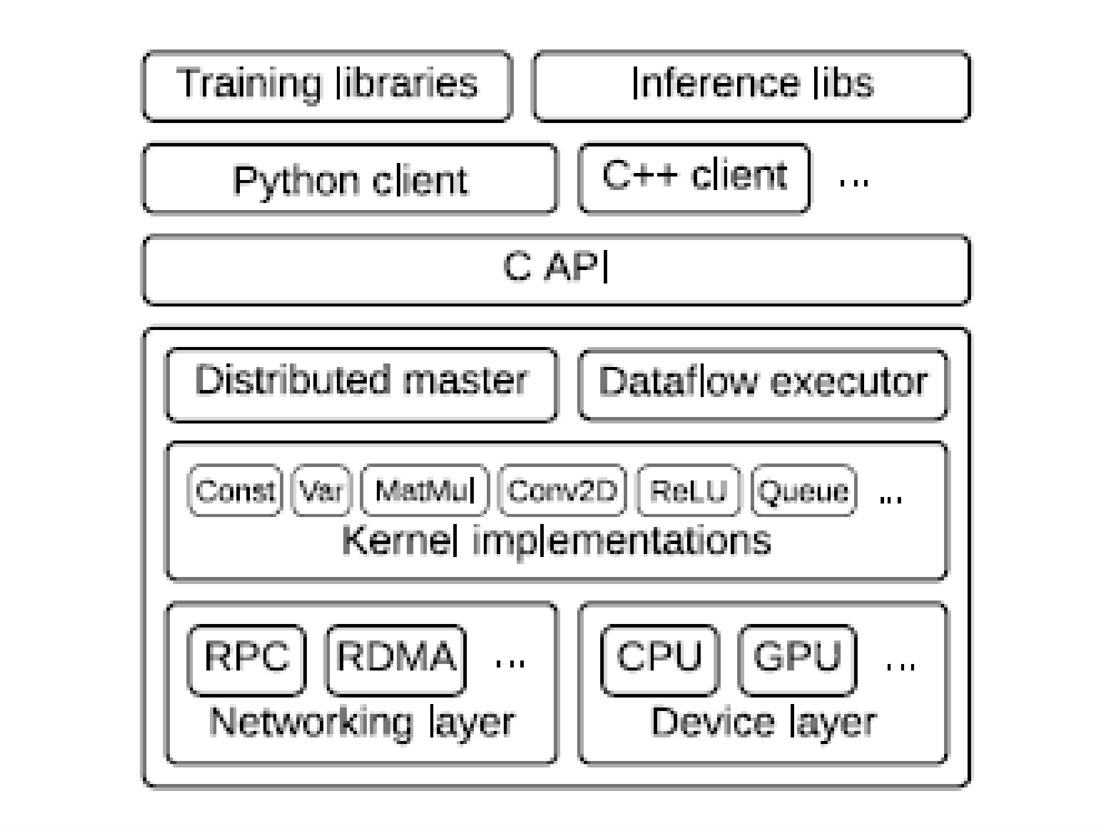
SYNCHRONOUS REPLICA COORDINATION

- Asynchronous training – Higher throughput vs. use of stale parameters for training.
- Synchronous – Using queue. Accumulate gradients. Stragglers limitation.
- Backup workers : proactive execution. Aggregate first m of n updates.



IMPLEMENTATION

- Core library : C++
- C API layer separates user-level code from core runtime.
- Master processes requests, generates tasks, partitions and caches graphs.
- Executor handles master requests and schedules the execution of tasks.
- CPU and GPU use cudaMemcpyAsync() API for data transfer.
- RPC and RDMA used for transfer between tasks.
- Visualization dashboard provided to follow progress and visualize graphs etc.



EVALUATION

- Single-machine benchmarks
- Six-core Intel Core i7-5930K CPU, NVIDIA Titan X GPU
- Training with 32-bit floats.

Library	Training step time (ms)			
	AlexNet	Overfeat	OxfordNet	GoogleNet
Caffe [38]	324	823	1068	1935
Neon [58]	87	**211**	**320**	**270**
Torch [17]	**81**	268	529	470
TensorFlow	**81**	279	540	445

- Neon outperforms TF because of hand-optimized convolutional kernels.

SYNCHRONOUS REPLICA MICROBENCHMARK

- Read shared model parameters from 16 PS tasks + trivial computation + update parameters.
- Scalar performs best. Fetches 4-byte value from each task.
- Dense curves have slowest step times as entire model is fetched.
- Sparse curves show throughput of embedding lookup operation. 32 randomly selected entries from large embedding matrix.

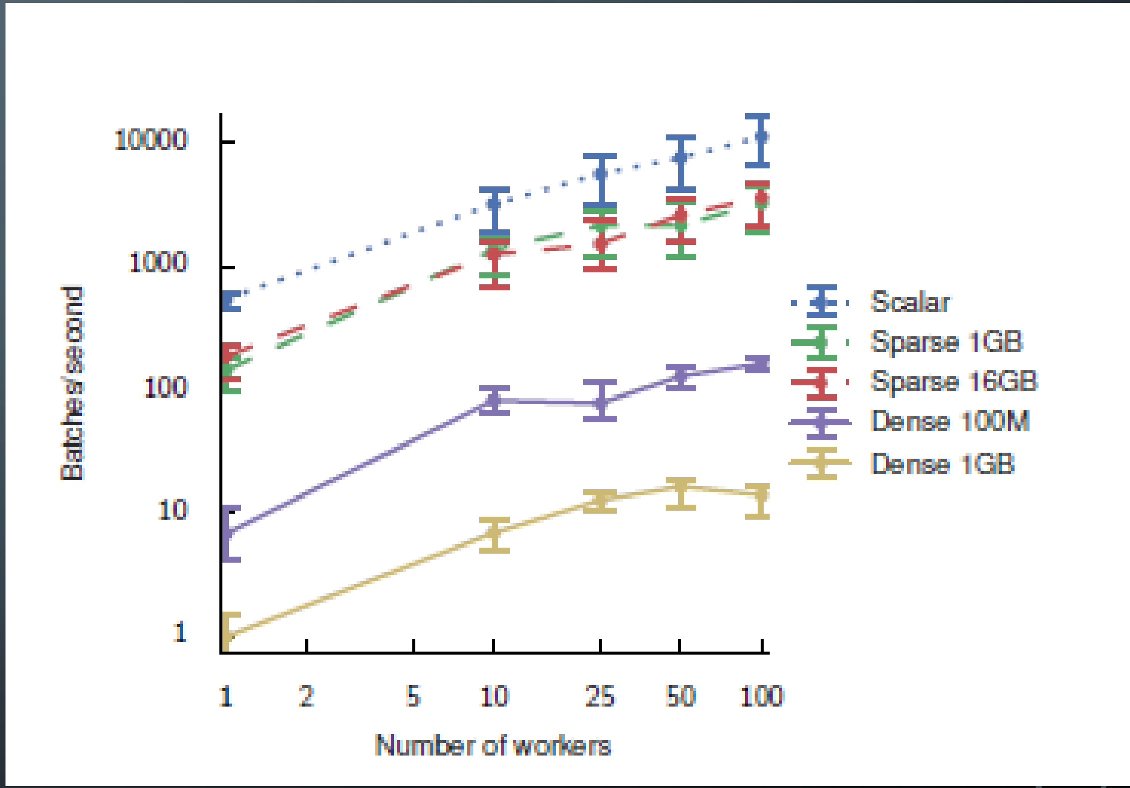
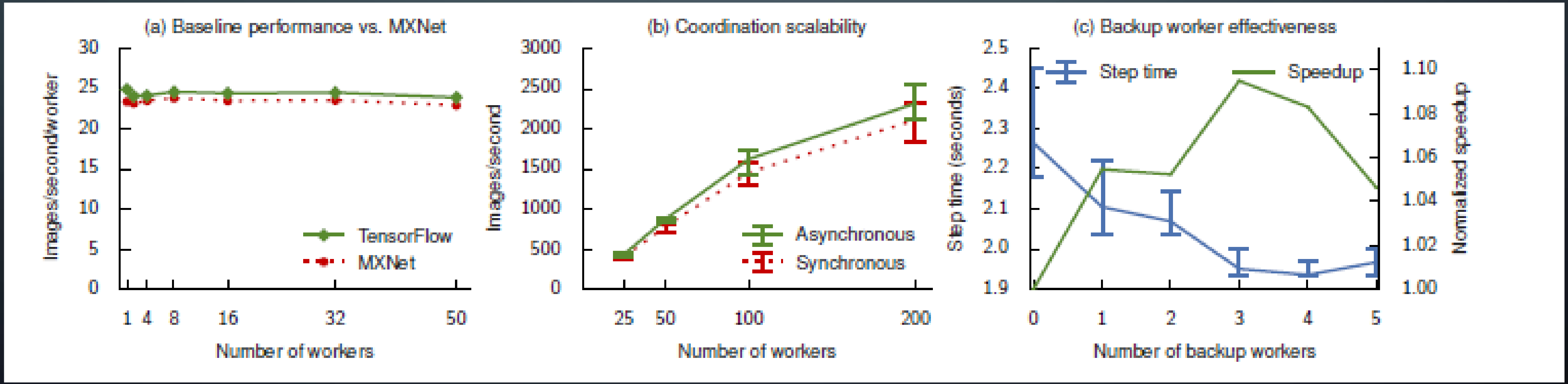


IMAGE CLASSIFICATION

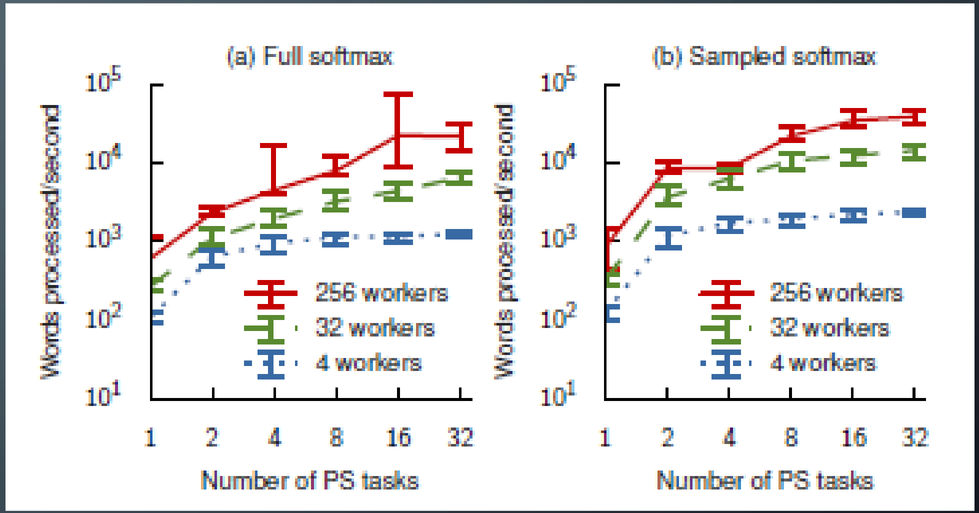
- Google's Inception-v3 model.
- Achieves 78.8% accuracy in ILSVRC 2012 image classification challenge.



LANGUAGE MODELLING

- Speech recognition, text prediction and translational applications.
- Use 40,000 most common words for experiment.
- Full SoftMax : Output * (512 x 40000) matrix
- Adding tasks more effective vs. workers.
- Sampled SM reduces data transferred and computation performed on the PS tasks.
- Sample SM : Output * random true class sparse matrix

          * random false class sample
- Reduces data transfer and computation by a factor of 78.



CONCLUSIONS & FUTURE WORK

- Harness large-scale heterogeneous system for production tasks and experimenting with new approaches.
- Performant and scalable.
- Open-source and already in production.
- Automation of placement algorithm?
- Supporting strong consistency?
- Limitation of static dataflow graph vs dynamic unfolding of computation.