Towards Distribution Transparency for Supervised ML With Oblivious Training Functions

Moritz Meister∗
Logical Clocks AB
Stockholm, Sweden

Sina Sheikholeslami
KTH Royal Institute of Technology
Stockholm, Sweden

Robin Andersson
Logical Clocks AB
Stockholm, Sweden

Alexandru A. Ormenisan
KTH Royal Institute of Technology
Logical Clocks AB
Stockholm, Sweden

Jim Dowling
KTH Royal Institute of Technology
Logical Clocks AB
Stockholm, Sweden

Abstract
Building and productionizing Machine Learning (ML) models is a process of interdependent steps of iterative code updates, including exploratory model design, hyperparameter tuning, ablation experiments, and model training. Industrial-strength ML involves doing this at scale, using many compute resources, and this requires rewriting the training code to account for distribution. The result is that moving from a single host program to a cluster hinders iterative development of the software, as iterative development would require multiple versions of the software to be maintained and kept consistent. In this paper, we introduce the distribution oblivious training function as an abstraction for ML development in Python, whereby developers can reuse the same training function when running a notebook on a laptop or performing scale-out hyperparameter search and distributed training on clusters. Programs written in our framework look like industry-standard ML programs as we factor out dependencies using best-practice programming idioms (such as functions to generate models and data batches). We believe that our approach takes a step towards unifying single-host and distributed ML development.

1 Introduction
Machine learning (ML) is a complex subject, and the process of learning to program (train) ML applications usually involves starting with the simplest possible program, avoiding complexities such as feature engineering and scalability (distributed programming), and slowly adding complexity over time. In particular, moving from single-host applications to distributed applications is challenging, especially for supervised ML as it requires rewriting entire applications. This keeps many developers, who are used to single host debugging and testing and have limited knowledge about distributed environments, from discovering the benefits of distributed ML: faster hyperparameter sweeps and reduced training times.

The contribution of this paper is the design and implementation of a framework that unifies single-host and distributed training functions based on an abstraction we call the distribution oblivious training function. We make training functions reusable by following the dependency inversion principle [6] to factor out those aspects of training functions that are subject to change between single-host and distributed applications. We demonstrate our framework for Keras/TensorFlow (TF) programs, but the approach generalizes to other frameworks that support distribution, such as PyTorch.

2 Distribution Transparency in ML

Transparency in distributed systems [8] refers to hiding distribution-specific aspects of an application from the developer - for example, a developer invoking a function may not know (or need to know) if the function she is calling is local to her application or on a remote server. Distribution transparency enables developers to write code that is reusable between single-host and distributed instantiations of a program.

In supervised ML, the core logic that is common across all programs is the training function - a series of steps including defining a model architecture, then ingesting labelled training data and feeding it to the model and iterating until some termination (or convergence) criteria are met. The output of the training function is a model that can be used to make predictions on new data, drawn from the same distribution as the training data.

Training functions, however, can be used in many different contexts when we distribute supervised ML programs: single host notebooks, distributed hyperparameter search, parallel ablation studies, and distributed training are common examples. However, existing frameworks for supervised ML, such as Keras/TensorFlow and PyTorch, require training functions to be rewritten to account for the distribution strategy, what accelerators the computations are scheduled on, and whether the optimizer needs to share its results with other hosts (for distributed training). In figure 1, we illustrate how training functions can be used - as part of (1) the inner loop when the same training function is either run on a single host or on many hosts in parallel (as part of data-parallel distributed training) using (distributed) stochastic gradient descent, or (2) the outer loop when the training function is run on different hosts for example

∗Correspondence to: M. Meister <moritz@logicalclocks.com>.
which a global optimizer (can be user-defined) draws the hyperpa-
ners and model architecture can vary, and code needs to be rewritten
to account for how many hardware accelerators are being used. For
the outer loop, the variable aspects related to model configuration
are controlled by a global optimizer, such as a Bayesian optimizer
proposing different configurations (trials), or an ablator, generating
trials by leaving one or more components out at a time. The trials
can be run in parallel on a cluster and the results collected by a
global optimizer or ablator.

<table>
<thead>
<tr>
<th>Process Step</th>
<th>Distributed aspects of the Training Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>HParam Search</td>
<td>Model Architecture, Hyperparameters, Early Stopping, Optimizer, Regularization</td>
</tr>
<tr>
<td>Ablation Study</td>
<td>Model Architecture, Features (Input Data), Regularization, Optimizer</td>
</tr>
<tr>
<td>Distributed Training</td>
<td>Features (Input Data), Optimizer, Acceleration-awareness</td>
</tr>
</tbody>
</table>

Table 1: Distributed aspects of the training function that need to be re-written for different distributed contexts

The distribution context and environment can be initialized out-
side the training function (it is oblivious to it) to make appropriate
use of the resources such as accelerators. Other means to achieve
transparency of the two loops include the use of pluggable hooks,
such as the Keras/TF callbacks.

4 UNIFIED EXECUTION FRAMEWORK WITH JUPYTER NOTEBOOKS ON HOPSWORKS

With Hopworks [4] and the Maggy framework [1][7], we provide
a unified development and execution framework for distribution
transparent Jupyter notebooks [5]. That is, the developer writes
a Jupyter notebook that can be run/debugged using a single host
Python kernel, and the same notebook can also be run on a clus-
ter using many hosts and hardware accelerators as a PySpark ap-
lication. The developer only needs to set a distribution context
parameter that controls which cells to run in the notebook - the
oblivious training function is a single cell used by all the different
distribution contexts. The notebooks can also be parameterized and
run by an external workflow manager (Airflow) in production ML
pipelines, similar to Papermill by Netflix [9].

5 RELATED WORK

Previous work on this topic can be categorized in three dimensions:
Pipeline orchestration, ML lifecycle management and automated
ML (AutoML). Pipeline orchestration covers the aspect of taking
an entire ML pipeline into production, which includes data prep-
ration and engineering, modeling, training, serving inference and
managing the deployments. TensorFlow Extended (TFX) [2] is a
TF based platform with the goal of minimizing glue code between
these pipeline steps. Compared to the previous category, ML life-
cycle management is concerned with the iterative nature of the
ML development process. By tracking artifacts, logs and experimen-
tials, results can be easily reproduced, making the process more
transparent with respect to the trained models themselves. MLFlow
[10] achieves this by allowing the user to make explicit calls to
log this meta-data. AutoML aims to automate every aspect of the
pipeline. However, due to the high computational requirements,
recent work was focusing on the automation of the separate steps
first. Because many parts of a ML model behave like a black-box
and can be encoded in hyperparameters, one can fall back on search
for optimization of such parameters [3].

6 SUMMARY

In this short paper, we introduced the distribution oblivious training
function for supervised ML and showed how it can be used to
write distribution transparent ML programs. In the Hopworks
platform, this provides developers with a unified framework and
codebase where Jupyter notebooks can first be developed as single-
host Python programs, then extended to distributed contexts, and
iterative development across single-host and distributed versions
is not just possible, but encouraged. The distribution oblivious
training function can have several benefits for ML systems. It can
(1) enable reductions in technical debt in pipeline orchestration,
(2) enable iterative development between laptops and clusters, and
(3) improve model training lifecycle management by factoring out
explicit logging calls from user code.
ACKNOWLEDGEMENT
This work is supported by the ExtremeEarth project funded by European Union’s Horizon 2020 Research and Innovation Programme under Grant Agreement No. 825258.

REFERENCES

1Project website: http://earthanalytics.eu.