Continuous Training for Production ML in the TensorFlow Extended (TFX) Platform

Denis Baylor, Kevin Haas, Konstantinos Katsiapis, Sammy Leong, Rose Liu, Clemens Menwald, Hui Miao, Neoklis Polyzotis, Mitchell Trott, and Martin Zinkevich, *Google Research*

https://www.usenix.org/conference/opml19/presentation/baylor

This paper is included in the Proceedings of the
2019 USENIX Conference on
Operational Machine Learning (OpML '19).

May 20, 2019 • Santa Clara, CA, USA

ISBN 978-1-939133-00-7

Open access to the Proceedings of the
2019 USENIX Conference on
Operational Machine Learning
is sponsored by USENIX.

Continuous Training for Production ML in the TensorFlow Extended (TFX) Platform

Denis Baylor
Google Research

Kevin Haas
Google Research

Konstantinos (Gus) Katsiapis
Google Research

Sammy Leong
Google Research

Rose Liu
Google Research

Clemens Menwald
Google Research

Hui Miao
Google Research

Neoklis Polyzotis
Google Research

Mitchell Trott
Google Research

Martin Zinkevich
Google Research

Abstract

Large organizations rely increasingly on continuous ML pipelines in order to keep machine-learned models continuously up-to-date with respect to data. In this scenario, disruptions in the pipeline can increase model staleness and thus degrade the quality of downstream services supported by these models. In this paper we describe the operation of continuous pipelines in the Tensorflow Extended (TFX) platform that we developed and deployed at Google. We present the main mechanisms in TFX to support this type of pipelines in production and the lessons learned from the deployment of the platform internally at Google.

1 Introduction

The workflows and underlying systems for machine learning (ML) in production systems come in different shapes and sizes. One key distinction is that between one-off and continuous pipelines. One-off pipelines are initiated by engineers to produce ML models "on demand". In contrast, continuous pipelines are "always on": they ingest new data and produce newly updated models continuously. The expectation is that a "fresh" model should be pushed to serving as frequently and timely as possible in order to reflect the latest trends in the incoming traffic.

Generally speaking, any ML task whose underlying data domain is non-stationary can benefit from continuous training to keep models fresh. Failing to update models in non-stationary settings can lead to performance degradation. The frequency with which models need to be updated depends on the speed with which the underlying data evolves. We describe two characteristic examples:

- **Recommender Systems**: In recommendation systems the inventory of items that represent the corpus keeps expanding. As an example, in YouTube new videos are added every second of the day. The models that retrieve those items and rank them for users have to be updated as the corpus expands to make sure that the recommendations are fresh.
- **Perception Problems**: In many perception problems, label acquisition can be slow and costly, while the models themselves still have not converged. In these cases, it is beneficial to continuously update the model with new labeled training data, as long as the performance keeps improving with newly arriving labels.

The most extreme case of refreshing models is *online learning* [3] which updates a model with every received request, i.e. the serving model is the training model. However, in practice it is more common to update a model in batches to ensure production safety by validating the data and models before they are updated. At Google, many ML pipelines update models on an hourly or daily basis. This is often enough for the most common use-cases we will discuss below.

A key metric for continuous pipelines is model freshness, as a delay in generating a new model can negatively affect downstream services. Given that the arrival of new data is highly irregular, this necessitates a "reactive" architecture where the pipeline can detect the presence of new inputs and trigger the generation of a new model accordingly. This also implies that continuous pipelines cannot be implemented effectively as the repeated execution of one-off pipelines at scheduled intervals, e.g., every 24h: if new data appears slightly after the scheduled execution of the pipeline, it can take more than one interval to produce a fresh model which may be unacceptable in a production setting.

In this paper we describe how we implemented support for continuous pipelines in the TensorFlow Extended (TFX) platform [1]. TFX enables Google's engineers to reliably run ML in production and is used across hundreds of teams internally. The design of TFX is influenced by Google's use cases and our experience with its deployment. However, we believe that the abstractions and lessons learned are relevant for large-scale deployments of continuous ML pipelines in other environments and organizations.

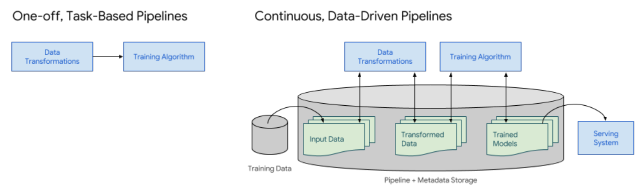

Figure 1: Continuous, data-driven pipelines need to be aware of artifacts, their properties, and lineage.

2 Continuous Pipelines in TFX

2.1 Maintaining State

Continuous pipelines need to maintain state in order to detect when new inputs appear and infer how they affect the generation of updated models. Moreover, this state can help the pipeline determine what results can be reused from previous runs. For instance, a pipeline that updates a deep learning model every hour needs to reinitialize (some of) the model's weights (also called warm-starting) from a previous run to avoid having to retrain over all data that has been accumulated up to this point. Similarly, model validation needs to retrieve the current production model in order to compare it against a new candidate model.

To manage this state, TFX introduces an ontology of artifacts which model the inputs and outputs of each pipeline component, e.g., data, statistics, models, analyses. Artifacts also have properties, e.g., a data artifact is characterized by its position in the timeline and the data split that it represents (e.g., training, testing, eval). Moreover, TFX maintains the lineage between artifacts.

2.2 Orchestration

Metadata about artifacts reflects the state of the pipeline and is recorded in a persistent store. The metadata store supports transactional updates, so that pipeline components can publish their output artifacts in a consistent fashion. Moreover, the store serves as the communication channel between components, e.g., the trainer can "listen" for the appearance of data artifacts and react accordingly. This pub/sub functionality, illustrated in Figure 1, forms the cornerstone of component execution and orchestration in TFX and enables several advanced properties. First, components can operate asynchronously at different iteration intervals, allowing fresh models to be produced as soon as possible. For instance, the trainer can generate a new model using the latest data and an old vocabulary, without having to wait for an updated vocabulary. The new model may still be better than the current model in production. Second, components can reuse results from previous runs if their inputs and configuration have not changed. Overall, this data-driven execution is essential for continuous pipelines and mostly absent from one-off pipelines.

2.3 Automated Validation

Any system that automatically generates new ML models must have validation safeguards in place before pushing a new model to production. Using human operators for these validation checks is prohibitively expensive and can slow down iteration cycles. Moreover, these safeguards need to apply at several points in the pipeline in order to catch different classes of errors before they propagate through the system. This implies more than just checking the quality of the updated model compared to the current production model. As an example, suppose that an error in the data leads to a suboptimal model. Whereas a model-validation check will prevent that model from being pushed to production, the trainer's checkpointed state might be affected by the corrupted data and thus propagate errors to any subsequent warm-started models.

TFX addresses these points by employing several validation checks at different stages of the pipeline. These checks ensure that models are trained on high-quality data (data validation [2][1]), are at least as good as or better than the current production model (model validation[2]), and are compatible with the deployment environment (serving infrastructure validation[3]).

3 Realizing One-Off, Task-Based Pipelines

TFX also supports one-off or task-based pipelines. The target audience is engineers who do not need the full power of continuous pipelines, or engineers who have set up a continuous pipeline but need to manually trigger execution of some components, e.g. experimenting with different model architectures while the input data remain unchanged.

Realizing one-off pipelines with a system that has been designed for continuous pipelines is technically straight forward, as a one-off run is just one iteration of a continuous pipeline without prior state. However, the mental model of task-based execution does not map to that of data-driven orchestration. Developers who are used to seeing jobs execute in sequence, as they were defined in a directed acyclic graph (DAG), are not accustomed to runs being triggered by the presence of a specific configuration of artifacts, as represented by the pipeline state.

As a solution, TFX introduces a framework that allows users to specify job dependency as they would in a task-based orchestration system. This also allows users of the open source version of TFX to orchestrate their TFX pipelines with task-based orchestration systems like Apache Airflow[4].

[1]Using TensorFlow Data Validation.

[2]Using TensorFlow Model Analysis for model validation.

[3]Using TensorFlow Serving.

[4]Details about the API that allows both modes of executions can only be added to this paper after March

References

[1] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, Chiu Yuen Koo, Lukasz Lew, Clemens Mewald, Akshay Naresh Modi, Neoklis Polyzotis, Sukriti Ramesh, Sudip Roy, Steven Euijong Whang, Martin Wicke, Jarek Wilkiewicz, Xin Zhang, and Martin Zinkevich. Tfx: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 1387–1395, New York, NY, USA, 2017. ACM.

[2] E. Breck, N. Polyzotis, S. Roy, S. Whang, and M. Zinkevich. Data validation for ML. In *To appear in Proceedings of SysML'19*.

[3] N. Cesa-Bianchi P. Auer and C. Gentile. Adaptive and self-confident on-line learning algorithms. In *Journal of Computer and System Sciences*, pages 48–75, 2002.