TENSORFLOW 2.0 AND KUBEFLOW FOR SCALABLE AND REPRODUCABLE ENTERPRISE AI

Romeo Kienzler[1, 2], Holger Kyas[2, 3, 4]

[1]IBM Center for Open Source Data and AI Technologies,
505 Howard St, San Francisco, CA, USA

[2]Berne University of Applied Sciences, Technology and Informatics,
Wankdorffeldstrasse 102, 3014 Berne, Switzerland

[3]Open Group, 548 Market St #54820, San Francisco, CA 94104-5401

[4]Helvetia Insurance Switzerland, St. Alban-Anlage 26, 4002 Basel,
Switzerland

ABSTRACT

Towards the End of 2015 Google released TensorFlow 1.0, which started out as just another numerical library, but has grown to become a de-facto standard in AI technologies. TensorFlow received a lot of hype as part of its initial release, in no small part because it was released by Google. Despite the hype, there have been complaints on usability as well. Especially, for example, the fact that debugging was only possible after construction of a static execution graph. In addition to that, neural networks needed to be expressed as a set of linear algebra operations which was considered as too low level by many practitioners. PyTorch and Keras addressed many of the flaws in TensorFlow and gained a lot of ground. TensorFlow 2.0 successfully addresses these complaints and promises to become the go-to framework for many AI problems. This paper introduces the most prominent changes in TensorFlow 2.0 targeted towards ease of use followed by introducing TensorFlow Extended Pipelines and KubeFlow in order to illustrate the latest TensorFlow and Kubernetes ecosystem movements towards simplification for large scale Enterprise AI adoption.

KEYWORDS

Artificial Intelligence, TensorFlow, Keras, Kubernetes, KubeFlow, TFX, TFX Pipelines

1. INTRODUCTION

In this paper we introduce the novelties of TensorFlow 2.0 and how those impact the AI ecosystem from a developer's as well as from an enterprise architecture point of view. Recommendations are given on how constant changes and disruptions in the AI ecosystem can be addressed. When Google released TensorFlow under the Apache License 2.0 in November 2015, Torch, Theano and Caffe have been the most widely used DeepLearning frameworks [1][2]. Although those frameworks supported the most important features like OpenMP, CUDA, Automatic Differentiation, Convolutional and Recurrent Layers, the fact that it came from Google led to a huge adoption of TensorFlow (Figure 1) and in contrast, to a fast decline in usage of other frameworks, or even to an abandonation of frameworks like Theano and

Natarajan Meghanathan et al. (Eds) : CoSIT, AIAPP, SIGL, CRIS, DMA, NLPML, CYBI - 2020
pp. 19-32, 2020. © CS & IT-CSCP 2020 DOI: 10.5121/csit.2020.100103

Torch [3][4]. Besides the hype, developers eventually started complaining about it as well [5]. First, and especially, the lack of an eager execution mode pulled many users away from TensorFlow towards PyTorch [6]. PyTorch creates a dynamic execution graph allowing for line by line execution and debugging. In contrast, TensorFlow, up to version 1.6, created a static execution graph which had be run in a TensorFlow session in order to execute. Decupling code from runtime made it hard to debug TensorFlow code. Second, many developers found TensorFlow too low level. Working at the level of linear algebra is well suited for DeepLearning Research, but for applied settings, productivity vastly increases when developers move from matrices to layers. Therefore, Section 2 demonstrates how the new features in TensorFlow 2.0 addresses those usability related complaints, whereas Section 3 illustrates how TFX Pipelines expand TensorFlow functionality addressing the complete ML/DeepLearning workflow. Section 4 highlights the fact how KubeFlow closes the devops / CICD loop by bringing TFX Pipelines to Kubernetes. Section 5 reflects on specific requirements for Enterprise AI. Finally, a conclusion is drawn on how this technology stack impacts the AI ecosystem now and in the future.

Figure 1. Google Trends of different DeepLearning frameworks over time

2. THE MOST IMPORTANT FEATURES OF TENSORFLOW 2.0

In the following the most important features of TensorFlow 2.0 are explained, and contrasted to TensorFlow 1.0, where applicable, using practical examples.

2.1. Eager Execution

Eager execution is undoubtedly the most prominent new feature – introduced in TensorFlow 1.7 – activated by default in TensorFlow 2.0. Lack of eager execution was one of the main complaints against TensorFlow 1.X, as many developers can relate. Having to execute the whole computation graph to debug based on stack traces is tedious. Especially, since values of intermediate results haven't been accessible without mixing debug code into production code. With Eager Execution, the Tensor interface remains stable, but internally, "EagerTensor" objects are used over "Tensor" objects. Therefore, TensorFlow code can be used and debugged like arbitrary python code (as using numpy for example). In the following, source code examples are given for clearification:

a = tf.constant(np.array([1., 2., 3.]))

This creates an object of type “tensorflow.python.framework.ops.Tensor” which is meant to be a node of a static execution graph running in a TensorFlow session. Outside the context of a session, this object is useless as the following code illustrates:

a.eval()

Calling the eval method on a Tensor object outside a TensorFlow session context returns the following error:

No default session is registered.

This means, debugging by looking inside a Tensor object outside an execution graph running in a TensorFlow session is impossible. Now this example is extended to a vector dot product computation, by creating a second Tensor:

b = tf.constant(np.array([4.,5.,6.]))

A static execution graph is created by applying the dot product operation on *a* and *b*:

c = tf.tensordot(a, b,1)

Variable *c* now contains a reference to a static execution graph of type "tensorflow.python.framework.ops.Tensor" which can be passed to a TensorFlow session for execution:

tf.Session().run(c)

Considering the same example using Eager Execution, the same Tensors will be of type "tensorflow.python.framework.ops.EagerTensor". The following code returns the content of Tensor *a* as numpy ndarray:

a.numpy()
Interestingly, the same code now executes without a TensorFlow session:
```
a = tf.constant(np.array([1., 2., 3.]))
b = tf.constant(np.array([4.,5.,6.]))
c = tf.tensordot(a, b,1)
print(c.numpy())
```

2.2. tf.function

Creating linear algebra code using the TensorFlow API can be confusing. Therefore, TensorFlow 2.0 introduced an amazing feature which allows for transforming arbitrary python code into a TensorFlow execution graph by annotating a function, called *autograph*. In the following, this is illustrated by a small example minimizing a nested function. Considering a function f(x)=x-(6/7)*x-1/7 and a function g(x)= f(f(f(f(x)))). An x such that g(x)=0 needs to be found. This is known as the "Turnip Seller Problem" [7]. The solution can be obtained by setting the first derivative g(x)'=0. TensorFlow 1.X and 2.0 support automatic differentiation of execution graphs, obtaining g'(x) from g(x). But in TensorFlow 2.0 this graph can now be generated using pure python code. The following code expresses f(x) and g(x) in python and creates a TensorFlow execution graph by annotating the python functions *f* and *g* with "tf.function":

```
@tf.function
def f(x):
   return x-(6/7)*x-1/7

@tf.function
def g(x):
   return f(f(f(f(x))))
```

As can be observed, besides the annotations, the code is plain python only. Interestingly, this code has not defined any python function at all. It defined two TensorFlow objects *f* and *g* of type "tensorflow.python.eager.def_function.Function". In order to inspect those objects the following code can be used:

tf.autograph.to_code(f.python_function)
This prints the code contents of *f* which has been created internally:

```
def tf__f(x):
  do_return = False
  retval_ = ag__.UndefinedReturnValue()
  do_return = True
  retval_ = x - 6 / 7 * x - 1 / 7
  cond = ag__.is_undefined_return(retval_)

  def get_state():
    return ()

  def set_state(_):
    pass

  def if_true():
    retval_ = None
    return retval_

  def if_false():
    return retval_
  retval_ = ag__.if_stmt(cond, if_true, if_false, get_state, set_state)
  return retval_
```

In order to solve for g'(x)=0, x has to be defined as TensorFlow variable and y as constant to be used later to set g'(x) to zero:

```
x = tf.Variable(0, trainable=True, dtype=tf.float64)
y = tf.constant([0], dtype=tf.float64)
```

The last step is to run a gradient descent loop to find the minimum of g'(x):

```
variables = [x]
optimizer = tf.optimizers.Adam(0.5)
loss_object = tf.keras.losses.MeanAbsoluteError()
with tf.GradientTape(persistent=True) as tape:
   for i in range(1000):
      y_pred = g(x)
      loss = loss_object(y,y_pred)
      grads = tape.gradient(loss, variables)
      optimizer.apply_gradients(zip(grads, variables))
```

As *x* is defined as (the only) TensorFlow variable, *x* is constantly adjusted in order to minimize the optimization objective (setting g'(x) to zero) and after some iterations *x* converges to 400, which is the solution to this equation as explained in [7]. This example highlights the simplicity of creating complex TensorFlow Execution graphs with simple python functions.

2.3. Keras as official high level API in TensorFlow

Keras, released in March 2015, just seven months before TensorFlow, is a high level deep learning framework supporting various low level linear algebra frameworks, including TensorFlow. High level frameworks abstract away the complex linear algebra deep learning involves by introducing components like layers over operations on tensors. In 2017 Google

added Keras to the TensorFlow distribution and in 2019 declared it as official high level API in TensorFlow [8]. At a first glance, this doesn't seem to make too much sense since TensorFlow could be used as execution engine for Keras anyway. And in addition, this means that two APIs have to be kept in sync. The Keras API of the official Keras framework and the Keras API within TensorFlow. The next section explains the benefits of offering a Keras compatible API within TensorFlow, but first the following example is used to illustrate – given the correct way of imports – the 1:1 compatibility of Keras code between standalone Keras and the Keras API within TensorFlow. Considering the following standalone Keras code:

```
from keras import Sequential
from keras.layers import Flatten, Dense
from keras.activations import relu, softmax
model = Sequential([
   Flatten(input_shape=(28, 28)),
   Dense(128, activation=relu),
   Dense(10, activation=softmax)
])
```

Changing the imports makes the same code compile without any changes using the Keras API TensorFlow provides:

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Flatten, Dense
from tensorflow.nn import relu
from tensorflow.nn import softmax
```

2.4. Distribution Strategies

As stated above, to the reader it might look tedious to maintain two implementations of the Keras API but when used in TensorFlow, distribution becomes very simple. Although both, native Keras and the Keras API of TensorFlow share the same interface which makes code look identical, the underlying implementation differs. Therefore when Keras is used in conjunction with TensorFlow, it's implementation supports distribution out of the box. The only necessity is running the Keras code within a context of a TensorFlow distribution strategy. This sounds complex, therefore the following code is given as an example for clearification:

```
ps_strategy = tf.distribute.experimental.ParameterServerStrategy()
with ps_strategy.scope():

   model = Sequential([
      Flatten(input_shape=(28, 28)),
      Dense(128, activation=Relu),
      Dense(10, activation=Softmax)
   ])

   model.compile(optimizer='adam',
         loss='sparse_categorical_crossentropy',
         metrics=['accuracy'])

   model.fit(train_images, train_labels, epochs=5)
```

As can be seen, the only necessity is instantiating a distribution strategy object and running within the scope of it. Interestingly, distribution is not limited to training but also works for

distributed evaluation and prediction. It's noteworthy that there exist different distribution strategies addressing distinct cluster configurations ranging from single node, multiple GPU environments to multi node, multiple GPU per node environments. Depending on the strategy in use, minor configuration needs to be made on the cluster nodes. Although it is beneficial to understand the implementation details of different strategies, TensorFlow's documentation clearly aligns them to distinct cluster configurations [9]. Distribution strategies are not only supported by Keras but by any computation within TensorFlow through a dedicated API and also by TensorFlow Estimators, which as of TensorFlow 2.0 are all distribution strategy-aware. TensorFlow Estimators are another high level API within TensorFlow, which is covered in the next subsection.

2.5. TensorFlow Estimators

TensorFlow Estimators are probably the most important driver towards consumable AI. Estimators serve as ready-made building blocks – greatly reducing complexity and skill requirements from developer side, by encapsulating everything necessary for model training, scoring and serving into a single object. As stated previously, Estimators can run on any kind of distributed cluster environment, including GPU and TPU clusters, within a TensorFlow distribution strategy scope, without changing the Estimator's implementation. TensorFlow ships a growing list of pre-compiled estimators, ranging from Single Layer Neural Networks over Gradient Boosted Trees to simple linear models. Therefore, TensorFlow is entering the traditional machine learning space as well which is dominated by Scikit-Learn in the python community [10]. In order to understand estimators, in the following, an example is introduced:

```
classifier = tf.estimator.LinearClassifier(feature_columns=[age])
classifier.train(train_inpf)
result = classifier.evaluate(test_inpf)
```

As can be seen, running a logistic regression classifier is as simple as executing three lines of code. Note that the *age* variable, implementing the *feature_column* interface, contains the schema / meta information of a single column/feature, whereas *train_inpf* and *test_inpf* are functions converting underlying data to the required format (e.g. dictionary of Tensors). This concept stems from the idea that the same data preprocessing pipeline has to be reusable for training, testing, scoring and serving [11]. Reusability and operationalization is a very active area of research and development [12]. Those features are extensively supported through TensorFlow Estimators.

3. TENSORFLOW EXTENDED (TFX) PIPELINES

As touched at the end of the previous section, developer focus shifts away from simple model development and focus on operationalization, Explainability [13], Fairness [14], Robustness [15] and Data Lineage [16] drastically increased over the last year. Therefore, a top level section is dedicated to TFX Pipelines. One has to distinguish between the TFX API [17], which provides low level implementations of different transformation and aggregation steps, and TFX Pipelines which facilitate development of reusable and explainable AI workflows. It's modules are explained in the following subsections.

3.1. TFX Pipelines ExampleGen

Although the name might be misleading, responsible for data loading, ExampleGen is the first stage of every TFX Pipeline. Implementing complex data loading code using the tf.data API can be avoided therefore. As in most machine learning frameworks like scikit-learn, SparkML and R, reading a CSV file can be done in one line of code. Using the tf.data API this needs a

whole code block. Using ExampleGen, the task now also boils down to one line of code like this:

example_gen = CsvExampleGen(csv_input('floorsensordata2604.csv'))

This creates an instance of ExampleGen based on the file *floorsensordata2604.csv*. Instances of ExampleGen are used by downstream stages of TFX Pipelines. ExampleGen also takes care of data partitioning for distributed training as well as implementing the ML best practices in data shuffling [18].

3.2. TFX Data Validation

Data validation is one of the first steps in every machine learning project [19] and a variety of tools exist [20]. TFX Data Validation, which is a low level API called by TFX Pipeline components, casts these steps in a nicely documented and feature rich tool suite with three main focus areas: Schema Based Example Validation, Training-Serving Skew Detection, Drift Detection. In Schema Based Example Validation, TensorFlow Extended Data Validation computes statistics on a given data set to be compared against a defined schema. Alternatively, schemas can be inferred from a given data set as well. A distinct feature of TFX are schema environments, which allow for different schemas under different circumstances, e.g. removal of a label column in a model serving environment. Training-Serving Skew Detection detects differences between training and serving data including changes in schema, data distributions and categorical imbalances. Drift Detection raises alerts if new, unseen data appears to be abnormal to the previously seen examples and a predefined threshold is exceeded. Currently, only L-infinity distance is supported [21]. Finally, TensorFlow Extended Data Validation also supports interactive tooling based on the facets [22] project which can be embedded in jupyter notebooks. Since DataValidation is a low level API, it's usage is exemplified through StatisticsGen and ExampleValidator, which are introduced in the next subsections.

3.3. TFX Pipelines StatisticsGen

StatisticsGen [23] creates feature statistics for the downstream TFX Pipeline. This is a very important object containing meta information about the data. Components like SchemaGenerator and ExampleValidator, which will be introduced below, are making use of those statistics. Again, computing those statistics resembles into one line of code:

```
stats = components.StatisticsGen(
    input_data=example_gen.outputs.examples,
    name='compute-eval-stats'
    )
```

Now, the *stats* object can be passed as parameters to downstream pipeline components as we can see in the next section.

3.4. TFX Pipelines SchemaGen

SchemaGen [24] is a component which infers a schema from a given dataset. A schema includes data types for each feature as well as expected value ranges and nullability for downstream validity checks. It makes use of the *stats* object explained in the previous section. Again, only a single line of code is required in order to execute this stage:

infered_schema = components.SchemaGen(stats=stats.outputs.output)

Using the *infered_schema* object in a pipeline reliefs the developer from maintaining a schema manually. Changes to schemas, e.g. additions or deletions of unused features can be handled without code changes therefore.

3.5.TFX Pipelines ExampleValidator

ExampleValidator [25] uses the data validator API to detect data quality issues by creating relevant validation statistics using a schema (implicitly created by SchemaGen or explicitly defined) and StatisticsGen objects as input. A single code line creates this validation statistics object:

```
validate_stats = components.ExampleValidator(
    stats=stats.outputs.output,
    schema=infered_schema.outputs.output
    )
```

This object contains statistics supported by the TFX Data Validator API and can be inspected interactively or thresholds can be defined to control (e.g. abort) downstream execution of a TFX Pipeline.

3.6. TFX Pipelines Transform

Transform [26] is core for any feature transformation and engineering task. It uses the TFX Transform low level API and facilitates transformation tasks by providing a rich set of predefined transformation functions [27]. Although using existing transformations is simple, it is important to note that implementing a custom transformation is a non-trivial task which requires deep understanding on Tensor operations. On the other hand, any type of arbitrary data transformation can be accomplished, making this stage a key feature of TFX Pipelines, ensuring that unsupported transformations can be implemented manually.

3.7. TFX Pipelines Trainer

Trainer [28] encapsulates either a TensorFlow Estimator or a TensorFlow Keras API Model in order to make them available as a component within a TFX Pipeline. A Trainer consumes examples and a schema. It creates a SavedModel or EvalSavedModel object. SavedModel is the standard model export format of TensorFlow. EvalSavedModel is explained in the next subsection.

3.8. TFX Pipelines Evaluator

Evaluator [29] creates performance statistics on a model created by the Trainer component. Therefore it must be saved as EvalSavedModel over SavedModel since EvalSavedModel contains additional information needed by the Evaluator in order to compute statistics. Those statistics are published to TensorFlow Metadata [30], a central TFX metadata store, for downstream analysis.

3.9. TFX Pipelines ModelValidator

ModelValidator [31] uses a schema and statistics created upstream in order to mark a model as deployable (blessing). This component enables a TFX Pipeline to be automatically deployed after a model has been retrained on new data because model performance metrics are automatically checked before deployment.

3.10. TFX Pipelines Pusher

Pusher [32] is a simple gatekeeper component which checks the results of a ModelValidator component and if the "blessing" was received, it saves the model to a specified target including TensorFlow Serving, TensorFlow.js or TensorFlow Lite.

4. KUBEFLOW PIPELINES

4.1. Kubernetes

Kubernetes [33], a container orchestrator initially released by Google, is transforming IT operations in cloud and enterprise data centers worldwide [34].

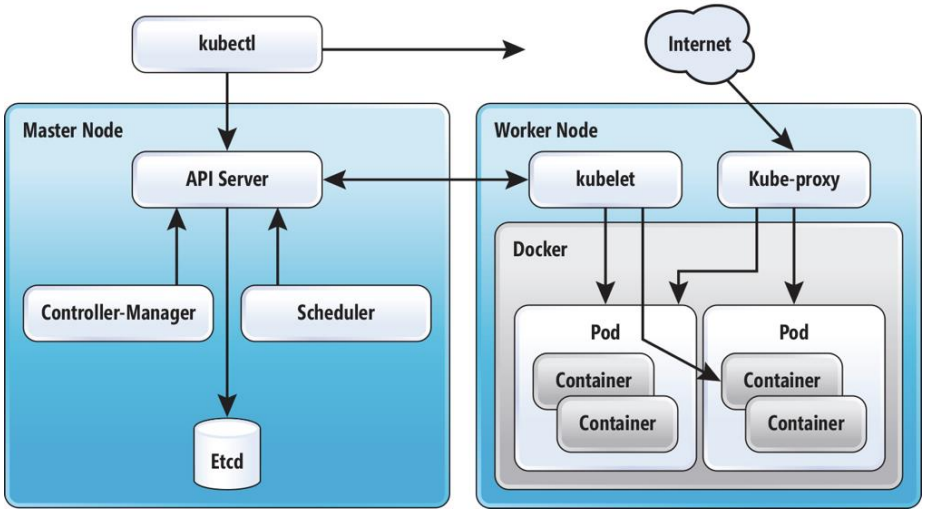


Figure 2. Kubernetes Architecture Overview.

Figure 2 illustrates the basic components of Kubernetes. The Master Node runs the Controller-Manager which is core for container orchestrations. E.g. it constantly verifies the is-requested difference and in case a container crashed, a replacement is scheduled to be created. On the Worker Node, a kubelet is responsible, among other tasks, to start and stop individual containers. In Kubernetes, a Pod groups a set of Containers into a namespace, sharing resources, e.g. virtual network interfaces. This makes Kubernetes a lightweight abstraction layer over a set of data center resources to be consumed by applications materialized as Pod.

4.2. KubeFlow

KubeFlow [35] is a workload agnostic pipeline execution framework. This means, every application which can be packaged as a container can be run within KubeFlow. For example, Apache Spark is supported natively. Individual steps of a data processing pipeline are divided into tasks which KubeFlow provisions individually on Kubernetes. This mitigates a common resource assignment problem for high variational resource demands between intermediate pipeline tasks.

4.3. KubeFlow as engine for TFX Pipelines

Although, TFX Pipelines are workflow engine agnostic, currently, implementations for Apache AirFlow [36] and KubeFlow exist. Since KubeFlow runs natively on Kubernetes, its particularly interesting as scalable TFX Pipelines execution engine, benefitting from

Kubernetes scalability and wide industry adoption. A TFX Pipeline can be imported into KubeFlow. After importing, a TFX Pipeline, as well as all other types of KubeFlow Pipelines, is available for execution (Figure 3). Before execution, a set of pipeline specific, predefined parameters can be set (e.g. path to data source). A running pipeline is called an experiment.

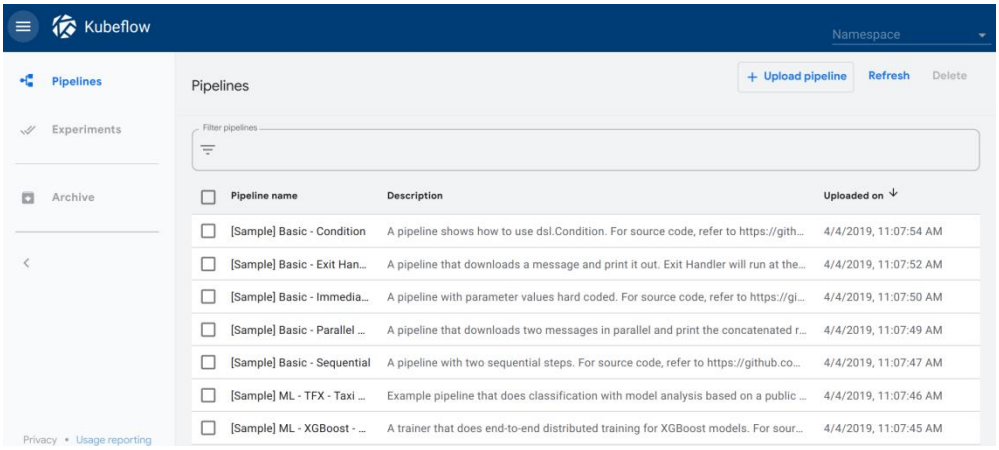


Figure 3. KubeFlow Pipelines. An overview of available (deployed) Pipelines.

Experiments are running on Kubernetes. Each stage of a pipeline runs in a separate Pod. Assets between stages are shared via file system. Figure 4 shows a screen shot of a successfully ran experiment and all interactions between individual stages as graph. The green arrow indicates successful completion. Resource limitations can be assigned to each individual stage allowing for scale-up and scale-out on Kubernetes.

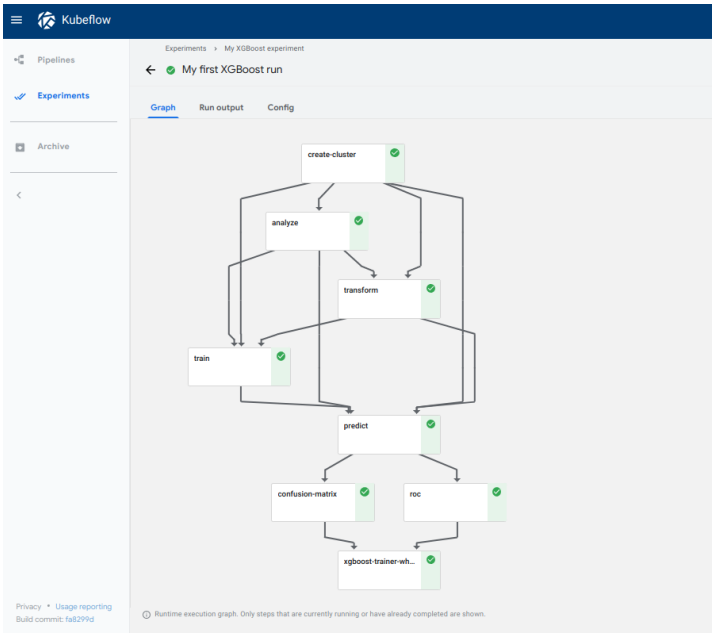
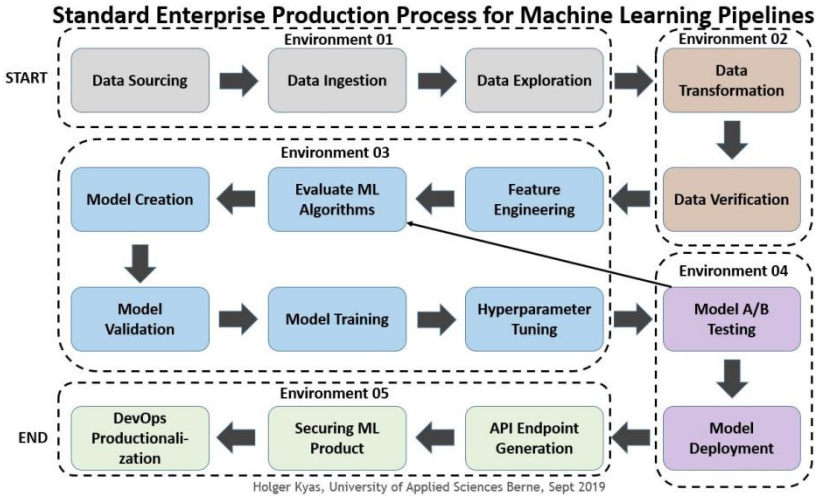


Figure 4. KubeFlow Pipelines. Single Experiment, all stages successfully ran.

5. INDUSTRY AND ENTERPRISE CONSIDERATIONS ON MACHINE LEARNING FRAMEWORKS TENSORFLOW, KERAS AND PYTORCH

The usage of Machine Learning Frameworks like Tensorflow 2.0, Keras and Pytorch is very promising for enterprises in many industries due to its evident and potential business value. Generating undiscovered, critical insights and predictive trends about customer behavior, internal fraud and/or market trends are key use cases leveraged by those Machine Learning capabilities leading to higher revenue while driving innovative shifts.

Dedicated Machine Learning Experts, Data Engineers and Scientists care for an end-to-end data quality within their pipelines, while also the algorithmic layer is a crucial element underpinned by the usability and separation of concerns within frameworks. The Process for Machine Learning Pipelines may vary between people and organizations. A Standard Enterprise Production template is suggested as follows:



Latest KubeFlow and TensorFlow Extended (TFX) distributions come with major key advantages:

Manage: Adaptability / Complexity / Consistency / Large Data Wrangling / Versioning
Enhance: Model Quality / Isolation / Portability / Training Quality / User Experience
Important architectural questions need to be addressed within enterprises of various industries due to technical and compliance factors. For instance, Data consistency, information age and network latency are factors and requirements driving architectural decisions like data location, integration and cloud deployment models.

Traditional enterprises usually protect their data on-premises while facing questions like moving data to a private, hybrid or public clouds. Instead of moving data to compute engines, the opposite is possible. Leaving data where they're hosted is often a better option if computing artefacts can be mobilized to the data host. This leads to a potential model where Machine Learning Frameworks like TensorFlow 2.0 require mobility.

Container technology provide mobile encapsulation of functionality. If the data host is able to host containers, it is a feasible alternative. Since this is not always the case it becomes a data and service integration architecture question. Cloud Computing does not make the integration architecture less complex, it raises new issues which go far beyond Machine Learning Frameworks. The stated complexity of instrumenting Machine Learning Frameworks like TensorFlow 2.0 are evident, while the industry solutions are usually individual to the related

enterprise context. It requires the ML pipelines to manage, operationalize and bring into production across different environments, for instance like the following illustration. TensorFlow 2.0 and KubeFlow support this.

The context of Machine Learning Frameworks is wider for enterprises than it seems from a higher use case and business value level. The materialization of Machine Learning promises imply the resolution of complex problems on Data Placement, Integration Architecture and Cloud Deployment.

To reach lean usage of Machine Learning Models exposed to business users by API's an agile Design and Development is more efficient thus recommended. Among others, Flexibility and Adaptability are guiding principles taking decisions on architectural questions in the context of Machine Learning and beyond. And regular Architecture Testing is an obligatory task, also since developments in technology and socio-economy are very fast and impactful.

6. CONCLUSIONS

As can be seen from the latest developments in the ecosystem, TensorFlow aggressively pushes into various directions to establish a de-facto standard in AI. The latest improvements of TensorFlow 2.0 are directed towards simplicity in model development and scaling. TFX Pipelines address DevOps and CICD requirements and compatibility to KubeFlow adds scalability into the mix. As KubeFlow is not only limited to run TFX Pipelines it can co-exist with other established technologies, including Apache Spark, the current standard for large scale ETL for example. This minimizes technology lock-in and provides a seamless technology migration path between different tools and frameworks as their popularity rises and falls. To reach lean usage of Machine Learning Models exposed to business users by API's an agile Design and Development is more efficient thus recommended. Flexibility and Adaptability are guiding principles taking decisions on architectural questions in the context of Machine Learning and beyond. Testing and adopting the architecture regularly is obligatory since developments in technology and socio-economy are very fast.

ACKNOWLEDGEMENTS

Special thanks to Animesh Singh and Tommy Li, both working for the IBM Center for Open Source Data and AI Technologies (CODAIT) in San Francisco, for their valuable inputs.

REFERENCES

[1] Andrej Karpathy. A Peek at Trends in Machine Learning. https://medium.com/@karpathy/a-peek-at-trends-in-machine-learning-ab8a1085a106 (Aug. 2019)"

[2] Andrej Karpathy. Unique mentions of deep learning frameworks in arxiv papers (full text) over time. https://twitter.com/karpathy/status/972295865187512320 (Aug. 2019)

[3] Pascal Lamblin. MILA and the future of Theano. https://groups.google.com/forum/#!topic/theano-users/7Poq8BZutbY (Aug 2019)

[4] Sam Stites. Development Status. https://github.com/torch/torch7/blob/master/README.md (Aug 2018)

[5] Daniel Kuster. The Good, Bad & Ugly of TensorFlow. https://www.kdnuggets.com/2016/05/good-bad-ugly-tensorflow.html (Aug 2019)

[6] Yaroslav Bulatov. TensorFlow meets PyTorch with Eager execution. https://medium.com/@yaroslavvb/tensorflow-meets-pytorch-with-eager-mode-714cce161e6c (Aug 2019)

[7] Ian Stewart. *Professor Stewart's Cabinet of Mathematical Curiosities*, Page 9, ISBN 978-0465013029

[8] TensorFlow Documentation. Keras. https://www.tensorflow.org/guide/keras (Aug 2019)

[9] TensorFlow Documentation. Distributed Training in TensorFlow. https://www.tensorflow.org/guide/distribute_strategy (Aug 2019)

[10] KDnuggets. Top 20 Python Machine Learning Open Source Projects. https://www.kdnuggets.com/2015/06/top-20-python-machine-learning-open-source-projects.html (Aug 2019)
[11] TensorFlow Documentation. Datasets for Estimators. https://www.tensorflow.org/guide/datasets_for_estimators (Aug 2019)
[12] Kumar Srivastava. Method and system for auto learning, artificial intelligence (ai) applications development, operationalization and execution. Patent US20190171950A1
[13] Ian Sample. *Computer says no: why making AIs fair, accountable and transparent is crucial.* the Guardian, 5 November 2017.
[14] Solon Barocas, Andrew D. Selbst. *Big Data's Disparate Impact.* California Law Review Jun 2016 Volume 104 No. 3
[15] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian M. Molloy, Ben Edwards. *Adversarial Robustness.* ArXiv 2018
[16] Yingwei Cui, Jennifer Widom. *Lineage tracing for general data warehouse transformations.* Volume 12 Issue 1, May 2003, Pages 41-58, The VLDB Journal — The International Journal on Very Large Data Bases
[17] Denis Baylor, Eric Breck, Heng-Tze, 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, Martin Zinkevich (2017) *TFX: A TensorFlow-Based Production-Scale Machine Learning Platform*, 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 2017
[18] TensorFlow Documentation. *The ExampleGen TFX Pipeline Component. https://www.tensorflow.org/tfx/guide/examplegen (Aug 2019)*
[19] Romeo Kienzler. *The lightweight IBM Cloud Garage Method for data science.* https://developer.ibm.com/articles/the-lightweight-ibm-cloud-garage-method-for-data-science/ (Aug 2019)
[20] Romeo Kienzler. *Architectural decisions guidelines.* https://developer.ibm.com/articles/data-science-architectural-decisions-guidelines/ (Aug 2019)
[21] Wikipedia. *Chebyshev distance.* https://en.wikipedia.org/wiki/Chebyshev_distance (Aug 2019)
[22] *Facets.* https://pair-code.github.io/facets/ (Aug 2019)
[23] TensorFlow Documentation. *The StatisticsGen TFX Pipeline Component.* https://www.tensorflow.org/tfx/guide/statsgen (Aug 2019)
[24] TensorFlow Documentation. *SchemaGen.* https://www.tensorflow.org/tfx/guide/schemagen (Aug 2019)
[25] TensorFlow Documentation. *The ExampleValidator TFX Pipeline Component.* https://www.tensorflow.org/tfx/guide/exampleval (Aug 2019)
[26] TensorFlow Documentation. *The Transform TFX Pipeline Component.* https://www.tensorflow.org/tfx/guide/transform (Aug 2019)
[27] TensorFlow Documentation. *Module: tft.* https://www.tensorflow.org/tfx/transform/api_docs/python/tft (Aug 2019)
[28] TensorFlow Documentation. *The Trainer TFX Pipeline Component.* https://www.tensorflow.org/tfx/guide/trainer (Aug 2019)
[29] TensorFlow Documentation. *The Evaluator TFX Pipeline Component.* https://www.tensorflow.org/tfx/guide/evaluator (Aug 2019)
[30] TensorFlow Documentation. *ML Metadata.* https://www.tensorflow.org/tfx/guide/mlmd (Aug 2019)
[31] TensorFlow Documentation. *The ModelValidator TFX Pipeline Component.* https://www.tensorflow.org/tfx/guide/modelval (Aug 2019)
[32] TensorFlow Documentation. *The Pusher TFX Pipeline Component.* https://www.tensorflow.org/tfx/guide/pusher (Aug 2019)
[33] Kubernetes. https://kubernetes.io/ (Aug 2019)
[34] Kelsey Hightower, Brendan Burns, Joe Beda. *Kubernetes: Up and Running: Dive Into the Future of Infrastructure. O'Reilly 2018 ISBN 978-1-491-935-67-5*
[35] KubeFlow. https://www.kubeflow.org/ (Aug 2019)
[36] Apache Airflow Documentation. https://airflow.apache.org/ (Aug 2019)

Authors

Romeo Kienzler is Chief Data Scientist at IBM Center for Open Source Data and AI Technologies (CODAIT) in San Francisco, USA, Associate Professor for Artificial Intelligence at Berne University of Applied Sciences and Adjunct Professor for Information Security at Northwestern University of Applied Sciences, Switzerland. He holds a M. Sc. (ETH) in Information Systems, Applied Statistics and Bioinformatics

Holger Kyas is Board Member at Open Group, Adjunct Professor for Cloud, BigData, Machine Learning and Information Technology at Berne University of Applied Sciences, Switzerland and Enterprise Architect at Helvetia Insurances Switzerland. He holds a Masters in Information Technology and is studying Masters of Business Administration at University of Applied Sciences and Arts in Basel,