ML Inference at scale, easy as learning your 5 X table!

Robert Crowe Developer Engineer TFX
Reza Rokni Developer Advocate Google Dataflow
Google
ML Inference at scale, easy as learning your 5 times table, with TFX and Apache Beam!
Production Machine Learning

Agenda

1. Overview of RunInference and its place within TFX ML-OPS
2. Doing our 5 times table, with Apache Beam and RunInference
3. Pre and Post-processing
4. Branching and sequential inference pipelines
5. GPUs with the Dataflow Runner
TFX / RunInference - Quick Overview
Machine Learning & Pipelines

ML development falls naturally into a series of tasks:

1. Data Ingestion
2. Data Validation
3. Feature Engineering
4. Train Model
5. Validate Model
6. Push If Good
7. Serve Model

Libraries:
- DATA INGESTION
- TENSORFLOW DATA VALIDATION
- TENSORFLOW TRANSFORM
- ESTIMATOR OR KERAS MODEL
- TENSORFLOW MODEL ANALYSIS
- VALIDATION OUTCOMES
- TENSORFLOW SERVING

Components:
- ExampleGen
- StatisticsGen
- SchemaGen
- Transform
- Trainer
- Evaluator
- Pusher
- Model Server
- Example Validator
- Tuner
- Infra Validator
- Bulk Inference
What do we mean by portability?

TFX runs just about anywhere

Execution

Orchestration

Processing
Why is portability important?

Meet the user’s needs, instead of requiring them to meet yours

- Execution environments
- Orchestrators
- Distributed processing frameworks
- Languages (to some extent)
- Teams / Business units
Hello TFX

TFX CONFIG

AIRFLOW RUNTIME  KUBEFLOW RUNTIME  OTHER

ExampleGen  StatisticsGen  SchemaGen

Example Validator  Transform  Trainer  Evaluator

Infra Validator  Pusher

ExampleGen  Bulk Inference

TensorFlow Extended

TRAINING & EVAL DATA

BATCH INFERENCE DATA (UNLABELED)

METADATA STORE

TFX INFRA

Validator

Extended
Component: BulkInferrer

**Inputs and Outputs**

- Trainer
- Evaluator
- ExampleGen

- Model
- Validation Outcome
- Unlabelled examples

**BulkInferrer**

- **bulk_inferrer** = BulkInferrer(
  - examples=inference_example_gen.outputs['examples'],
  - model_export=trainer.outputs['output'],
  - model_blessing=evaluator.outputs['blessing'],
  - data_spec=bulk_inferrer_pb2.DataSpec(example_splits=['unlabelled']),
  - model_spec=bulk_inferrer_pb2.ModelSpec())

**Configuration Options**

- Block batch inference on a successful model validation.
- Choose the inference examples from example gen's output.
- Choose the signatures and tags of inference model.

**Inference Result**

- Contains features and predictions.
BulkInferrer & RunInference

TFX components run data processing in Beam pipelines

- BulkInferrer runs inference on Beam using RunInference
- RunInference included in tensorflow/tfx-bsl
- RunInference can also be used in pure Beam
  - Beam pipelines with no TFX
  - Beam pipelines inside TFX custom components
A simple model

Y vs. X
What can you do with a model?

Imagine trying to do these without a model!

- Speech to text
- Text to customer sentiment
- Recognize objects in images
- Look for manufacturing defects in images
- Predict failures in equipment
- Select the products that a customer is most likely to want
- Select the support doc that fixes a customer issue
Build a simple linear regression model.
Note the model has a shape of (1) for its input layer, it will expect a single int64 value.

```python
input_layer = keras.layers.Input(shape=(1), dtype=tf.float32, name='x')
output_layer = keras.layers.Dense(1)(input_layer)
model = keras.Model(input_layer, output_layer)
model.compile(optimizer=tf.optimizers.Adam(), loss='mean_absolute_error')
model.summary()
```
```
tfx_bsl.public.beam.RunInference

Features

- Batches inputs when possible
- Outputs a predictlog proto, includes input
- Key forwarding
- Remote or Local mode
- Local mode, loads the model once and shares across threads
```
RunInference - Local model mode

Dataflow
Cloud Storage
Source
PCollection (bounded or unbounded)
Sink

RunInference
<pull saved model file>
Pre-processing choices

Dataflow

Pre-processing

Model

Source

PCollection (bounded or unbounded)

Sink
Build a simple linear regression model. Note the model has a shape of (1) for its input layer, it will expect a single int64 value.

```python
input_layer = keras.layers.Input(shape=(1), dtype=tf.float32, name='x')
output_layer = keras.layers.Dense(1)(input_layer)
model = keras.Model(input_layer, output_layer)
model.compile(optimizer=tf.optimizers.Adam(), loss='mean_absolute_error')
model.summary()
```
The serving signature

TFRecord → ReadFromTFRecord Transform → Serialized tf.example → Apache Beam RunInference Transform → Model (tf.function())
@tf.function(input_signature=[tf.TensorSpec(shape=[None], dtype=tf.string, name='examples')])

def serve_tf_examples_fn(serialized_tf_examples):
    """Returns the output to be used in the serving signature."""
    features = tf.io.parse_example(serialized_tf_examples, RAW_DATA_PREDICT_SPEC)
    return model(features, training=False)

signature = {'serving_default': serve_tf_examples_fn}

tf.keras.models.save_model(model, save_model_dir_multiply, signatures=signature)
Using TFX inference with Dataflow for large scale ML inference patterns

In part I of this blog series we discussed best practices and patterns for efficiently deploying a machine learning model for inference with Google Cloud Dataflow. Amongst other techniques, it showed efficient batching of the inputs and the use of shared.py to make efficient use of a model.
Metadata - Attaching a key

Dataflow

Value

Apache Beam RunInference Transform Model f(x) function()

Value + Prediction
Metadata - Attaching a key

Dataflow

Key - Value

Apache Beam
RunInference Transform
Model
\( f(x) \)

Key - Value +

Prediction
Metadata - Attaching a key

- $\text{PCollection[bytes]} \rightarrow \text{PCollection[PredictionLog]}$
- $\text{PCollection[Tuple[K, bytes]]} \rightarrow \text{PCollection[Tuple[K, PredictionLog]]}$
- $\text{PCollection[Example]} \rightarrow \text{PCollection[PredictionLog]}$
- $\text{PCollection[Tuple[K, Example]]} \rightarrow \text{PCollection[Tuple[K, PredictionLog]]}$
- $\text{PCollection[Tuple[K, SequenceExample]]} \rightarrow \text{PCollection[Tuple[K, PredictionLog]]}$
Notebook
Live Experimentation

- Model metrics are usually not exact matches for business objectives
- Example: Recommender systems
  - Model trained on clicks
  - Business wants to maximize profit
  - Example: Different products have different profit margins
Live Experimentation - A/B Testing

- Users are divided into two groups
- Users are randomly routed to different models in environment
- You gather business results from each model to see which one is performing better
Multiple Models

\[ y = 5x \]

\[ y = 10x \]
Cascade Ensembles: Model > Model
Cascade Ensembles: Model > Model

Voice

Speech to Text

Text

Text to Speech

Language Understanding

Sentiment Analysis

Logs

Product Recommender

Support Recommender

Response
GPUs
Logical Pipeline View - The need for GPUs

- Execution graph created from code
- Use CPU for stages are more suited for CPU based processing
- Leverage GPU to accelerate specific stages of the pipeline
Dataflow GPU

Attach GPU to your Dataflow workers to accelerate your pipelines

- Select from a range of GPU types (NVIDIA T4, V100, P100, and P4) for your job. Up to 8 GPUs per instance
- Automatic provisioning/deprovisioning
- Simplify application lifecycle with support for Docker containers
Dataflow GPU: Architecture View

- **Container Image**
  - (SDK Worker Container)
  - User’s Beam code
  - ML Framework (TensorFlow, Pytorch)
  - NVIDIA CUDA-X libraries
  - NVIDIA CUDA Toolkit
  - Apache Beam SDK

- **Dataflow Worker**
  - GPU drivers
  - Dataflow Runner Process

Diagram showing the architecture of a Dataflow GPU setup, including worker nodes, dataflow runner processes, and integration with ML frameworks and NVIDIA libraries.
Thank you!