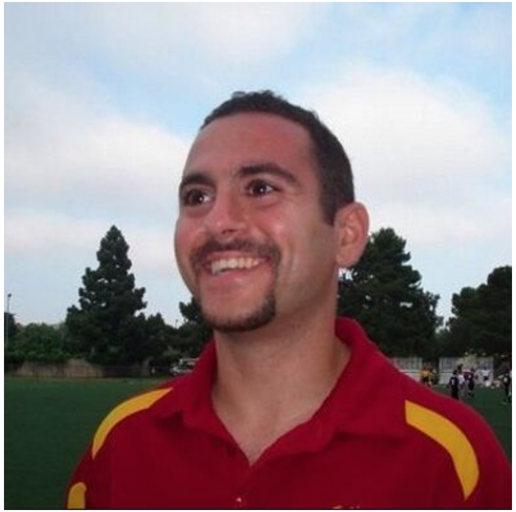
TensorFlow Extended Part 2

Model Build, Analysis & Serving

Armen Donigian

Who am I?



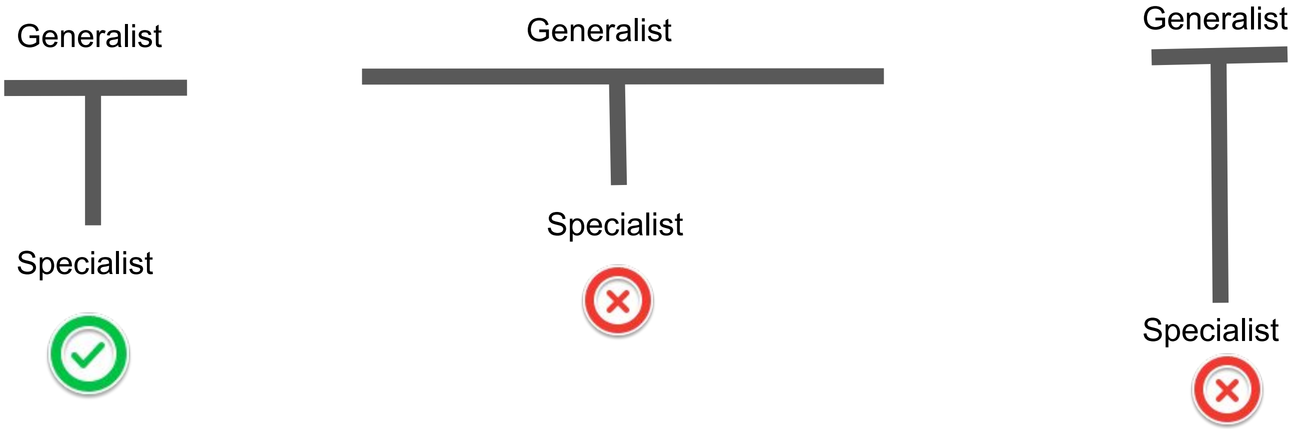
- Computer Science Undergrad degree @UCLA
- Computer Science Grad degree @USC
- 15+ years experience as Software & Data Engineer
- Computer Science Instructor
- Mentor @Udacity Deep Learning Nanodegree
- Real-time wagering algorithms @GamePlayerNetwork
- Differential GPS corrections @Jet Propulsion Laboratory, landing sequence for Mars Curiosity
- Years of experience in design, implementation & productionalization of machine learning models for several FinTech underwriting businesses
- Currently, head of personalization & recommender systems @Honey
- Available for Consulting (donigian@LevelUpInference.com)

Goals, Breadth vs Depth...

Goal: Provide context of the *requirements*, *tools* & *methodologies* involved with developing a production grade machine learning pipeline.

Slides will provide you with *breadth*.

Notebooks will provide you with *depth* (i.e. implementation details).



Lesson Roadmap

- **Overview of TFX: What problems it can help you solve (30 mins)**
  a. What is TFX & Why Should You Care?
  b. What can you leverage? TFX Ecosystem
  c. Which problems can TFX help you solve?
  d. TFX Components

  10 min Break

- **TensorFlow Estimator Overview (35 mins)**
  a. What is TensorFlow & Why Should You Care?
  b. What is TF Estimator?
  c. How to train a model using TF Estimator?
  d. Dataset Overview
  e. TF Estimator notebook demo

  10 min Break

- **TensorFlow Model Analysis Overview (40 mins)**
  a. What is it & why should you care?
  b. TFMA API Overview
  c. TFMA Usage
  d. TFMA notebook demo

  10 min Break

- **Tensorflow Serving (45 mins)**
  a. What is it & why should you care?
  b. TF Serving Intro
  c. TF Serving w/ Docker notebook demo
     i. CPU / GPU / TPU
  d. TF Model Server REST API

TensorFlow Extended Overview

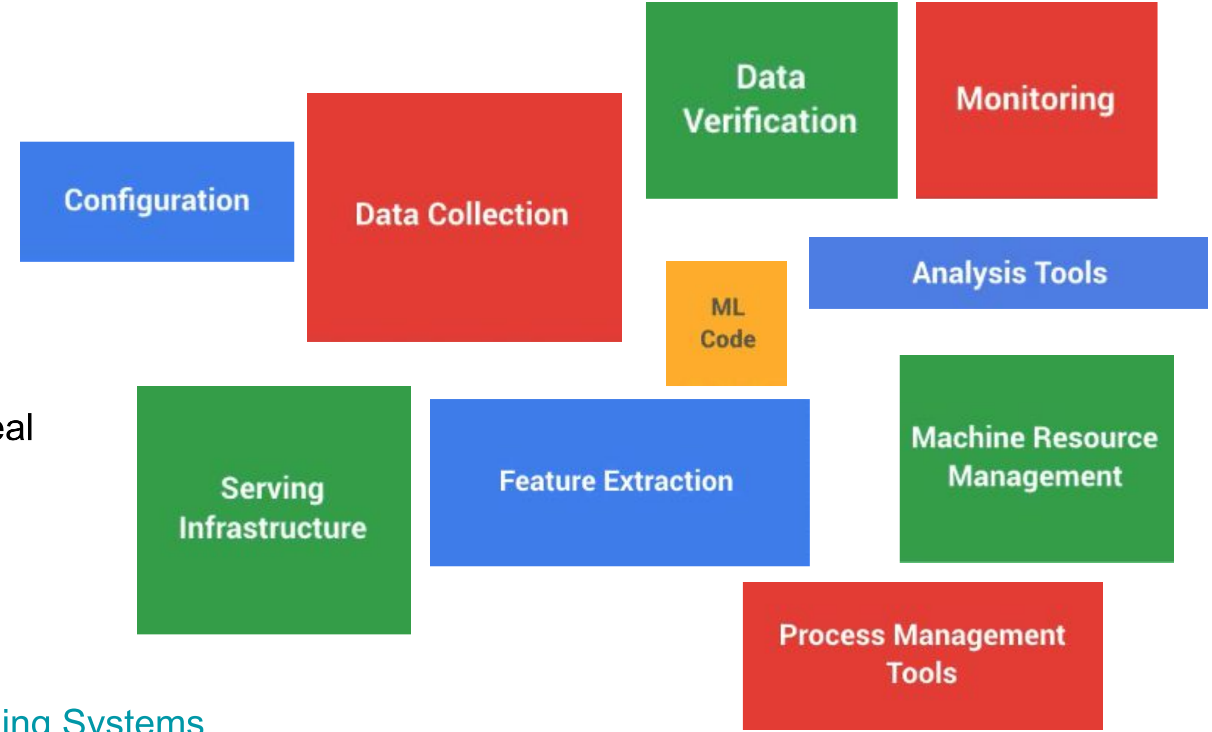
TensorFlow Extended (TFX)

TFX is…

- A general purpose machine learning platform implemented @Google
- A set of gluable components into one platform simplifying the development of end to end ML pipelines.
- An open source solution to reduce the time to production from months to weeks while minimizing custom, fragile solutions filled with tech debt.
- Used by Google to create & deploy their machine learning models.

Why Should You Care?

What you first think? **vs...** Real World ML Use Cases

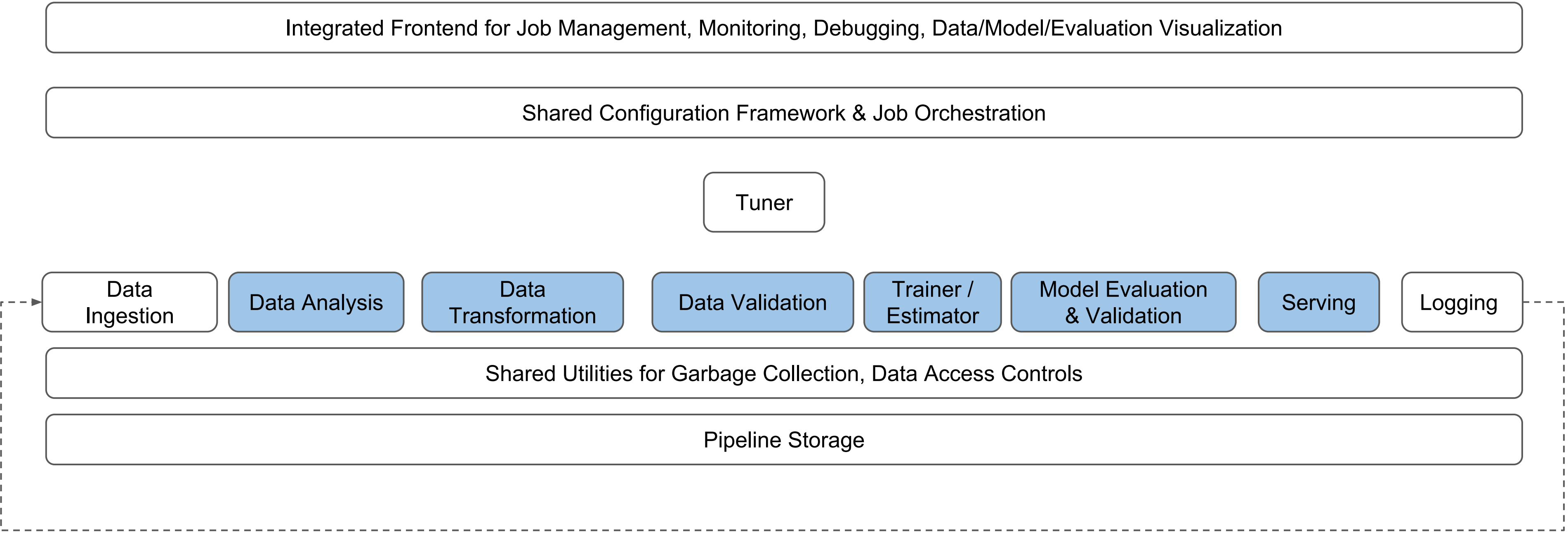


Takeaway: Doing machine learning in real world is HARD!

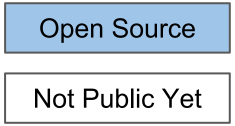
Building custom solutions is expensive, duplicative, fragile & leads to tech debt.

[Hidden Technical Debt in Machine Learning Systems]

What Can I Leverage: TFX Ecosystem



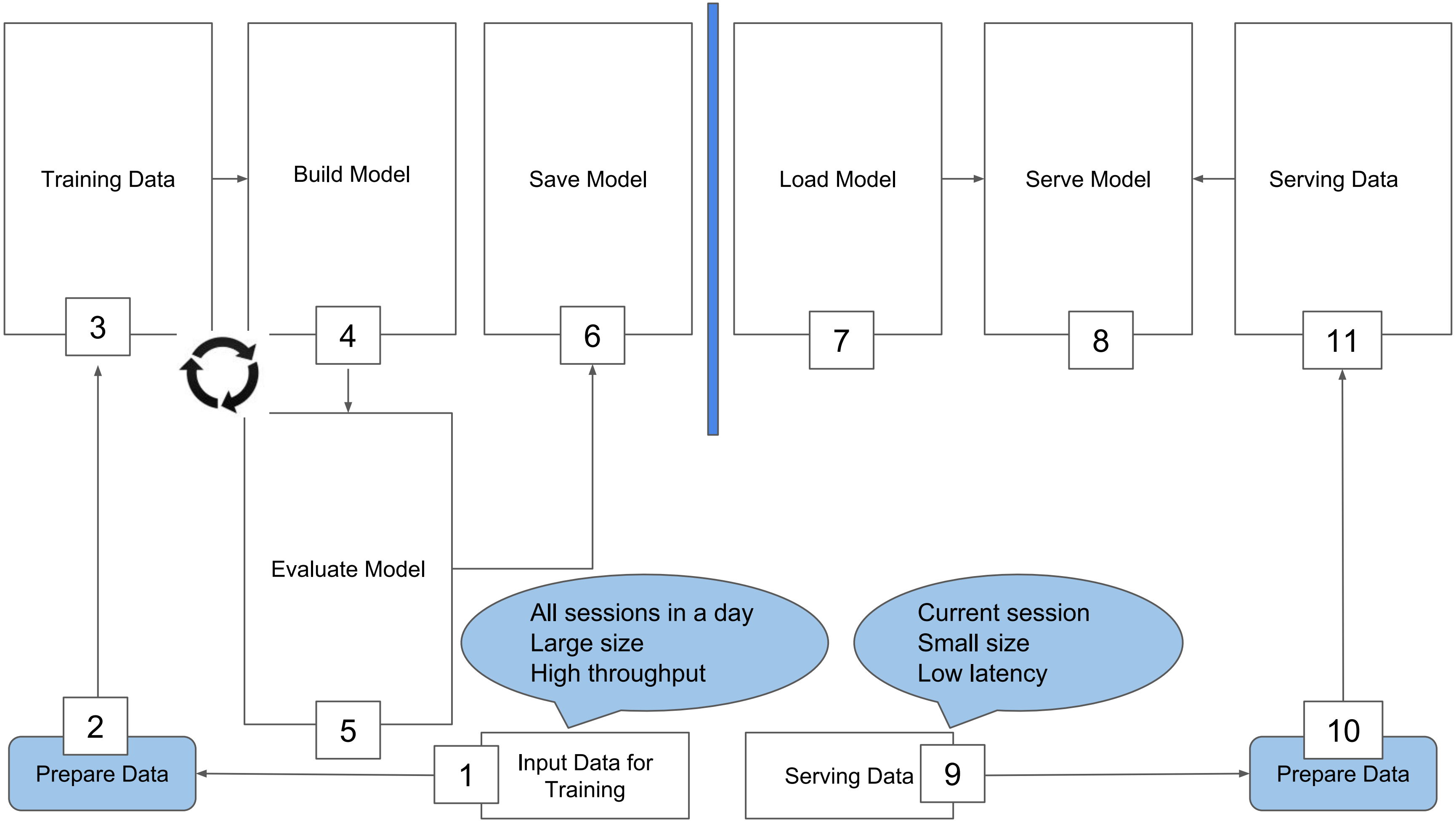
Machine Learning Platform Overview



[Link](#) to TFX paper

Train / Serving Data Flows

A Data Science Workflow, Glossary of ML terms, Diagram Reference



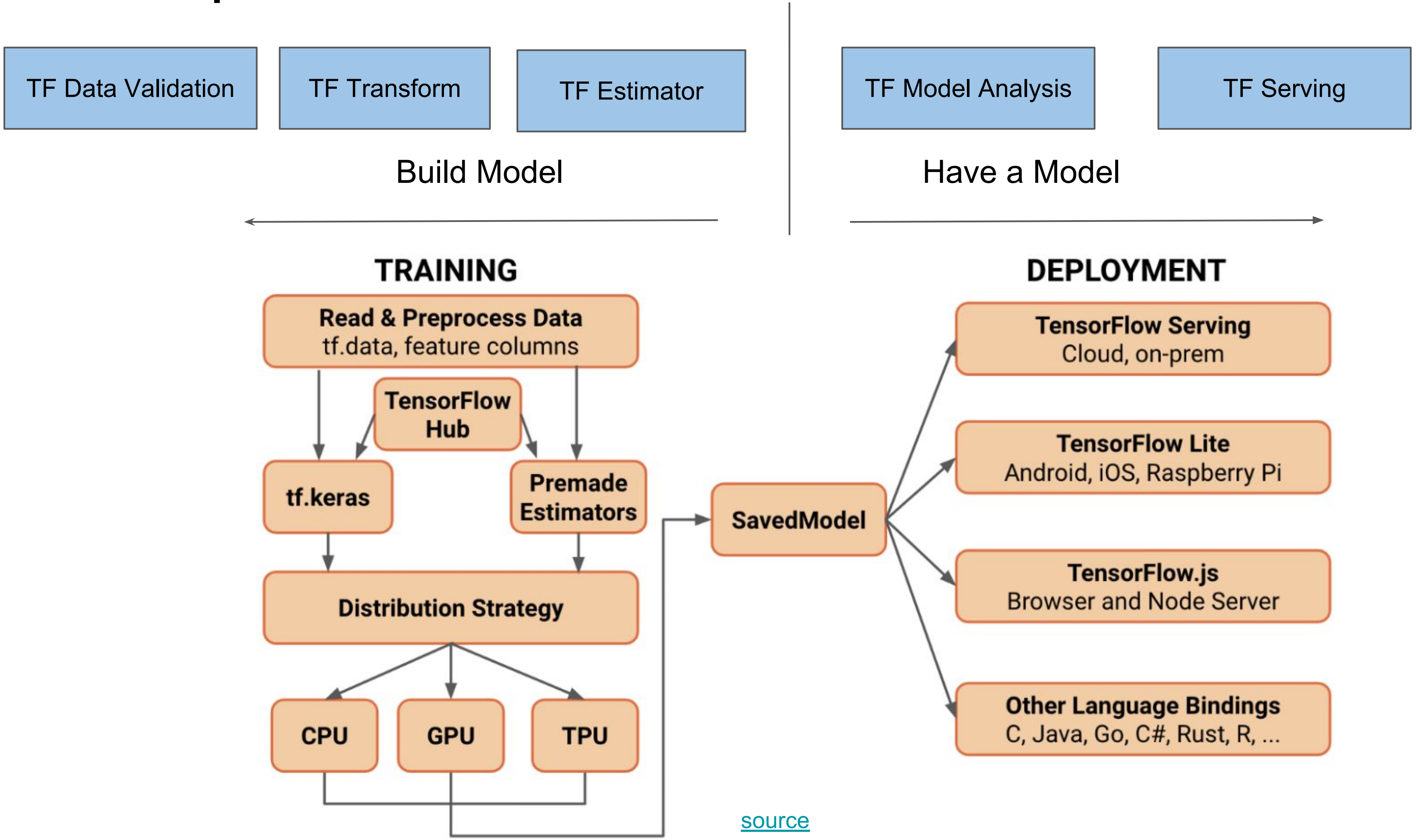
What Could Go Wrong...

In no particular order…

- What errors are in the data? Finding errors in GBs or TBs w/ O(1000s) of features is hard!
- How do I gain an understanding (analysis or visualization) of GBs or TBs w/ O(1000s) of features?
    - What is a reasonable data schema? How can I define a training vs serving context?
    - Does new data conform to previously inferred schema (validation)?
    - How can I detect when a signal is available in training but not in serving?
- Which data significantly affects the performance of the model?
- How different are the training vs test vs serving sets?
    - Are these differences important?
    - How can I define constraints on distribution of values?
- Which part of the data is problematic?
- How can I apply data transformations to GBs or TBs w/ O(1000s) of features in a scalable way?

Click links above to find related research papers & projects.

TFX Pipeline



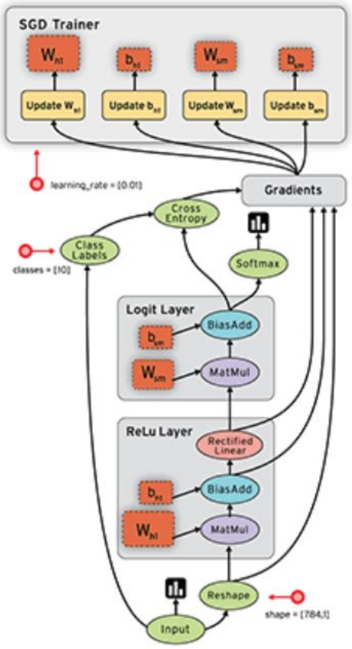
source

TensorFlow
Model Build

What is TF & Why Should You Care?

TensorFlow is an open source high performance library which uses directed graphs (see overview)

- Dataflow Model (link to paper)
  - Nodes represent math operations, Edges represent arrays of data
  - tf.math.add represented as..
    - single node w/ 2 input edges (matrices to be added)
    - 1 output edge (result of addition)
- **Flexible**
  - Works w/ image, audio, text and numerical data
- **Parallelism**
  - dataflow graph represents dependencies between operations, figure out which ops can execute in parallel)
- **Distributed Execution**
  - TF partitions your program across multiple devices (CPUs, GPUs, TPUs attached to different machines)
  - TF takes care of networking between machines
- **Compilation**
  - Benefit from compiler optimizations for dataflow graph using XLA
- **Portability**
  - Train model in Python, export SavedModel, serve in C++)

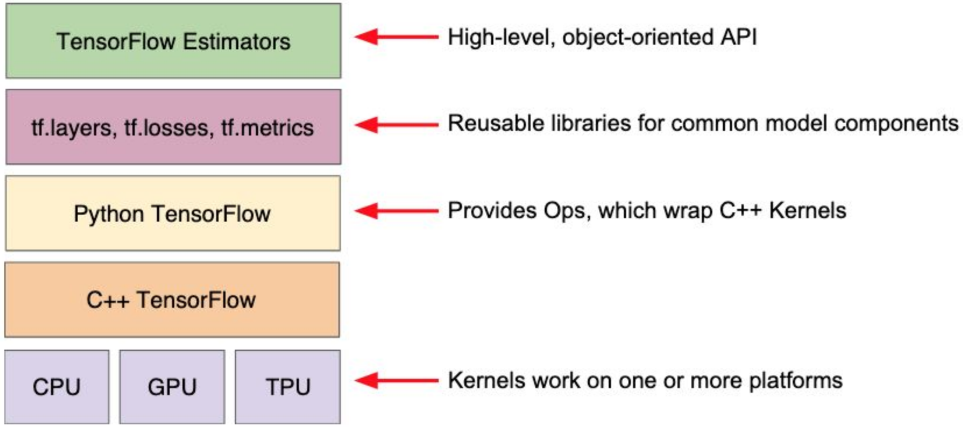


source

What is TF Estimator & Why Should You Care?

TF Estimator is a high level OOP API which makes it easier to train models (see overview)

- TF Estimator is compatible with the scikit-learn API
- Train models using CPU / GPU / TPUs
- Quicker model (graph) development
- Load large amounts of data
- Model checkpointing & recover from failures
- Train / Evaluation / Monitor
- Distributed Training
- Save summaries for TensorBoard
- Hyper-parameter tuning using ML Engine
- Serving predictions from a trained model
- Easily create Estimators from Keras models
- How to create custom estimators



source

How to build a model using TF Estimator?

```
import tensorflow as tf

# define feature columns
population = tf.feature_column.numeric_column('population')
crime_rate = tf.feature_column.numeric_column('crime_rate')
# assume education is a categorical
highest_education_level = tf.feature_column.categorical_column_with_vocabulary_list('highest_education_level',
                    vocabulary_list = ['high school', 'some college', 'graduated college'])

state_of_residence = tf.feature_column.categorical_column_with_identity('state_id',
                                                                        num_buckets = 50)

# one hot encoding
has_graduated = tf.feature_column.indicator_column('graduated')

# create an estimator using feature columns
model = tf.estimator.LinearClassifier(feature_columns=[population,
                                                       crime_rate,
                                                       highest_education_level,
                                                       state_of_residence,
                                                       has_graduated], './model_trained')

def train_input_fn():
   # preprocess dataset, extract features and the label
   return features, label

def serving_input_fn():
   # ...
   return features

# train, by using train_input_fn, you can use datasets larger than what you can put in memory
model.train(train_input_fn, steps=2000)

# predict
model.predict(serving_input_fn)
```

Dataset

Bucket Features	Dense Float Features	Vocab Features	Categorical Features
pickup_hour	trip_distance		
pickup_month	passenger_count		
pickup_day_of_week	tip_amount		
dropoff_month			
dropoff_hour			
dropoff_day_of_week			

bucketize	scale_to_z_score

Transformations

Target to predict: fare_amount

New York Yellow Cab dataset available via BigQuery public datasets

NYC Taxi Train/Evaluate

```python
def train_and_maybe_evaluate(hparams):
    schema = read_schema(hparams.schema_file)

    train_input = lambda: input_fn(
      hparams.train_files,
      hparams.tf_transform_dir,
      batch_size=TRAIN_BATCH_SIZE
    )
    eval_input = lambda: input_fn(
      hparams.eval_files,
      hparams.tf_transform_dir,
      batch_size=EVAL_BATCH_SIZE
    )

    train_spec = tf.estimator.TrainSpec(
      train_input, max_steps=hparams.train_steps)

    serving_receiver_fn = lambda: example_serving_receiver_fn(
      hparams.tf_transform_dir, schema)

    exporter = tf.estimator.FinalExporter('nyc-taxi', serving_receiver_fn)
    eval_spec = tf.estimator.EvalSpec(
      eval_input,
      steps=hparams.eval_steps,
      exporters=[exporter],
      name='nyc-taxi-eval')

    run_config = tf.estimator.RunConfig(
      save_checkpoints_steps=999, keep_checkpoint_max=1)

    serving_model_dir = os.path.join(hparams.output_dir, SERVING_MODEL_DIR)
    run_config = run_config.replace(model_dir=serving_model_dir)

    estimator = build_estimator(
      hparams.tf_transform_dir,

      # Construct layers sizes with exponetial decay
      hidden_units=[
          max(2, int(FIRST_DNN_LAYER_SIZE * DNN_DECAY_FACTOR**i))
          for i in range(NUM_DNN_LAYERS)
      ],
      config=run_config)
    tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
    return estimator
```

+ Using train_input func, allows for datasets larger than memory

+ HParams is a class to hold hyper-parameters (key, value pairs)

Things you can pass to a training method

Export model for TF Serving

Supports Distributed training & evaluation

TF Model Build Knowledge Check

Q: Which of the following is a supported feature column method in TF Estimator?

a) tf.feature_column.numeric_column()
b) tf.feature_column.categorical_column_with_vocabulary_list()
c) tf.feature_column.categorical_column_with_identity()
d) tf.feature_column.indicator_column()
e) All of the above

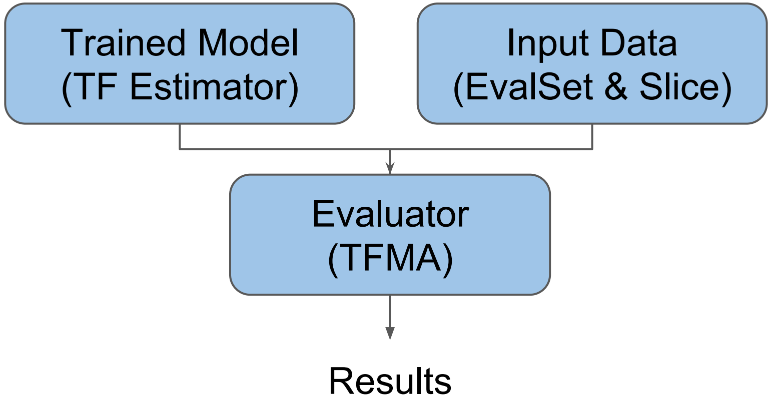
TensorFlow
Model Analysis

What is TF Model Analysis & Why Should You Care?

TF Model Analysis is a library for evaluating TF models.

Benefits include…

- Allows you to evaluate models on large amounts of data
- You can choose which metric & what slice/segment of your data to evaluate model predictions on
    - This helps you find slices of data for a given feature where the model performs poorly
    - Great model debugging tool
- Track performance over time
    - Trends of different models over time
    - As you get new data
- User friendly visualization tool



TensorFlow Model Analysis API

Feature Engineering @ Scale	**Transformations**
Evaluate & persist results.	tfma.ExtractEvaluateAndWriteResults()
Creates an EvalResult object for use with the visualization functions.	tfma.load_eval_result()
Run model analysis for a single model on multiple data sets.	tfma.multiple_data_analysis()
Run model analysis for multiple models on the same data set.	tfma.multiple_model_analysis()
Runs TensorFlow model analysis.	tfma.run_model_analysis()

TFMA Usage

```python
def run_tfma(slice_spec, tf_run_id, tfma_run_id, input_csv, schema_file, add_metrics_callbacks=None):
    eval_model_base_dir = os.path.join(get_tf_output_dir(tf_run_id), EVAL_MODEL_DIR)
    eval_model_dir = os.path.join(eval_model_base_dir, next(os.walk(eval_model_base_dir))[1][0])

    eval_shared_model = tfma.default_eval_shared_model(
        eval_saved_model_path=eval_model_dir,
        add_metrics_callbacks=add_metrics_callbacks)

    schema = read_schema(schema_file)
    display_only_data_location = input_csv

    with beam.Pipeline() as pipeline:
        csv_coder = make_csv_coder(schema)
        raw_data = (
            pipeline
            | 'ReadFromText' >> beam.io.ReadFromText(
                input_csv,
                coder=beam.coders.BytesCoder(),
                skip_header_lines=True)
            | 'ParseCSV' >> beam.Map(csv_coder.decode))

        # Examples must be in clean tf-example format.
        coder = make_proto_coder(schema)
        raw_data = (
            raw_data
            | 'ToSerializedTFExample' >> beam.Map(coder.encode))

        _ = (raw_data
             | 'ExtractEvaluateAndWriteResults' >>
             tfma.ExtractEvaluateAndWriteResults(
                 eval_shared_model=eval_shared_model,
                 slice_spec=slice_spec,
                 output_path=get_tfma_output_dir(tfma_run_id),
                 display_only_data_location=input_csv))

    return tfma.load_eval_result(output_path=get_tfma_output_dir(tfma_run_id))
```

TFMA Usage

```
tf.logging.set_verbosity(tf.logging.INFO)

# an empty slice spec means the whole dataset
OVERALL_SLICE_SPEC = tfma.SingleSliceSpec()

# data can be sliced along a feature column.
FEATURE_COLUMN_SLICE_SPEC = tfma.SingleSliceSpec(columns=['pickup_hour'])

# slices are computed for pickup_day_of_week x pickup_month.
FEATURE_COLUMN_CROSS_SPEC = tfma.SingleSliceSpec(columns=['trip_distance', 'tip_amount'])

# metrics is computed for all data where pickup_hour is 12
FEATURE_VALUE_SPEC = tfma.SingleSliceSpec(features=[('pickup_hour', 12)])

# we can mix column cross and feature value cross
COLUMN_CROSS_VALUE_SPEC = tfma.SingleSliceSpec(columns=['pickup_day_of_week'], features=[('pickup_hour', 12)])

ALL_SPECS = [
    OVERALL_SLICE_SPEC,
    FEATURE_COLUMN_SLICE_SPEC,
    FEATURE_COLUMN_CROSS_SPEC,
    FEATURE_VALUE_SPEC,
    COLUMN_CROSS_VALUE_SPEC
]

tfma_result_1 = run_tfma(input_csv=os.path.join(EVAL_DATA_DIR, 'eval.csv'),
                         tf_run_id=0,
                         tfma_run_id=1,
                         slice_spec=ALL_SPECS,
                         schema_file=get_schema_file())
```

```
[29]: tfma.view.render_slicing_metrics(tfma_result_1, slicing_column='pickup_hour')
```

Visualization

Slices Overview

Examples (Weighted) Threshold

0

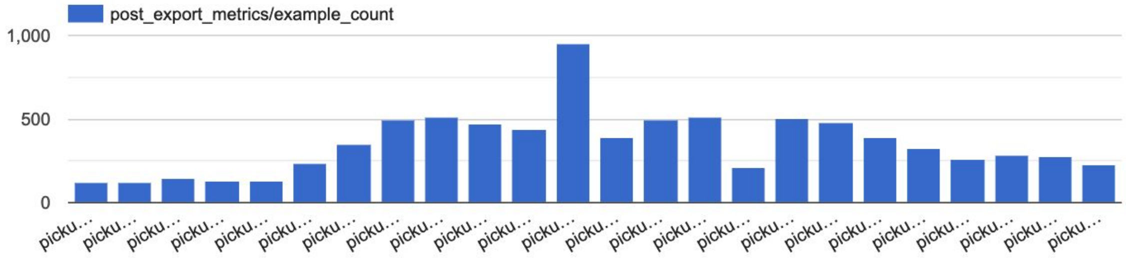
Number of rows per hour of day!

Show

post_export_metrics/example_count

Sort by

Slice



feature	average_loss	label/mean	post_export_metrics/example_count	prediction/mean
pickup_hour:10	525.09387	33.54107	470	32.56553
pickup_hour:11	176.18619	32.91829	443	32.55243
pickup_hour:8	88.13461	32.06847	498	31.39557
pickup_hour:9	86.65687	32.35592	515	31.66985
pickup_hour:14	93.70968	33.71512	496	31.97082
pickup_hour:15	101.93111	34.50546	513	32.26447
pickup_hour:12	79.43258	35.09818	952	33.88065
pickup_hour:13	80.08283	33.61849	392	33.47749

```
tfma.view.render_slicing_metrics(tfma_result_1, slicing_column='pickup_hour')
```

Visualization

Slices Overview

Examples (Weighted) Threshold

0

Note distribution of mean prediction

Show

prediction/mean

Sort by

Slice

feature	average_loss	label/mean	post_export_metrics/example_count	prediction/mean
pickup_hour:10	525.09387	33.54107	470	32.56553
pickup_hour:11	176.18619	32.91829	443	32.55243
pickup_hour:8	88.13461	32.06847	498	31.39557
pickup_hour:9	86.65687	32.35592	515	31.66985
pickup_hour:14	93.70968	33.71512	496	31.97082
pickup_hour:15	101.93111	34.50546	513	32.26447
pickup_hour:12	79.43258	35.09818	952	33.88065
pickup_hour:13	80.08283	33.61849	392	33.47749

```
tfma.view.render_slicing_metrics(tfma_result_1, slicing_column='pickup_hour')
```

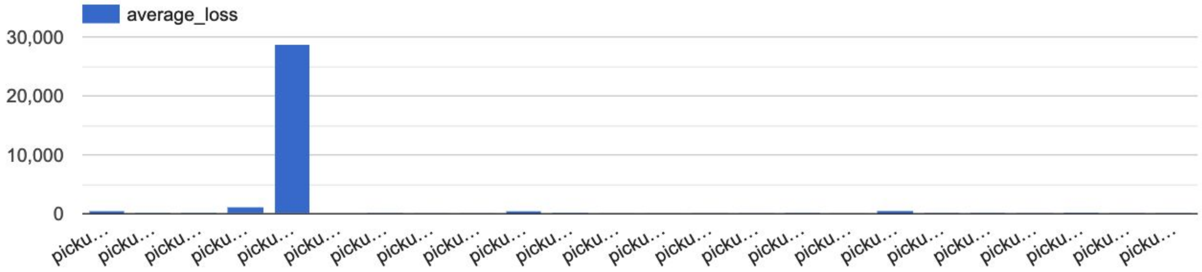
Visualization
Slices Overview

Examples (Weighted) Threshold
0

Note average loss anomaly!

Show
average_loss

Sort by
Slice



feature	average_loss	label/mean	post_export_metrics/example_count	prediction/mean
pickup_hour:10	525.09387	33.54107	470	32.56553
pickup_hour:11	176.18619	32.91829	443	32.55243
pickup_hour:8	88.13461	32.06847	498	31.39557
pickup_hour:9	86.65687	32.35592	515	31.66985
pickup_hour:14	93.70968	33.71512	496	31.97082
pickup_hour:15	101.93111	34.50546	513	32.26447
pickup_hour:12	79.43258	35.09818	952	33.88065
pickup_hour:13	80.08283	33.61849	392	33.47749

```
tfma.view.render_slicing_metrics(tfma_result_1, slicing_spec=COLUMN_CROSS_VALUE_SPEC)
```

Visualization

Slices Overview

Examples (Weighted) Threshold

0

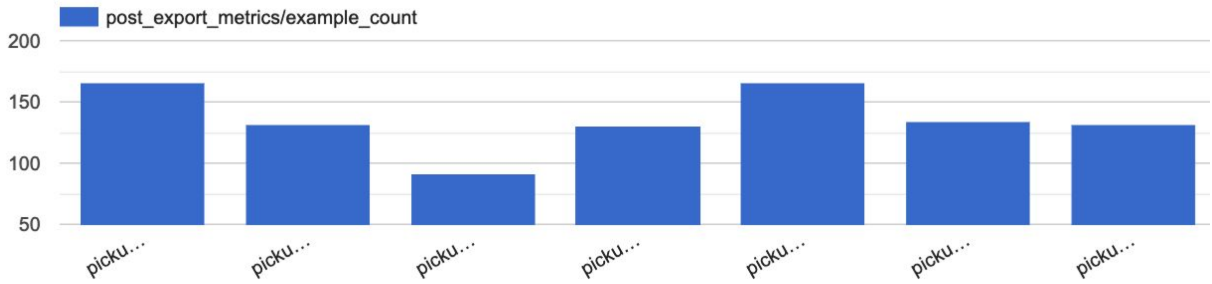
Feature Cross Analysis

Show

post_export_metrics/example_count

Sort by

Slice



feature	average_loss	label/mean	post_export_metrics/example_coun
pickup_day_of_week_X_pickup_hour:3_X_12	102.29386	36.47470	16
pickup_day_of_week_X_pickup_hour:2_X_12	37.41487	34.57879	13
pickup_day_of_week_X_pickup_hour:1_X_12	92.20728	39.41304	9
pickup_day_of_week_X_pickup_hour:5_X_12	93.57827	32.34354	13
pickup_day_of_week_X_pickup_hour:6_X_12	83.22913	36.78675	16
pickup_day_of_week_X_pickup_hour:4_X_12	53.81750	35.88806	13
pickup_day_of_week_X_pickup_hour:7_X_12	91.09421	30.66667	13

TF Model Analysis Knowledge Check

Q: TFMA is only useful if you're building a model using TensorFlow?

a) True
b) False

TensorFlow Serving

What is TF Serving & Why Should You Care?

Requirements of a Model Serving System...

1. Low latency
   a. Isolation of load & serve threads
2. Efficient
   a. Dynamic request batching
3. Scale Horizontally
4. Reliable & Robust
5. Support loading/hosting multiple model versions dynamically
   a. Serve one model, while sending canary requests to new model
   b. Built in A/B testing
6. Deployment roll forward / backward
7. Serves over 1,500 models @Google, 100 predictions/sec

Dockerfile(s) maintained by Google

- Dockerfile, VM w/ TensorFlow Serving
- Dockerfile.gpu, VM w/ TensorFlow Serving (GPU support to be used with nvidia-docker)
- Dockerfile.devel, VM w/ all dependencies needed to build TensorFlow Serving
- Dockerfile.devel-gpu, VM w/ all dependencies needed to build TensorFlow Serving w/ GPU support.

Test Drive it Yourself...

```bash
#!/bin/bash

# Download the TensorFlow Serving Docker image and repo
docker pull tensorflow/serving

# for GPU, use...
docker pull tensorflow/serving:latest-gpu

git clone https://github.com/tensorflow/serving
# Location of demo models
TESTDATA="$(pwd)/serving/tensorflow_serving/servables/tensorflow/testdata"

# Start TensorFlow Serving container and open the REST API port
docker run -t --rm -p 8501:8501 \
    -v "$TESTDATA/saved_model_half_plus_two_cpu:/models/half_plus_two" \
    -e MODEL_NAME=half_plus_two \
    tensorflow/serving &

# For GPU, use...
# docker run --runtime=nvidia -p 8501:8501 \
#  --mount type=bind,\
#  source=/tmp/tfserving/serving/tensorflow_serving/servables/tensorflow/testdata/saved_model_half_plus_two_gpu,\
#  target=/models/half_plus_two \
#  -e MODEL_NAME=half_plus_two -t tensorflow/serving:latest-gpu &

# Query the model using the predict API
curl -d '{"instances": [1.0, 2.0, 5.0]}' \
    -X POST http://localhost:8501/v1/models/half_plus_two:predict

# Returns => { "predictions": [2.5, 3.0, 4.5] }
```

TF Serving Out of the Box (w/ Docker)

```
# set name of Tensorflow Serving docker image & download it
# https://github.com/tensorflow/serving/tree/master/tensorflow_serving/tools/docker
DOCKER_IMAGE_NAME=tensorflow/serving
echo Download TF Serving docker image: $DOCKER_IMAGE_NAME
docker pull $DOCKER_IMAGE_NAME

# location of local model to be used by Tensorflow Serving
# should contain a folder named with a UTC timestamp
MODEL_BASE_PATH=$(pwd)/tf/run_0/serving_model_dir/export/nyc-taxi

# location of model dir within container
CONTAINER_MODEL_BASE_PATH=/models/nyc-taxi

# local port where to send inference requests
HOST_PORT=9000

# container listening port for inference requests
CONTAINER_PORT=8500

echo Model directory: $MODEL_BASE_PATH

docker run -it \
  -p 127.0.0.1:$HOST_PORT:$CONTAINER_PORT \
  -v $MODEL_BASE_PATH:$CONTAINER_MODEL_BASE_PATH \
  -e MODEL_NAME=nyc-taxi\
  --rm $DOCKER_IMAGE_NAME
```

SavedModel Artifacts

After training, we have a trained saved model (universal format)

- Learned variable weights
- Graph
- Embeddings & Vocabs
- Inferred Schema
- Transformed features

```
notebooks/tft_train/run_0
├── tftransform_tmp
│   ├── 50a3468d584b42839cfc3c72e5a56e5f
│   │   ├── saved_model.pb
│   │   └── variables
│   ├── 582667bf1cf54f3a9fee504dbaef4ffd
│   │   ├── saved_model.pb
│   │   └── variables
│   └── 956a82774bf1480892cce94dba33eddd
│       ├── saved_model.pb
│       └── variables
├── train_transformed-00000-of-00001.gz
├── transform_fn
│   ├── saved_model.pb
│   └── variables
└── transformed_metadata
    └── v1-json
        └── schema.json
```

TF Serving (w/ Docker)

```python
def do_inference(model_handle, examples_file, num_examples, schema):
    filtered_features = [
      feature for feature in schema.feature if feature.name != LABEL_KEY
    ]
    del schema.feature[:]
    schema.feature.extend(filtered_features)

    csv_coder = make_csv_coder(schema)
    proto_coder = make_proto_coder(schema)

    input_file = open(examples_file, 'r')
    # skip header line
    input_file.readline()

    serialized_examples = []
    for _ in range(num_examples):
        one_line = input_file.readline()
        if not one_line:
            print('End of example file reached')
            break
        one_example = csv_coder.decode(one_line)

        serialized_example = proto_coder.encode(one_example)
        serialized_examples.append(serialized_example)

    parsed_model_handle = model_handle.split(':')
    do_local_inference(
      host=parsed_model_handle[0],
      port=parsed_model_handle[1],
      serialized_examples=serialized_examples)
```

TF Serving Inference

```
SERVER="127.0.0.1:9000"
SCHEMA_FILE="./schema.pbtxt"
NUM_INSTANCES=15
INSTANCE_FILE="../data/train/train.csv"

do_inference(SERVER,
             INSTANCE_FILE,
             NUM_INSTANCES,
             read_schema(SCHEMA_FILE))
```

```
outputs {
  key: "predictions"
  value {
    dtype: DT_FLOAT
    tensor_shape {
      dim {
        size: 15
      }
      dim {
        size: 1
      }
    }
    float_val: 32.9800186157
    float_val: 17.2102165222
    float_val: 31.7519054413
    float_val: 33.0067481995
    float_val: 36.803691864
    float_val: 33.8259849548
    float_val: 34.7675018311
    float_val: 22.1285438538
    float_val: 43.9800224304
    float_val: 22.4845104218
    float_val: 37.6693458557
    float_val: 34.8329048157
    float_val: 31.9676322937
    float_val: 70.7163391113
    float_val: 37.2004470825
  }
}
model_spec {
  name: "nyc-taxi"
  version {
    value: 1551254584
  }
  signature_name: "predict"
}
```

TF Serving ModelServer REST API

```
First, you'll need to install TF Model Server...
apt-get remove tensorflow-model-server

POST http://host:port/<URI>:<VERB>
URI - /v1/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]
VERBS - classify | regress | predict

Classify Format:
POST http://host:port/v1/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]:classify

Classify Example:
POST http://host:port/v1/models/iris/versions/1:classify


Predict Format:
POST http://host:port/v1/models/${MODEL_NAME}[/versions/${MODEL_VERSION}]:predict

Predict Example:
POST http://host:port/v1/models/mnist/versions/1:predict
```

End-2-End example

Next Steps:

- Work through TFMA & TFServing Notebooks