QuickDraw End2end

EE-559 Deep Learning, EPFL, 5/30/18
Andreas Steiner, Guest Lecture
https://fleuret.org/dlc

Agenda

Today

- In Practice ⚠
- Low-Level Reps
- Data
- Estimator
- Cloud
- Q+A

A Word of Warning

"Machine Learning: The High-Interest Credit Card of Technical Debt"

- Performance improvement vs. technical debt.
- CACE : Changing Anything Changes Everything.
- Hidden feedback loops.
- Data dependencies (and the world moves on).

(Sculley 2014)

Martin Zinkevich
https://developers.google.com/machine-learning/rules-of-ml/

Rules of ML

Before ML
1. Don't be afraid to launch a product without machine learning.
2. First, design and implement metrics.
3. Choose machine learning over a complex heuristic.

ML Phase I: Your First Pipeline
4. Keep the first model simple and get the infrastructure right.
5. Test the infrastructure independently from the machine learning.
6. Be careful about dropped data when copying pipelines.
7. Turn heuristics into features, or handle them externally.

Monitoring
8. Know the freshness requirements of your system.
9. Detect problems before exporting models.
10. Watch for silent failures.
11. Give feature columns owners and documentation.

Your First Objective:
12. Don't overthink which objective you choose to directly optimize.
13. Choose a simple, observable and attributable metric for your first objective
14. Starting with an interpretable model makes debugging easier.
15. Separate Spam Filtering and Quality Ranking in a Policy Layer.

ML Phase II: Feature Engineering
16. Plan to launch and iterate.
17. Start with directly observed and reported features as opposed to learned features.
18. Explore with features of content that generalize across contexts.
19. Use very specific features when you can.
20. Combine and modify existing features to create new features in human--understandable ways.
21. The number of feature weights you can learn in a linear model is roughly proportional to the amount of data you have.
22. Clean up features you are no longer using.

Human Analysis of the System
23. You are not a typical end user.
24. Measure the delta between models.
25. When choosing models, utilitarian performance trumps predictive power.
26. Look for patterns in the measured errors, and create new features.
27. Try to quantify observed undesirable behavior.
28. Be aware that identical short-term behavior does not imply identical long-term behavior.

Training-Serving Skew
29. The best way to make sure that you train like you serve is to save the set of features used at serving time, and then pipe those features to a log to use them at training time.
30. Importance-weight sampled data, don't arbitrarily drop it!
31. Beware that if you join data from a table at training and serving time, the data in the table may change.
32. Re-use code between your training pipeline and your serving pipeline whenever possible.
33. If you produce a model based on the data until January 5th, test the model on the data from January 6th and after.
34. In binary classification for filtering (such as spam detection or determining interesting emails), make small short-term sacrifices in performance for very clean data.
35. Beware of the inherent skew in ranking problems.
36. Avoid feedback loops with positional features.
37. Measure Training/Serving Skew.

ML Phase III: Slowed Growth, Optimization Refinement, and Complex Models
38. Don't waste time on new features if unaligned objectives have become the issue.
39. Launch decisions are a proxy for long-term product goals.
40. Keep ensembles simple.
41. When performance plateaus, look for qualitatively new sources of information to add rather than refining existing signals.
42. Don't expect diversity, personalization, or relevance to be as correlated with popularity as you think they are.
43. Your friends tend to be the same across different products. Your interests tend not to be.

Martin Zinkevich

https://developers.google.com/machine-learning/rules-of-ml/

Rules of ML

#2: First, design and implement metrics.

Before ML

1. Don't be afraid to launch a product without machine learning.
2. First, design and implement metrics.
3. Choose machine learning over a complex heuristic.

ML Phase I: Your First Pipeline

4. Keep the first model simple and get the infrastructure right.
5. Test the infrastructure independently from the machine learning.
6. Be careful about dropped data when copying pipelines.
7. Turn heuristics into features, or handle them externally.

Monitoring

8. Know the freshness requirements of your system.
9. Detect problems before exporting models.
10. Watch for silent failures.
11. Give feature columns owners and documentation.

Your First Objective:

12. Don't overthink which objective you choose to directly optimize.
13. Choose a simple, observable and attributable metric for your first objective
14. Starting with an interpretable model makes debugging easier.
15. Separate Spam Filtering and Quality Ranking in a Policy Layer.

ML Phase II: Feature Engineering

16. Plan to launch and iterate.
17. Start with directly observed and reported features as opposed to learned features.
18. Explore with features of content that generalize across contexts.
19. Use very specific features when you can.
20. Combine and modify existing features to create new features in human--understandable ways.
21. The number of feature weights you can learn in a linear model is roughly proportional to the amount of data you have.
22. Clean up features you are no longer using.

...lysis of the System

... a typical end user.

...e delta between models.

...sing models, utilitarian performance trumps predictive power.

... patterns in the measured errors, and create new features.

27. Try to quantify observed undesirable behavior.
28. Be aware that identical short-term behavior does not imply identical long-term behavior.

Training-Serving Skew

29. The best way to make sure that you train like you serve is to save the set of features used at serving time, and then pipe those features to a log to use them at training time.
30. Importance-weight sampled data, don't arbitrarily drop it!
31. Beware that if you join data from a table at training and serving time, the data in the table may change.
32. Re-use code between your training pipeline and your serving pipeline whenever possible.
33. If you produce a model based on the data until January 5th, test the model on the data from January 6th and after.
34. In binary classification for filtering (such as spam detection or determining interesting emails), make small short-term sacrifices in performance for very clean data.
35. Beware of the inherent skew in ranking problems.
36. Avoid feedback loops with positional features.
37. Measure Training/Serving Skew.

ML Phase III: Slowed Growth, Optimization Refinement, and Complex Models

38. Don't waste time on new features if unaligned objectives have become the issue.
39. Launch decisions are a proxy for long-term product goals.
40. Keep ensembles simple.
41. When performance plateaus, look for qualitatively new sources of information to add rather than refining existing signals.
42. Don't expect diversity, personalization, or relevance to be as correlated with popularity as you think they are.
43. Your friends tend to be the same across different products. Your interests tend not to be.

Martin Zinkevich

https://developers.google.com/machine-learning/rules-of-ml/

Rules of ML

Before ML

1. Don't be afraid to launch a product without machine learning.
2. First, design and implement metrics.
3. Choose machine learning over a complex heuristic.

ML Phase I: Your First Pipeline

4. Keep the first model simple and get the infrastructure right.
5. Test the infrastructure independently from the machine learning.
6. Be careful about dropped data when copying pipelines.
7. Turn heuristics into features, or handle them externally.

Monitoring

8. Know the freshness requirements of your system.
9. Detect problems before exporting models.
10. Watch for silent failures.
11. Give feature columns owners and documentation.

Your First Objective:

12. Don't overthink which objective you choose to directly optimize.
13. Choose a simple, observable and attributable metric for your first objective
14. Starting with an interpretable model makes debugging easier.
15. Separate Spam Filtering and Quality Ranking in a Policy Layer.

ML Phase II: Feature Engineering

16. Plan to launch and iterate.
17. Start with directly observed and reported features as opposed to learned features.
18. Explore with features of content that generalize across contexts.
19. Use very specific features when you can.
20. Combine and modify existing features to create new features in human--understandable ways.
21. The number of feature weights you can learn in a linear model is roughly proportional to the amount of data you have.
22. Clean up features you are no longer using.

Human Analysis of the System

23. You are not a typical end user.
24. Measure the delta between models.
25. When choosing models, utilitarian performance trumps predictive power.
26. Look for patterns in the measured errors, and create new features.
27. Try to quantify observed undesirable behavior.
28. Be aware that identical short-term behavior does not imply identical long-term behavior.

#4: Keep the first model simple and get the infrastructure right.

...e the set of ...log to use them

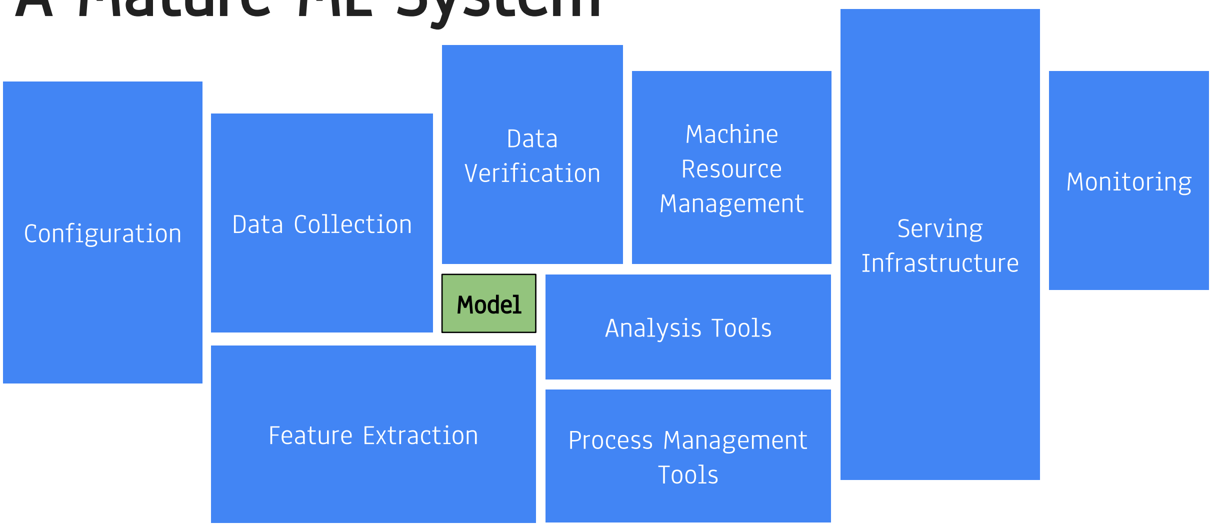
... time, the data in the table may change.

32. Re-use code between your training pipeline and your serving pipeline whenever possible.
33. If you produce a model based on the data until January 5th, test the model on the data from January 6th and after.
34. In binary classification for filtering (such as spam detection or determining interesting emails), make small short-term sacrifices in performance for very clean data.
35. Beware of the inherent skew in ranking problems.
36. Avoid feedback loops with positional features.
37. Measure Training/Serving Skew.

ML Phase III: Slowed Growth, Optimization Refinement, and Complex Models

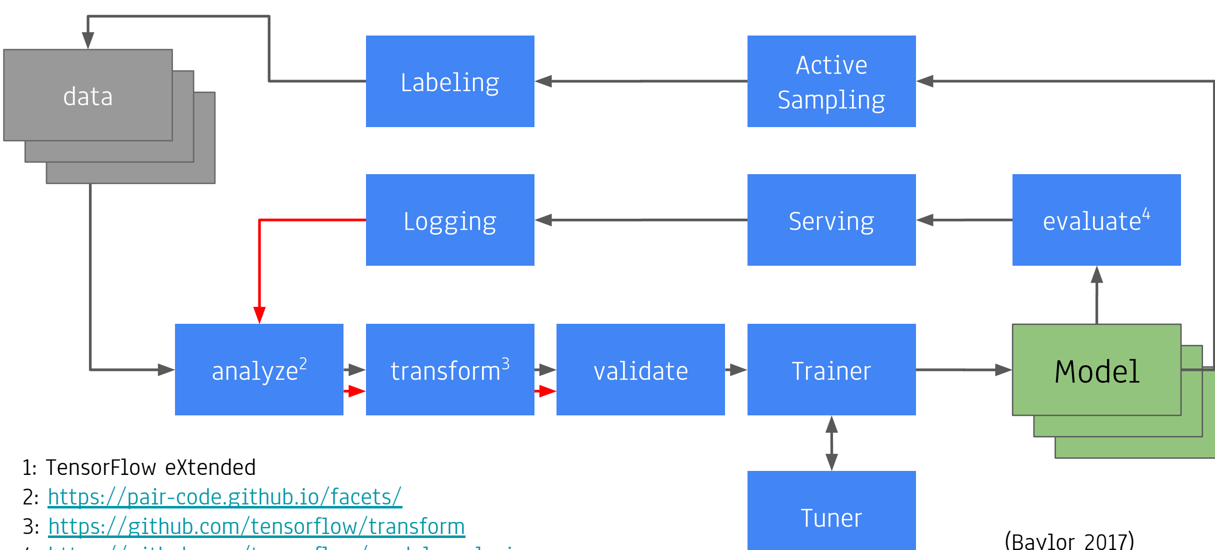
38. Don't waste time on new features if unaligned objectives have become the issue.
39. Launch decisions are a proxy for long-term product goals.
40. Keep ensembles simple.
41. When performance plateaus, look for qualitatively new sources of information to add rather than refining existing signals.
42. Don't expect diversity, personalization, or relevance to be as correlated with popularity as you think they are.
43. Your friends tend to be the same across different products. Your interests tend not to be.

A Mature ML System



(Sculley 2015)

TFX[1]: general-purpose ML platform



1: TensorFlow eXtended
2: https://pair-code.github.io/facets/
3: https://github.com/tensorflow/transform
4: https://github.com/tensorflow/model-analysis

(Baylor 2017)

Agenda

Today

- In Practice ⚠
- Low-Level Reps
- Data
- Estimator
- Cloud
- Q+A

About that Feedback ...

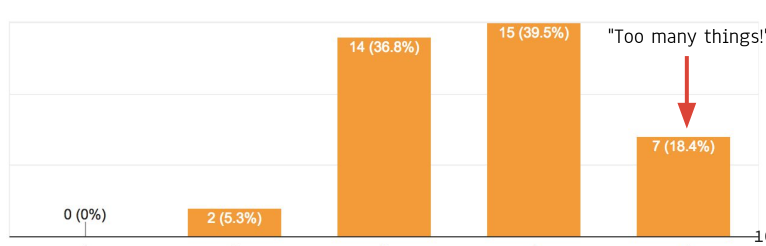
Different Parts:

- ML Research : 4.3/5
- From ML To App : 4.1/5
- TF Goodness : 4.0/5
- Low-Level TF : 4.0/5
- **Putting It Together: 4.3/5**
- TF vs. PyTorch 4.1/5
- Overall : 4.4/5

Prior TF Experience:

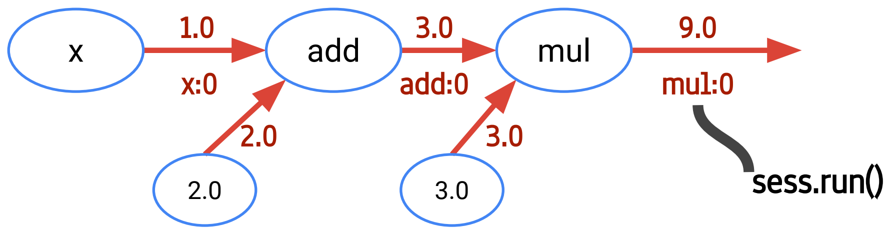
- Already developed some models: 10%
- Using daily : 8%

Information Density:



Low-Level TensorFlow - Reloaded

The Session **executes the Graph**, similar to the JVM running Java bytecode.



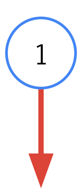
build graph...

```
x = tf.constant(1.0, name="x")
add = tf.add(x, 2.0, name="add")
mul = tf.multiply(add, 3.0, name="mul")
```

...run graph

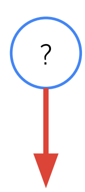
```
with tf.Session() as sess:
  mul_, = sess.run([mul])
  print(mul_)
```


Const. vs. Placeholder vs. Variable

Const 

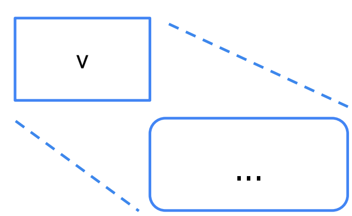

```
a = tf.constant(1)
a.eval()
```

- Value part of graph
- Immutable.

Placeholder 


```
b = tf.placeholder(tf.int32)
b.eval(feed_dict={b: 1})
```

- Value must be provided




```
v = tf.Variable(1, 'v')
sess.run(tf.global_variables_init
ializer())
sess.run(tf.assign_add(v, 1))
```

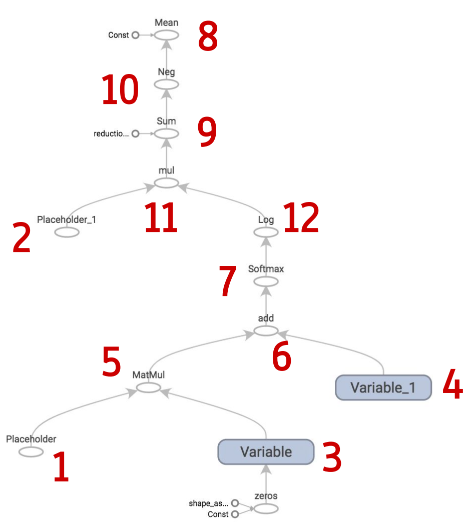
- Value part of graph.
- **Mutable**!

…, Defining the Model, …

```
graph = tf.Graph()

with graph.as_default():
  x = tf.placeholder(tf.float32, [None, 784])
  y_ = tf.placeholder(tf.float32, [None, 10])

  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))

  logits = tf.matmul(x, W) + b
  y = tf.nn.softmax(logits)

  cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), axis=1))
  optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.5)
  train_step = optimizer.minimize(cross_entropy)
```

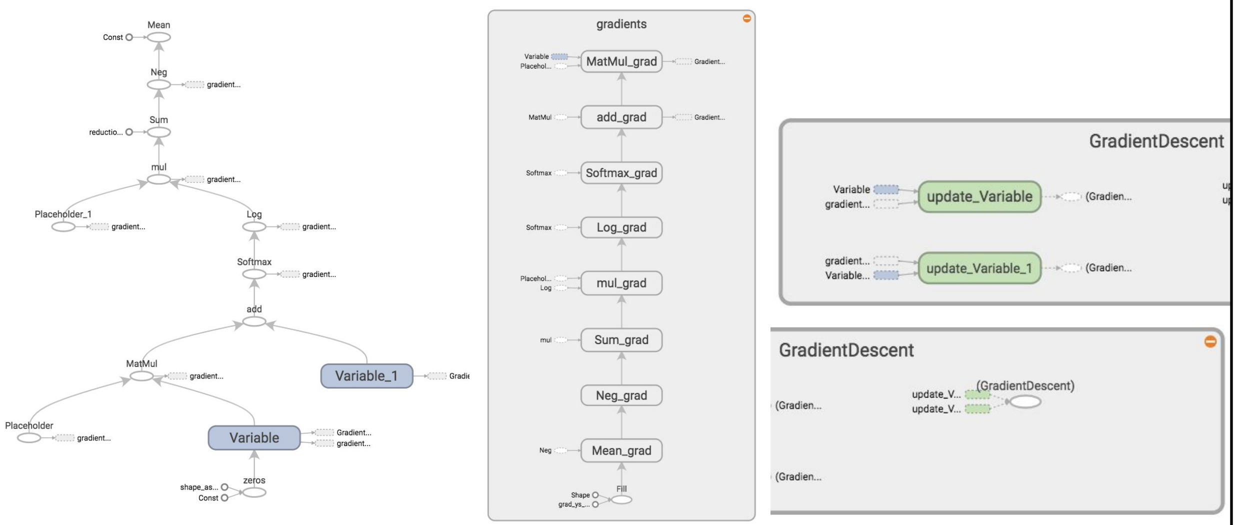
Creates operations for backward pass…

The Graph *Before* optimizer.minimize()




```
 1:   x = tf.placeholder(tf.float32, [None, 784])
 2:   y_ = tf.placeholder(tf.float32, [None, 10])

 3:   W = tf.Variable(tf.zeros([784, 10]))
 4:   b = tf.Variable(tf.zeros([10]))

 5:   logits = (tf.matmul(x, W)
 6:             + b)
 7:   y = tf.nn.softmax(logits)

      cross_entropy = (
 8:       tf.reduce_mean(
 9,10         -tf.reduce_sum(
11:             y_ *
12:             tf.log(y), axis=1))
```

The Graph *After* optimizer.minimize()

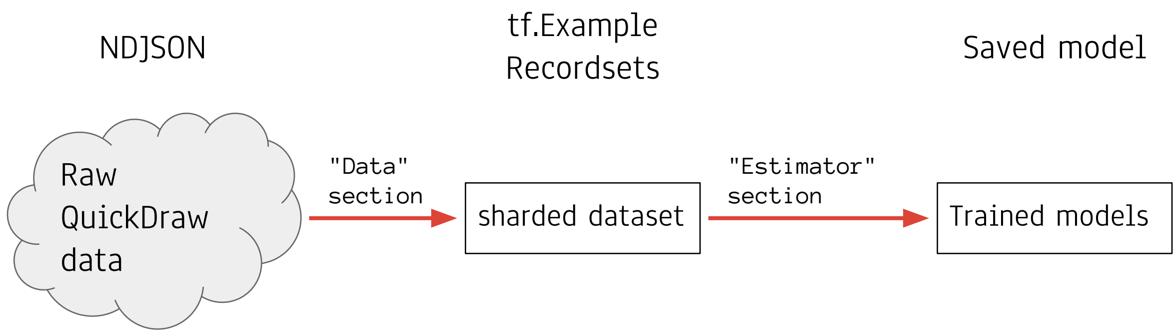


Agenda

Today

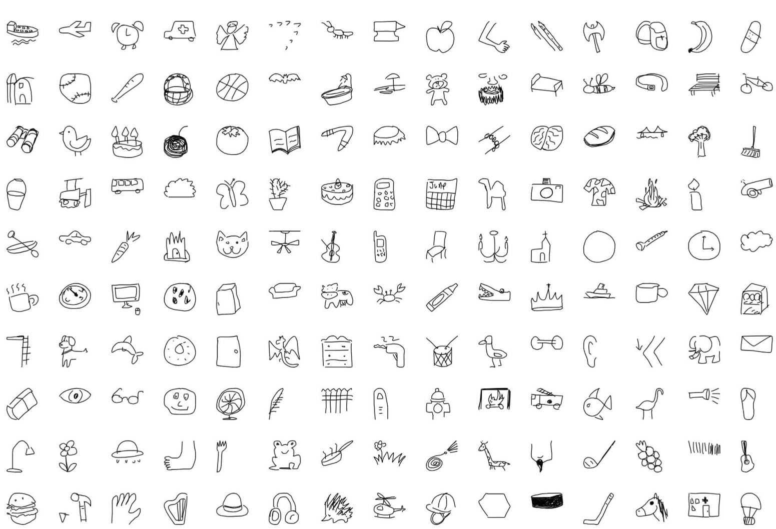
- In Practice ⚠
- Low-Level Reps
- Data
- Estimator
- Cloud
- Q+A

Machine Learning - In Two Steps



QuickDraw Dataset

https://quickdraw.withgoogle.com/data/



- 50M drawings
- 15M players
- 309 categories
- Raw stroke data

Download + Inspect

https://console.cloud.google.com/storage/browser/quickdraw_dataset/full/simplified

garden hose.ndjson	51.52 MB
garden.ndjson	97.89 MB
giraffe.ndjson	58.27 MB
goatee.ndjson	124.9 MB
golf club.ndjson	60.56 MB

File format: NDJSON

```
{"word":"giraffe","countrycode":"BR","time...
{"word":"giraffe","countrycode":"FR","time...
```

1 JSON object / line:

```
{
  "word": "giraffe",
  "countrycode": "BR",
  "timestamp": "2017-01-26 ...",
  "recognized": true,
  "key_id": "5807561943023616",
  "drawing": [...]
}
```

Drawing

```
[
 [[20, 39], [8, 23]],
 [[51, 52], [0, 11]],
 [[56, 46, 46, 55], [55, 91, 127, 183]],
 [[74, 70, 73], [58, 95, 172]],
 ...
]
```

Stroke 1

Stroke 2

X coordinates

Y coordinates

Protocol Buffers

Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data.

person.proto ← Protobuf definition

```
message Person {
  optional string name = 1;
  optional string email = 2;
  repeated int32 lucky_numbers = 3;
}
```

protoc person.proto

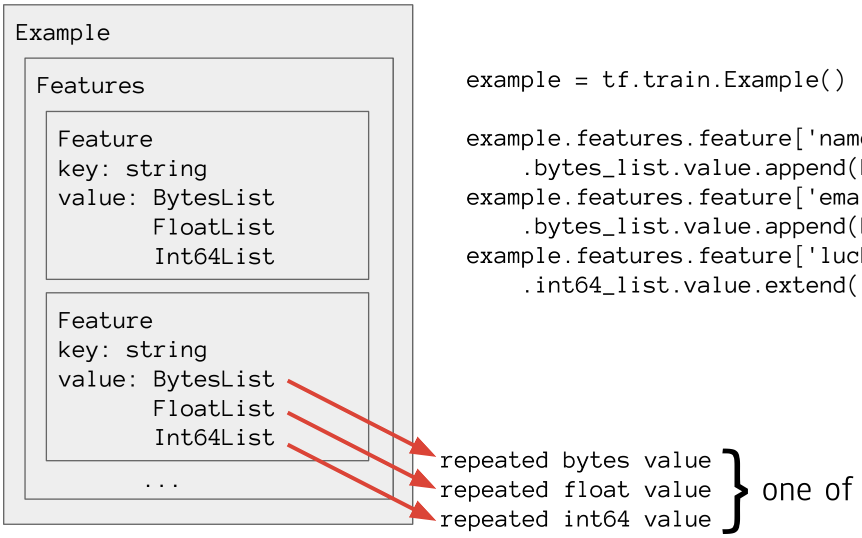
person_pb2.py ← Generated code

Import

```
from person_pb2 import Person

person = Person()
person.name = 'John Doe'
person.email = 'john.doe@gmail.com'
person.lucky_numbers.extend([13, 99])
person.SerializeToString()
```

```
b'\n\x08John Doe\x12\x12
john.doe@gmail.com\x18\r\x18c'
```

tf.train.Example

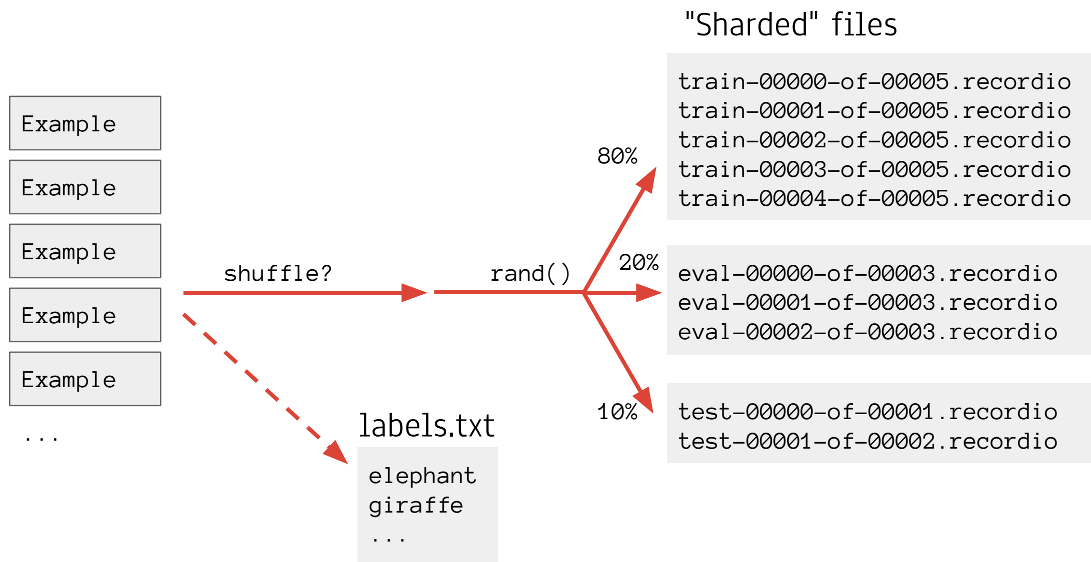



```
example = tf.train.Example()

example.features.feature['name']\
    .bytes_list.value.append(b'John Doe')
example.features.feature['email']\
    .bytes_list.value.append(b'john.doe@gmail.com')
example.features.feature['lucky_numbers']\
    .int64_list.value.extend([13, 99])
```


Converting QuickDraw Data

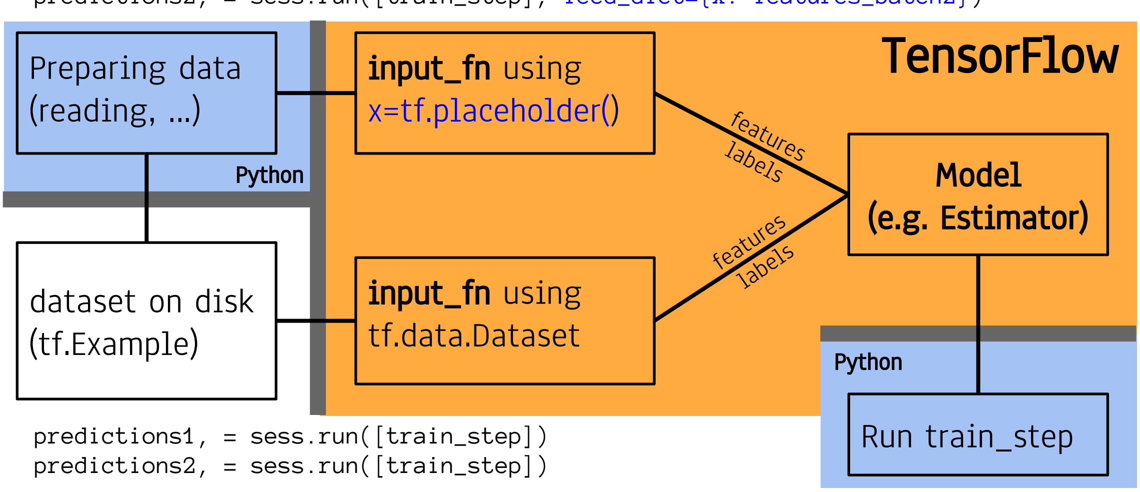
Source data		tf.train.Example
label : "monkey"	➪	.features.feature['label'] .**int64**_list.value == [2]
image : 	➪	.features.feature['img_64'] .**int64**_list.value == [0, 0, 0, ..., 255, ...]
⋮		⋮

Create Dataset



Connecting Data to the Graph

```
predictions1, = sess.run([train_step], feed_dict={x: features_batch1})
predictions2, = sess.run([train_step], feed_dict={x: features_batch2})
```




```
predictions1, = sess.run([train_step])
predictions2, = sess.run([train_step])
```

Read Data in Python

1. File Iterator

```
record = next(tf.python_io.tf_record_iterator(train_files[0]))
example = tf.train.Example.FromString(record)
```

2. Parse Example

```
img_64 = example.features.feature['img_64'].int64_list.value
img_64 = np.array(img_64).reshape((64, 64)) / 255.
label = example.features.feature['label'].int64_list.value[0]
```

3. Get features

Read Data in TensorFlow

tf.Tensor(dtype=tf.string)

```
def parse_example(serialized_example):
    features = tf.parse_single_example(serialized_example, feature_spec)
    label = features.pop('label')
    features['img_64'] = tf.cast(features['img_64'], tf.float32) / 255.
    return features, label
```

tf.Tensor

Dict[str, tf.Tensor]

```
def make_input_fn(files_pattern, batch_size=100):
    def input_fn():
        ds = tf.data.TFRecordDataset(tf.gfile.Glob(files_pattern))
        ds = ds.map(parse_example).batch(batch_size)
        ds = ds.shuffle(buffer_size=5*batch_size).repeat()
        features, labels = ds.make_one_shot_iterator().get_next()
        return features, labels
    return input_fn
```

Agenda

Today

In Practice ⚠

Low-Level Reps

Data

Estimator

Cloud

Q+A

Abstractions...

Canned

tf.estimator

tf.keras

tf.layers

Python Fronted

TF Distributed Execution Engine

"Low-Level"
TensorFlow

Canned Estimators

tf.estimator.BaselineClassifier

Learns to predict average value of each label.

tf.estimator.LinearClassifier

Train a linear model.

tf.estimator.DNNClassifier

Train a model of fully connected layers.

tf.estimator.DNNLinearCombinedClassifier

Also known as "wide'n'deep".

.
.
.

1. Specify input layer

```
feature_columns = [
  tf.feature_column.numeric_column(...),
    ...
]
```

2. Instantiate Estimator

```
classifier = LinearClassifier(
  feature_columns=...)
```

3. Train / Eval

```
classifier.train(
  input_fn=make_input_fn(...), steps=...)
```


Feature Spec vs. Feature Columns

```
        TFRecordDataset
features {
  feature {
    key: "countrycode"
    value {
      bytes_list {
        value: "AU"
      }
    }
  }
  feature {
    key: "img_64"
    value {
      int64_list {
        value: 0
        value: 0
...
```

tf.contrib.layers.create_feature_spec_for_parsing()

feature_spec

```
                                    input_fn() returns
(dict(img_64=
  tf.Tensor(shape=[batch_size, 64, 64], dtype=float)
), tf.Tensor(shape=[batch_size], dtype=tf.int64)
```

feature_columns

```
                                    First Layer of NN
tf.Tensor(shape=[batch_size, ...], dtype=float)
```

Estimator: .train(), .evaluate(), .predict()

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/estimator/estimator.py

```
classifier = tf.estimator.LinearClassifier(
  model_dir='/tmp/models/linear',
  run_config=run_config,
  feature_columns=feature_columns)
```

```
classifier.train(
  input_fn=input_fn, steps=steps, hooks=hooks)
```

```
classifier.evaluate(
  input_fn=input_fn, steps=steps, hooks=hooks)
```

```
classifier.predict(
  input_fn=input_fn, hooks=hooks)
```

- **Model directory** will contain graph definition, checkpoints, events, exports.
- **Run config** specifies *how* to train/eval : # of checkpoints written, session config, distributed strategy, ...
- New graph is created every time train(), evaluate(), predict() are called.
- train() will create new **checkpoints** in model_dir (recording variable values).
- evaluate() will load model from **checkpoints**.
- Use hooks to access graph/session.

Custom Estimators

```
estimator = tf.estimator.Estimator(model_fn=model_fn)
```

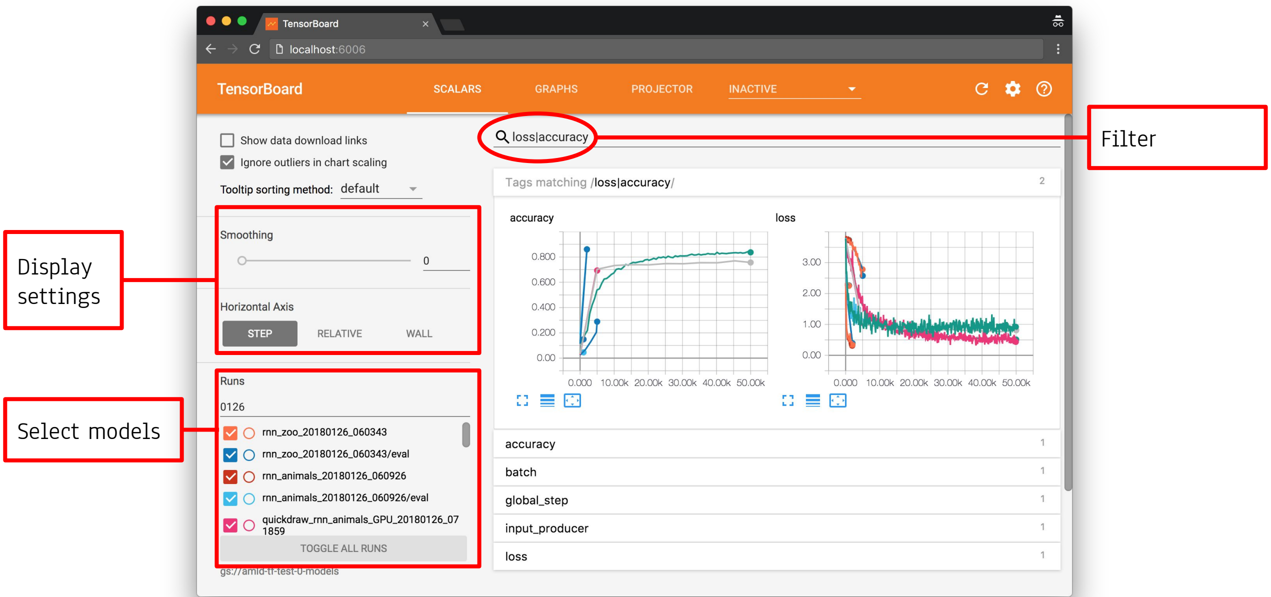
model_fn signature

Params
- features : Dict of Tensors
- labels : Tensor
- mode : EVAL, TRAIN, PREDICT
- params : Dict of values

returns tf.estimator.EstimatorSpec
- loss : Tensor
- mode : EVAL, TRAIN, PREDICT
- predictions : Dict of Tensors
- export_outputs : Dict
- train_op : Operation
- eval_metric_ops : Dict of Operation

Typical model_fn body

- Compute logits
- TRAIN : compute loss, optimize loss (=train_op)
- EVAL : compute (streaming) metrics (->eval ops)
- **Implement complicated signature**



Running TensorBoard

Start TensorBoard

```
tensorboard --logdir=./models --host 0.0.0.0 --port 6006
```

Colab : Forward port

```
ngrok http 6006
```

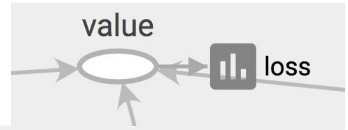
Navigate to TensorBoard

```
http://localhost:6006
```

Colab : Connect via ngrok

```
http://db1234567.ngrok.io
```

Summaries



Define summary operations in graph

```
tf.summary.scalar('loss', loss)
```

Stored in model directory

```
$MODEL_DIR/events.out.tfevents.*
$MODEL_DIR/eval/events.out.tfevents.*
```

Read summary information in Python

```
for e in tf.train.summary_iterator(events_eval_path):
    for v in e.summary.value:
        events[v.tag].append(v.simple_value)
```


Saved Model

"language-neutral, recoverable, hermetic serialization format"

- "Tags" (SERVING, TRAIN)
- The graph.
- Variables.
- Assets (e.g. vocabulary files).
- Signatures:
  - Key (can have multiple heads).
  - Inputs : string -> Tensor info
  - Outputs : string -> Tensor info
  - Method name (CLASSIFY_METHOD_NAME etc.)

As Saved Model - Simple

```
with graph.as_default():
  tf.saved_model.simple_save(sess, "/tmp/models/mnist",
                             inputs={"x": x}, outputs={"y": y})
```

More control:
SavedModelBuilder

- Tag : SERVING
- Serialized graph.
- All variables contained in graph.
- Signatures:
  - Key : DEFAULT_SERVING_SIGNATURE_DEF_KEY
  - Inputs : "x" -> x
  - Outputs : "y" -> y
  - Method name : PREDICT_METHOD_NAME

As Saved Model - Detailed

```
with tf.Graph().as_default(), tf.Session() as sess:
  x = tf.placeholder(tf.float32, shape=[None, 3])
  y = tf.nn.softmax(x)

  builder = tf.saved_model.builder.SavedModelBuilder(export_dir)
  builder.add_meta_graph_and_variables(
      sess,
      [tf.saved_model.tag_constants.SERVING],
      signature_def_map={
          tf.saved_model.signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY:
          tf.saved_model.signature_def_utils.build_signature_def(
              inputs={'x': tf.saved_model.utils.build_tensor_info(x)},
              outputs={'y': tf.saved_model.utils.build_tensor_info(y)},
              method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME
          )
      }
  builder.save()
```

Inspect Saved Model

```
!saved_model_cli show --dir $export_dir --all
```

```
MetaGraphDef with tag-set: 'serve' contains the following
SignatureDefs:

signature_def['serving_default']:
  The given SavedModel SignatureDef contains the following input(s):
    inputs['x'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 3)
        name: Placeholder:0
  The given SavedModel SignatureDef contains the following output(s):
    outputs['y'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 3)
        name: Softmax:0
  Method name is: tensorflow/serving/predict
```


Run a Saved Model

```
!saved_model_cli run --dir $export_dir --tag_set serve \
                     --signature_def serving_default \
                     --input_exprs 'x=[[99, 0, 0], [1, 1, -99]]'
```

```
Result for output key y:
[[1.  0.  0. ]
 [0.5 0.5 0. ]]
```

```
with tf.Graph().as_default(), tf.Session(graph=tf.Graph()) as sess:
  model = tf.saved_model.loader.load(sess, [tag_constants.SERVING], export_dir)
  signature = model.signature_def[signature_constants.DEFAULT_SERVING_SIGNATURE_DEF_KEY]
  x_name = signature.inputs["x"].name
  y_name = signature.outputs["y"].name
  print(sess.run([y_name], feed_dict={x_name: [[99, 0, 0], [1, 1, -99]]}))
```

```
[array([[1. , 0. , 0. ],
       [0.5, 0.5, 0. ]], dtype=float32)]
```

Agenda

Today

- In Practice ⚠
- Low-Level Reps
- Data
- Estimator
- Cloud
- Q+A

Move Code out of Colab

Command line interface

```
--model-dir=/tmp/models/run1 \
--n-classes=10 \
--train-files="${DS}/train-*" \
--eval-files="${DS}/eval-*"
```

train_spec

eval_spec

estimator "params"

Colab code

- make_input_fn()
- create_estimator()

```
tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

- Run locally : python -m trainer.task
- Run using Cloud ML : gcloud ml-engine jobs submit training

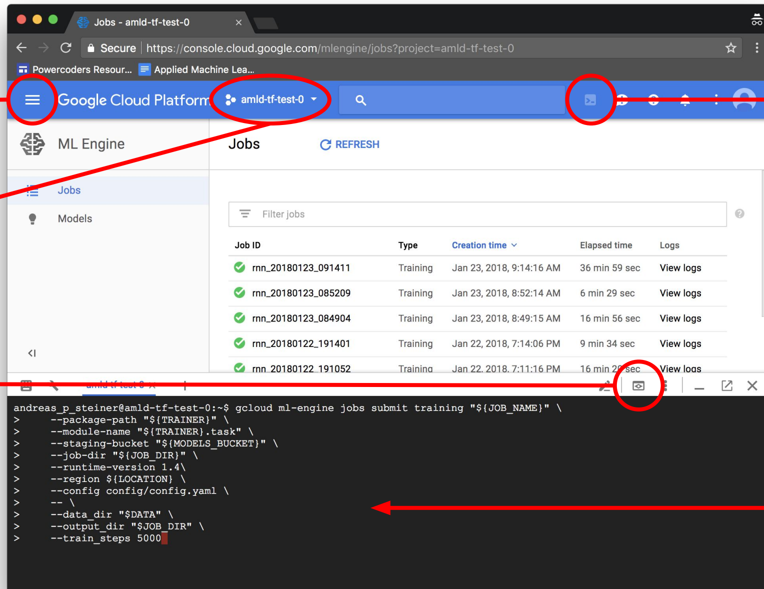
Main Menu

Project

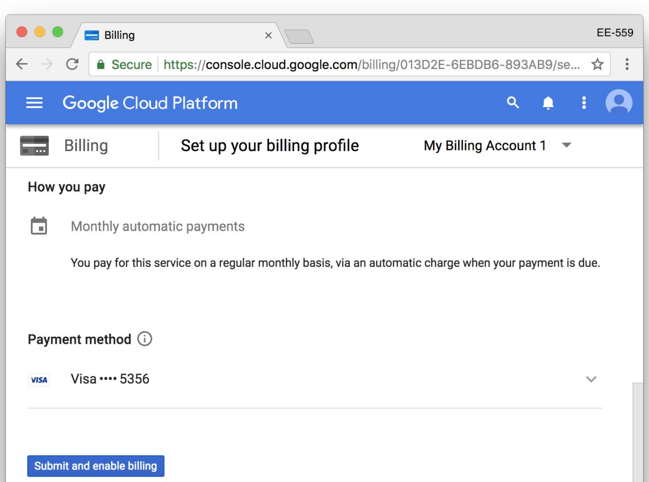
Tensorboard:

"Preview on port 8080..."

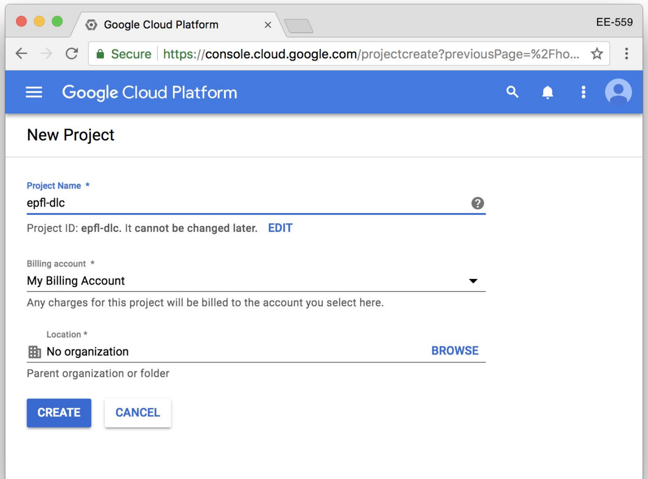
Cloud Shell



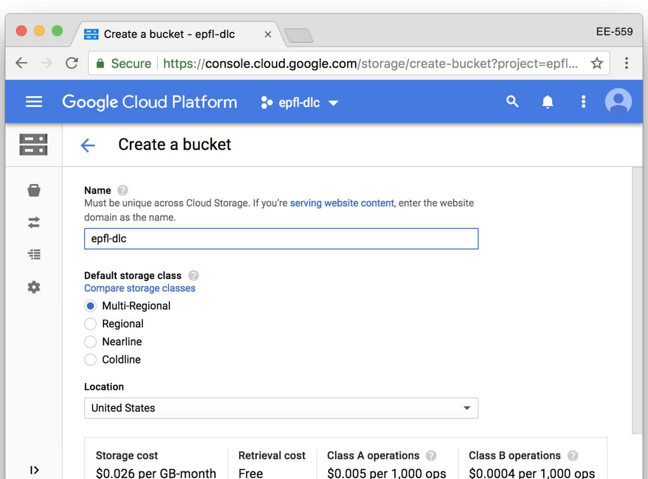
GCE : Billing



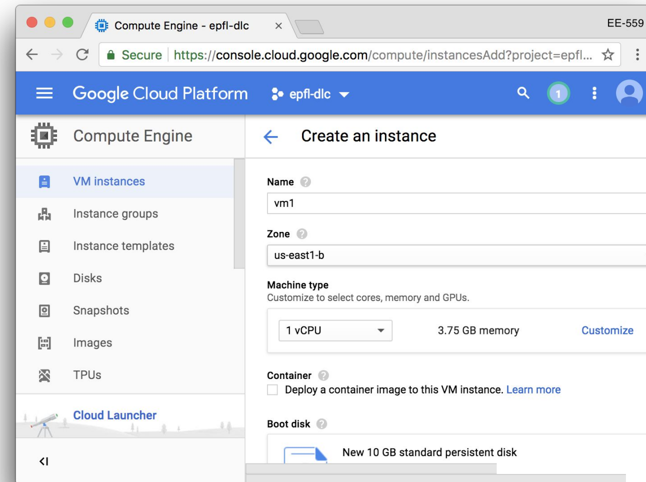
GCE : Create Project



GCE : Create Storage



GCE : Create VM



Cloud ML

Jupyter notebook

estimator = ...
estimator.train()

Saving checkpoints into ...

estimator.eval()

accuracy=0.1284

Cloud ML

master

parameter server

eval worker

train worker

model directory

model.ckpt-N.*
events.out.tfevents.*
eval/events.out.tfevents.*
export/

GCE : Jobs on Cloud ML

```
GCS_MODEL_DIR='gs://amld-models'
JOB_NAME='quickdraw_cnn_1'
GCS_JOB_DIR="${GCS_MODEL_DIR}/models/${JOB_NAME}"
DATASET='gs://amld-datasets/zoo_img'
gcloud ml-engine jobs submit training $JOB_NAME \
    --region=us-east1 \
    --scale-tier=standard-1 \
    --runtime-version=1.7 \
    --job-dir=$GCS_JOB_DIR \
    --module-name=trainer_cnn.task --package-path=trainer_cnn/ \
    -- \
    --n-classes=10 \
    --train-files="${DATASET}/train-*" \
    --region=us-central1 \
    --eval-files="${DATASET}/eval-*"
```

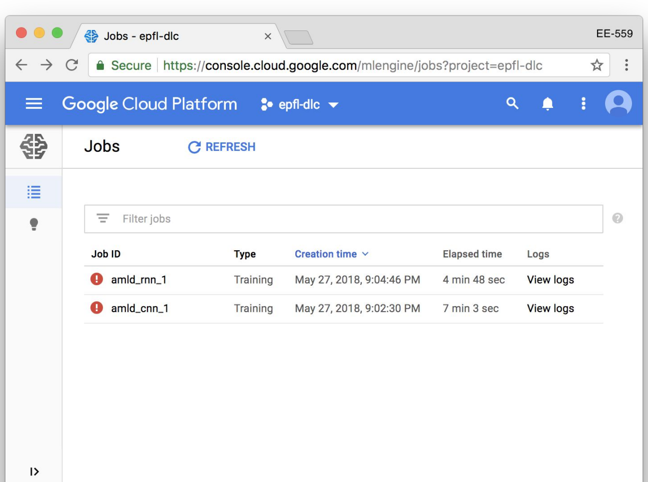
Writable

Readable

Costs $2.9/hour

Binary parameters

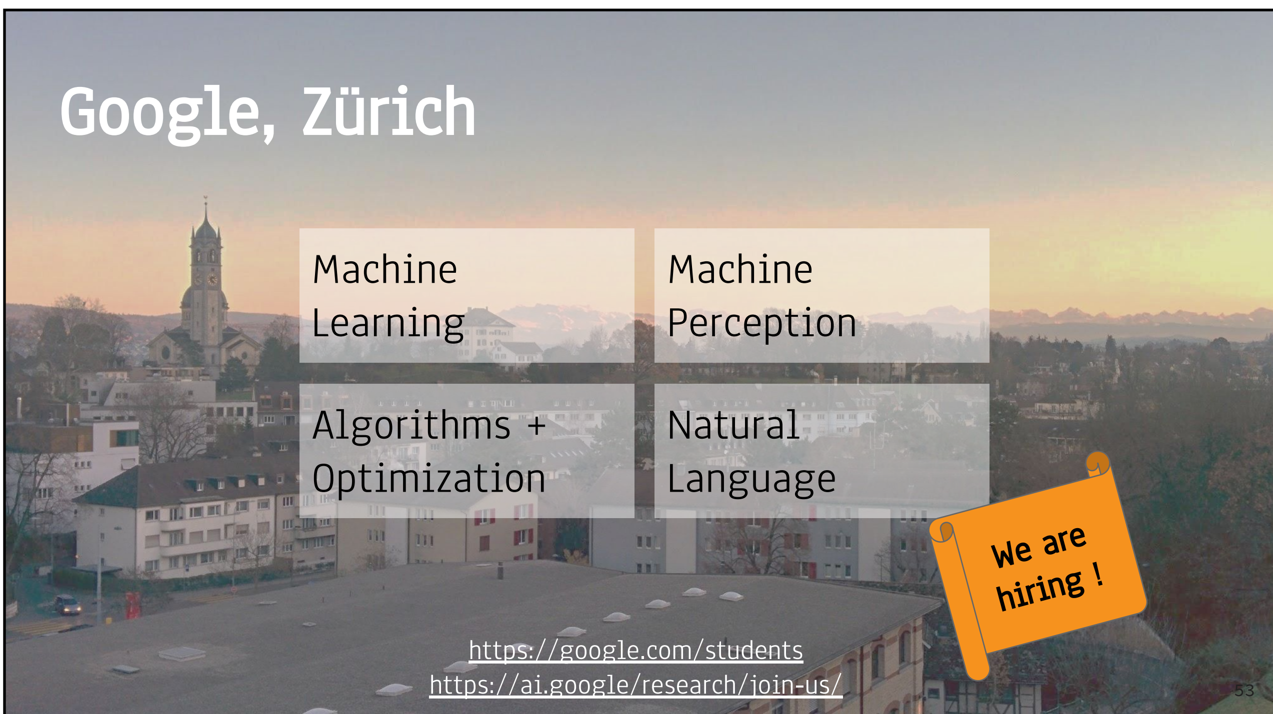
GCE : Jobs on Cloud ML



Agenda

Today

In Practice ⚠	Estimator
Low-Level Reps	
Data	Cloud
	Q+A ⇦



Google, Zürich

Machine Learning	Machine Perception
Algorithms + Optimization	Natural Language

We are hiring !

https://google.com/students
https://ai.google/research/join-us/

References

(Baylor 2017) TFX: A TensorFlow-Based Production-Scale Machine Learning Platform
http://www.kdd.org/kdd2017/papers/view/tfx-a-tensorflow-based-production-scale-machine-learning-platform

(Sculley 2014) Machine Learning: The High-Interest Credit Card of Technical Debt
https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43146.pdf

(Sculley 2015) Hidden Technical Debt in Machine Learning Systems
https://papers.nips.cc/paper/5656-hidden-technical-debt-in-machine-learning-systems.pdf

Links 1/2

Rules of ML
https://developers.google.com/machine-learning/rules-of-ml/

Tensorflow Extended (TFX) (TensorFlow Dev Summit 2018)
https://youtu.be/vdG7uKQ2eKk

Protocol Buffers Tutorial (Python)
https://developers.google.com/protocol-buffers/docs/pythontutorial

tf.data.Dataset
https://www.tensorflow.org/api_docs/python/tf/data/Dataset
https://www.tensorflow.org/programmers_guide/datasets

Canned Estimators
https://www.tensorflow.org/get_started/premade_estimators
https://www.tensorflow.org/versions/master/api_docs/python/tf/estimator

Feature Columns
https://www.tensorflow.org/get_started/feature_columns
https://www.tensorflow.org/programmers_guide/low_level_intro#feature_columns

Links 2/2

Estimator's RunConfig
https://www.tensorflow.org/api_docs/python/tf/estimator/RunConfig

TensorBoard, Summaries
https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard
https://www.tensorflow.org/api_guides/python/summary

Saved Model
https://www.tensorflow.org/programmers_guide/saved_model

Training on Cloud
https://cloud.google.com/ml-engine/docs/tensorflow/training-overview

Cloud Pricing
https://cloud.google.com/ml-engine/docs/pricing

Cloud TensorFlow Tutorial
https://cloud.google.com/blog/big-data/2018/02/easy-distributed-training-with-tensorflow-using-tfestimatortrain-and-evaluate-on-cloud-ml-engine