Google Cloud

Building Machine Learning models with TensorFlow

Data Engineering on Google Cloud Platform

Notes:

30 slides + 4 labs: 2.5 hours

Agenda

What Is TensorFlow? + Lab

TensorFlow for Machine Learning + Lab

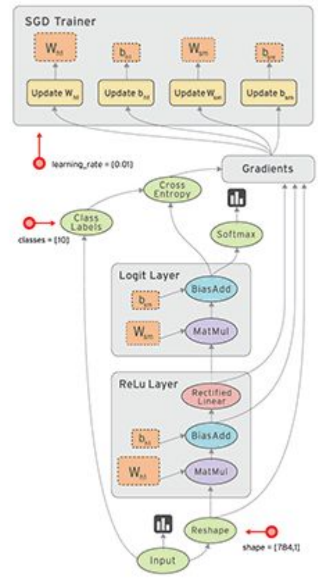
Gaining more flexibility + Lab

The experiment framework + Lab

TensorFlow is an open-source high-performance library for numeric computation

- Portable across GPUs, CPUs, mobile ...
- Developed at Google
- Uses data flow graphs: neural network training and evaluation can be represented as data flow graphs

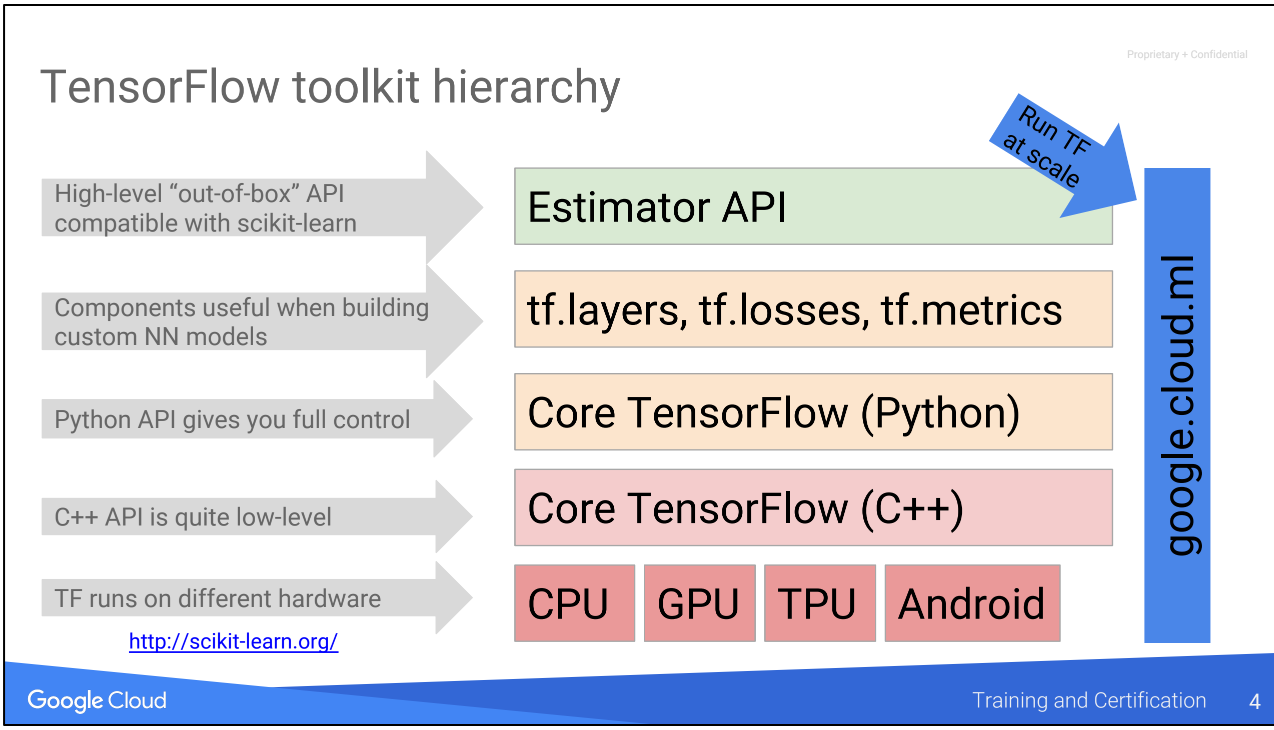
http://www.tensorflow.org/



Notes:

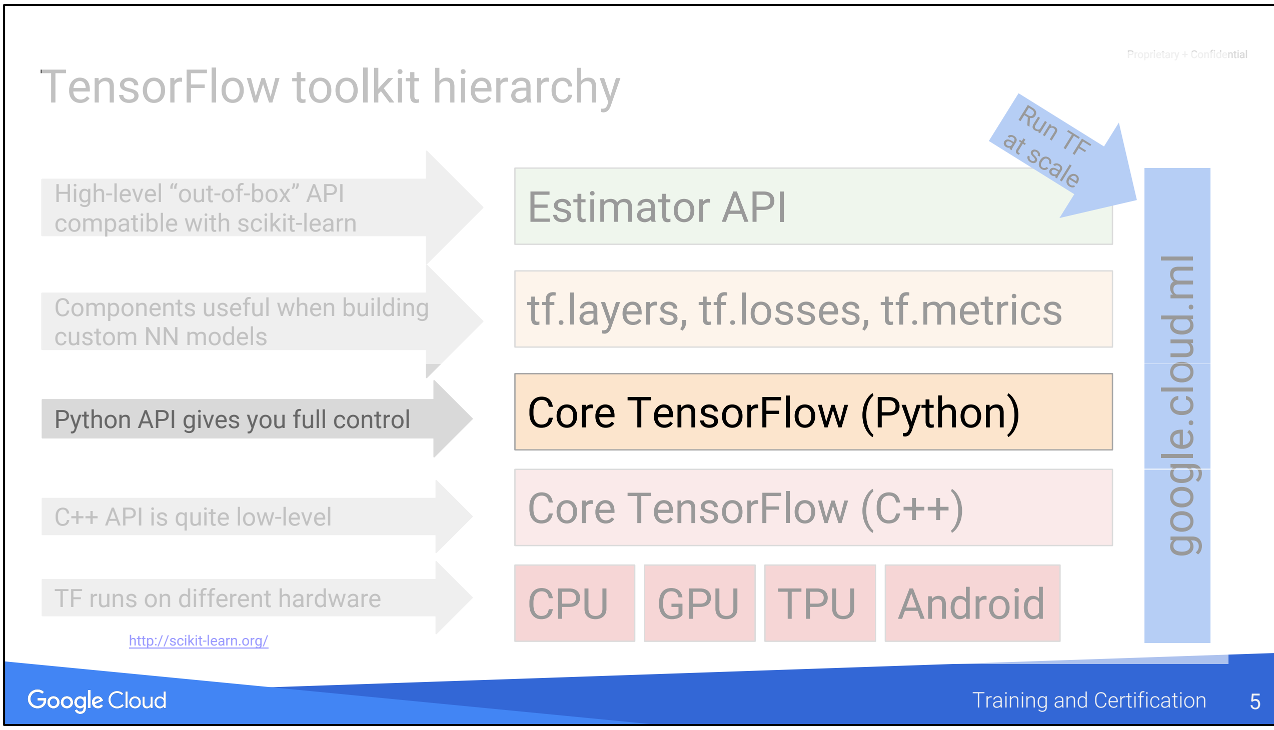
Operates over tensors: n-dimensional arrays.

Using a flow graph: data flow computation framework.



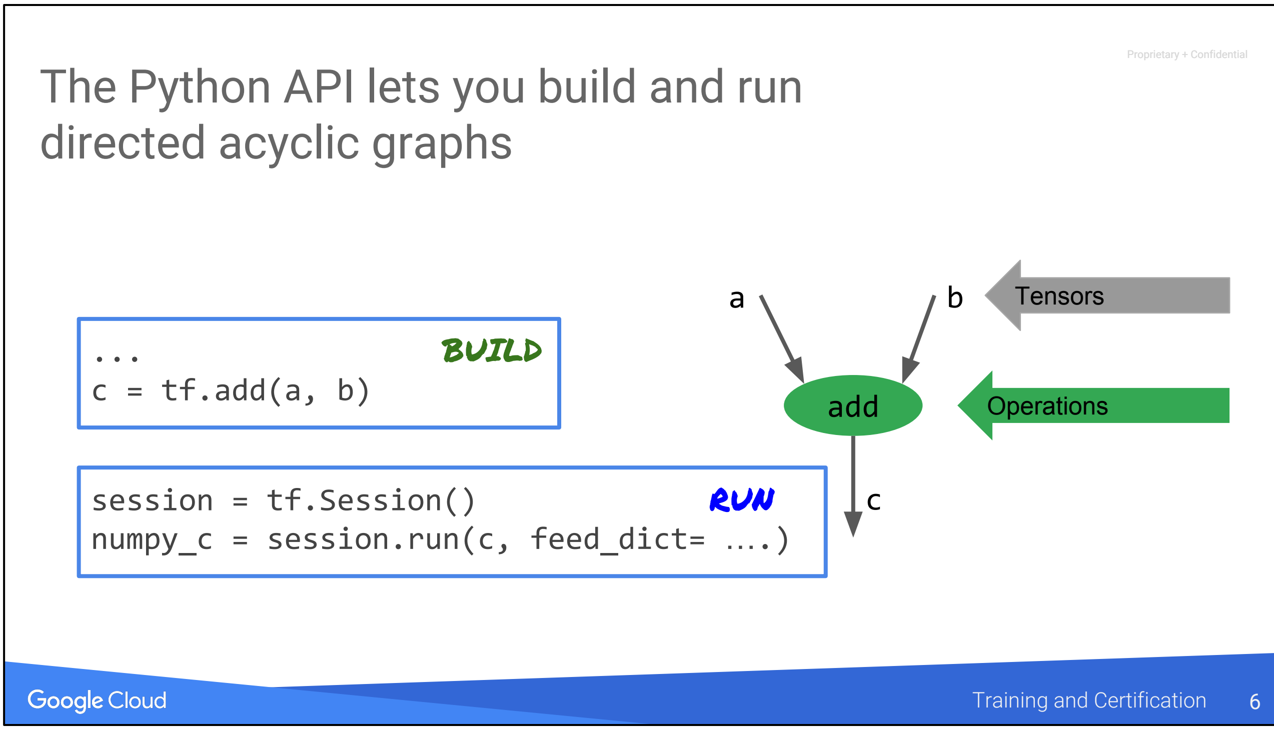
Notes:

Akin to how Java programs run on many different types of hardware because of the JVM. TensorFlow C++ plays the role of the JVM here, providing hardware instruction sets.



Notes:

Let's look at what the Python API does.



Notes:

Programming TensorFlow involves programming a DAG. Create graph, then run it.

The graph definition is separate from the training loop because this is a lazy evaluation model. It minimizes the Python/C++ context switches and enables the computation to be very efficient. Conceptually, this is like writing a program, compiling it, then running it on some data. Don't take the analogy too far, though.There is no explicit compile phase here.

Note that c is not the actual values -- you have to evaluate c in the context of a TensorFlow session to get a numpy array of values ('c').

TensorFlow does lazy evaluation; you need to run the graph to get results

NUMPY

```
a = np.array([5, 3, 8])
b = np.array([3, -1, 2])
c = np.add(a, b)
print c
```

```
[ 8  2 10]
```

TENSORFLOW

```
a = tf.constant([5, 3, 8])
b = tf.constant([3, -1, 2])
c = tf.add(a, b)
print c
```

BUILD

```
Tensor("Add_7:0", shape=(3,), dtype=int32)
```

```
with tf.Session() as sess:
  result = sess.run(c)
  print result
```

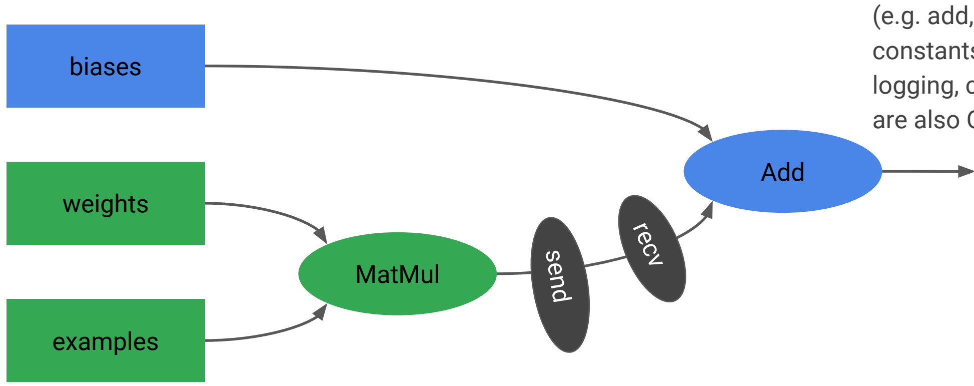
RUN

```
[ 8  2 10]
```


Notes:

Unlike with numpy, in TensorFlow c is not the actual values. Instead, C is a tensor -- you have to evaluate c in the context of a TensorFlow session to get a numpy array of values (‘result’). What gets printed out in the first box is the “debug” output of the Tensor class. It includes the shape (3,), data type (int32), and a system-assigned unique name (Add_7:0).

Graphs can be processed, compiled, remotely executed, assigned to devices



Besides computation (e.g. add, matmul), constants, variables, logging, control flow are also Ops

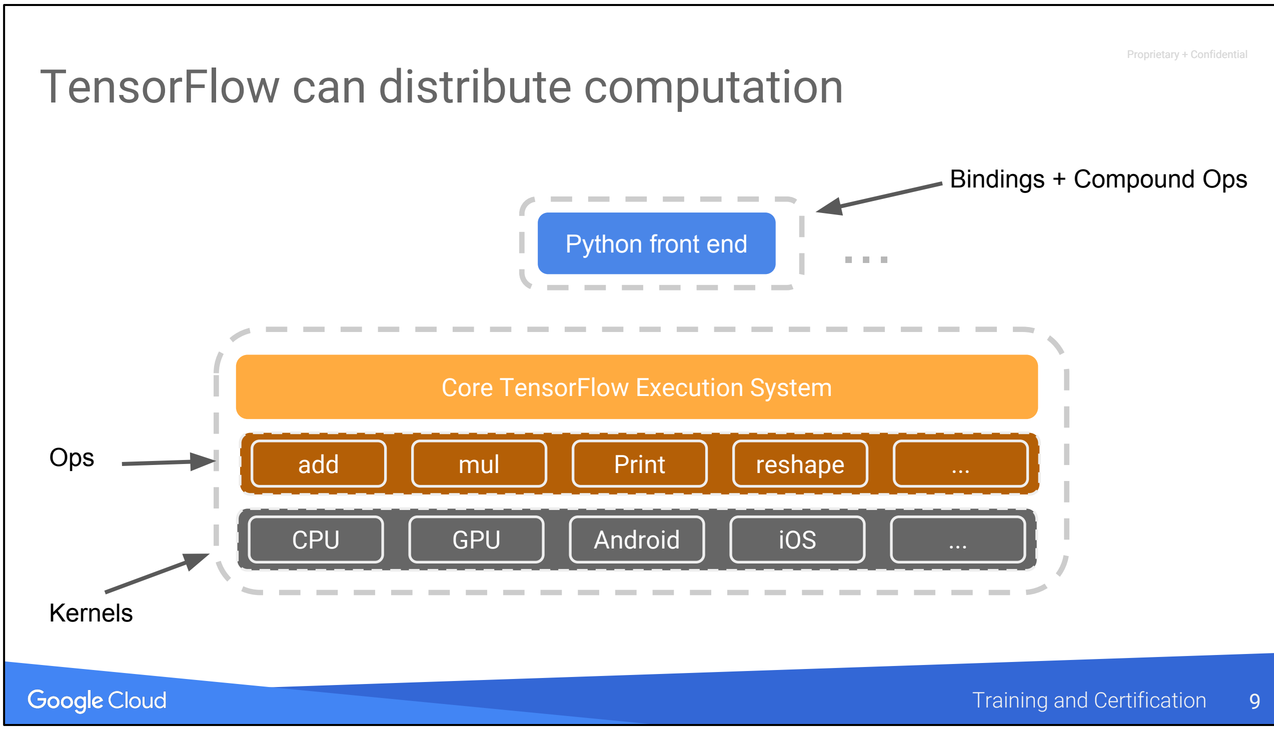
Notes:

Example of each mentioned use-case:

Processed -- add quantization or data types, add debug nodes; create summaries to write values out so that Tensorboard can read ...

Compiled -- fuse ops to improve performance. For example, two consecutive add nodes can be fused into a single one.

Remotely executed, assigned to devices -- **note colors**, several parts of the graph can be on different devices, doesn’t matter whether GPU or several computers. Automatically insert send/recv nodes.



Notes:

One key benefit of this model -- to be able to distribute computation across many machines, and many types of machines. No need to assign a Print op to a GPU.

Lab: Serverless Machine Learning

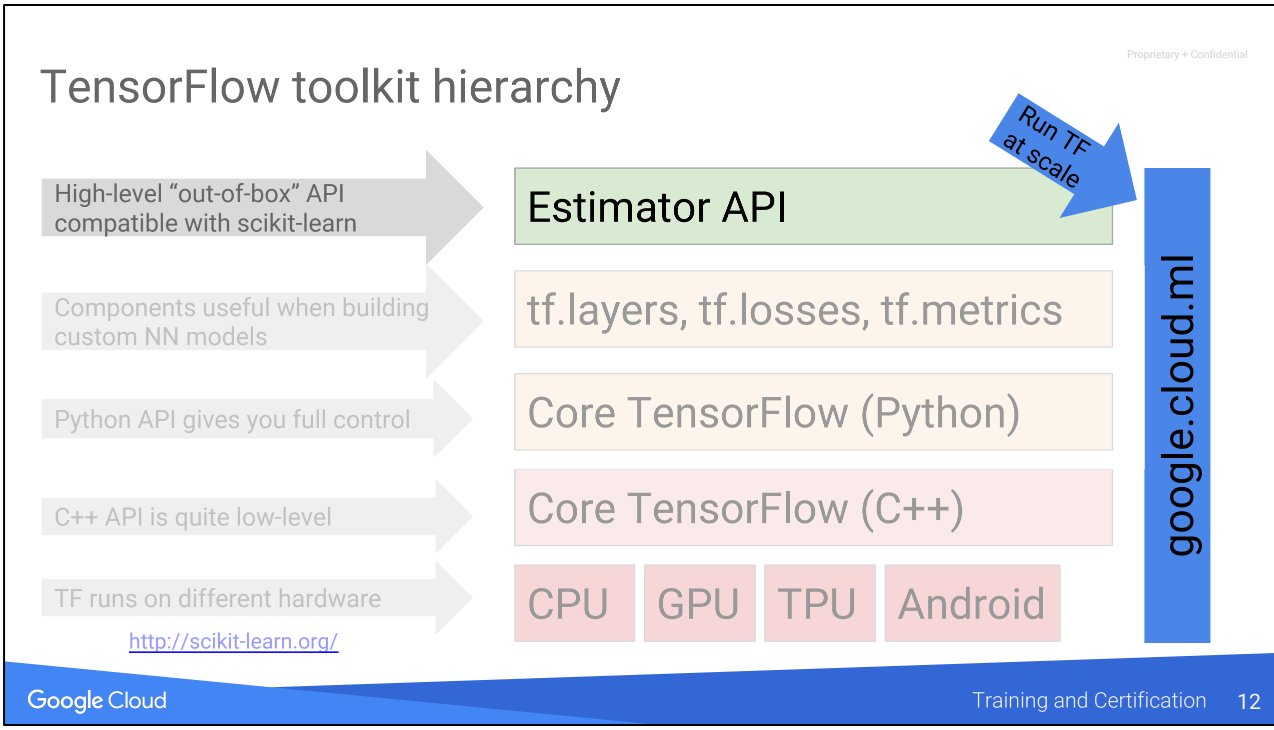
Part 2a. Getting Started with TensorFlow

In this lab, you will learn how:

- The TensorFlow Python API works
  - Building a graph
  - Running a graph
  - Feeding values into a graph
  - Find area of a triangle using TensorFlow

Agenda

TensorFlow for Machine Learning + Lab

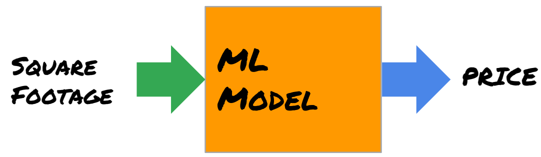


Notes:

This is the level at which we will mostly operate. The TensorFlow libraries are what will write the lower-level code.

Working with Estimator API

Set up machine learning model
1. Regression or classification?
2. What is the label?
3. What are the features?



Carry out ML steps
1. Train the model
2. Evaluate the model
3. Predict with the model

Notes:

Two parts to it. One is static -- how to set the ML problem up.

The second is the ML steps you carry out.

Imagine that you want to create a ML model to predict cost of a house given the sq footage. Answer the first three questions.

The structure of an Estimator API ML model

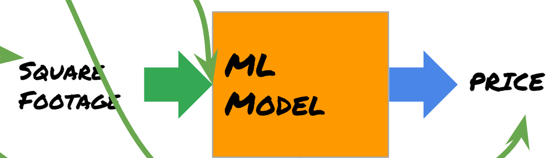
```
import tensorflow as tf

#Define input feature columns
feature_columns = [tf.contrib.layers.real_valued_column("sq_footage")]

#Instantiate Linear Regression Model
estimator = tf.contrib.learn.LinearRegressor(
    feature_columns=feature_columns)

#Train
def input_fn_train():
    feature_data = {"sq_footage": tf.constant([1000,2000])}
    label_data = tf.constant([100000,200000])
    return feature_data, label_data
estimator.fit(input_fn=input_fn_train, steps=100)

#Predict
def input_fn_pred():
    feature_data = {"sq_footage": tf.constant([1500])}
    return feature_data
list(estimator.predict(input_fn=input_fn_pred))
```



Notes:

Conceptually, square footage shows up twice. Once as the placeholder (“feature_column”) and next as an input feed (“input_fn”).

Steps to define an Estimator API model


```
import tensorflow as tf

#Define input feature columns
feature_columns = [tf.contrib.layers.real_valued_column("sq_footage")]

#Instantiate Linear Regression Model
estimator = tf.contrib.learn.LinearRegressor(
    feature_columns=feature_columns)

#Train
def input_fn_train():
    feature_data = {"sq_footage": tf.constant([1000,2000])}
    label_data = tf.constant([100000,200000])
    return feature_data, label_data
estimator.fit(input_fn=input_fn_train, steps=100)

#Predict
def input_fn_pred():
    feature_data = {"sq_footage": tf.constant([1500])}
    return feature_data
list(estimator.predict(input_fn=input_fn_pred))
```

1. SET UP FEATURE COLUMN
(OTHER TYPES EXIST: WE'LL LOOK AT THEM IN CHAPTER 4)

2. CREATE A MODEL, PASSING
IN THE FEATURE COLUMNS

3. WRITE INPUT_FN (RETURNS
FEATURES, LABELS)
FEATURES IS A DICT

Steps to do Machine Learning with model


```
import tensorflow as tf

#Define input feature columns
feature_columns = [tf.contrib.layers.real_valued_column("sq_footage")]

#Instantiate Linear Regression Model
estimator = tf.contrib.learn.LinearRegressor(
    feature_columns=feature_columns)

#Train
def input_fn_train():
    feature_data = {"sq_footage": tf.constant([1000,2000])}
    label_data = tf.constant([100000,200000])
    return feature_data, label_data
estimator.fit(input_fn=input_fn_train, steps=100)

#Predict
def input_fn_pred():
    feature_data = {"sq_footage": tf.constant([1500])}
    return feature_data
list(estimator.predict(input_fn=input_fn_pred))
```

4. Train the model

5. Use trained model to predict

Notes:

estimator.predict() returns a generator function, not the actual values. You have to iterate through it to get the values. Or use list() to get all the values in one go.

Beyond linear regression with Estimators

- Deep neural network

```
model = DNNRegressor(feature_columns=[...],
                     hidden_units=[128, 64, 32])
```

- Classification

```
model = LinearClassifier(feature_columns=[...])
model = DNNClassifier(feature_columns=[...], hidden_units=[...])
```


Notes:

The hourglass indicates that we'll cover this later in the course.

https://pixabay.com/en/hourglass-sandglass-patience-time-297765/ (cc0)

Lab: Serverless Machine Learning

Part 2b. Machine Learning with tf.learn

In this lab, you will learn how to:

- Train from data in a Pandas dataframe
- Implement a Linear Regression model in TensorFlow
  - Train the model
  - Evaluate the model
  - Predict with the model
- Repeat with a Deep Neural Network model in TensorFlow

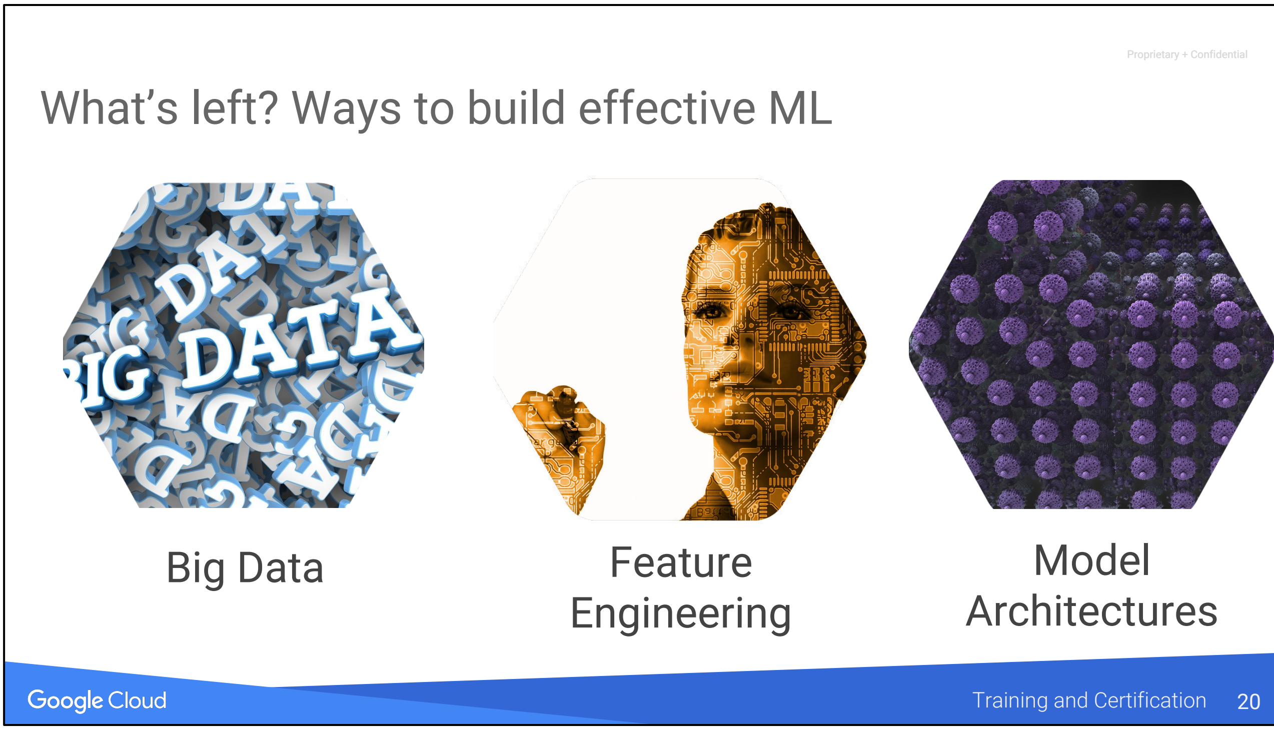
Notes:

It’s easy to ML --- just write 4 lines of Python code, throw some data into the hopper and voila!
But wait … the ML model doesn’t perform that well. More effort is needed.

If you are curious about why the ML model is performing so poorly: the problem is nonlinear in the inputs, but we are not scaling the inputs, and so the optimizer is not able to find the optimal weights. Also, the form of the solution requires some feature engineering on the raw inputs. Some preprocessing and feature engineering will enable us to get to better performance -- it’s important to realize that if you don’t do the necessary legwork, just making the model more complex is not going to work.
(the .csv files have on the order of 7700 samples, which should be enough for the models we are trying, so it’s not a case of not having enough data).

Agenda

Gaining more flexibility + Lab



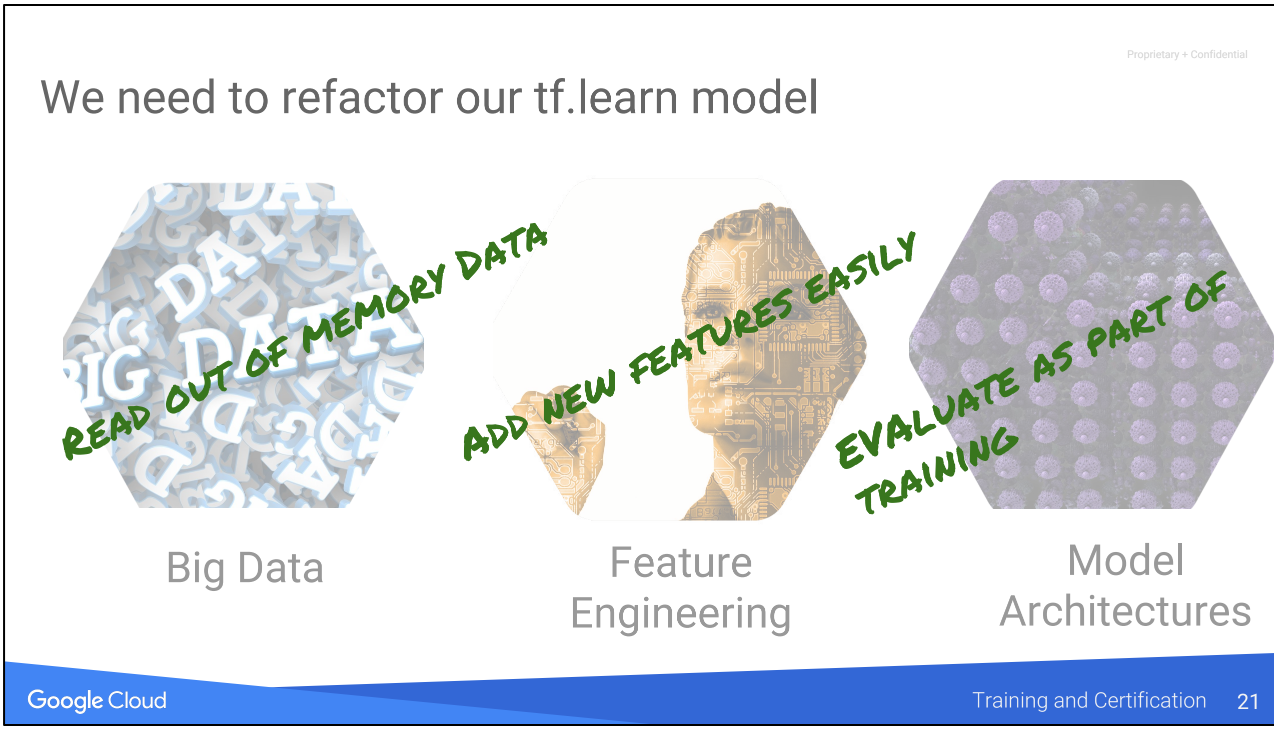
Notes:

https://pixabay.com/en/large-data-dataset-word-895563/ (cc0)
https://pixabay.com/en/fractal-complexity-render-3d-1232494/ (cc0)
https://pixabay.com/en/robot-artificial-intelligence-woman-507811/ (cc0)

Now that you know *how* to build ML, let's learn how to do it well in the rest of the course.

Ordered from easiest to most difficult.

Before we can do that though, we need to refactor our simple TensorFlow model.



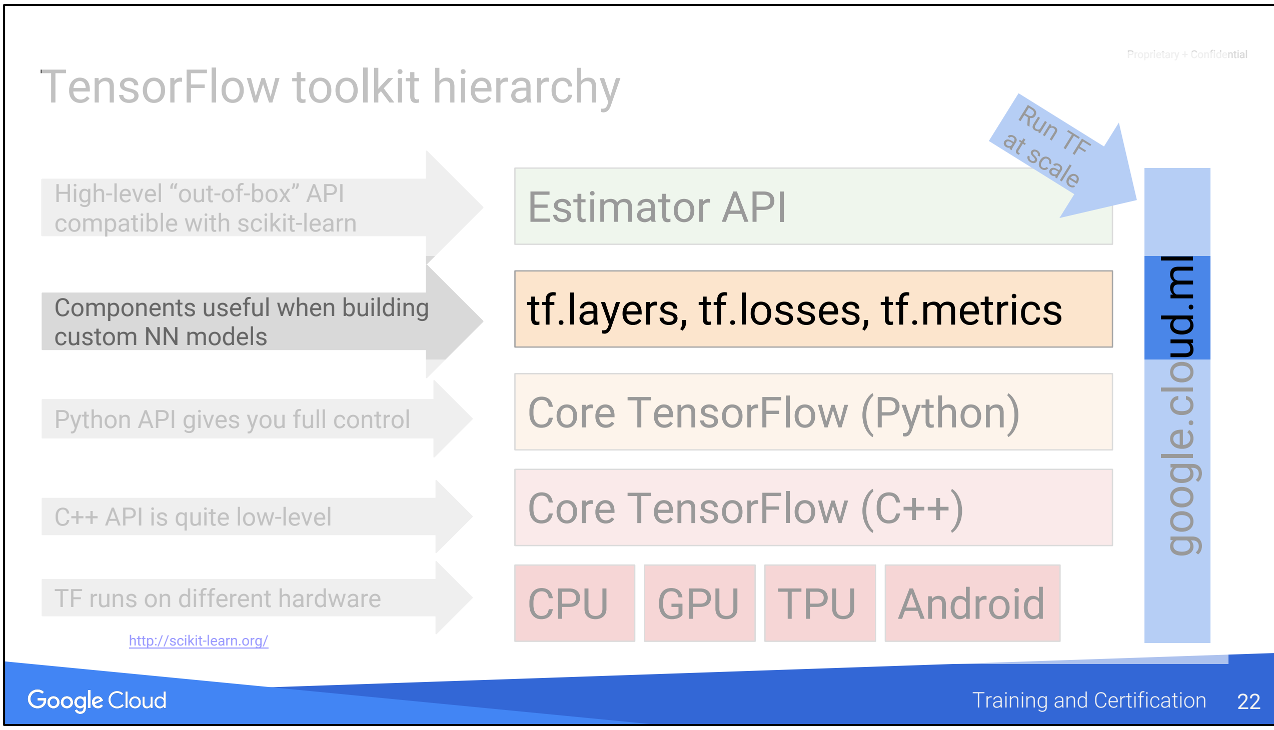
Notes:

Refactor == improve design of model without adding any new capability.
That’s what we are going to do in the next two labs.

To handle big data, we need to be able to read out-of-memory datasets.

To do feature engineering, we need to be able add new feature columns easily.

To try out new model architectures, we have to move evaluation as a bonafide part of training, so that we can test other architectures systematically.



Notes:

We'll use the next lower level down in refactoring.

Reading data in batches

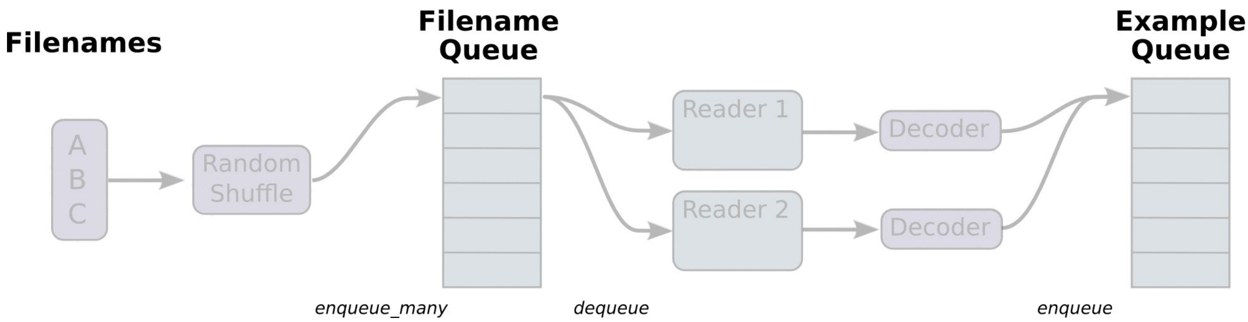


Image from https://www.tensorflow.org/programmers_guide/reading_data

Notes:

One of the advantages of reading data in batches is to use “parameter servers”. Each batch of data can be one part of a gradient descent and can be executed in parallel. This makes the searches much faster.

Read CSV file(s) num_epochs times

```
CSV_COLUMNS = ['fare_amount', 'pickuplon','pickuplat',..., 'key']
LABEL_COLUMN = 'fare_amount'
DEFAULTS = [[0.0], [-74.0], [40.0], [-74.0], [40.7], [1.0], ['nokey']]

def input_fn():
    input_file_names = tf.train.match_filenames_once(filename)
    filename_queue = tf.train.string_input_producer(
        input_file_names, num_epochs=num_epochs, shuffle=True)

    reader = tf.TextLineReader()
    _, value = reader.read_up_to(filename_queue, num_records=batch_size)

    value_column = tf.expand_dims(value, -1)
    columns = tf.decode_csv(value_column, record_defaults=DEFAULTS)
    features = dict(zip(CSV_COLUMNS, columns))
    label = features.pop(LABEL_COLUMN)
    return features, label
```

WILD-CARD MATCH

NUM_EPOCHS TIMES

Notes:

Wild-card matches are particularly useful when it comes to reading sharded files such as the output of Spark, etc.

We cycle through the files num_epochs times.

Read local/GCS CSV files in batches

```
CSV_COLUMNS = ['fare_amount', 'pickuplon','pickuplat',..., 'key']
LABEL_COLUMN = 'fare_amount'
DEFAULTS = [[0.0], [-74.0], [40.0], [-74.0], [40.7], [1.0], ['nokey']]

def input_fn():
    input_file_names = tf.train.match_filenames_once(filename)
    filename_queue = tf.train.string_input_producer(
        input_file_names, num_epochs=num_epochs, shuffle=True)

    reader = tf.TextLineReader()
    _, value = reader.read_up_to(filename_queue, num_records=batch_size)

    value_column = tf.expand_dims(value, -1)
    columns = tf.decode_csv(value_column, record_defaults=DEFAULTS)
    features = dict(zip(CSV_COLUMNS, columns))
    label = features.pop(LABEL_COLUMN)
    return features, label
```

TEXT LINE READER READS FROM GCS ALSO

EACH READ IS OF BATCH_SIZE LINES

Notes:

Wild-card matches are particularly useful when it comes to reading sharded files such as the output of Spark, etc.

We cycle through the files num_epochs times.

Return a dict of features and label

```
CSV_COLUMNS = ['fare_amount', 'pickuplon','pickuplat',..., 'key']
LABEL_COLUMN = 'fare_amount'
DEFAULTS = [[0.0], [-74.0], [40.0], [-74.0], [40.7], [1.0], ['nokey']]

def input_fn():
    input_file_names = tf.train.match_filenames_once(filename)
    filename_queue = tf.train.string_input_producer(
        input_file_names, num_epochs=num_epochs, shuffle=True)

    reader = tf.TextLineReader()
    _, value = reader.read_up_to(filename_queue, num_records=batch_size)

    value_column = tf.expand_dims(value, -1)
    columns = tf.decode_csv(value_column, record_defaults=DEFAULTS)
    features = dict(zip(CSV_COLUMNS, columns))
    label = features.pop(LABEL_COLUMN)
    return features, label
```

DECODE CSV
DICT OF FEATURES
LABEL, FROM CSV

Notes:

The headers are defined above.

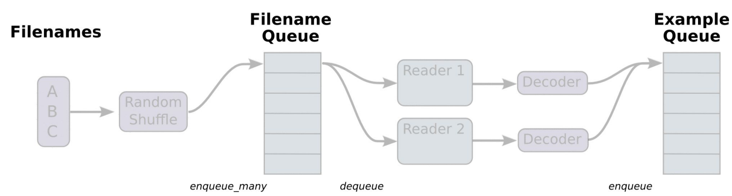
Lab: Serverless Machine Learning

Part 2c. TensorFlow on Big Data

In this lab, you will learn how to refactor the tf.learn model to:

- Read from a potentially large file in batches
- Do a wildcard match on filenames
- Break the one-to-one relationship between inputs and features

These two refactorings will help us add Big Data and Feature Engineering capability to our tf.learn model



Agenda

The experiment framework + Lab

We need to make our ML pipeline more robust

In our tf.learn examples so far, we:

1. ran the training_op for num_steps or num_epochs iterations
2. saved checkpoints during training
3. Used final checkpoint as model

For realistic, real-world ML models, we need to:

1. Use a fault-tolerant distributed training framework
2. Choose model based on validation dataset
3. Monitor training, especially if it will take days
4. Resume training if necessary

Notes:

https://pixabay.com/en/hang-out-plush-toys-kermit-1521663/ (cc0)
https://pixabay.com/en/london-england-hdr-boats-ships-123778/ (cc0)

Our tf.learn examples have been a toy and is okay if the only thing we are going to be handling are plush-toys. We can’t just hang a clothesline and call it a bridge. In the real world, we need to make things more robust.

Monitoring: Experiment will export summaries, so that we can see them in TensorBoard.

Use experiment for distributed training

```
def experiment_fn(output_dir):
    return tflearn.Experiment(
        tflearn.LinearRegressor(feature_columns=feature_cols,
model_dir=output_dir),
        train_input_fn=get_train(),
        eval_input_fn=get_valid(),
        eval_metrics={
            'rmse': tflearn.MetricSpec(
                metric_fn=metrics.streaming_root_mean_squared_error
            )
        }
    )

learn_runner.run(experiment_fn, 'taxi_trained')
```

MODEL AS BEFORE
OUTPUT CAN BE GCS

TWO INPUT FUNCTIONS
TRAIN, EVAL

EVALUATION METRIC

RUN EXPERIMENT

Proprietary + Confidential

Change logging level from WARN

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

```
INFO:tensorflow:Transforming feature_column _RealValuedColumn(column_nam
alue=None, dtype=tf.float32)
INFO:tensorflow:Create CheckpointSaver
INFO:tensorflow:Step 1: loss = 218.036
INFO:tensorflow:Step 101: loss = 89.9517
INFO:tensorflow:Step 201: loss = 89.9487
INFO:tensorflow:Saving checkpoints for 300 into taxi_model/model.ckpt.
INFO:tensorflow:Step 301: loss = 89.9468
INFO:tensorflow:Step 401: loss = 89.9453
INFO:tensorflow:Step 501: loss = 89.944
INFO:tensorflow:Saving checkpoints for 600 into taxi_model/model.ckpt.
INFO:tensorflow:Step 601: loss = 89.9429
INFO:tensorflow:Step 701: loss = 89.9419
INFO:tensorflow:Step 801: loss = 89.941
INFO:tensorflow:Saving checkpoints for 900 into taxi_model/model.ckpt.
INFO:tensorflow:Step 901: loss = 89.9402
```

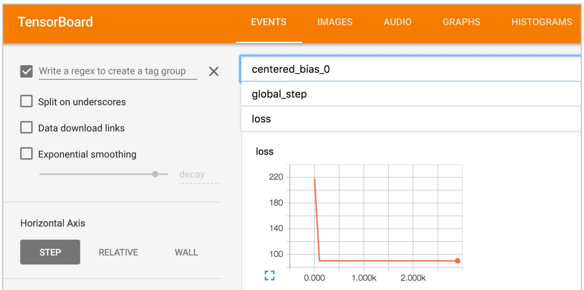

Notes:

The default is WARN; change it to INFO to see logs as TF trains.

The levels are DEBUG, INFO, WARN, ERROR, and FATAL.

Proprietary + Confidential

To monitor training, use TensorBoard



Notes:

Events at top left shows “loss”.

Graphs at bottom right shows the linear model graph as built by TensorFlow.

Point TensorBoard at model output directory.

Lab: Serverless Machine Learning

Part 2d. Refactor for distributed training and monitoring (optional)

In this lab, you will learn how to:

- Use the Experiment class
- Monitor training using TensorBoard

Resources

tf.contrib.learn API	https://www.tensorflow.org/versions/master/api_docs/python/contrib.learn.html
TensorFlow (all)	https://www.tensorflow.org/api_docs/python/index.html
Understanding neural networks with TensorFlow playground	https://cloud.google.com/blog/big-data/2016/07/understanding-neural-networks-with-tensorflow-playground
TensorFlow examples	https://github.com/aymericdamien/TensorFlow-Examples
TensorFlow MNIST	https://www.youtube.com/watch?v=vq2nnJ4g6N0&t=3686s
Another TF Datalab	https://github.com/kazunori279/TensorFlow-Intro/


cloud.google.com
Images by Connie Zhou