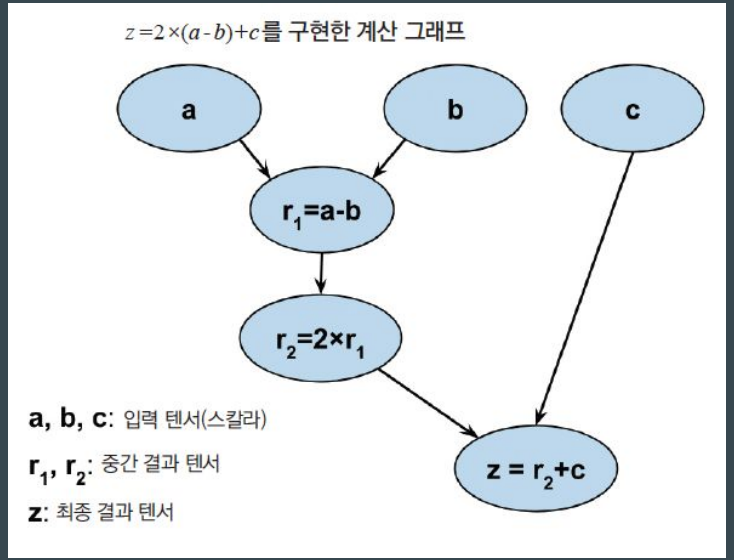
Keras in TensorFlow 2.0

박해선 - ML GDE
haesun.park@tensorflow.blog

slide: bit.ly/keras-in-tf20

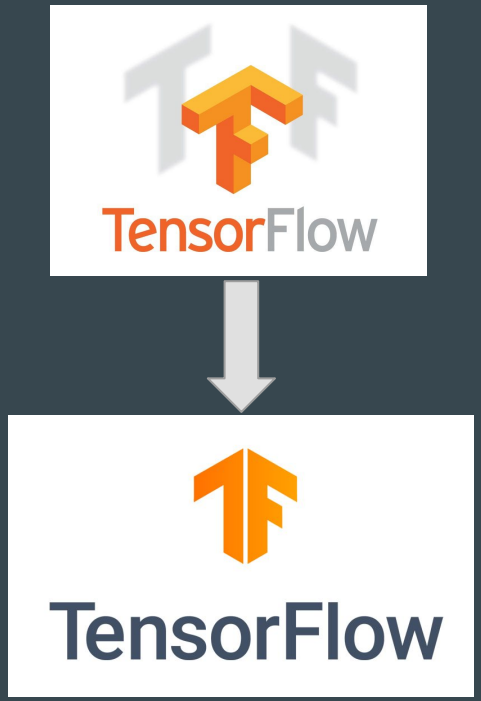
What is TensorFlow

- https://tensorflow.org
- Machine Learning Framework based on dataflow graph, AutoDiff
- TensorFlow 0.5 Release (2015. 11)
- C++ core, Python API
- Several High-level API including Keras
- Define and Run (kitchen sink)



What is TensorFlow 2.0

- 2019. 10 TensorFlow 2.0 Release
- Eager execution (Define by Run)
- Functions, not session
- AutoGraph
- Keras Python API
- API Cleanup(no more globals)
- And much more

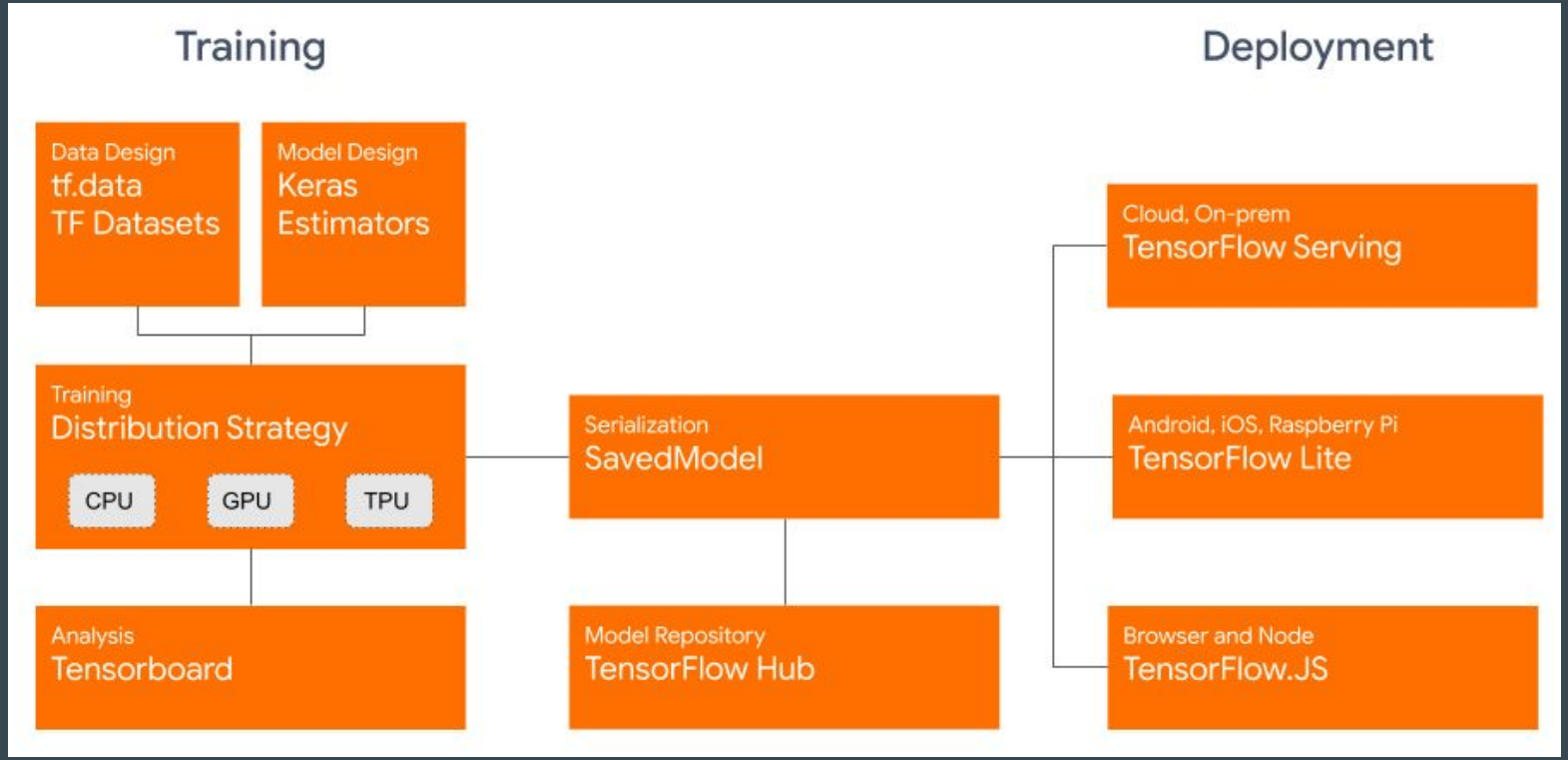


Install TensorFlow

Windows, macOS, Linux + Python (2.7), 3.x

```
$ pip install tensorflow
$ pip install tensorflow-gpu

$ pip install tensorflow==2.0.0
$ pip install tensorflow-gpu==2.0.0
```

TF 2.0 Concept Diagram



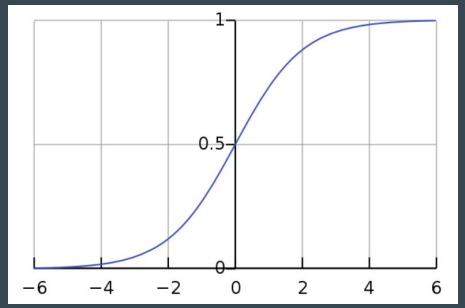
https://medium.com/tensorflow/tensorflow-2-0-is-now-available-57d706c2a9ab

Eager Execution

- 1.x : `tf.enable_eager_execution()`
- 2.x: Default
- Define by Run support (like PyTorch, Chainer)
- Rapid Development
- Easy Debugging (use Python toolchain) → easy to check bottlenecks
- Native Control Flow (if, for etc) → easy to make complex model
- Boost performance by AutoGraph

TensorFlow 1.x

```
>>> import tensorflow as tf
>>>
>>> t = tf.nn.sigmoid([0.])
>>>
>>> print(t)

Tensor("Sigmoid_1:0", shape=(1,),
dtype=float32)
```



TensorFlow 2.x

```
>>> import tensorflow as tf
>>>
>>> t = tf.nn.sigmoid([0.])
>>>
>>> print(t)

tf.Tensor([0.5], shape=(1,), dtype=float32)

>>>
>>> print(t.numpy())

[0.5]
```

TensorFlow 1.x

```
import tensorflow as tf

## 그래프를 정의합니다
g = tf.Graph()
with g.as_default():
    x = tf.placeholder(dtype=tf.float32,
                       shape=(None), name='x')
    w = tf.Variable(2.0, name='weight')
    b = tf.Variable(0.7, name='bias')
    z = w * x + b
    init = tf.global_variables_initializer()

## 세션을 만들고 그래프 g를 전달합니다
with tf.Session(graph=g) as sess:
    ## w와 b를 초기화합니다
    sess.run(init)
    ## z를 평가합니다
    for t in [1.0, 0.6, -1.8]:
        print('x=%4.1f --> z=%4.1f'%(
              t, sess.run(z, feed_dict={x:t})))
```

TensorFlow 2.x

```
import tensorflow as tf

w = tf.Variable(2.0, name='weight')
b = tf.Variable(0.7, name='bias')

# z를 평가합니다
for x in [1.0, 0.6, -1.8]:
    z = w * x + b
    print('x=%4.1f --> z=%4.1f'%(x, z))
```

AutoGraph

tf.Graph() + tf.Session() → @tf.function

```
# TensorFlow 1.x
output = session.run(ops, feed_dict={placeholder: input})
```

```
# TensorFlow 2.x
@tf.function
def simple_func():
    # complex computation with pure python
    ...
    return z

output = simple_func(input)
```

- for/while → tf.while_loop
- if → tf.cond

Graph definition

```
# 텐서플로 1.x 방식
g = tf.Graph()

# 그래프에 노드를 추가합니다.
with g.as_default():
...

g.as_graph_def()

node {
  name: "a"
  op: "Const"
  attr {
    key: "dtype"
    value {
      type: DT_INT32
    }
  }
  ...
```

```
# 텐서플로 2.x 방식
@tf.function
def simple_func():
    ...
    return z

con_func = simple_func.get_concrete_function()
con_func.graph.as_graph_def()

node {
  name: "a"
  op: "Const"
  attr {
    key: "dtype"
    value {
      type: DT_INT32
    }
  }
  ...
```

Keras

- High-Level Neural Networks Specification (https://keras.io) (2015. 03)
- Add to `tf.contrib.keras` at TensorFlow 1.2
- Promote to `tf.keras` at TensorFlow 1.4 (`tf.layers` → `tf.keras`)
- 1st Class Python API for TensorFlow 2.0
- Deprecated `tf.layer`, `tf.contrib.layers`(Slim)
- Keras 2.3.x is last major release of multi-backend Keras. Instead use tf.keras

Sequential API

```
from tensorflow import tf

model = tf.keras.Sequential()
# 64개의 유닛을 가진 완전 연결 층을 모델에 추가합니다:
model.add(tf.keras.layers.Dense(64, activation='relu'))
# 또 하나를 추가합니다:
model.add(tf.keras.layers.Dense(64, activation='relu'))
# 10개의 출력 유닛을 가진 소프트맥스 층을 추가합니다:
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# 컴파일
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# 모델 훈련
model.fit(train_data, labels, epochs=10, batch_size=32)
# 모델 평가
model.evaluate(test_data, labels)
# 샘플 예측
model.predict(new_sample)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(10),
])
```

Functional API

```
from tensorflow import tf

# 입력과 출력을 연결
input = tf.keras.Input(shape=(784,), name='img')
h1 = tf.keras.layers.Dense(64, activation='relu')(inputs)
h2 = tf.keras.layers.Dense(64, activation='relu')(h1)
output = tf.keras.layers.Dense(10, activation='softmax')(h2)

# 모델 생성
model = tf.keras.Model(input, output)

# 컴파일
...
# 훈련
...
```

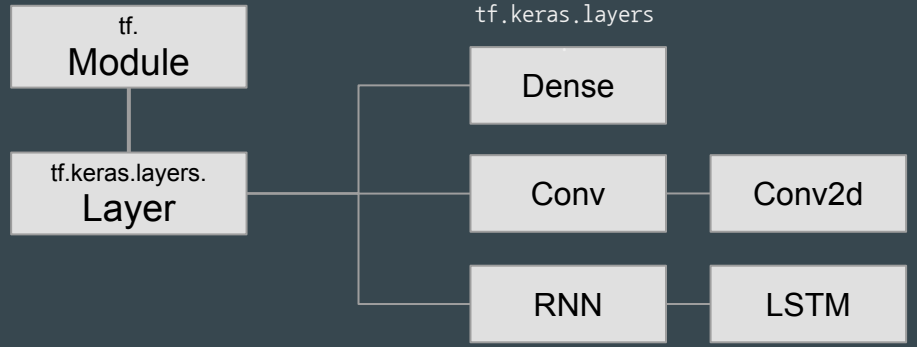
Class Hierarchy

`tf.Variable` Container
`variables(), trainable_variables()`

Class Hierarchy

`tf.Variable` Container
`variables()`, `trainable_variables()`

`__call__()` → `build()` → `add_weights()`
`¦→ call()`
`add_loss()`

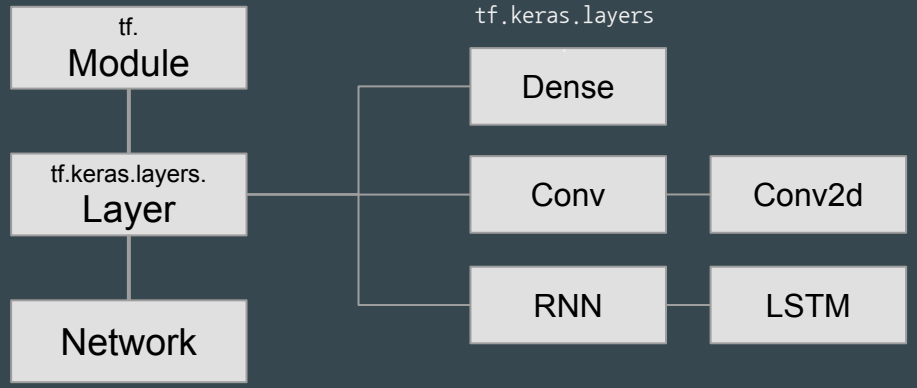


Class Hierarchy

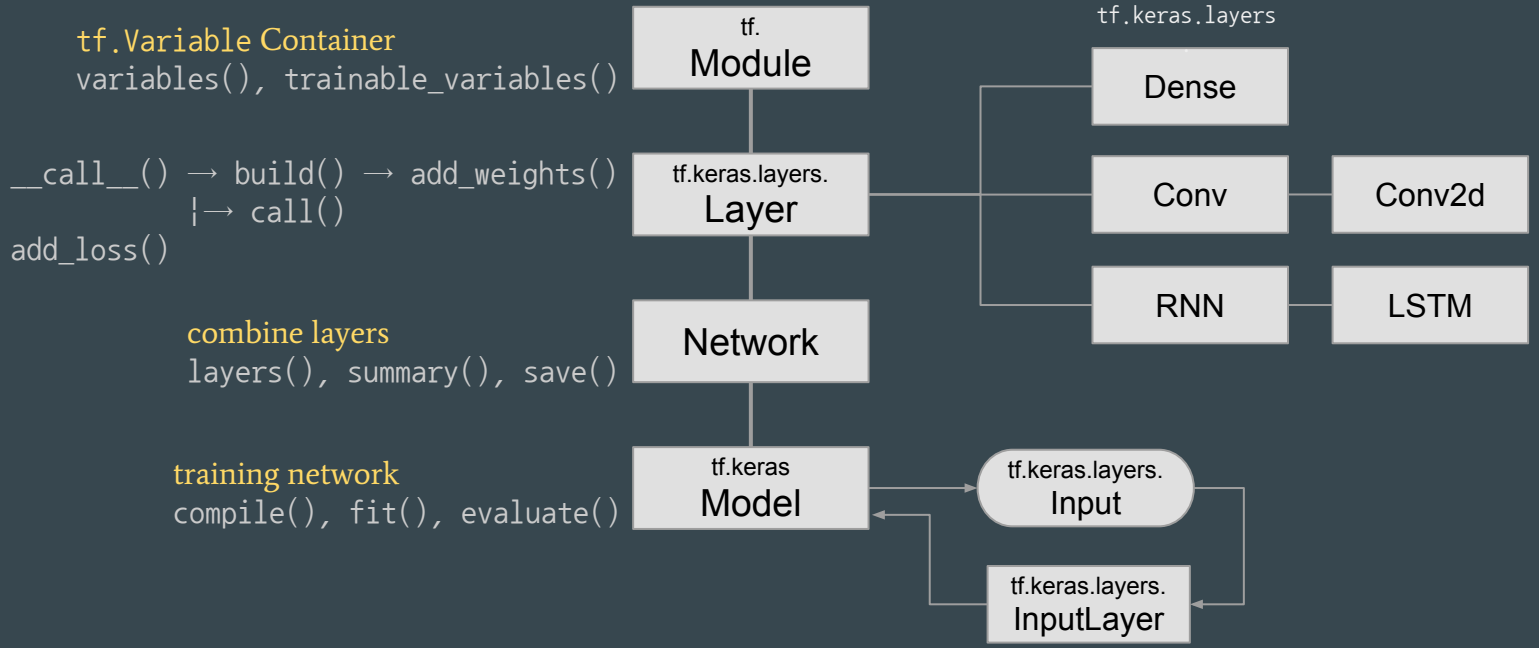
tf.Variable Container
variables(), trainable_variables()

__call__() → build() → add_weights()
          ¦→ call()
add_loss()

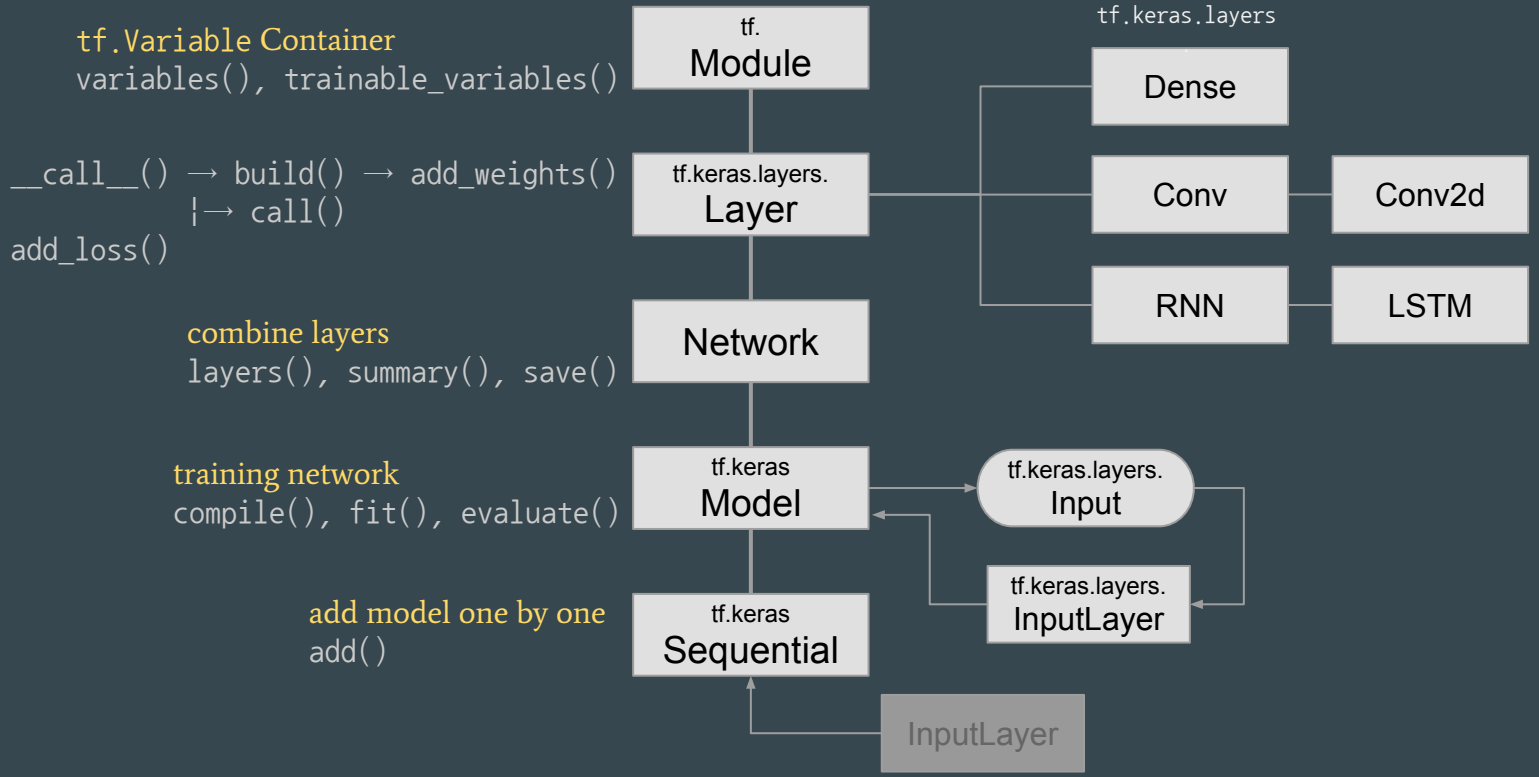
combine layers
layers(), summary(), save()



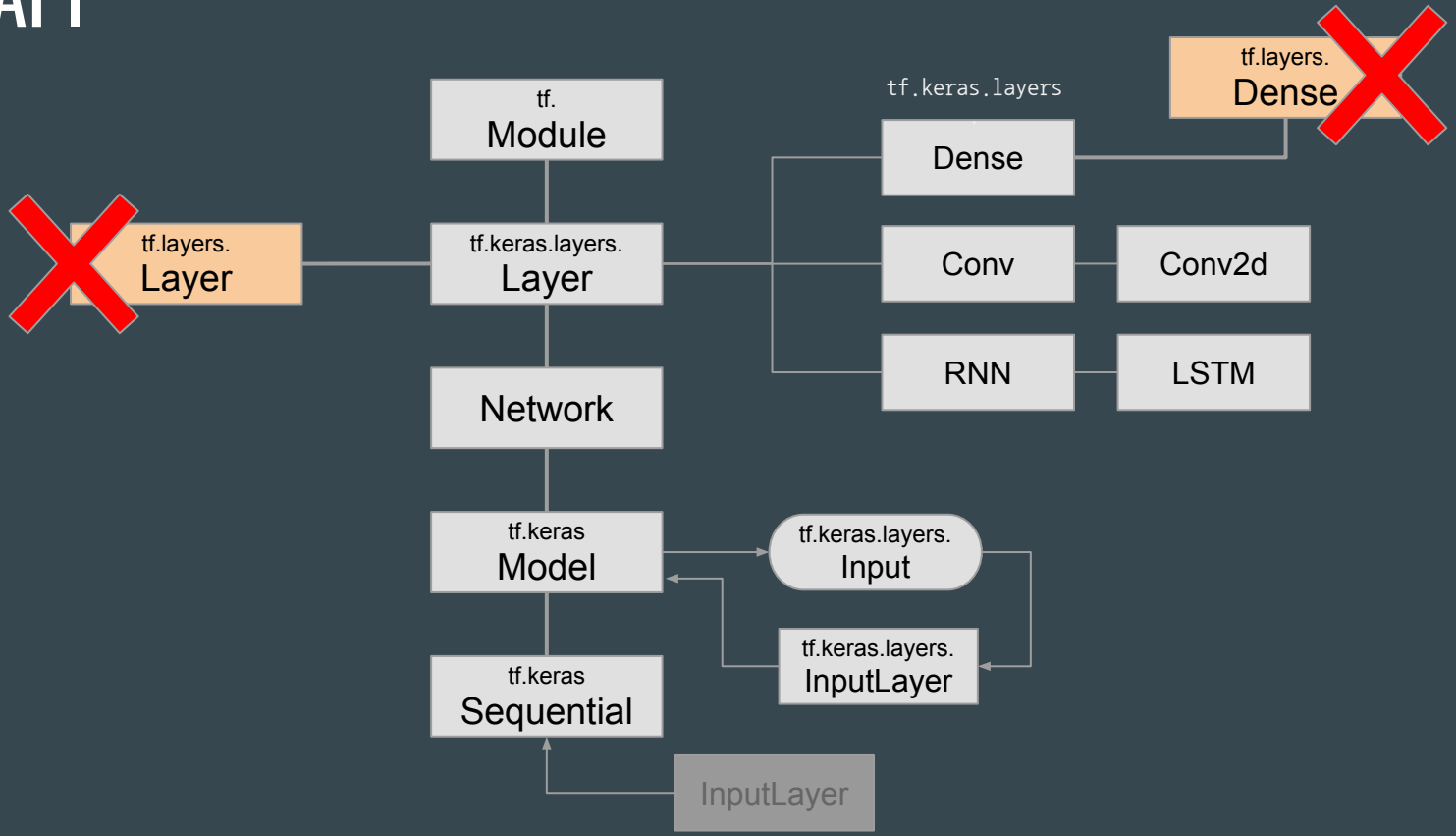
Class Hierarchy



Class Hierarchy



Drop 1.x API


tf.
Module
tf.keras.layers
tf.layers.
Dense
tf.layers.
Layer
tf.keras.layers.
Layer
Dense
Conv
Conv2d
RNN
LSTM
Network
tf.keras
Model
tf.keras.layers.
Input
tf.keras.layers.
InputLayer
tf.keras
Sequential
InputLayer

Custom Layer

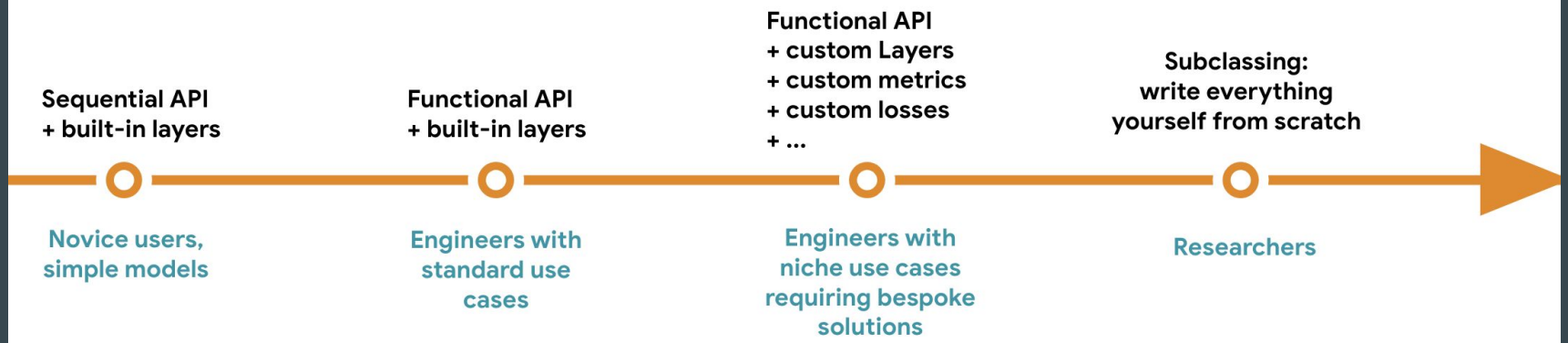
```
class MyLayer(keras.layers.Layer):

    def __init__(self, units, activation=None, **kwargs):
        self.units = units
        self.activation = keras.activations.get(activation)
        super().__init__(**kwargs)

    def build(self, input_shape):
        self.kernel = self.add_weight(name='kernel',
                                      shape=(input_shape[1], self.units),
                                      initializer='uniform')
        self.bias = self.add_weight(name='bias',
                                    shape=(self.units,),
                                    initializer='zeros')
        super().build(input_shape)

    def call(self, X):
        z = tf.matmul(X, self.kernel) + self.bias
        return self.activation(z)
```

Custom Model

```
class MyModel(keras.models.Model):

    def __init__(self, **kwargs):
        self.hidden = MyLayer(10, activation="relu")
        self.output = MyLayer(1)
        super().__init__(**kwargs)

    def call(self, input):
        h = self.hidden(input)
        return self.output(h)

model = MyModel()
model.compile(...)
model.fit(...)
model.evaluate(...)
```

Model building: from **simple** to **arbitrarily flexible**

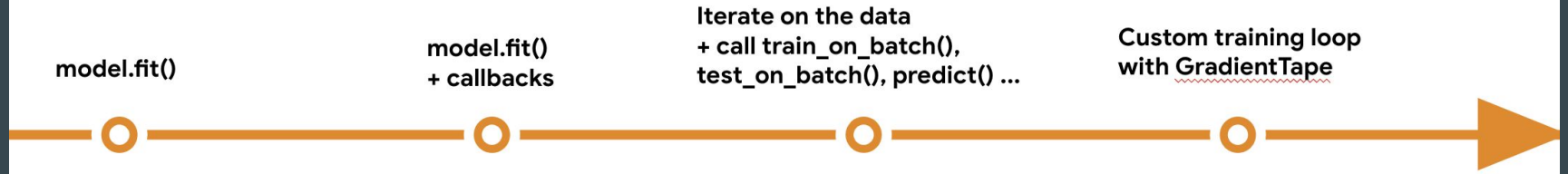
Progressive disclosure of complexity



https://colab.research.google.com/drive/1UCJt8EYjlzCs1H1d1X0iDGYJsHKwu-NO

Model training: from **simple** to **arbitrarily flexible**

Progressive disclosure of complexity



Quick experiment

Customize a typical supervised training loop
e.g. add Checkpointing, Early Stopping, TensorBoard monitoring, send Slack notifications...

Bespoke training loop using regular optimizer / losses / metrics
e.g. GANs, curriculum learning

Whatever you want
e.g. new optimization algorithm, learning to learn

https://colab.research.google.com/drive/1UCJt8EYjlzCs1H1d1X0iDGYJsHKwu-NO

Keras + eager mode

- Keras use TF function by default.
- For using eager mode,
  - `model = Model(dynamic=True)` or
  - `model.compile(..., run_eagerly=True)`

Upgrade to 2.0

- `import tensorflow.compat.v1 as tf`
  `tf.disable_v2_behavior()`

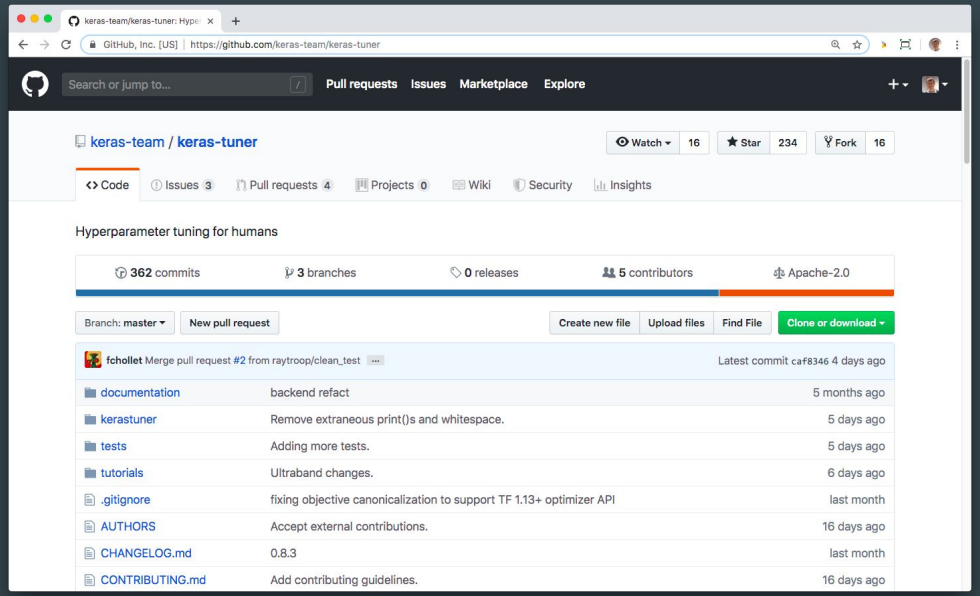
- `tf_upgrade_v2 --infile tensorfoo.py --outfile tensorfoo-upgraded.py`
  `tf_upgrade_v2.py --intree ~/code/old --outtree ~/code/new`
- `tf.layers` → `tf.keras.layers`
- `tf.contrib.layers(slim)` → `tf.layers` → `tf.keras.layers`
- https://www.tensorflow.org/beta/guide/migration_guide

TensorFlow 2.0 Recommendation

- Use Keras layers, models
- Use small python function
- `@tf.function` (AutoGraph)
- `tf.data.Datasets`

Keras Tuner

- Automatic Hyperparameter Searching (AutoML) for tf.keras (TF 2.0)
- github.com/keras-team/keras-tuner (beta)



Keras RFC

API Change: Adding preprocessing layers such as text vectorization, data normalization, and data discretization for model saving and loading

```
normalization = keras.layers.Normalization()
discretization = keras.layers.Discretization()
preprocessing_stage = keras.layers.PreprocessingStage([normalization,
                                                       discretization])
preprocessing_stage.adapt(data_sample)

model = keras.Sequential([
    preprocessing_stage,
    keras.layers.Dense(10, activation='softmax'),
])
model.fit(data, targets, epochs=10)
```

https://github.com/keras-team/governance/blob/master/rfcs/20190502-preprocessing-layers.md

One more thing

API Change: unify keras.preprocessing and the recently-introduced Preprocessing Layers API. keras.preprocessing compatible with tf.data

```
class ImagePipeline(Sequential):

    def __init__(self, layers:List[Layer]):
        ...

preprocessor = ImagePipeline([
    RandomFlip(horizontal=True),
    RandomRotation(0.2, fill_mode='constant'),
    RandomZoom(0.2, fill_mode='constant'),
    RandomTranslation(0.2, fill_mode='constant'),
    Normalization(),
])

dataset = preprocessor.from_directory(dir_name, image_size=(512, 512))
model.fit(dataset, epochs=10)
```

https://github.com/keras-team/governance/blob/master/rfcs/20190729-keras-preprocessing-redesign.md


TensorFlow World
PRESENTED WITH TensorFlow
October 28-31, 2019
Santa Clara, CA

Thank you