


Examining the TensorFlow Graph

 [Run in Google Colab \(https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/graphs\)](https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/graphs)

Overview

TensorBoard's **Graphs dashboard** is a powerful tool for examining your TensorFlow model. You can quickly view a conceptual graph of your model's structure and ensure it matches your intended design. You can also view a op-level graph to understand how TensorFlow understands your program. Examining the op-level graph can give you insight as to how to change your model. For example, you can redesign your model if training is progressing slower than expected.

This tutorial presents a quick overview of how to generate graph diagnostic data and visualize it in TensorBoard's Graphs dashboard. You'll define and train a simple Keras Sequential model for the Fashion-MNIST dataset and learn how to log and examine your model graphs. You will also use a tracing API to generate graph data for functions created using the new `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) annotation.

Setup

```
# Load the TensorBoard notebook extension.  
%load_ext tensorboard
```

```
from datetime import datetime  
from packaging import version
```

```
import tensorflow as tf
```

```
from tensorflow import keras

print("TensorFlow version: ", tf.__version__)
assert version.parse(tf.__version__).release[0] >= 2, \
    "This notebook requires TensorFlow 2.0 or above."
```

TensorFlow version: 2.2.0

```
import tensorboard
tensorboard.__version__
```

'2.2.1'

```
$ # Clear any logs from previous runs
$ rm -rf ./logs/
```

Define a Keras model

In this example, the classifier is a simple four-layer Sequential model.

```
# Define the model.
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(32, activation='relu'),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(
    optimizer='adam',
```

```
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

Download and prepare the training data.

```
(train_images, train_labels), _ = keras.datasets.fashion_mnist.load_data()
train_images = train_images / 255.0
```

Train the model and log data

Before training, define the Keras TensorBoard callback

(https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/TensorBoard), specifying the log directory. By passing this callback to `Model.fit()`, you ensure that graph data is logged for visualization in TensorBoard.

```
# Define the Keras TensorBoard callback.
logdir="logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir)

# Train the model.
model.fit(
    train_images,
    train_labels,
    batch_size=64,
    epochs=5,
    callbacks=[tensorboard_callback])
```

Epoch 1/5

938/938 [=====] - 2s 2ms/step - loss: 0.6955 - accuracy

Epoch 2/5

938/938 [=====] - 2s 2ms/step - loss: 0.4877 - accuracy

Epoch 3/5

938/938 [=====] - 2s 2ms/step - loss: 0.4458 - accuracy

Epoch 4/5

```
938/938 [=====] - 2s 2ms/step - loss: 0.4246 - accuracy  
Epoch 5/5  
938/938 [=====] - 2s 2ms/step - loss: 0.4117 - accuracy  
<tensorflow.python.keras.callbacks.History at 0x7f656ecc3fd0>
```

Op-level graph

Start TensorBoard and wait a few seconds for the UI to load. Select the Graphs dashboard by tapping “Graphs” at the top.

```
%tensorboard --logdir logs
```

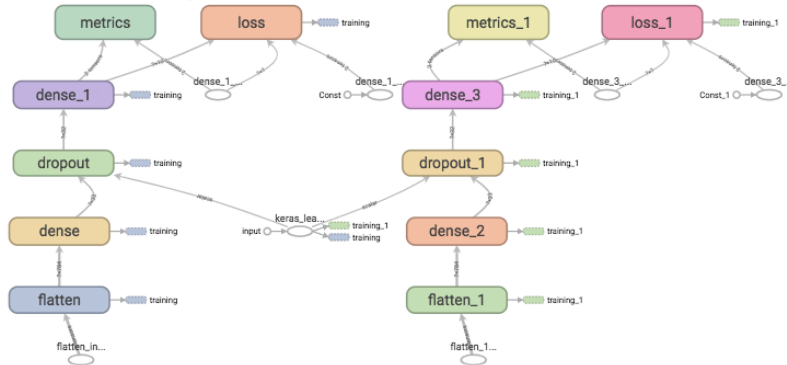
By default, TensorBoard displays the **op-level graph**. (On the left, you can see the “Default” tag selected.) Note that the graph is inverted; data flows from bottom to top, so it’s upside down compared to the code. However, you can see that the graph closely matches the Keras model definition, with extra edges to other computation nodes.

Graphs are often very large, so you can manipulate the graph visualization:

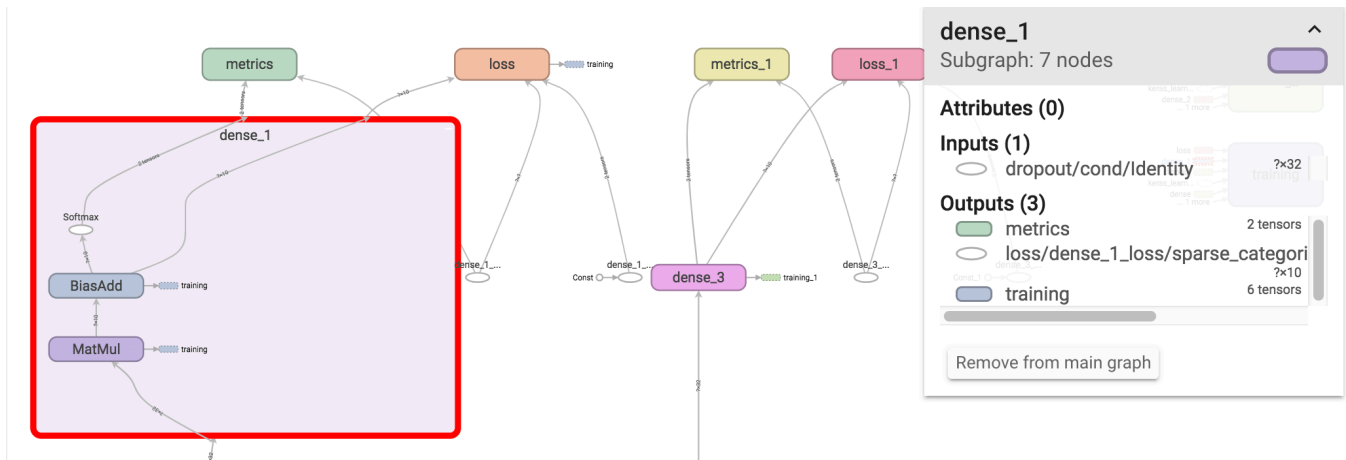
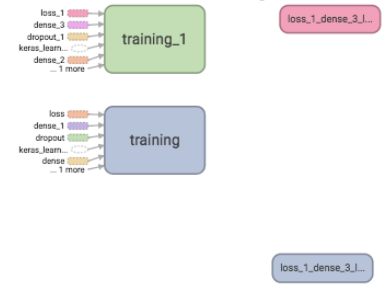
- Scroll to **zoom** in and out
- Drag to **pan**
- Double clicking toggles **node expansion** (a node can be a container for other nodes)

You can also see metadata by clicking on a node. This allows you to see inputs, outputs, shapes and other details.

Main Graph



Auxiliary Nodes



Conceptual graph

In addition to the execution graph, TensorBoard also displays a **conceptual graph**. This is a view of just the Keras model. This may be useful if you're reusing a saved model and you want to examine or validate its structure.

To see the conceptual graph, select the "keras" tag. For this example, you'll see a collapsed **Sequential** node. Double-click the node to see the model's structure:

Run (2) fit/20190302-182835/train ▼

Tag (3) **Default** ▼

Upload batch_2 Choose File

☒ Graph

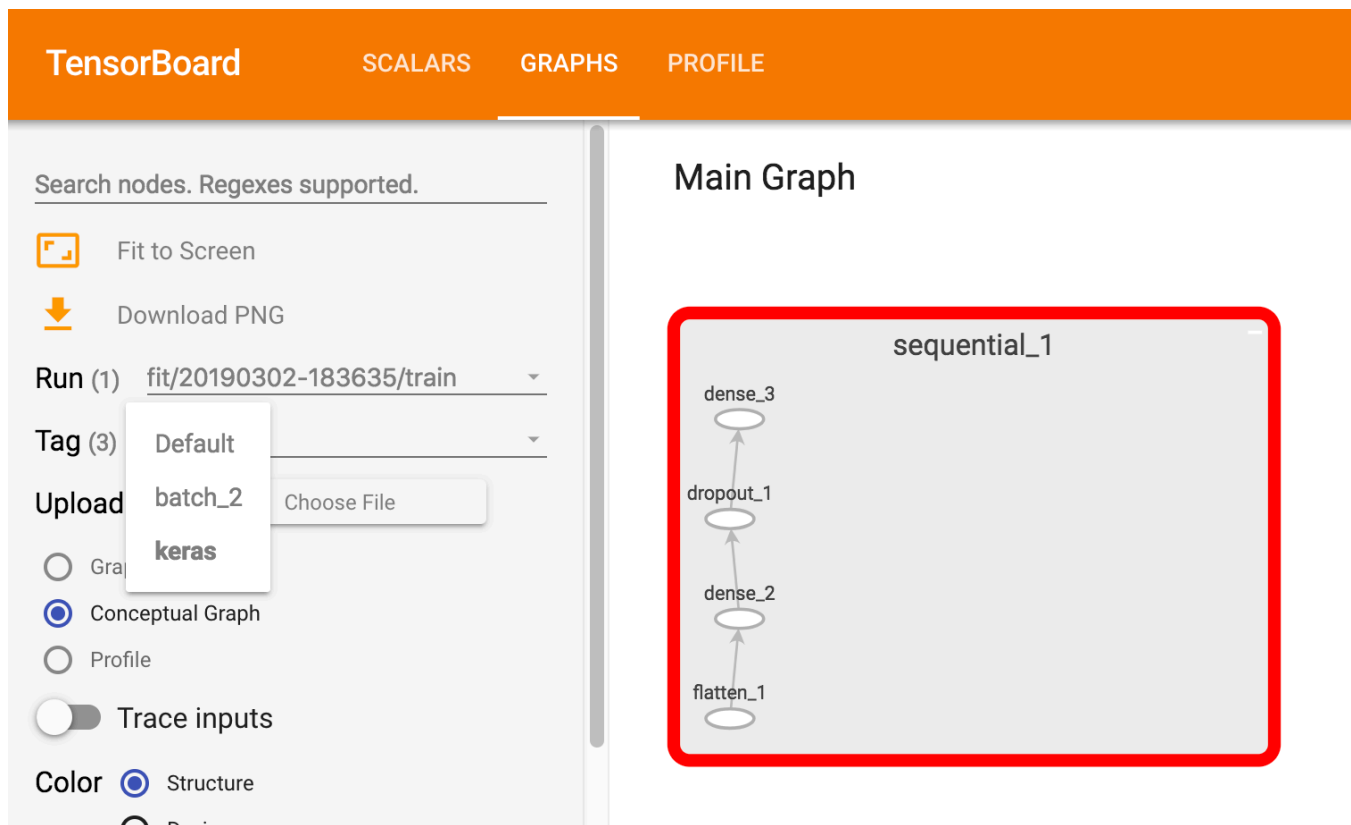
☐ Conceptual Graph

☐ Profile

☐ Trace inputs

Color ☒ Structure

☐ Device



Graphs of tf.functions

The examples so far have described graphs of Keras models, where the graphs have been created by defining Keras layers and calling `Model.fit()`.

You may encounter a situation where you need to use the `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function) annotation to "autograph" (<https://www.tensorflow.org/guide/function>), i.e., transform, a Python computation function into a high-performance TensorFlow graph. For these situations, you use **TensorFlow Summary Trace API** to log autographed functions for visualization in TensorBoard.

To use the Summary Trace API:

- Define and annotate a function with `tf.function` (https://www.tensorflow.org/api_docs/python/tf/function)
- Use `tf.summary.trace_on()` (https://www.tensorflow.org/api_docs/python/tf/summary/trace_on) immediately before your function call site.

- Add profile information (memory, CPU time) to graph by passing `profiler=True`
- With a Summary file writer, call `tf.summary.trace_export()`.
(https://www.tensorflow.org/api_docs/python/tf/summary/trace_export) to save the log data

You can then use TensorBoard to see how your function behaves.

```
# The function to be traced.
@tf.function
def my_func(x, y):
    # A simple hand-rolled layer.
    return tf.nn.relu(tf.matmul(x, y))

# Set up logging.
stamp = datetime.now().strftime("%Y%m%d-%H%M%S")
logdir = 'logs/func/%s' % stamp
writer = tf.summary.create_file_writer(logdir)

# Sample data for your function.
x = tf.random.uniform((3, 3))
y = tf.random.uniform((3, 3))

# Bracket the function call with
# tf.summary.trace_on() and tf.summary.trace_export().
tf.summary.trace_on(graph=True, profiler=True)
# Call only one tf.function when tracing.
z = my_func(x, y)
with writer.as_default():
    tf.summary.trace_export(
        name="my_func_trace",
        step=0,
        profiler_outdir=logdir)

%tensorboard --logdir logs/func
```


The screenshot shows the TensorBoard interface with the 'Main Graph' and the 'MatMul' node details.

TensorBoard **GRAPHS** **INACTIVE** [Settings] [Refresh] [Help]

Search nodes (regex)

Fit to screen
Download PNG
Upload file

Run (1) 20221014-212842
Tag (1) my_func_trace

Graph type

- ☐ Op graph
- ☐ Conceptual graph
- ☒ Profile

Node options

Legend

- colors same substructure
- unique substructure
- unused substructure (* = expandable)
- Namespace* 2
- OpNode 2

Main Graph

```

graph TD
    identity_R[identity_R...] --> Identity[Identity]
    Identity --> Relu[Relu]
    Relu --> MatMul[MatMul]
    MatMul --> output1[ ]
    MatMul --> output2[ ]
  
```

MatMul
Operation: MatMul

Attributes (3)

- T {"type": "DT_FLOAT"}
- transpose_a {"b": false}
- transpose_b {"b": false}

Inputs (2)

- x 3x3
- y 3x3

Outputs (1)

- Relu

Remove from main graph

You can now see the structure of your function as understood by TensorBoard. Click on the "Profile" radiobutton to see CPU and memory statistics.

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 4.0 License](https://creativecommons.org/licenses/by/4.0/) (<https://creativecommons.org/licenses/by/4.0/>), and code samples are licensed under the [Apache 2.0 License](https://www.apache.org/licenses/LICENSE-2.0) (<https://www.apache.org/licenses/LICENSE-2.0>). For details, see the [Google Developers Site Policies](https://developers.google.com/site-policies) (<https://developers.google.com/site-policies>). Java is a registered trademark of Oracle and/or its affiliates.

Last updated 2023-10-25 UTC.