Google Cloud

Core TensorFlow

Lak Lakshmanan

Learn how to...

Write lazy evaluation and imperative programs

Learn how to...

Write lazy evaluation and imperative programs

Work with graphs, sessions, and variables

Learn how to...

Write lazy evaluation and imperative programs

Work with graphs, sessions, and variables

Visualize TensorFlow graphs

Learn how to...

Write lazy evaluation and imperative programs

Work with graphs, sessions, and variables

Visualize TensorFlow graphs

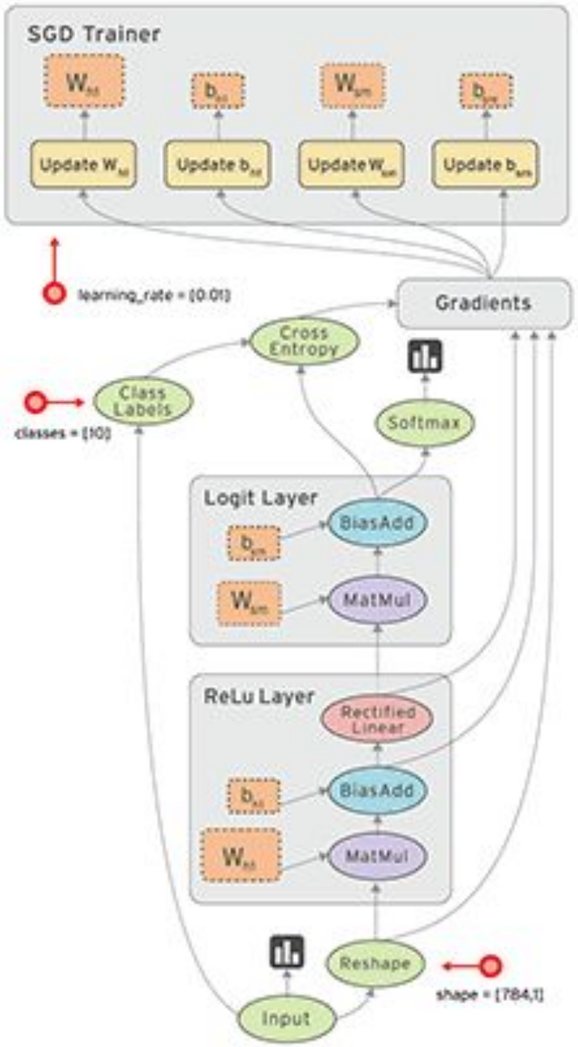
Debug TensorFlow programs

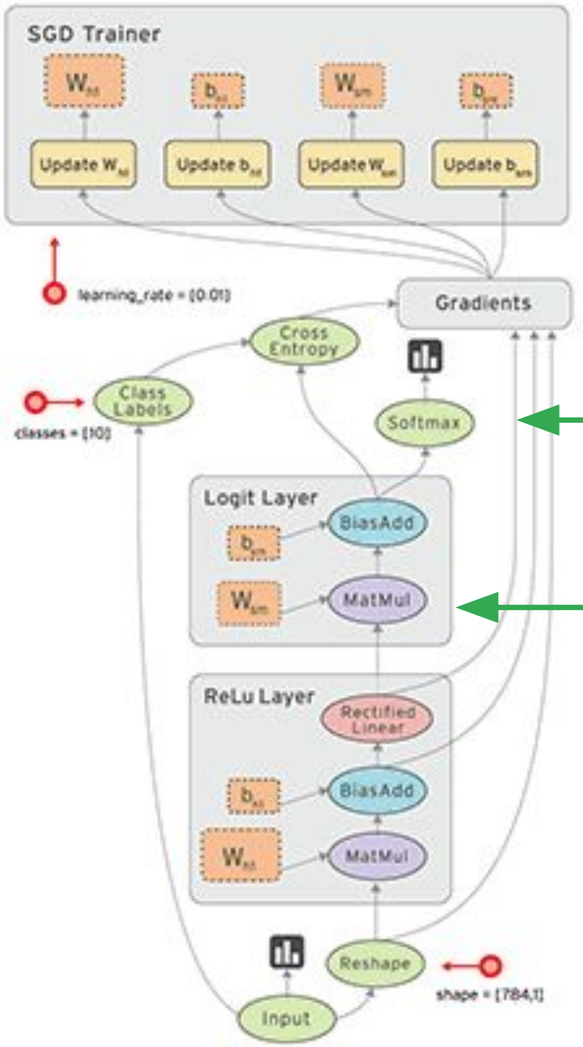
Google Cloud

What is TensorFlow

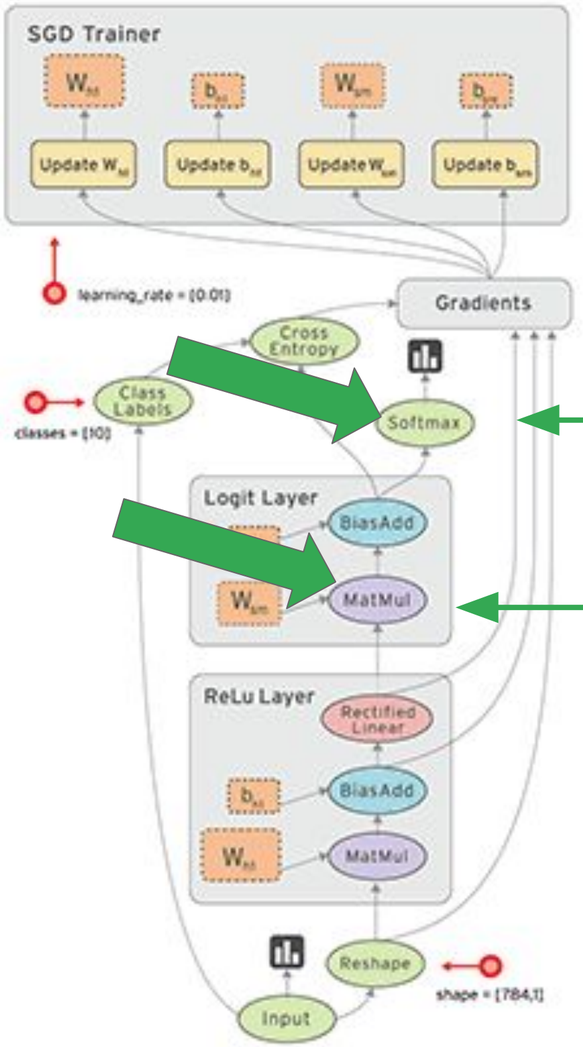
Lak Lakshmanan

TensorFlow is an
open-source
high-performance library
for numerical
computation that uses
directed graphs

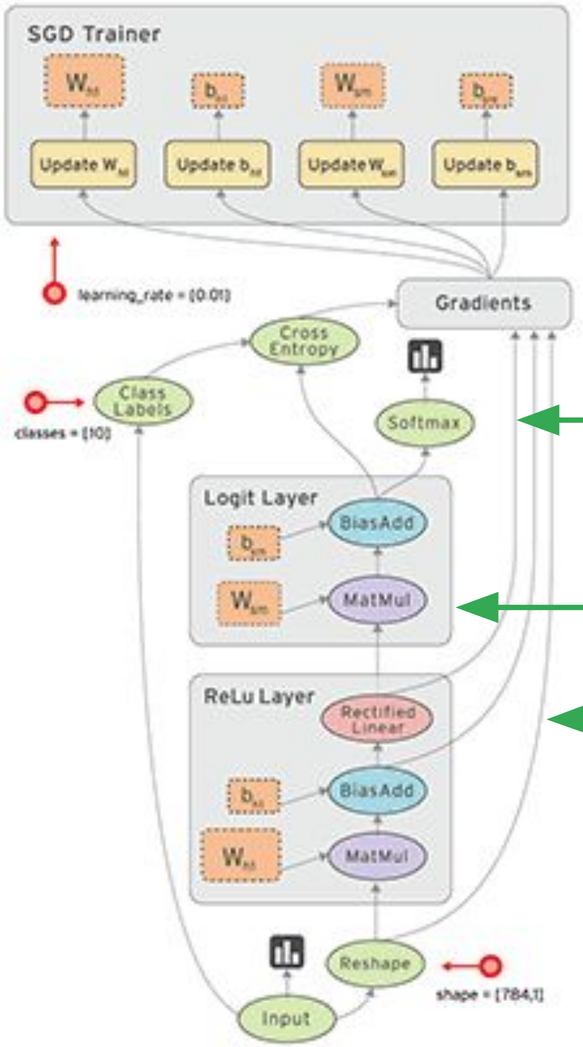

SGD Trainer
Update W_h1
Update b_h1
Update W_sm
Update b_sm
learning_rate = (0.01)
Gradients
Cross Entropy
Class Labels
classes = (10)
Softmax
Logit Layer
BiasAdd
MatMul
ReLu Layer
Rectified Linear
BiasAdd
MatMul
Reshape
shape = [784,1]
Input



Nodes represent mathematical operations

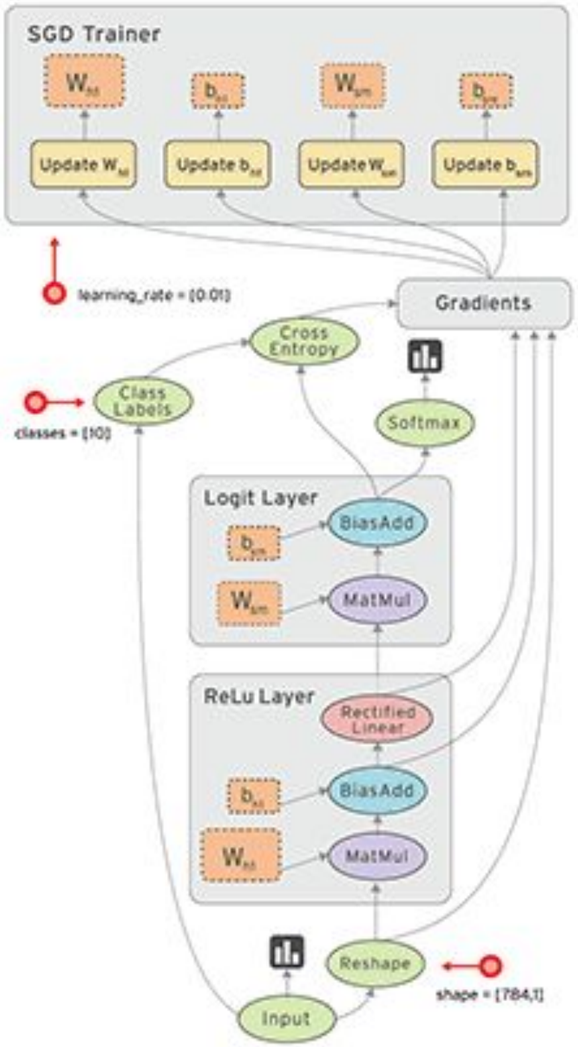


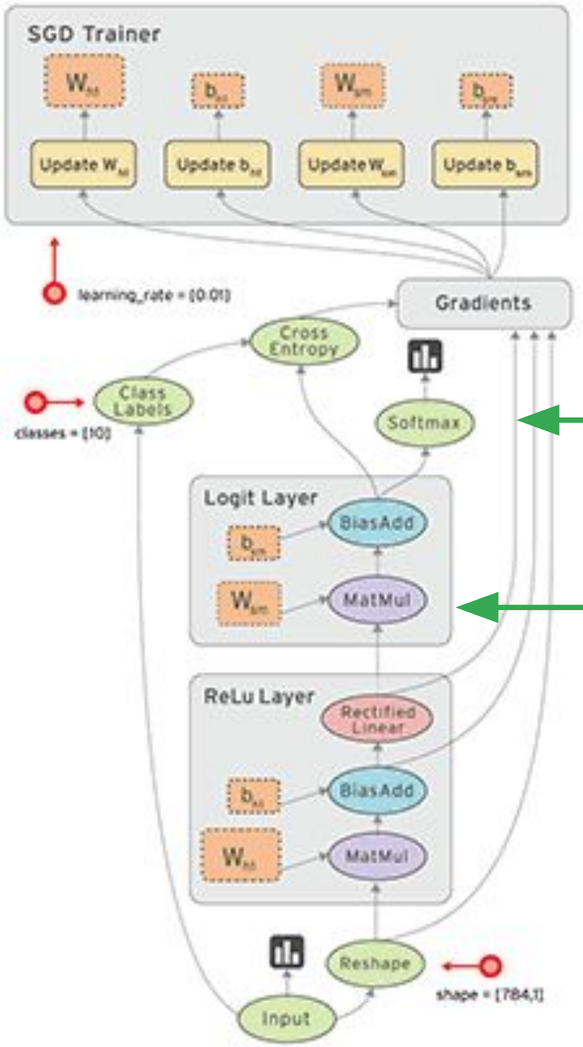
Nodes represent mathematical operations



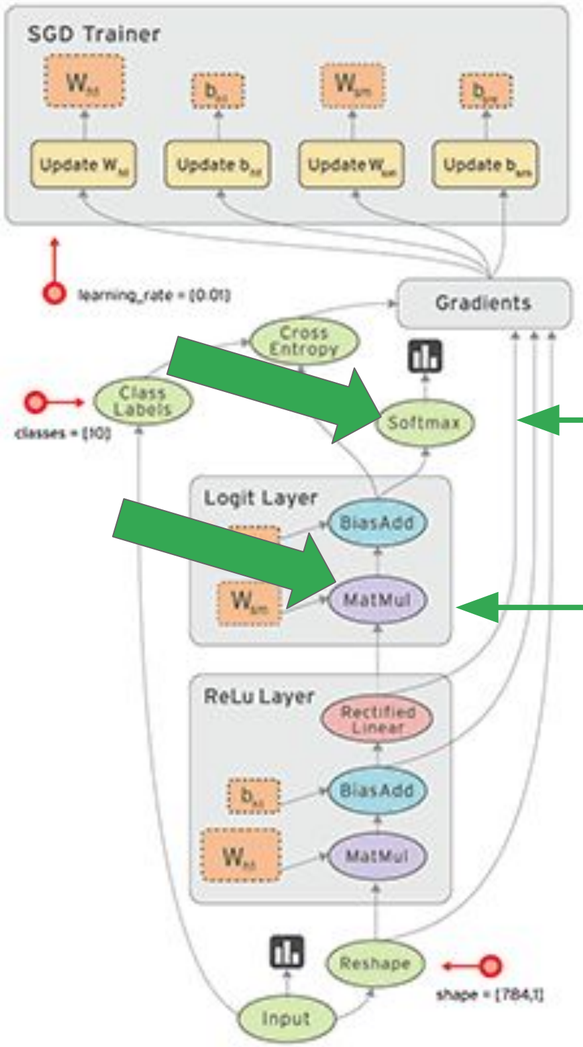
Nodes represent mathematical operations

Edges represent arrays of data

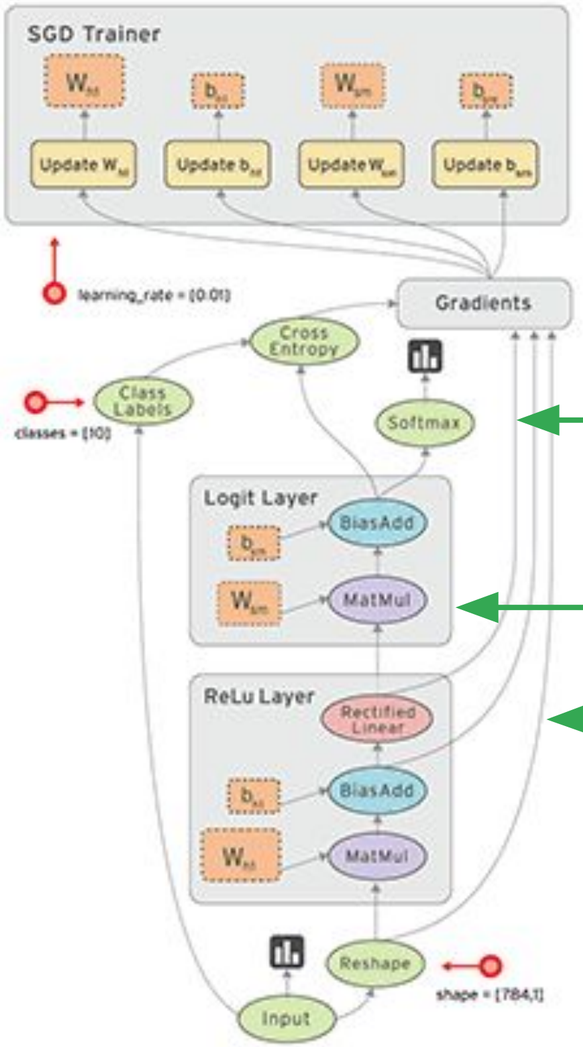

SGD Trainer
Update W_h1
Update b_h1
Update W_sm
Update b_sm
learning_rate = (0.01)
Gradients
Cross Entropy
Class Labels
classes = (10)
Softmax
Logit Layer
BiasAdd
MatMul
ReLu Layer
Rectified Linear
BiasAdd
MatMul
Reshape
shape = [784,1]
Input



Nodes represent mathematical operations



Nodes represent mathematical operations



Nodes represent mathematical operations

Edges represent arrays of data

A tensor is an N-dimensional array of data

Rank 0
Tensor
scalar

A tensor is an N-dimensional array of data

Rank 0
Tensor
scalar

Rank 1
Tensor
vector

A tensor is an N-dimensional array of data

Rank 0
Tensor
scalar

Rank 1
Tensor
vector

Rank 2
Tensor
matrix

A tensor is an N-dimensional array of data

Rank 0 Tensor
scalar

Rank 1 Tensor
vector

Rank 2 Tensor
matrix

Rank 3 Tensor

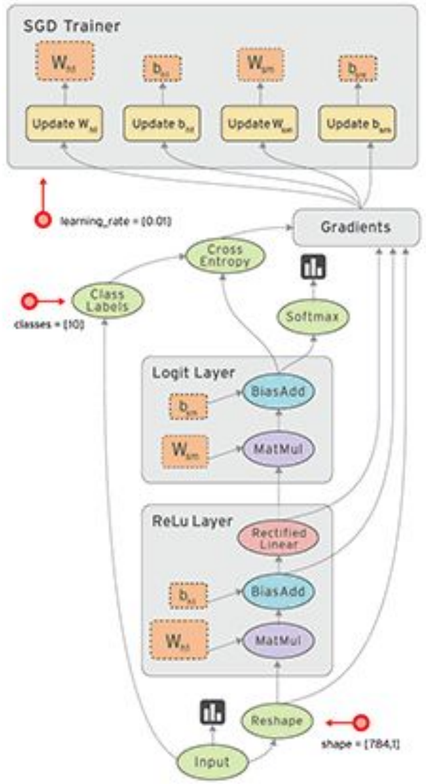
Rank 4 Tensor

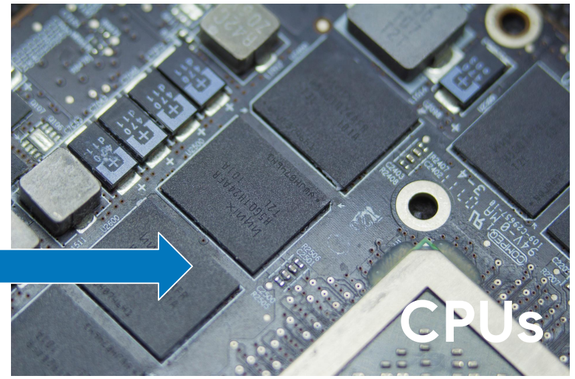
Google Cloud

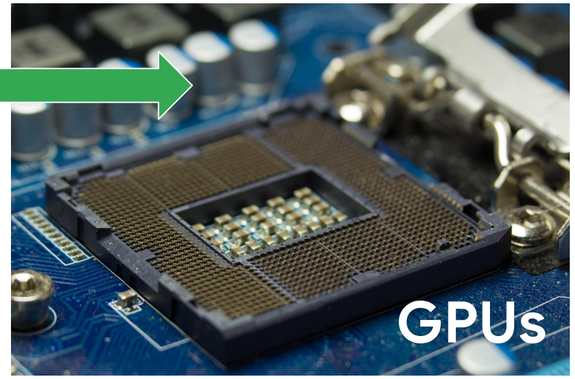
Benefits of a Directed Graph

Lak Lakshmanan

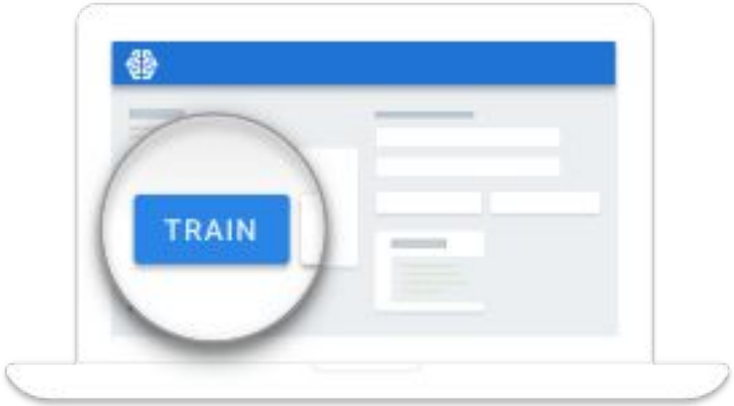
TensorFlow graphs are portable between different devices





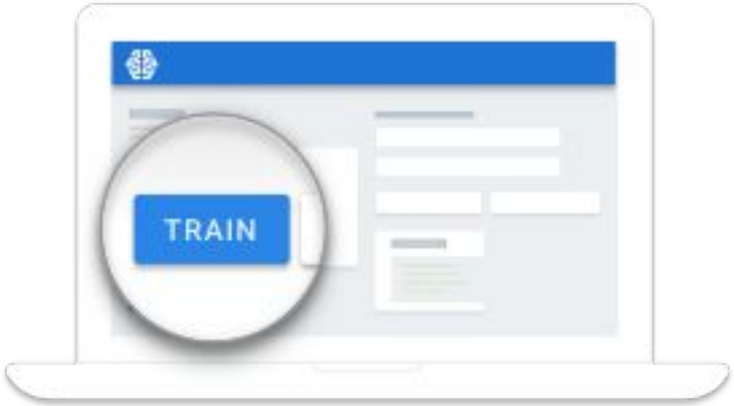


TensorFlow Lite provides on-device inference of ML models on mobile devices and is available for a variety of hardware



Train on cloud

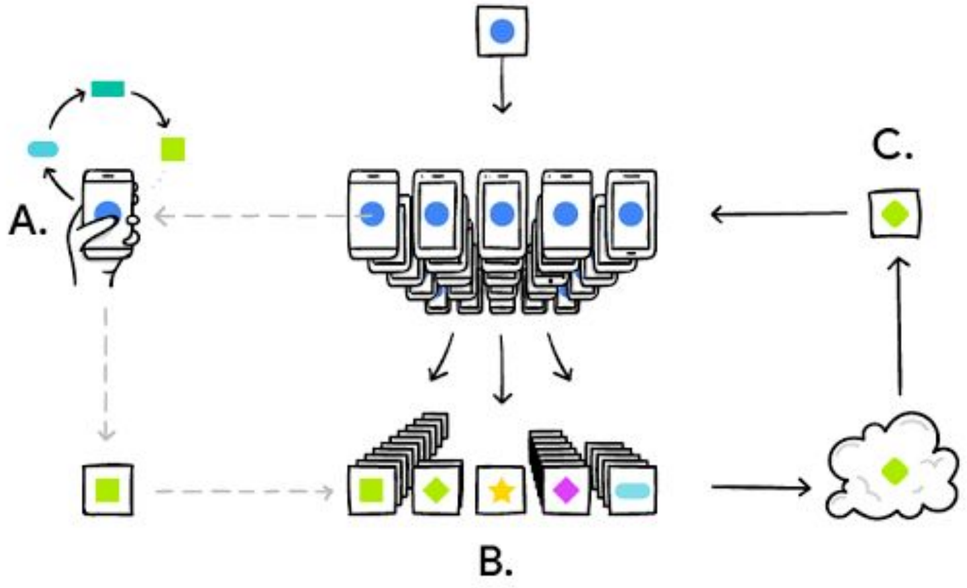
TensorFlow Lite provides on-device inference of ML models on mobile devices and is available for a variety of hardware



Train on cloud

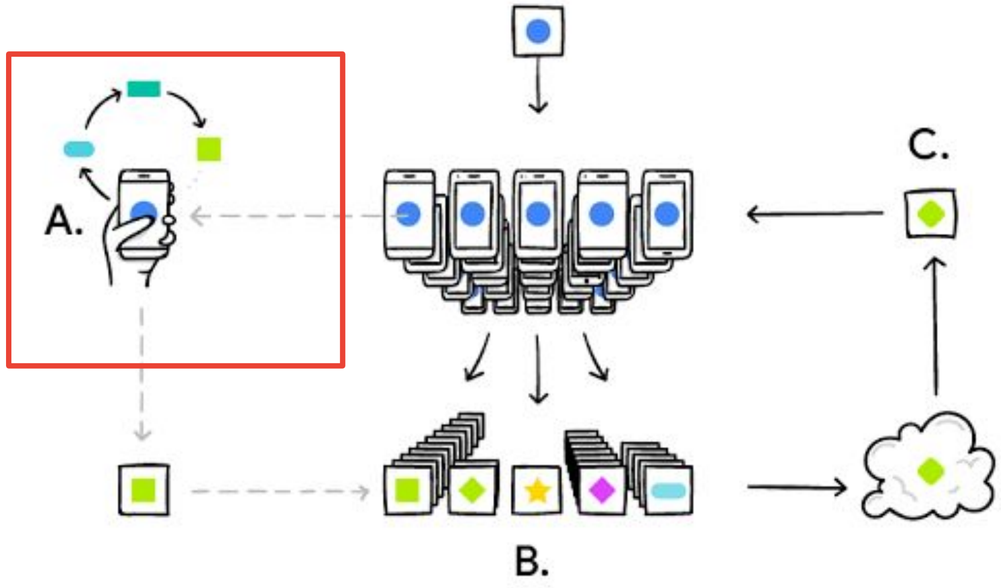
Run inference on iOS, Android, Raspberry Pi, etc.

TensorFlow even supports federated learning



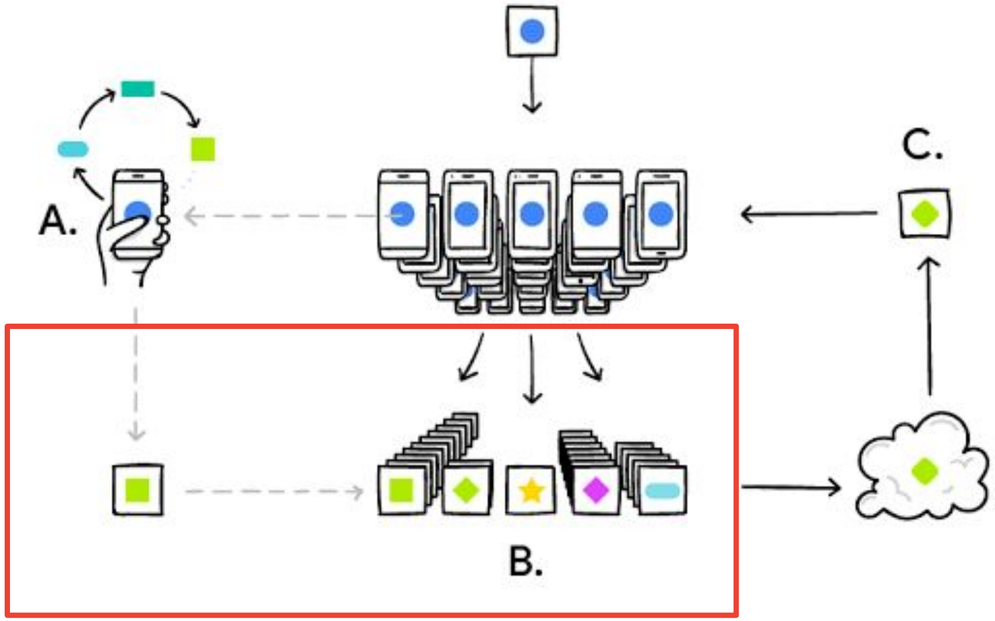
https://research.googleblog.com/2017/04/federated-learning-collaborative.html

TensorFlow even supports federated learning



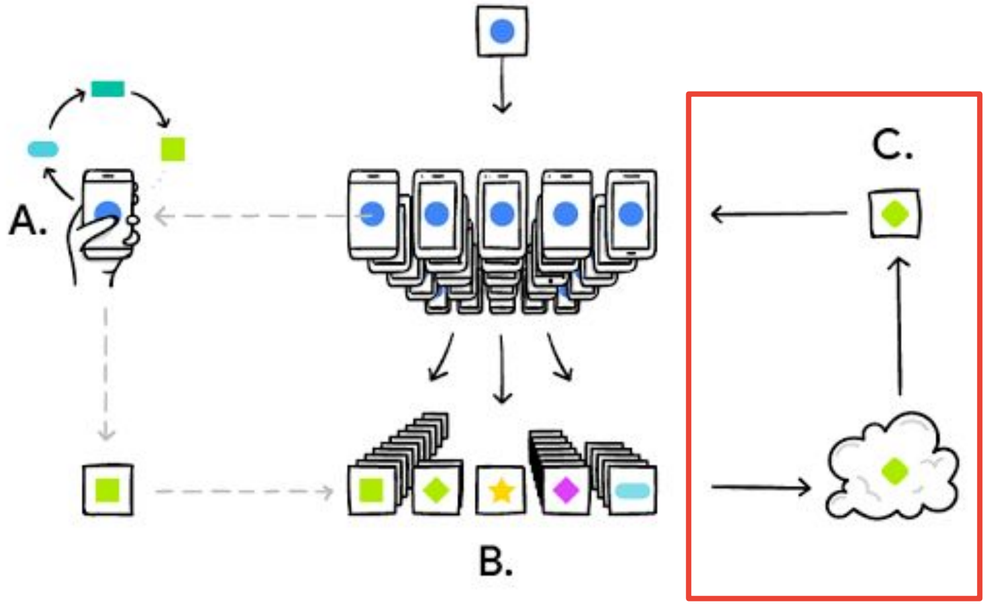
https://research.googleblog.com/2017/04/federated-learning-collaborative.html

TensorFlow even supports federated learning



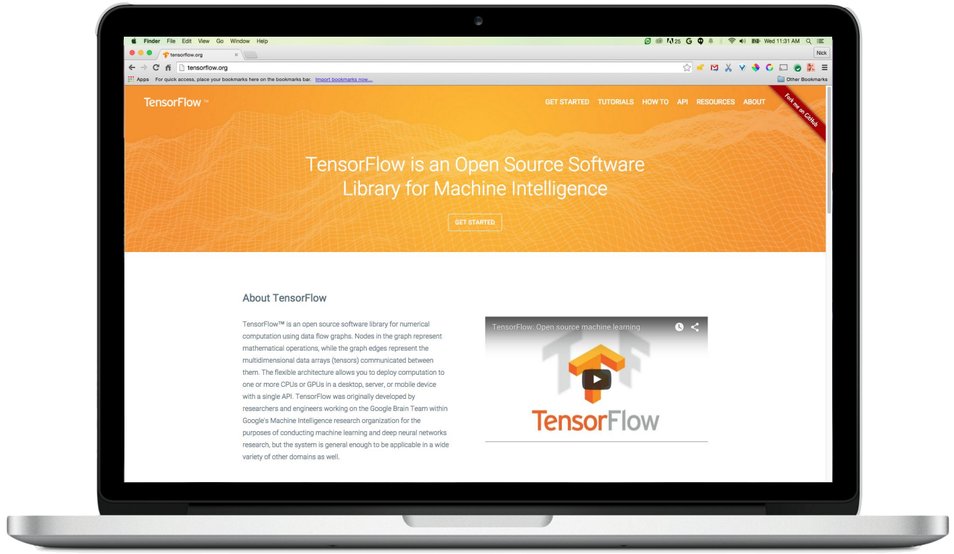
https://research.googleblog.com/2017/04/federated-learning-collaborative.html

TensorFlow even supports federated learning

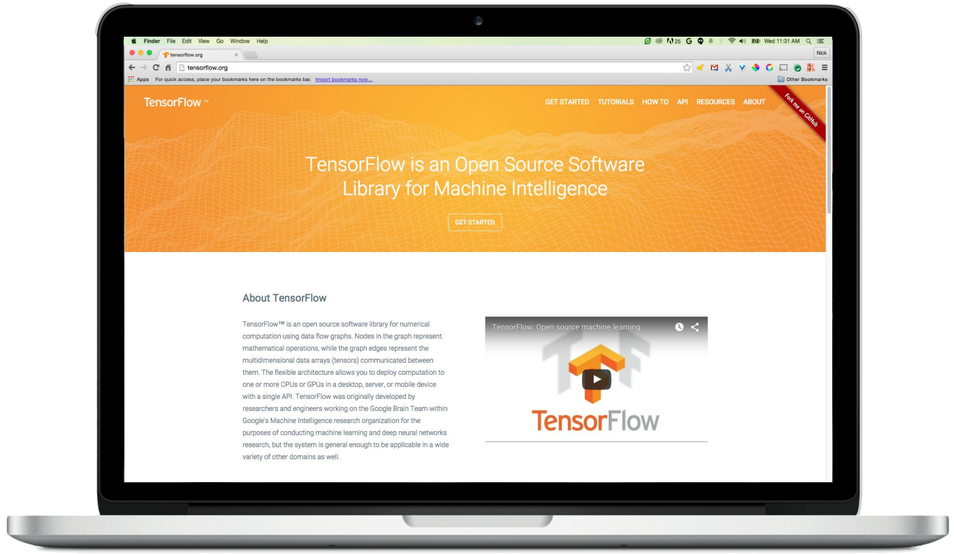


https://research.googleblog.com/2017/04/federated-learning-collaborative.html

TensorFlow is popular among both deep learning researchers and machine learning engineers

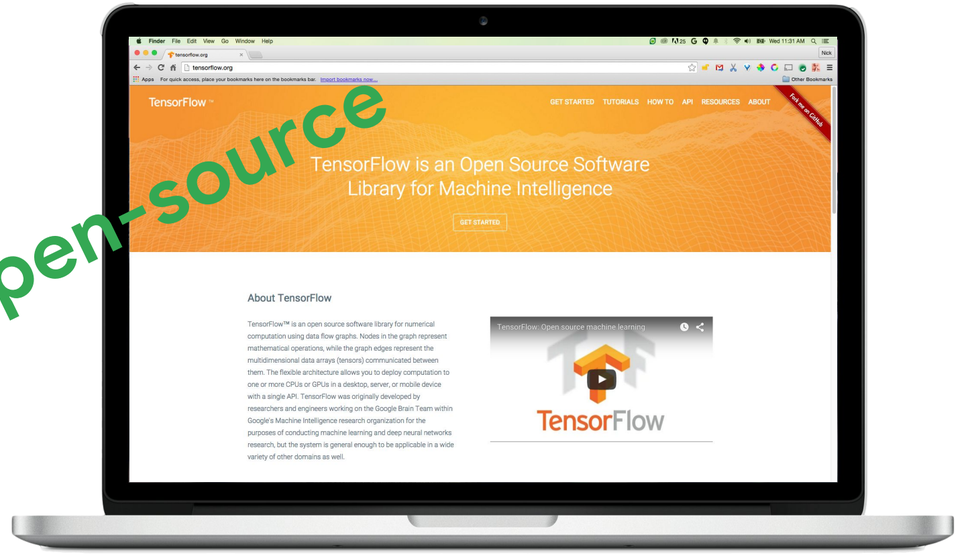


TensorFlow is popular among both deep learning researchers and machine learning engineers



#1 repository for “machine learning” category on GitHub

TensorFlow is popular among both deep learning researchers and machine learning engineers



#1 repository for “machine learning” category on GitHub

Google Cloud

TensorFlow API hierarchy

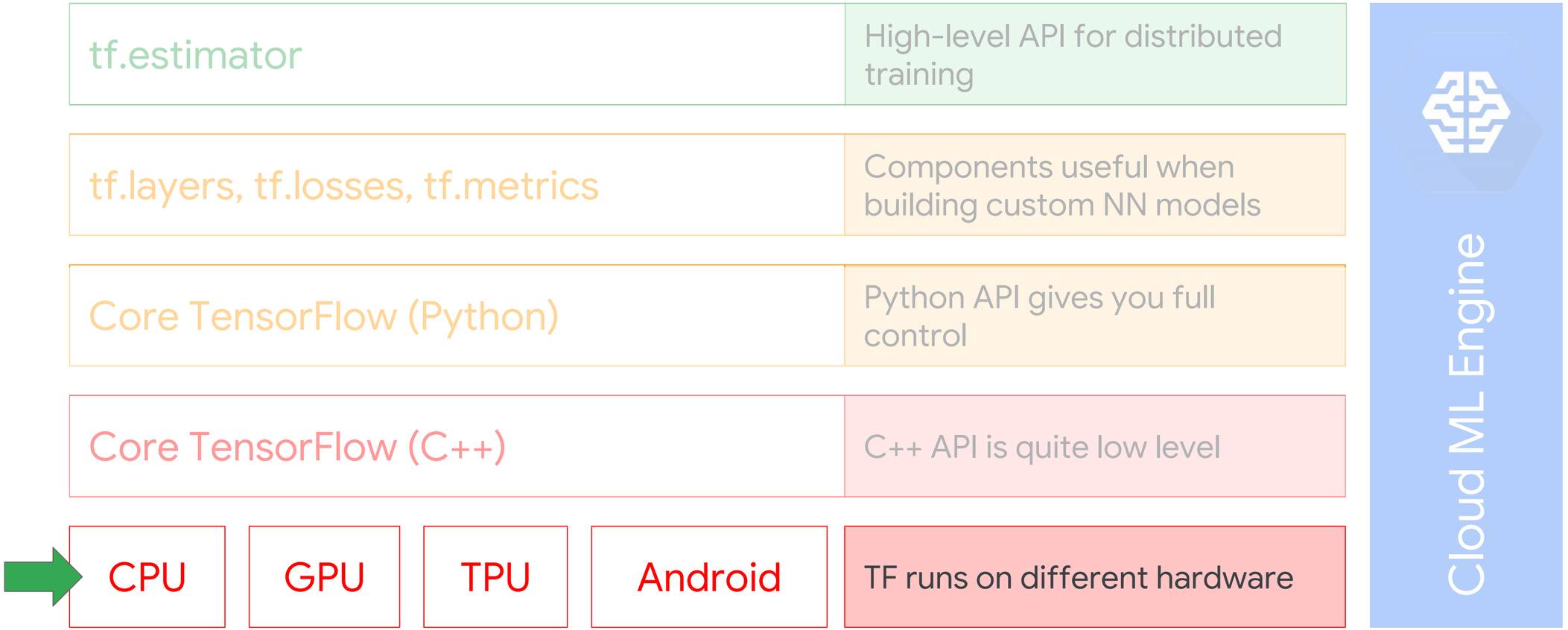
Lak Lakshmanan

TensorFlow toolkit hierarchy

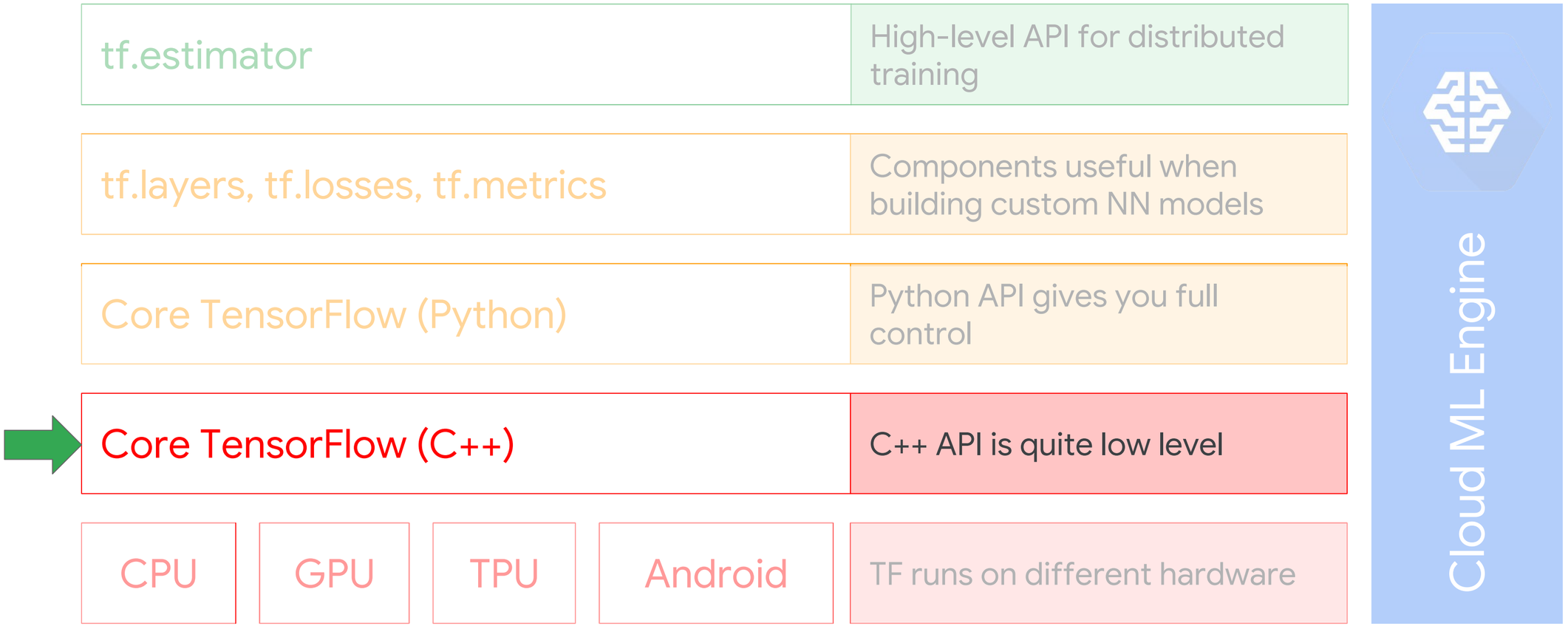
tf.estimator	High-level API for distributed training
tf.layers, tf.losses, tf.metrics	Components useful when building custom NN models
Core TensorFlow (Python)	Python API gives you full control
Core TensorFlow (C++)	C++ API is quite low level
CPU GPU TPU Android	TF runs on different hardware

Run TF at scale with CMLE

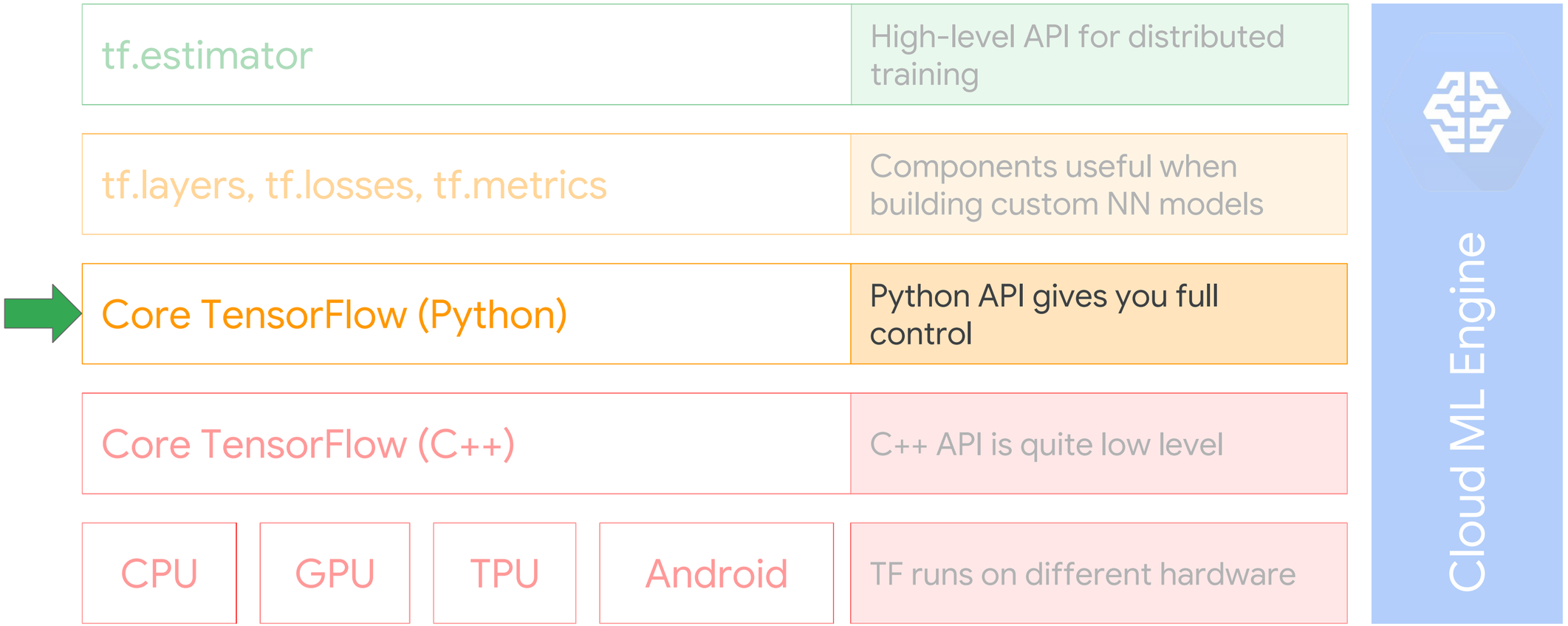
TensorFlow toolkit hierarchy



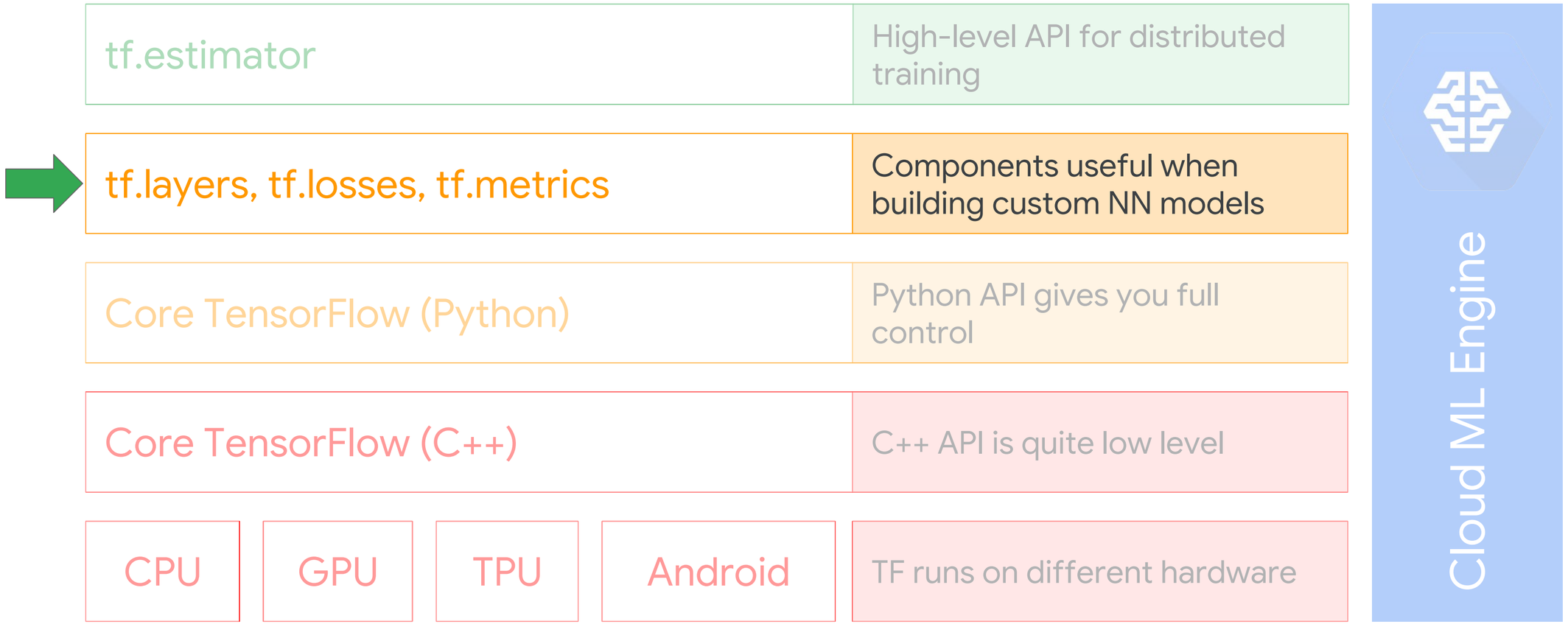
TensorFlow toolkit hierarchy



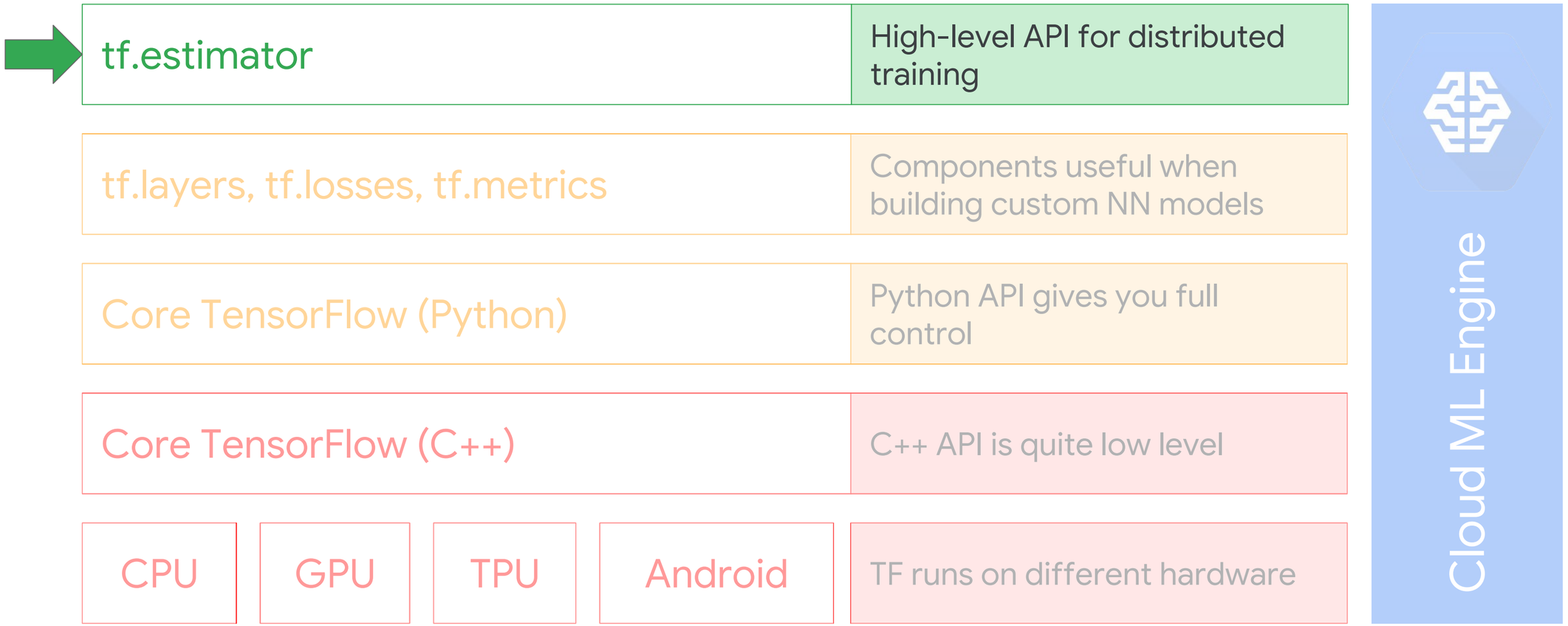
TensorFlow toolkit hierarchy



TensorFlow toolkit hierarchy



TensorFlow toolkit hierarchy



TensorFlow toolkit hierarchy

Run TF at scale with CMLE

tf.estimator	High-level API for distributed training
tf.layers, tf.losses, tf.metrics	Components useful when building custom NN models
Core TensorFlow (Python)	Python API gives you full control
Core TensorFlow (C++)	C++ API is quite low level
CPU GPU TPU Android	TF runs on different hardware

The Python API lets you build and run Directed Graphs

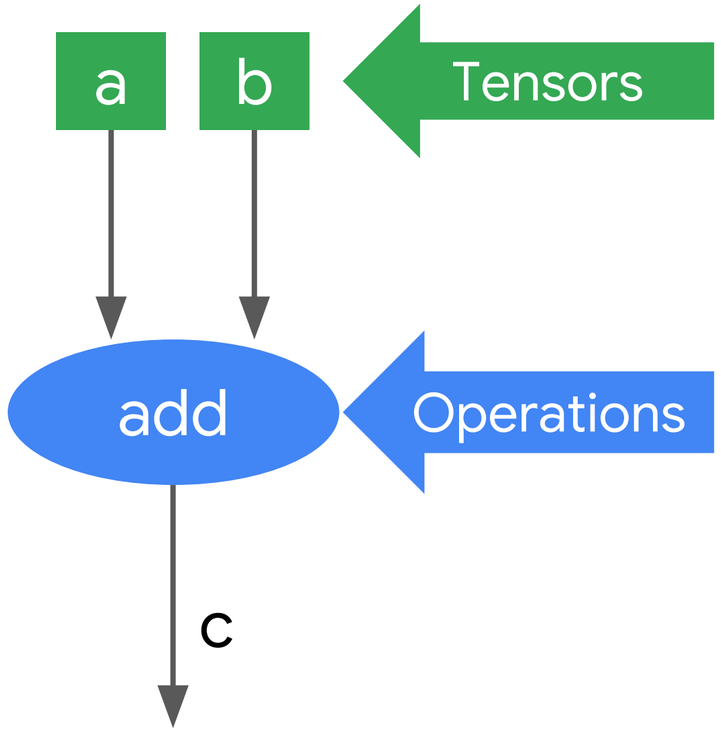
Build

```
...
c = tf.add(a, b)
```

The Python API lets you build and run Directed Graphs

Build

```
...
c = tf.add(a, b)
```



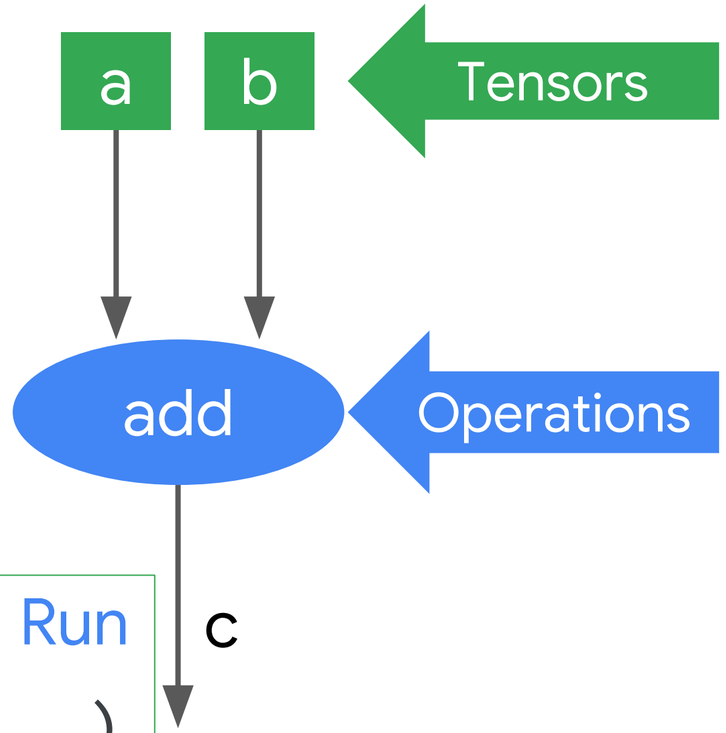
The Python API lets you build and run Directed Graphs

Build

```
...
c = tf.add(a, b)
```

Run

```
session = tf.Session()
numpy_c = session.run(c, feed_dict= ....)
```



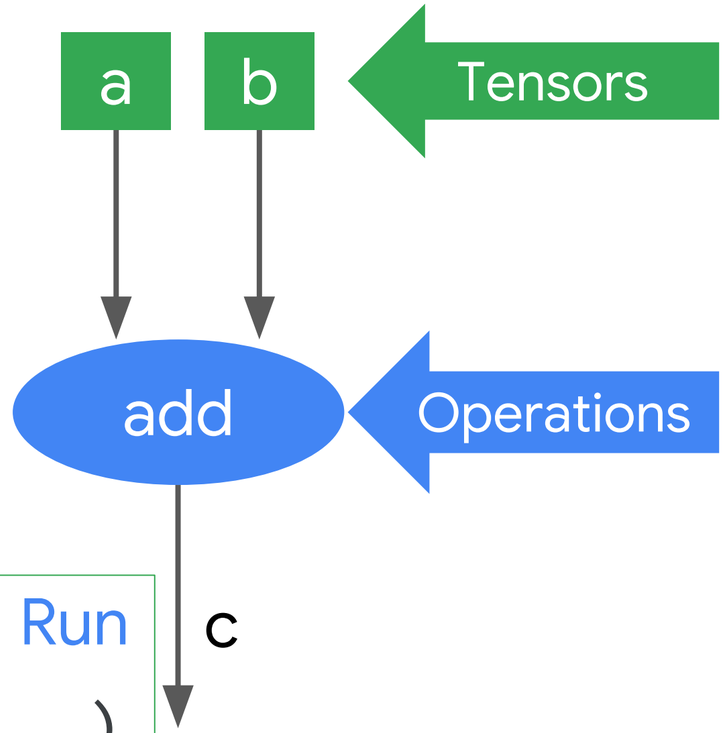
The Python API lets you build and run Directed Graphs

Build

```
...
c = tf.add(a, b)
```

Run

```
session = tf.Session()
numpy_c = session.run(c, feed_dict= ....)
```



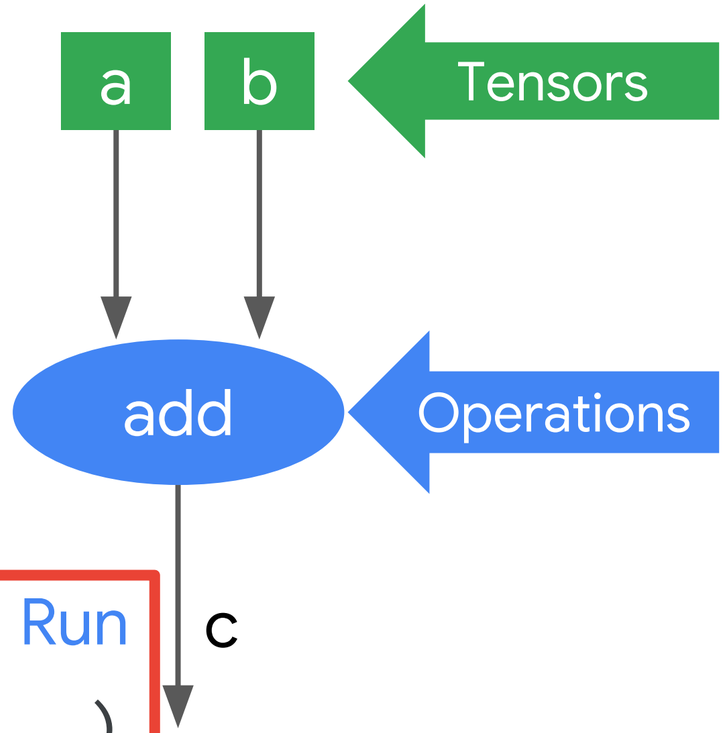
The Python API lets you build and run Directed Graphs

Build

```
...
c = tf.add(a, b)
```

Run

```
session = tf.Session()
numpy_c = session.run(c, feed_dict= ....)
```



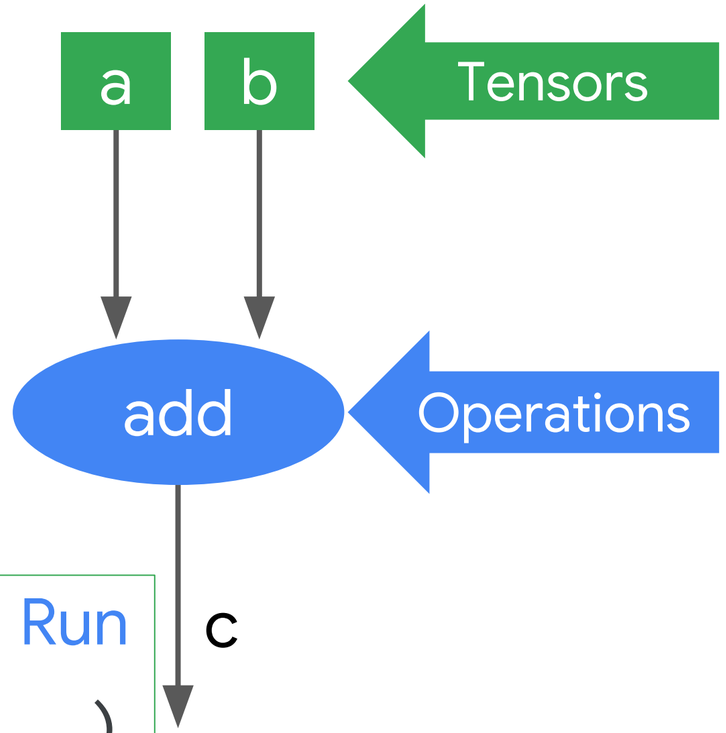
The Python API lets you build and run Directed Graphs

Build

```
...
c = tf.add(a, b)
```

Run

```
session = tf.Session()
numpy_c = session.run(c, feed_dict= ....)
```



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

Build

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

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

Run

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

```
[8 2 10]
```

*tf.eager, however, allows you to execute operations imperatively link

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

Build

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

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

Run

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

```
[8 2 10]
```

*tf.eager, however, allows you to execute operations imperatively link

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

Build

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

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

Run

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

```
[8 2 10]
```

*tf.eager, however, allows you to execute operations imperatively link

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

Build

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

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

Run

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

```
[8 2 10]
```

*tf.eager, however, allows you to execute operations imperatively link

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

Build

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

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

Run

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

```
[8 2 10]
```

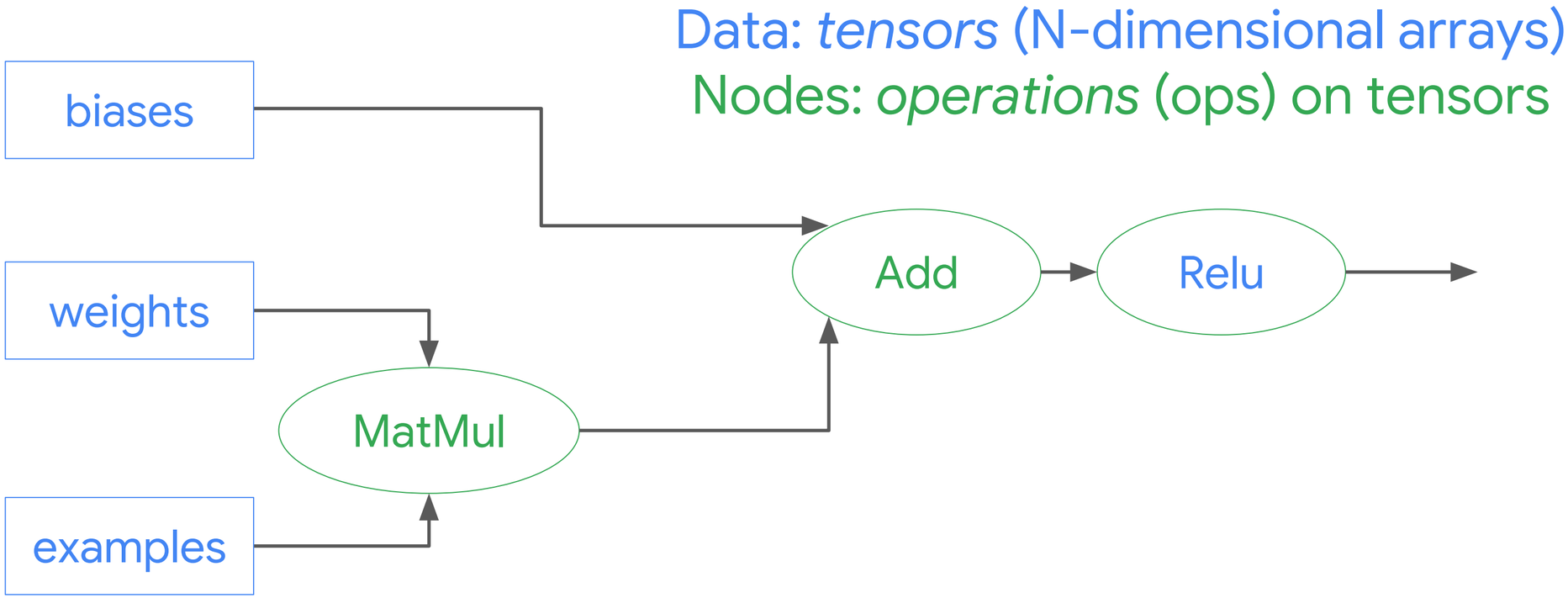
*tf.eager, however, allows you to execute operations imperatively link

Google Cloud

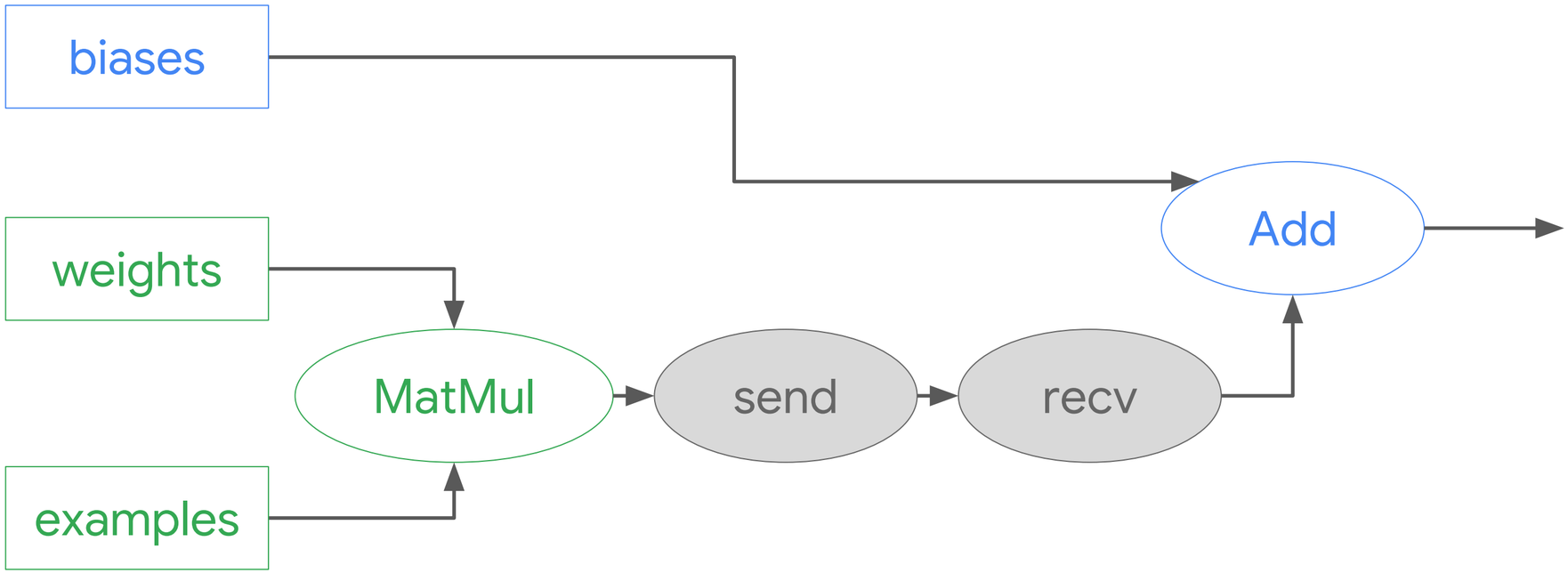
Graph and Session

Lak Lakshmanan

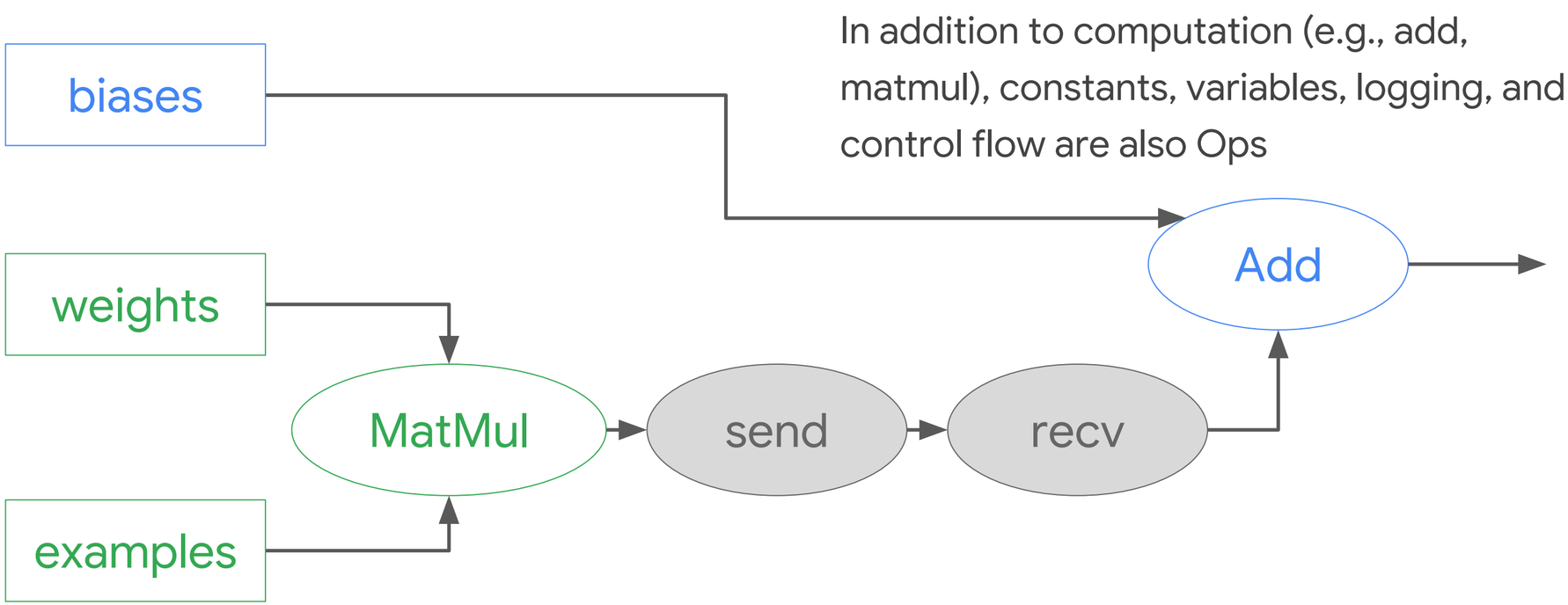
Graphs can be processed, compiled, remotely executed, and assigned to devices



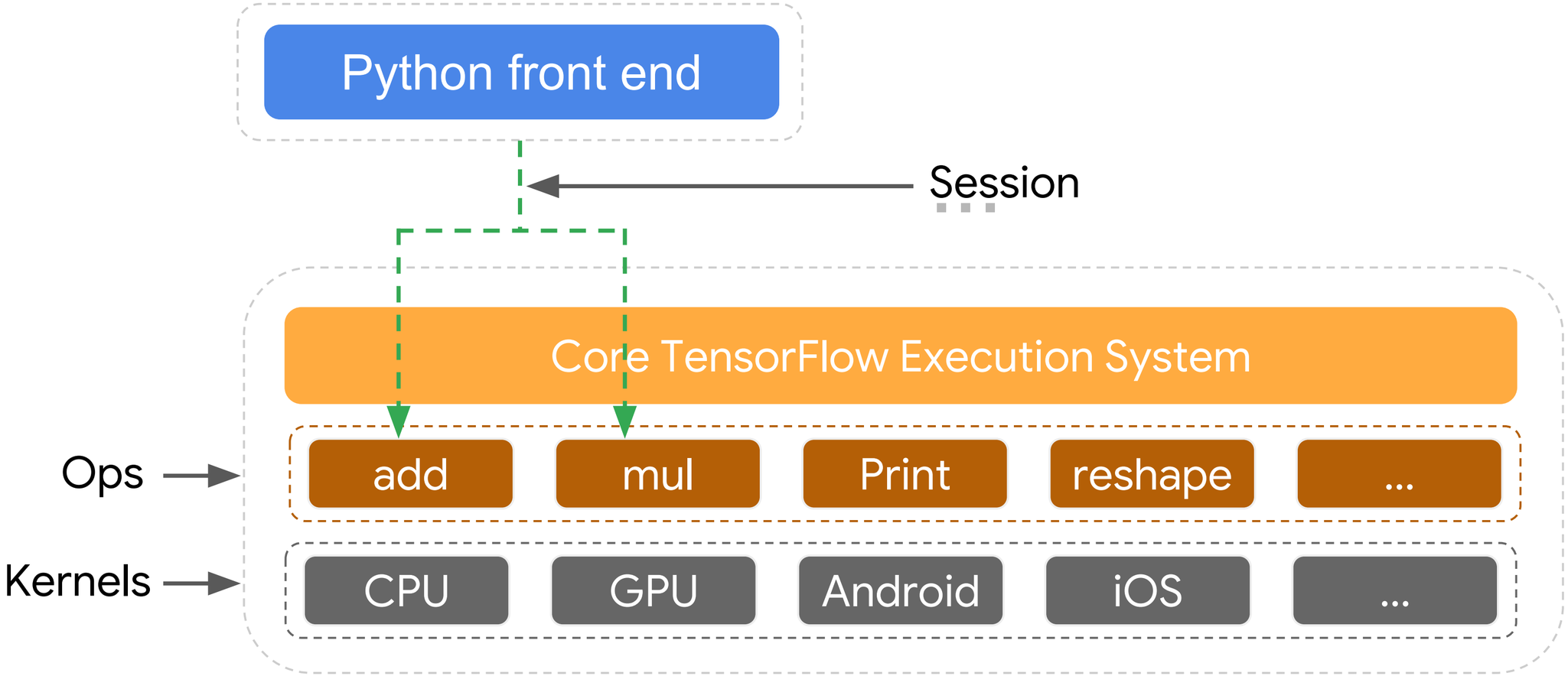
Graphs can be processed, compiled, remotely executed, and assigned to devices



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



Session allows TensorFlow to cache and distribute computation



Execute TensorFlow graphs by calling run() on a tf.Session

```
import tensorflow as tf

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
z = tf.add(x, y)

with tf.Session() as sess:
 print sess.run(z)
```

```
[4 7 10]
```

Execute TensorFlow graphs by calling run() on a tf.Session

```
import tensorflow as tf

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
z = tf.add(x, y)

with tf.Session() as sess:
 print sess.run(z)
```

```
[4 7 10]
```

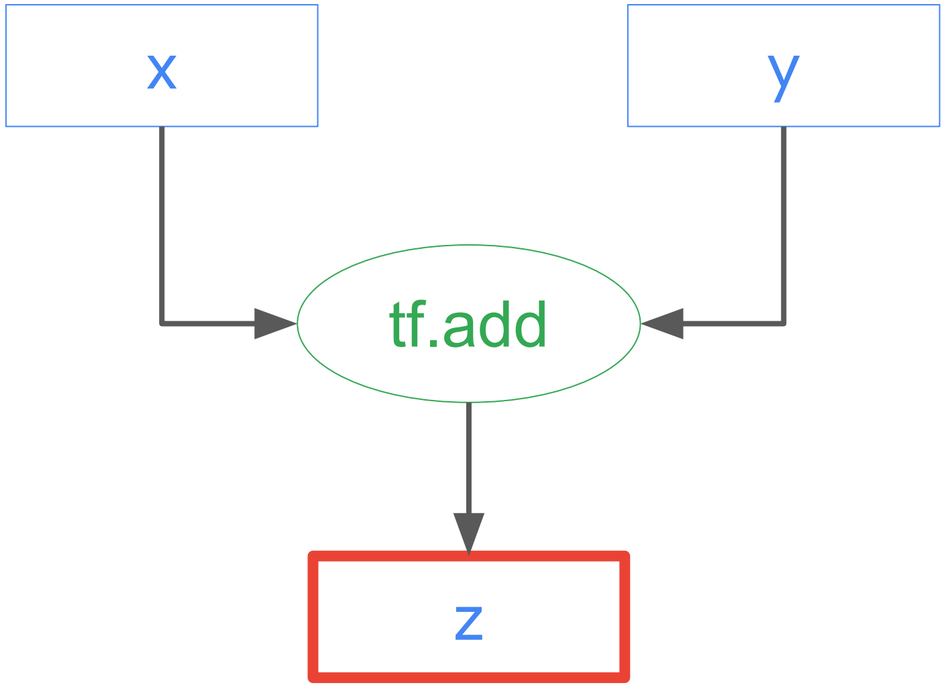
Execute TensorFlow graphs by calling run() on a tf.Session

```
import tensorflow as tf

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
z = tf.add(x, y)

with tf.Session() as sess:
 print sess.run(z)
```

```
[4 7 10]
```



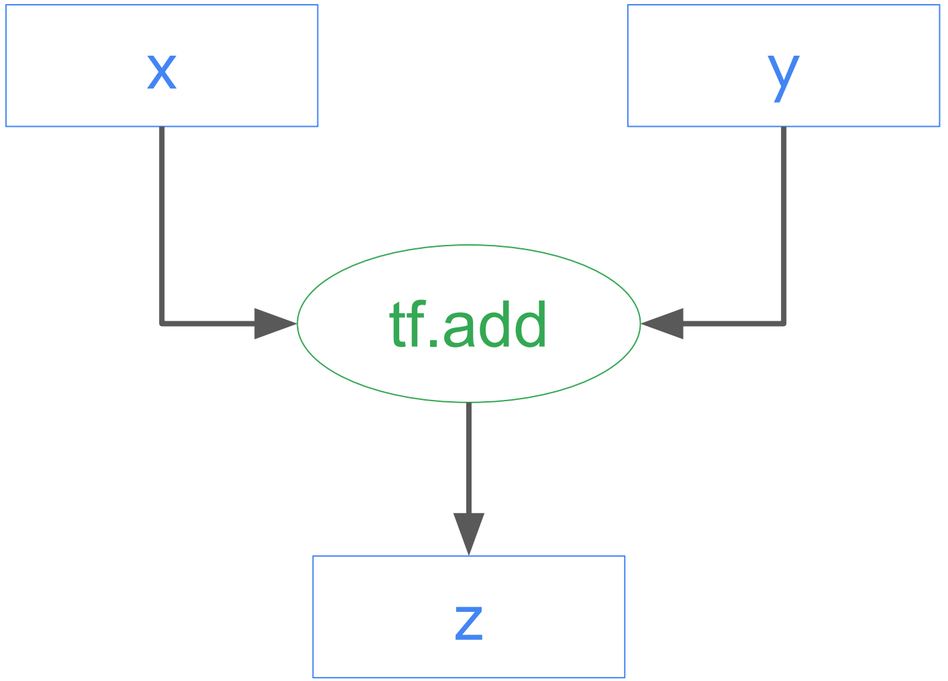
Execute TensorFlow graphs by calling run() on a tf.Session

```
import tensorflow as tf

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
z = tf.add(x, y)

with tf.Session() as sess:
 print sess.run(z)
```

```
[4 7 10]
```



Evaluating a tensor is a shortcut to calling run() on the graph's default session

```
import tensorflow as tf

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
z = tf.add(x, y)

with tf.Session() as sess:
 print z.eval()
```

```
[4 7 10]
```

It is possible to evaluate a list of tensors

```
import tensorflow as tf

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
z1 = tf.add(x, y)
z2 = x * y
z3 = z2 - z1

with tf.Session() as sess:
  a1, a3 = sess.run([z1, z3])
  print a1
  print a3
```

```
[ 4  7 10]
[-1  3 11]
```

Shortcuts to common arithmetic operators

run() accepts a list of tensors

It is possible to evaluate a list of tensors

```
import tensorflow as tf

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
z1 = tf.add(x, y)
z2 = x * y
z3 = z2 - z1

with tf.Session() as sess:
  a1, a3 = sess.run([z1, z3])
  print a1
  print a3
```

```
[ 4  7 10]
[-1  3 11]
```

Shortcuts to common arithmetic operators

run() accepts a list of tensors

TensorFlow in Eager mode makes it easier to try out things, but is not recommended for production code

```
import tensorflow as tf
from tensorflow.contrib.eager.python import tfe

tfe.enable_eager_execution()

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
print (x-y)

tf.Tensor([2 3 4], shape=(3,), dtype=int32)
```

Import tf.eager

Call exactly once

Note the value being printed

TensorFlow in Eager mode makes it easier to try out things, but is not recommended for production code

```
import tensorflow as tf
from tensorflow.contrib.eager.python import tfe

tfe.enable_eager_execution()

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
print (x-y)

tf.Tensor([2 3 4], shape=(3,), dtype=int32)
```

Import tf.eager

Call exactly once

Note the value being printed

TensorFlow in Eager mode makes it easier to try out things, but is not recommended for production code

```
import tensorflow as tf
from tensorflow.contrib.eager.python import tfe

tfe.enable_eager_execution()

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
print (x-y)

tf.Tensor([2 3 4], shape=(3,), dtype=int32)
```

Import tf.eager

Call exactly once

TensorFlow in Eager mode makes it easier to try out things, but is not recommended for production code

```
import tensorflow as tf
from tensorflow.contrib.eager.python import tfe

tfe.enable_eager_execution()

x = tf.constant([3, 5, 7])
y = tf.constant([1, 2, 3])
print (x-y)

tf.Tensor([2 3 4], shape=(3,), dtype=int32)
```

Import tf.eager

Call exactly once

Note the value being printed

You can write the graph out using tf.summary.FileWriter

```
import tensorflow as tf

x = tf.constant([3, 5, 7], name="x")
y = tf.constant([1, 2, 3], name="y")
z1 = tf.add(x, y, name="z1")
z2 = x * y
z3 = z2 - z1

with tf.Session() as sess:
  with tf.summary.FileWriter('summaries', sess.graph) as writer:
      a1, a3 = sess.run([z1, z3])
```

```
!ls summaries

events.out.tfevents.1517032067.e7cbb0325e48
```

Name the tensors and the operations

Write the session graph to a summary directory

You can write the graph out using tf.summary.FileWriter

```
import tensorflow as tf

x = tf.constant([3, 5, 7], name="x")
y = tf.constant([1, 2, 3], name="y")
z1 = tf.add(x, y, name="z1")
z2 = x * y
z3 = z2 - z1

with tf.Session() as sess:
  with tf.summary.FileWriter('summaries', sess.graph) as writer:
      a1, a3 = sess.run([z1, z3])
```

```
!ls summaries

events.out.tfevents.1517032067.e7cbb0325e48
```

Name the tensors and the operations

Write the session graph to a summary directory

The graph can be visualized in TensorBoard

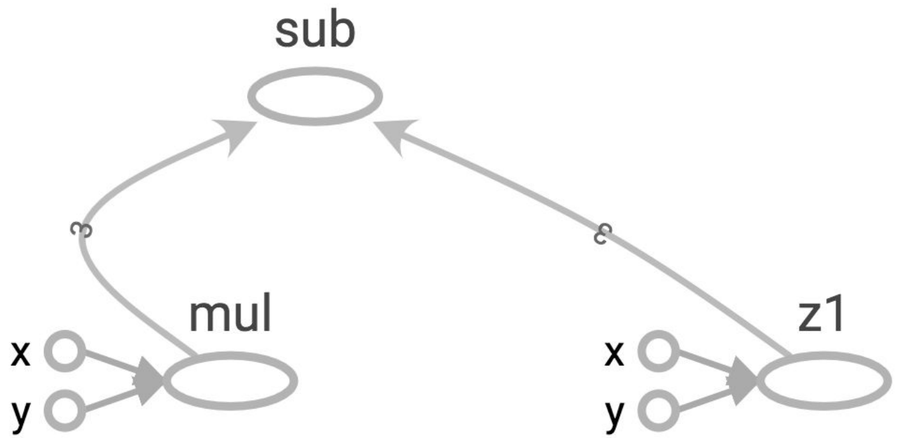
```
from google.datalab.ml import TensorBoard
TensorBoard().start('./summaries')
```

TensorBoard was started successfully with pid 13045. Click here to access it.

The graph can be visualized in TensorBoard

```
x = tf.constant([3, 5, 7], name="x")
y = tf.constant([1, 2, 3], name="y")
z1 = tf.add(x, y, name="z1")
z2 = x * y
z3 = z2 - z1
```

GRAPHS INAC



You can also write to gs:// and start TensorBoard from CloudShell

1 Run the following command in Cloud Shell to start TensorBoard:

```
tensorboard --port 8080 --logdir gs://${BUCKET}/${SUMMARY_DIR}
```

You can also write to gs:// and start TensorBoard from CloudShell

2 To open a new browser window, select **Preview on port 8080** from the **Web preview** menu in the top-right corner of the Cloud Shell toolbar.

In the new window, you can use TensorBoard to see the training summary and the visualized network graph.

You can also write to gs:// and start TensorBoard from CloudShell

3 Press Control+C to stop TensorBoard in Cloud Shell.

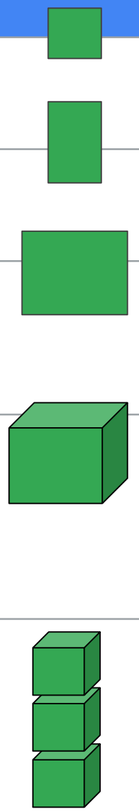
https://cloud.google.com/ml-engine/docs/distributed-tensorflow-mnist-cloud-datalab

Google Cloud

Tensor and Variable

Lak Lakshmanan

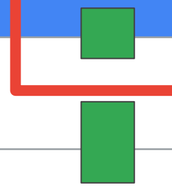
A tensor is an N-dimensional array of data

	Common name	Rank (Dimension)	Example	Shape of example
	Scalar	0	`x = tf.constant(3)`	()
	Vector	1	`x = tf.constant([3, 5, 7])`	(3,)
	Matrix	2	`x = tf.constant([[3, 5, 7],` `[4, 6, 8]])`	(2, 3)

	3D Tensor	3	`tf.constant([[[3, 5, 7],[4, 6, 8]],`
			`[[1, 2, 3],[4, 5, 6]]`
			`])`
				(2, 2, 3)

	nD Tensor	n	`x1 = tf.constant([2, 3, 4])`
			`x2 = tf.stack([x1, x1])`
			`x3 = tf.stack([x2, x2, x2, x2])`
			`x4 = tf.stack([x3, x3])`
			`...`
				(3,)
				(2, 3)
				(4, 2, 3)
				(2, 4, 2, 3)

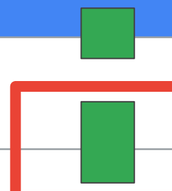
A tensor is an N-dimensional array of data

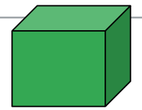
	Common name	Rank (Dimension)	Example	Shape of example
	Scalar	0	`x = tf.constant(3)`	()
	Vector	1	`x = tf.constant([3, 5, 7])`	(3,)
	Matrix	2	`x = tf.constant([[3, 5, 7],` `[4, 6, 8]])`	(2, 3)

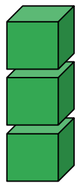
	3D Tensor	3	`tf.constant([[[3, 5, 7],[4, 6, 8]],`
			`[[1, 2, 3],[4, 5, 6]]`
			`])`
				(2, 2, 3)

	nD Tensor	n	`x1 = tf.constant([2, 3, 4])`
			`x2 = tf.stack([x1, x1])`
			`x3 = tf.stack([x2, x2, x2, x2])`
			`x4 = tf.stack([x3, x3])`
			`...`
				(3,)
				(2, 3)
				(4, 2, 3)
				(2, 4, 2, 3)

A tensor is an N-dimensional array of data

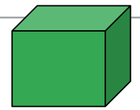
	Common name	Rank (Dimension)	Example	Shape of example
	Scalar	0	`x = tf.constant(3)`	()
	Vector	1	`x = tf.constant([3, 5, 7])`	(3,)
	Matrix	2	`x = tf.constant([[3, 5, 7],` `[4, 6, 8]])`	(2, 3)

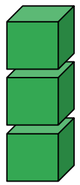
	3D Tensor	3	`tf.constant([[[3, 5, 7],[4, 6, 8]],`
			`[[1, 2, 3],[4, 5, 6]]`
			`])`
				(2, 2, 3)

	nD Tensor	n	`x1 = tf.constant([2, 3, 4])`
			`x2 = tf.stack([x1, x1])`
			`x3 = tf.stack([x2, x2, x2, x2])`
			`x4 = tf.stack([x3, x3])`
			`...`
				(3,)
				(2, 3)
				(4, 2, 3)
				(2, 4, 2, 3)

A tensor is an N-dimensional array of data

	Common name	Rank (Dimension)	Example	Shape of example
	Scalar	0	`x = tf.constant(3)`	()
	Vector	1	`x = tf.constant([3, 5, 7])`	(3,)
	Matrix	2	`x = tf.constant([[3, 5, 7],` `[4, 6, 8]])`	(2, 3)

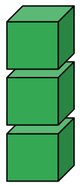
	3D Tensor	3	`tf.constant([[[3, 5, 7],[4, 6, 8]],`
			`[[1, 2, 3],[4, 5, 6]]`
			`])`
				(2, 2, 3)

	nD Tensor	n	`x1 = tf.constant([2, 3, 4])`
			`x2 = tf.stack([x1, x1])`
			`x3 = tf.stack([x2, x2, x2, x2])`
			`x4 = tf.stack([x3, x3])`
			`...`
				(3,)
				(2, 3)
				(4, 2, 3)
				(2, 4, 2, 3)

A tensor is an N-dimensional array of data

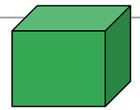
	Common name	Rank (Dimension)	Example	Shape of example
	Scalar	0	`x = tf.constant(3)`	()
	Vector	1	`x = tf.constant([3, 5, 7])`	(3,)
	Matrix	2	`x = tf.constant([[3, 5, 7],` `[4, 6, 8]])`	(2, 3)

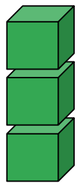
	3D Tensor	3	`tf.constant([[[3, 5, 7],[4, 6, 8]],`
			`[[1, 2, 3],[4, 5, 6]]`
			`])`
				(2, 2, 3)

	nD Tensor	n	`x1 = tf.constant([2, 3, 4])`
			`x2 = tf.stack([x1, x1])`
			`x3 = tf.stack([x2, x2, x2, x2])`
			`x4 = tf.stack([x3, x3])`
			`...`
				(3,)
				(2, 3)
				(4, 2, 3)
				(2, 4, 2, 3)

A tensor is an N-dimensional array of data

	Common name	Rank (Dimension)	Example	Shape of example
	Scalar	0	`x = tf.constant(3)`	()
	Vector	1	`x = tf.constant([3, 5, 7])`	(3,)
	Matrix	2	`x = tf.constant([[3, 5, 7],` `[4, 6, 8]])`	(2, 3)

	3D Tensor	3	`tf.constant([[[3, 5, 7],[4, 6, 8]],`
			`[[1, 2, 3],[4, 5, 6]]`
			`])`
				(2, 2, 3)

	nD Tensor	n	`x1 = tf.constant([2, 3, 4])`
			`x2 = tf.stack([x1, x1])`
			`x3 = tf.stack([x2, x2, x2, x2])`
			`x4 = tf.stack([x3, x3])`
			`...`
				(3,)
				(2, 3)
				(4, 2, 3)
				(2, 4, 2, 3)

Tensors can be sliced

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = x[:, 1]
with tf.Session() as sess:
  print y.eval()
```

```
[5 6]
```

Tensors can be sliced

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = x[:, 1]
with tf.Session() as sess:
  print y.eval()
```

```
[5 6]
```

Tensors can be sliced

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = x[:, 1]
with tf.Session() as sess:
  print y.eval()
```

```
[5 6]
```

Tensors can be sliced

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = x[:, 1]
with tf.Session() as sess:
  print y.eval()
```

```
[5 6]
```

Quiz:
(1) What would x[1, :] do?

(2) How about x[1, 0:2]?

Tensors can be sliced

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = x[:, 1]
with tf.Session() as sess:
  print y.eval()
```

```
[5 6]
```

Quiz:
(1) What would x[1, :] do?
    [4,6,8]

(2) How about x[1, 0:2]?
    [4,6,8]

Tensors can be reshaped

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])
with tf.Session() as sess:
  print y.eval()
```

```
[[3 5]
 [7 4]
 [6 8]]
```

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])[1, :]
with tf.Session() as sess:
  print y.eval()
```

```
[7 4]
```

Tensors can be reshaped

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])
with tf.Session() as sess:
  print y.eval()
```

```
[[3 5]
 [7 4]
 [6 8]]
```

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])[1, :]
with tf.Session() as sess:
  print y.eval()
```

```
[7 4]
```

Tensors can be reshaped

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])
with tf.Session() as sess:
  print y.eval()
```

```
[[3 5]
 [7 4]
 [6 8]]
```

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])[1, :]
with tf.Session() as sess:
  print y.eval()
```

```
[7 4]
```

Tensors can be reshaped

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])
with tf.Session() as sess:
  print y.eval()
```

```
[[3 5]
 [7 4]
 [6 8]]
```

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])[1, :]
with tf.Session() as sess:
  print y.eval()
```

```
[7 4]
```

Tensors can be reshaped

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])
with tf.Session() as sess:
  print y.eval()
```

```
[[3 5]
 [7 4]
 [6 8]]
```

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])[1, :]
with tf.Session() as sess:
  print y.eval()
```

```
[7 4]
```

Tensors can be reshaped

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])
with tf.Session() as sess:
  print y.eval()
```

```
[[3 5]
 [7 4]
 [6 8]]
```

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])[1, :]
with tf.Session() as sess:
  print y.eval()
```

```
[7 4]
```

Tensors can be reshaped

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])
with tf.Session() as sess:
  print y.eval()
```

```
[[3 5]
 [7 4]
 [6 8]]
```

```
import tensorflow as tf
x = tf.constant([[3, 5, 7],
                 [4, 6, 8]])
y = tf.reshape(x, [3, 2])[1, :]
with tf.Session() as sess:
  print y.eval()
```

```
[7 4]
```

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

“Training loop” of 5 updates to weights

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
with tf.Session() as sess:
  preds = train_loop(tf.constant([[3.2, 5.1, 7.2],[4.3, 6.2, 8.3]])) # 2 x 3 matrix
  tf.global_variables_initializer().run()
  for i in xrange(len(preds)):
      print "{}:{}".format( i, preds[i].eval() )
```

```
0:[[-0.53224635 -1.4080029  -2.3759441 ]]
1:[[ 0.21775389 -0.27800274 -0.82594395]]
2:[[0.96775365 0.8519969  0.72405624]]
3:[[1.7177541 1.981997  2.2740564]]
4:[[2.4677541 3.1119976 3.8240576]]
```

Multiplying [1,~~2~~] x [~~2~~,3] yields a [1,3] matrix

Initialize all variables

Print [1,3] matrix at each of the 5 iterations

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
with tf.Session() as sess:
  preds = train_loop(tf.constant([[3.2, 5.1, 7.2],[4.3, 6.2, 8.3]])) # 2 x 3 matrix
  tf.global_variables_initializer().run()
  for i in xrange(len(preds)):
      print "{}:{}".format( i, preds[i].eval() )
```

```
0:[[-0.53224635 -1.4080029  -2.3759441 ]]
1:[[ 0.21775389 -0.27800274 -0.82594395]]
2:[[0.96775365 0.8519969  0.72405624]]
3:[[1.7177541 1.981997  2.2740564]]
4:[[2.4677541 3.1119976 3.8240576]]
```

Multiplying [1,~~2~~] x [~~2~~,3] yields a [1,3] matrix

Initialize all variables

Print [1,3] matrix at each of the 5 iterations

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
with tf.Session() as sess:
  preds = train_loop(tf.constant([[3.2, 5.1, 7.2],[4.3, 6.2, 8.3]])) # 2 x 3 matrix
  tf.global_variables_initializer().run()
  for i in xrange(len(preds)):
      print "{}:{}".format( i, preds[i].eval() )
```

```
0:[[-0.53224635 -1.4080029  -2.3759441 ]]
1:[[ 0.21775389 -0.27800274 -0.82594395]]
2:[[0.96775365 0.8519969  0.72405624]]
3:[[1.7177541 1.981997  2.2740564]]
4:[[2.4677541 3.1119976 3.8240576]]
```

Multiplying [1,~~2~~] x [~~2~~,3] yields a [1,3] matrix

Initialize all variables

Print [1,3] matrix at each of the 5 iterations

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

```
with tf.Session() as sess:
  preds = train_loop(tf.constant([[3.2, 5.1, 7.2],[4.3, 6.2, 8.3]])) # 2 x 3 matrix
  tf.global_variables_initializer().run()
  for i in xrange(len(preds)):
      print "{}:{}".format( i, preds[i].eval() )
```

Multiplying [1,~~2] x [2~~,3] yields a [1,3] matrix

Initialize all variables

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

```
with tf.Session() as sess:
  preds = train_loop(tf.constant([[3.2, 5.1, 7.2],[4.3, 6.2, 8.3]])) # 2 x 3 matrix
  tf.global_variables_initializer().run()
  for i in xrange(len(preds)):
      print "{}:{}".format( i, preds[i].eval() )
```

Multiplying [1,~~2] x [2~~,3] yields a [1,3] matrix

Initialize all variables

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

```
with tf.Session() as sess:
  preds = train_loop(tf.constant([[3.2, 5.1, 7.2],[4.3, 6.2, 8.3]])) # 2 x 3 matrix
  tf.global_variables_initializer().run()
  for i in xrange(len(preds)):
      print "{}:{}".format( i, preds[i].eval() )
```

Multiplying [1,~~2~~] x [~~2~~,3] yields a [1,3] matrix

Initialize all variables

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

```
with tf.Session() as sess:
  preds = train_loop(tf.constant([[3.2, 5.1, 7.2],[4.3, 6.2, 8.3]])) # 2 x 3 matrix
  tf.global_variables_initializer().run()
  for i in xrange(len(preds)):
      print "{}:{}".format( i, preds[i].eval() )
```

Multiplying [1,~~2] x [2~~,3] yields a [1,3] matrix

Initialize all variables

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

```
with tf.Session() as sess:
  preds = train_loop(tf.constant([[3.2, 5.1, 7.2],[4.3, 6.2, 8.3]])) # 2 x 3 matrix
  tf.global_variables_initializer().run()
  for i in xrange(len(preds)):
      print "{}:{}".format( i, preds[i].eval() )
```

Multiplying [1,~~2] x [2~~,3] yields a [1,3] matrix

Initialize all variables

A variable is a tensor whose value is initialized and then typically changed as the program runs

```
def forward_pass(w, x):
    return tf.matmul(w, x)

def train_loop(x, niter=5):
  with tf.variable_scope("model", reuse=tf.AUTO_REUSE):
    w = tf.get_variable("weights",
                        shape=(1,2),  # 1 x 2 matrix
                        initializer=tf.truncated_normal_initializer(),
                        trainable=True)
  preds = []
  for k in xrange(niter):
    preds.append(forward_pass(w, x))
    w = w + 0.1 # "gradient update"
  return preds
```

Create variable, specifying how to init and whether it can be tuned

"Training loop" of 5 updates to weights

```
with tf.Session() as sess:
  preds = train_loop(tf.constant([[3.2, 5.1, 7.2],[4.3, 6.2, 8.3]])) # 2 x 3 matrix
  tf.global_variables_initializer().run()
  for i in xrange(len(preds)):
      print "{}:{}".format( i, preds[i].eval() )
```

Multiplying [1,~~2] x [2~~,3] yields a [1,3] matrix

Initialize all variables

Placeholders allow you to feed in values, such as by reading from a text file

```
import tensorflow as tf

a = tf.placeholder("float", None)
b = a * 4
print a
with tf.Session() as session:
    print(session.run(b, feed_dict={a: [1,2,3]}))
```

Placeholders allow you to feed in values, such as by reading from a text file

```
import tensorflow as tf

a = tf.placeholder("float", None)
b = a * 4
print a
with tf.Session() as session:
    print(session.run(b, feed_dict={a: [1,2,3]}))
```

```
Tensor("Placeholder
:0", dtype=float32)
```

Placeholders allow you to feed in values, such as by reading from a text file

```
import tensorflow as tf

a = tf.placeholder("float", None)
b = a * 4
print a
with tf.Session() as session:
    print(session.run(b, feed_dict={a: [1,2,3]}))
```

```
Tensor("Placeholder
:0", dtype=float32)
```

Placeholders allow you to feed in values, such as by reading from a text file

```
import tensorflow as tf

a = tf.placeholder("float", None)
b = a * 4
print a
with tf.Session() as session:
    print(session.run(b, feed_dict={a: [1,2,3]}))
```

```
Tensor("Placeholder
:0", dtype=float32)
```

```
[  4.   8.  12.]
```

LAB

Writing low-level TensorFlow programs

Lak Lakshmanan

Lab: Writing low-level TensorFlow programs

In this lab, you will learn how the TensorFlow Python API works

1. Building a graph
2. Running a graph
3. Feeding values into a graph
4. Finding the area of a triangle by using TensorFlow

Google Cloud

Debugging TensorFlow programs

Lak Lakshmanan

Debugging TensorFlow programs is similar to debugging any piece of software

1

Read Error Messages

2

Isolate the method in question

3

Send made-up data into the method

4

Know how to solve common problems

Read error messages to understand the problem


```
<ipython-input-4-d03f16eba551> in
some_method(data)
      2   a = data[:,0:2]
      3   c = data[:,1]
----> 4   s = (a + c)
      5   return tf.sqrt(tf.matmul(s,
tf.transpose(s)))
```

Read error messages to understand the problem

output:

```
ValueError: Dimensions must be equal,
but are 2 and 4 for 'add' (op: 'Add')
with input shapes: [4,2], [4].
```

Isolate the method in question

2

```
def some_method(data):
  a = data[:,0:2]
  c = data[:,1]
  s = (a + c)
  return tf.sqrt(tf.matmul(s,
tf.transpose(s)))
```

Call the problematic method with fake data

3


```
def some_method(data):
  a = data[:,0:2]
  print a.get_shape()
  c = data[:,1]
  print c.get_shape()
  s = (a + c)
  return tf.sqrt(tf.matmul(s, tf.transpose(s)))
with tf.Session() as sess:
  fake_data = tf.constant([
      [5.0, 3.0, 7.1],
      [2.3, 4.1, 4.8],
      [2.8, 4.2, 5.6],
      [2.9, 8.3, 7.3]
    ])
  print sess.run(some_method(fake_data))
```

Call the problematic method with fake data

3


```
def some_method(data):
  a = data[:,0:2]
  print a.get_shape()
  c = data[:,1]
  print c.get_shape()
  s = (a + c)
  return tf.sqrt(tf.matmul(s, tf.transpose(s)))
with tf.Session() as sess:
  fake_data = tf.constant([
      [5.0, 3.0, 7.1],
      [2.3, 4.1, 4.8],
      [2.8, 4.2, 5.6],
      [2.9, 8.3, 7.3]
    ])
  print sess.run(some_method(fake_data))
```

Call the problematic method with fake data

3


```
def some_method(data):
  a = data[:,0:2]
  print a.get_shape()
  c = data[:,1]
  print c.get_shape()
  s = (a + c)
  return tf.sqrt(tf.matmul(s, tf.transpose(s)))
with tf.Session() as sess:
  fake_data = tf.constant([
      [5.0, 3.0, 7.1],
      [2.3, 4.1, 4.8],
      [2.8, 4.2, 5.6],
      [2.9, 8.3, 7.3]
    ])
  print sess.run(some_method(fake_data))
```

```
(4, 2)

(4,)
```

Why does the addition fail?

Know how to solve common problems

4

Tensor shape

Scalar-vector mismatch

Data type mismatch

The most common problem tends to be tensor shape

```
def some_method(data):
  a = data[:,0:2]
  print a.get_shape()
  c = data[:,1]
  print c.get_shape()
  s = (a + c)
  return tf.sqrt(tf.matmul(s, tf.transpose(s)))
with tf.Session() as sess:
  fake_data = tf.constant([
      [5.0, 3.0, 7.1],
      [2.3, 4.1, 4.8],
      [2.8, 4.2, 5.6],
      [2.9, 8.3, 7.3]
    ])
  print sess.run(some_method(fake_data))
```

```
(4, 2)

(4,)
```

Why does the addition fail?

The most common problem tends to be tensor shape

```
c = data[:,1:3]
```

```
def some_method(data):
    a = data[:,0:2]
    print a.get_shape()
    c = data[:,1:3]
    assert len(c.get_shape()) == 2
    s = (a+c)
    return tf.sqrt(tf.matmul(s,tf.transpose(s)))

with tf.Session() as sess:
    fake_data=tf.constant([
        [5.0, 3.0, 7.1],
        [2.3, 4.1, 4.8],
        [2.8, 4.2, 5.6],
        [2.9, 8.3, 7.3]
    ])
    print sess.run(some_method(fake_data))
```

Add an assert

output:

```
(4, 2)
[[ 12.88448715  11.87813091  12.44909573  15.72132301]
 [ 11.87813091  10.96220779  11.48999596  14.50930595]
 [ 12.44909573  11.48999596  12.04325485  15.20789242]
 [ 15.72132301  14.50930595  15.20789242  19.20416641]]
```

Shape problems also happen because of batch size or because you have a scalar when a vector is needed (or vice versa)

```
Tensor("Placeholde
r_4:0", shape=(?,
3), dtype=float32)
```

```
n_input = tf.constant([3])
X = tf.placeholder(tf.float32, [None, n_input])
print X
…
fake_data = tf.constant([5.0, 3.0, 7.1])
```

output

```
ValueError: Cannot feed value of shape (3,)

Tensor("Placeholder_4:0", shape=(?, 3), dtype=float32)
```

Shape problems also happen because of batch size or because you have a scalar when a vector is needed (or vice versa)

```
Tensor("Placeholde
r_4:0", shape=(?,
3), dtype=float32)
```

```
n_input = tf.constant([3])
X = tf.placeholder(tf.float32, [None, n_input])
print X
…
fake_data = tf.constant([5.0, 3.0, 7.1])
```

output

```
ValueError: Cannot feed value of shape (3,)

Tensor("Placeholder_4:0", shape=(?, 3), dtype=float32)
```

Shape problems also happen because of batch size or because you have a scalar when a vector is needed (or vice versa)

```
Tensor("Placeholde
r_4:0", shape=(?,
3), dtype=float32)
```

```
n_input = tf.constant([3])
X = tf.placeholder(tf.float32, [None, n_input])
print X
…
fake_data = tf.constant([5.0, 3.0, 7.1])
```

output

```
ValueError: Cannot feed value of shape (3,)

Tensor("Placeholder_4:0", shape=(?, 3), dtype=float32)
```

Shape problems also happen because of batch size or because you have a scalar when a vector is needed (or vice versa)

```
Tensor("Placeholde
r_4:0", shape=(?,
3), dtype=float32)
```

```
n_input = tf.constant([3])
X = tf.placeholder(tf.float32, [None, n_input])
print X
…
fake_data = tf.constant([5.0, 3.0, 7.1])
```

output

```
ValueError: Cannot feed value of shape (3,)

Tensor("Placeholder_4:0", shape=(?, 3), dtype=float32)
```

Shape problems also happen because of batch size or because you have a scalar when a vector is needed (or vice versa)

```
Tensor("Placeholde
r_4:0", shape=(?,
3), dtype=float32)
```

```
n_input = tf.constant([3])
X = tf.placeholder(tf.float32, [None, n_input])
print X
…
fake_data = tf.constant([5.0, 3.0, 7.1])
```

output

```
ValueError: Cannot feed value of shape (3,)

Tensor("Placeholder_4:0", shape=(?, 3), dtype=float32)
```

Shape problems can often be fixed using ...

1. tf.reshape()
2. tf.expand_dims()
3. tf.slice()
4. tf.squeeze()

tf.expand_dims inserts a dimension of 1 into a tensor's shape

```
x = tf.constant([[3, 2],
                 [4, 5],
                 [6, 7]])
print "x.shape", x.shape
expanded = tf.expand_dims(x, 1)
print "expanded.shape", expanded.shape

with tf.Session() as sess:
  print "expanded:\n", expanded.eval()
```

tf.expand_dims()

output

```
x.shape (3, 2)
expanded.shape (3, 1, 2)


expanded:

[[[3 2]]

 [[4 5]]

 [[6 7]]]
```

tf.expand_dims inserts a dimension of 1 into a tensor's shape

```
x = tf.constant([[3, 2],
                 [4, 5],
                 [6, 7]])
print "x.shape", x.shape
expanded = tf.expand_dims(x, 1)
print "expanded.shape", expanded.shape

with tf.Session() as sess:
  print "expanded:\n", expanded.eval()
```

tf.expand_dims()

output

```
x.shape (3, 2)
expanded.shape (3, 1, 2)


expanded:

[[[3 2]]

 [[4 5]]

 [[6 7]]]
```

tf.expand_dims inserts a dimension of 1 into a tensor's shape

```
x = tf.constant([[3, 2],
                 [4, 5],
                 [6, 7]])
print "x.shape", x.shape
expanded = tf.expand_dims(x, 1)
print "expanded.shape", expanded.shape

with tf.Session() as sess:
  print "expanded:\n", expanded.eval()
```

tf.expand_dims()

output

```
x.shape (3, 2)
expanded.shape (3, 1, 2)


expanded:

[[[3 2]]

 [[4 5]]

 [[6 7]]]
```

tf.expand_dims inserts a dimension of 1 into a tensor's shape

```
x = tf.constant([[3, 2],
                 [4, 5],
                 [6, 7]])
print "x.shape", x.shape
expanded = tf.expand_dims(x, 1)
print "expanded.shape", expanded.shape

with tf.Session() as sess:
  print "expanded:\n", expanded.eval()
```

tf.expand_dims()

output

```
x.shape (3, 2)
expanded.shape (3, 1, 2)


expanded:
[[[3 2]]

 [[4 5]]

 [[6 7]]]
```

tf.slice extracts a slice from a tensor

```
x = tf.constant([[3, 2],
                 [4, 5],
                 [6, 7]])
print "x.shape", x.shape
sliced = tf.slice(x, [0, 1], [2, 1])
print "sliced.shape", sliced.shape

with tf.Session() as sess:
  print "sliced:\n", sliced.eval()
```

tf.slice() output

```
x.shape (3, 2)
sliced.shape (2, 1)
sliced:

[[2]
 [5]]
```

tf.slice extracts a slice from a tensor

```
x = tf.constant([[3, 2],
                 [4, 5],
                 [6, 7]])
print "x.shape", x.shape
sliced = tf.slice(x, [0, 1], [2, 1])
print "sliced.shape", sliced.shape

with tf.Session() as sess:
  print "sliced:\n", sliced.eval()
```

tf.slice() output

```
x.shape (3, 2)
sliced.shape (2, 1)
sliced:

[[2]
 [5]]
```

tf.slice extracts a slice from a tensor

```
x = tf.constant([[3, 2],
                 [4, 5],
                 [6, 7]])
print "x.shape", x.shape
sliced = tf.slice(x, [0, 1], [2, 1])
print "sliced.shape", sliced.shape

with tf.Session() as sess:
  print "sliced:\n", sliced.eval()
```

tf.slice() output

```
x.shape (3, 2)
sliced.shape (2, 1)
sliced:

[[2]
 [5]]
```

tf.squeeze removes dimensions of size 1 from the shape of a tensor

```
t = tf.constant([[[1],[2],[3],[4]],[[5],[6],[7],[8]]])
with tf.Session() as sess:
    print("t")
    print(sess.run(t))
    print("t squeezed")
    print(sess.run(tf.squeeze(t)))
```

output

```
t
[[[1]
  [2]
  [3]
  [4]]

 [[5]
  [6]
  [7]
  [8]]]
t squeezed
[[1 2 3 4]
 [5 6 7 8]]
```

tf.squeeze removes dimensions of size 1 from the shape of a tensor

```
t = tf.constant([[[1],[2],[3],[4]],[[5],[6],[7],[8]]])
with tf.Session() as sess:
    print("t")
    print(sess.run(t))
    print("t squeezed")
    print(sess.run(tf.squeeze(t)))
```

output

```
t
[[[1]
  [2]
  [3]
  [4]]

 [[5]
  [6]
  [7]
  [8]]]
t squeezed
[[1 2 3 4]
 [5 6 7 8]]
```

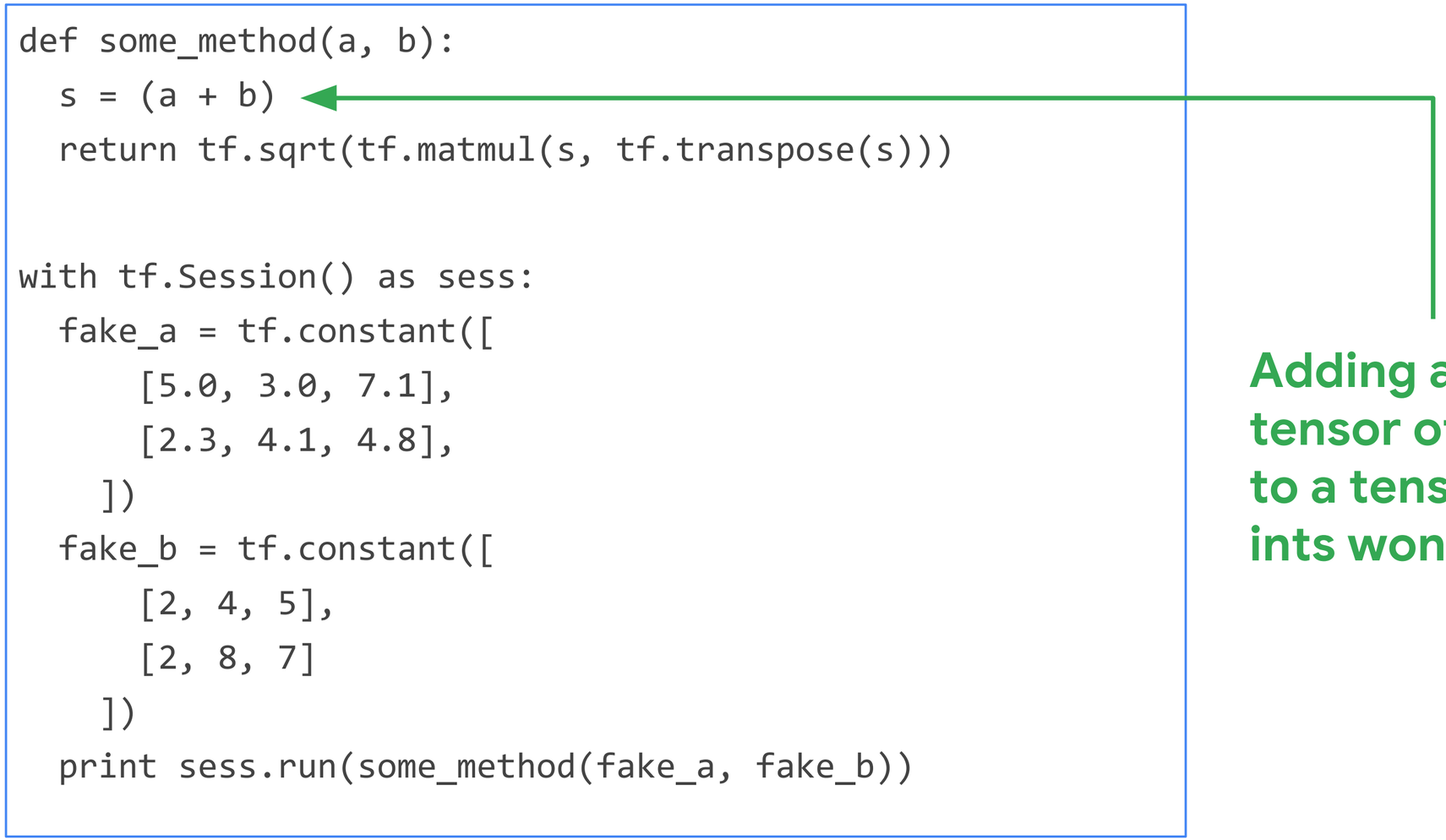
Another common problem is data type

```
ValueError: Tensor conversion
requested dtype float32 for
Tensor with dtype int32:
'Tensor("Const_34:0", shape=(2,
3), dtype=int32)'
```

The reason is because we are mixing types

```
def some_method(a, b):
  s = (a + b)
  return tf.sqrt(tf.matmul(s, tf.transpose(s)))


with tf.Session() as sess:
  fake_a = tf.constant([
      [5.0, 3.0, 7.1],
      [2.3, 4.1, 4.8],
    ])
  fake_b = tf.constant([
      [2, 4, 5],
      [2, 8, 7]
    ])
  print sess.run(some_method(fake_a, fake_b))
```



Adding a tensor of floats to a tensor of ints won't work

The reason is because we are mixing types

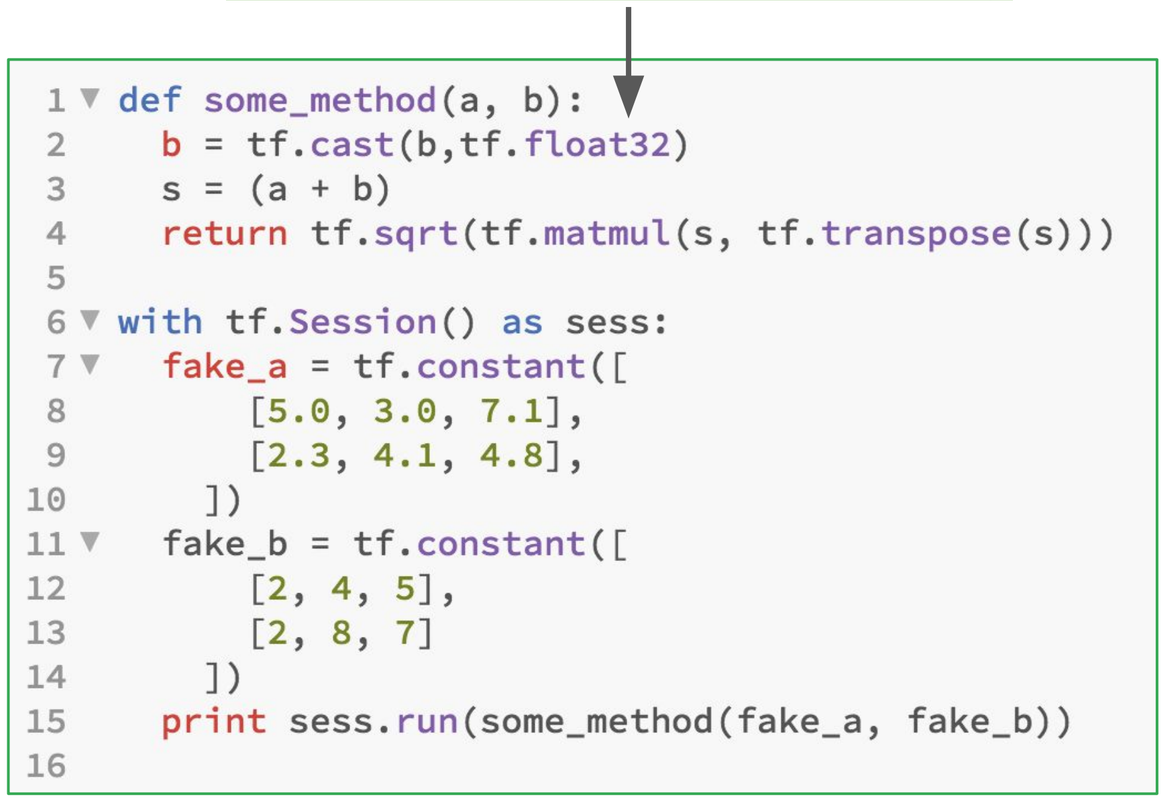
```
def some_method(a, b):
  s = (a + b)
  return tf.sqrt(tf.matmul(s, tf.transpose(s)))


with tf.Session() as sess:
  fake_a = tf.constant([
      [5.0, 3.0, 7.1],
      [2.3, 4.1, 4.8],
    ])
  fake_b = tf.constant([
      [2, 4, 5],
      [2, 8, 7]
    ])
  print sess.run(some_method(fake_a, fake_b))
```

Adding a tensor of floats to a tensor of ints won't work

One solution is to do a cast with tf.cast()

```
b = tf.cast(b,tf.float32)
```



```
def some_method(a, b):
    b = tf.cast(b,tf.float32)
    s = (a + b)
    return tf.sqrt(tf.matmul(s, tf.transpose(s)))

with tf.Session() as sess:
    fake_a = tf.constant([
        [5.0, 3.0, 7.1],
        [2.3, 4.1, 4.8],
      ])
    fake_b = tf.constant([
        [2, 4, 5],
        [2, 8, 7]
      ])
    print sess.run(some_method(fake_a, fake_b))

```

One solution is to do a cast with tf.cast()

output

```
[[ 15.63361835  16.04929924]
 [ 16.04929924  17.43960953]]
```

To debug full-blown programs, there are three methods

To debug full-blown programs, there are three methods

tf.Print()

To debug full-blown programs, there are three methods

tf.Print()

tfdbg

To debug full-blown programs, there are three methods

tf.Print()

tfdbg

TensorBoard

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
```

tf.Print() can be used to log specific tensor values

```python
import tensorflow as tf

def some_method(a, b):
    b = tf.cast(b, tf.float32)
    s = (a / b) # oops! NaN
    print_ab = tf.Print(s, [a, b])
    s = tf.where(tf.is_nan(s), print_ab, s)
    return tf.sqrt(tf.matmul(s, tf.transpose(s)))

with tf.Session() as sess:
    fake_a = tf.constant([[5.0, 3.0, 7.1], [2.3, 4.1, 4.8] ])
    fake_b = tf.constant([[2, 0, 5],[2, 8, 7]])
    print sess.run(some_method(fake_a, fake_b))

```

```
%bash
python xyz.py
```

```
Output:
[[         nan          nan]
 [         nan  1.43365264]]
```

https://www.tensorflow.org/programmers_guide/debugger

tf.Print() can be used to log specific tensor values

```python
import tensorflow as tf

def some_method(a, b):
    b = tf.cast(b, tf.float32)
    s = (a / b) # oops! NaN
    print_ab = tf.Print(s, [a, b])
    s = tf.where(tf.is_nan(s), print_ab, s)
    return tf.sqrt(tf.matmul(s, tf.transpose(s)))

with tf.Session() as sess:
    fake_a = tf.constant([[5.0, 3.0, 7.1], [2.3, 4.1, 4.8] ])
    fake_b = tf.constant([[2, 0, 5],[2, 8, 7]])
    print sess.run(some_method(fake_a, fake_b))

```

```
%bash
python xyz.py
```

```
Output:
[[         nan          nan]
 [         nan  1.43365264]]
```

https://www.tensorflow.org/programmers_guide/debugger

tf.Print() can be used to log specific tensor values

```
1  import tensorflow as tf
2
3  def some_method(a, b):
4      b = tf.cast(b, tf.float32)
5      s = (a / b) # oops! NaN
6      print_ab = tf.Print(s, [a, b])
7      s = tf.where(tf.is_nan(s), print_ab, s)
8      return tf.sqrt(tf.matmul(s, tf.transpose(s)))
9
10 with tf.Session() as sess:
11     fake_a = tf.constant([[5.0, 3.0, 7.1], [2.3, 4.1, 4.8] ])
12     fake_b = tf.constant([[2, 0, 5],[2, 8, 7]])
13     print sess.run(some_method(fake_a, fake_b))
14
```

```
%bash
python xyz.py
```

```
Output:
[[        nan         nan]
 [        nan  1.43365264]]
```

https://www.tensorflow.org/programmers_guide/debugger

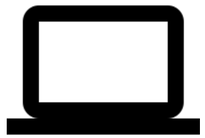
TensorFlow has a dynamic, interactive debugger (no easy way to use it from Datalab currently)

```
1  import tensorflow as tf
2  from tensorflow.python import debug as tf_debug
3
4  def some_method(a, b):
5    b = tf.cast(b, tf.float32)
6    s = (a / b) # oops! NaN
7    return tf.sqrt(tf.matmul(s, tf.transpose(s)))
8
9  with tf.Session() as sess:
10   fake_a = tf.constant([[5.0, 3.0, 7.1], [2.3, 4.1, 4.8] ])
11   fake_b = tf.constant([[2, 0, 5],[2, 8, 7]])
12
13   sess = tf_debug.LocalCLIDebugWrapperSession(sess)
14   sess.add_tensor_filter("has_inf_or_nan", tf_debug.has_inf_or_nan)
15   print sess.run(some_method(fake_a, fake_b))
16
```

```
# in a Terminal window
python xyz.py --debug
```

https://www.tensorflow.org/programmers_guide/debugger

Use TensorFlow debugger to step through code



```
1  import tensorflow as tf
2  from tensorflow.python import debug as tf_debug
3
4  def some_method(a, b):
5    b = tf.cast(b, tf.float32)
6    s = (a / b) # oops! NaN
7    return tf.sqrt(tf.matmul(s, tf.transpose(s)))
8
9  with tf.Session() as sess:
10   fake_a = tf.constant([[5.0, 3.0, 7.1], [2.3, 4.1, 4.8] ])
11   fake_b = tf.constant([[2, 0, 5],[2, 8, 7]])
12
13   sess = tf_debug.LocalCLIDebugWrapperSession(sess)
14   sess.add_tensor_filter("has_inf_or_nan", tf_debug.has_inf_or_nan)
15   print sess.run(some_method(fake_a, fake_b))
16
```

Use TensorFlow debugger to step through code

```
--- Node Stepper: run #1: 1 fetch (Sqrt:0); 0 feeds --------------
| <-- --> | (-1) lt
Topologically-sorted transitive input(s) and fetch(es):

  -->(1 / 14)  [      ] transpose/Range/start
     (2 / 14)  [      ] transpose/Range/delta
     (3 / 14)  [      ] Const
     (4 / 14)  [      ] Const_1
     (5 / 14)  [      ] Cast
     (6 / 14)  [      ] div
     (7 / 14)  [      ] transpose/Rank
     (8 / 14)  [      ] transpose/Range
     (9 / 14)  [      ] transpose/sub/y
     (10 / 14)  [      ] transpose/sub
     (11 / 14)  [      ] transpose/sub_1
     (12 / 14)  [      ] transpose
     (13 / 14)  [      ] MatMul
     (14 / 14)  [      ] Sqrt

Legend:
  P - Placeholder
  U - Unfeedable
  H - Already continued-to; Tensor handle available from output sl
  I - Unfeedable
  O - Has overriding (injected) tensor value
  D - Dirty variable: Variable already updated this node stepper.
```

cloud.google.com