

Faster ML Development with TensorFlow

Shanqing Cai (cais@google.com)

Senior Software Engineer, Google Brain (Cambridge, MA)

Guest Lecture @ [MIT 6.S191](#)

February, 2018



How are machine learning models represented?

Model is a **Data Structure**

e.g. A Graph

aka

“Symbolic” | “Deferred Execution” |
“Define-and-run”

Model is a **Program**

e.g. Python Code

aka

“Imperative” | “Eager Execution” |
“Define-by-run”

TensorFlow: Symbolic Mode

By default, TensorFlow is a **symbolic** engine.

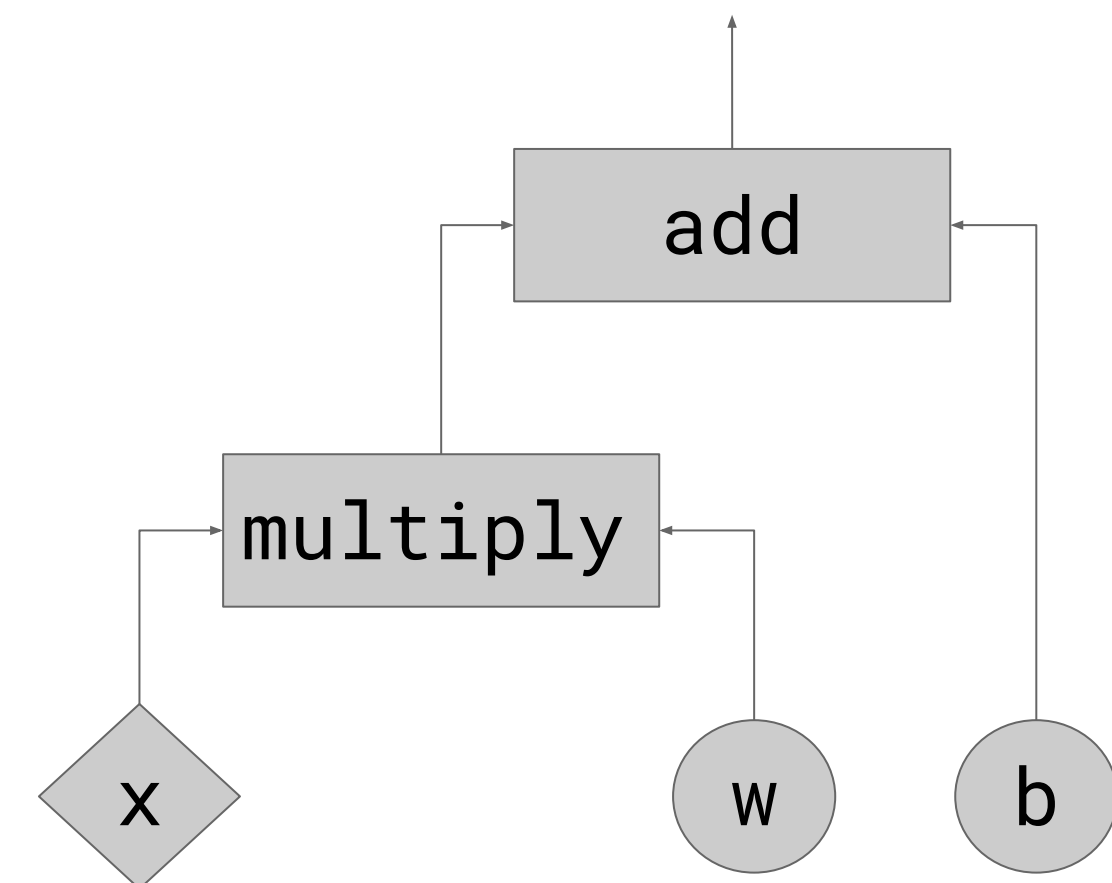
```
import tensorflow as tf

x = tf.constant(10.0)
w = tf.constant(4.0)
b = tf.constant(2.0)

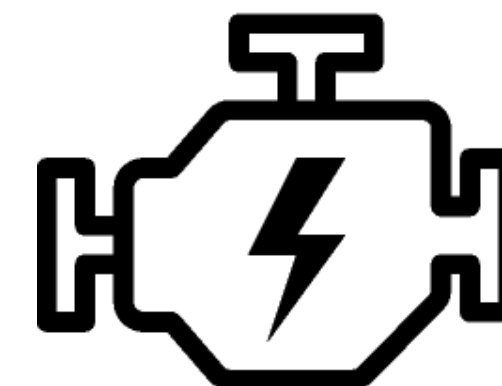
y = tf.multiply(x, w)
print(y)
# You get: Tensor("Mul:0",shape=(), dtype=float32)

z = tf.add(y, b)
print(z)
# You get: Tensor("Add:0",shape=(), dtype=float32)

# You need to create a "session" to perform the
# actual computation.
sess = tf.Session()
print(sess.run(z))
# You get: 42.0.
```



Model as a Data Structure



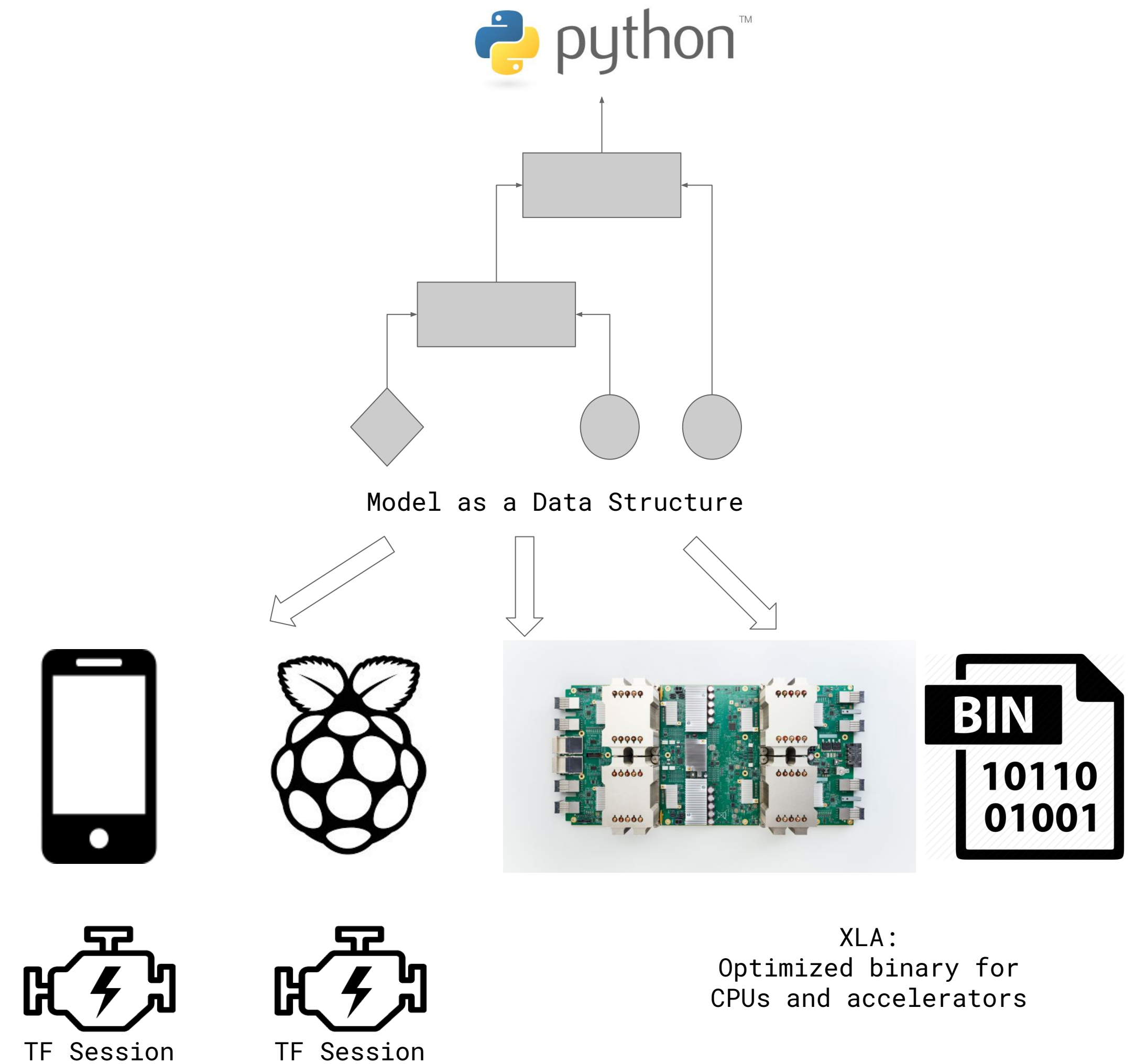
tf.Session

Output and/or model updates

Symbolic Execution in TensorFlow

Pros:

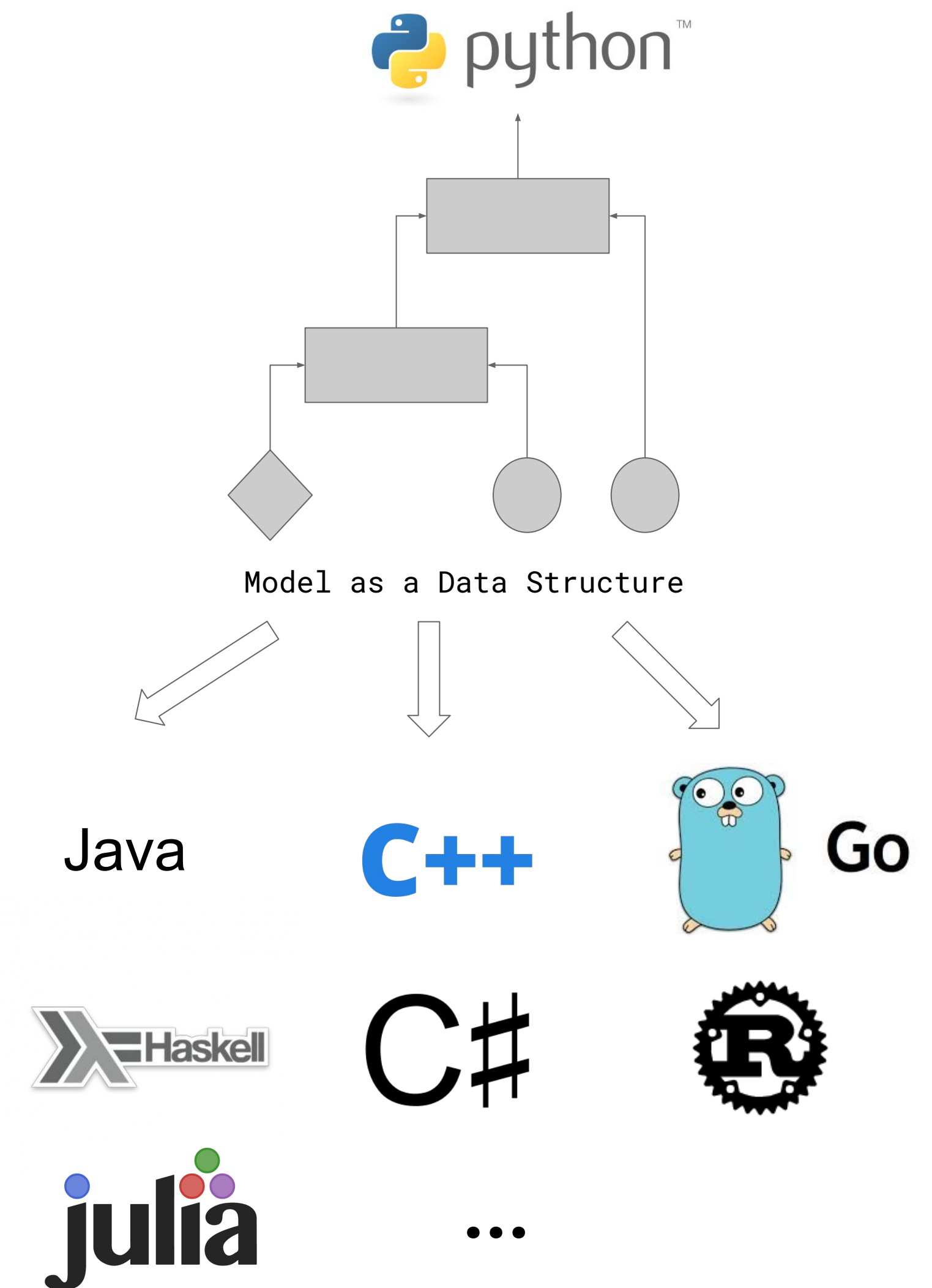
- + makes (de)serialization easier
- + deployment on devices
(e.g., mobile, TPU, XLA)



Symbolic Execution in TensorFlow

Pros:

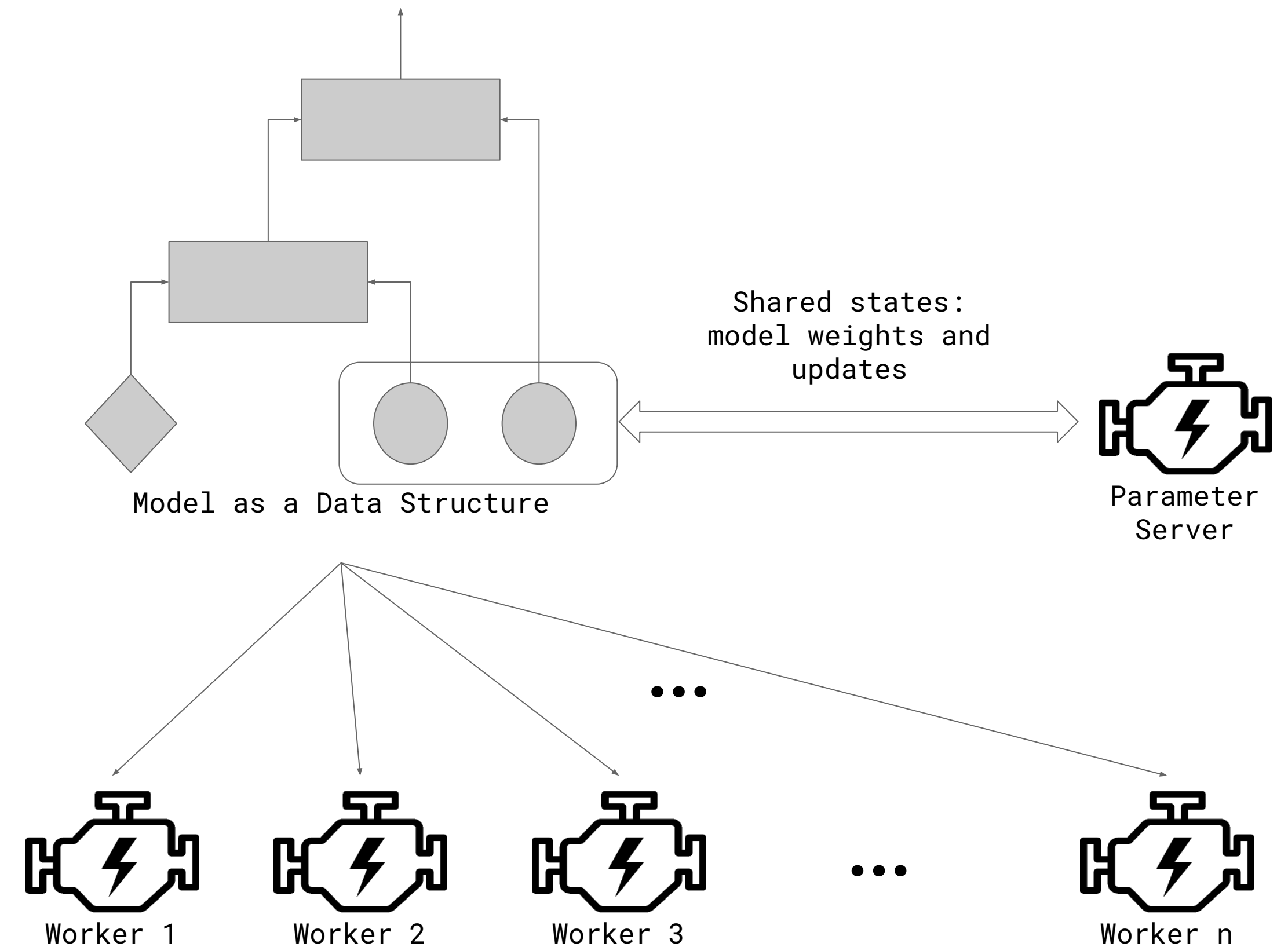
- + makes (de)serialization easier
- + deployment on devices
(e.g., mobile, TPU, XLA)
- + **interoperability between languages**



Symbolic Execution in TensorFlow

Pros:

- + makes (de)serialization easier
- + deployment on devices
(e.g., mobile, TPU, XLA)
- + interoperability between languages
- + **distributed training**



Symbolic Execution in TensorFlow

Pros:

- + makes (de)serialization easier
 - + deployment on devices
(e.g., mobile, TPU, XLA)
- + interoperability between languages
- + distributed training
- + **speed and concurrency not limited by language**
(e.g., Python global interpreter lock)

Model is a Data Structure

e.g. A Graph

aka

“Symbolic” | “Deferred Execution”

Symbolic Execution in TensorFlow

Pros:

- + makes (de)serialization easier
 - + deployment on devices
(e.g., mobile, TPU, XLA)
- + interoperability between languages
- + distributed training
- + speed and concurrency not limited by language
(e.g., Python global interpreter lock)

Model is a **Data Structure**

e.g. A Graph

aka

“Symbolic” | “Deferred Execution”

Symbolic Execution in TensorFlow

Pros:

- + makes (de)serialization easier
 - + deployment on devices
(e.g., mobile, TPU, XLA)
- + interoperability between languages
- + distributed training
- + speed and concurrency not limited by language
(e.g., Python global interpreter lock)

Cons:

- less intuitive
- harder to debug (*but see later slides)
- harder to write control flow structures
- harder to write dynamic models

Eager Execution in TensorFlow

- + easier to learn (“Pythonic”)
- + easier to debug
- + makes dynamic (data-dependent)
neural structures easier to write

Model is a **Program**

e.g. Python Code

aka

“Imperative” | “Eager Execution”

Eager Execution in TensorFlow

By default, TensorFlow is a **symbolic** engine.

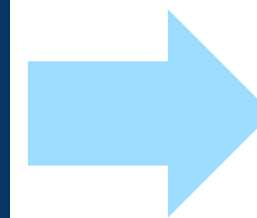
But since version 1.5, you can switch to the **imperative (eager)** mode.

```
import tensorflow as tf

x = tf.constant(10.0)
w = tf.constant(4.0)
b = tf.constant(2.0)

y = tf.multiply(x, w)
print(y)
# You get: Tensor("Mul:0",shape=(), dtype=float32)

z = tf.add(y, b)
print(z)
# You get: Tensor("Add:0",shape=(), dtype=float32)
```



```
import tensorflow as tf

import tensorflow.contrib.eager as tfe
tfe.enable_eager_execution()

x = tf.constant(10.0)
w = tf.constant(4.0)
b = tf.constant(2.0)

y = tf.multiply(x, w)
print(y)
# You get: tf.Tensor(40.0,shape=(), dtype=float32)

z = tf.add(y, b)
print(z)
# You get: tf.Tensor(42.0,shape=(), dtype=float32)
```

See eager-mode [examples](#) and [notebooks](#).

Symbolic vs. Eager Mode

- + easier to learn (“Pythonic”)
- + easier to debug
- + **makes dynamic (data-dependent)**
neural structures easier to write

Model is a Program

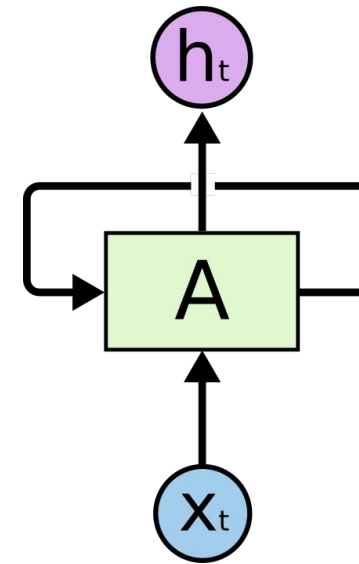
e.g. Python Code

aka

“Imperative” | “Eager Execution”

TensorFlow: Control Flow in Symbolic vs. Eager

Writing a basic RNN:



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Symbolic

```
dense1 = tf.layers.Dense(state_size, activation='tanh')
dense2 = tf.layers.Dense(state_size)

def loop_cond(i, state, output):
    return i < max_sequence_len

def loop_body(i, state, output):
    input_slice = input_array.read(i)
    combined = tf.concat([input_slice, state], axis=1)
    state_updated = dense1(combined)
    state = tf.where(i >= sequence_lengths, state, state_updated)
    output_updated = dense2(state)
    output = tf.where(
        i >= sequence_lengths, output, output_updated)
    return i + 1, state, output

_, final_state, final_output = tf.nn.nn_loop(
    loop_cond, loop_body,
    [i, initial_state, dummy_initial_output])

sess.run([final_state, final_output])
```

Eager

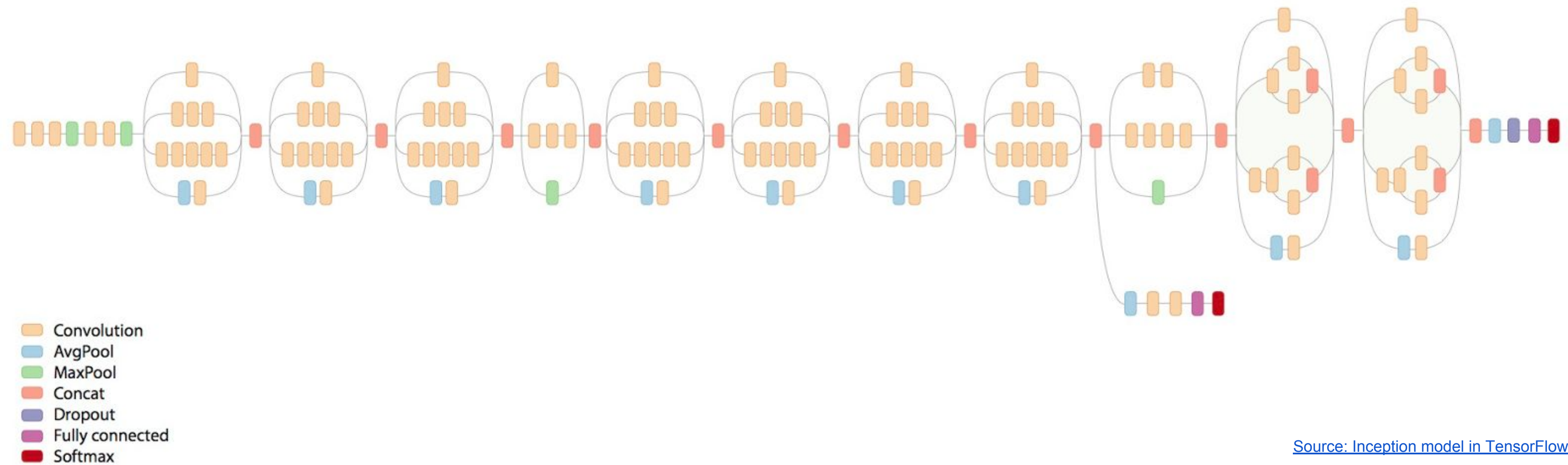
```
dense1 = tf.layers.Dense(state_size, activation='tanh')
dense2 = tf.layers.Dense(state_size)

for i in xrange(max_sequence_len):
    input_slice = input_array.read(i)
    combined = tf.concat([input_slice, state], axis=1)
    state_updated = dense1(combined)
    state = tf.where(i >= sequence_lengths, state, state_updated)
    output_updated = dense2(state)
    output = tf.where(
        i >= sequence_lengths, output, output_updated)

final_state, final_output = state, output
```

Model Structures: Static vs. Dynamic

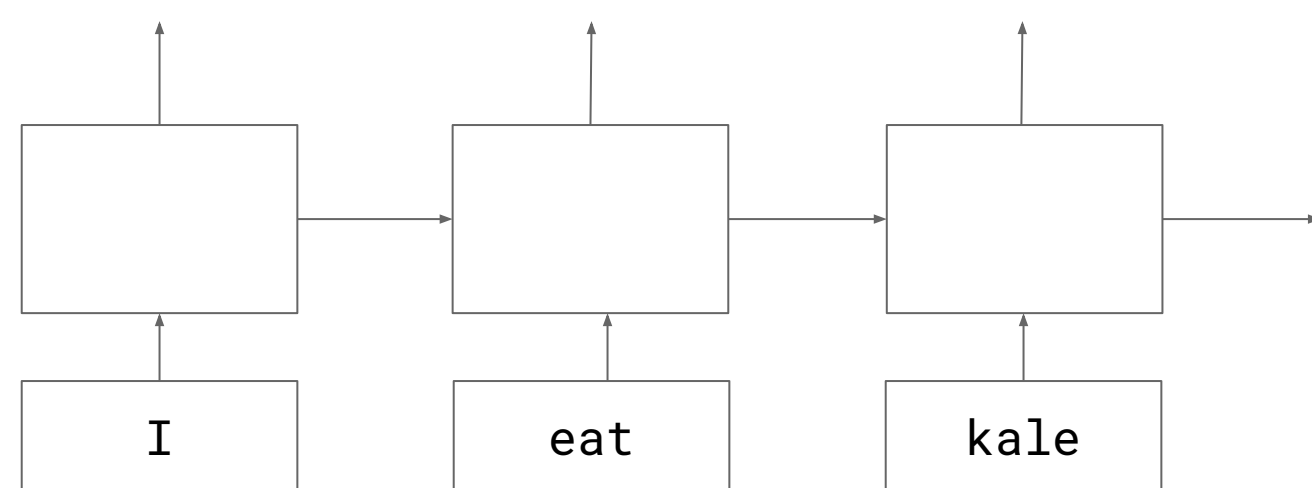
Static models



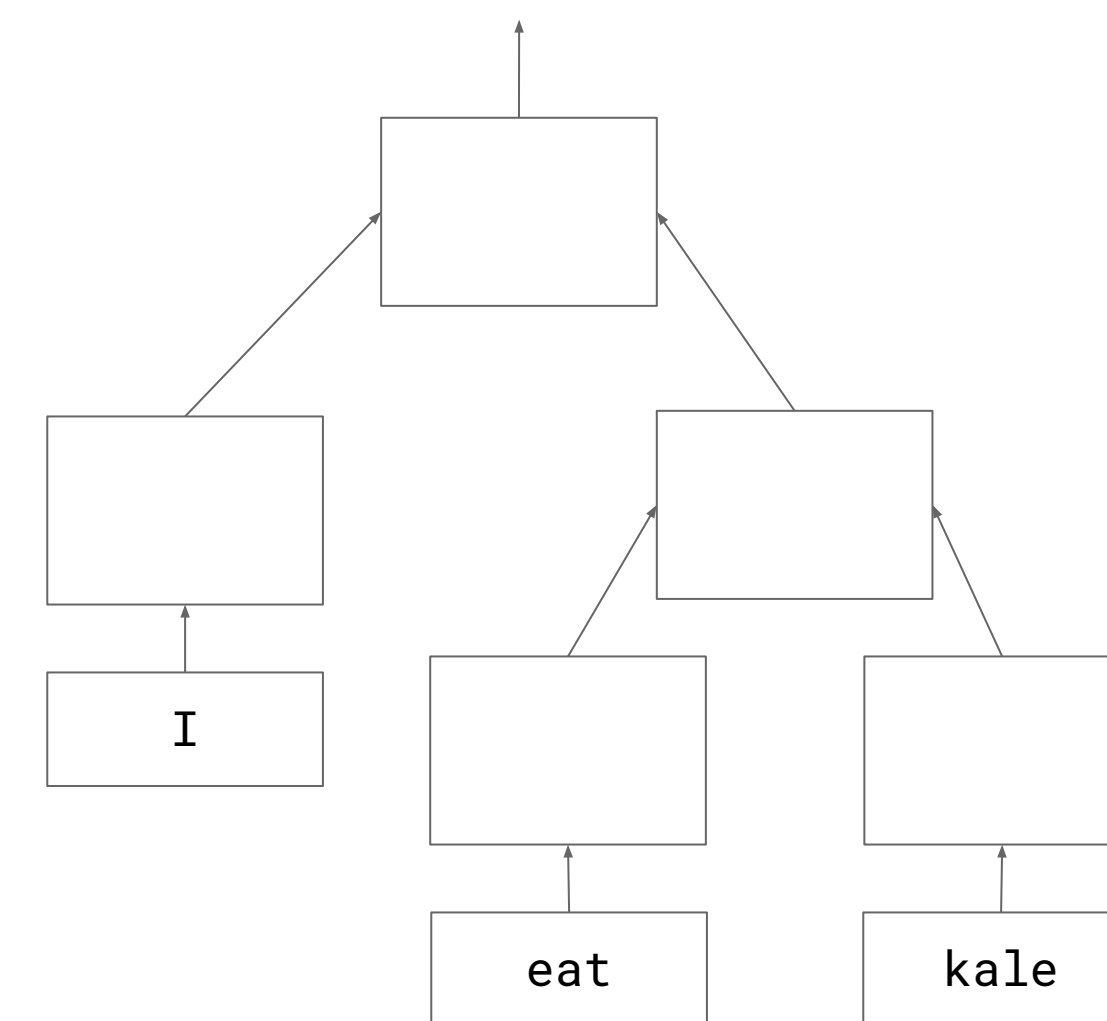
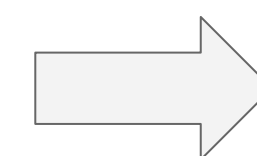
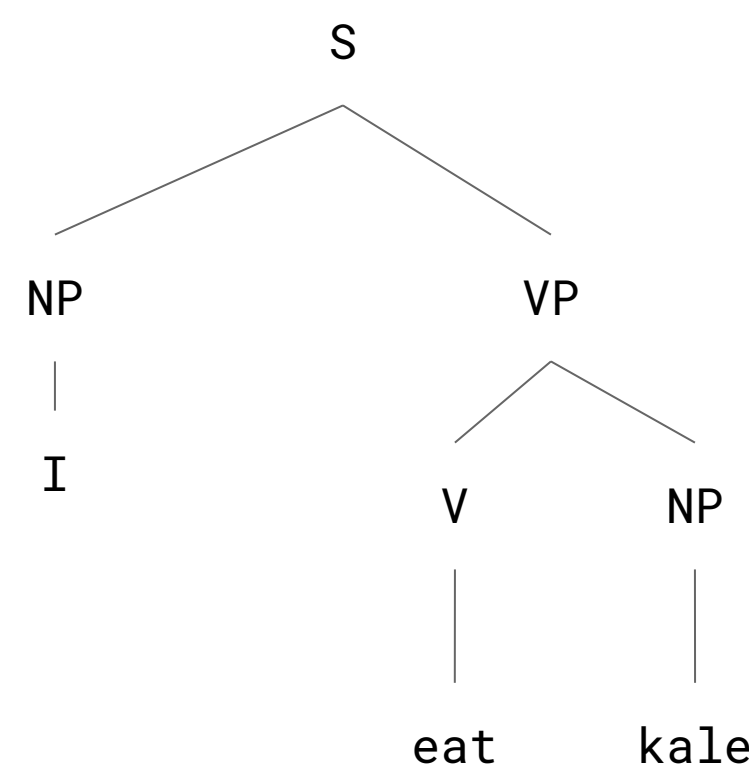
- + Model structure is fixed regardless of input data.
- + The majority of DL models for image, audio and numerical data.

Model Structures: Static vs. Dynamic

Traditional RNN



Dynamic Models, e.g., Tree RNN



- + Models whose structure cannot be easily described as a graph, i.e., changes a lot with input data.
- + Used by some state-of-the-art models that deal with hierarchical structures in natural language.
- + Difficult to write in the symbolic way (using `tf.cond` and `tf.while_loop`)
- + Straightforward with Eager: using the native Python control flow. See [the SPINN example](#).

What if you want to debug symbolic execution?

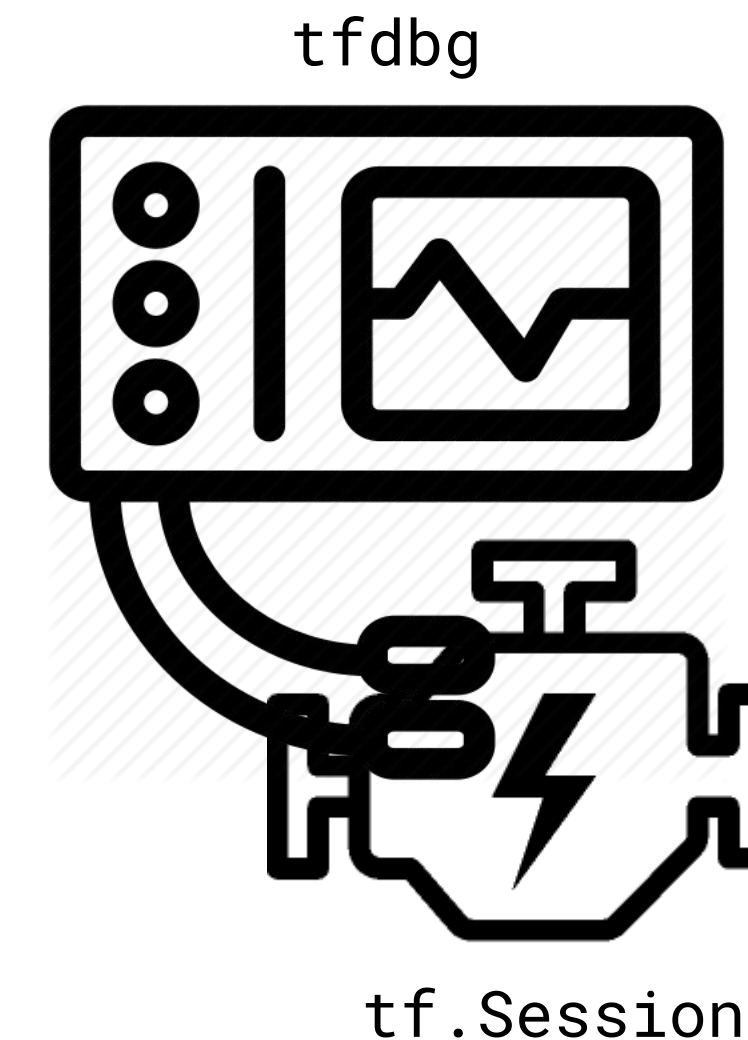
TensorFlow Debugger (tfdbg): Command Line Interface

```
import tensorflow as tf
from tensorflow.python import debug as tfdbg

a = tf.constant(10.0)
b = tf.Variable(4.0)
c = tf.Variable(2.0)

x = tf.multiply(a, b)
y = tf.add(c, x)

sess = tf.Session()
sess = tfdbg.LocalCLIDebugWrapperSession(sess)
sess.run(tf.global_variables_initializer())
sess.run(y)
```



What if you want to debug symbolic execution?

```
import tensorflow as tf
from tensorflow.python import debug as tfdbg

a = tf.constant(10.0)
b = tf.Variable(4.0)
c = tf.Variable(2.0)

x = tf.multiply(a, b)
y = tf.add(c, x)

sess = tf.Session()
sess = tfdbg.LocalCLIDebugWrapperSession(sess)
sess.run(tf.global_variables_initializer())
sess.run(y)
```

- Presents after each Session.run:
 - All tensor values in the computation graph
 - Graph structure
- ... in an interactive, mouse-clickable CLI.

```
--- run-end: run #1: 1 fetch (init); 0 feeds -----
| <-- --> | Lt
| list_tensors | node_info | print_tensor | list_inputs | list_outputs | run_info
6 dumped tensor(s):
t (ms)  Size (B) Op type  Tensor name
[0.000]  174   VariableV2  Variable:0
[0.008]  178   VariableV2  Variable_1:0
[5.207]  208    Const     Variable/initial_value:0
[5.526]  212    Const     Variable_1/initial_value:0
[10.375] 194    Assign     Variable/Assign:0
[10.427] 198    Assign     Variable_1/Assign:0

--- Scroll (PgDn): 0.00% ----- Mouse: ON ---
tfdbg>
```

```
--- run-end: run #2: 1 fetch (Mul:0); 0 feeds -----
| <-- --> | Lt
| list_tensors | node_info | print_tensor | list_inputs | list_outputs | run_info
7 dumped tensor(s):
t (ms)  Size (B) Op type  Tensor name
[0.000]  174    Const     Const:0
[0.012]  184   VariableV2  Variable_1:0
[0.527]  180   VariableV2  Variable:0
[0.605]  194   Identity    Variable_1/read:0
[0.778]  190   Identity    Variable/read:0
[1.089]  170    Add        Add:0
[1.301]  170    Mul        Mul:0

--- Scroll (PgDn): 0.00% ----- Mouse: ON ---
tfdbg>
```

TensorFlow: Debugging Numerical Instability (NaNs and Infinities)

```
--- run-end: run #4: 1 fetch (train/Adam); 2 feeds -----  
| <-- --> | lt -f has_inf_or_nan  
| list_tensors | node_info | print_tensor | list_inputs | list_outputs | run_i  
36 dumped tensor(s) passing filter "has_inf_or_nan":  
  
t (ms)  Size  Op type  Tensor name  
[14.385] 3.97k Log      cross_entropy/Log:0  
[14.490] 3.97k Mul      cross_entropy/mul:0  
[14.862] 4.00k Mul      train/gradients/cross_entropy/mul_grad/mul:0  
[14.935] 4.00k Sum      train/gradients/cross_entropy/mul_grad/Sum:0  
[14.995] 4.00k Reshape  train/gradients/cross_entropy/mul_grad/Reshape:0  
[15.037] 4.00k Reciprocal train/gradients/cross_entropy/Log_grad/Reciprocal:0
```

tfdbg> run -f has_inf_or_nan

See walkthrough at

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

Common causes of NaNs and infinities in DL models:

- **underflow** followed by:
 - division by zero
 - logarithm of zero
- **overflow** caused by:
 - learning rate too high
 - bad training examples

New Tool: Graphical Debugger for TensorFlow

(TensorBoard Debugger Plugin)

```
# Do the following in a terminal.

# Install nightly builds.
pip install --upgrade --force-reinstall \
    tf-nightly tb-nightly grpcio

# Start tensorboard with debugger enabled.
tensorboard \
    --logdir /tmp/logdir \
    --port 6006 \
    --debugger_port 7007

# Open a browser and navigate to:
# http://localhost:6006/#debugger

# Then save the code in a file and run it. -->
```

```
import tensorflow as tf
from tensorflow.python import debug as tf_debug

a = tf.random_normal([10, 1])
b = tf.random_normal([10, 10])
c = tf.random_normal([10, 1])

x = tf.matmul(b, a)
y = tf.add(c, x)

sess = tf.Session()
sess = tf_debug.TensorBoardDebugWrapperSession(
    sess, 'localhost:7007')
for _ in xrange(100):
    sess.run(y)
```

- Not publicly announced yet (coming in TensorFlow 1.6)
- But available for preview in nightly builds of tensorflow and tensorboard

Try it yourself!

New Tool: Visual Debugger for TensorFlow

TensorBoard

DEBUGGER

INACTIVE

Node List

Filter Mode

Filter Regex

Show Code: ☒

☒ /job:localhost/replica:0/task:0/device:CPU:0

☒ [Add] Add

☒ [Const] Const

☒ [Mul] Mul

☒ Variable

Source Code

tdp_demo.py

```
1 import tensorflow as tf
2 from tensorflow.python import debug as tf_debug
3
4 1/1 ▼ a = tf.constant(10.0)
5 2/4 ▼ b = tf.Variable(4.0)
6 2/4 ▼ c = tf.Variable(2.0)
7
8 1/1 ▼ x = tf.multiply(a, b)
9 1/1 ▼ y = tf.add(c, x)
10
11 sess = tf.Session()
12 sess = tf_debug.TensorBoardDebugWrapperSession(
13     sess, 'localhost:7007')
14 0/1 ▼ sess.run(tf.global_variables_initializer())
15 sess.run(y)
```

Session Runs

Feeds	Fetches	Targets	#(Devices)	Count
		init	1	1
	Add:0		1	1

STEP

CONTINUE...

Runtime Graphs

/job:localhost/replica:0/task:0/device:CPU:0 (device 1 of 1) Device name

Right-click nodes and select "expand and highlight" to go to the corresponding line in the source code.

View the runtime graph structure.

View summaries of watched tensor values.

Tensor Value Overview

Tensor	Count	DType	Shape	Value
Variable:0	1	float32	[]	4
Variable/read:0	1	float32	[]	4
Const:0	1	float32	[]	10
Variable_1:0	1	float32	[]	2

Tensor Values

Detailed view of watched tensor values.

Health Pill ☒

Legend: NaN -∞ 0 + +

4.0

4.0

10.0

2.0

A tree view of all graph nodes.

Checkbox = watch.

Typing graph nodes back to the Python lines that created them.

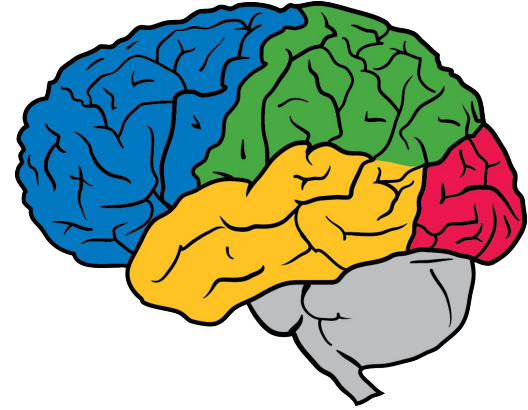
Continue over Session.runs or to a certain tensor-value condition.

Step node by node (tensor by tensor).

Summary

- ML/DL models can be represented in two ways:
 - as a **data structure** → **Symbolic Execution**:
good for deployment, distribution, and optimization
 - as a **program** → **Eager Execution**:
good for prototyping, debugging and dynamic models; easier to learn
- TensorFlow supports both modes
- TensorFlow Debugger (tfdbg) provides visibility into symbolically-executing models and help you debug/understand them in:
 - command line
 - browser

Acknowledgements



Google Brain Team in Mountain View, CA and Cambridge, MA.

Chi Zeng and Mahima Pushkarna: Collaborators on the visual tfdbg project.

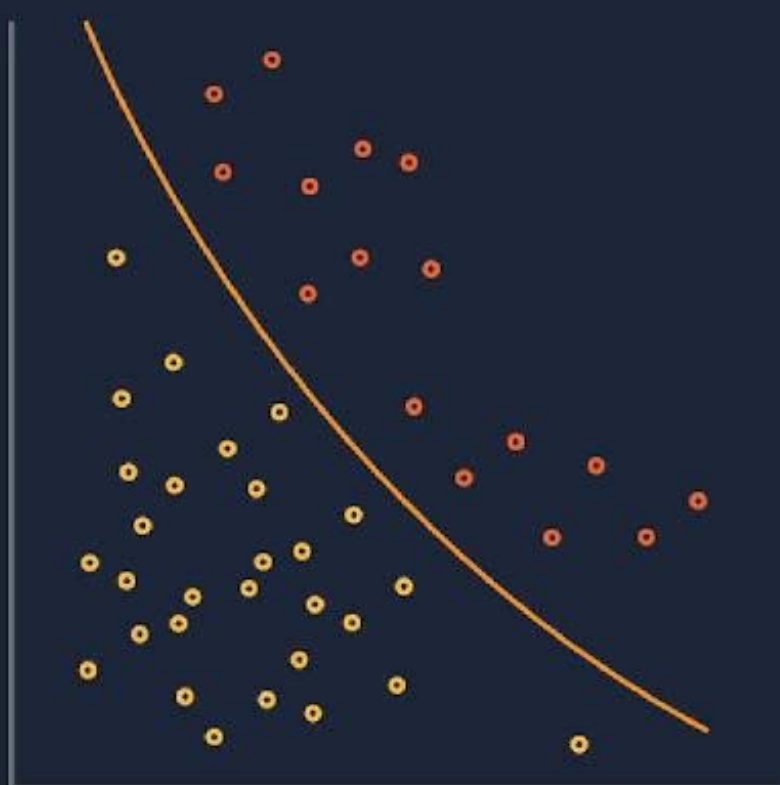
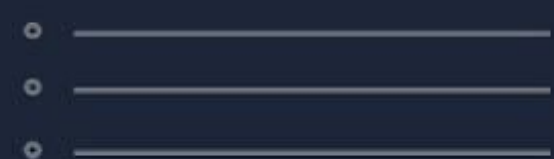
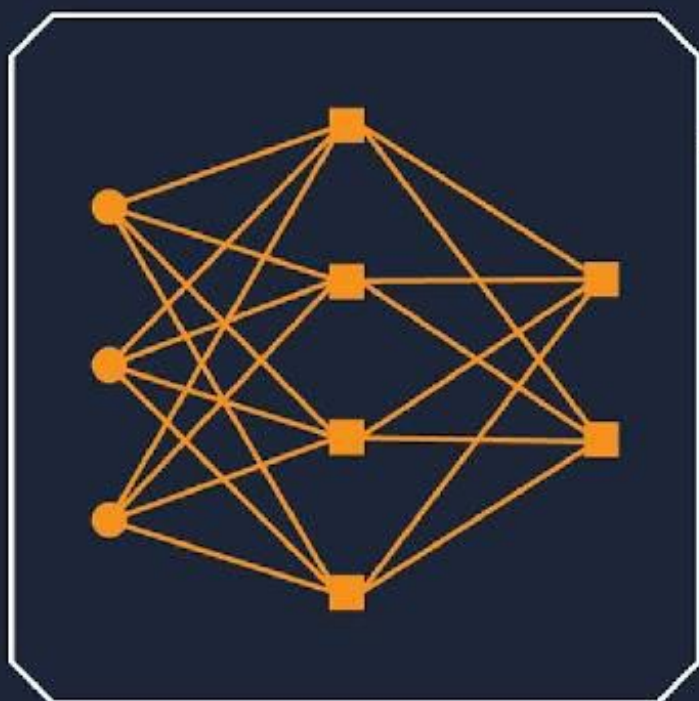
Open-source contributors to TensorFlow.

Thank you!

For questions, email cais@google.com

For TensorFlow issues, go to <https://github.com/tensorflow/tensorflow/issues>

For TensorBoard issues, go to <https://github.com/tensorflow/tensorboard/issues>



TensorFlow

