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

```python
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

Output and/or model updates
```

`tf.Session`
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.

```python
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)
```

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

```python
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

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.while_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.
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 \texttt{tf.cond} and \texttt{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

```python
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)
```

 tfdbg
 tf.Session

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.
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
# 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)
```

### New Tool: Graphical Debugger for TensorFlow

**TensorBoard Debugger Plugin**

- 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

- A tree view of all graph nodes. Checkbox = watch.
- Tying graph nodes back to the Python lines that created them.
- Right-click nodes and select “expand and highlight” to go to the corresponding line in the source code.
- Detailed view of watched tensor values.
- View the runtime graph structure.
- View summaries of watched tensor values.
- Step node by node (tensor by tensor).
- Continue over Session.runs or to a certain tensor-value condition.
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