

TensorFlow warmup

This is a notebook to get you started with TensorFlow.

```
In [1]: import numpy as np
import tensorflow as tf
```

Graph visualisation

This is for visualizing a TF graph in an iPython notebook; the details are not interesting. (Borrowed from the [DeepDream iPython notebook](https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/deepdream/deepdream.ipynb) (<https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/deepdream/deepdream.ipynb>))

```
In [2]: # This is for graph visualization.

from IPython.display import clear_output, Image, display, HTML

def strip_consts(graph_def, max_const_size=32):
    """Strip large constant values from graph_def."""
    strip_def = tf.GraphDef()
    for n0 in graph_def.node:
        n = strip_def.node.add()
        n.MergeFrom(n0)
        if n.op == 'Const':
            tensor = n.attr['value'].tensor
            size = len(tensor.tensor_content)
            if size > max_const_size:
                tensor.tensor_content = "<stripped %d bytes>"%size
    return strip_def

def show_graph(graph_def, max_const_size=32):
    """Visualize TensorFlow graph."""
    if hasattr(graph_def, 'as_graph_def'):
        graph_def = graph_def.as_graph_def()
    strip_def = strip_consts(graph_def, max_const_size=max_const_size)
    code = """
    <script>
        function load() {{
            document.getElementById("{id}").pbtxt = {data};
        }}
    </script>
    <link rel="import" href="https://tensorboard.appspot.com/tf-graph-basic.build.html" onload=load()>
    <div style="height:600px">
        <tf-graph-basic id="{id}"></tf-graph-basic>
    </div>
    """.format(data=repr(str(strip_def)), id='graph'+str(np.random.rand()))

    iframe = """
    <iframe seamless style="width:1200px;height:620px;border:0" srcdoc="{}"></iframe>
    """.format(code.replace("'", '"'))
    display(HTML(iframe))
```

The execution model

TensorFlow allows you to specify graphs representing computations and provides a runtime for efficiently executing those graphs across a range of hardware.

The graph nodes are Ops and the edges are Tensors.

```
In [13]: # This code only creates the graph. No computation is done yet.
tf.reset_default_graph()
x = tf.constant(7.0, name="x")
y = tf.add(x, tf.constant(2.0, name="y"), name="add_op")
z = tf.subtract(x, tf.constant(2.0, name="z"), name="sub_op")
w = tf.multiply(y, tf.constant(3.0)) # If no name is given, TF will chose a unique name for us.

# Visualize the graph. [ ] GFW needed 這裡必須翻牆
show_graph(tf.get_default_graph().as_graph_def())
```

Fit to screen

Run

Upload

Choose File

Color

Structure

color: same substructure
gray: unique substructure

Graph

(* = expandable)

Namespace*

OpNode

Unconnected series*

Connected series*

Constant

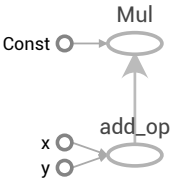
Summary

Dataflow edge

Control dependency edge

Reference edge

Main Graph



Auxiliary nodes



Ops

- Every node in the computation graph corresponds to an op. tf.constant, tf.sub and tf.add are Ops.
- There are many built-in Ops for low-level manipulation of numeric Tensors, e.g.:
 - Arithmetic (with matrix and complex number support)
 - Tensor operations (reshape, reduction, casting)
 - Image manipulation (cropping, sizing, coloring, ...)
 - Batching (arranging training examples into batches)
- **Almost every object in TensorFlow is an op.** Even things that don't look like they are! TensorFlow uses the op abstraction for a surprising range of things:
 - Queues
 - Variables
 - Variable initializers

This can be confusing at first. For now, remember that because many things are Ops, some things have to be done in a somewhat non-obvious fashion.

A list of TF Ops can be found at https://www.tensorflow.org/api_docs/python/ (https://www.tensorflow.org/api_docs/python/).

Tensors

- x, y, w and z are **Tensors** - a description of a multidimensional array.
- A Tensor is a symbolic handle to one of the outputs of an Operation. It does not hold the values of that operation's output, but instead provides a means of computing those values in a TensorFlow tf.Session.
- Tensor shapes can usually be derived from the computation graph. This is called shape inference.
 - For example, if you perform a matrix multiply of a [4,2] and a [2,3] Tensor, then TensorFlow infers that the output Tensor has shape [4,3].

In [8]:

```
import peforth
peforth.ok()

__main__ :> x value x // ( -- tensor )
OK x . cr
Tensor("x:0", shape=(), dtype=float32)
OK __main__ :> y value y // ( -- tensor )
OK __main__ :> z value z // ( -- tensor )
OK __main__ :> w value w // ( -- tensor )
OK y . cr
Tensor("add_op:0", shape=(), dtype=float32)
OK z . cr
Tensor("sub_op:0", shape=(), dtype=float32)
OK w . cr
Tensor("Mul:0", shape=(), dtype=float32)
OK exit
OK
```

In [12]:

```
# We can also use shorthand syntax
# Notice the default names TF chooses for us.
tf.reset_default_graph()
x = tf.constant(7.0)
y = x + 2
z = x - 2
w = y * 3

# Visualize the graph.
show_graph(tf.get_default_graph().as_graph_def())
```

Fit to screen

Run

Upload

Choose File

Color

Structure

color: same substructure
gray: unique substructure

Graph

(* = expandable)

Namespace*

OpNode

Unconnected series*

Connected series*

Constant

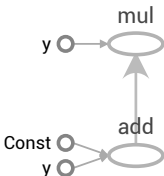
Summary

Dataflow edge


Control dependency edge

Reference edge

Main Graph



Auxiliary nodes



Session

- The actual computations are carried out in a Session.
- Each session has exactly one graph, but it is completely valid to have multiple disconnected subgraphs in the same graph.
- The same graph can be used to initialize two different Sessions, yielding two independent environments with independent states.
- Unless specified otherwise, nodes and edges are added to the default graph.
 - By default, a Session will use the default graph.

http://localhost:8888/nbconvert/html/Documents/GitHub/tensorflow-workshop/zurich/01_tensorflow_warmup.ipynb?download=false

3/7

```
In [10]: tf.reset_default_graph()
x = tf.constant(7.0, name="x")
y = tf.add(x, tf.constant(2.0, name="y"), name="add_op")
z = y * 3.0
# Create a session, which is the context for running a graph.
with tf.Session() as sess:
    # When we call sess.run(y) the session is computing the value of Tensor y.
    print(sess.run(y))
    print(sess.run(z))

9.0
27.0
```

Variables

- Variables maintain state in a Session across multiple calls to Session.run().
- You add a variable to the graph by constructing an instance of the class tf.Variable.
- For example, model parameters (weights and biases) are stored in Variables.
 - We train the model with multiple calls to Session.run(), and each call updates the model parameters.
- For more information on Variables see https://www.tensorflow.org/programmers_guide/variables (https://www.tensorflow.org/programmers_guide/variables).

```
In [15]: tf.reset_default_graph()
# tf.get_variable returns a tf.Variable object. Creating such objects directly
# is possible, but does not have a sharing mechanism. Hence, tf.get_variable is
# preferred.
x = tf.get_variable("x", shape=[], initializer=tf.zeros_initializer())

assign_x = tf.assign(x, 10, name="assign_x")
z = tf.add(x, 1, name="z")

# Variables in TensorFlow need to be initialized first. The following op
# conveniently takes care of that and initializes all variables.
init = tf.global_variables_initializer()

# Visualize the graph.
show_graph(tf.get_default_graph().as_graph_def())
```

Fit to screen

Run

Upload

Choose File

Color

Structure

color: same substructure
gray: unique substructure

Graph

(* = expandable)

Namespace*

OpNode

Unconnected series*

Connected series*

Constant

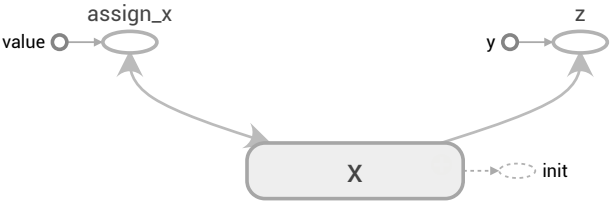
Summary

Dataflow edge

Control dependency edge

Reference edge

Main Graph



Auxiliary nodes



- The variable we added represents a variable in the computational graph, **but is not an instance of the variable**.
- The computational graph represents a program, and the variable will exist when we run the graph in a session.
- The value of the variable is stored in the session.

Take a guess: what is the output of the code below?

```
In [16]: with tf.Session() as sess:
        # Assign an initial value to the instance of the variable in this session,
        # determined by the initializer provided above.
        sess.run(init)
        print (sess.run(z))

1.0
```

The output might surprise you: it's 1.0! The op `assign_x` is not a dependency of `x` or `z`, and hence is never evaluated.

One way to solve this problem is:

```
In [17]: with tf.Session() as sess:
        # When we create a new session we need to initialize all Variables again.
        sess.run(init)
        sess.run(assign_x)
        print (sess.run(z))

11.0
```

Placeholders

So far you have seen Variables, but there is a more basic construct: the placeholder. A placeholder is simply a variable that we will assign data to at a later date. It allows us to create our operations and build our computation graph, without needing the data. In TensorFlow terminology, we then feed data into the graph through these placeholders.

```
In [18]: tf.reset_default_graph()

x = tf.placeholder("float", None)
y = x * 2

# Visualize the graph.
show_graph(tf.get_default_graph().as_graph_def())
```

Fit to screen

Run

Upload

Color

Structure

color: same substructure

gray: unique substructure

Graph

(* = expandable)

Namespace*

OpNode

Unconnected series*

Connected series*

Constant

Summary

Dataflow edge

Control dependency edge

Reference edge

Main Graph

y

mul

Placeholder

At execution time, we feed data into the graph using a `feed_dict`: for each placeholder, it contains the value we want to assign to it. This can be useful for batching up data, as you will see later.

```
In [19]: with tf.Session() as session:
         result = session.run(y, feed_dict={x: [1, 2, 3]})
         print(result)

[ 2.  4.  6.]
```

Queues

Queues are TensorFlow’s primitives for writing asynchronous code.

- Queues provide Ops with queue semantics.
- Queue Ops, like all Ops, need to be executed to do anything.
- Are often used for asynchronously processing data (e.g., an input pipeline with data augmentation).
- Queues are stateful graph nodes. The state is associated with a session.
- There are several different types of queues, e.g., FIFOQueue and RandomShuffleQueue.

See the [Threading and Queues](https://www.tensorflow.org/programmers_guide/threading_and_queues) (https://www.tensorflow.org/programmers_guide/threading_and_queues) for more details.

Note: You probably will never need to directly use these low level implementations of queues yourself. Do note, however, that several important operations (for example, reading and batching) are implemented as queues.

```
In [20]: tf.reset_default_graph()
q = tf.FIFOQueue(3, "float", name="q")
initial_enqueue = q.enqueue_many([[0., 0., 0.],], name="init")

x = q.dequeue()
y = x + 1
q_inc = q.enqueue([y])

with tf.Session() as session:
    session.run(initial_enqueue)
    outputs = []
    for _ in range(20):
        _, y_val = session.run([q_inc, y])
        outputs.append(y_val)
    print(outputs)

# Visualize the graph.
show_graph(tf.get_default_graph().as_graph_def())

[1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 3.0, 3.0, 3.0, 4.0, 4.0, 4.0, 5.0, 5.0, 5.0, 6.0, 6.0, 6.0, 7.0, 7.0]
```

Fit to screen

Run

Upload

Color

Choose File

Structure

color: same substructure

gray: unique substructure

Graph

(* = expandable)

Namespace*

OpNode

Unconnected series*

Connected series*

Constant

Summary

Dataflow edge

Control dependency edge

Reference edge

Main Graph

The diagram illustrates the TensorFlow computational graph for the provided code. It features several nodes and edges:

- init**: A node representing the initial enqueue operation, connected to a **componen...** (component) node.
- q**: A central node representing the **FIFOQueue** object.
- q_enqueue**: A node representing the **enqueue_many** operation, connected to **q**.
- q_Dequeue**: A node representing the **dequeue** operation, connected to **q**.
- add**: A node representing the **add** operation, which takes the output of **q_Dequeue** and a constant **y** (represented by a small circle) as inputs.
- y**: A constant node representing the value 1, which is added to the dequeued value.

 The graph shows the flow of data and control dependencies between these operations.

http://localhost:8888/nbconvert/html/Documents/GitHub/tensorflow-workshop/zurich/01_tensorflow_warmup.ipynb?download=false

6/7

Exercise: Collatz Conjecture

And now some fun! Collatz conjecture states that after applying the following rule

$$f(n) = \begin{cases} n/2 & \text{if } n \equiv 0 \pmod{2}, \\ 3n + 1 & \text{if } n \equiv 1 \pmod{2}. \end{cases}$$

a finite number of times to any given number, we will end up at 1 (cf. <https://xkcd.com/710/> (<https://xkcd.com/710/>)).

Implement the checking routine in TensorFlow (i.e. implement some code that given a number, checks that it satisfies Collatz conjecture). Bonus: use a queue.

```
In [21]: tf.reset_default_graph()

number_to_check = 29

# Define graph.
a = tf.Variable(number_to_check, dtype=tf.int32)
pred = tf.equal(0, tf.mod(a, 2))
b = tf.cast(
    tf.cond(
        pred,
        lambda: tf.div(a, 2),
        lambda: tf.add(tf.multiply(a, 3), 1)),
    tf.int32)
assign_op = tf.assign(a, b)

with tf.Session() as session:
    # 1. Implement graph execution.
    pass
```