
TensorFlow

Google TensorFlow

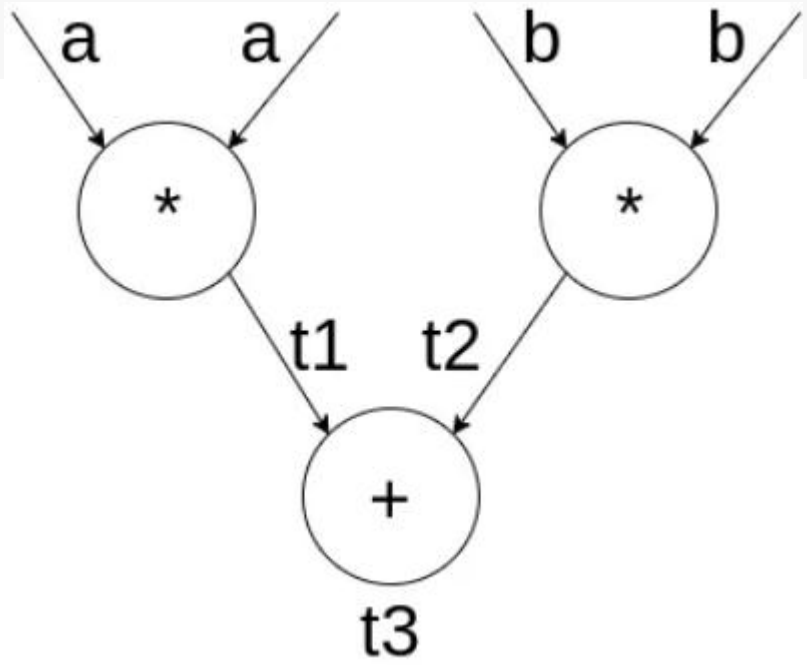
- An open source software library for numerical computation using **data flow graphs**
- Originally developed by the **Google** Brain Team within Google's Machine Intelligence research organization.
- TensorFlow provides primitives for defining functions on **tensor**s and automatically computing their derivatives.
- **Launched Nov 2015**

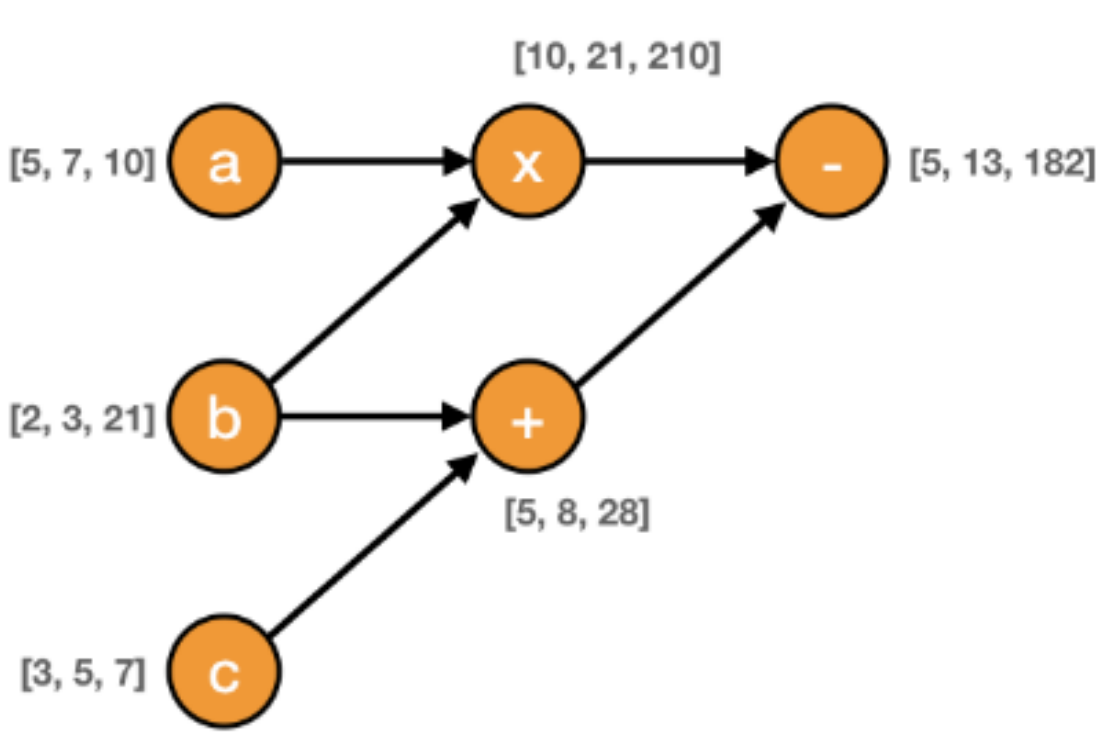
Companies using TensorFlow

Google
OpenAI
DeepMind
Snapchat
Uber
Airbus
eBay
Dropbox
A bunch of startups

DataFlow Graph

It is a common programming model for parallel computing. In a dataflow graph, the nodes represent units of computation, and the edges represent the data consumed or produced by a computation.





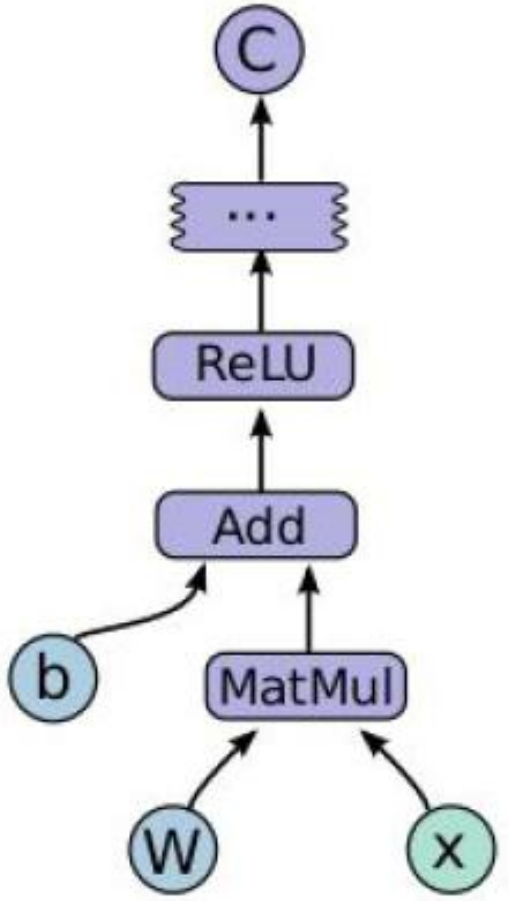
Data Flow Graph

Computations are represented as **graphs**:

- Nodes are the operations (*ops)*
- Edges are the *Tensors* (multidimensional arrays)

Typical program consists of 2 phases:

- **construction phase**: assembling a graph (model)
- **execution phase**:pushing data through the graph



Advantages of Dataflow- leveraged in TensorFlow.

- **Parallelism.** By using explicit edges to represent dependencies between operations, it is easy for the system to identify operations that can execute in parallel.
- **Distributed execution.** By using explicit edges to represent the values that flow between operations, it is possible for TensorFlow to partition your program across multiple devices (CPUs, GPUs, and TPUs) attached to different machines. TensorFlow inserts the necessary communication and coordination between devices.
- **Compilation.** TensorFlow's compiler can use the information in your dataflow graph to generate faster code, for example, by fusing together adjacent operations.
- **Portability.** The dataflow graph is a language-independent representation of the code in your model. You can build a dataflow graph in Python, store it in a SavedModel, and restore it in a C++ program if needed.

TensorFlow Programs

TensorFlow Core programs consist of two discrete sections:

- Building the computational graph (a tf.Graph).
- Running the computational graph (using a tf.Session).

Tensors

Simply put: Tensors can be viewed as a **multidimensional array of numbers**.

This means that:

- A scalar is a tensor,
- A vector is a tensor,
- A matrix is a tensor

t =

Tensor Ranks

In the TensorFlow system, tensors are described by a unit of dimensionality known as *rank*

Rank	Math entity
0	Scalar (magnitude only)
1	Vector (magnitude and direction)
2	Matrix (table of numbers)
3	3-Tensor (cube of numbers)
n	n-Tensor (you get the idea)

Eg: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] Rank is 2

Tensor Shapes

Similar to ndarray shapes in python variable

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

Tensor Data Types

Egs:

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_UINT16	tf.uint16	16 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32 bits floating points: real and imaginary parts.

Graph

A **computational graph** is a series of TensorFlow operations arranged into a graph. The graph is composed of two types of objects.

- Operations (or "ops"): The nodes of the graph. Operations describe calculations that consume and produce tensors.
- Tensors: The edges in the graph. These represent the values that will flow through the graph.

Most TensorFlow functions return tf.Tensors

Program

```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0)
total = a + b
print(a)
print(b)
print(total)
```

The print statements produce

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("add:0", shape=(), dtype=float32)
```

Google TensorFlow Basic Elements

- Tensor
- Variable
- Operation
- Session
- Placeholder
- TensorBoard

Tensor

- TensorFlow programs use a tensor data structure to represent all data
- Think of a TensorFlow tensor as an n-dimensional array or list

In the following example, c, d and e are symbolic Tensor Objects, where as result is a numpy array

```
# Build a dataflow graph.
c = tf.constant([[1.0, 2.0], [3.0, 4.0]])
d = tf.constant([[1.0, 1.0], [0.0, 1.0]])
e = tf.matmul(c, d)

# Construct a `Session` to execute the graph.
sess = tf.Session()

# Execute the graph and store the value that `e` represents in `result`.
result = sess.run(e)
```

Tensor Types

Constant Value Tensors

Sequences

Random Tensors

Constant Value Tensors

Constant Value Tensors

- `tf.zeros(shape, dtype=tf.float32, name=None)`
- `tf.zeros_like(tensor, dtype=None, name=None)`
- `tf.ones(shape, dtype=tf.float32, name=None)`
- `tf.ones_like(tensor, dtype=None, name=None)`
- `tf.fill(dims, value, name=None)`
- `tf.constant(value, dtype=None, shape=None, name=Const)`

Constant Value Tensors- Examples

```
tf.zeros([3, 4], tf.int32)  # [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

```
tensor = tf.constant([[1, 2, 3], [4, 5, 6]])
tf.zeros_like(tensor)  # [[0, 0, 0], [0, 0, 0]]
```

```
# Output tensor has shape [2, 3].
tf.fill([2, 3], 9) ==> [[9, 9, 9]
                        [9, 9, 9]]
```

Sequence Tensors

Sequences

- `tf.linspace(start, stop, num, name=None)`

- `tf.range(start, limit=None, delta=1, name=range)`

tf.linspace

Generates values in an interval.
A sequence of num evenly-spaced values are generated beginning at start.
If num > 1, the values in the sequence increase by stop - start / num - 1,
so that the last one is exactly stop

```
tf.linspace(10.0, 12.0, 3, name="linspace") => [ 10.0  11.0  12.0]
```

tf.range()

```
start = 3
limit = 18
delta = 3
tf.range(start, limit, delta)  # [3, 6, 9, 12, 15]
```

Variable

In-memory buffers containing tensors

Initial value defines the type and shape of the variable.

They must be explicitly initialized and can be saved to disk during and after training.

```
# Create two variables.
weights = tf.Variable(tf.random_normal([784, 200], stddev=0.35),
                      name="weights")
biases = tf.Variable(tf.zeros([200]), name="biases")
```

Operation

- An Operation is a node in a TensorFlow Graph
- Takes zero or more Tensor objects as input, and produces zero or more Tensor objects as output.

c = tf.matmul(a, b)

Creates an Operation of type "MatMul" that takes tensors a and b as input, and produces c as output.

Session & Interactive Session

A class for running TensorFlow operations

InteractiveSession is a TensorFlow Session for use in interactive contexts, such as a shell and Ipython notebook.

```
# Build a graph.
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b

# Launch the graph in a session.
sess = tf.Session()

# Evaluate the tensor `c`.
print(sess.run(c))
```

```
sess = tf.InteractiveSession()
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
# We can just use 'c.eval()' without passing 'sess'
print(c.eval())
sess.close()
```

reduce_sum

```
x = tf.constant([[1, 1, 1], [1, 1, 1]])
tf.reduce_sum(x)  # 6
tf.reduce_sum(x, 0)  # [2, 2, 2]
tf.reduce_sum(x, 1)  # [3, 3]
tf.reduce_sum(x, 1, keepdims=True)  # [[3], [3]]
tf.reduce_sum(x, [0, 1])  # 6
```