TensorFlow Internals

By Mohit Kumar

TensorFlow:Alternatives

- Alternatives include Theano, Torch, Caffe, Neon, and Keras.
- Based on two simple criteria (expressiveness and presence of an active developer community), ultimately narrowed the field of options to Tensor-Flow, Theano (built by the LISA Lab out of the University of Montreal), and Torch (largely maintained by Facebook AI Research).
- One of the drawbacks of **Torch**, however, is that the framework is written in Lua.
- **Theano** drawback is an additional “graph compilation” step that took significant amounts of time while setting up certain kinds of deep learning architectures.
- **TensorFlow** has the cleanest interface as compared to Theano and others. Many classes of models can be expressed in significantly fewer lines without sacrificing the expressiveness of the framework

TensorFlow:When to use

- Researching, developing, and iterating through new machine learning architectures
- Taking models directly from training to deployment
- Implementing existing complex architectures
- Large-scale distributed models
- Create and train models for mobile/embedded systems

TensorFlow:Strengths

- **Usability**
  - The TensorFlow workflow is relatively easy to wrap your head around, and it’s consistent API means that you don’t need to learn an entire new way to work when you try out different models.
  - TensorFlow’s API is stable, and the maintainers fight to ensure that every incorporated change is backwards-compatible.
  - TensorFlow integrates seamlessly with Numpy, which will make most Python-savvy data scientists feel right at home.
  - Unlike some other libraries, TensorFlow does not have any compile time. This allows you to iterate more quickly over ideas without sitting around.
  - There are multiple higher-level interfaces built on top of TensorFlow already, such as Keras and SkFlow. This makes it possible to use the benefits of TensorFlow even if a user doesn’t want to implement the entire model by hand.

TensorFlow:Strengths

- **Usability:API**
  - It provides a very simple Python API called TF.Learn2 (tensorflow.con trib.learn), compatible with Scikit-Learn. Can be used to train various types of neural networks in just a few lines of code. It was previously an independent project called Scikit Flow (or skflow).
  - It also provides another simple API called TF-slim (tensorflow.contrib.slim) to simplify building, training, and evaluating neural networks.
  - Several other high-level APIs have been built independently on top of Tensor-Flow, such as Keras or Pretty Tensor.
  - It includes highly efficient C++ implementations of many ML operations, particularly those needed to build neural networks. There is also a C++ API to define your own high-performance operations.
  - Google also launched a cloud service to run TensorFlow graphs.

TensorFlow:Strengths

- **Flexibility**
  - TensorFlow is capable of running on machines of all shapes and sizes. This allows it to be useful from supercomputers all the way down to embedded systems- and everything in between.
  - It’s distributed architecture allows it to train models with massive datasets in a reasonable amount of time.
  - TensorFlow can utilize CPUs, GPUs, or both at the same time.

TensorFlow:Strengths

- **Efficiency**
  - When TensorFlow was first released, it was surprisingly slow on a number of popular machine learning benchmarks.
  - Since that time, the development team has devoted a on of time and effort into improving the implementation of much of TensorFlow’s code.
  - The result is that TensorFlow now boasts impressive times for much of it’s library, vying for the top spot amongst the open-source machine learning frameworks.
  - TensorFlow’s efficiency is still improving as more and more developers work towards better implementations.
  - **Today it is the quickest**

TensorFlow:Strengths

- **TensorFlow is backed by Google.**
  - Google is throwing a ton of resources into TensorFlow, since it wants TensorFlow to be the lingua franca of machine learning researchers and developers. Additionally, Google uses TensorFlow in its own work daily, and is invested in the continued support of TensorFlow.
  - An incredible community has developed around TensorFlow, and it’s relatively easy to get responses from informed members of the the community or developers on GitHub.
  - Google has released several pre-trained machine learning models in TensorFlow.
  - They are free to use and can allow prototypes to get off the ground without needing massive data pipelines.

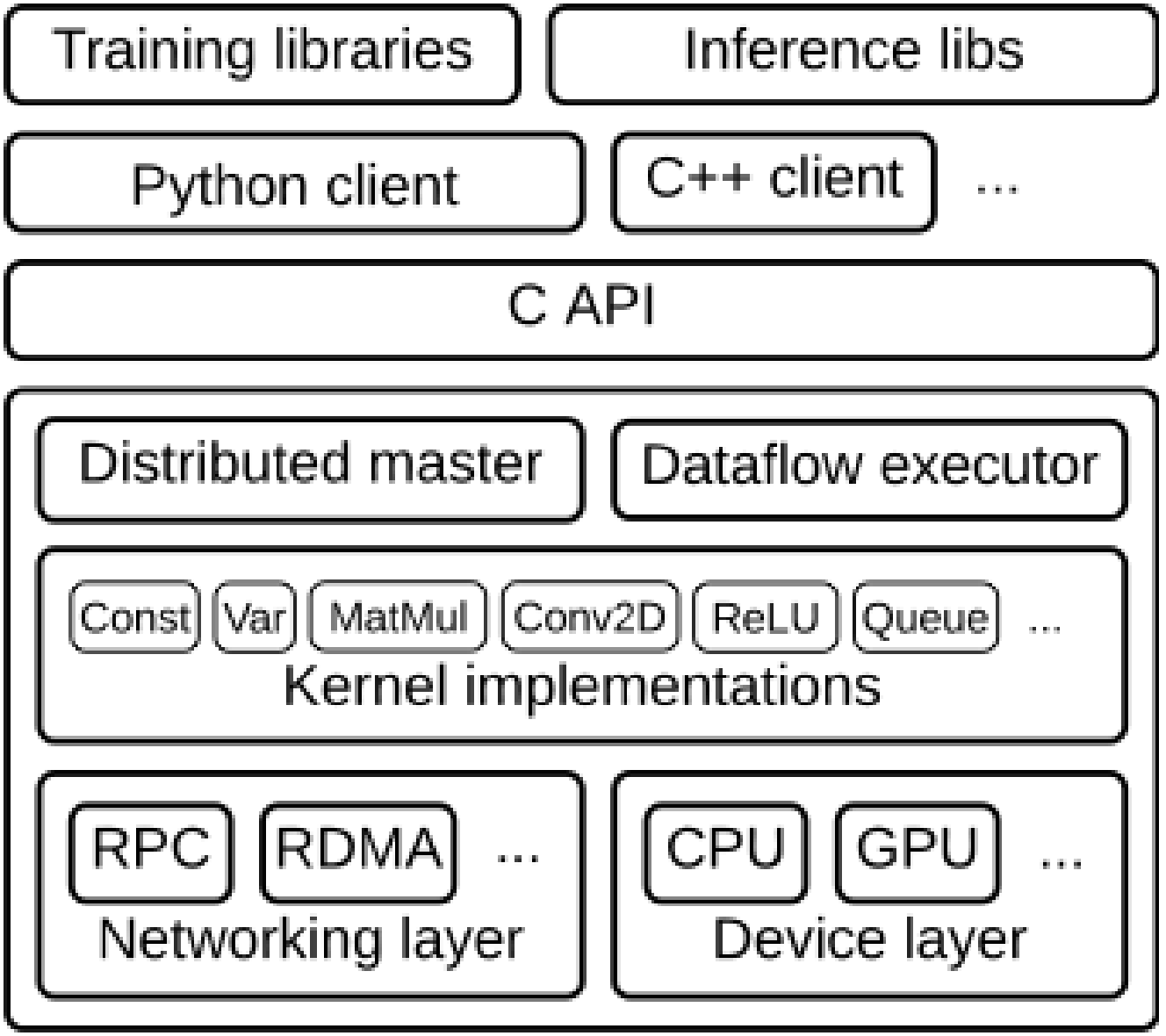
TensorFlow:Strengths

- **Support**
  - **TensorFlow is backed by Google.**
  - Google is throwing a ton of resources into TensorFlow, since it wants TensorFlow to be the lingua franca of machine learning researchers and developers. Additionally, Google uses TensorFlow in its own work daily, and is invested in the continued support of TensorFlow.
  - An incredible community has developed around TensorFlow, and it’s relatively easy to get responses from informed members of the the community or developers on GitHub.
  - Google has released several pre-trained machine learning models in TensorFlow.
  - They are free to use and can allow prototypes to get off the ground without needing massive data pipelines.

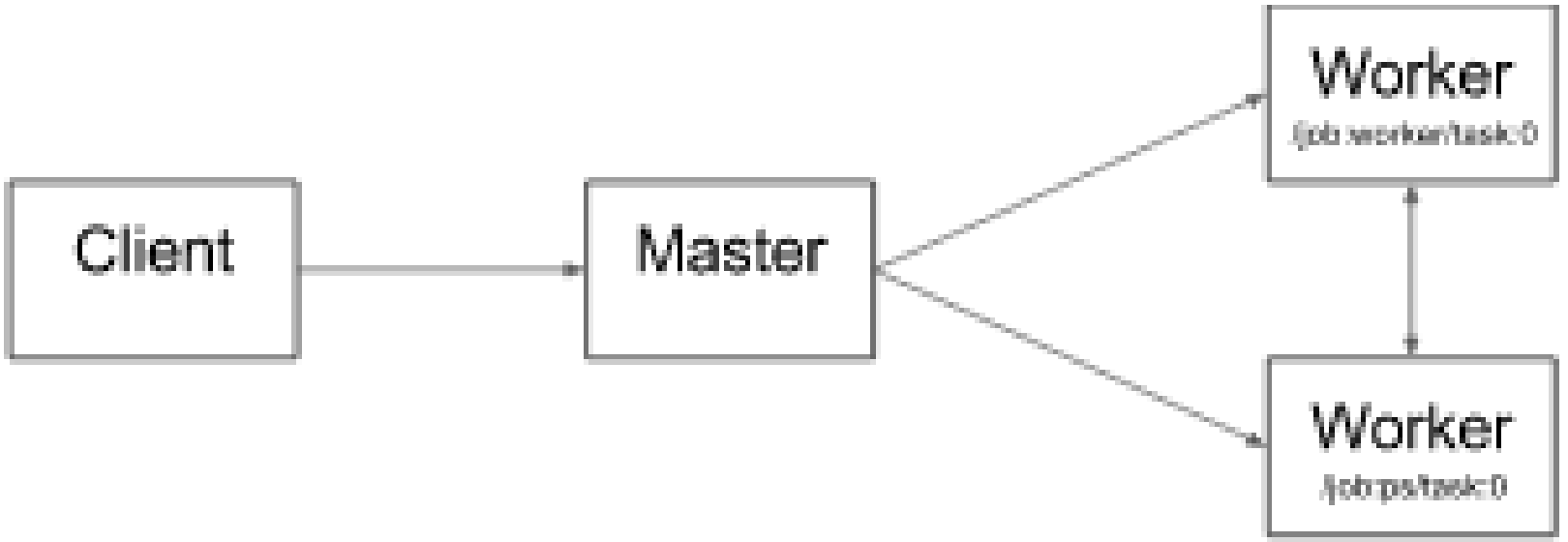
TensorFlow:Strengths

- **Extra features**
  - **TensorBoard** is invaluable when debugging and visualizing your model, and there is nothing quite like it available in other machine learning libraries.
  - TensorFlow Serving may be the piece of software that allows more startup companies to devote services and resources to machine learning, as the cost of reimplementing code in order to deploy a model is no joke.

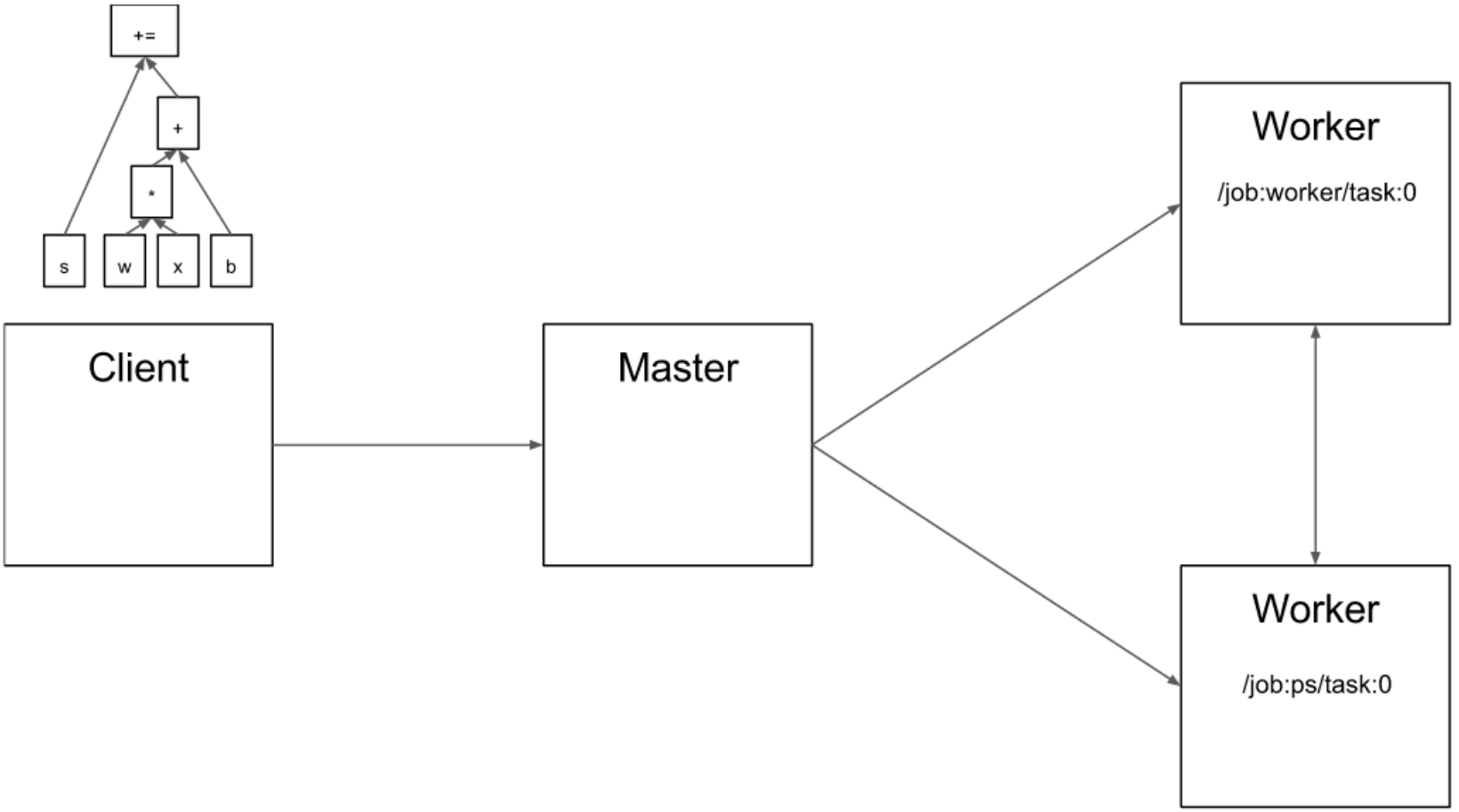
TensorFlow Architecture



TensorFlow Architecture

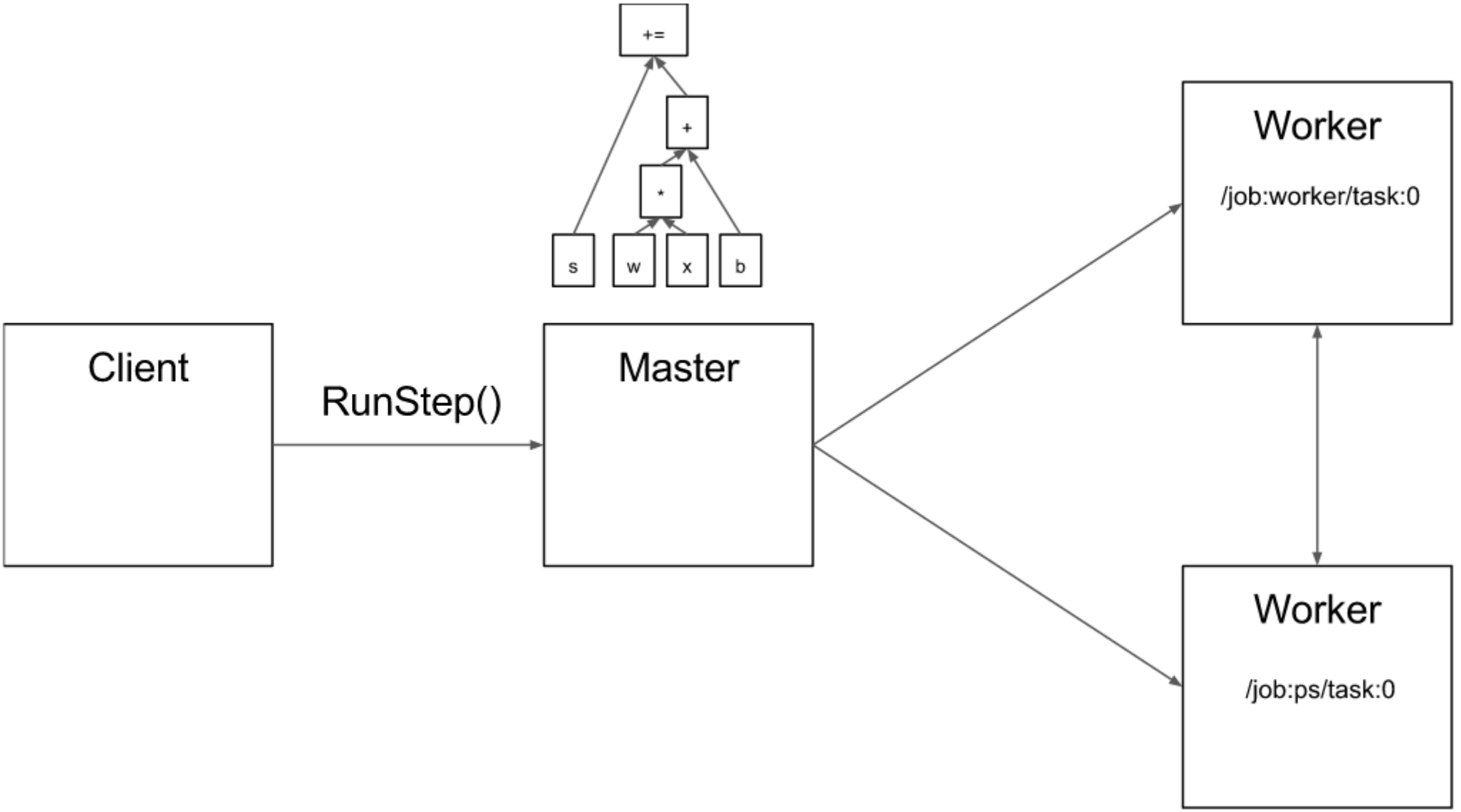


TensorFlow Architecture

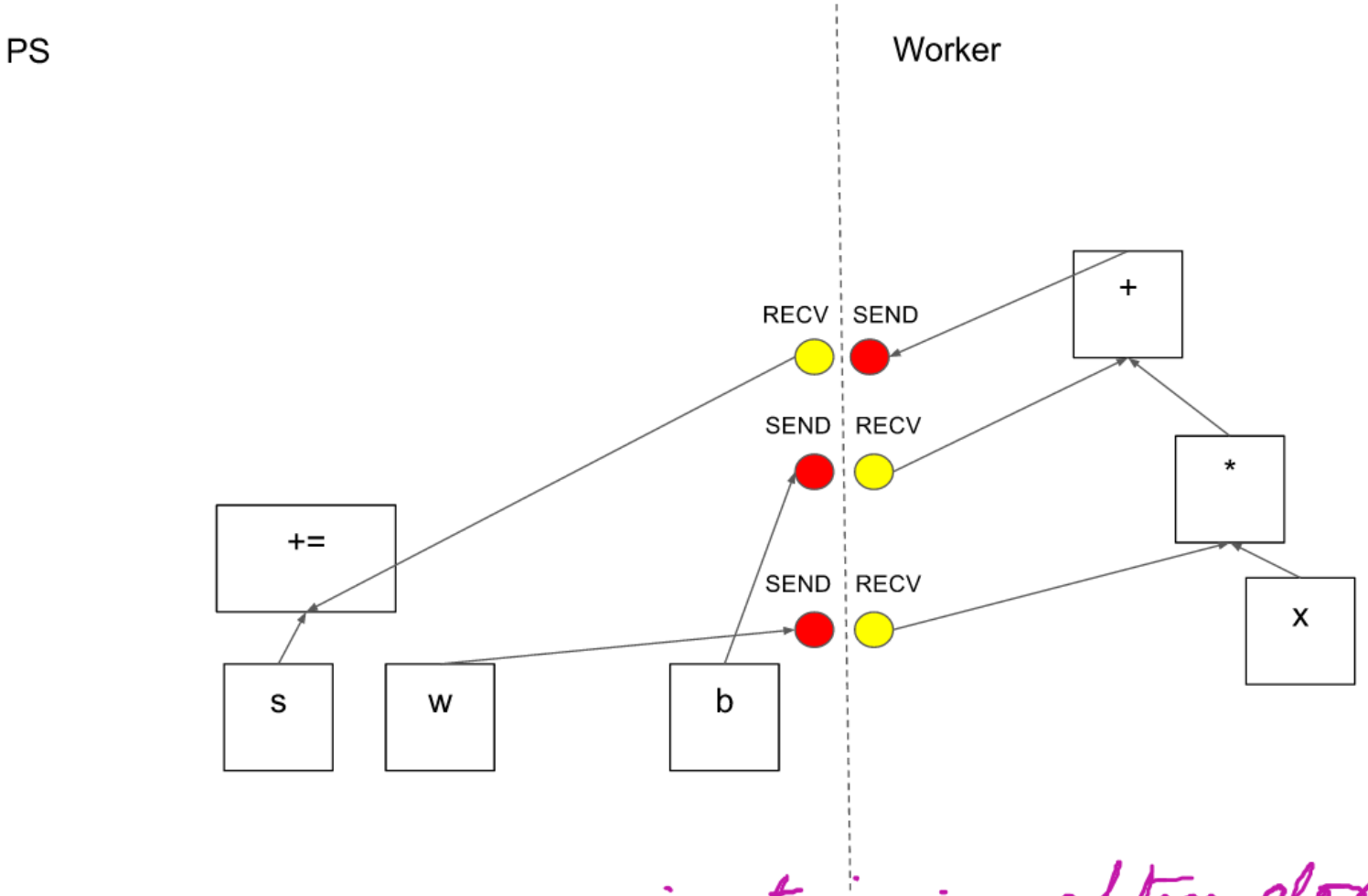


- The client has built a graph that applies a weight(w) to a feature vector(x), adds a bias term(b) and saves the result in a variable (s).

TensorFlow Architecture

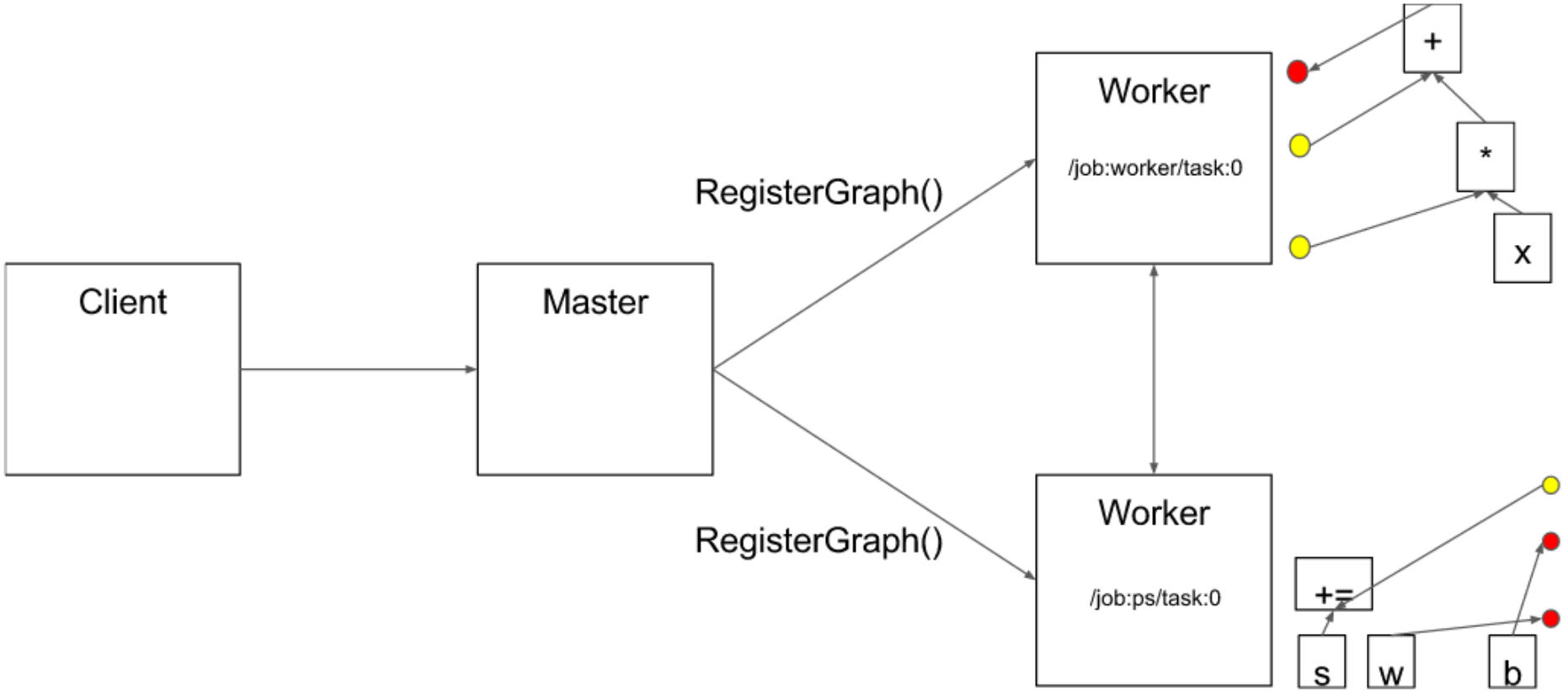


TensorFlow Architecture



- This communication is often done over RDMA, Infiniban, or solar flare, hardware permitting.

TensorFlow Architecture

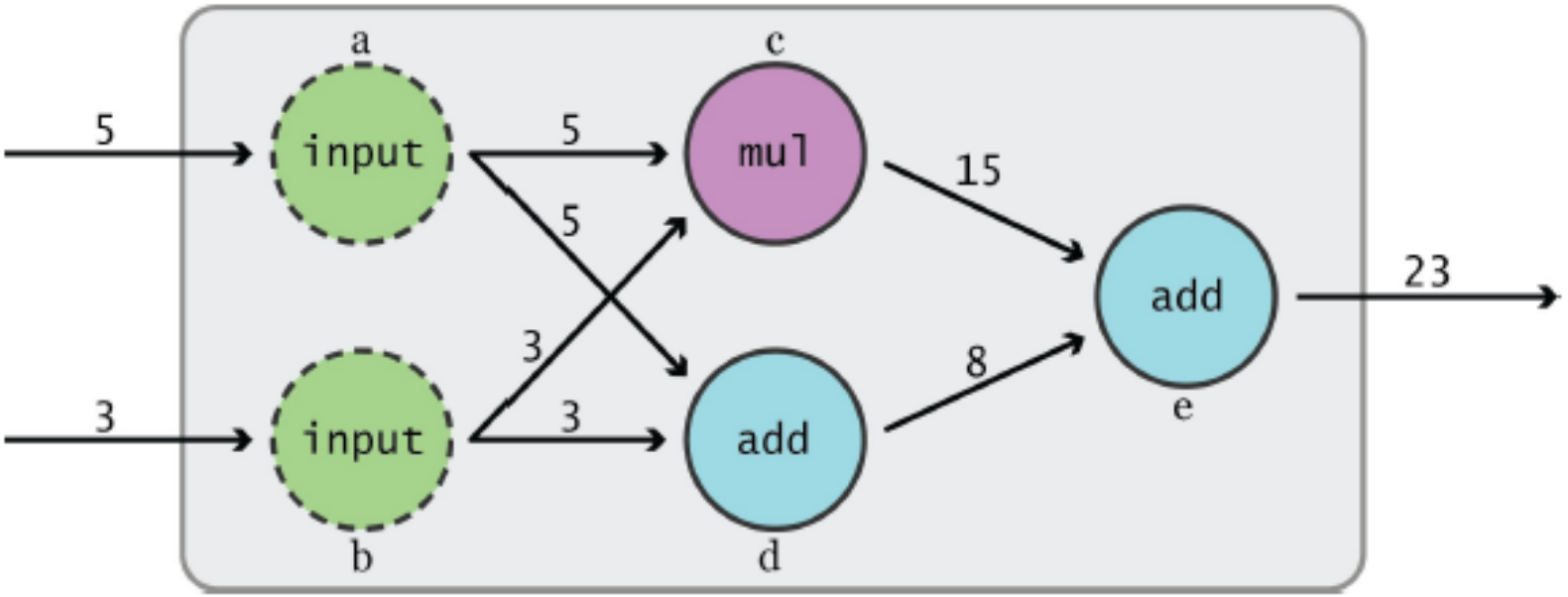


. The master ships the graph peices across to the workers

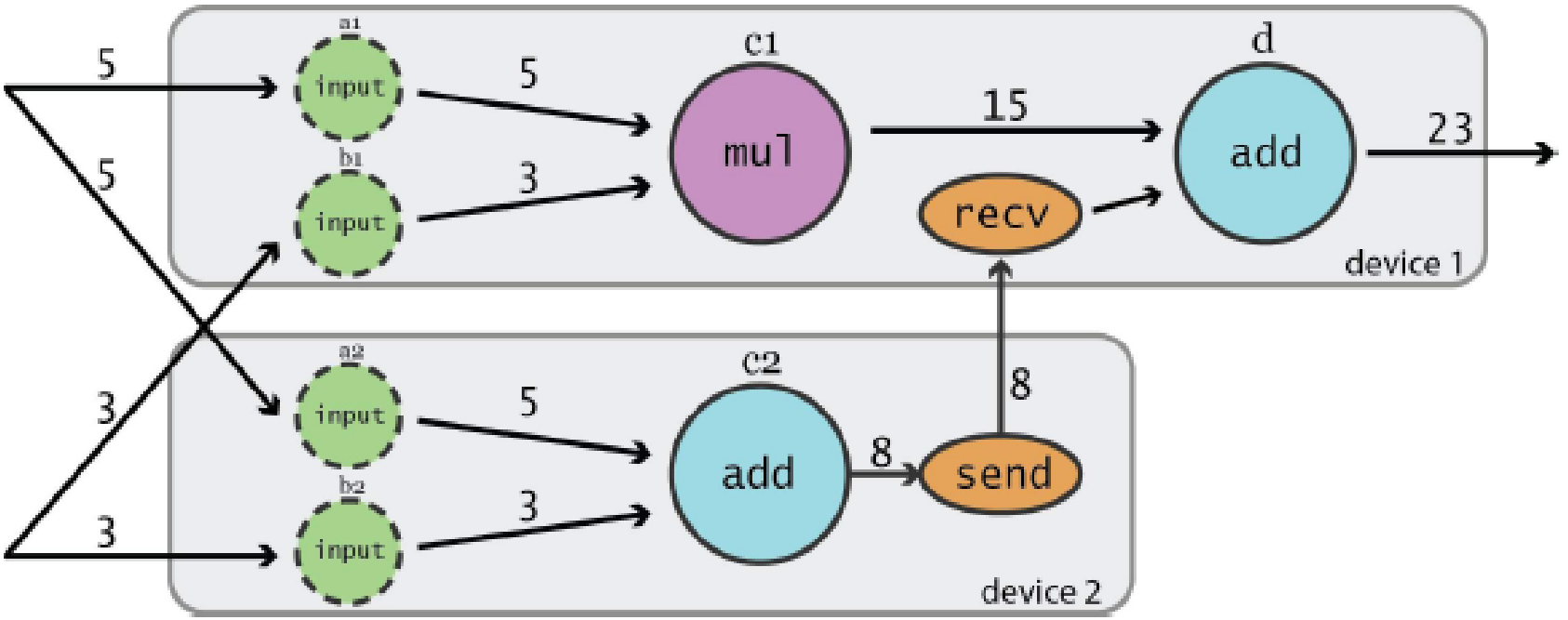
TensorFlow:What is it?

- Looking at the TensorFlow website, the very first words greeting visitors is the following (rather vague) proclamation:
  - ***TensorFlow is an open source software library for machine intelligence.***
- Just below, in the first paragraph under “About TensorFlow,” we are given an alternative description:
  - ***TensorFlow™ is an open source software library for numerical computation using data flow graphs.***
    - Instead of calling itself a “library for machine learning”, it uses the broader term “numerical computation.”
    - While TensorFlow does have extensive support for ML-specific functionality, it is just as well suited to **performing complex mathematical computations**.

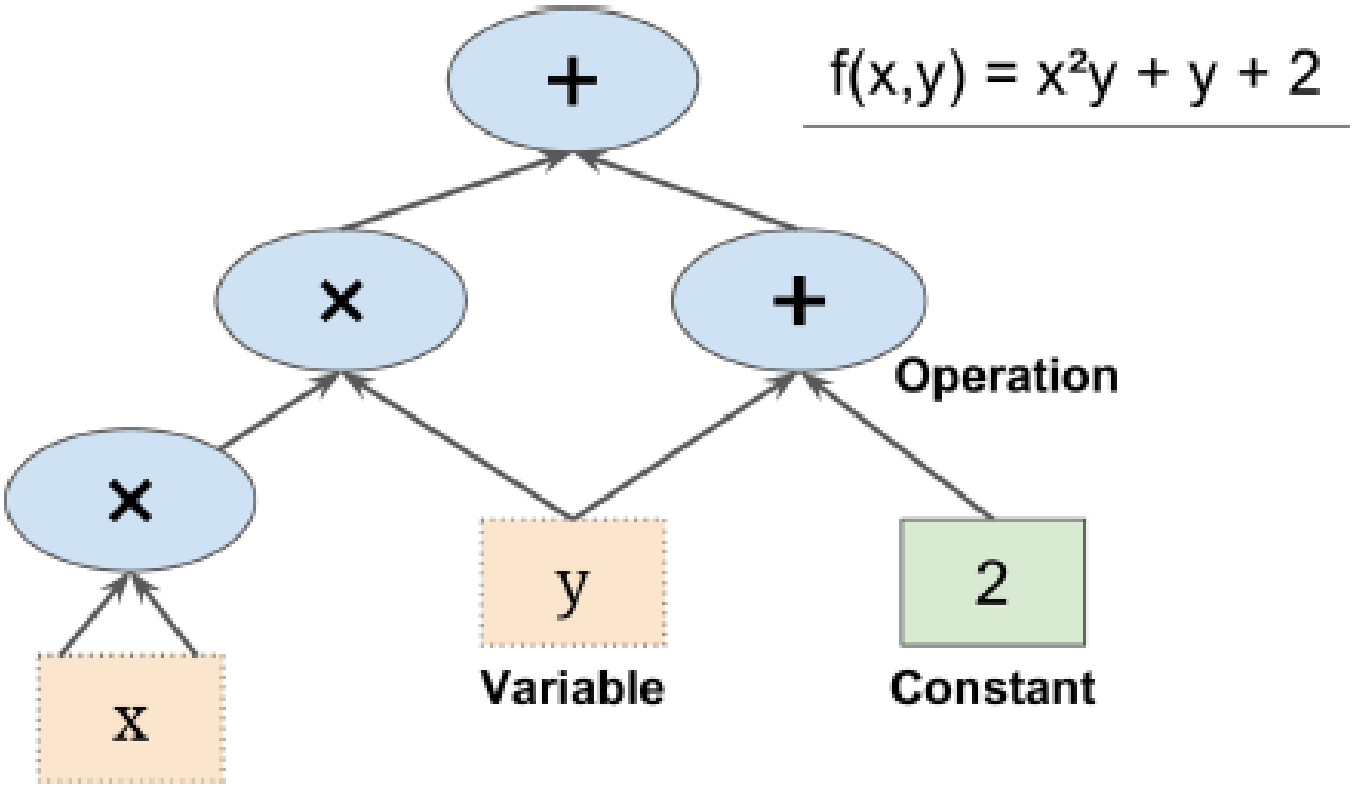
TensorFlow:Data flow graph



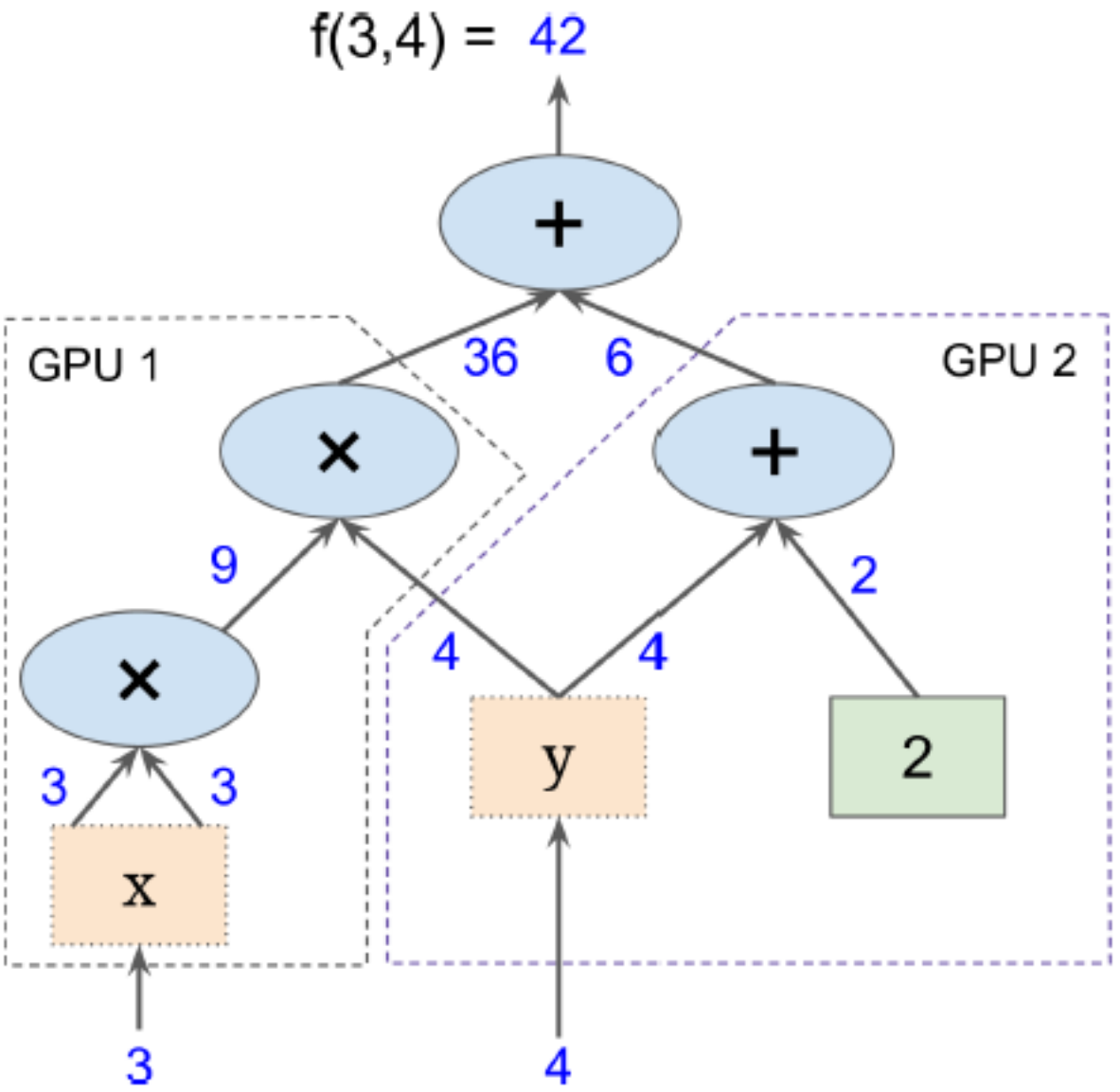
TensorFlow:Data flow graph



TensorFlow:Data flow graph



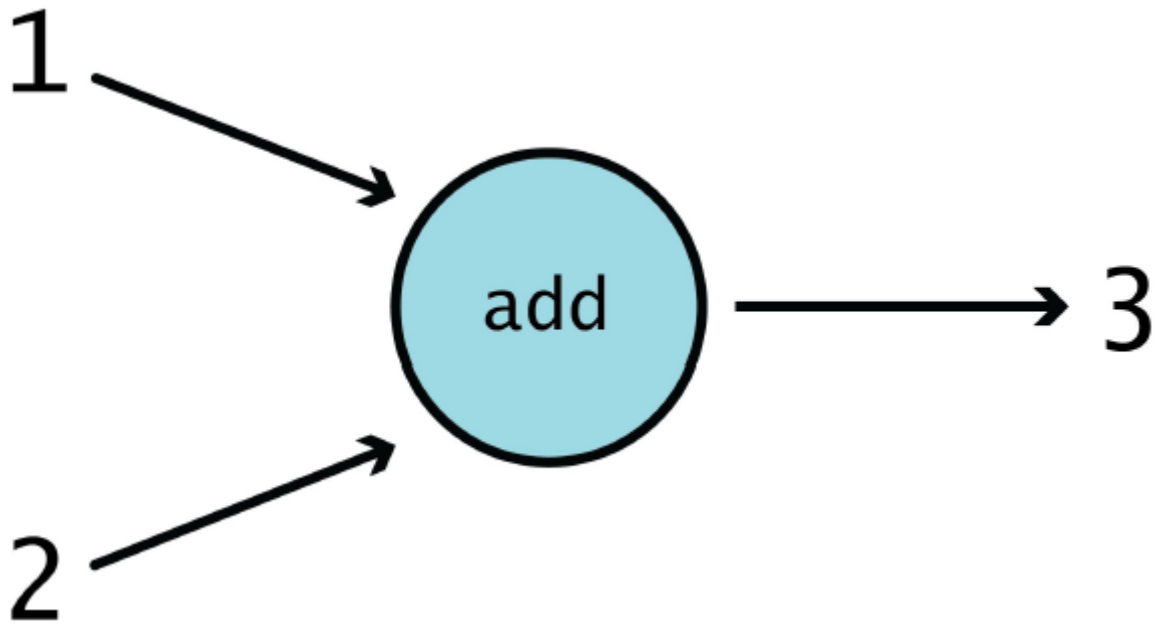
TensorFlow:Data flow graph



TensorFlow:Data flow graph

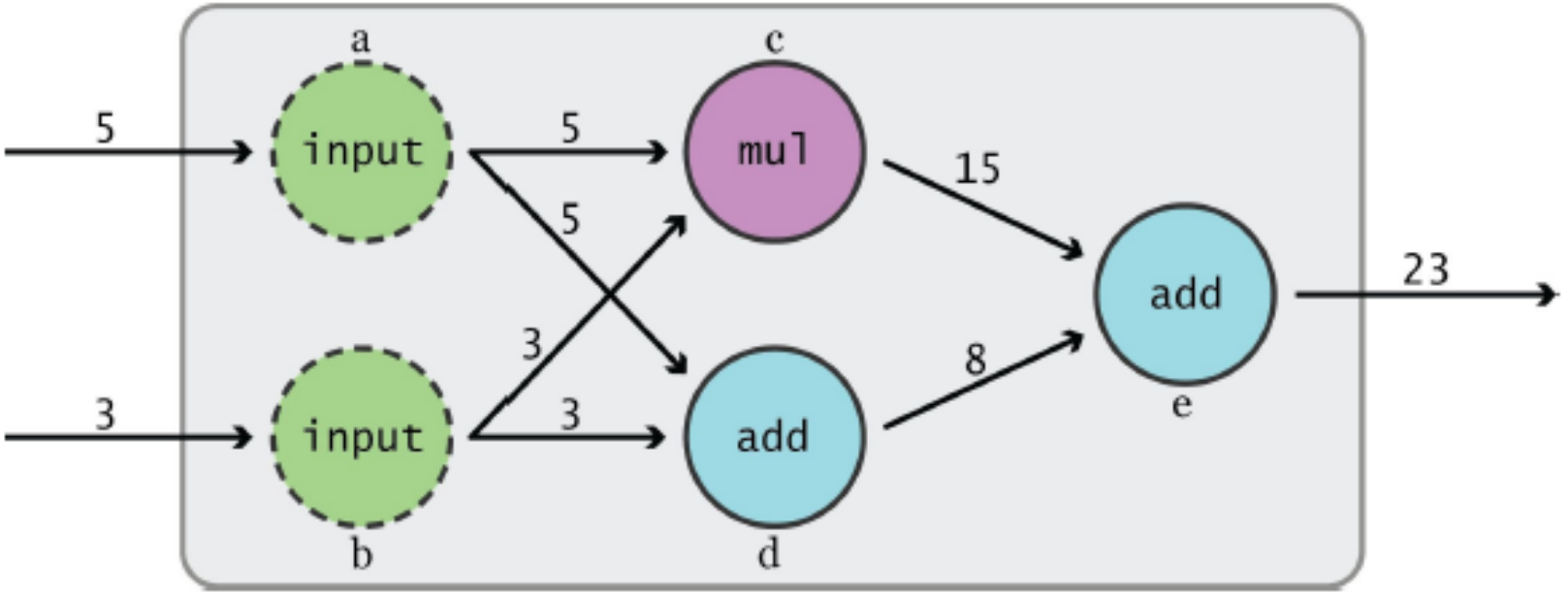
- **what is a tensor?**
  - A tensor, put simply, is an n-dimensional matrix. So a 2-dimensional tensor is the same as a standard matrix.
  - Visually, if we view an matrix as a square array of numbers (m numbers tall, and m numbers wide), we can view an tensor as a cube array of numbers (m numbers tall, m numbers wide, and m numbers deep).
  - In general, you can think about tensors the same way you would matrices, if you are morecomfortable with matrix math!

TensorFlow:Data flow graph



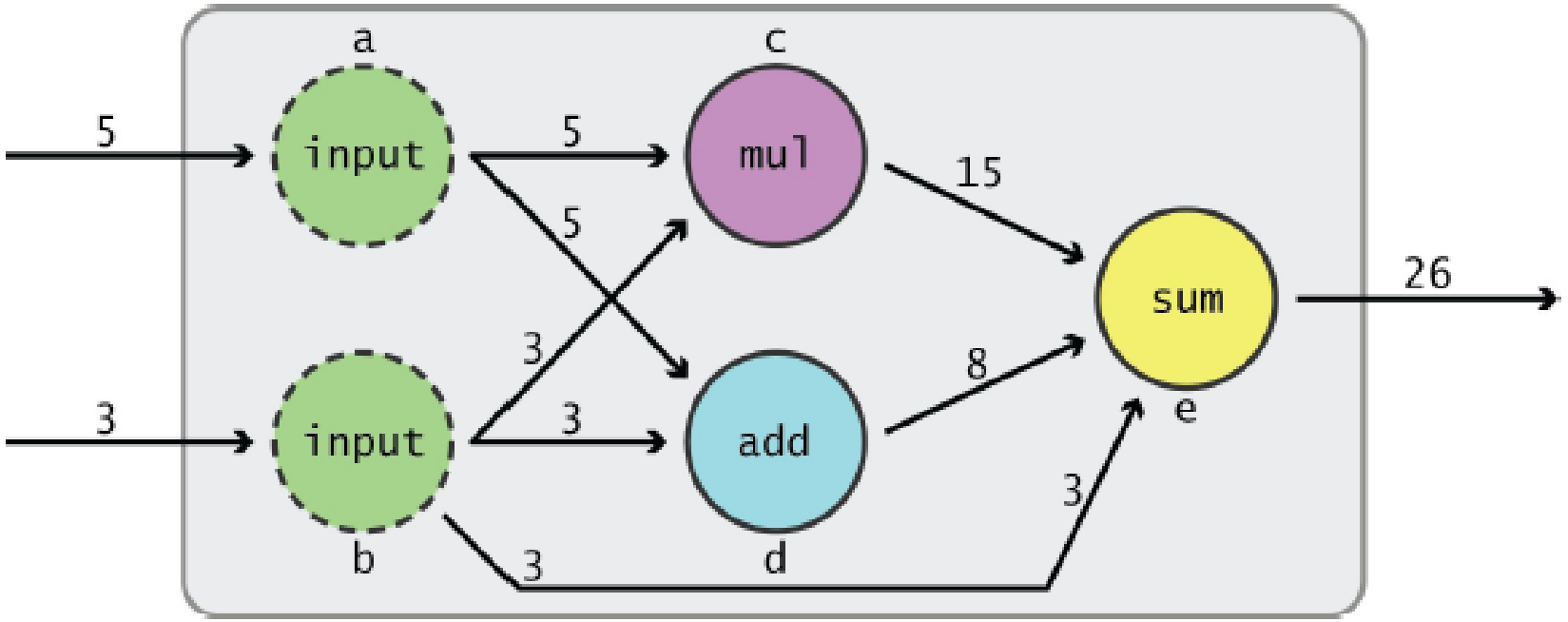
$$f(1, 2) = 1 + 2 = 3$$

TensorFlow:Data flow graph

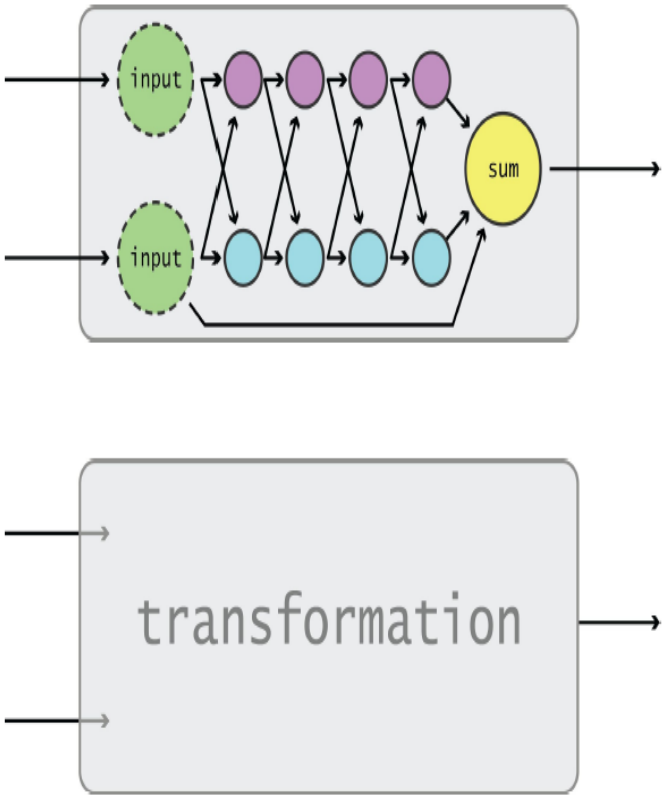


$$a = input_1;\ b = input_2$$
$$c = a \cdot b;\ d = a + b$$
$$e = c + d$$

TensorFlow:Data flow graph

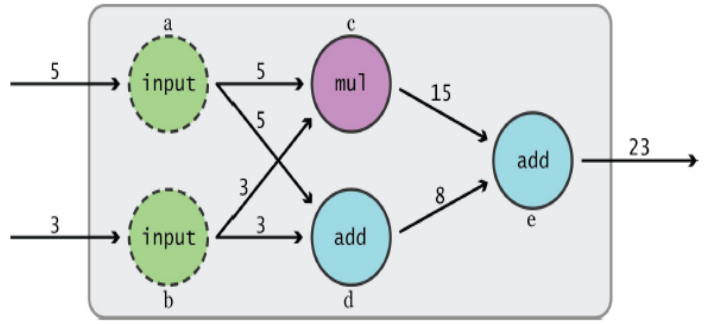


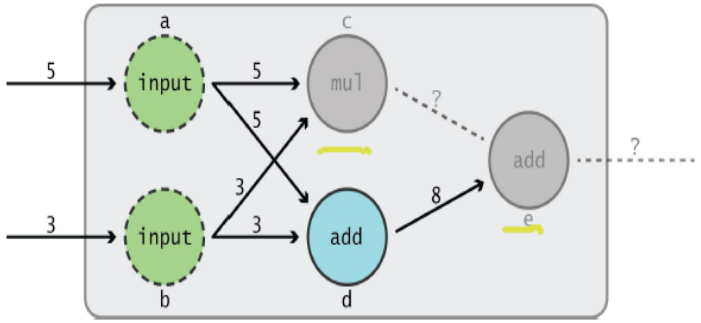
TensorFlow:Data flow graph

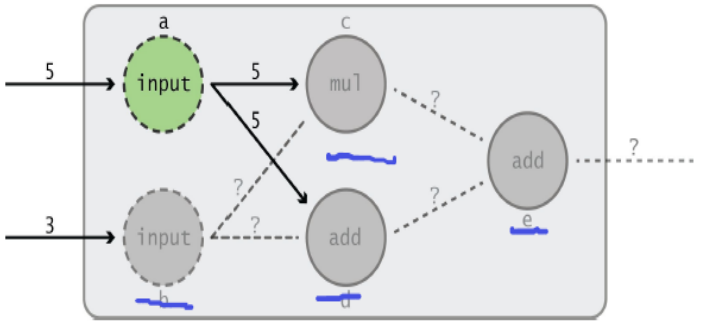


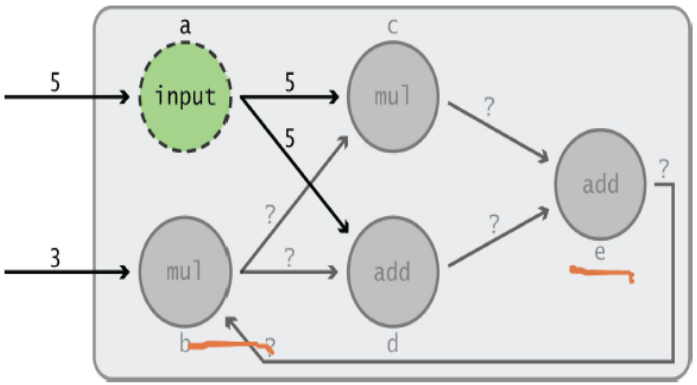
- In general any node could have passed the value to any node in the future and the graph would be valid.
- As in case of threads in OS if there is no dependency between 2 parallel "flows" they can be scheduled parallelly or even distributed across.

TensorFlow:Data flow graph:Dependencies









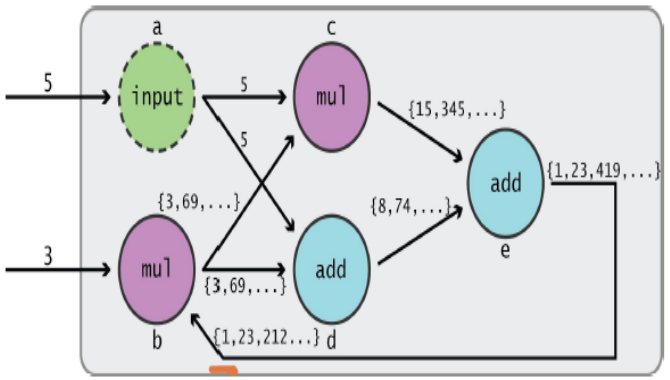
- simple Dataflow graph

- showing direct dependency

- if input is halted, most of the computation grinds to a halt.
- showing transitive dependency

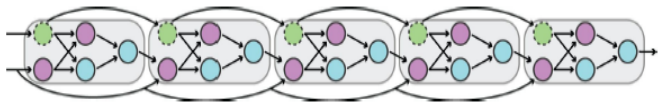
- Interdependant, and it looks like they cannot execute.

TensorFlow:Data flow graph:Dependencies



- To break the dependency, an initial value can be provided.
- However infinite loops are bad for softwares like tensorflow because.
  1. Termination is not graceful
  2. Dependencies tend to infinity, as node's output value changes, it is counted again.
  3. Overflow or underflow of values.

TensorFlow:Data flow graph:Dependencies



- Same graph unrolled.
- In practice we simulate these sort of dependencies by unrolling.
- These graphs help tensorflow optimize computations. For e.g. the value of 'c' does not require the complete graph to execute.

- Circular dependencies cannot be expressed in TF. Which is not a bad thing. Same applies to other libraries.

TensorFlow:API:Simple Example

```python
import tensorflow as tf

tf.reset_default_graph()

x = tf.Variable(3, name="x")
y = tf.Variable(4, name="y")
f = x*x*y + y + 2
print("f:",f)

sess = tf.Session()
sess.run(x.initializer)
sess.run(y.initializer)
print(sess.run(f))
sess.close()
```

- Internal tensor flow representation, not evaluated yet.
- Like other graph processing frameworks evaluates the graph lazily, so that optimizations can be applied.
- Evaluated now

```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./TENSORFLOW/simple.py
f: Tensor("add_1:0", shape=(), dtype=int32)
2017-06-29 08:47:11.429928: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.1 instruct
ions, but these are available on your machine and could speed up CPU computations.
2017-06-29 08:47:11.429949: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use SSE4.2 instruct
ions, but these are available on your machine and could speed up CPU computations.
2017-06-29 08:47:11.429964: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX instruction
s, but these are available on your machine and could speed up CPU computations.
2017-06-29 08:47:11.429971: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use AVX2 instructio
ns, but these are available on your machine and could speed up CPU computations.
2017-06-29 08:47:11.430028: W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library wasn't compiled to use FMA instruction
s, but these are available on your machine and could speed up CPU computations.
42
```

- Warning that tensorflow can be compiled with vector processing instructions that this hardware offers.

TensorFlow:API:with Session

```python
import tensorflow as tf

tf.reset_default_graph()

x = tf.Variable(3, name="x")
y = tf.Variable(4, name="y")
f = x*x*y + y + 2
print("f:",f)

with tf.Session() as sess:
    x.initializer.run()
    y.initializer.run()
    result = f.eval()

print(result)
```

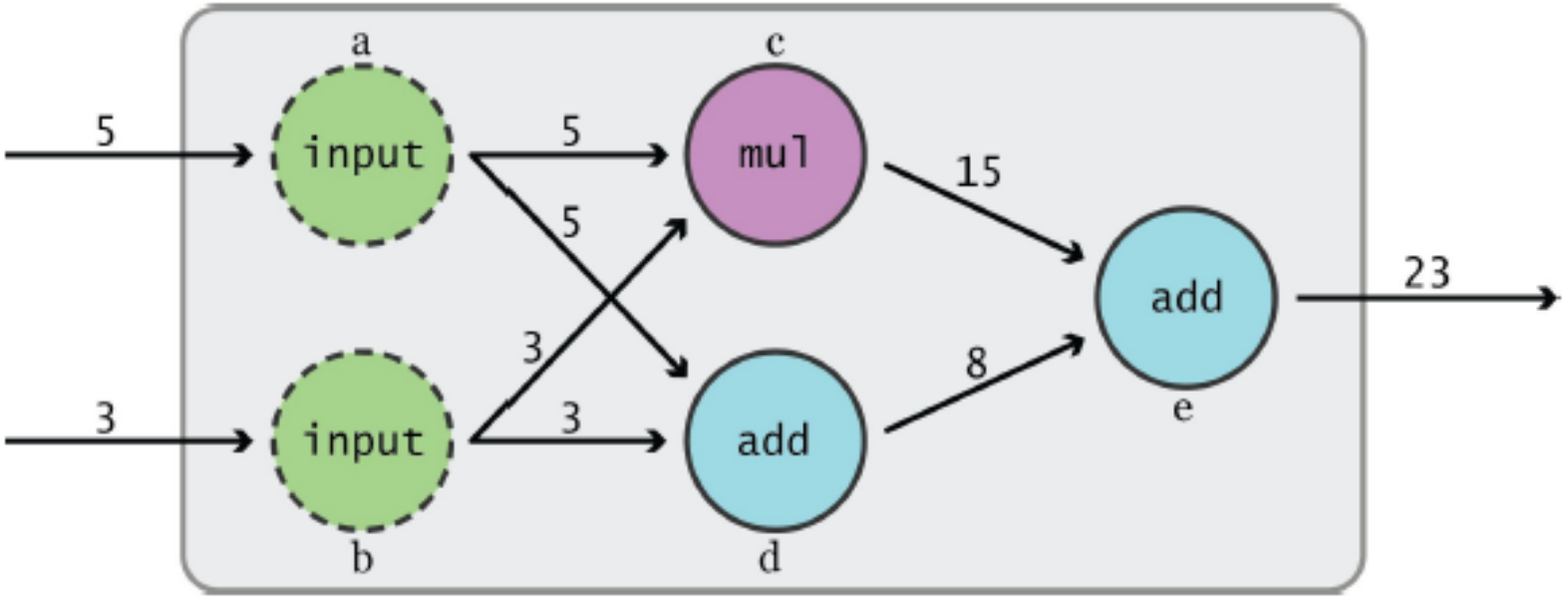
- Convenient way of running.

TensorFlow:API:global variable initializer

```python
import tensorflow as tf

tf.reset_default_graph()

x = tf.Variable(3, name="x")
y = tf.Variable(4, name="y")
f = x*x*y + y + 2
print("f:",f)

#prepare init node
init = tf.global_variables_initializer()


with tf.Session():
    init.run() # actually initialize variables
    result = f.eval()

print(result)
```

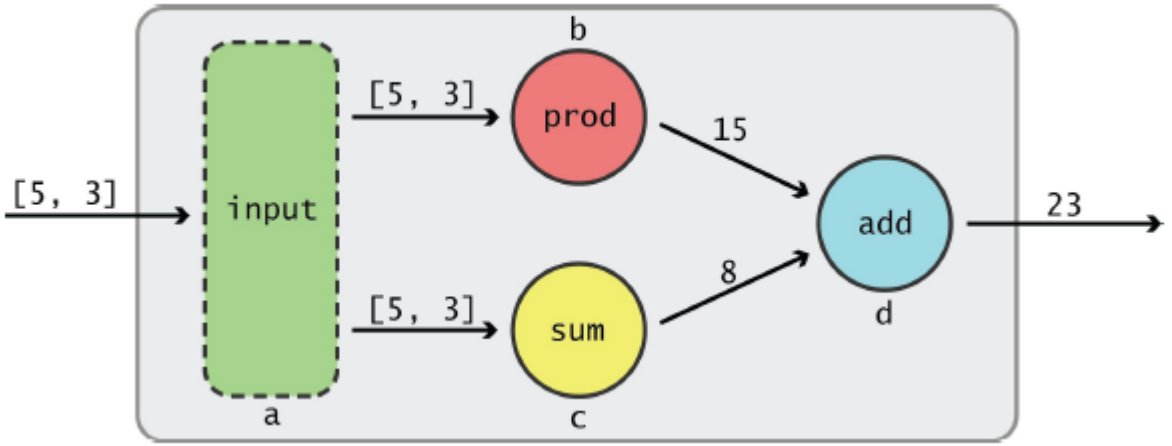
- Initialize all variables at once.

TensorFlow:API:Thinking in tensors




```
a = tf.constant(5, name="input_a")
b = tf.constant(3, name="input_b")
c = tf.mul(a,b, name="mul_c")
d = tf.add(a,b, name="add_d")
e = tf.add(c,d, name="add_e")
```

TensorFlow:API:Thinking in tensors




```
a = tf.constant([5,3], name="input_a")
b = tf.reduce_prod(a, name="prod_b")
c = tf.reduce_sum(a, name="sum_c")
d = tf.add(b,c, name="add_d")
```

TensorFlow:API:Thinking in tensors

```
t_0 = 50                              # Treated as 0-D Tensor, or "scalar"

t_1 = [b"apple", b"peach", b"grape"] # Treated as 1-D Tensor, or "vector"

t_2 = [[True, False, False],          # Treated as 2-D Tensor, or "matrix"
       [False, False, True],
       [False, True, False]]

t_3 = [[ [0, 0], [0, 1], [0, 2] ],    # Treated as 3-D Tensor
       [ [1, 0], [1, 1], [1, 2] ],
       [ [2, 0], [2, 1], [2, 2] ]]
...
```

TensorFlow:API:Data Types

Data type (dtype)	Description
`tf.float32`	32-bit floating point
`tf.float64`	64-bit floating point
`tf.int8`	8-bit signed integer
`tf.int16`	16-bit signed integer
`tf.int32`	32-bit signed integer
`tf.int64`	64-bit signed integer
`tf.uint8`	8-bit unsigned integer
`tf.string`	String (as bytes array, *not* Unicode)

TensorFlow:API:Data Types

`tf.bool`	Boolean
`tf.complex64`	Complex number, with 32-bit floating point real portion, and 32-bit floating point imaginary portion
`tf.qint8`	8-bit signed integer (used in quantized Operations)
`tf.qint32`	32-bit signed integer (used in quantized Operations)
`tf.quint8`	8-bit unsigned integer (used in quantized Operations)

TensorFlow:API:Data Types:Numpy

- TensorFlow is tightly integrated with NumPy, the scientific computing package designed for manipulating N-dimensional arrays.
- As a bonus, you can use the functionality of the numpy library both before and after running your graph, as the tensors returned from Session.run are NumPy arrays.

```
import numpy as np  # Don't forget to import NumPy!

# 0-D Tensor with 32-bit integer data type
t_0 = np.array(50, dtype=np.int32)

# 1-D Tensor with byte string data type
# Note: don't explicitly specify dtype when using strings in NumPy
t_1 = np.array([b"apple", b"peach", b"grape"])

# 1-D Tensor with boolean data type
t_2 = np.array([[True, False, False],
                [False, False, True],
                [False, True, False]],
                dtype=np.bool)

# 3-D Tensor with 64-bit integer data type
t_3 = np.array([[ [0, 0], [0, 1], [0, 2] ],
                [ [1, 0], [1, 1], [1, 2] ],
                [ [2, 0], [2, 1], [2, 2] ]],
                dtype=np.int64)
...
```

TensorFlow:API:Data Types:Shapes

```
# Shapes that specify a 0-D Tensor (scalar)
# e.g. any single number: 7, 1, 3, 4, etc.
s_0_list = []
s_0_tuple = ()

# Shape that describes a vector of length 3
# e.g. [1, 2, 3]
s_1 = [3]

# Shape that describes a 3-by-2 matrix
# e.g [[1 ,2],
#      [3, 4],
#      [5, 6]]
s_2 = (3, 2)
```

```
# Shape for a vector of any length:
s_1_flex = [None]

# Shape for a matrix that is any amount of rows tall, and 3 columns wide:
s_2_flex = (None, 3)

# Shape of a 3-D Tensor with length 2 in its first dimension, and variable-
# length in its second and third dimensions:
s_3_flex = [2, None, None]

# Shape that could be any Tensor
s_any = None
```

TensorFlow:API:Operations

```
import tensorflow as tf
import numpy as np

# Initialize some tensors to use in computation
a = np.array([2, 3], dtype=np.int32)
b = np.array([4, 5], dtype=np.int32)

# Use `tf.add()` to initialize an "add" Operation
# The variable `c` will be a handle to the Tensor output of this Op
c = tf.add(a, b)
```

TensorFlow:API:Operations

Operator	Related TensorFlow Operation	Description
-x	tf.neg()	Returns the negative value of each element in x
~x	tf.logical_not()	Returns the logical NOT of each element in x. Only compatible with `Tensor` objects with `dtype` of `tf.bool`
abs(x)	tf.abs()	Returns the absolute value of each element in x

Binary operators

Operator	Related TensorFlow Operation	Description
x + y	tf.add()	Add x and y, element-wise

TensorFlow:API:Operations

x - y	tf.sub()	Subtract y from x, element-wise
x * y	tf.mul()	Multiply x and y, element-wise
x / y (Python 2)	tf.div()	Will perform element-wise integer division when given an integer type tensor, and floating point ("true") division on floating point tensors
x / y (Python 3)	tf.truediv()	Element-wise floating point division (including on integers)
x // y (Python 3)	tf.floordiv()	Element-wise floor division, not returning any remainder from the computation
x % y	tf.mod()	Element-wise modulo
x ** y	tf.pow()	The result of raising each element in x to its corresponding element y, element-wise
x < y	tf.less()	Returns the truth table of x < y, element-wise
x <= y	tf.less_equal()	Returns the truth table of x <= y, element-wise
x > y	tf.greater()	Returns the truth table of x > y, element-wise
x >= y	tf.greater_equal()	Returns the truth table of x >= y, element-wise
x & y	tf.logical_and()	Returns the truth table of x & y, element-wise. `dtype` must be `tf.bool`
x | y	tf.logical_or()	Returns the truth table of x | y, element-wise. `dtype` must be `tf.bool`
x ^ y	tf.logical_xor()	Returns the truth table of x ^ y, element-wise. `dtype` must be `tf.bool`

TensorFlow:Variables-1

```
import tensorflow as tf
# Pass in a starting value of three for the variable
my_var = tf.Variable(3, name="my_variable")

add = tf.add(5, my_var)
mul = tf.multiply(8, my_var)

init = tf.global_variables_initializer()
sess = tf.Session()

print(sess.run(init))

var1 = tf.Variable(0, name="initialize_me")
var2 = tf.Variable(1, name="no_initialization")
init = tf.variables_initializer([var1], name="init_var1")
sess = tf.Session()
print(sess.run(init))
```

TensorFlow:Variables-2

```
import tensorflow as tf

# Create variable with starting value of 1
my_var = tf.Variable(1)
# Create an operation that multiplies the variable by 2 each time it is run
my_var_times_two = my_var.assign(my_var * 2)
# Initialization operation
init = tf.global_variables_initializer()
# Start a session
sess = tf.Session()
# Initialize variable
sess.run(init)
# Multiply variable by two and return it
print(sess.run(my_var_times_two))
## OUT: 2
# Multiply again
print(sess.run(my_var_times_two))
## OUT: 4
# Multiply again
print(sess.run(my_var_times_two))
## OUT: 8

# Increment by 1
print(sess.run(my_var.assign_add(1)))
# Decrement by 1
print(sess.run(my_var.assign_sub(1)))
```

TensorFlow:Variables-3

```
# Create Ops
my_var = tf.Variable(0)
init = tf.global_variables_initializer()
# Start Sessions
sess1 = tf.Session()
sess2 = tf.Session()
# Initialize Variable in sess1, and increment value of my_var in that Session
sess1.run(init)
sess1.run(my_var.assign_add(5))
## OUT: 5
# Do the same with sess2, but use a different increment value
sess2.run(init)
sess2.run(my_var.assign_add(2))
## OUT: 2
# Can increment the Variable values in each Session independently
print(sess1.run(my_var.assign_add(5)))
## OUT: 10
print(sess2.run(my_var.assign_add(2)))
## OUT: 4
```

TensorFlow:Variables:Life cycle

```
import tensorflow as tf

w = tf.constant(3)
x = w + 2
y = x + 5
z = x * 3

#All node values are dropped between graph runs, except variable values, which are
#maintained by the session across graph runs
# In short, the preceding code evaluates w and x twice.
with tf.Session() as sess:
    print(y.eval())  # 10
    print(z.eval())  # 15

#evaluate y and z efficiently, without evaluating w and x twice as in the
# previous code,
with tf.Session() as sess:
    y_val, z_val = sess.run([y, z])
    print(y_val) # 10
    print(z_val) # 15
```

TensorFlow:fetches

```
tf.reset_default_graph()

a = tf.add(2, 5)
b = tf.multiply(a, 3)
sess = tf.Session()

#identical to above line
sess = tf.Session(graph=tf.get_default_graph())

#fetches accepts any graph element (either an Operation or Tensor object), which specifies
#what the user would like to execute. If the requested object is a Tensor, then the output of
#run() will be a NumPy array.
print("sess.run(b):",sess.run(b))

#When fetches is a list, the output of run() will be a list with values corresponding to the
#output of the requested elements
print("sess.run([a,b]):",sess.run([a,b]))

#If the object is an Operation, then the output will be None.
# Performs the computations needed to initialize Variables, but returns `None`
print(sess.run(tf.initialize_all_variables()))
```

TensorFlow:feed dictionary

```
# Create Operations, Tensors, etc (using the default graph)
a = tf.add(2, 5)
b = tf.multiply(a, 3)
# Start up a `Session` using the default graph
sess = tf.Session()
# Define a dictionary that says to replace the value of `a` with 15
replace_dict = {a: 15}
# Run the session, passing in `replace_dict` as the value to `feed_dict`
print(sess.run(b, feed_dict=replace_dict)) # returns 45
#This means that if you have a large
#graph and want to test out part of it with dummy values, TensorFlow won't waste time
#with unnecessary computations.
```

TensorFlow:placeholder

```
# Creates a placeholder vector of length 2 with data type int32
a = tf.placeholder(tf.int32, shape=[2], name="my_input")
# Use the placeholder as if it were any other Tensor object
b = tf.reduce_prod(a, name="prod_b")
c = tf.reduce_sum(a, name="sum_c")
# Finish off the graph
d = tf.add(b, c, name="add_d")


# Open a TensorFlow Session
sess = tf.Session()
# Create a dictionary to pass into `feed_dict`
# Key: `a`, the handle to the placeholder's output Tensor
# Value: A vector with value [5, 3] and int32 data type
input_dict = {a: np.array([5, 3], dtype=np.int32)}
# Fetch the value of `d`, feeding the values of `input_vector` into `a`
print(sess.run(d, feed_dict=input_dict))
```

TensorFlow:graphs

```
tf.reset_default_graph()

x1 = tf.Variable(1)
print("x1.graph is tf.get_default_graph():",x1.graph is tf.get_default_graph())

graph = tf.Graph()
with graph.as_default():
    x2 = tf.Variable(2)
    print("x2.graph is tf.get_default_graph():",x2.graph is tf.get_default_graph())

print("x2.graph is tf.get_default_graph():",x2.graph is tf.get_default_graph())
print("x2.graph is graph:",x2.graph is graph)
```

- Usually one graph is enough, but there are complicated cases where multiple graphs may be required.
- Best Practices with graph is shown later.

TensorFlow:graphs:Best practices

Correct - Create new graphs, ignore default graph:

```
import tensorflow as tf

g1 = tf.Graph()
g2 = tf.Graph()

with g1.as_default():
    # Define g1 Operations, tensors, etc.
    ...
with g2.as_default():
    # Define g2 Operations, tensors, etc.
    ...
```

Correct - Get handle to default graph

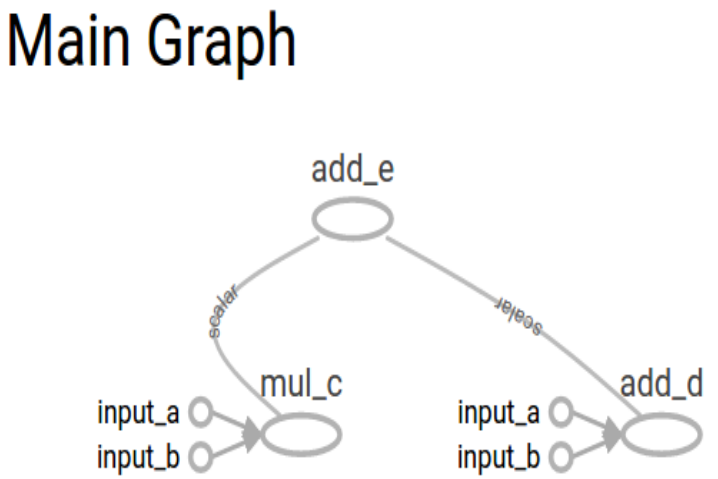
```
import tensorflow as tf

g1 = tf.get_default_graph()
g2 = tf.Graph()

with g1.as_default():
    # Define g1 Operations, tensors, etc.
    ...

with g2.as_default():
    # Define g2 Operations, tensors, etc.
    ...
```

Incorrect: Mix default graph and user-created graph styles

```
import tensorflow as tf

g2 = tf.Graph()

# Define default graph Operations, tensors, etc.
...

with g2.as_default():
    # Define g2 Operations, tensors, etc.
    ...
```

TensorFlow:graphs

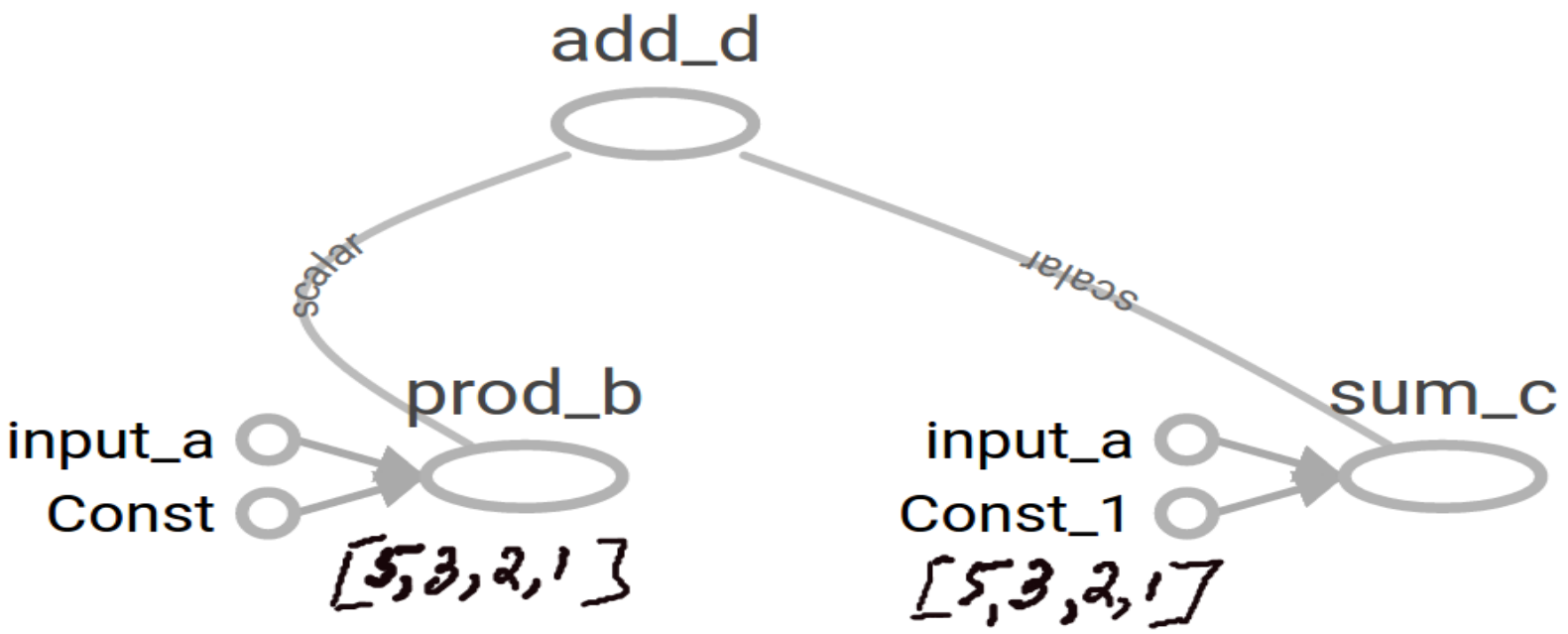
```
WORK_HOME = os.environ['WORK_HOME']
TMP= WORK_HOME+"/resources/tmp"

# Build our graph nodes, starting from the inputs
a = tf.constant(5, name="input_a")
b = tf.constant(3, name="input_b")
c = tf.multiply(a,b, name="mul_c")
d = tf.add(a,b, name="add_d")
e = tf.add(c,d, name="add_e")

# Open up a TensorFlow Session
sess = tf.Session()

# Execute our output node, using our Session
result=sess.run(e)
print("result:",result," result",tf.shape(result))

# Open a TensorFlow SummaryWriter to write our graph to disk
writer = tf.summary.FileWriter(TMP+'/my_graph', sess.graph)


# Close our SummaryWriter and Session objects
writer.close()
sess.close()
```



TensorFlow:graphs

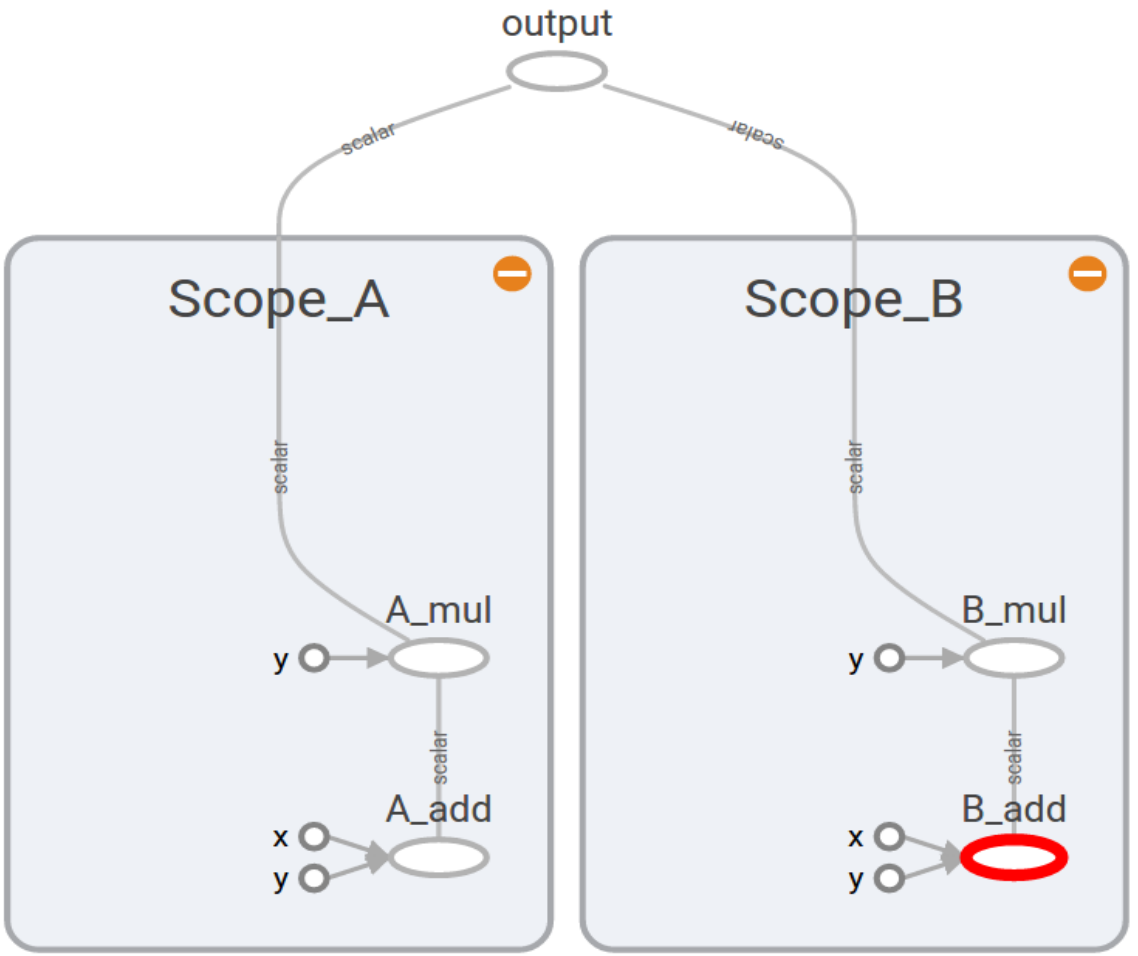
```
# Build our graph nodes, starting from the inputs
a = tf.constant([[5,3],[2,1]], name="input_a")
b = tf.reduce_prod(a, name="prod_b")
c = tf.reduce_sum(a, name="sum_c")
d = tf.add(b,c, name="add_d")

# Open up a TensorFlow Session
sess = tf.Session()

# Execute our output node, using our Session
result=sess.run(d)
print("result:",result," result",tf.shape(result))

# Open a TensorFlow SummaryWriter to write our graph to disk
writer = tf.summary.FileWriter(TMP+'/tensor_graph', sess.graph)

# Close our SummaryWriter and Session objects
writer.close()
sess.close()
```

prints (41)



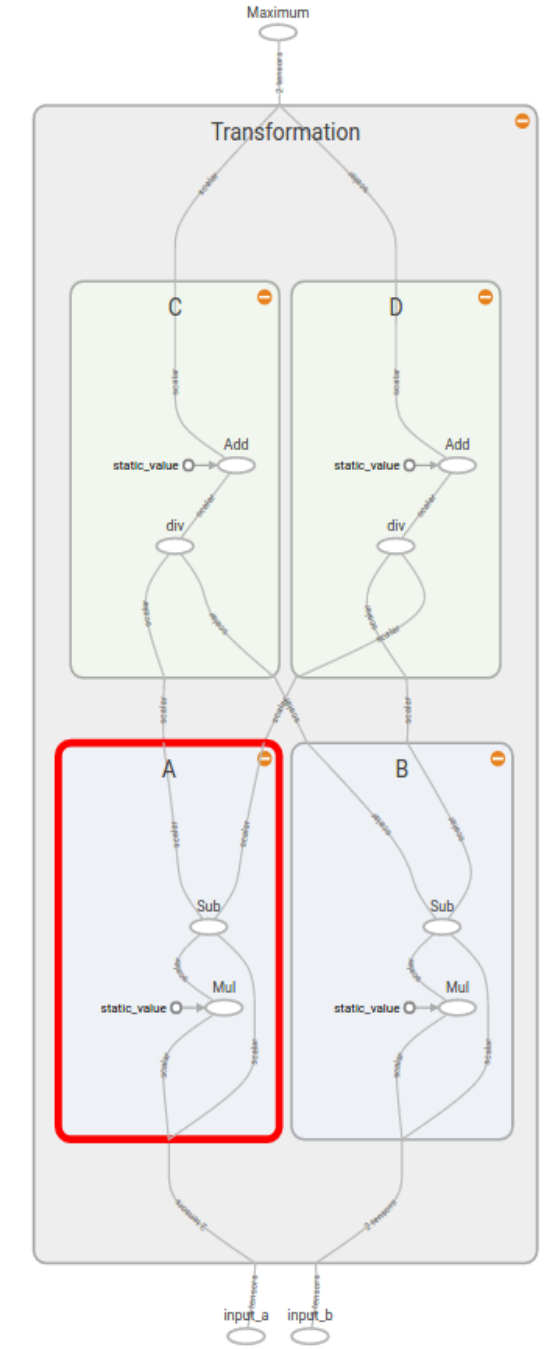
TensorFlow:graphs:named-scope

```
# Example 1
with tf.name_scope("Scope_A"):
    a = tf.add(1, 2, name="A_add")
    b = tf.multiply(a, 3, name="A_mul")

with tf.name_scope("Scope_B"):
    c = tf.add(4, 5, name="B_add")
    d = tf.multiply(c, 6, name="B_mul")

e = tf.add(b, d, name="output")
print("example1:",e)
writer = tf.summary.FileWriter(TMP+'/name_scope_1', graph=tf.get_default_graph())
writer.close()
```



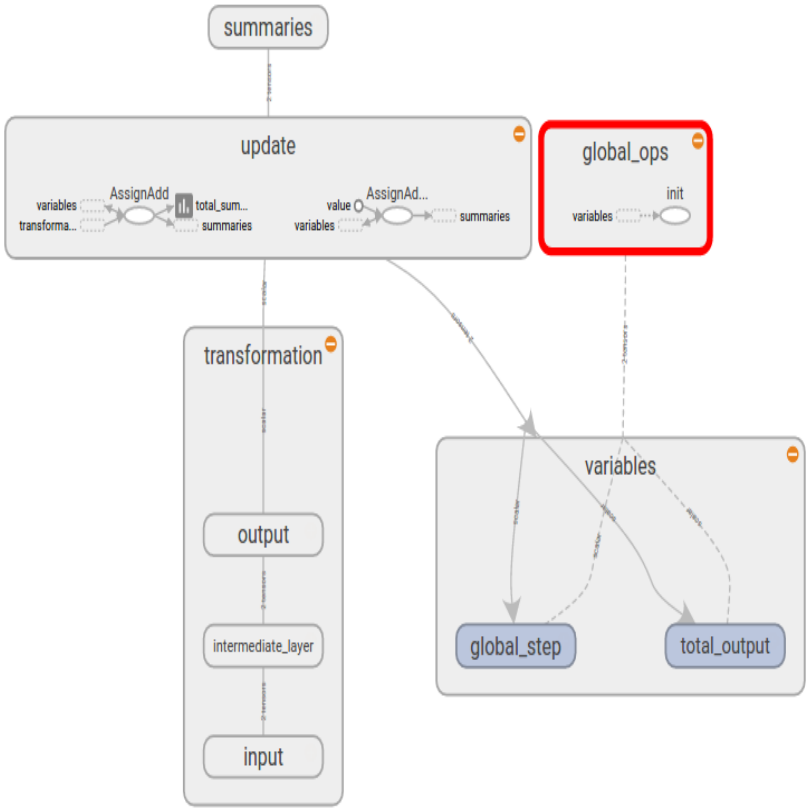
TensorFlow:graphs:named-scope

```python
# Example 2
graph = tf.Graph()
with graph.as_default():
    in_1 = tf.placeholder(tf.float32, shape=[], name="input_a")
    in_2 = tf.placeholder(tf.float32, shape=[], name="input_b")
    const = tf.constant(3, dtype=tf.float32, name="static_value")

    with tf.name_scope("Transformation"):

        with tf.name_scope("A"):
            A_mul = tf.multiply(in_1, const)
            A_out = tf.subtract(A_mul, in_1)

        with tf.name_scope("B"):
            B_mul = tf.multiply(in_2, const)
            B_out = tf.subtract(B_mul, in_2)

        with tf.name_scope("C"):
            C_div = tf.div(A_out, B_out)
            C_out = tf.add(C_div, const)

        with tf.name_scope("D"):
            D_div = tf.div(B_out, A_out)
            D_out = tf.add(D_div, const)

    out = tf.maximum(C_out, D_out)

print("example2:",out)
writer = tf.summary.FileWriter(TMP+'/name_scope_2', graph=graph)
writer.close()
```



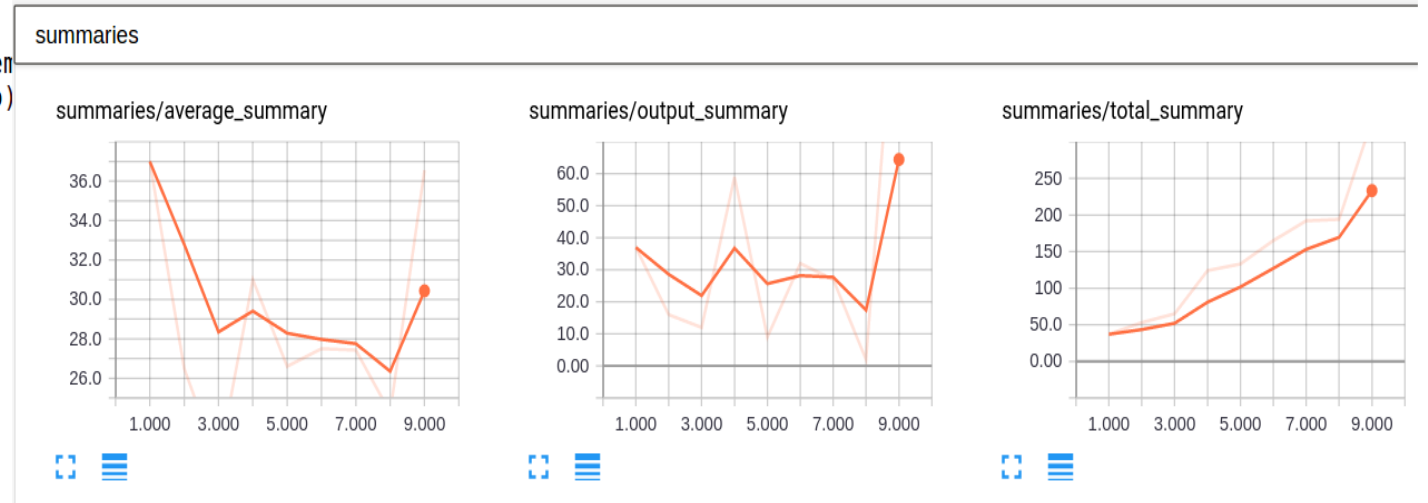
TensorFlow:graphs:running the graph-1

```
# Explicitly create a Graph object
graph = tf.Graph()

with graph.as_default():

    with tf.name_scope("variables"):
        # Variable to keep track of how many times the graph has been run
        global_step = tf.Variable(0, dtype=tf.int32, name="global_step")

        # Variable that keeps track of the sum of all output values over time:
        total_output = tf.Variable(0.0, dtype=tf.float32, name="total_output")

    # Primary transformation Operations
    with tf.name_scope("transformation"):

        # Separate input layer
        with tf.name_scope("input"):
            # Create input placeholder- takes in a Vector
            a = tf.placeholder(tf.float32, shape=[None], name="input_placeholder_a")

        # Separate middle layer
        with tf.name_scope("intermediate_layer"):
            b = tf.reduce_prod(a, name="product_b")
            c = tf.reduce_sum(a, name="sum_c")

        # Separate output layer
        with tf.name_scope("output"):
            output = tf.add(b, c, name="output")

    with tf.name_scope("update"):
        # Increments the total_output Variable by the latest output
        update_total = total_output.assign_add(output)

        # Increments the above `global_step` Variable, should be run whenever the graph is run
        increment_step = global_step.assign_add(1)

    # Summary Operations
    with tf.name_scope("summaries"):
```



TensorFlow:graphs:running the graph-2

```
    with tf.name_scope("summaries"):
        avg = tf.div(update_total, tf.cast(increment_step, tf.float32), name="average")

        # Creates summaries for output node
        tf.summary.scalar("output_summary", output)
        tf.summary.scalar("total_summary", update_total)
        tf.summary.scalar("average_summary", avg)

    # Global Variables and Operations
    with tf.name_scope("global_ops"):
        # Initialization Op
        init = tf.global_variables_initializer()
        # Merge all summaries into one Operation
        merged_summaries = tf.summary.merge_all()

# Start a Session, using the explicitly created Graph
sess = tf.Session(graph=graph)

# Open a SummaryWriter to save summaries
writer = tf.summary.FileWriter(TMP+'/improved_graph', graph)

# Initialize Variables
sess.run(init)

def run_graph(input_tensor):
    """
    Helper function; runs the graph with given input tensor and saves summaries
    """
    feed_dict = {a: input_tensor}
    out, step, summary = sess.run([output, increm
    writer.add_summary(summary, global_step=step)


# Run the graph with various inputs
#run_graph([2,8])
run_graph([3,1,3,3])
run_graph([8])
run_graph([1,2,3])
```



TensorFlow:LinearRegression:Normal

```
housing = fetch_california_housing()
m, n = housing.data.shape
housing_data_plus_bias = np.c_[np.ones((m, 1)), housing.data]

"""tensorflow version
The main benefit of this code versus computing the Normal Equation directly using
NumPy is that TensorFlow will automatically run this on your GPU card if you have
one"""
tf.reset_default_graph()
X = tf.constant(housing_data_plus_bias, dtype=tf.float64, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float64, name="y")
XT = tf.transpose(X)
theta = tf.matmul(tf.matmul(tf.matrix_inverse(tf.matmul(XT, X)), XT), y)
with tf.Session() as sess:
    result = theta.eval()

print("tf:result:",result)

#pure numpy version
X = housing_data_plus_bias
y = housing.target.reshape(-1, 1)
theta_numpy = np.linalg.inv(X.T.dot(X)).dot(X.T).dot(y)

print("numpy:result:",theta_numpy)

#scikit version
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(housing.data, housing.target.reshape(-1, 1))

print("scikit:result:",np.r_[lin_reg.intercept_.reshape(-1, 1), lin_reg.coef_.T])
```

TensorFlow:LinearRegression:GD

```
#tf.constant when values dont change
X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
#tf.constant when values dont change
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
#tf.Variable when values change in place
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42), name="theta")
#theta transpose X
y_pred = tf.matmul(X, theta, name="predictions")
#diff
error = y_pred - y
#MSE
mse = tf.reduce_mean(tf.square(error), name="mse")
#refer to Linear regression slide
gradients = 2/m * tf.matmul(tf.transpose(X), error)
training_op = tf.assign(theta, theta - learning_rate * gradients)

init = tf.global_variables_initializer()
print("theta:",theta)
with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE =", mse.eval())
        sess.run(training_op)

    best_theta = theta.eval()

print("Best theta:")
print(best_theta)
```

TensorFlow:LinearRegression:GD:TennsorFlow:AutoDiff

```
n_epochs = 1000
learning_rate = 0.01

#tf.constant when values dont change
X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
#tf.constant when values dont change
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
#tf.Variable when values change in place
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42), name="theta")
#theta transpose X
y_pred = tf.matmul(X, theta, name="predictions")
#diff
error = y_pred - y
#MSE
mse = tf.reduce_mean(tf.square(error), name="mse")
# Tensor flowdoes an autodiff for gradient.
gradients = tf.gradients(mse, [theta])[0]
training_op = tf.assign(theta, theta - learning_rate * gradients)

init = tf.global_variables_initializer()
print("theta:",theta)
with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE =", mse.eval())
        sess.run(training_op)

    best_theta = theta.eval()

print("Best theta:")
print(best_theta)
```

TensorFlow:LinearRegression:GD:TennsorFlow:AutoDiff:GDOptimizer

```
n_epochs = 1000
learning_rate = 0.01

X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42), name="theta")
y_pred = tf.matmul(X, theta, name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
#tf.train.GradientDescentOptimizer is designed to use a constant learning rate for all variables in all steps
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(mse)

init = tf.global_variables_initializer()
print("theta:",theta)
with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE =", mse.eval())
        sess.run(training_op)

    best_theta = theta.eval()

print("Best theta:")
print(best_theta)
```

TensorFlow:LinearRegression:GD:TennsorFlow:AutoDiff:MomentumOptimizer

```
n_epochs = 1000
learning_rate = 0.01

X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42), name="theta")
y_pred = tf.matmul(X, theta, name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
#momentum helps SGD to navigate along the relevant directions and softens the
#oscillations in the irrelevant. It simply adds a fraction of the direction of
#the previous step to a current step. This achieves amplification of speed in
#the correct dirrection and softens oscillation in wrong directions.
#This fraction is usually in the (0, 1) range
optimizer = tf.train.MomentumOptimizer(learning_rate=learning_rate, momentum=0.25)
training_op = optimizer.minimize(mse)

init = tf.global_variables_initializer()
print("theta:",theta)
with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE =", mse.eval())
        sess.run(training_op)

    best_theta = theta.eval()

print("Best theta:")
print(best_theta)
```

TensorFlow:GDOptimizers

- AdaDelta resolves the problem of monotonically decreasing learning rate in AdaGrad. In AdaGrad the learning rate was calculated approximately as one divided by the sum of square roots. At each stage you add another square root to the sum, which causes denominator to constantly decrease. In AdaDelta instead of summing all past square roots it uses sliding window which allows the sum to decrease. RMSprop is very similar to AdaDelta
- Adam or adaptive momentum is an algorithm similar to AdaDelta. But in addition to storing learning rates for each of the parameters it also stores momentum changes for each of them separately

TensorFlow:Model:Saving

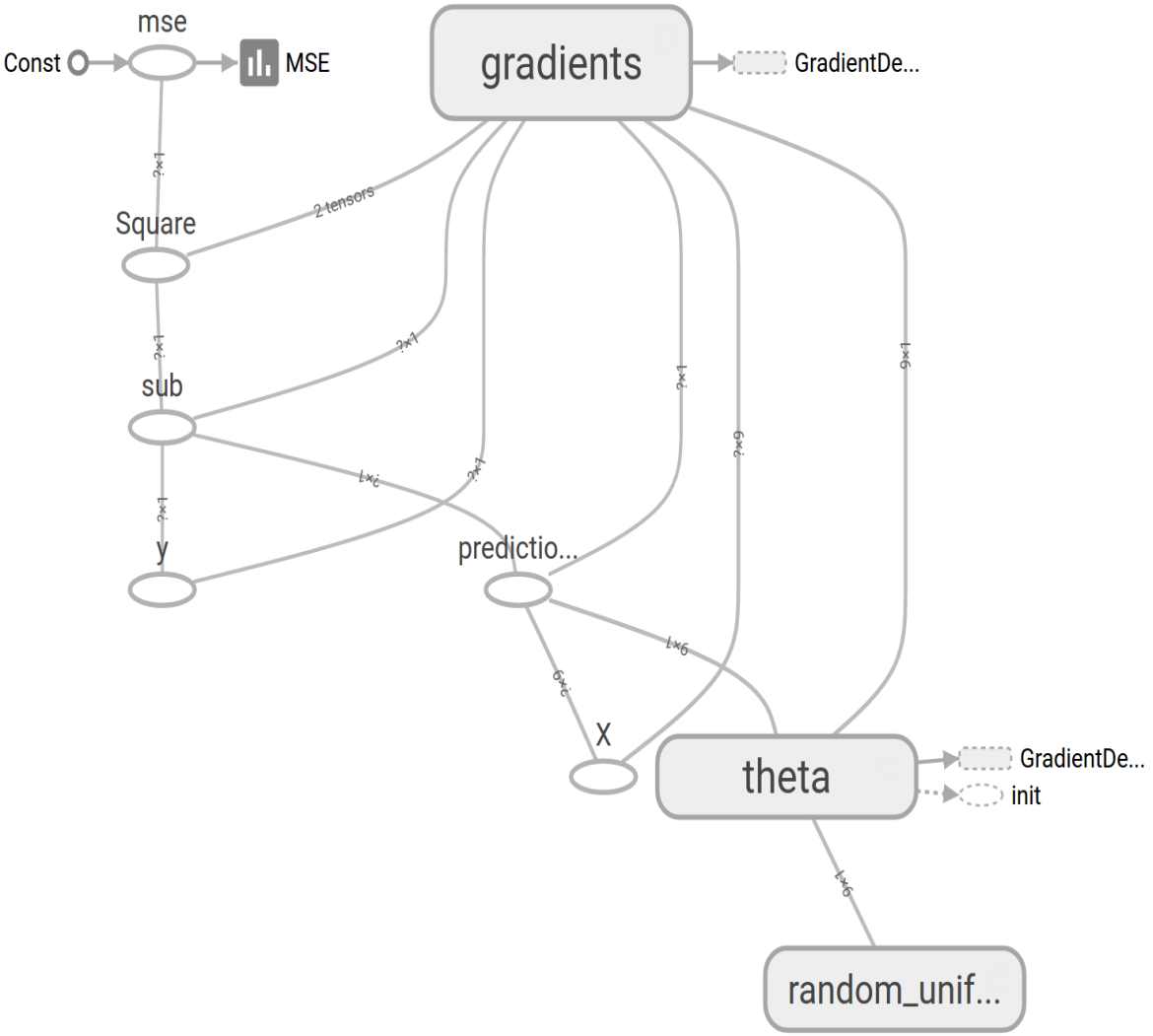
```
housing = fetch_california_housing()
m, n = housing.data.shape
scaler = StandardScaler()
scaled_housing_data = scaler.fit_transform(housing.data)
scaled_housing_data_plus_bias = np.c_[np.ones((m, 1)), scaled_housing_data]

print(scaled_housing_data_plus_bias.mean(axis=0))
print(scaled_housing_data_plus_bias.mean(axis=1))
print(scaled_housing_data_plus_bias.mean())
print(scaled_housing_data_plus_bias.shape)
print("housing shape:",m,n)
tf.reset_default_graph()

n_epochs = 1000
learning_rate = 0.01

X = tf.constant(scaled_housing_data_plus_bias, dtype=tf.float32, name="X")
y = tf.constant(housing.target.reshape(-1, 1), dtype=tf.float32, name="y")
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42), name="theta")
y_pred = tf.matmul(X, theta, name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(mse)

init = tf.global_variables_initializer()
saver = tf.train.Saver()
print("theta:",theta)
with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        if epoch % 100 == 0:
            print("Epoch", epoch, "MSE =", mse.eval())
        sess.run(training_op)

    best_theta = theta.eval()
    save_path = saver.save(sess, TMP+"/my_model_final.ckpt")

print("Best theta:")
print(best theta)
```

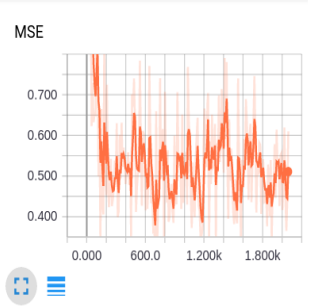
TensorFlow:MGD with placeholders

```
def fetch_batch(epoch, batch_index, batch_size):
    rnd.seed(epoch * n_batches + batch_index)
    indices = rnd.randint(m, size=batch_size)
    X_batch = scaled_housing_data_plus_bias[indices]
    y_batch = housing.target.reshape(-1, 1)[indices]
    return X_batch, y_batch

tf.reset_default_graph()

n_epochs = 1000
learning_rate = 0.01

X = tf.placeholder(tf.float32, shape=(None, n + 1), name="X")
y = tf.placeholder(tf.float32, shape=(None, 1), name="y")
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42), name="theta")
y_pred = tf.matmul(X, theta, name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
gradients = 2/m * tf.matmul(tf.transpose(X), error)
training_op = tf.assign(theta, theta - learning_rate * gradients)

init = tf.global_variables_initializer()
n_epochs = 10
batch_size = 100
n_batches = int(np.ceil(m / batch_size))

with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        for batch_index in range(n_batches):
            X_batch, y_batch = fetch_batch(epoch, batch_index, batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})

    best_theta = theta.eval()

print("Best theta:")
print(best_theta)
```

TensorFlow:Tensor Visualization

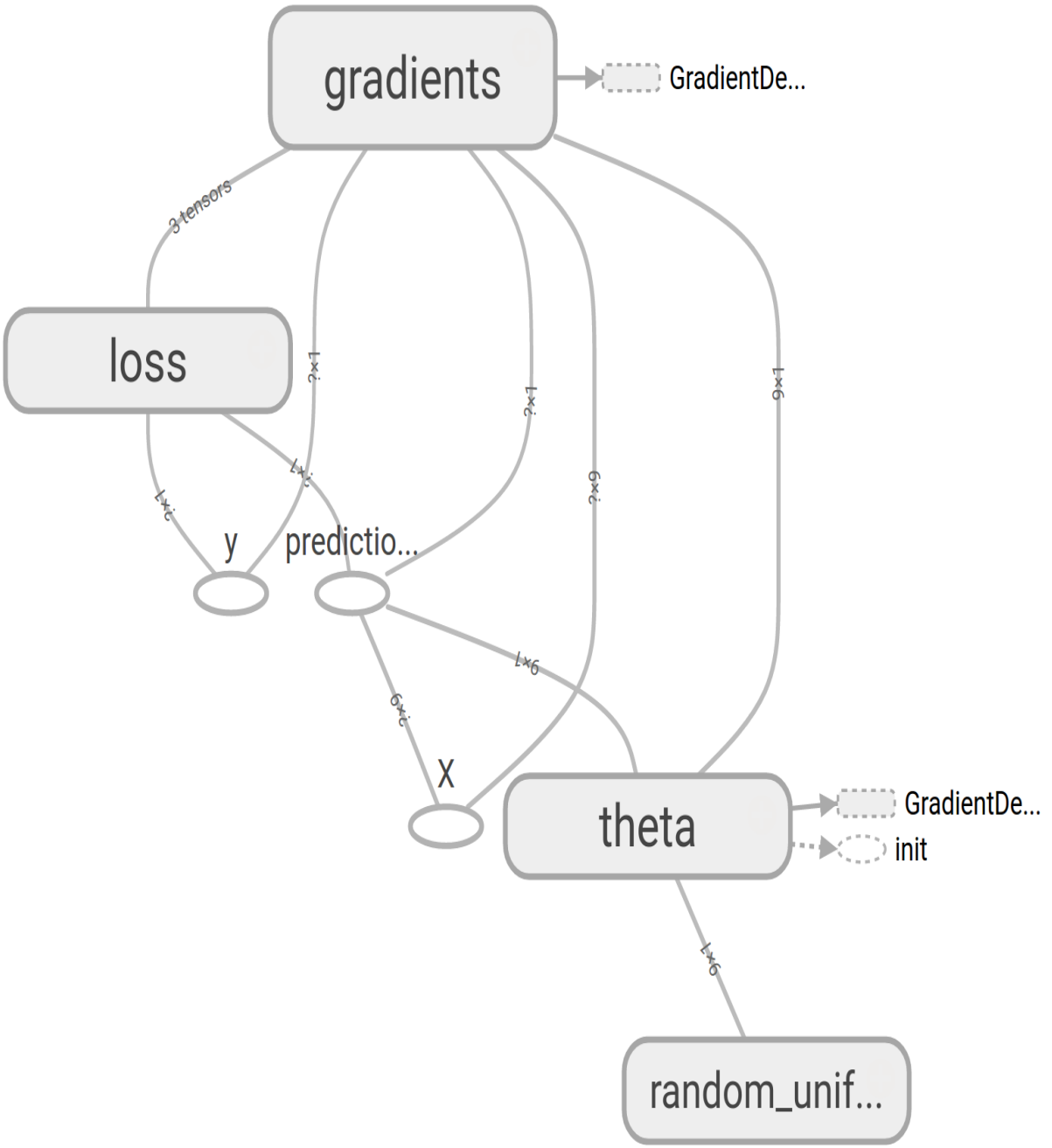
```
def fetch_batch(epoch, batch_index, batch_size):
    rnd.seed(epoch * n_batches + batch_index)
    indices = rnd.randint(m, size=batch_size)
    X_batch = scaled_housing_data_plus_bias[indices]
    y_batch = housing.target.reshape(-1, 1)[indices]
    return X_batch, y_batch

n_epochs = 1000
learning_rate = 0.01

X = tf.placeholder(tf.float32, shape=(None, n + 1), name="X")
y = tf.placeholder(tf.float32, shape=(None, 1), name="y")
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42), name="theta")
y_pred = tf.matmul(X, theta, name="predictions")
error = y_pred - y
mse = tf.reduce_mean(tf.square(error), name="mse")
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(mse)

init = tf.global_variables_initializer()

mse_summary = tf.summary.scalar('MSE', mse)
summary_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())

n_epochs = 10
batch_size = 100
n_batches = int(np.ceil(m / batch_size))

with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        for batch_index in range(n_batches):
            X_batch, y_batch = fetch_batch(epoch, batch_index, batch_size)
            if batch_index % 10 == 0:
                summary_str = mse_summary.eval(feed_dict={X: X_batch, y: y_batch})
                step = epoch * n_batches + batch_index
                summary_writer.add_summary(summary_str, step)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})

    best_theta = theta.eval()

summary_writer.flush()
summary_writer.close()
print("Best theta:")
print(best_theta)
```





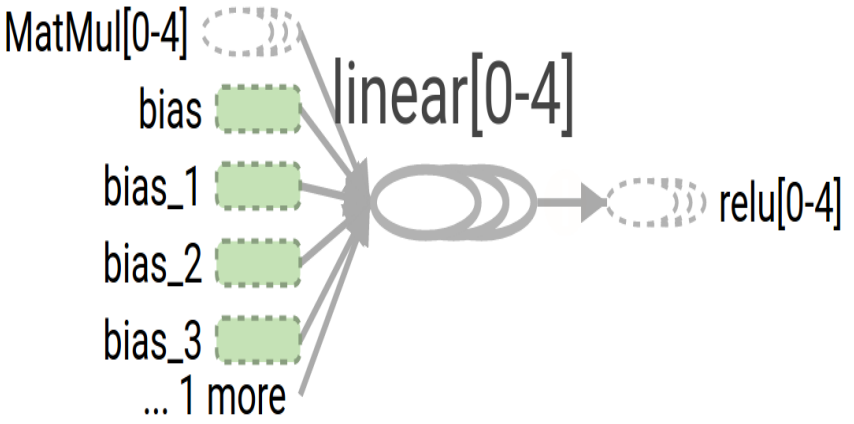
TensorFlow:Tensor Visualization:Named scope

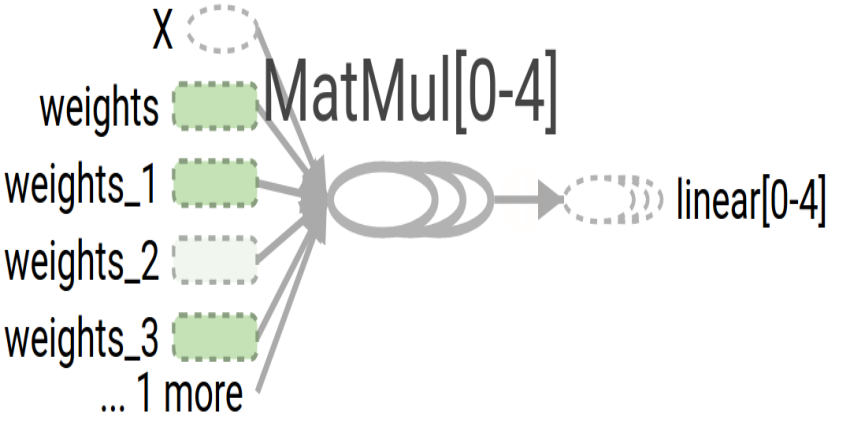
```
n_epochs = 1000
learning_rate = 0.01

X = tf.placeholder(tf.float32, shape=(None, n + 1), name="X")
y = tf.placeholder(tf.float32, shape=(None, 1), name="y")
theta = tf.Variable(tf.random_uniform([n + 1, 1], -1.0, 1.0, seed=42), name="theta")
y_pred = tf.matmul(X, theta, name="predictions")
with tf.name_scope('loss') as scope:
    error = y_pred - y
    mse = tf.reduce_mean(tf.square(error), name="mse")
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(mse)

init = tf.global_variables_initializer()

mse_summary = tf.summary.scalar('MSE', mse)
summary_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())

n_epochs = 10
batch_size = 100
n_batches = int(np.ceil(m / batch_size))

with tf.Session() as sess:
    sess.run(init)

    for epoch in range(n_epochs):
        for batch_index in range(n_batches):
            X_batch, y_batch = fetch_batch(epoch, batch_index, batch_size)
            if batch_index % 10 == 0:
                summary_str = mse_summary.eval(feed_dict={X: X_batch, y: y_batch})
                step = epoch * n_batches + batch_index
                summary_writer.add_summary(summary_str, step)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})

    best_theta = theta.eval()

summary_writer.flush()
summary_writer.close()
print("Best theta:")
print(best_theta)
```



TensorFlow:Tensor Visualization:Modularity

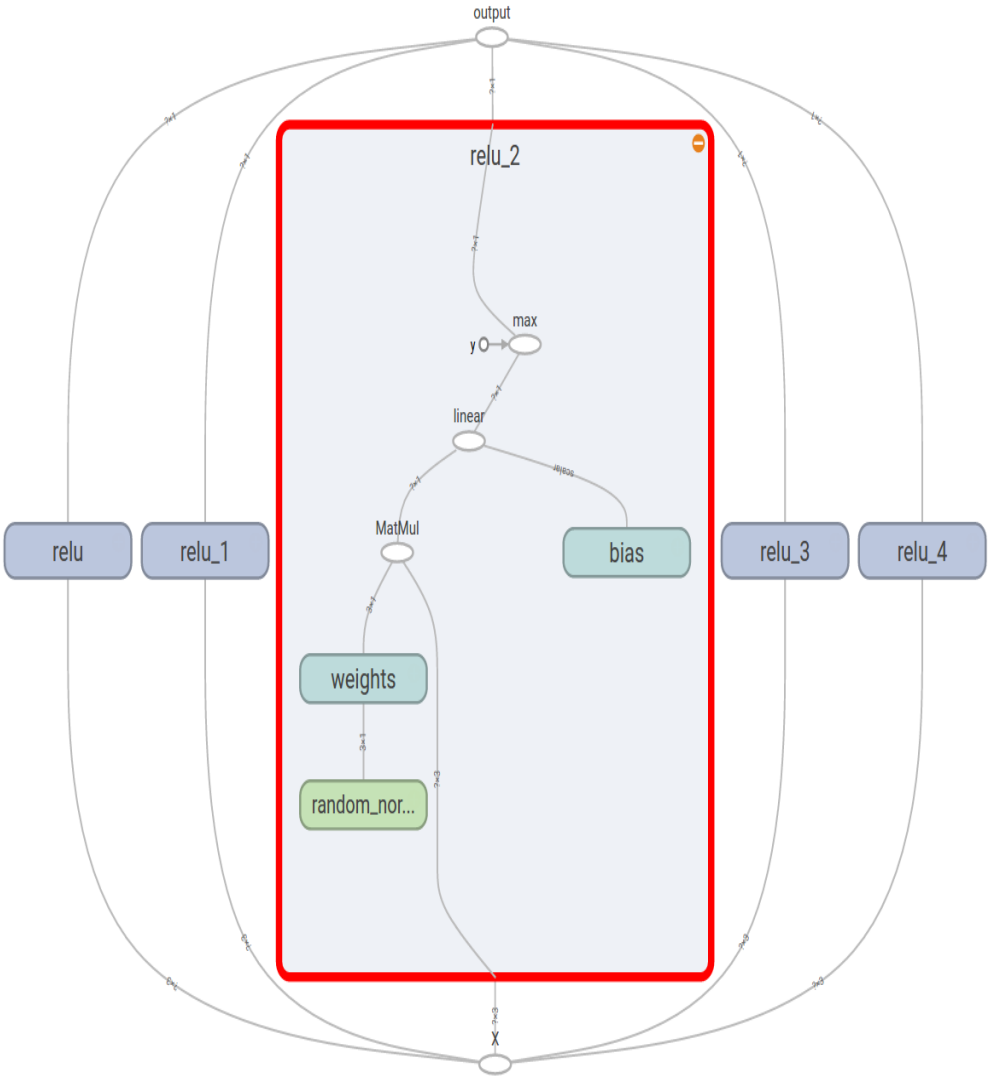
```
tf.reset_default_graph()

# Oops, cut&paste error! Did you spot it?
"""n_features = 3
X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")

w1 = tf.Variable(tf.random_normal((n_features, 1)), name="weights1")
w2 = tf.Variable(tf.random_normal((n_features, 1)), name="weights2")
b1 = tf.Variable(0.0, name="bias1")
b2 = tf.Variable(0.0, name="bias2")

linear1 = tf.add(tf.matmul(X, w1), b1, name="linear1")
linear2 = tf.add(tf.matmul(X, w2), b2, name="linear2")

relu1 = tf.maximum(linear1, 0, name="relu1")
relu2 = tf.maximum(linear1, 0, name="relu2")  # Oops, cut&paste error! Did you spot it?

output = tf.add_n([relu1, relu2], name="output")
print("output:",output)"""

def relu(X):
    w_shape = int(X.get_shape()[1]), 1
    w = tf.Variable(tf.random_normal(w_shape), name="weights")
    b = tf.Variable(0.0, name="bias")
    linear = tf.add(tf.matmul(X, w), b, name="linear")
    return tf.maximum(linear, 0, name="relu")

n_features = 3
X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
relus = [relu(X) for i in range(5)]
output = tf.add_n(relus, name="output")
summary_writer = tf.summary.FileWriter(TMP+"/logs/relu1", tf.get_default_graph())
summary_writer.flush()
summary_writer.close()
```





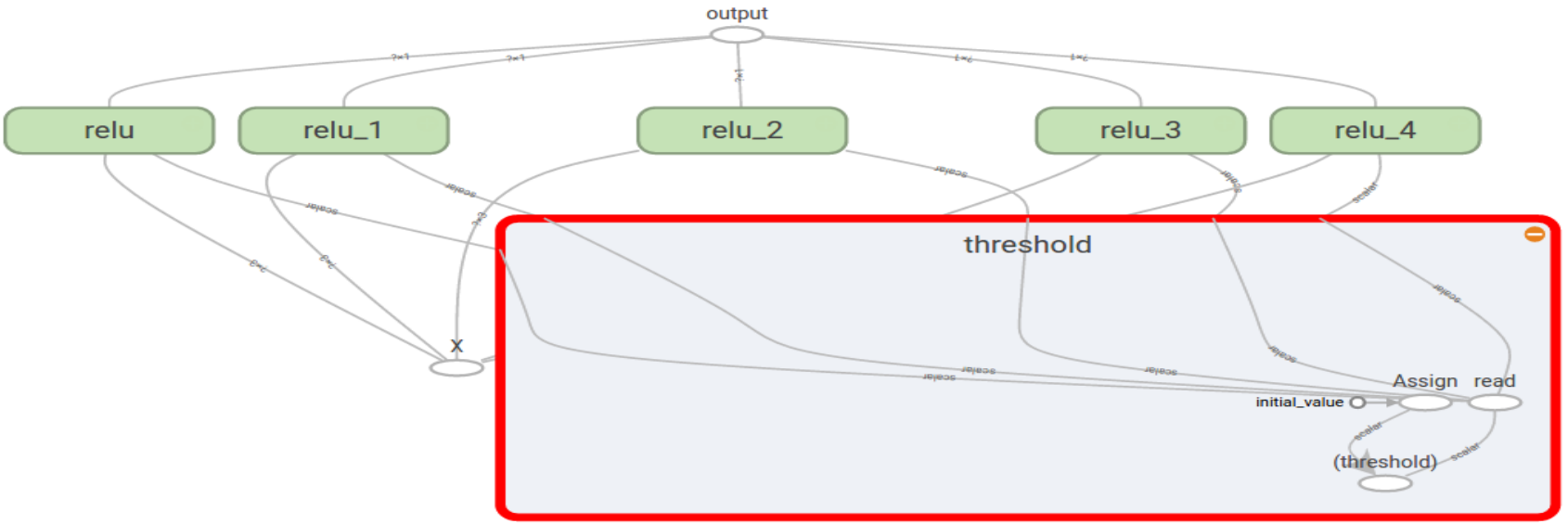
TensorFlow:Tensor Visualization:Modularity with named scope

```
def relu(X):
    with tf.name_scope("relu"):
        w_shape = int(X.get_shape()[1]), 1
        w = tf.Variable(tf.random_normal(w_shape), name="weights")
        b = tf.Variable(0.0, name="bias")
        linear = tf.add(tf.matmul(X, w), b, name="linear")
        return tf.maximum(linear, 0, name="max")

n_features = 3
X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
relus = [relu(X) for i in range(5)]
output = tf.add_n(relus, name="output")

summary_writer = tf.summary.FileWriter(TMP+"/logs/relu2", tf.get_default_graph())
summary_writer.flush()
summary_writer.close()
```



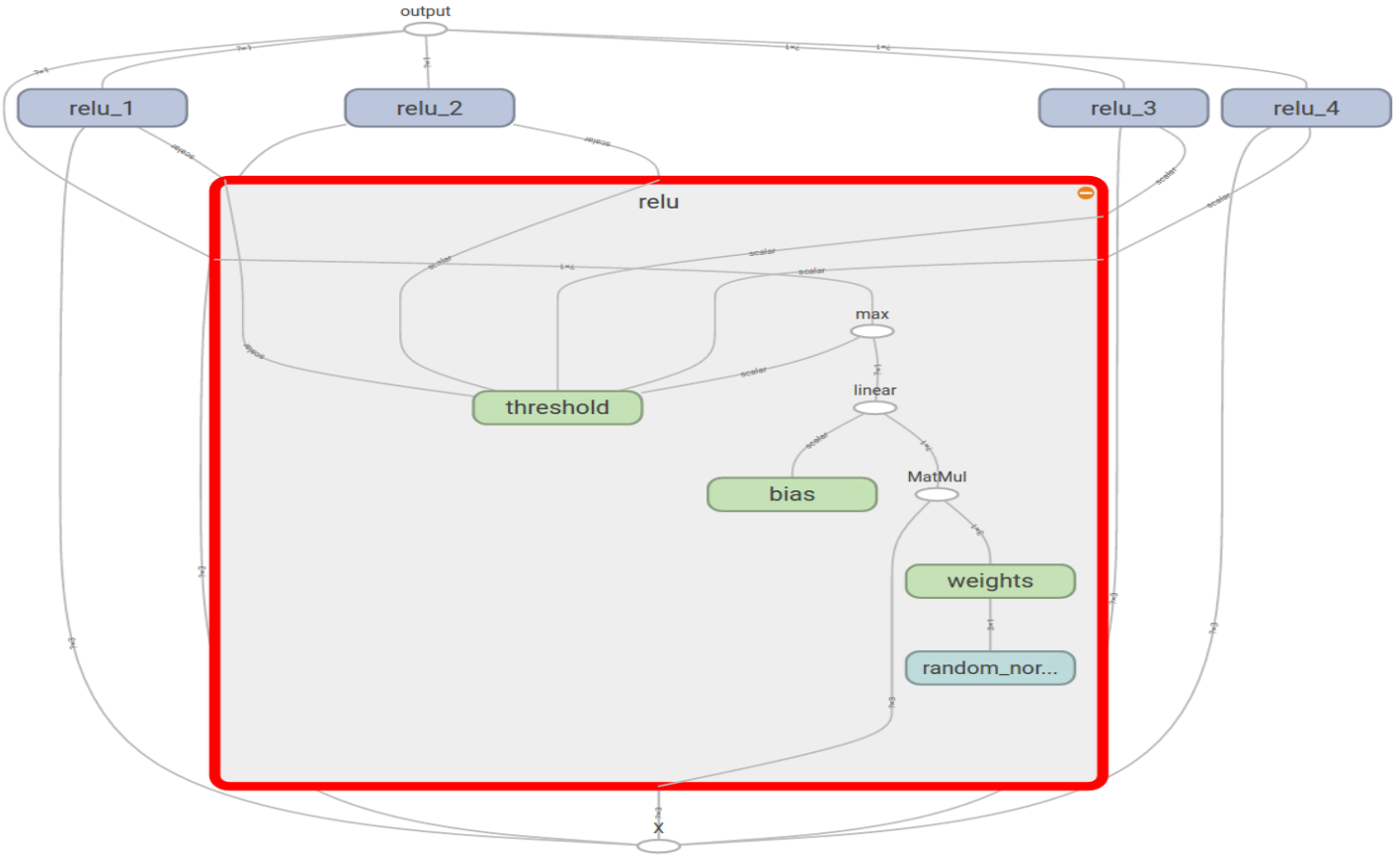
TensorFlow:Sharing variable with graph

```
tf.reset_default_graph()

n_features = 3
def relu(X, threshold):
    with tf.name_scope("relu"):
        w_shape = int(X.get_shape()[1]), 1
        w = tf.Variable(tf.random_normal(w_shape), name="weights")
        b = tf.Variable(0.0, name="bias")
        linear = tf.add(tf.matmul(X, w), b, name="linear")
        return tf.maximum(linear, threshold, name="max")

#However, if there are many shared parameters such as this one, it will be
#painful to have to pass them around as parameters all the time.
#other options is to create python dict or a ReLU class
threshold = tf.Variable(0.0, name="threshold")
X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
relus = [relu(X, threshold) for i in range(5)]
output = tf.add_n(relus, name="output")

summary_writer = tf.summary.FileWriter(TMP+"/logs/relu3", tf.get_default_graph())
summary_writer.flush()
summary_writer.close()
```



TensorFlow:Sharing variable with graph-2

```
tf.reset_default_graph()
n_features = 3
def relu(X):
    with tf.name_scope("relu"):
        #another option is to set the shared variable
        #as an attribute of the relu() function upon the first call:
        if not hasattr(relu, "threshold"):
            relu.threshold = tf.Variable(0.0, name="threshold")
        w_shape = int(X.get_shape()[1]), 1
        w = tf.Variable(tf.random_normal(w_shape), name="weights")
        b = tf.Variable(0.0, name="bias")
        linear = tf.add(tf.matmul(X, w), b, name="linear")
        return tf.maximum(linear, relu.threshold, name="max")

X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
relus = [relu(X) for i in range(5)]
output = tf.add_n(relus, name="output")

summary_writer = tf.summary.FileWriter(TMP+"/logs/relu4", tf.get_default_graph())
summary_writer.flush()
summary_writer.close()
```



TensorFlow:Sharing variable with graph-3

```
tf.reset_default_graph()
n_features = 3


def relu(X):
    with tf.variable_scope("relu", reuse=True):
        #TensorFlow offers another option, which may lead to slightly cleaner and more modular
        #code than the previous solutions.The
        #idea is to use the get_variable() function to create the shared variable if it does not
        #exist yet, or reuse it if it already exists. The desired behavior (creating or reusing) is
        #controlled by an attribute of the current variable_scope(). For example, the following
        #code will create a variable named "relu/threshold" (as a scalar, since shape=(),
        #and using 0.0 as the initial value):
        #Note that if the variable has already been created by an earlier call to get_vari
        #able(), this code will raise an exception. This behavior prevents reusing variables by
        #mistake.
        #If you want to reuse a variable "with tf.variable_scope("relu", reuse=True):"
        threshold = tf.get_variable("threshold", shape=(), initializer=tf.constant_initializer(0.0))
        w_shape = int(X.get_shape()[1]), 1
        w = tf.Variable(tf.random_normal(w_shape), name="weights")
        b = tf.Variable(0.0, name="bias")
        linear = tf.add(tf.matmul(X, w), b, name="linear")
        return tf.maximum(linear, threshold, name="max")

X = tf.placeholder(tf.float32, shape=(None, n_features), name="X")
with tf.variable_scope("relu"):
    threshold = tf.get_variable("threshold", shape=(), initializer=tf.constant_initializer(0.0))
relus = [relu(X) for i in range(5)]
output = tf.add_n(relus, name="output")

summary_writer = tf.summary.FileWriter(TMP+"/logs/relu5", tf.get_default_graph())
summary_writer.flush()
summary_writer.close()
```