Νευρο-Ασαφής Υπολογιστική
Neuro-Fuzzy Computing

Διδάσκων –

Δημήτριος Κατσαρός

@ Τμ. HMMY

Πανεπιστήμιο Θεσσαλίας

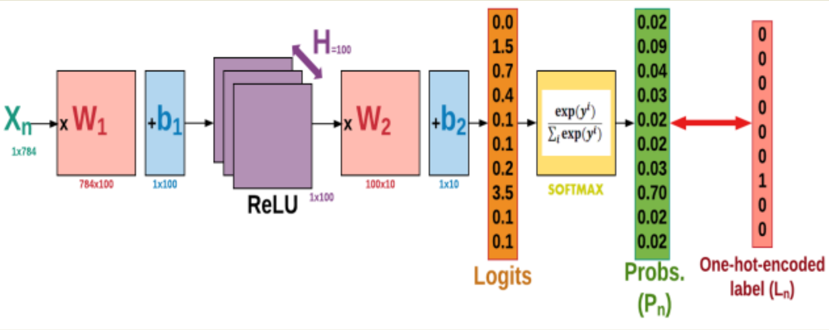
Διάλεξη 2η

Introduction to TensorFlow

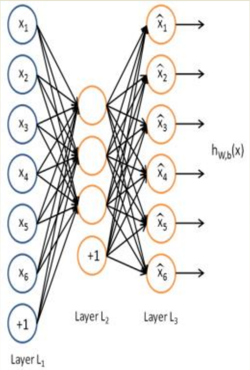
What do you learn at this lecture?

- TensorFlow Basics
  - **Datatypes**: Efficiently use your memory
  - **Graph & Session:** Save computation time by running needed operations
  - **TensorBoard:** Flashlight to your Neural Network **BLACK BOX**

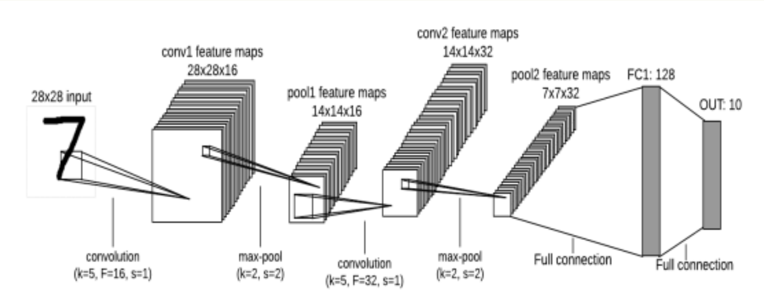
- **Neural Network**



- **AutoEncoder**



- **Convolutional Neural Network**



28x28 input
7
conv1 feature maps 28x28x16
pool1 feature maps 14x14x16
conv2 feature maps 14x14x32
pool2 feature maps 7x7x32
FC1: 128
OUT: 10
convolution (k=5, F=16, s=1)
max-pool (k=2, s=2)
convolution (k=5, F=32, s=1)
max-pool (k=2, s=2)
Full connection
Full connection

Outline

- About TensorFlow
  - What is TensorFlow?
  - Why TensorFlow?
- TensorFlow Basics
  - Introduction
  - Graph & Session
  - Datatypes
- Logistic Regression (linear classifier)

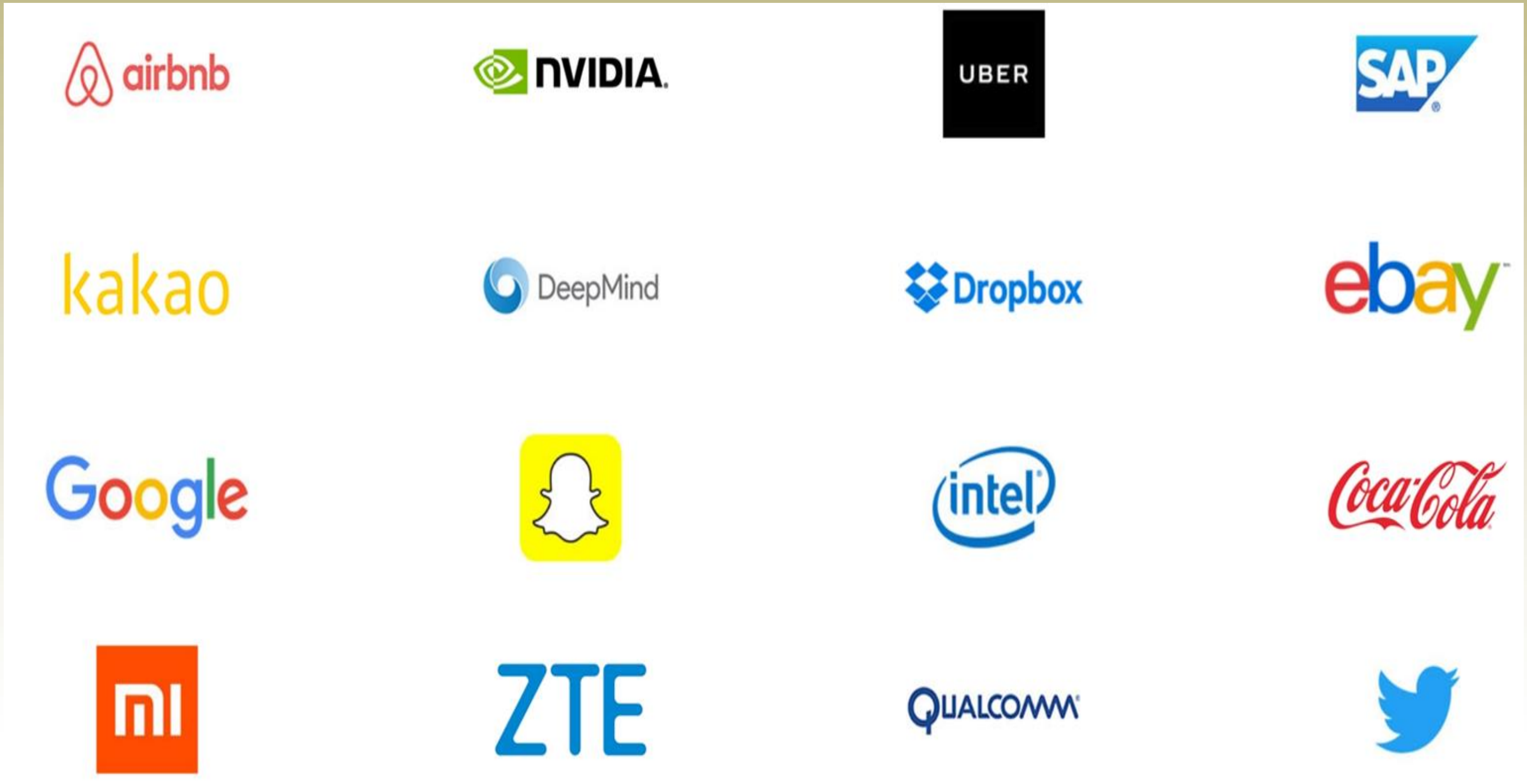
What is TensorFlow?

"TensorFlow™ is an open source software library for numerical computation using data flow graphs."

"… software library for Machine Intelligence"

- Created by Google
- API available for multiple languages (Python, C++, Java, Go, etc.)
- Suitable for both **production** & **research**

Companies using TensorFlow



Trends on Deep Learning libraries

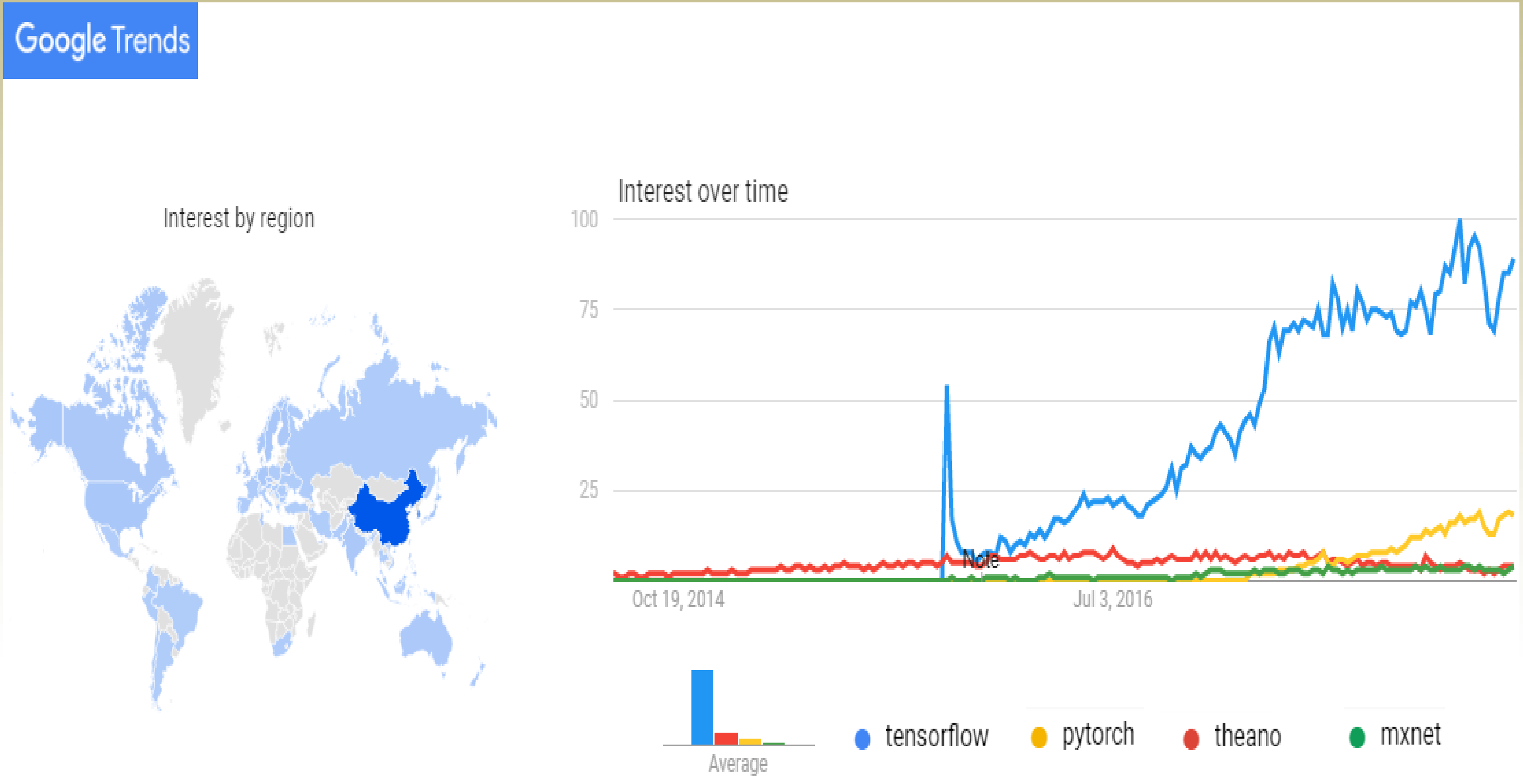


Image style transfer with TensorFlow

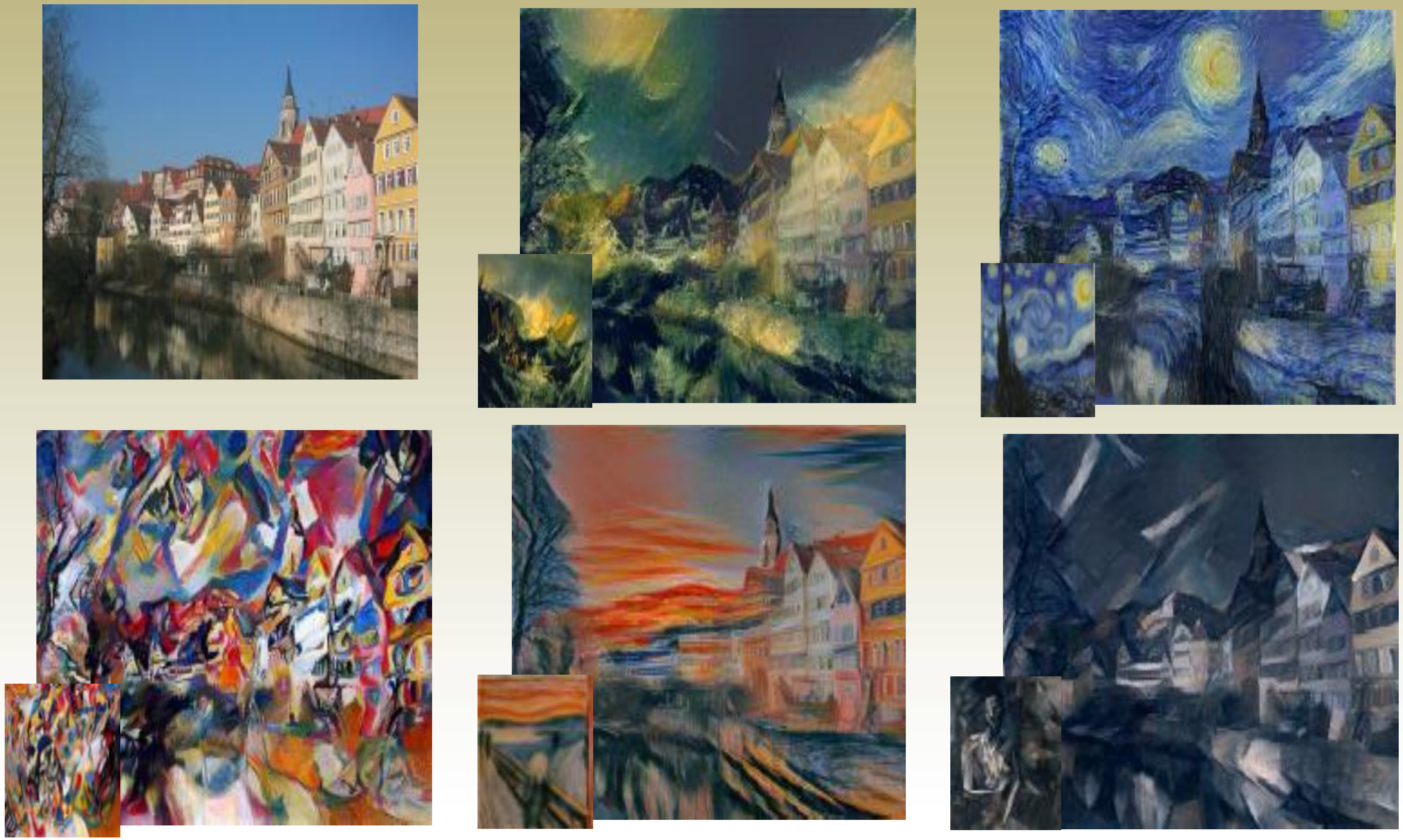


Image Style Transfer Using Convolutional Neural Networks (Gatys et. al. 2016)

Why TensorFlow?

- Developed and maintained by Google
- Very large and active community + Nice documentation
- Python API
- Multi-GPU support
- TensorBoard (A very powerful visualization tool)
- Faster model compilation than Theano-based options
- High level APIs build on top of TensorFlow (Keras, TFlearn, …)

How to set it up?

- **Python** – Programming language
- **Anaconda** – Package manager (Optional; instead of installing Python directly)
  - What is Anaconda?
- **TensorFlow**
- **IDE** – Editor (preferably **PyCharm**)

http://www.easy-tensorflow.com/install

Introduction to TensorFlow

- What is a Tensor?
  - Multi-dimensional array
    - 0-d tensor: scalar
    - 1-d tensor: vector
    - 2-d tensor: matrix

Input tensor for images:
[*batch_size*, *image_height*, *image_width*, *channels*]

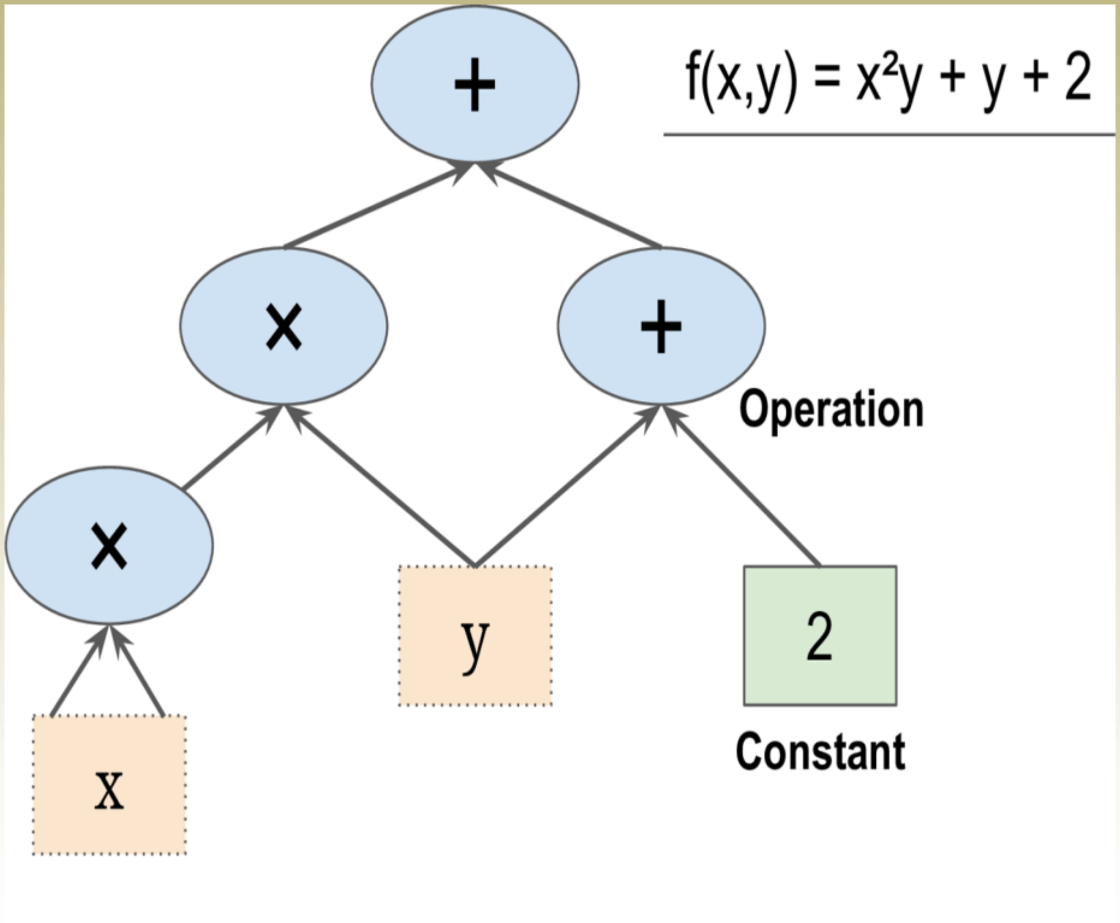
- Importing the library
  - `import tensorflow as tf`

- "Computational Graph" approach
  1. Build the GRAPH which represents the data flow of the computation
  2. Run the SESSION which executes the operations on the graph

Graph and Session

Graph

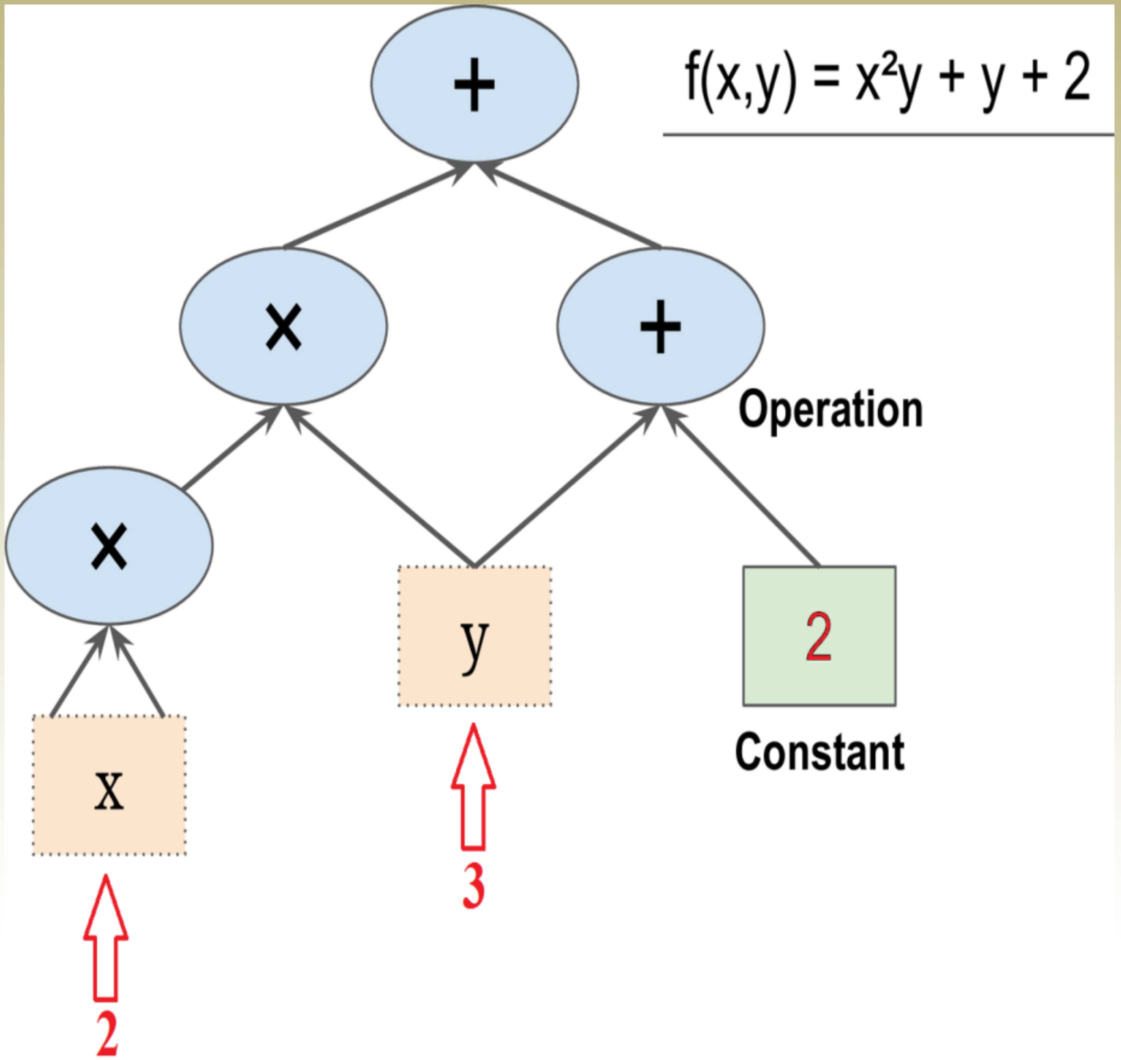
- Nodes = operations



Graph and Session

Graph

- Nodes = operations
- Edges= tensors



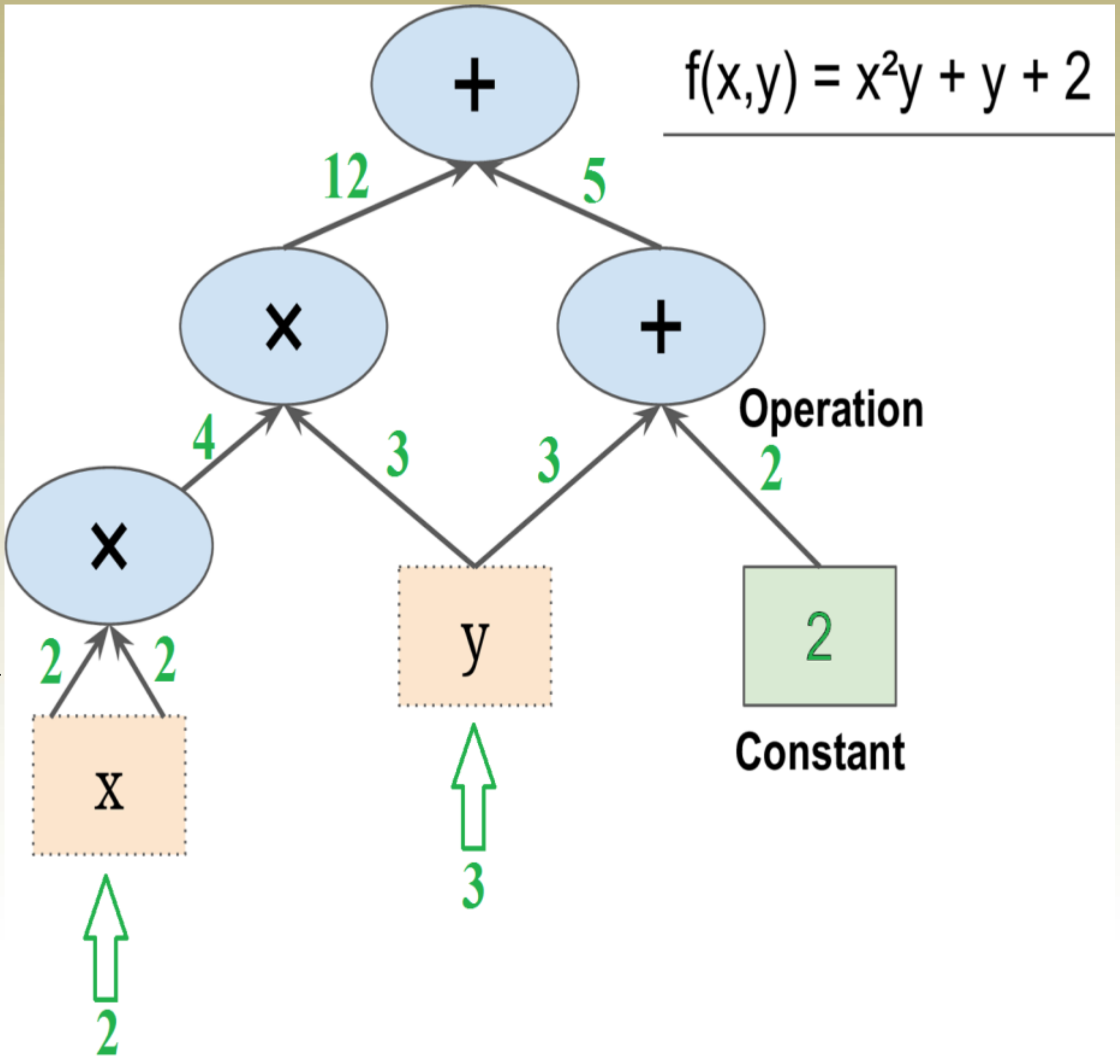
Graph and Session

Graph

- Nodes = operations
- Edges= tensors

Session

- Tensor = data
- Tensor + flow =
  data + flow

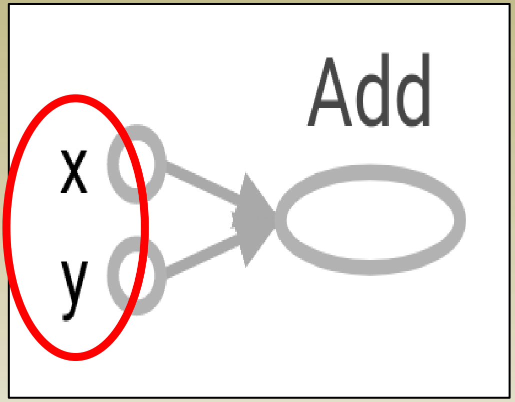


Graph and Session

Example 1:

```
import tensorflow as tf
c = tf.add(2, 3, name='Add')
print(c)
```

Graph



TensorFlow names the node if you don’t !

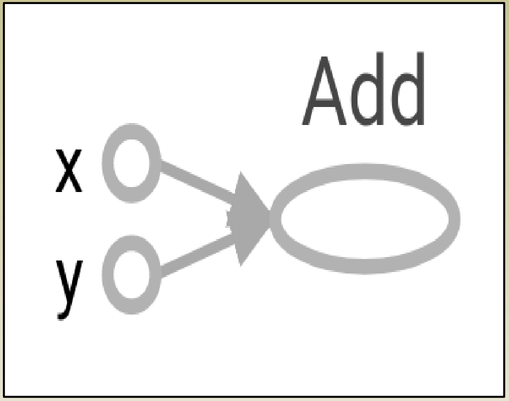
LAZY PROGRAMMING: Call-by-need

Graph and Session

Example 1:

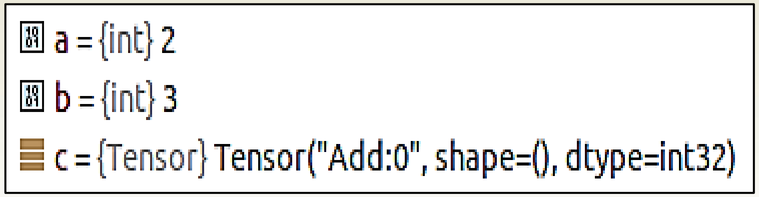
```
import tensorflow as tf
a = 2
b = 3
c = tf.add(a, b, name='Add')
print(c)
```

Graph



?Tensor("Add:0", shape=(), dtype=int32)

Variables



```
a = {int} 2
b = {int} 3
c = {Tensor} Tensor("Add:0", shape=(), dtype=int32)
```

"Computational Graph" approach

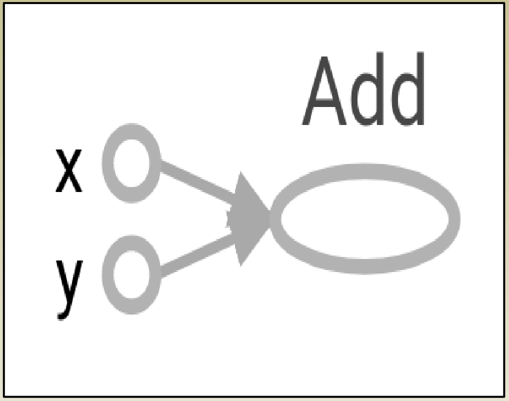
1. Build the GRAPH which represents the data flow of the computation
2. Run the SESSION which executes the operations on the graph

Graph and Session

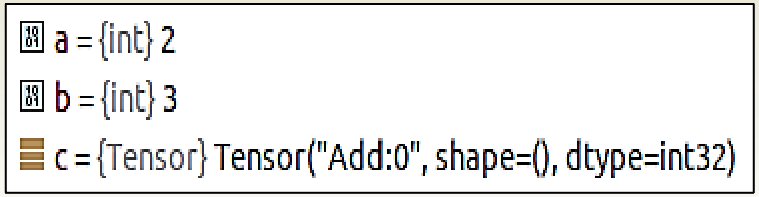
Example 1:

```
import tensorflow as tf
a = 2
b = 3
c = tf.add(a, b, name='Add')
print(c)

sess = tf.Session()
print(sess.run(c))
sess.close()
5
```

Graph



Variables



```
a = {int} 2
b = {int} 3
c = {Tensor} Tensor("Add:0", shape=(), dtype=int32)
```

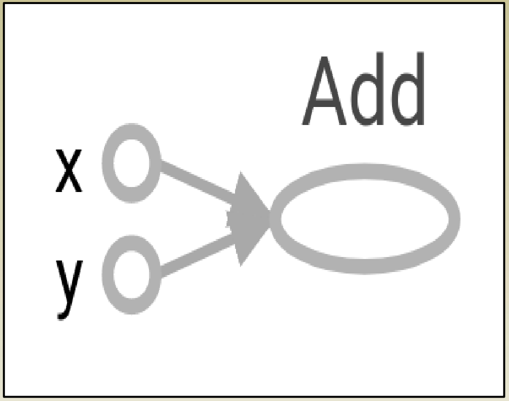
Graph and Session

Example 1:

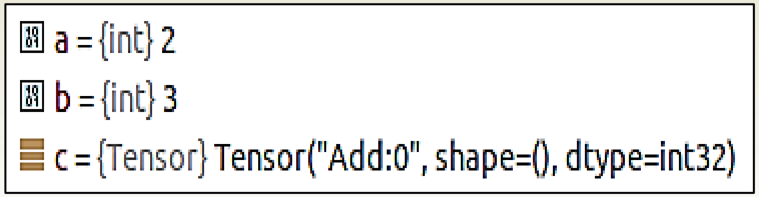
```
import tensorflow as tf
a = 2
b = 3
c = tf.add(a, b, name='Add')
print(c)
~~sess = tf.Session()~~
with tf.Session() as sess:
    print(sess.run(c))
~~sess.close()~~
```

```
5
```

Graph



Variables



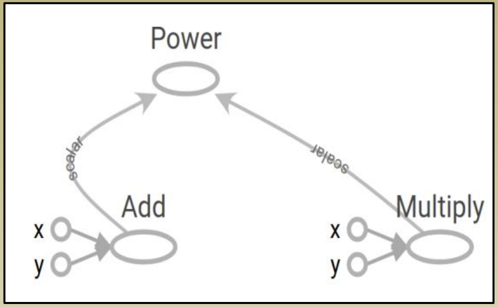
```
a = {int} 2
b = {int} 3
c = {Tensor} Tensor("Add:0", shape=(), dtype=int32)
```

Graph and Session

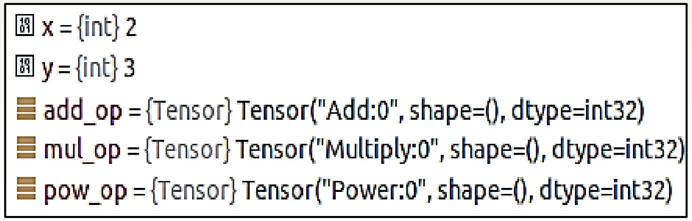
Example 2:

```
import tensorflow as tf
x = 2
y = 3
add_op = tf.add(x, y, name='Add')
mul_op = tf.multiply(x, y, name='Multiply')
pow_op = tf.pow(add_op, mul_op, name='Power')

with tf.Session() as sess:
    pow_out = sess.run(pow_op)
```

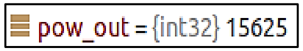
Graph



Variables



```
x = {int} 2
y = {int} 3
add_op = {Tensor} Tensor("Add:0", shape=(), dtype=int32)
mul_op = {Tensor} Tensor("Multiply:0", shape=(), dtype=int32)
pow_op = {Tensor} Tensor("Power:0", shape=(), dtype=int32)
```



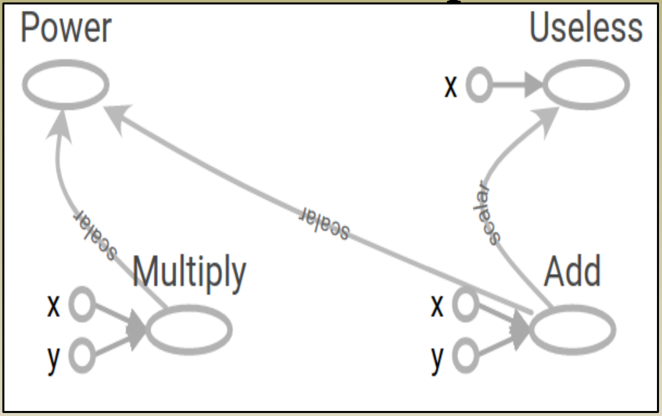
```
pow_out = {int32} 15625
```

Graph and Session

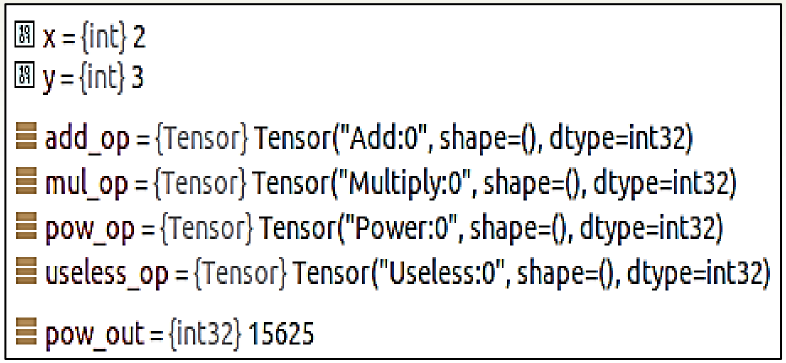
Example 3:

```
import tensorflow as tf
x = 2
y = 3
add_op = tf.add(x, y, name='Add')
mul_op = tf.multiply(x, y, name='Multiply')
pow_op = tf.pow(add_op, mul_op, name='Power')
useless_op = tf.multiply(x, add_op,
name='Useless')

with tf.Session() as sess:
    pow_out = sess.run(pow_op)
```

Graph



Variables



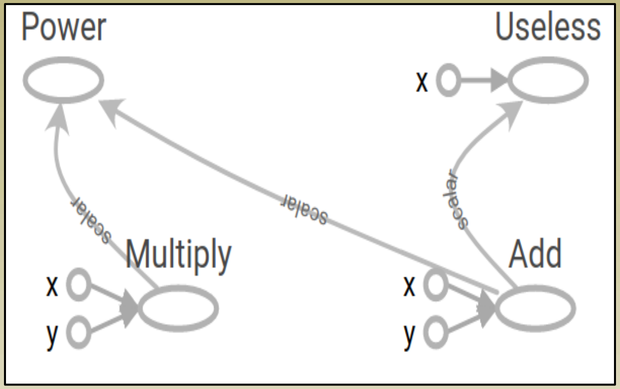
```
x = {int} 2
y = {int} 3
add_op = {Tensor} Tensor("Add:0", shape=(), dtype=int32)
mul_op = {Tensor} Tensor("Multiply:0", shape=(), dtype=int32)
pow_op = {Tensor} Tensor("Power:0", shape=(), dtype=int32)
useless_op = {Tensor} Tensor("Useless:0", shape=(), dtype=int32)
pow_out = {int32} 15625
```

Graph and Session

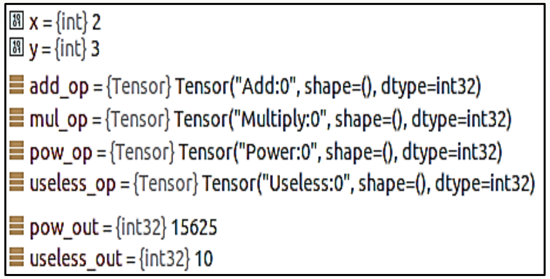
Example 3:

```
import tensorflow as tf
x = 2
y = 3
add_op = tf.add(x, y, name='Add')
mul_op = tf.multiply(x, y, name='Multiply')
pow_op = tf.pow(add_op, mul_op, name='Power')
useless_op = tf.multiply(x, add_op, name='Useless')

with tf.Session() as sess:
    [pow_out,useless_out= sess.run([pow_op, useless_op]) 1
```

Graph



Variables



```
x = {int} 2
y = {int} 3
add_op = {Tensor} Tensor("Add:0", shape=(), dtype=int32)
mul_op = {Tensor} Tensor("Multiply:0", shape=(), dtype=int32)
pow_op = {Tensor} Tensor("Power:0", shape=(), dtype=int32)
useless_op = {Tensor} Tensor("Useless:0", shape=(), dtype=int32)
pow_out = {int32} 15625
useless_out = {int32} 10
```

Data types

1. **Constants** are used to create constant values

```
tf.constant( value,
             dtype=None,
             shape=None,
             name='Const',
             verify_shape=False
           )
```

Example

```
s = tf.constant(2, name='scalar')
m = tf.constant([[1, 2], [3, 4]], name='matrix')
```

Data types

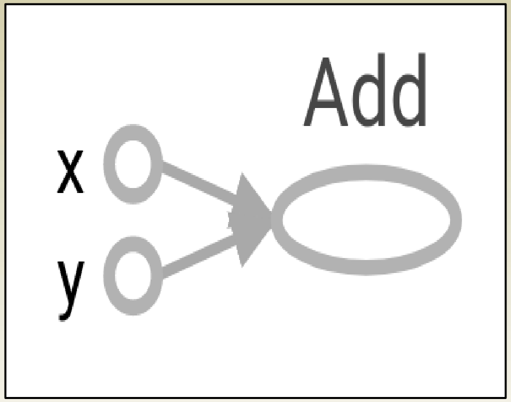
1. **Constants** are used to create constant values

Before:

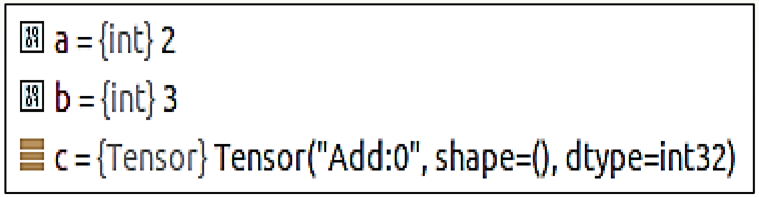
```
import tensorflow as tf
a = 2
b = 3
c = tf.add(a, b, name='Add')

with tf.Session() as sess:
    print(sess.run(c))
```

```
5
```

Graph



Variables



```
a = {int} 2
b = {int} 3
c = {Tensor} Tensor("Add:0", shape=(), dtype=int32)
```

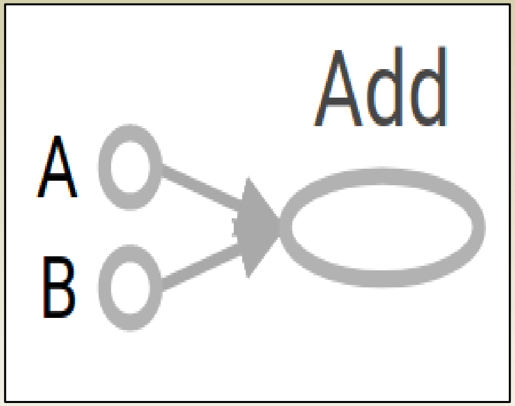
Data types

1. **Constants** are used to create constant values

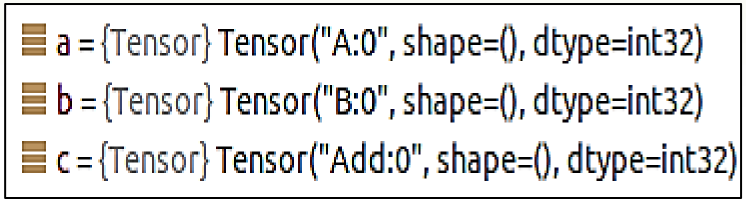
Now:

```
import tensorflow as tf
a = tf.constant(2, name='A')
b = tf.constant(3, name='B')
c = tf.add(a, b, name='Add')

with tf.Session() as sess:
    print(sess.run(c))
5
```

Graph



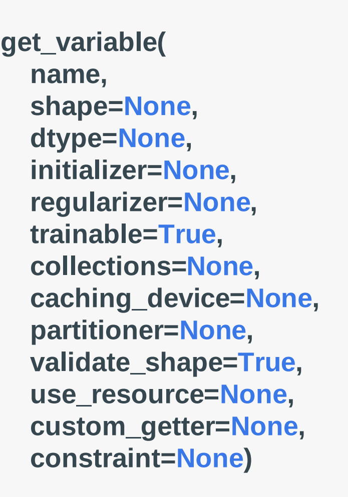
Variables



```
a = {Tensor} Tensor("A:0", shape=(), dtype=int32)
b = {Tensor} Tensor("B:0", shape=(), dtype=int32)
c = {Tensor} Tensor("Add:0", shape=(), dtype=int32)
```

Data types

2. **Variables** are stateful nodes (=ops) which output their current value
   1. They can be saved and restored
   2. Gradient updates will apply to all variables in the graph



```
get_variable(
   name,
   shape=None,
   dtype=None,
   initializer=None,
   regularizer=None,
   trainable=True,
   collections=None,
   caching_device=None,
   partitioner=None,
   validate_shape=True,
   use_resource=None,
   custom_getter=None,
   constraint=None)
```

⇒ Network Parameters (weights and biases)

Example

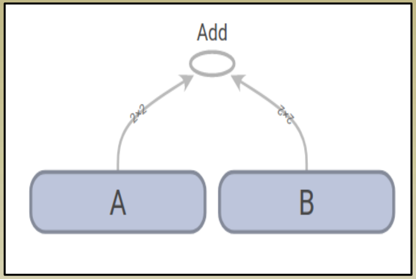
```
s1 = tf.get_variable(name='scalar1', initializer=2)
s2 = tf.get_variable(name='scalar2', initializer=tf.constant(2))
m = tf.get_variable('matrix', initializer=tf.constant([[0, 1], [2, 3]]))
M = tf.get_variable('big_matrix', shape=(784, 10), initializer=tf.zeros_initializer())
W = tf.get_variable('weight', shape=(784, 10), initializer=tf.truncated_normal_initializer(mean=0.0,

stddev=0.01))
```

Data types

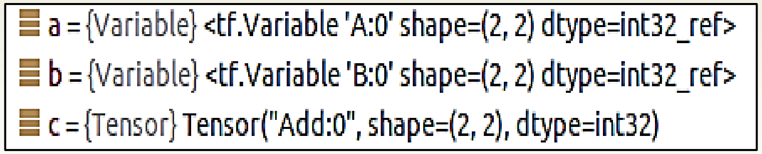
2. Variables

```
import tensorflow as tf

# create graph
a = tf.get_variable(name="A",
initializer=tf.constant([[0, 1], [2, 3]]))
b = tf.get_variable(name="B",
initializer=tf.constant([[4, 5], [6, 7]]))
c = tf.add(a, b, name="Add")
```

Graph



Variables



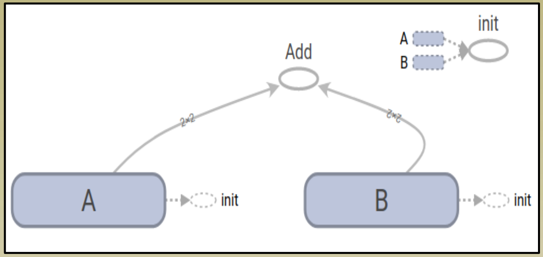
```
a = {Variable} <tf.Variable 'A:0' shape=(2, 2) dtype=int32_ref>
b = {Variable} <tf.Variable 'B:0' shape=(2, 2) dtype=int32_ref>
c = {Tensor} Tensor("Add:0", shape=(2, 2), dtype=int32)
```

```
# launch the graph in a session
with tf.Session() as sess:
    # now we can run the desired operation
    print(sess.run(c))
```

? FailedPreconditionError: Attempting to use uninitialized value

Data types

Graph

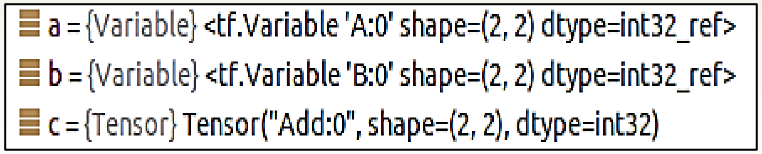


2. Variables

```
import tensorflow as tf

# create graph
a = tf.get_variable(name="A", initializer=tf.constant([[0,
1], [2, 3]]))
b = tf.get_variable(name="B", initializer=tf.constant([[4,
5], [6, 7]]))
c = tf.add(a, b, name="Add")
# Add an Op to initialize variables
init_op = tf.global_variables_initializer()
# launch the graph in a session
with tf.Session() as sess:
    # run the variable initializer
    sess.run(init_op)
    # now we can run the desired operation
    print(sess.run(c))
```

```
[[ 4  6]
[ 8 10]]
```

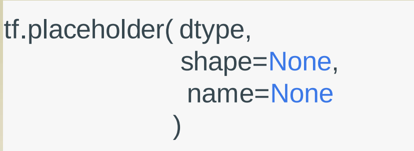
Variables



```
a = {Variable} <tf.Variable 'A:0' shape=(2, 2) dtype=int32_ref>
b = {Variable} <tf.Variable 'B:0' shape=(2, 2) dtype=int32_ref>
c = {Tensor} Tensor("Add:0", shape=(2, 2), dtype=int32)
```

Data types

3. **Placeholder** is a node whose value is fed in at execution time.
    1. Assemble the graph without knowing the values needed for computation
    2. We can later supply the data at the execution time.



```
tf.placeholder( dtype,
                        shape=None,
                         name=None
                   )
```

⇒ **Input data (in classification task: Inputs and labels)**

Example

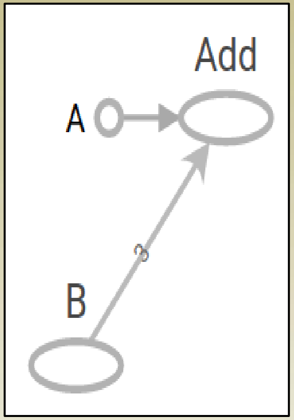
```
a = tf.placeholder(tf.float32, shape=[5])
b = tf.placeholder(dtype=tf.float32, shape=None,
name=None)
X = tf.placeholder(tf.float32, shape=[None, 784],
name='input')
Y = tf.placeholder(tf.float32, shape=[None, 10],
name='label')
```

Data types

3. Placeholder

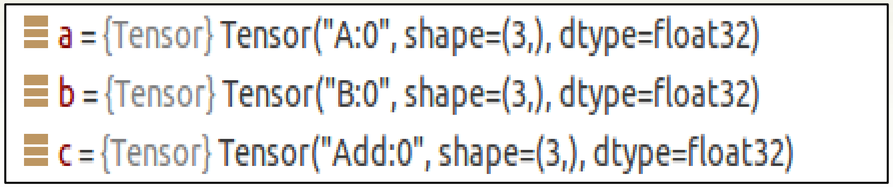
```
import tensorflow as tf
a = tf.constant([5, 5, 5], tf.float32, name='A')
b = tf.placeholder(tf.float32, shape=[3], name='B')
c = tf.add(a, b, name="Add")


with tf.Session() as sess:
    print(sess.run(c))
```

Graph



? You must feed a value for placeholder tensor 'B' with dtype float and shape [3]

Variables



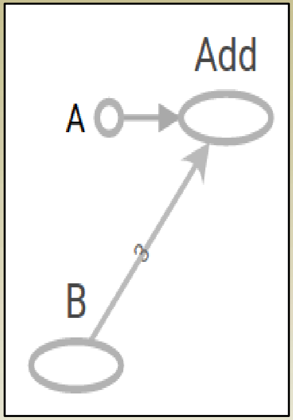
```
a = {Tensor} Tensor("A:0", shape=(3,), dtype=float32)
b = {Tensor} Tensor("B:0", shape=(3,), dtype=float32)
c = {Tensor} Tensor("Add:0", shape=(3,), dtype=float32)
```

Data types

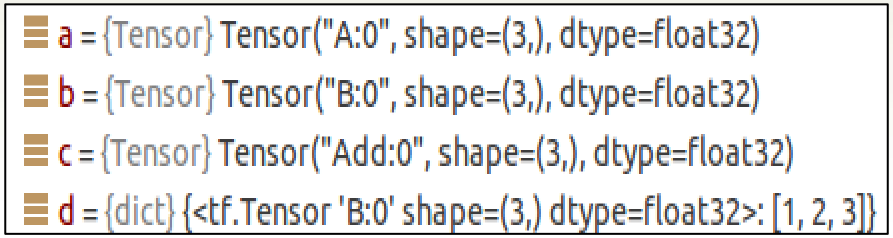
3. Placeholder

```python
import tensorflow as tf
a = tf.constant([5, 5, 5], tf.float32, name='A')
b = tf.placeholder(tf.float32, shape=[3], name='B')
c = tf.add(a, b, name="Add")


with tf.Session() as sess:
    # create a dictionary:
    d = {b: [1, 2, 3]}
    # feed it to the placeholder
    print(sess.run(c, feed_dict=d))
```

Graph



Variables



```
a = {Tensor} Tensor("A:0", shape=(3,), dtype=float32)
b = {Tensor} Tensor("B:0", shape=(3,), dtype=float32)
c = {Tensor} Tensor("Add:0", shape=(3,), dtype=float32)
d = {dict} {<tf.Tensor 'B:0' shape=(3,) dtype=float32>: [1, 2, 3]}
```

```
[6. 7. 8.]
```