Developer(s)	Google Brain Team
Initial release	November 9, 2015;

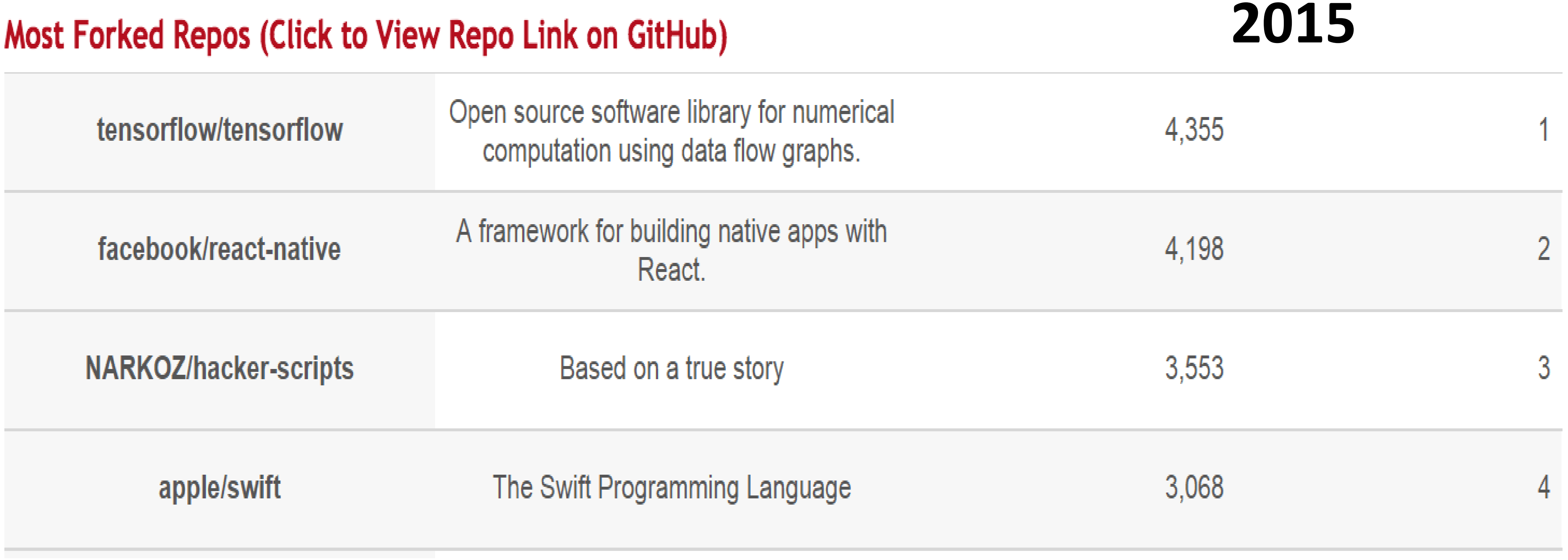
What is TensorFlow? (tensorflow.org)

- Open source software library for numerical computation using data flow graphs
- Originally developed by Google Brain Team to conduct machine learning and deep neural networks research
- General enough to be applicable in a wide variety of other domains as well

- TensorFlow provides an extensive suite of functions and classes that allow users to build various models from scratch.

Launched 2015

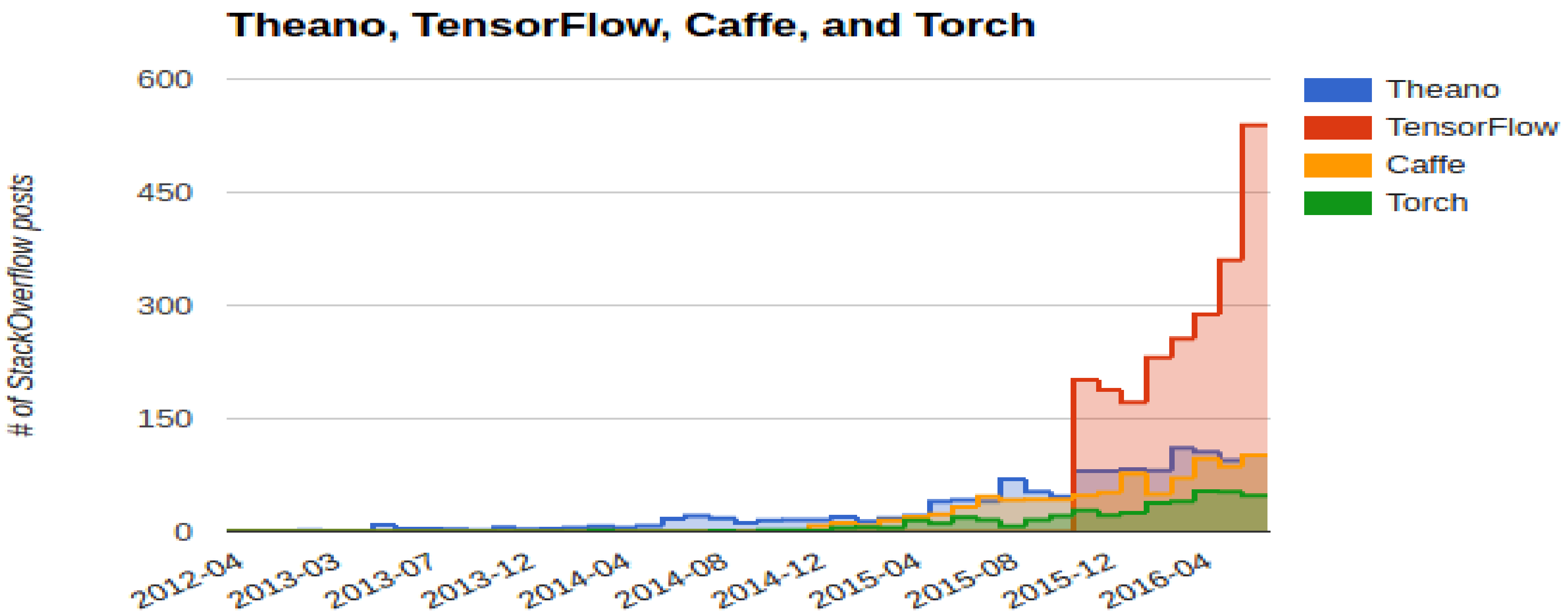
2015



Most Forked Repos (Click to View Repo Link on GitHub)

tensorflow/tensorflow	Open source software library for numerical computation using data flow graphs.	4,355	1
facebook/react-native	A framework for building native apps with React.	4,198	2
NARKOZ/hacker-scripts	Based on a true story	3,553	3
apple/swift	The Swift Programming Language	3,068	4

Other Deep Learning Library



TensorFlow

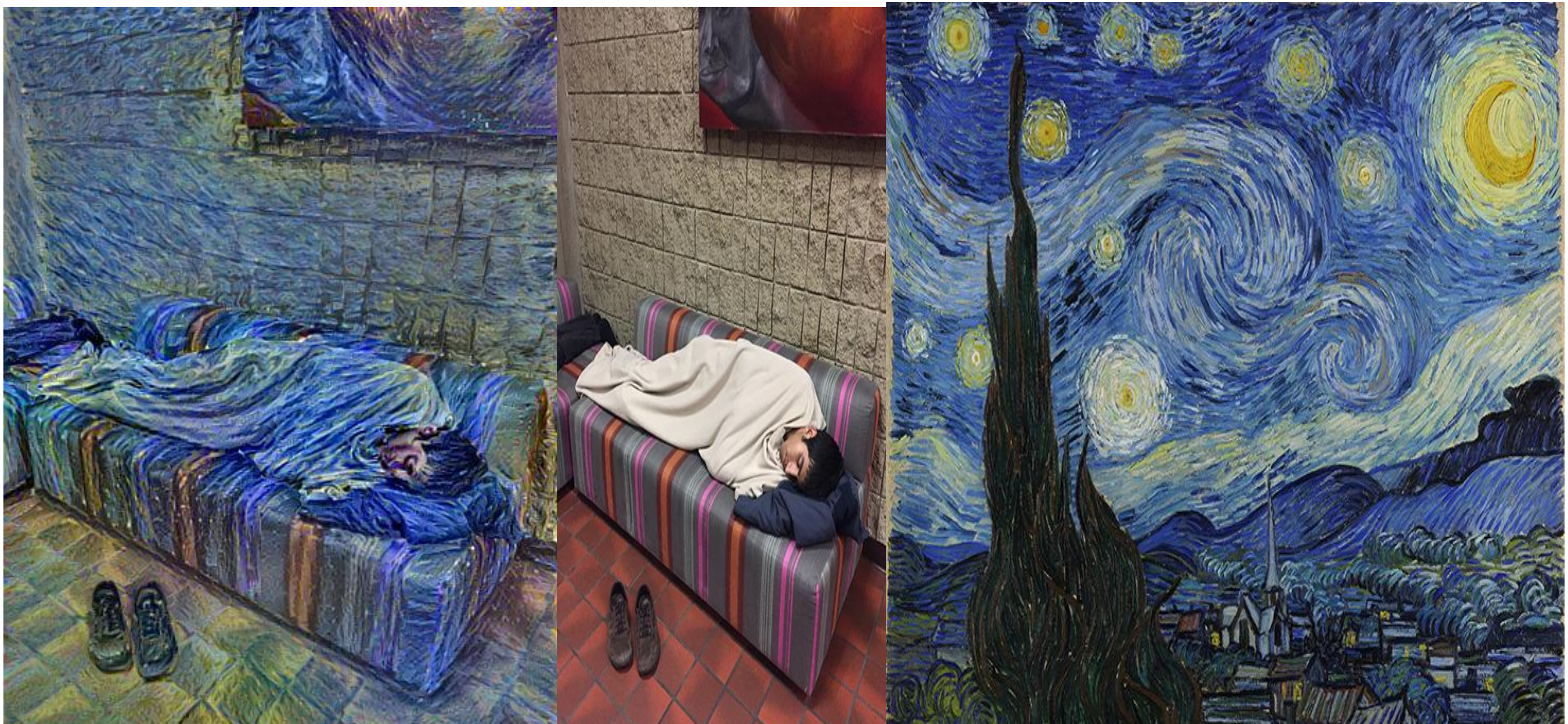
- Python API
- Portability: deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
- Flexibility: from Raspberry Pi, Android, Windows, iOS, Linux to server farms
- Visualization (TensorBoard)
- Checkpoints (for managing experiments)
- Auto-differentiation *autodiff*
- Large community (> 10,000 commits and > 3000 TF-related repos in 1 year)

Companies using TensorFlow

- Google
- OpenAI
- DeepMind
- Snapchat
- Uber
- Airbus
- eBay
- Dropbox
- startups

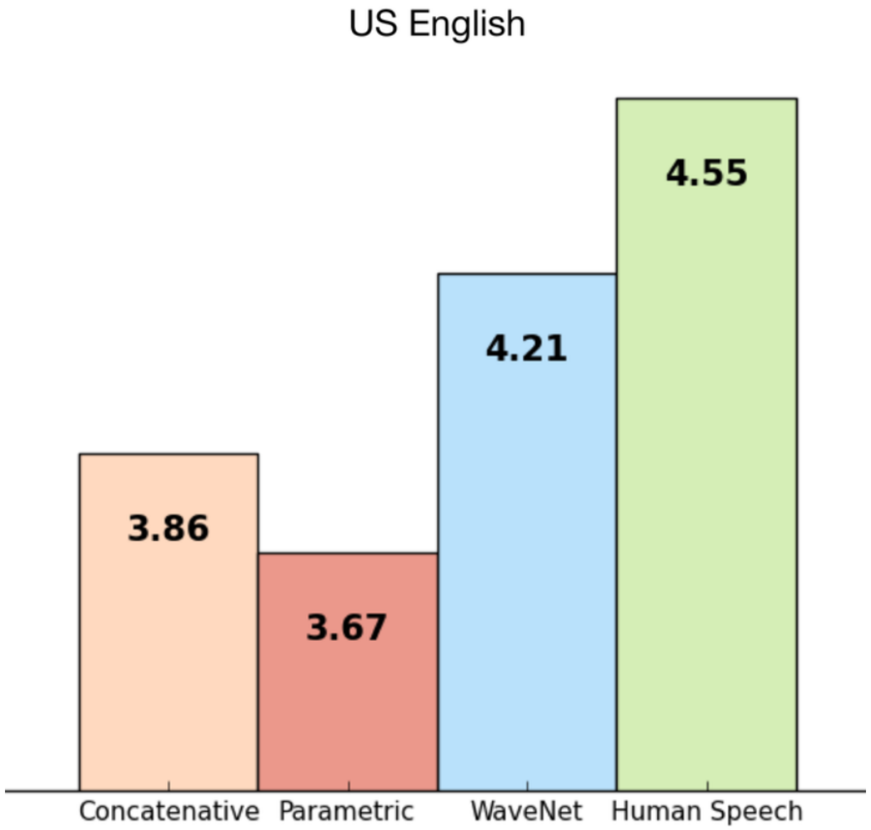
Neural Style Translation (Cool TF projects)

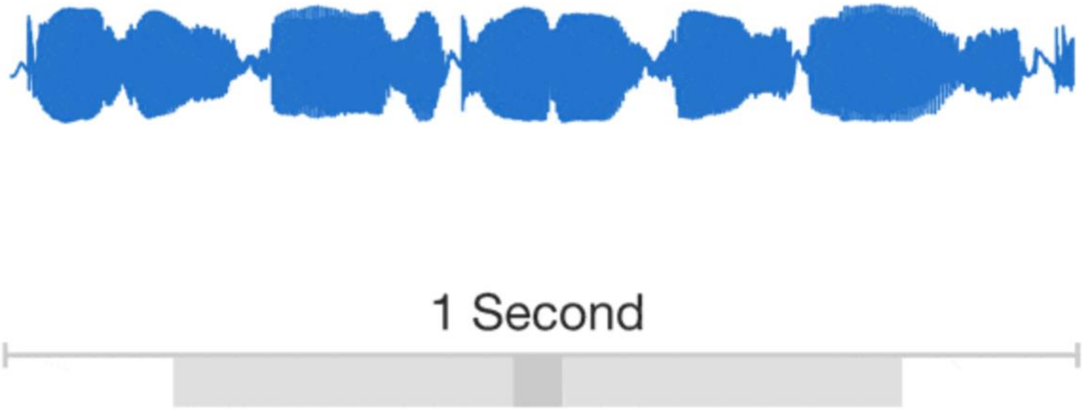
- https://github.com/anishathalye/neural-style



WaveNet: Text to Speech

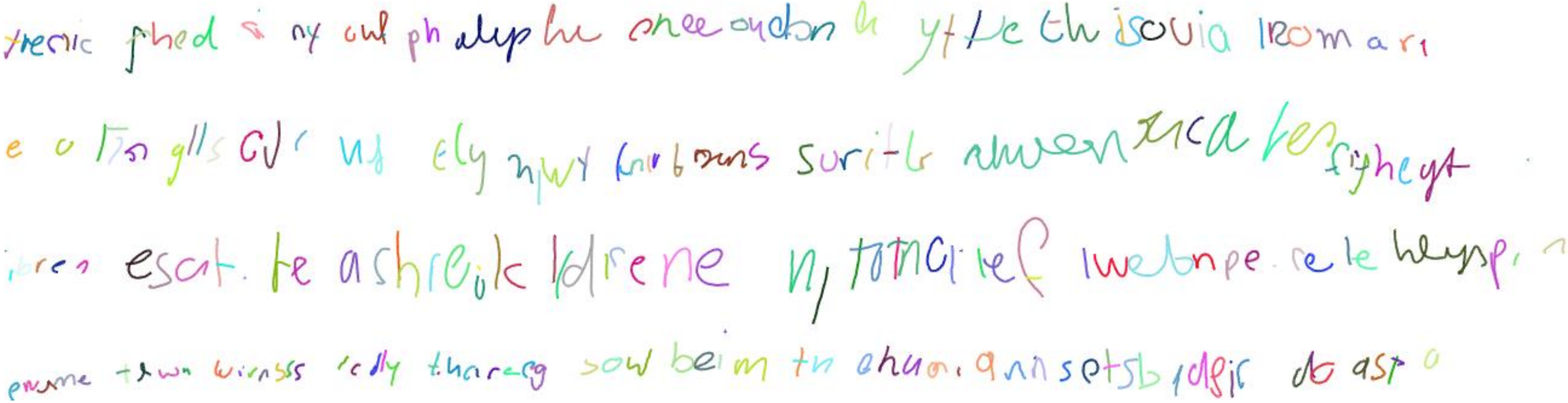
- https://deepmind.com/blog/wavenet-generative-model-raw-audio/





Generative Handwriting

- https://github.com/hardmaru/write-rnn-tensorflow

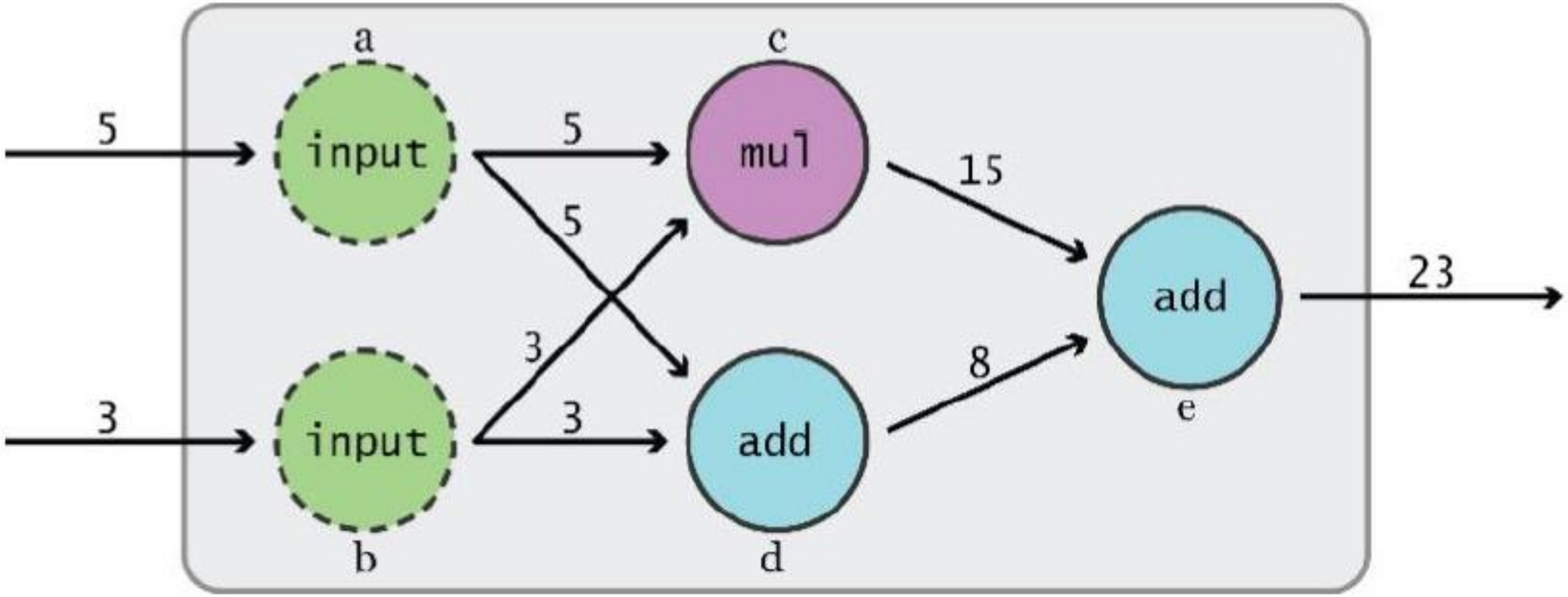


TensorFlow

- TF Learn (tf.contrib.learn): simplified interface that helps users transition from the world of one-liner such as scikit-learn
- TF Slim (tf.contrib.slim): lightweight library for defining, training and evaluating complex models in TensorFlow.
- High level API: Keras, TFLearn, Pretty Tensor
- TensorFlow provides an extensive suite of functions and classes that allow users to define models from scratch.

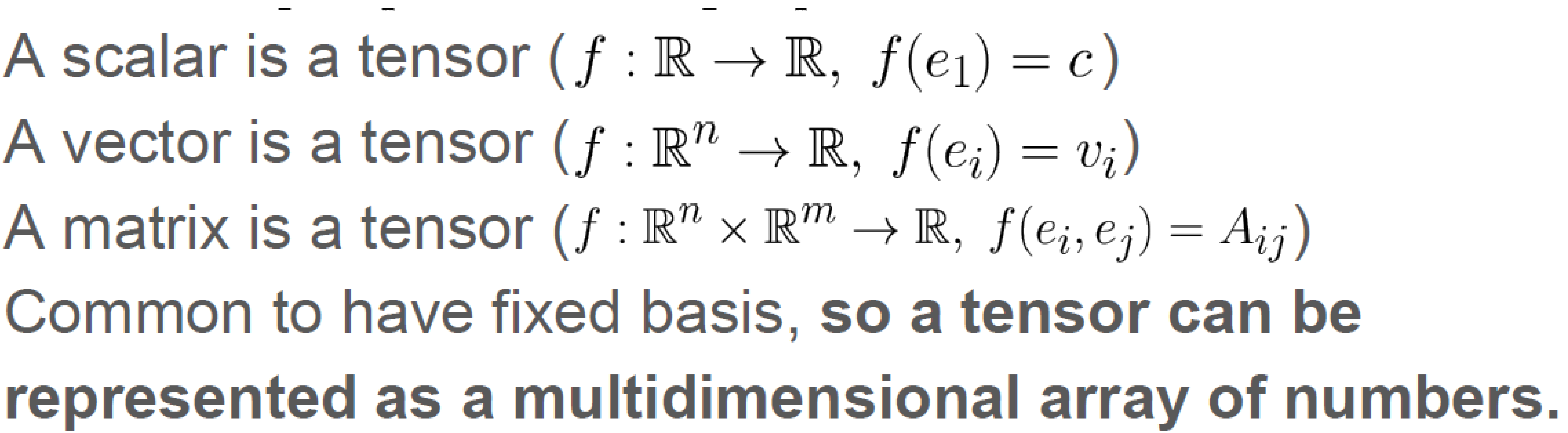
Data Flow Graphs

- Phase 1: assemble a graph
- Phase 2: use a session to execute operations in the graph



What is a tensor

- An n-dimensional array



A scalar is a tensor ($f : \mathbb{R} \rightarrow \mathbb{R},\ f(e_1) = c$)

A vector is a tensor ($f : \mathbb{R}^n \rightarrow \mathbb{R},\ f(e_i) = v_i$)

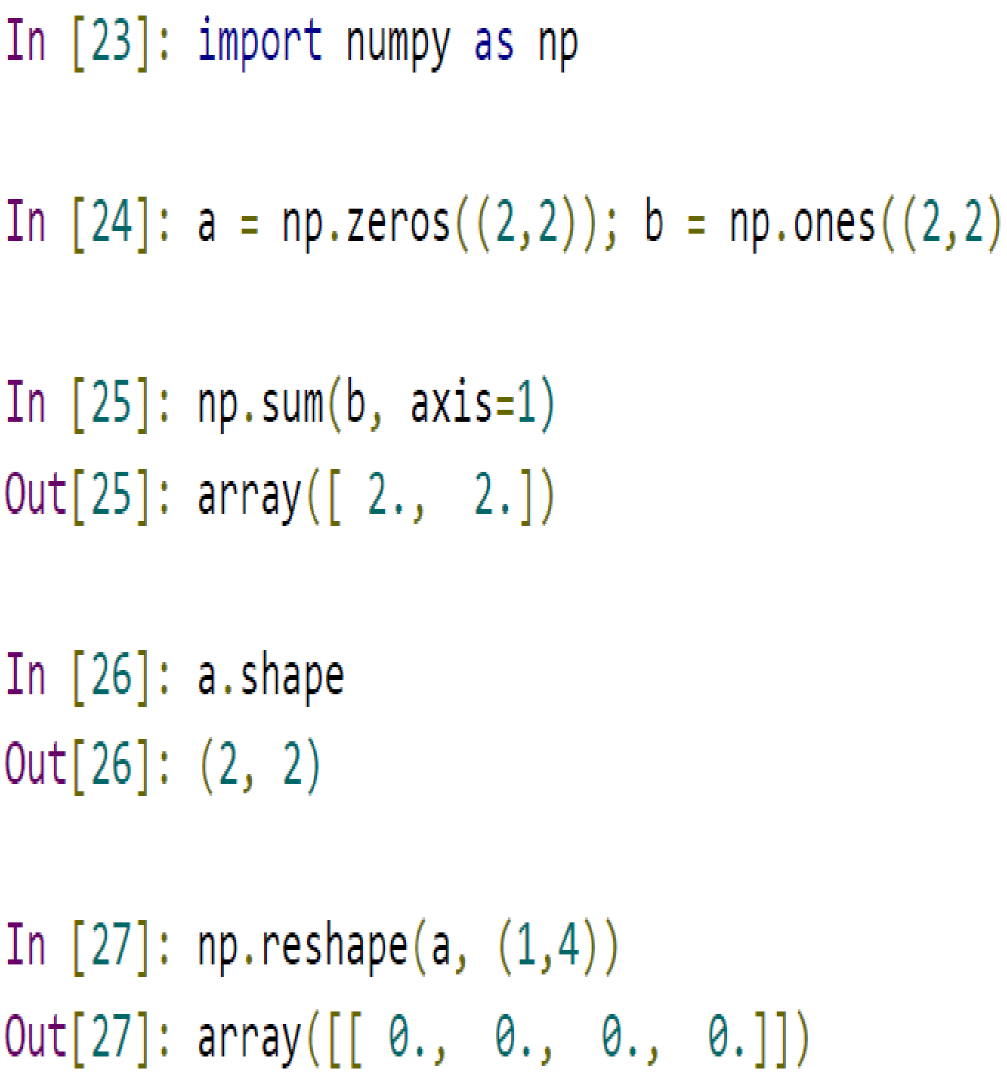
A matrix is a tensor ($f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R},\ f(e_i, e_j) = A_{ij}$)

Common to have fixed basis, **so a tensor can be represented as a multidimensional array of numbers.**

Bharath Ramsundar

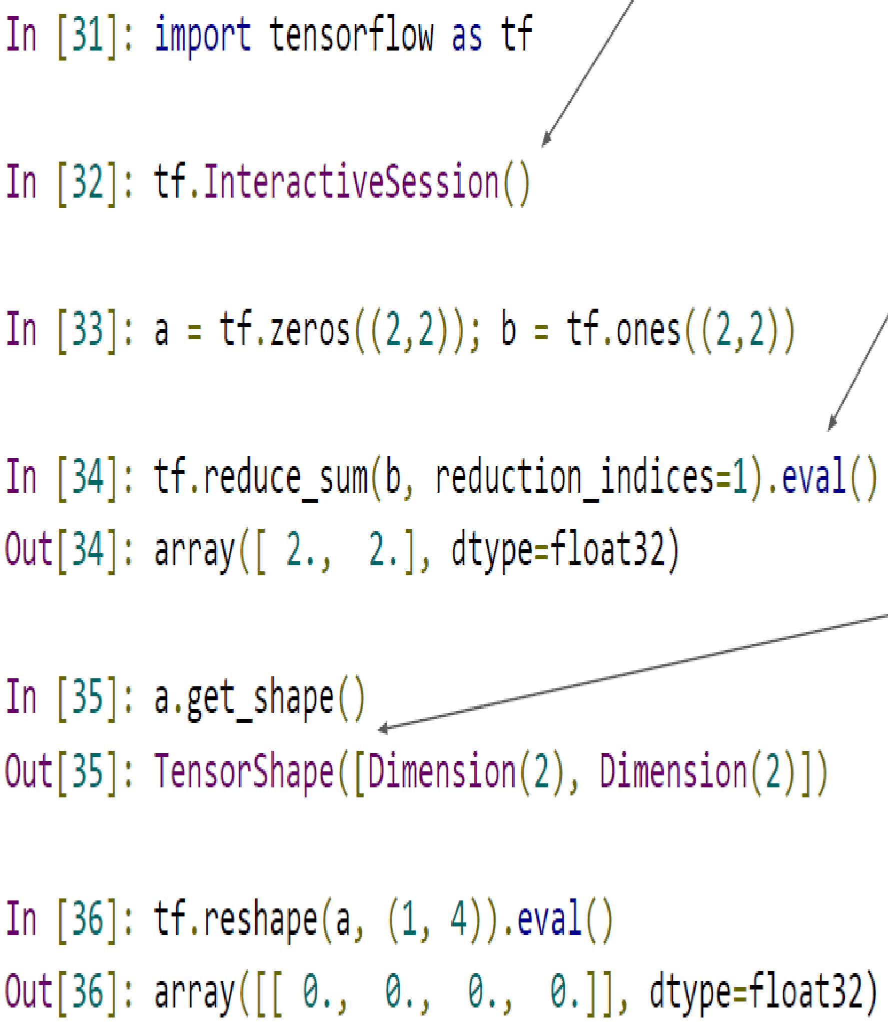
TensorFlow vs. Numpy

- Few people make this comparison, but TensorFlow and Numpy are quite similar. (Both are N-d array libraries!)
- Numpy has Ndarray support, but doesn’t offer methods to create tensor functions and automatically compute derivatives (+ no GPU support).



```
In [23]: import numpy as np

In [24]: a = np.zeros((2,2)); b = np.ones((2,2)

In [25]: np.sum(b, axis=1)
Out[25]: array([ 2.,  2.])

In [26]: a.shape
Out[26]: (2, 2)

In [27]: np.reshape(a, (1,4))
Out[27]: array([[ 0.,  0.,  0.,  0.]])
```

Repeat in TensorFlow



```
In [31]: import tensorflow as tf

In [32]: tf.InteractiveSession()

In [33]: a = tf.zeros((2,2)); b = tf.ones((2,2))

In [34]: tf.reduce_sum(b, reduction_indices=1).eval()
Out[34]: array([ 2.,  2.], dtype=float32)

In [35]: a.get_shape()
Out[35]: TensorShape([Dimension(2), Dimension(2)])

In [36]: tf.reshape(a, (1, 4)).eval()
Out[36]: array([[ 0.,  0.,  0.,  0.]], dtype=float32)
```

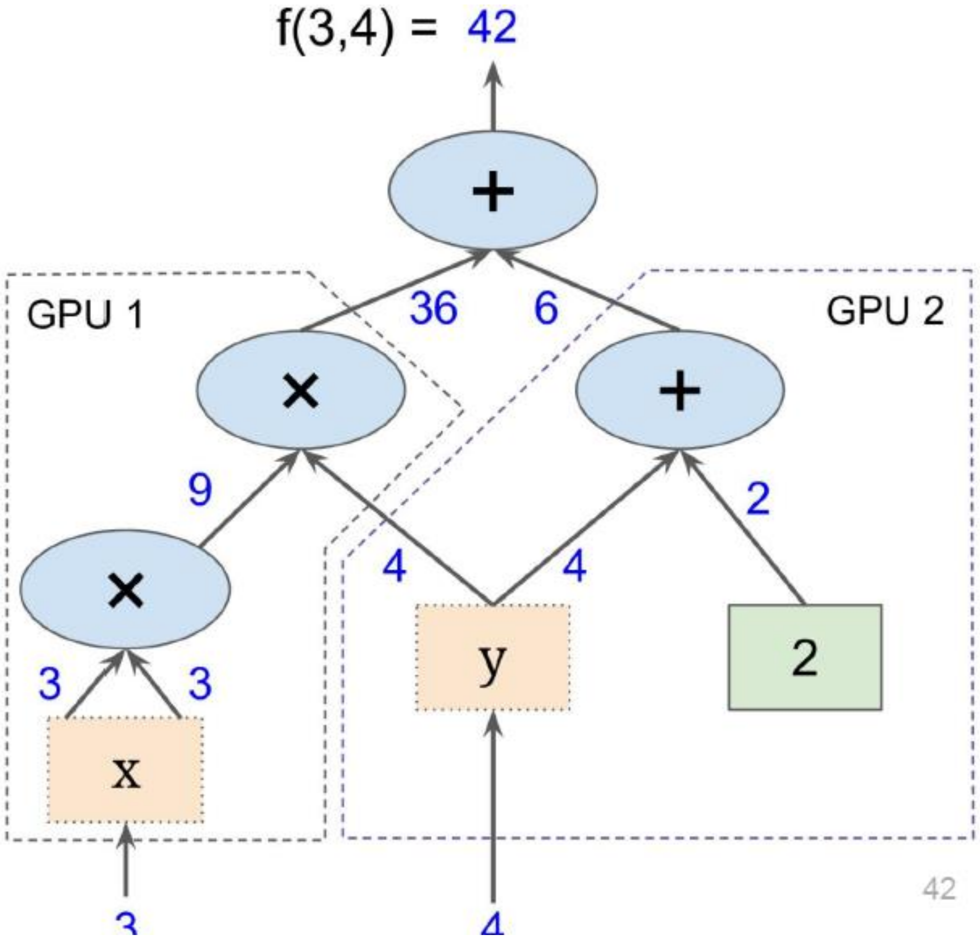
More on `Session` soon

More on `.eval()` in a few slides

`TensorShape` behaves like a python tuple.

tf.Session()

- A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.



- Subgraphs
- Possible to break graphs into several chunks and run them parallel across multiple CPUs, GPUs, or devices

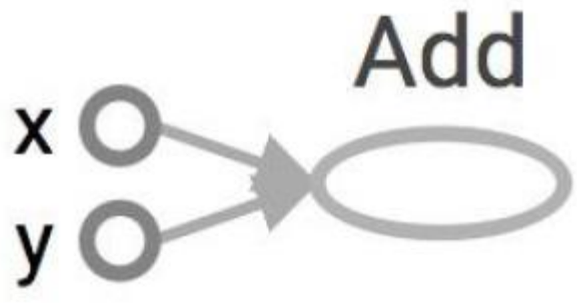
To Summarize

TensorFlow separates the definition of computations from execution

Data Flow Graphs Contd…

Import tensorflow as tf

a = tf.add(2,3)



Tensorflow automatically names the

Nodes when you don’t explicitly name them

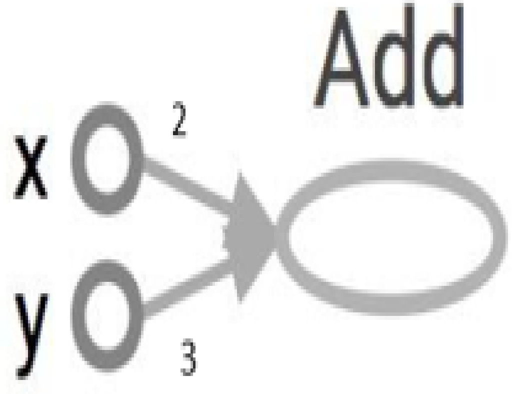
x = 2

y = 3

Data Flow Graphs Contd…

Elements of Graph

- Nodes – operators, variables and constants
- Edges – Tensors
- Tensors are data. In this case 2 represents a tensor, so does 3



Data Flow Graphs Contd…

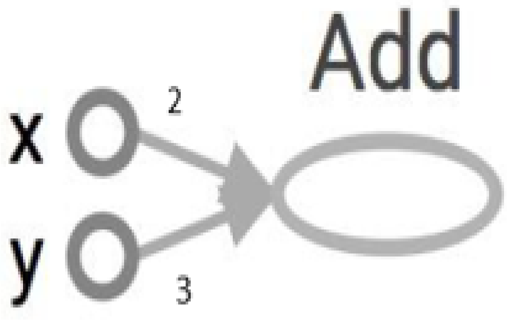
Import tensorflow as tf

a = tf.add(2,3)

Print a

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

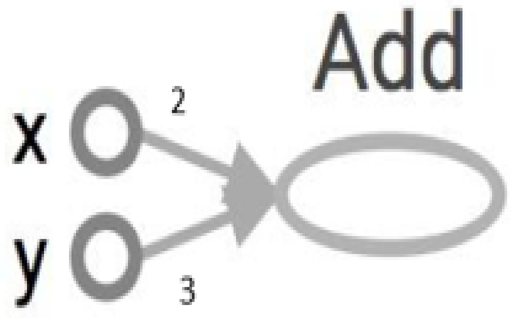
Not 5



Data Flow Graphs Contd…

How to get value of a?

- Create a session and run the add operation represented by ‘a’ using session

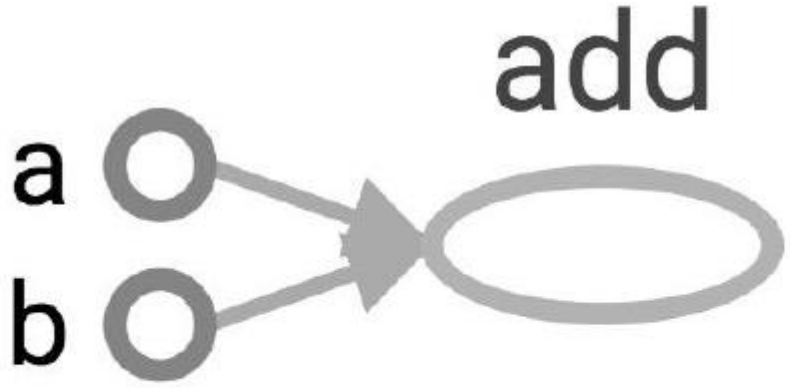


```
Import tensorflow as tf
a = tf.add(2,3)
with tf.Session() as sess:
    print sess.run(a)                        >> 5
```

Data Flow Graphs Contd…

Explicitly name the nodes

```
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
```



Constants Contd…

tf.constant(value, dtype=None, shape = None, name = ‘Constant’, verify_shape = False)

Tensors filled with a specific value

- tf.zeros(shape, dtype=tf.float32, name=None)

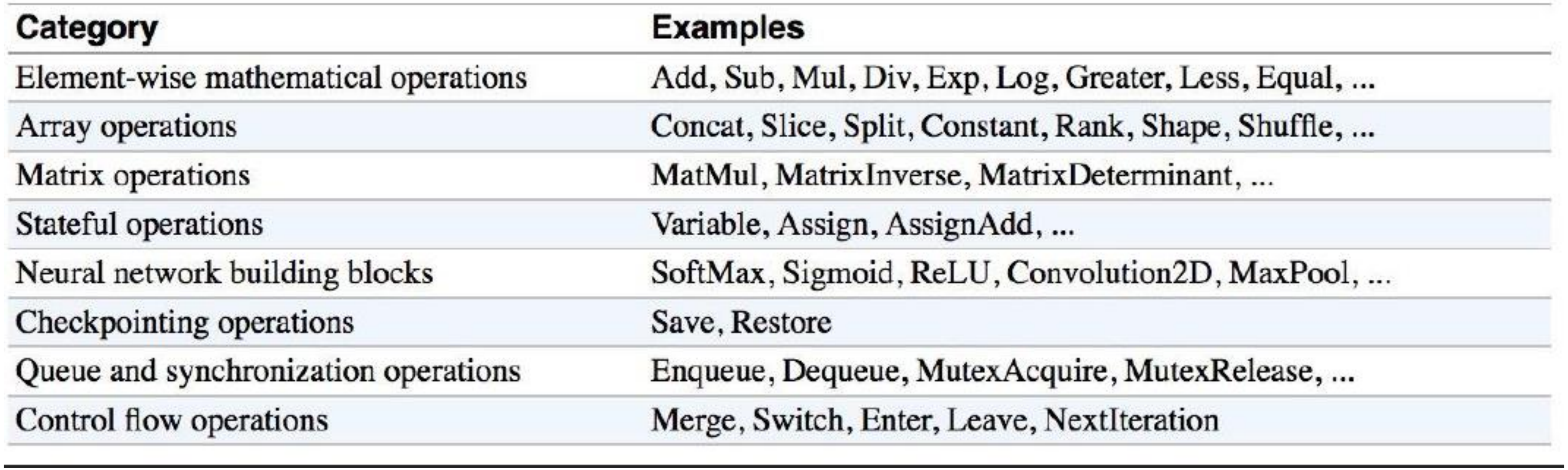
  creates a tensor of shape and all elements will be zeros

tf.zeros([2, 3], tf.int32) ==> [[0, 0, 0], [0, 0, 0]]

Randomly Generated Constants

- **tf.random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)**
- **tf.truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None,**
- **name=None)**
- **tf.random_uniform(shape, minval=0, maxval=None, dtype=tf.float32, seed=None,**
- **name=None)**
- **tf.random_shuffle(value, seed=None, name=None)**
- **tf.random_crop(value, size, seed=None, name=None)**
- **tf.multinomial(logits, num_samples, seed=None, name=None)**
- **tf.random_gamma(shape, alpha, beta=None, dtype=tf.float32, seed=None, name=None)**

Operations

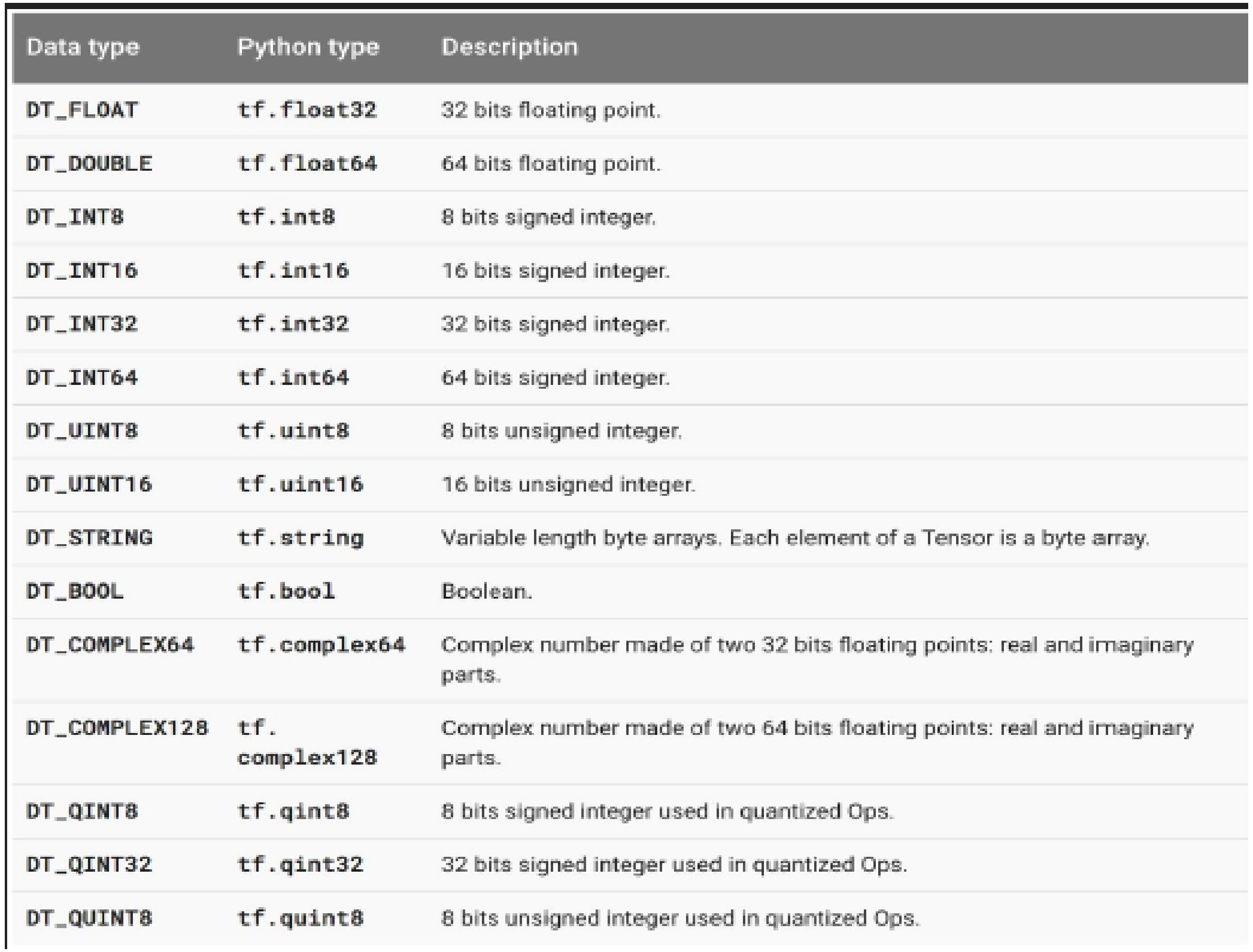


Category	Examples
Element-wise mathematical operations	Add, Sub, Mul, Div, Exp, Log, Greater, Less, Equal, ...
Array operations	Concat, Slice, Split, Constant, Rank, Shape, Shuffle, ...
Matrix operations	MatMul, MatrixInverse, MatrixDeterminant, ...
Stateful operations	Variable, Assign, AssignAdd, ...
Neural network building blocks	SoftMax, Sigmoid, ReLU, Convolution2D, MaxPool, ...
Checkpointing operations	Save, Restore
Queue and synchronization operations	Enqueue, Dequeue, MutexAcquire, MutexRelease, ...
Control flow operations	Merge, Switch, Enter, Leave, NextIteration

Operations Continued

```
a = tf.constant([2, 3])
b = tf.constant([4, 5])
tf.add(a, b) # >> [6  8]
tf.add_n([a, b, b]) # >> [10 13]. Equivalent to a + b + b
tf.mul(a, b) # >> [8 15] because mul is element wise
tf.matmul(a, b) # >> ValueError
tf.matmul(tf.reshape(a, [1, 2]), tf.reshape(b, [2, 1])) # >> [[23]]
tf.div(a, b) # >> [0 0]
tf.mod(a, b) # >> [2 3]
```

Data Types



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.
`DT_COMPLEX128`	`tf.complex128`	Complex number made of two 64 bits floating points: real and imaginary parts.
`DT_QINT8`	`tf.qint8`	8 bits signed integer used in quantized Ops.
`DT_QINT32`	`tf.qint32`	32 bits signed integer used in quantized Ops.
`DT_QUINT8`	`tf.quint8`	8 bits unsigned integer used in quantized Ops.

Variables

```
# create variable s with scalar value
s = tf.Variable(3, name="scalar")
# create variable v as a vector
v = tf.Variable([1, 2], name="vector")
# create variable m as a 3x2 matrix
m = tf.Variable([[1, 2], [2, 3], [3, 4]], name="matrix")
# create variable t as 500x12 tensor, filled with zeros
t = tf.Variable(tf.zeros([500,12]))
```

Variables

Why tf.constant but tf.Variable and not tf.variable?

Because tf.Variable is a class, but tf.constant is an operation

- tf.Variable holds several ops:

1. x = tf.Variable(...)
2. x.initializer # init op
3. x.value() # read op
4. x.assign(...) # write op
5. x.assign_add(...) # and many more

Initialising Variables

- **You have to initialize your variables**

1. The easiest way is initializing all variables at once:

   ```
   init = tf.global_variables_initializer()
   with tf.Session() as sess:
         sess.run(init)
   ```

2. Initialize only specific set of variables

   ```
   init = tf.variables_initializer([a,b], name = “initialize_ab”)
   with tf.Session() as sess:
         sess.run(init)
   ```

3. Initialize a single variable

   ```
   a = tf.Variable(tf.ones([10, 10])
    with tf.Session() as sess:
         sess.run(a.initializer)
   ```

tf.Variable.assign()

```
V = tf.Variable(50)
assign = W.assign(150)
with tf.Session() as sess:
		sess.run(V.initializer)
		sess.run(assign)
print V.eval() # >> 150
```

Placeholder

TF program has two phases

1. Assemble a Graph
2. Use session to run operations in the graph

Placeholders allow you to assemble a graph without knowing the values needed for computation

i.e we can define a function y = 2*x + 3 without knowing the value of x.
x is placeholder for actual value

Placeholders Contd..

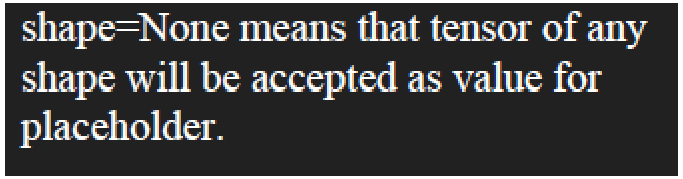
tf.placeholder(dtype, shape=None, name=None)

```
# create a placeholder of type float 32-bit, shape is a vector of 3 elements
a = tf.placeholder(tf.float32, shape=[3])

# create a constant of type float 32-bit, shape is a vector of 3 elements
b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b # Short for tf.add(a, b)

with tf.Session() as sess:
print sess.run(c) # Error because a doesn’t have any value
```

Feed Values using Dictionary

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

```
# create a placeholder of type float 32-bit, shape is a vector of 3 elements
a = tf.placeholder(tf.float32, shape=[3])

# create a constant of type float 32-bit, shape is a vector of 3 elements
b = tf.constant([5, 5, 5], tf.float32)

# use the placeholder as you would a constant or a variable
c = a + b # Short for tf.add(a, b)

with tf.Session() as sess:
        # feed [1, 2, 3] to placeholder a via the dict {a: [1, 2, 3]}
        # fetch value of c
        print sess.run(c, {a: [1, 2, 3]}) # the tensor a is the key, not the string ‘a’

# >> [6, 7, 8]
```



shape=None means that tensor of any shape will be accepted as value for placeholder.

What if want to feed multiple data points in?

```
with tf.Session() as sess:
        for value in list_of_values:
        print sess.run(c, {a: a_value})
```

Next Class

1. Linear Regression in TensorFlow
2. Optimizers
3. Logistic Regression