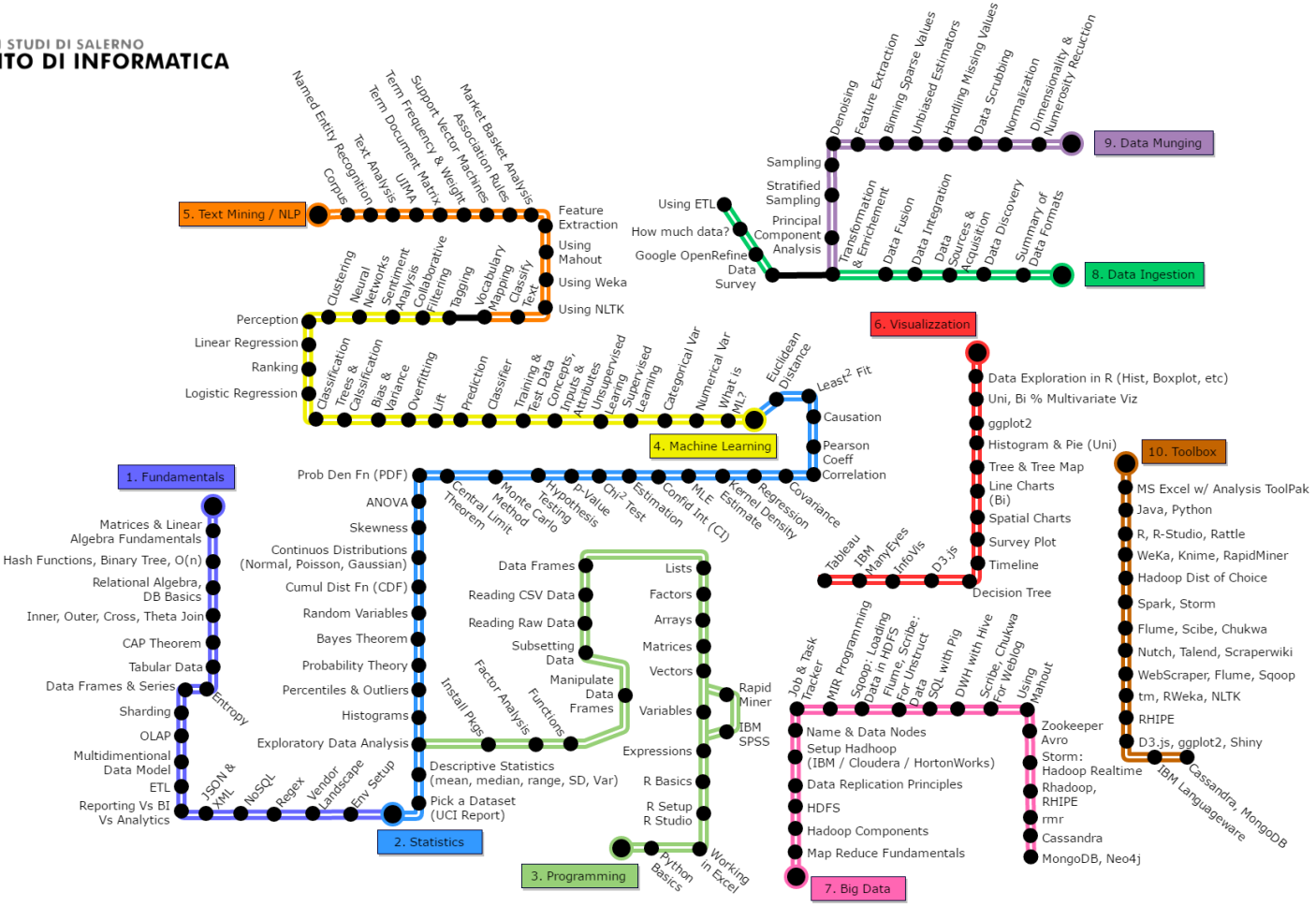
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA



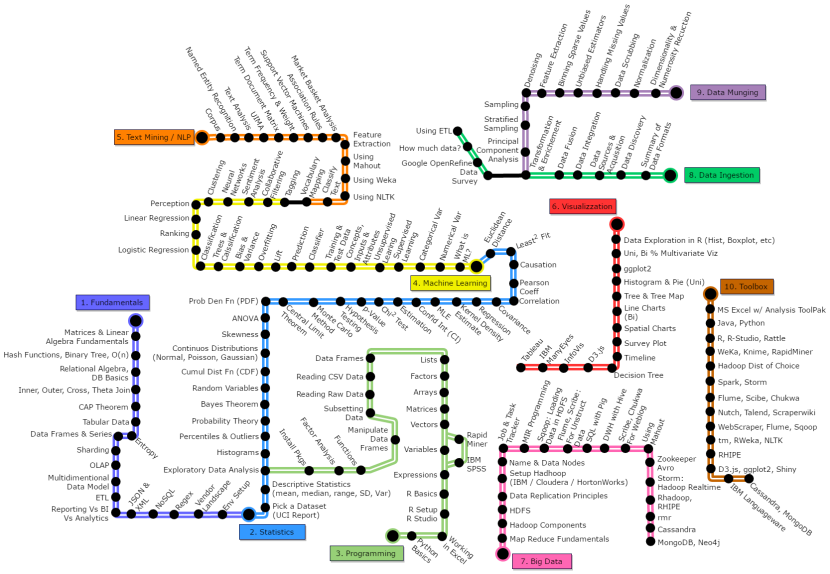
Fondamenti di Data Science e Machine Learning

Introduction to TensorFlow

Prof. Giuseppe Polese, aa 2024-25

Outline

- **Introduction to TensorFlow**
- Installation
- The DataFlow graph
- TensorFlow programs
- TensorBoard



TensorFlow

A Graph of Operations

A Geometric Object

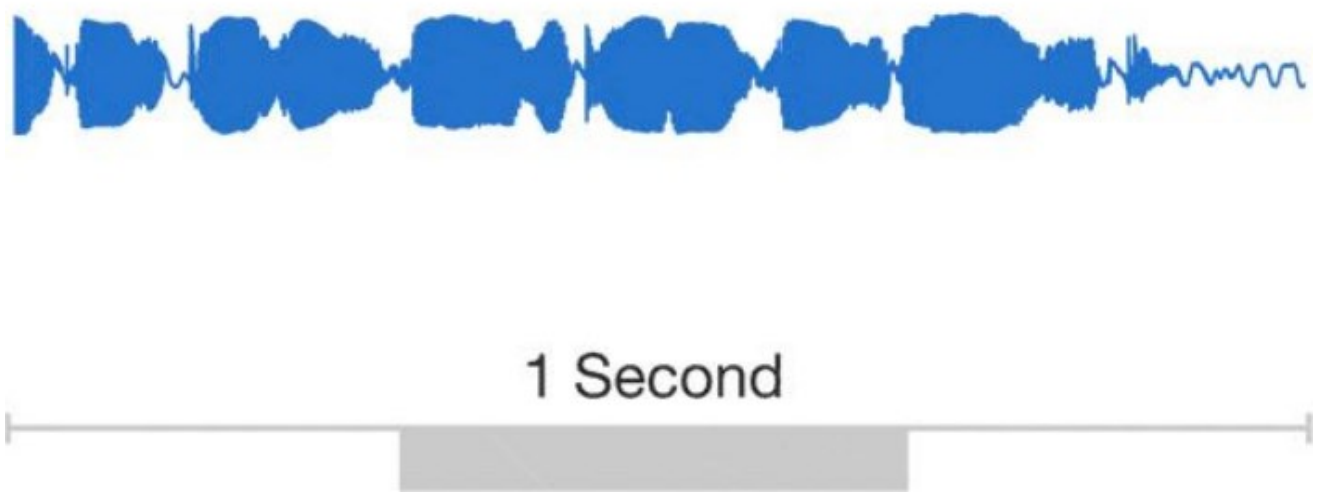
What's TensorFlow?

- 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
- Especially useful for Deep Learning
- **Apache 2.0** license

History of TensorFlow

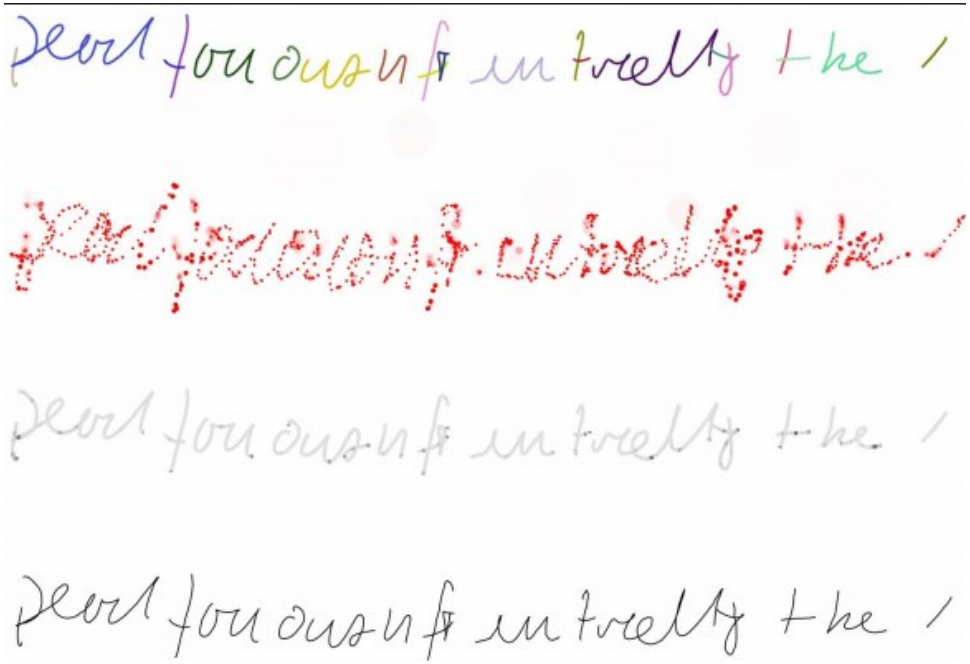
- TensorFlow was developed by Google and it powers many of Google's large-scale services, such as Google Cloud Speech, Google Photos, and Google Search
- It was open-sourced in November 2015
- Most of its features already existed in some previous Deep Learning library, but none contained all of them
- TensorFlow was designed to be flexible, scalable, and production-ready

WaveNet: Text to Speech



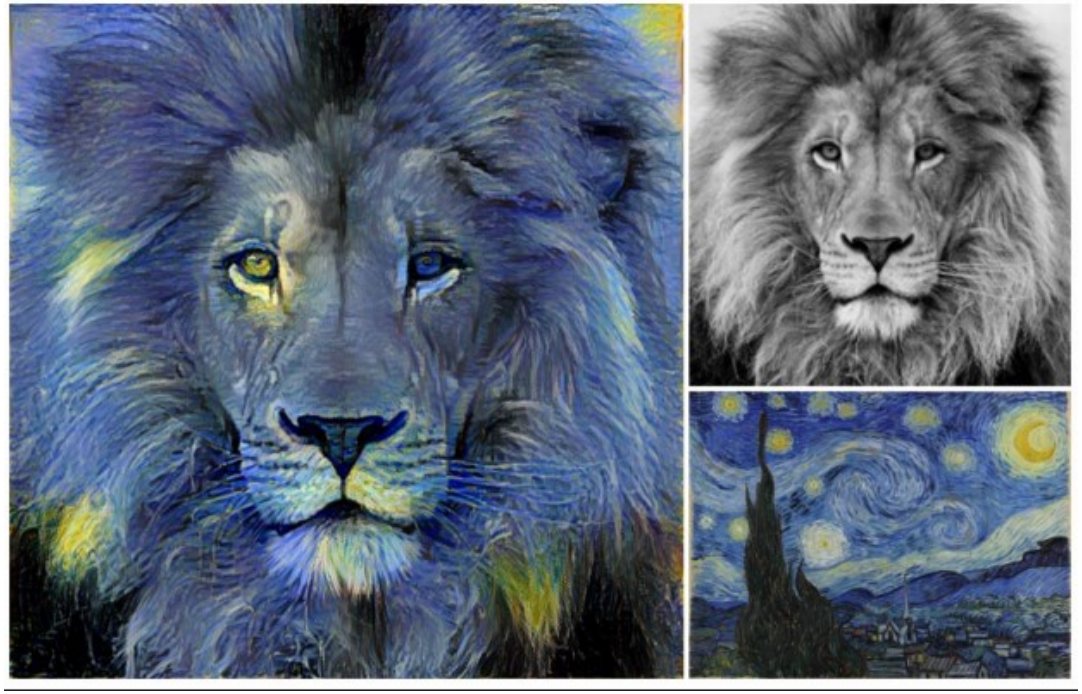
"Wavenet: A generative model for raw audio" by Aaron van den Oord et al. (2016)

Generative Handwriting



“Generative Handwriting using LSTM Mixture Density Network with TensorFlow” by hardmaru@GitHub (2016)

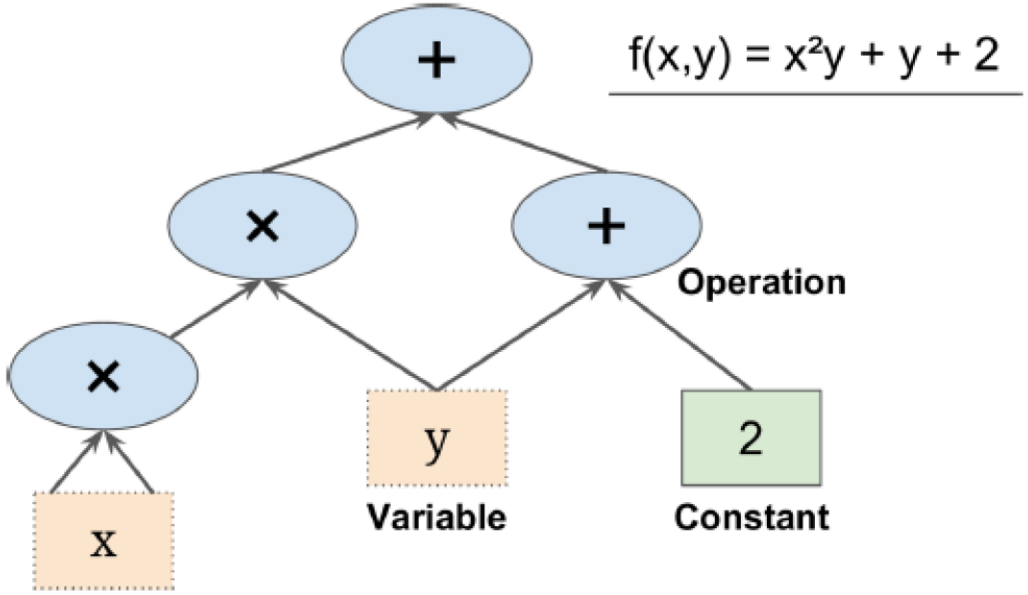
Neural Style Traslation



“Image Style Transfer Using Convolutional Neural Networks” by Leon A. Gatys et al. (2016)
Tensorflow adaptation by Cameroon Smith

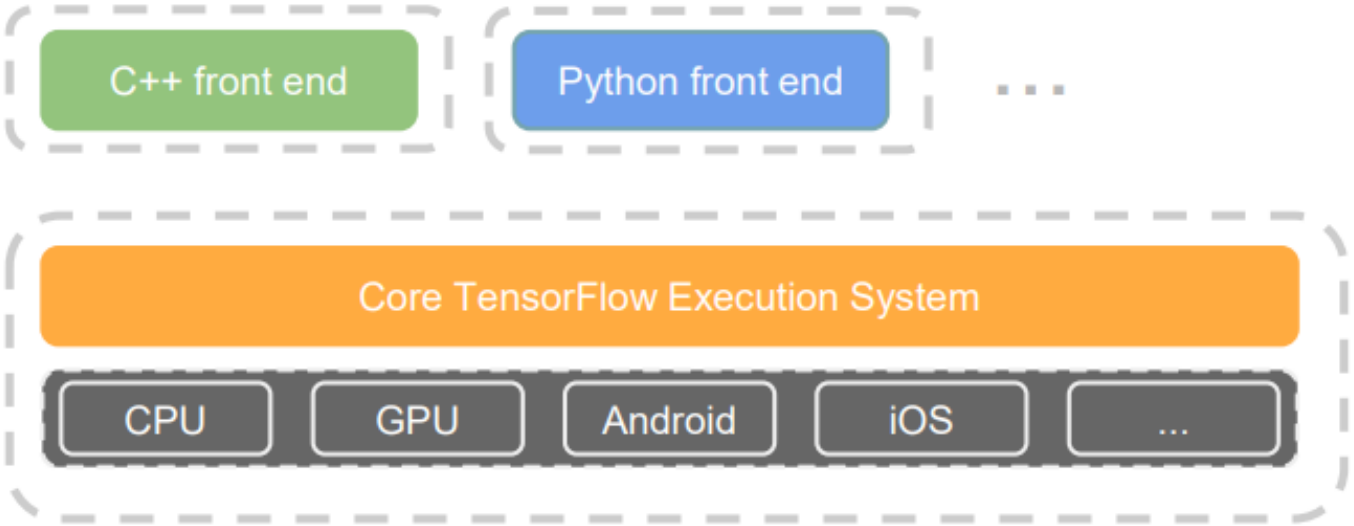
Graph based computation

- TensorFlow is an open source software library for numerical computation, well suited and fine-tuned for large-scale Machine Learning
- Its basic principle is: we first define in Python a graph of computations to perform, and TensorFlow runs it using optimized C++



Architecture

- Core: C++
- Front ends: Python, C++, Go, R, Scala and more



Main Characteristics

- TensorFlow runs not only on Windows, Linux, and macOS, but also on mobile devices like iOS and Android
- Provides the Python API TF.Learn, compatible with Scikit-Learn, which can be used to train various types of neural networks in few lines of code
- Provides the TF-slim API to simplify building, training, and evaluating neural networks.
- Other high-level APIs have been built on top of Tensor-Flow, such as Keras or Pretty Tensor.
- Its main Python API offers great flexibility to create all sorts of computations, including any neural network architecture

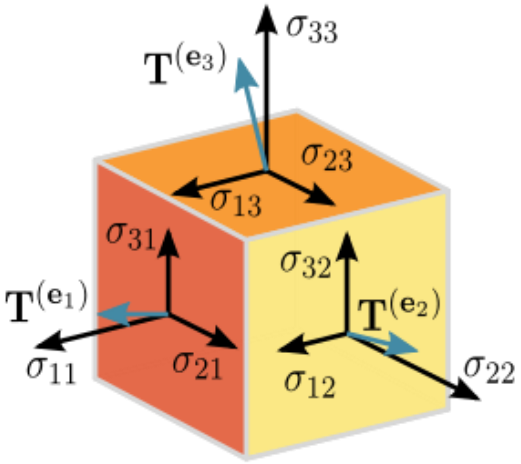
Main Characteristics (2)

- It includes efficient C++ implementations of many ML operations, including those to build neural networks
- Provides several optimization nodes to search for the parameters minimizing a cost function: it automatically computes gradients of user-defined functions (autodiff).
- Provides TensorBoard, a visualization tool allowing to browse the computation graph, view learning curves, etc.
- Google launched a cloud service to run TensorFlow graphs
- It has a team of developers, and a community contributing to improving it. It is one of the most popular open source projects on GitHub, and new projects are being built on it

Tensor

- A tensor is a geometric object that maps in a multi-linear manner geometric vectors, scalars, and other tensors to a resulting tensor
- (V vector space, V^* dual space)

$$f : \underbrace{V^* \times \cdots V^*}_{p \text{ copies}} \times \underbrace{V \times \cdots V}_{q \text{ copies}} \rightarrow \mathbb{R}$$



- A **vector** is a tensor $\left(f : \mathbb{R}^n \rightarrow \mathbb{R},\ f(e_i) = v_i\right)$
- A **matrix** is a tensor $\left(f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R},\ f(e_i, e_j) = A_{ij}\right)$
- A **scalar** is a tensor $\left(f : \mathbb{R} \rightarrow \mathbb{R},\ f(e_1) = c\right)$
- Tensor can be viewed as a **multidimensional array of numbers**

OpenSource Deep Learning Libraries

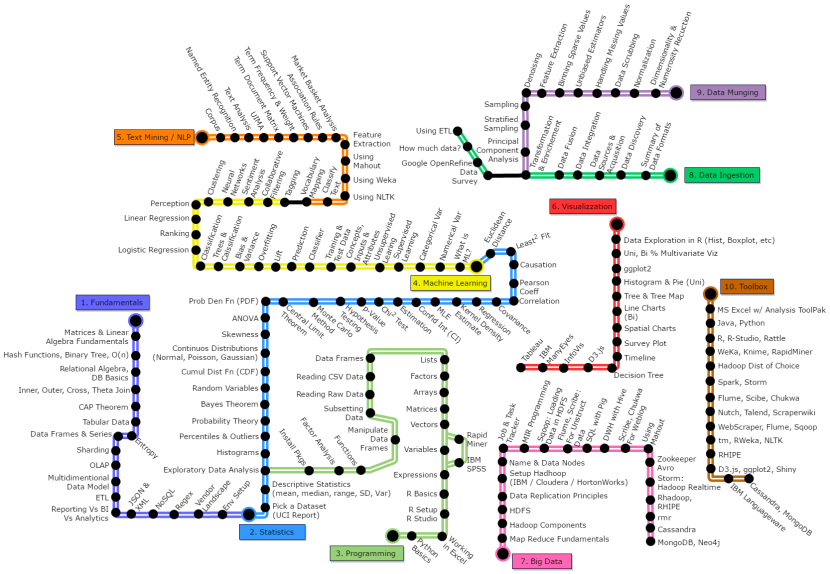
Library	API	Platforms	Started by	Year
Caffe	Python, C++, Matlab	Linux, macOS, Windows	Y. Jia, UC Berkeley (BVLC)	2013
Deeplearning4j	Java, Scala, Clojure	Linux, macOS, Windows, Android	A. Gibson, J.Patterson	2014
H2O	Python, R	Linux, macOS, Windows	H2O.ai	2014
MXNet	Python, C++, others	Linux, macOS, Windows, iOS, Android	DMLC	2015
TensorFlow	Python, C++	Linux, macOS, Windows, iOS, Android	Google	2015
Theano	Python	Linux, macOS, iOS	University of Montreal	2010
Torch	C++, Lua	Linux, macOS, iOS, Android	R. Collobert, K. Kavukcuoglu, C. Farabet	2002

Web Resources

- Resources page on https://www.tensorflow.org/, or https://github.com/jtoy/awesome-tensorflow.
- To ask technical questions http://stackoverflow.com/ and tag your question with "tensorflow".
- It is possible to file bugs and feature requests through GitHub.
- For general discussions, join the Google group

Outline

- Introduction to TensorFlow
- **Installation**
- The DataFlow graph
- TensorFlow programs
- TensorBoard



Installation

- After installing Jupyter and Scikit-Learn we can install TensorFlow by using pip
- If we created an isolated environment using virtualenv, we first need to activate it:

```
$ cd $ML_PATH # Our ML working directory (e.g., $HOME/ml)
$ source env/bin/activate
```

- And then install TensorFlow:

```
$ pip3 install --upgrade tensorflow
```

- For GPU support, install tensorflow-gpu instead
- To verify the TensorFlow version installed:

```
$ python3 -c 'import tensorflow; print(tensorflow.__version__)'
1.3.0
```

Installing TensorFlow with *pip*

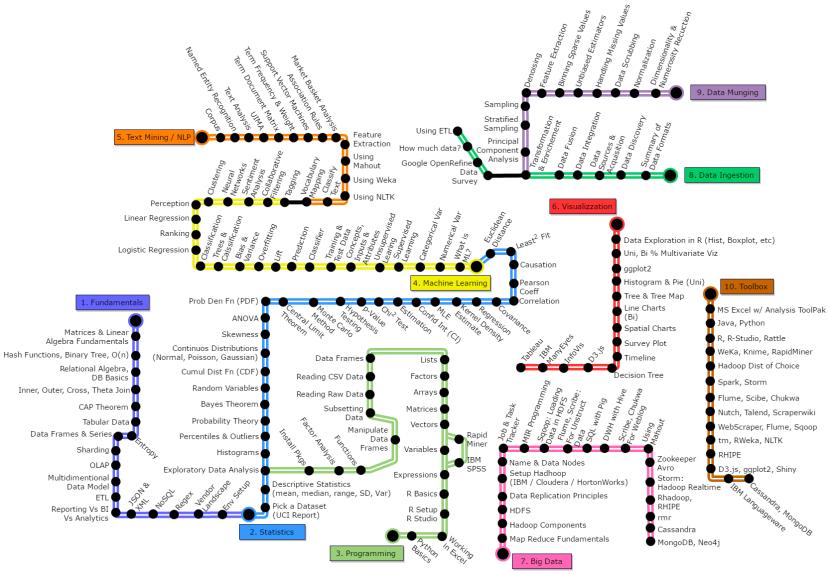
- Install TensorFlow with Python's *pip* package manager:

```
# Current release for CPU-only
$ pip install tensorflow

# Nightly build for CPU-only (unstable)
$ pip install tf-nightly

# GPU package for CUDA-enabled GPU cards
$ pip install tensorflow-gpu

# Nightly build with GPU support (unstable)
$ pip install tf-nightly-gpu
```

- GPU packages require a CUDA®-enabled GPU card

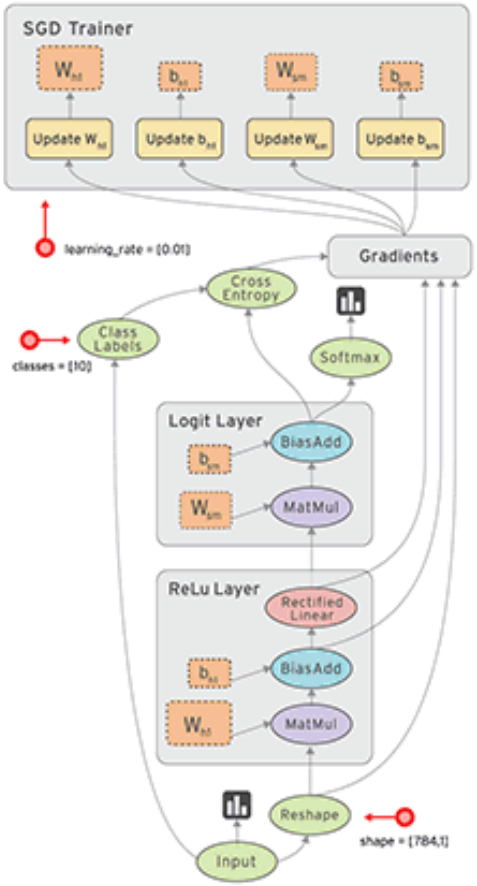
Outline

- Introduction to TensorFlow
- Installation
- **The DataFlow graph**
- TensorFlow programs
- TensorBoard



DataFlow Graph

- **Dataflow** is a common programming model for parallel computing
- TensorFlow uses a **dataflow graph** to represent your computation in terms of the dependencies between individual operations
- Graph is defined in high-level language
- Graph is compiled and optimized
- Graph is executed (in parts or fully) on available low level devices (CPU, GPU, TPU)



DataFlow Advantages

- **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:** Program execution can be distributed to GPUs, CPUs, and TPUs
- **Compilation:** TensorFlow's XLA compiler can use the information in your dataflow graph to generate faster code
- **Portability:** The dataflow graph is a language-independent representation of the code which can be saved and executed with other compilers

Creating and Running a Graph

- This code creates the graph of the previous expression:

```
import tensorflow as tf
x = tf.Variable(3, name="x")
y = tf.Variable(4, name="y")
f = x*x*y + y + 2
```

- The code just creates a computation graph
- To evaluate it we need to open a TensorFlow session and use it to initialize the variables and evaluate f.
- The session places operations onto devices such as CPUs and GPUs, runs them, and holds their values

Graph Evaluation

- This code creates a session, initializes variables, evaluates f, and closes the session (freeing resources):

```
>>> sess = tf.Session()
>>> sess.run(x.initializer)
>>> sess.run(y.initializer)
>>> result = sess.run(f)
>>> print(result)
42
>>> sess.close()
```

- To avoid repeating sess.run():
- The with block sets the session to *default*

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

- Calling x.initializer.run() is equivalent to calling tf.get_default_session().run(x.initializer)
- The session is automatically closed at the end of the block

Graph Evaluation

- Alternatively, we can use global_variables_initializer() (creates a graph node that initializes variables when run)

```
init = tf.global_variables_initializer() # prepare an init node
```

- The code does not perform the initialization immediately, but it creates a graph node that will initialize all variables when it is run:

```
with tf.Session() as sess:
init.run() # actually initialize all the variables
result = f.eval()
```

Lifecycle of a Node Value

- When evaluating a node, TensorFlow determines the nodes it depends on, and evaluates them first

```
w = tf.constant(3)
x = w + 2
y = x + 5
z = x * 3
with tf.Session() as sess:
print(y.eval()) # 10
print(z.eval()) # 15
```

- y depends on x, which depends on w, so TensorFlow first evaluates w, then x, then y, and returns y. Then, the code runs the graph to evaluate z. It will not reuse previous evaluations of w and x, but it evaluates them twice

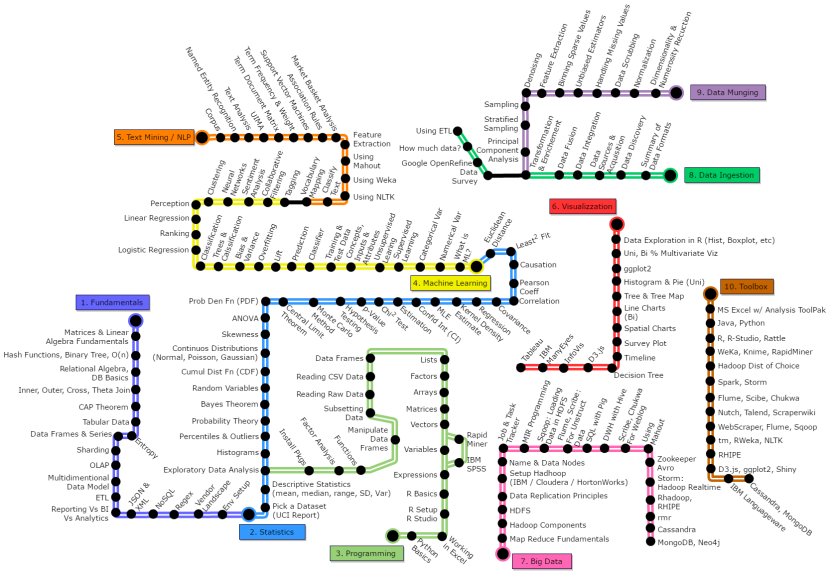
Lifecycle of a Node Value (2)

- Values are dropped between graph runs, except variable values, which are maintained across graph runs
- A variable starts its life when its initializer is run, and it ends when the session is closed
- If we want to avoid evaluating w and x twice when evaluating y and z efficiently, we must ask TensorFlow to compute y and z in one graph

```
with tf.Session() as sess:
y_val, z_val = sess.run([y, z])
print(y_val) # 10
print(z_val) # 15
```

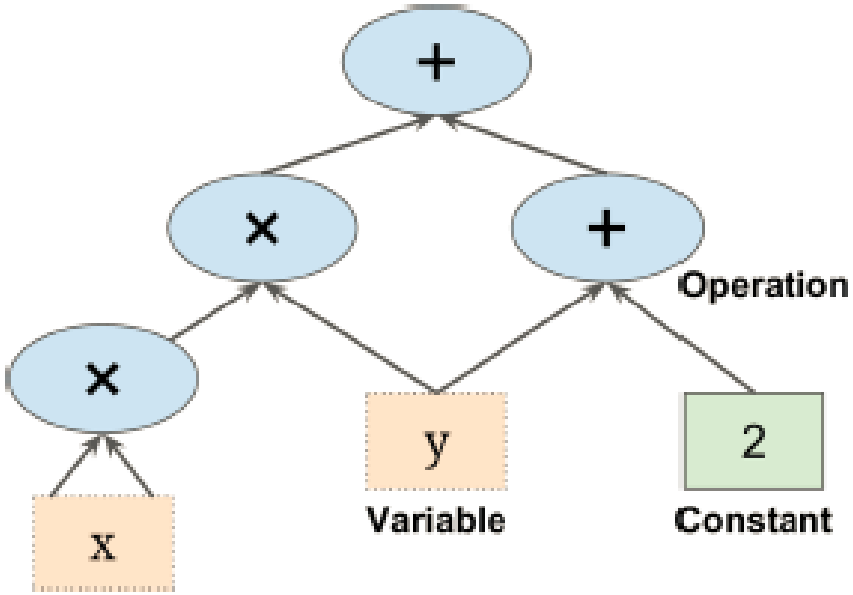
Outline

- Introduction to TensorFlow
- Installation
- The DataFlow graph
- **TensorFlow programs**
- TensorBoard



TensorFlow Programs

- A typical program consists of 2 phases:
  - **Construction Phase:** assembling a graph (model)
  - **Execution Phase**: starting session and pushing data through the graph



Construction & Execution Phases

- A TensorFlow program consists of two parts:
  - the first one builds a computation graph (*construction phase)*
  - the second one runs it (*execution phase)*
- The construction phase builds a computation graph representing the ML model and the computations required to train it.
- The execution phase runs a loop evaluating a training step repeatedly, gradually improving model parameters

TensorFlow Operations

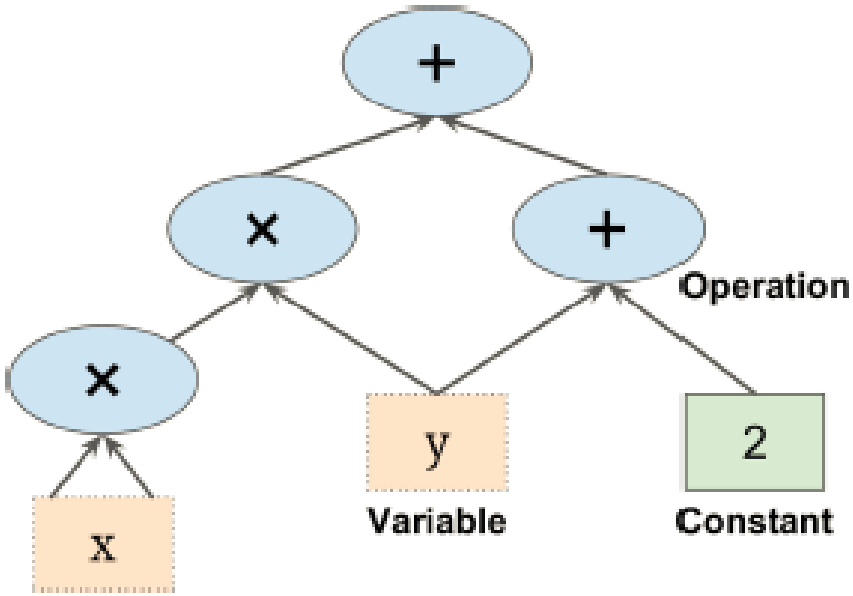
- TensorFlow operations (a.k.a. ops) can take any number of inputs and produce any number of outputs
- Constants and variables (a.k.a. source ops) take no input
- Inputs and outputs are multidimensional arrays, called tensors
- Like NumPy arrays, tensors have a type and a shape. In fact, in Python's API they are represented by NumPy ndarrays.
- Typically contain floats, but they can also carry strings

Functions as Computational Graphs

- View functions as computational graphs:

$$f(x, y) = x^2 y + y + 2$$

- Nodes are operators (ops), variables, and constants
- Edges are tensors
  - 0-d is a scalar
  - 1-d is a vector
  - 2-d is a matrix
  - etc

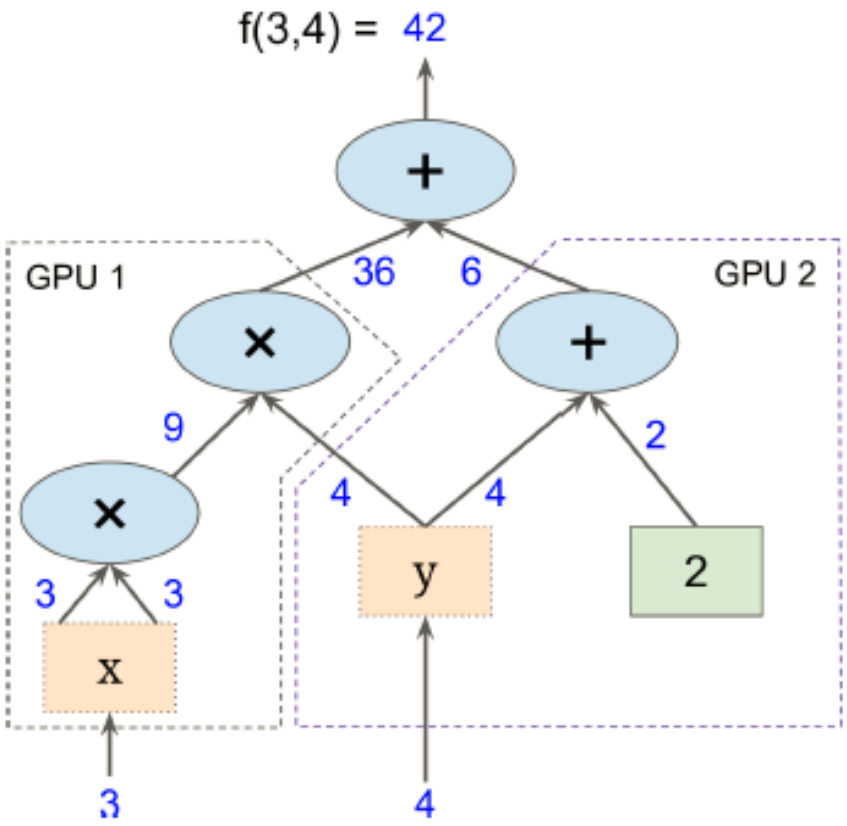


Types of Tensors

- **Constants** are fixed value tensors - *not trainable*
- **Variables** are tensors initialized in a session - *trainable*
- **Placeholders** are tensors of values that are unknown during the graph construction, but passed as input during a session
- **Ops** are functions on tensors

Parallel Execution of Graphs

- It is possible to break up the graph into several chunks and run them in parallel across multiple CPUs or GPUs
- TensorFlow can train a network with millions of parameters on a training set composed of billions of instances with millions of features each



Variables

- Variables can be created using `tf.Variable()` function

```
import tensorflow as tf
initial_value = 3
w = tf.Variable(initial_value, name="w")
```

- When launch the graph, variables have to be explicitly initialized before you can run `Ops` that use their value:
  - **Single Initialization:**

```
w = tf.Variable(initial_value, name="w")
with tf.Session() as s:
    s.run(w.initializer)
```

  - **Global Initialization:**

```
# Add an Op to initialize global variables.
init_op = tf.global_variables_initializer()
with tf.Session() as s:
    s.run(init_op)
```

Costants

- Constant can be created using `tf.constant()` function

```
import tensorflow as tf
x = tf.constant([1, 2, 3, 4])
```

- A constant has the following arguments which can be tweaked as required to get the desired function
  - **value**: A constant value (or list) of output type dtype
  - **dtype**: The type of the elements of the resulting tensor
  - **shape**: Optional dimensions of resulting tensor
  - **name**: Optional name for the tensor
  - **verify_shape**: Boolean that enables verification of a shape of values

Placeholders

- Placeholders can be created using `tf.Placeholder()` function

```
import tensorflow as tf
A = tf.placeholder(tf.float32, shape=(None, 3))
B = A + 5
with tf.Session() as sess:
    B_val_1 = B.eval(feed_dict={A: [[1, 2, 3]]})
    B_val_2 = B.eval(feed_dict={A: [[4, 5, 6], [7, 8, 9]]})
    print(B_val_1, B_val_2)
```

- Inserts a placeholder for a tensor that will be always fed
- Its value must be fed using the `feed_dict` optional argument to `Session.run()`, `Tensor.eval()`, or `Operation.run()`

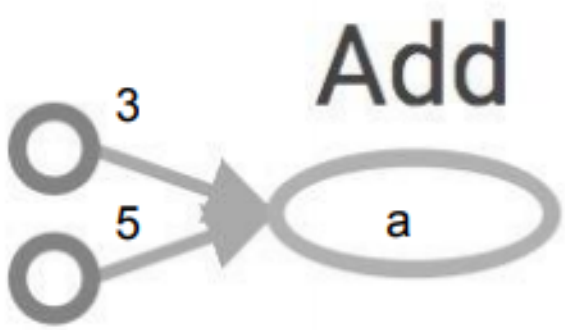
Example (1)


```
# Import `tensorflow`
import tensorflow as tf
a = tf.add(3,5)
print(a)
```

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

Example (2)

- Session creation:



Output: 8

```
import tensorflow as tf
a = tf.add(3,5)
s = tf.Session()
output = s.run(a)
print(output)
s.close()
```

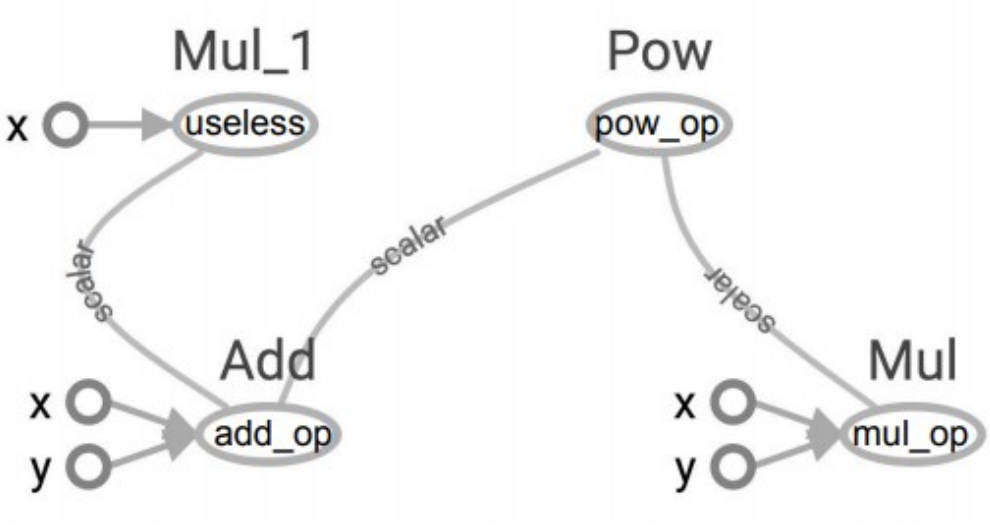
or

```
import tensorflow as tf
a = tf.add(3,5)
with tf.Session() as s:
    output = s.run(a)
    print(output)
```

Example (3)

- Complex graph:

```
import tensorflow as tf

x = tf.Variable(2, name="x")
y = tf.Variable(3, name="y")

add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)
with tf.Session() as s:
    #Initialization of variables
    s.run(x.initializer)
    s.run(y.initializer)
    z = s.run(pow_op)
    print(z)
```



Output: 15625

Distributed Execution

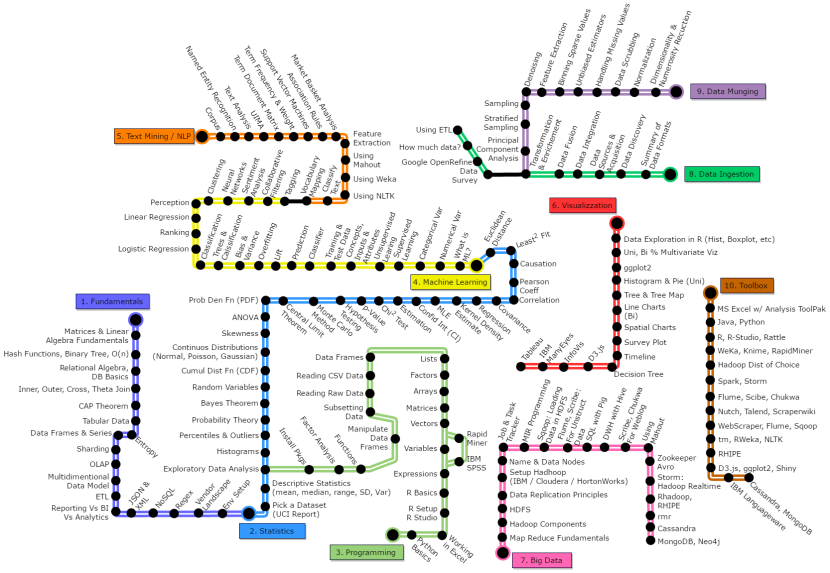
- Computation on specific CPUs/GPUs/servers:

```
# Creates a graph.
with tf.device('/gpu:0'):
    a = tf.constant([1, 2, 3, 4, 5, 6], shape=[2, 3])
    b = tf.constant([7, 8, 9, 10, 11, 12], shape=[3, 2])
    c = tf.matmul(a, b)

# Creates a session with log_device_placement set to True.
with tf.Session(config = tf.ConfigProto(log_device_placement=True)) as s:
    # Runs the op.
    print(s.run(c))
```

- The setting `log_device_placement=True` enable the printing of the logs in the console

Outline

- Introduction to TensorFlow
- Installation
- The DataFlow graph
- TensorFlow programs
- **TensorBoard**



TensorBoard: Visualizing Learning

- To make it easier to understand, debug, and optimize TensorFlow programs, you can use a suite of visualization tools called **TensorBoard:**
  - **Visualize** TensorFlow graphs
  - **Plot** quantitative metrics about the execution of your graph
  - **Show** additional data like images that pass through it
- Installing TensorFlow via *pip* should also automatically install TensorBoard

TensorBoard: Configuration

- To use TensorBoard and view an application's graph, you must save the graph from the source code:

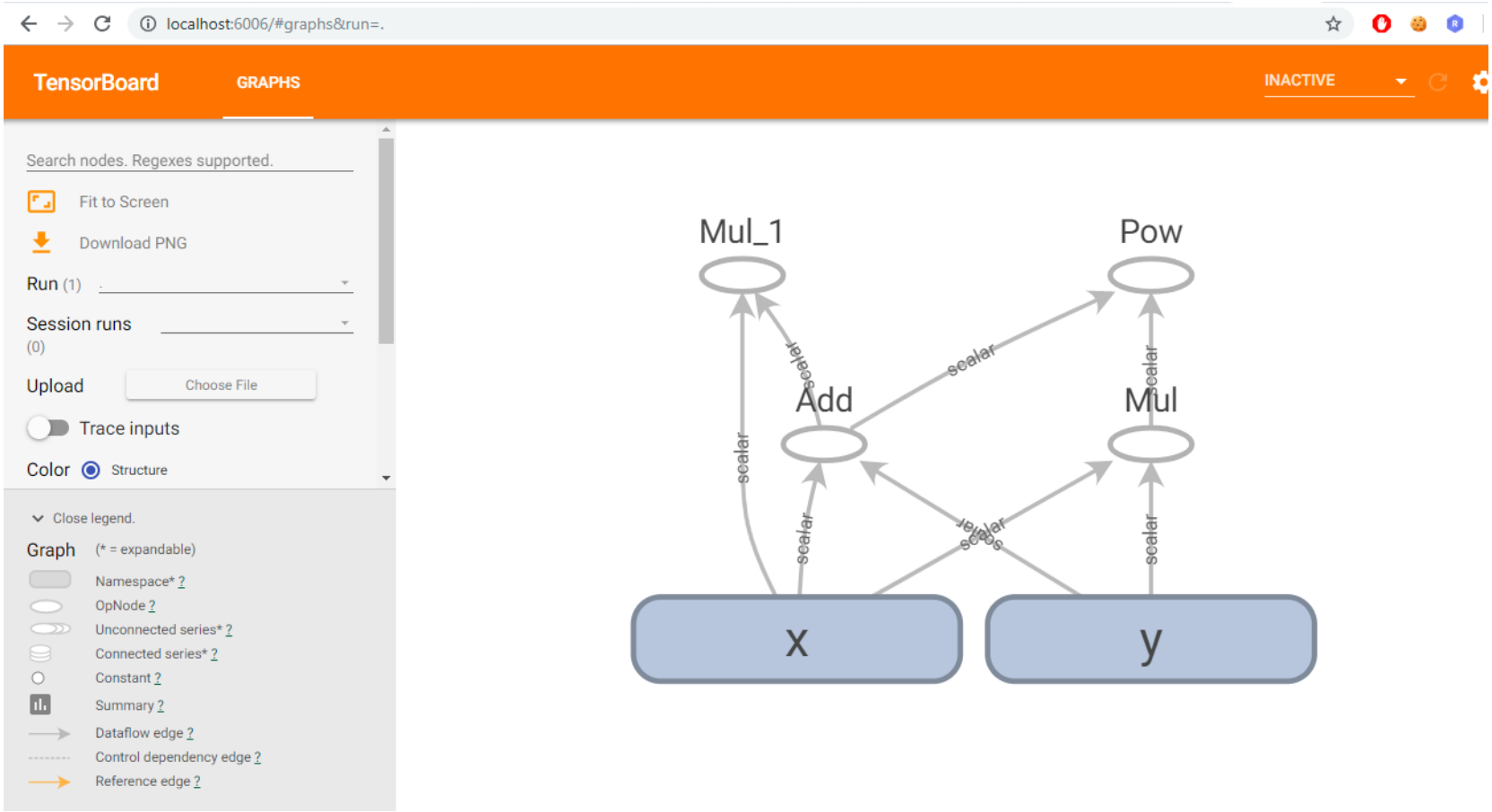
```
import tensorflow as tf
x = tf.Variable(2, name="x")
y = tf.Variable(3, name="y")

add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
useless = tf.multiply(x, add_op)
pow_op = tf.pow(add_op, mul_op)
with tf.Session() as sess:
    # add this line to use TensorBoard.
    sess.run(x.initializer)
    sess.run(y.initializer)
    writer = tf.summary.FileWriter('./graphs', sess.graph)
    z = sess.run(pow_op)
    writer.close() # close the writer when you're done using it
```

TensorBoard: Run it

- Go to terminal, run:
  - $ cd ./project-directory/
  - $ python [yourprogram].py
  - $ tensorboard --logdir="./graphs" --port 6006
- If the command line tensorboard and python don't work, add the enviroment variables:
  - PATH -> Add -> C:\[user-dir]\[python-dir]\
  - PATH -> Add -> C:\[user-dir]\[python-dir]\Scripts\

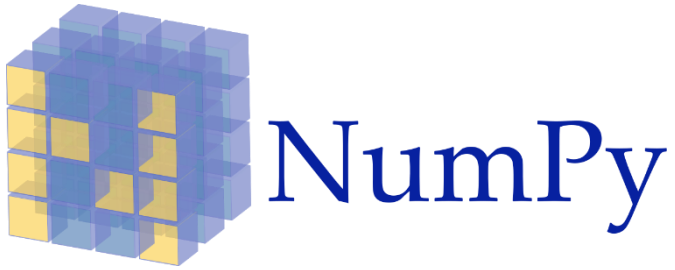
TensorBoard: Visualizing Learning

- Then open your browser and go to: http://localhost:6006/



TensorFlow vs NumPy (1)

- 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
- NumPy doesn't support running on GPUs or TPUs



vs

TensorFlow vs NumPy (2)

- TensorFlow integrates seamlessly with NumPy

```
print(tf.int32 == np.int32) # True
```

- Can pass NumPy types to TensorFlow ops

```
tf.ones([2, 2], np.float32) # ⇒ [[1.0 1.0], [1.0 1.0]]
```