Up and Running with TensorFlow

Alex Rudi 859366

What is TensorFlow?

- open source software library for numerical computation (Deep Learning)
- open-sourced in November 2015
- TensorFlow was developed by the Google Brain team
- used in Google Cloud Speech, Google Photos, and Google Search
- defines in Python a graph of computations
- TensorFlow takes that graph and runs it efficiently using optimized C++ code

TensorFlow

- runs on Windows, Linux, macOS, iOS and Android
- includes highly efficient C++ implementations of many ML operations
- automatically takes care of computing the gradients of the functions (automatic differentiating or autodiff)
- comes with visualization tool called TensorBoard that allows to browse through the computation graph

Installation

```
C:\Users\noob>pip3 install --upgrade tensorflow
Collecting tensorflow
  Downloading https://files.pythonhosted.org/packages/e7/88/417f18ca7eed5ba9
/tensorflow-1.9.0-cp36-cp36m-win_amd64.whl (37.1MB)
    100% |████████████████████████████████| 37.1MB 648kB/s
```

```
C:\Users\noob>pip install --upgrade tensorflow-gpu
Collecting tensorflow-gpu
  Downloading https://files.pythonhosted.org/packages/51/bc/29202147b513f0e
/tensorflow_gpu-1.9.0-cp36-cp36m-win_amd64.whl (103.3MB)
    4% |▌                               | 4.8MB 3.2MB/s eta 0:00:31
```

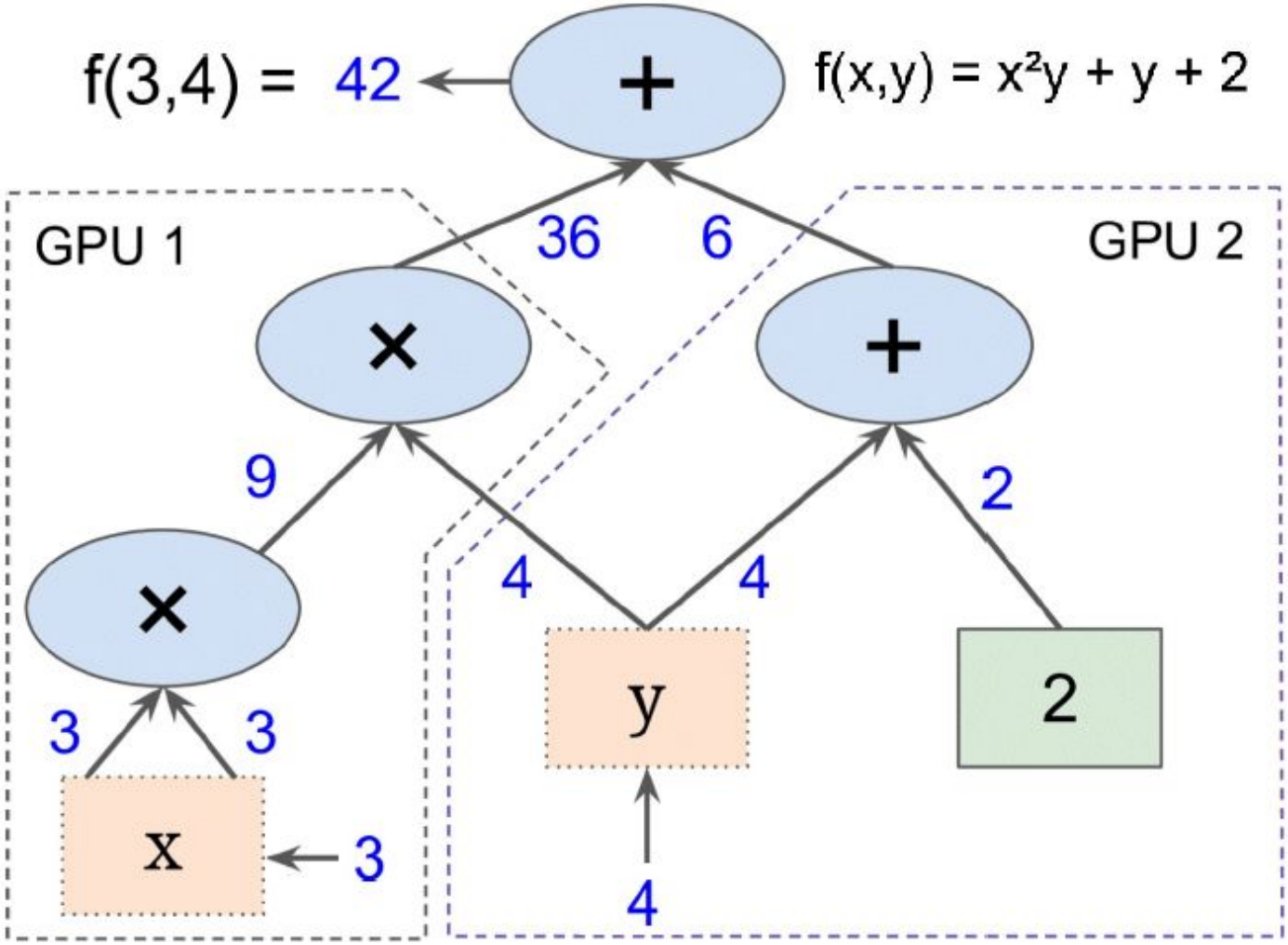
Typical TensorFlow program

- typically split into two parts

- first part (construction phase):
  - builds computation graph representing ML-Model and the computations required to train it

- second part (execution phase):
  - runs a loop that evaluates a training step repeatedly

  - gradually improving the model parameters


f(3,4) = 42
+
f(x,y) = x²y + y + 2
GPU 1
GPU 2
36
6
×
+
9
2
×
4
4
y
2
3
3
x
3
4

First Graph and Running it in a Session

- create the graph

```
import tensorflow as tf

reset_graph()

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

- start computions

```
init = tf.global_variables_initializer()
sess = tf.InteractiveSession()
init.run()
result = f.eval()
sess.close()
print(result)
#42
```

Managing Graphs

- Any node is automatically added to the default graph
- In most cases, OK
- for multiple independent graphs - not good

```
graph = tf.Graph()
with graph.as_default():
    x2 = tf.Variable(2)

x2.graph is graph
#true
x2.graph is tf.get_default_graph()
#false
```

Lifecycle of a Node Value

- TensorFlow automatically determines the set of nodes that it depends on and evaluates these nodes first
- w and x are evaluated twice

```
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
```

- better, to evaluate both y and z in just one graph

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

Using autodiff

- computes the gradients automatically and efficiently
- the gradients() function takes an operation and a list of variables
- creates a list of operations to compute the gradients of the operation with regards to each variable
- So the gradients node will compute the gradient vector of the operation with regards to variables.

Using an Optimizer

- TensorFlow computes the gradients
- it provides a number of optimizers, including a Gradient Descent optimizer

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(mse)
```

- For example, you can use a momentum optimizer

```
optimizer = tf.train.MomentumOptimizer(learning_rate=learning_rate,momentum=0.9)
```

Feeding Data to the Training Algorithm

- replace variables at every iteration
- simplest way to do this, use placeholder nodes
- these nodes don’t perform any computation
- they just output the data at runtime
- typically used to pass the training data to TensorFlow during training
- to create a placeholder node, you must call the placeholder() function
- execution phase: fetch the mini-batches one by one
- then provide the value of variables via the feed_dict parameter when evaluating a node that depends on either of them

Saving and Restoring Models

- TensorFlow makes saving and restoring a model very easy
- create a Saver node at the end of the construction phase

```
save_path = saver.save(sess, "/tmp/my_model_final.ckpt")
```

- at the beginning of the execution phase, instead of initializing the variables using the init node
- call the restore() method of the Saver object

```
with tf.Session() as sess:
    saver.restore(sess, "/tmp/my_model_final.ckpt")
```

- default Saver saves and restores all variables under their own name

Using TensorBoard

- This is very useful tool to identify errors in the graph, to find bottlenecks, and so on
- first step: tweak your program, so it writes the graph definition and some training stats
- create a FileWriter, to write summaries to logfiles in the log directory
- update the execution phase to evaluate the summary node regularly during training
- output a summary-data using the file_writer

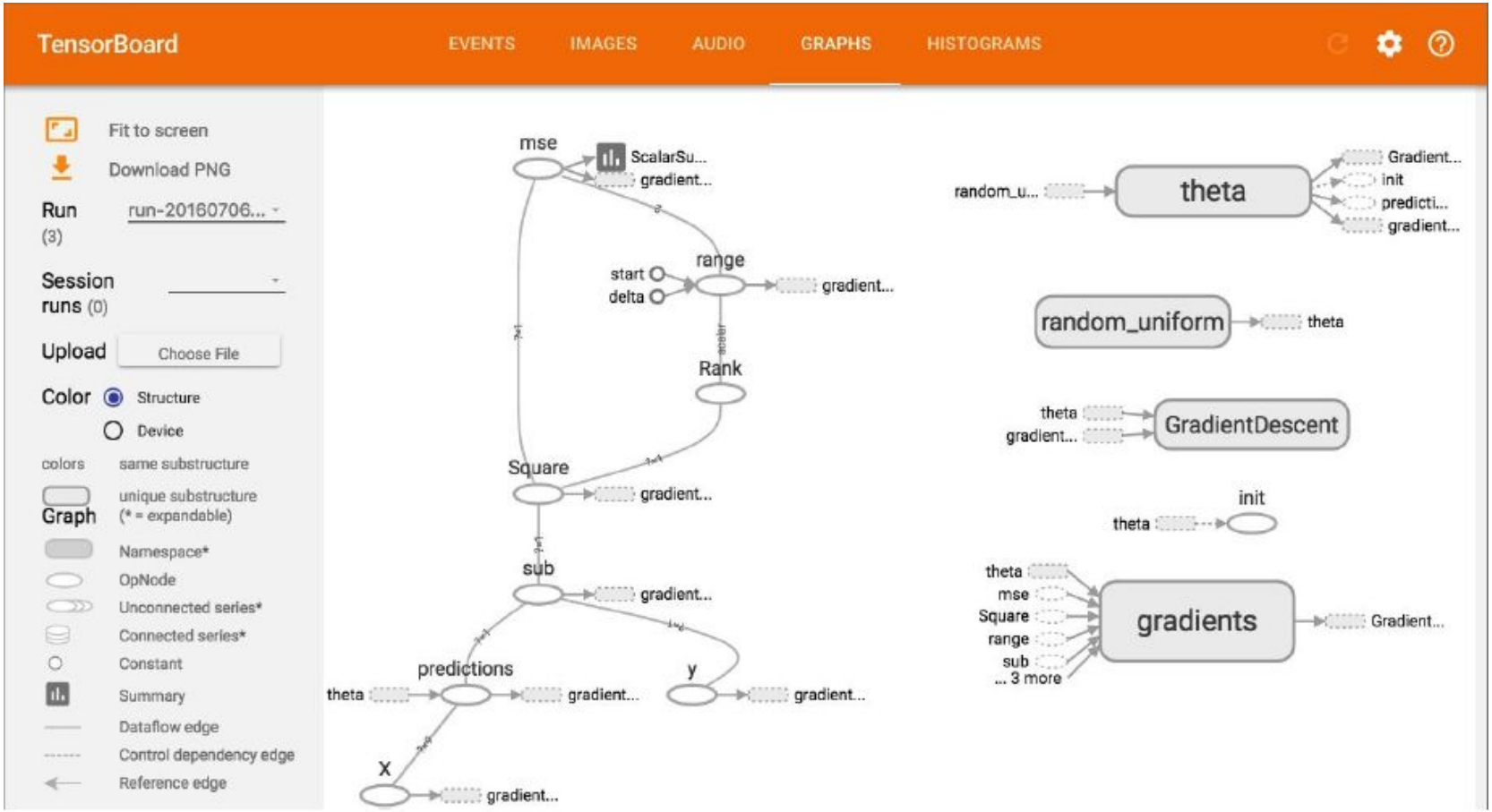
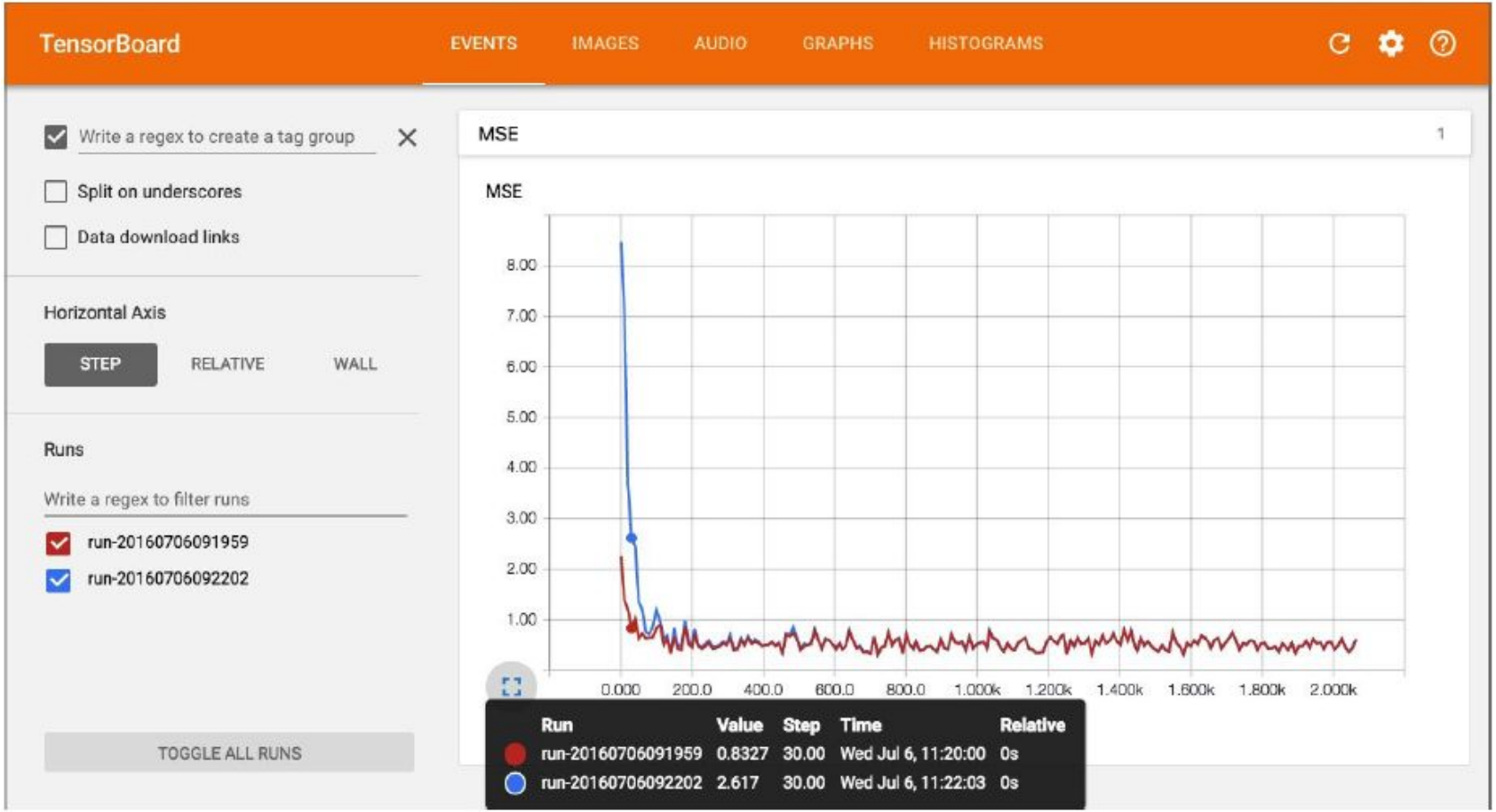


Figure 9-4. Visualizing the graph using TensorBoard


TensorBoard
EVENTS
IMAGES
AUDIO
GRAPHS
HISTOGRAMS
Write a regex to create a tag group
Split on underscores
Data download links
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
run-20160706091959
run-20160706092202
TOGGLE ALL RUNS
MSE
1
MSE
8.00
7.00
6.00
5.00
4.00
3.00
2.00
1.00
0.000
200.0
400.0
600.0
800.0
1.000k
1.200k
1.400k
1.600k
1.800k
2.000k
Run Value Step Time Relative
run-20160706091959 0.8327 30.00 Wed Jul 6, 11:20:00 0s
run-20160706092202 2.617 30.00 Wed Jul 6, 11:22:03 0s

Sources

- Hands-On Machine Learning with Scikit-Learn and Machine Learning Chapter 9
- https://github.com/gorik/d_s_a