

Deep Learning Frameworks

Fabio A. González

[Machine Learning](#)

2019-1



UNIVERSIDAD
NACIONAL
DE COLOMBIA

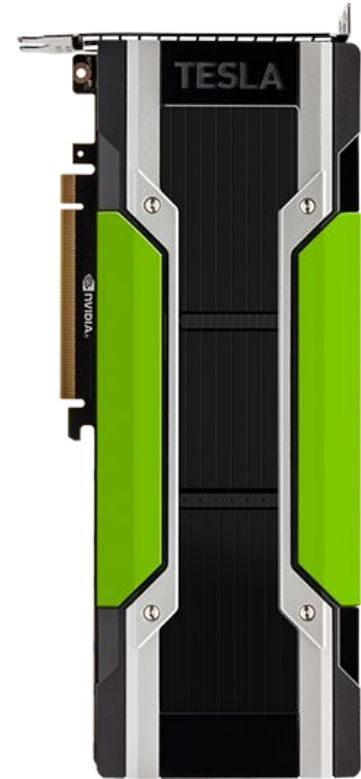
Today's lecture

- ★ Deep learning software overview
- ★ TensorFlow
- ★ Keras
- ★ Practical

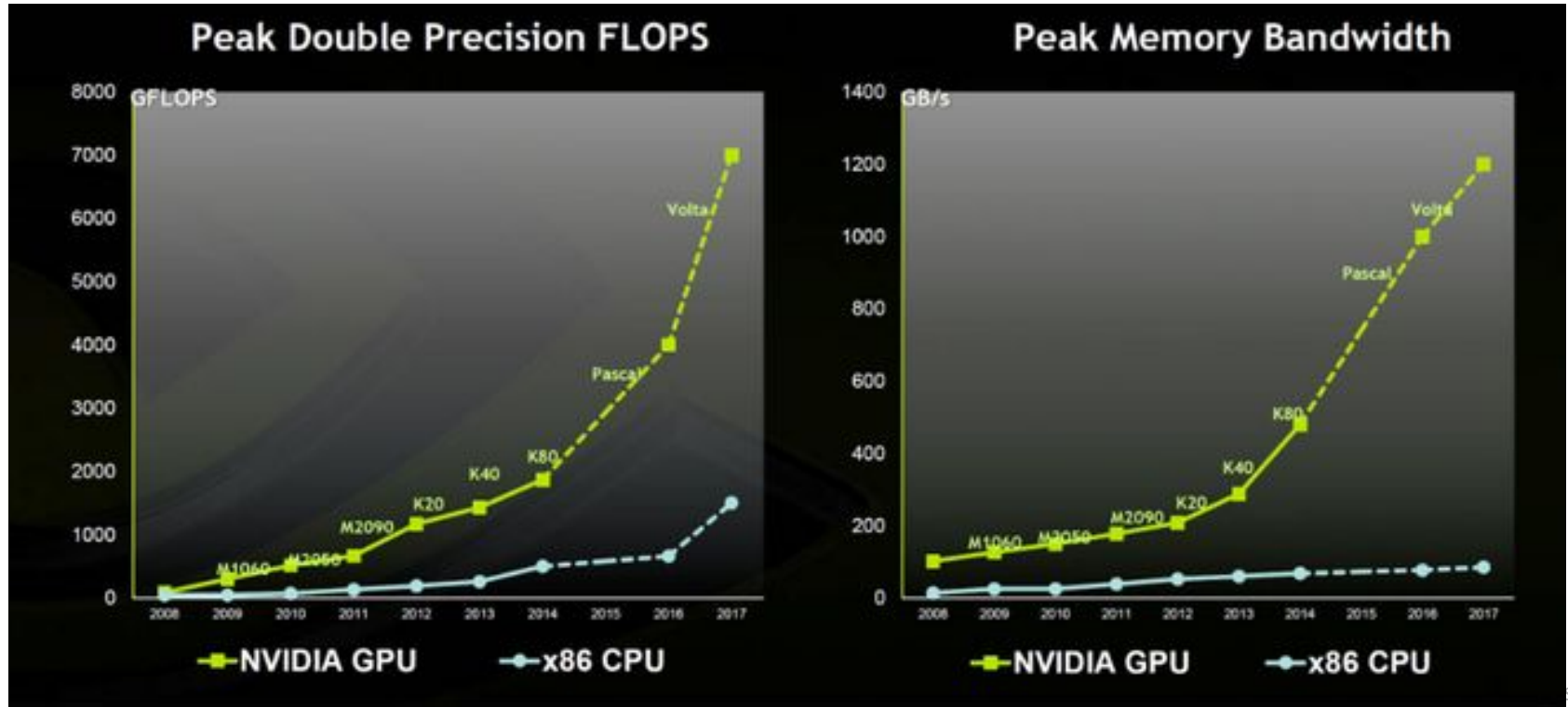


Graphical Processing Unit (GPU)

- ★ From graphical computing to general numerical processing GPGPU
- ★ Single Instruction Multiple Data architecture
- ★ High-throughput type computations with data-parallelism
- ★ Commodity hardware
- ★ Two main vendors: NVidia, AMD



Performance evolution



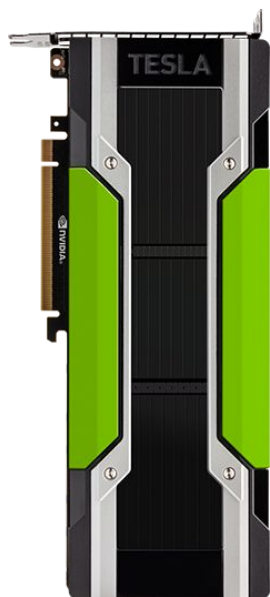
NVIDIA Tesla

PERFORMANCE SPECIFICATIONS FOR NVIDIA TESLA P4, P40 AND P100 ACCELERATORS

	Tesla P4 for Ultra-Efficient Scale-Out Servers	Tesla P40 for Maximum-Inference Throughput Servers	Tesla P100: The Universal Datacenter GPU
Single-Precision Performance (FP32)	5.5 TeraFLOPS	12 TeraFLOPS	10.6 TeraFLOPS
Half-Precision Performance (FP16)	--	--	21 TeraFLOPS
Integer Operations (INT8)	22 TOPS*	47 TOPS*	--
GPU Memory	8 GB	24 GB	16 GB
Memory Bandwidth	192 GB/s	346 GB/s	732 GB/s
System Interface	Low-Profile PCI Express Form Factor	Dual-Slot, Full-Height PCI Express Form Factor	Dual-Slot, Full-Height PCI Express Form Factor, or SXM2 Form Factor with NVLink
Power	50 W/75 W	250 W	250 W (PCIe) 300W (SXM2)
Hardware-Accelerated Video Engine	1x Decode Engine, 2x Encode Engines	1x Decode Engine, 2x Encode Engines	--



GPU vs CPU for deep learning



Deep Learning Inference Throughput

Up to 33X Higher Throughput



CPU Server: Single Xeon E5-2690 v4 @ 2.6GHz | GPU Servers: Single Xeon E5-2690 v4@2.6GHz with 1xP100 (PCIe) | Ubuntu 14.04.5 | TensorRT 2.0 | CUDA 8.0.42 | cuDNN 6.0.5 | NCCL 1.6.1 | Batch size 128 | Precision FP32 (CPU) | FP16 (P100 GPU).

Programming GPUs

★ CUDA:

- NVidia's parallel computing platform
- Access to the GPU's virtual instruction for the execution of compute kernels on the parallel computational elements.
- CUDA C: a specialized version of C (also CUDA Fortran)
- Optimized libraries

★ OpenCL:

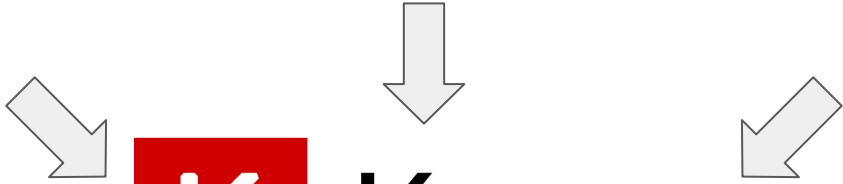
- Similar to CUDA but multiplatform no vendor dependant.
- The way to go with AMD GPU cards.
- A step behind CUDA



Deep learning frameworks



facebook



My Advice:

TensorFlow is a safe bet for most projects. Not perfect but has huge community, wide usage. Maybe pair with high-level wrapper (Keras, Sonnet, etc)

I think **PyTorch** is best for research. However still new, there can be rough patches.

Use **TensorFlow** for one graph over many machines

Consider **Caffe**, **Caffe2**, or **TensorFlow** for production deployment

Consider **TensorFlow** or **Caffe2** for mobile



TensorFlow



UNIVERSIDAD
NACIONAL
DE COLOMBIA

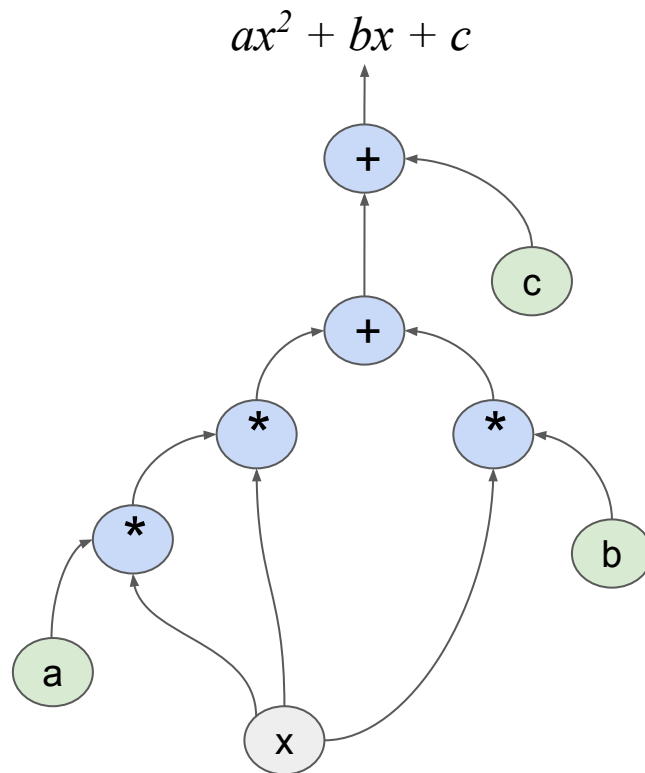
Overview

- ★ Numerical computation based on dataflow graphs
- ★ Developed in C++
- ★ Python and C++ frontends
- ★ Automatic differentiation
- ★ Easy visualization using TensorBoard
- ★ Abstraction layers
 - Tf.contrib.learn, tf.contrib.slim
 - TFLearn, Keras
- ★ Support of heterogeneous architectures: multi-CPU, GPU, multi-GPU, distributed, mobile



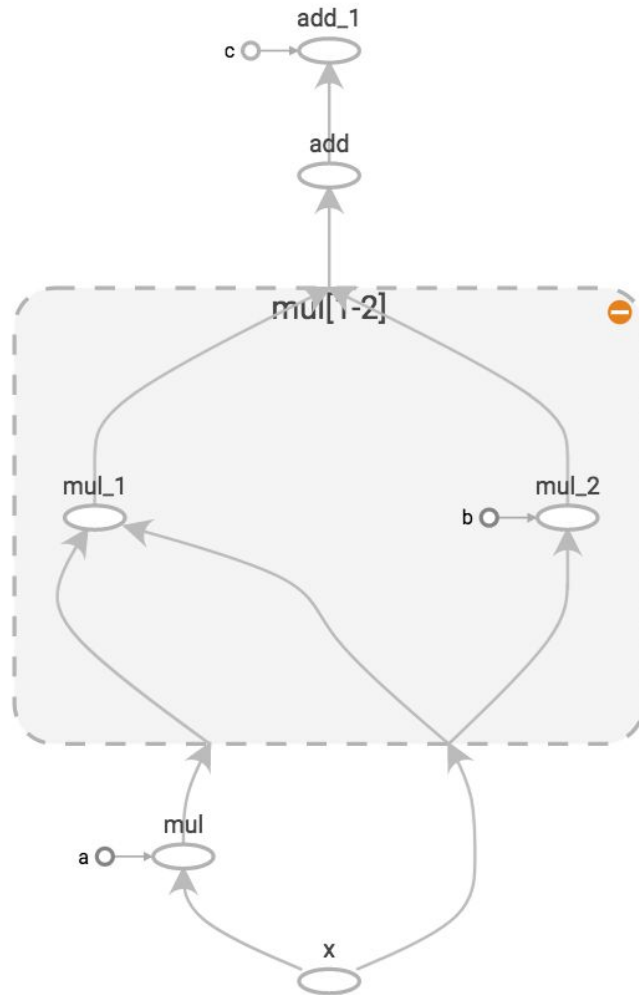
Computation graphs (CG)

- ★ A CG defines the operations that have to be performed over a set of constants and variables.
- ★ TF works over CG where the variables are usually tensors (scalars, vectors, matrices, multidimensional matrices).
- ★ In TF the CG is first created and then it can be executed.
- ★ CG can be symbolically manipulated: e.g. to calculate its gradient or to simplify it.



Creating a graph in TF

```
1 import tensorflow as tf
2
3 graph = tf.Graph()
4
5 with graph.as_default():
6     a = tf.constant(10, tf.float32, name= 'a')
7     b = tf.constant(-5, tf.float32, name= 'b')
8     c = tf.constant(4, tf.float32, name= 'c')
9
10    x = tf.placeholder(tf.float32, name= 'x')
11
12    y = a * x * x + b * x + c
13
14 show_graph(graph.as_graph_def())
```



Executing a graph

- ★ Executing a graph requires to create a session.
- ★ A Session object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.
- ★ Sessions have to be closed so that assigned resources are released

```
1 import tensorflow as tf
2
3 # Graph definition
4 a = tf.constant(10, tf.float32, name= 'a')
5 b = tf.constant(-5, tf.float32, name= 'b')
6 c = tf.constant(4, tf.float32, name= 'c')
7 x = tf.placeholder(tf.float32, name= 'x')
8 y = a * x * x + b * x + c
9
10 #Graph execution
11 sess = tf.Session()
12 result = sess.run(y, {x: 5.0})
13 sess.close()
14
15 print(result)
```



Tensors

- ★ In general, a tensor is a multidimensional array:
 - Vector: one dimensional tensor.
 - Matrix: two dimensional tensor.
- ★ In TF, a tensor is a symbolic handle to one of the outputs of an operation.
- ★ It does not hold the values of that operation's output, but instead provides a means of computing those values in a session.
- ★ The two main attributes of a tensor are its data type and its shape.

```
1 import tensorflow as tf
2
3 a = tf.constant(10)
4 b = tf.constant([1, 2.5, 3])
5 c = tf.constant([[[1, 2], [3, 4]],
6                 [[11, 12], [13, 14]],
7                 [[21, 22], [23, 24]]])
8
9 print(a)
10 print(b)
11 print(c)
```

```
Tensor("Const_24:0", shape=(), dtype=int32)
Tensor("Const_25:0", shape=(3,), dtype=float32)
Tensor("Const_26:0", shape=(3, 2, 2), dtype=int32)
```



Variables and placeholders

- ★ A variable maintains state in the graph across calls to `run()`
- ★ The `Variable()` constructor requires an initial value for the variable, which can be a Tensor of any type and shape.
- ★ The initial value defines the type and shape of the variable.
- ★ Placeholders allow to input values to the graph.
- ★ Placeholder values value must be fed using the `feed_dict` optional argument to `Session.run()`.

```
1 import tensorflow as tf
2
3 import tensorflow as tf
4 2
5 3 # Graph definition
6 4 a = tf.constant(10, tf.float32, name= 'a')
7 5 b = tf.constant(-5, tf.float32, name= 'b')
8 6 c = tf.constant(4, tf.float32, name= 'c')
9 7 x = tf.placeholder(tf.float32, name= 'x')
10 8 y = a * x * x + b * x + c .02)
11 9
12 10 #Graph execution
13 11 sess = tf.Session()
14 12 result = sess.run(y, {x: 5.0})
15 13 sess.close()
16 14
17 15 print(result)
18 16 print(1, val_x, val_y)
19 sess.close()
```



Optimization

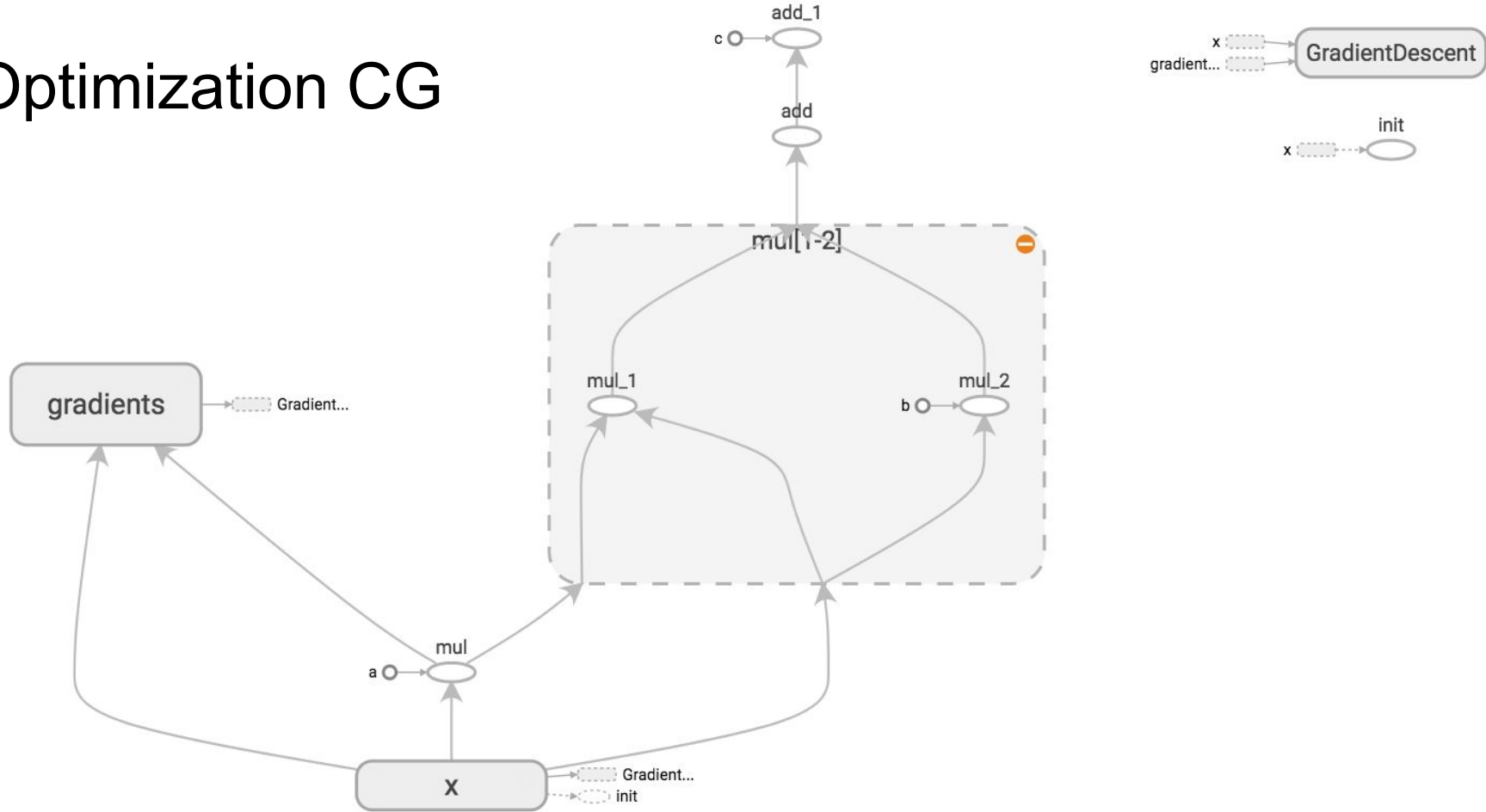
- ★ TF can automatically calculate gradients of a graph.
- ★ You can use the gradients to implement your own optimization strategy,
- ★ or you can use optimization methods already implemented in the system.
- ★ Parameters to optimize must be declared as variables.
- ★ When an optimizer instance is created, it receives parameters such as the learning rate.
- ★ Optimizer must be called with the objective function.
- ★ Variables must be initialized.

```
1 import tensorflow as tf
2
3 # Graph definition
4 a = tf.constant(10, tf.float32, name= 'a')
5 b = tf.constant(-5, tf.float32, name= 'b')
6 c = tf.constant(4, tf.float32, name= 'c')
7 x = tf.Variable(0.0, name= 'x')
8 y = a * x * x + b * x + c
9
10 optimizer = tf.train.GradientDescentOptimizer(0.02)
11 update = optimizer.minimize(y)
12
13 # Graph execution
14 sess = tf.Session()
15 sess.run(tf.global_variables_initializer())
16 for i in range(20):
17     val_y, val_x, _ = sess.run([y, x, update])
18     print(i, val_x, val_y)
19 sess.close()
```

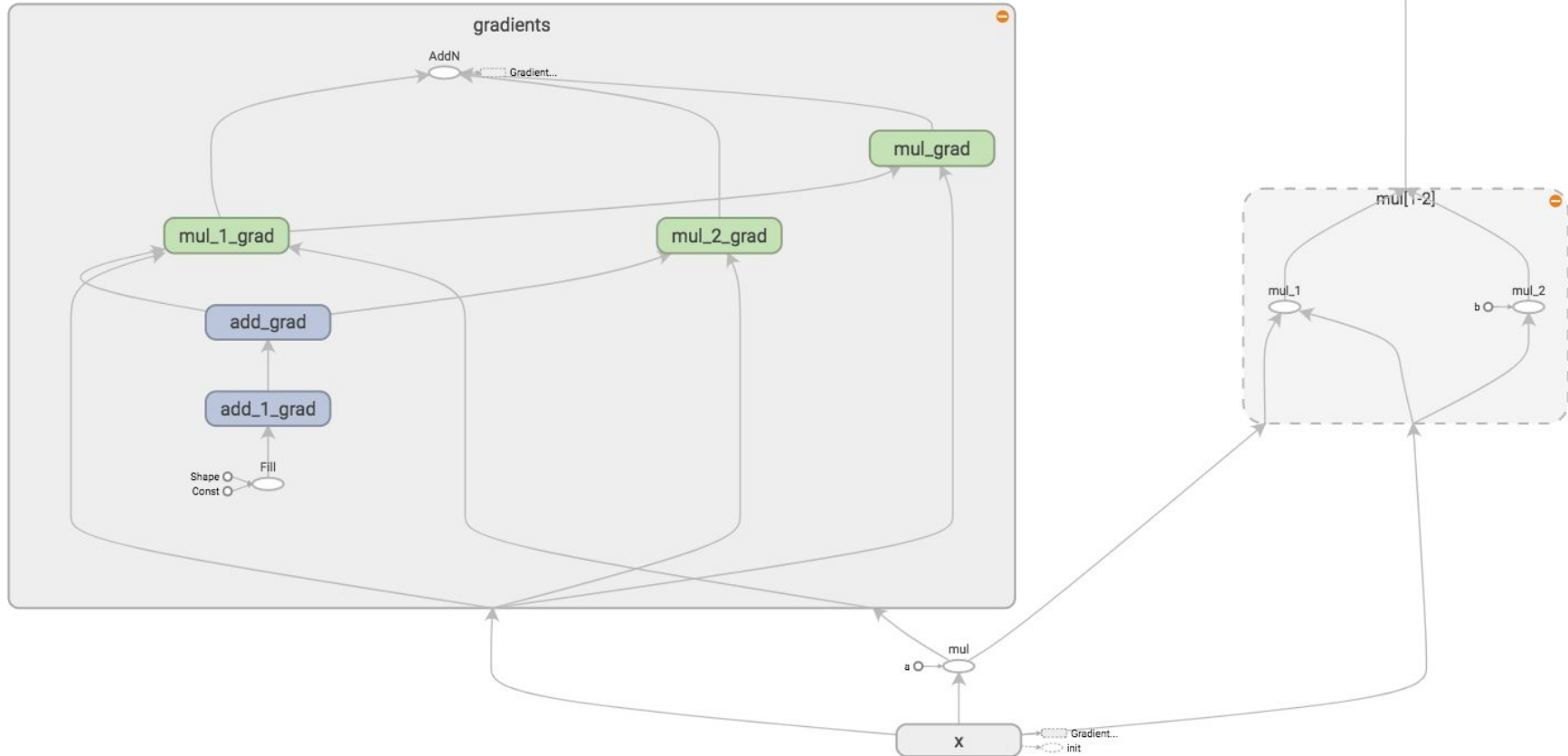
```
0 0.1 4.0
1 0.16 3.6
2 0.196 3.456
3 0.2176 3.40416
4 0.23056 3.3855
5 0.238336 3.37878
6 0.243002 3.37636
7 0.245801 3.37549
```



Optimization CG



Optimization CG



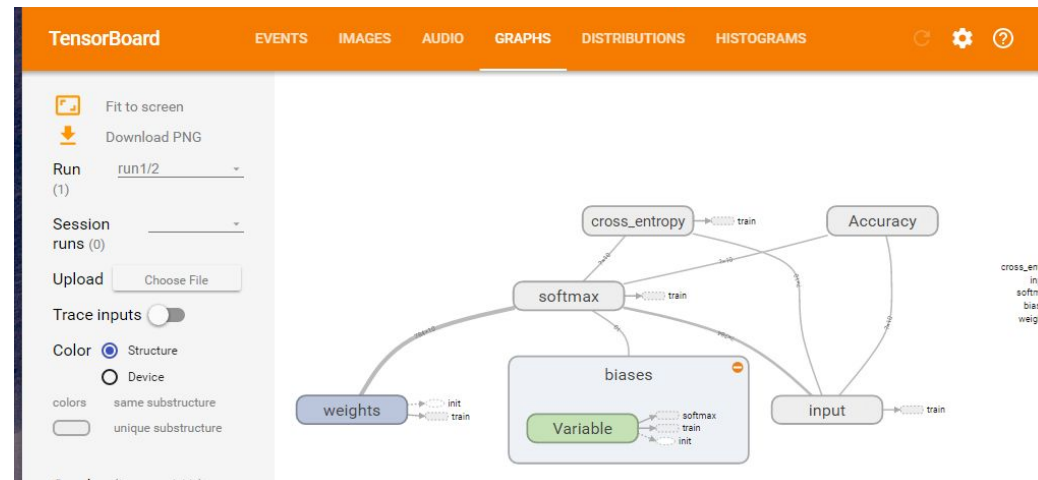
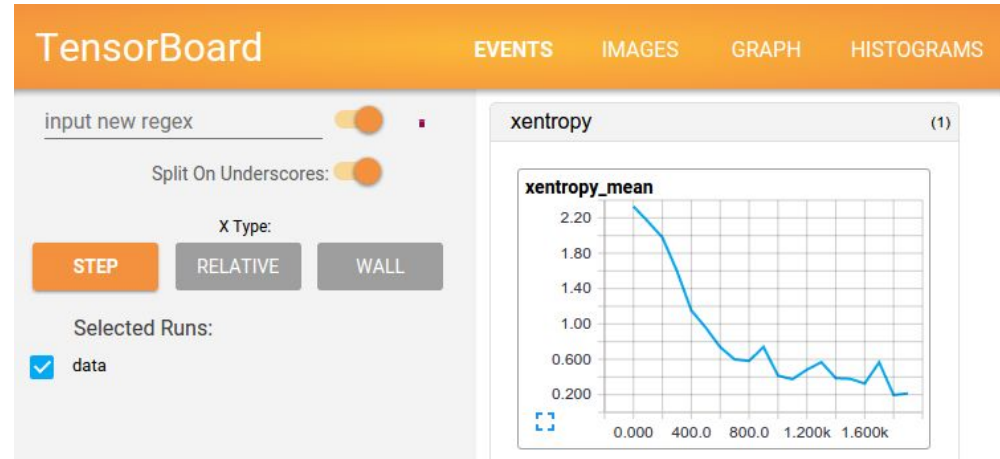
Optimizers

- ★ `tf.train.GradientDescentOptimizer`
- ★ `tf.train.AdadeltaOptimizer`
- ★ `tf.train.AdagradOptimizer`
- ★ `tf.train.MomentumOptimizer`
- ★ `tf.train.AdamOptimizer`
- ★ `tf.train.FtrlOptimizer`
- ★ `tf.train.ProximalGradientDescentOptimizer`
- ★ `tf.train.ProximalAdagradOptimizer`
- ★ `tf.train.RMSPropOptimizer`



Monitoring with TensorBoard

- ★ TensorBoard is a visualization application provided by TensorFlow.
- ★ It visualizes summary data which is written to log files during training.
- ★ It also visualizes the computing graph as well as complementary information such as images.



Devices

- ★ TF supports different target devices: CPU, GPU, multi GPU
- ★ A graph me be distributed among different devices
- ★ TF take care of consolidating the data

```
# Creates a graph.
c = []
for d in ['/gpu:2', '/gpu:3']:
    with tf.device(d):
        a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3])
        b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2])
        c.append(tf.matmul(a, b))
with tf.device('/cpu:0'):
    sum = tf.add_n(c)
# Creates a session with log_device_placement set to True.
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(sum))
```



Additional topics

- ★ **Estimators**: a high-level TF API that greatly simplifies machine learning programming. Encapsulates main ML tasks: training, evaluation, prediction.
- ★ **Saving and loading models**: TF provides different tools to persists trained models.
- ★ **Dataset API**: makes it easy to deal with large amounts of data, different data formats, and complicated transformations.
- ★ **tf.layers**: provides a high-level API that makes it easy to construct a neural network. It provides methods that facilitate the creation of dense (fully connected) layers and convolutional layers.
- ★ **tf.nn**: Neural network support.
- ★ **tf.contrib**: contains volatile or experimental code.

○

TensorFlow Demo





Keras

- ★ Developed by François Chollet
- ★ High-level Python framework able to run on top of TensorFlow, Theano or CNTK,
- ★ Guiding principles:
 - User friendliness
 - Modularity
 - Easy extensibility
 - Work with Python
- ★ Highly popular
- ★ Fast prototyping
- ★ Easy to extend
- ★ Many pretrained models



Sequential model

```
from keras.models import Sequential
```

```
model = Sequential()
```

```
from keras.layers import Dense, Activation
```

```
model.add(Dense(units=64, input_dim=100))  
model.add(Activation('relu'))  
model.add(Dense(units=10))  
model.add(Activation('softmax'))
```

```
model.compile(loss='categorical_crossentropy',  
              optimizer='sgd',  
              metrics=['accuracy'])
```

```
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

```
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
```

```
classes = model.predict(x_test, batch_size=128)
```

- ★ The simplest model is sequential
- ★ Layers are stacked one above the other
- ★ The learning process is configured with compile
- ★ Training is performed with one line.
- ★ The trained model can be easily evaluated
- ★ And applied to new data



The functional API

- ★ The sequential model is easy to use, but somewhat restricted.
- ★ The functional API gives more flexibility that allows to construct more complex models:
 - Multiple outputs (multi-task)
 - Multi inputs
 - Shared layers

Layers

- ★ Layers are the building blocks of models
- ★ Keras provides several predefined layers for building different types of networks
- ★ Layers have different methods that allow to get and set their weights, to define an initialization function, to control the regularization, the activation function etc.



Preprocessing

★ Sequences:

- Pad_sequences
- Skip-grams

★ Text

- Text to word sequence
- One hot
- Hashing
- Tokenizer

★ Images

- Normalization
- Data augmentation



Keras Demo

