TensorFlow 101

Deep Learning PhD Course
2017/2018

Marco Ciccone

Dipartimento di Informatica Elettronica e Bioingegneria
Politecnico di Milano

About me

Present

- **2nd year PhD student** in Deep Learning
- supervised by Prof. Matteo Matteucci

Background

Machine Learning, Signal Processing

- MSc at Politecnico di Milano
- BSc at Università degli studi di Firenze

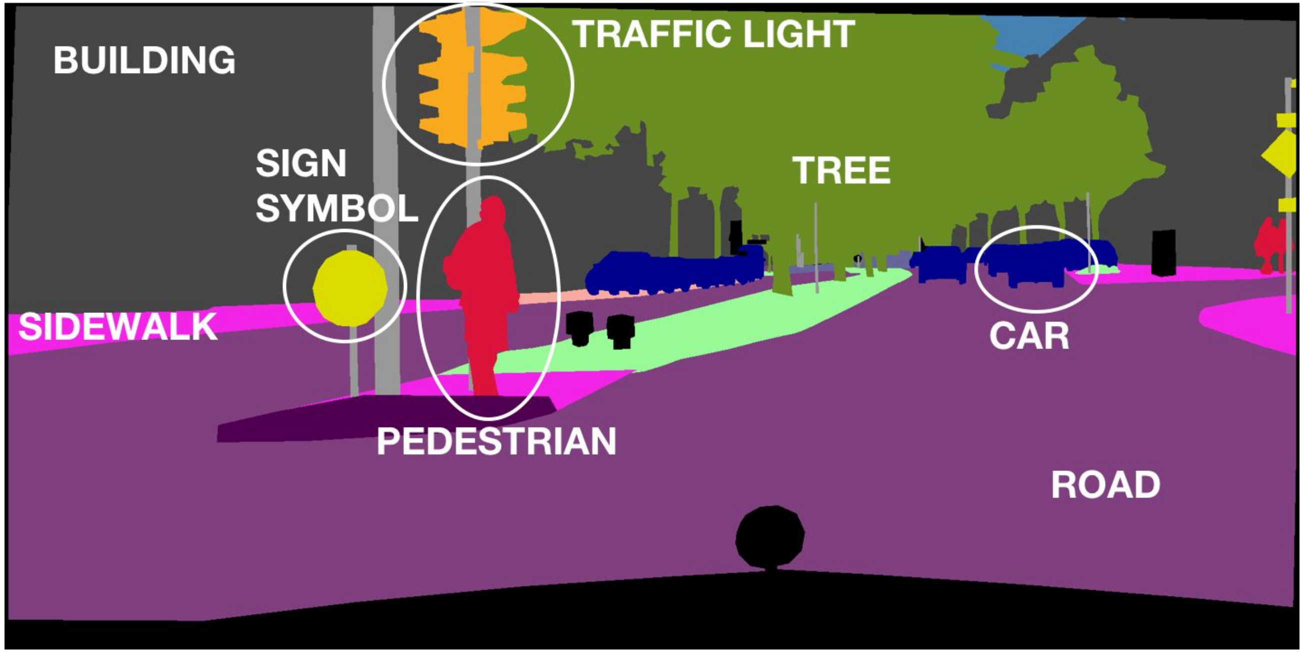
Contacts

marco.ciccone@polimi.it

Semantic Segmentation

From Cityscapes Dataset

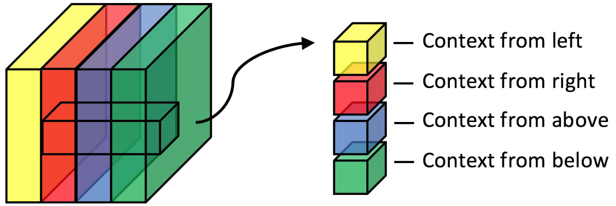
Structure Prediction Task: assign to each pixel of the image a semantic class

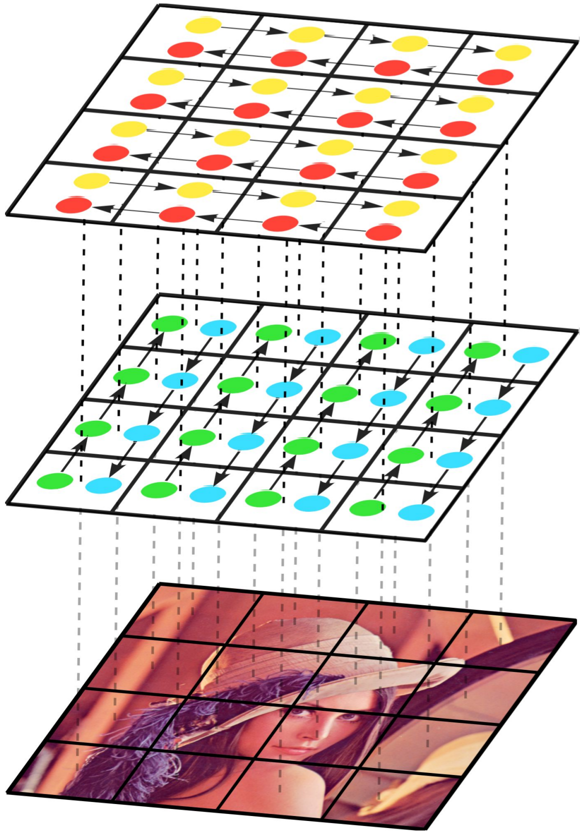


ReNet and ReSeg

4 RNNs that scans the pixels of the image in different directions:

- Top-Down + Bottom-Up
- Left-Right + Right-Left





Reference:

- [ReNet, Visin et Al.](#)
- [ReSeg, Visin, Ciccone et Al.](#) (M.Sc. Thesis)
- Code: https://github.com/fvisin/reseg

If you have questions regarding the course/projects drop me an email with **[PHD_DL2018]** in the subject

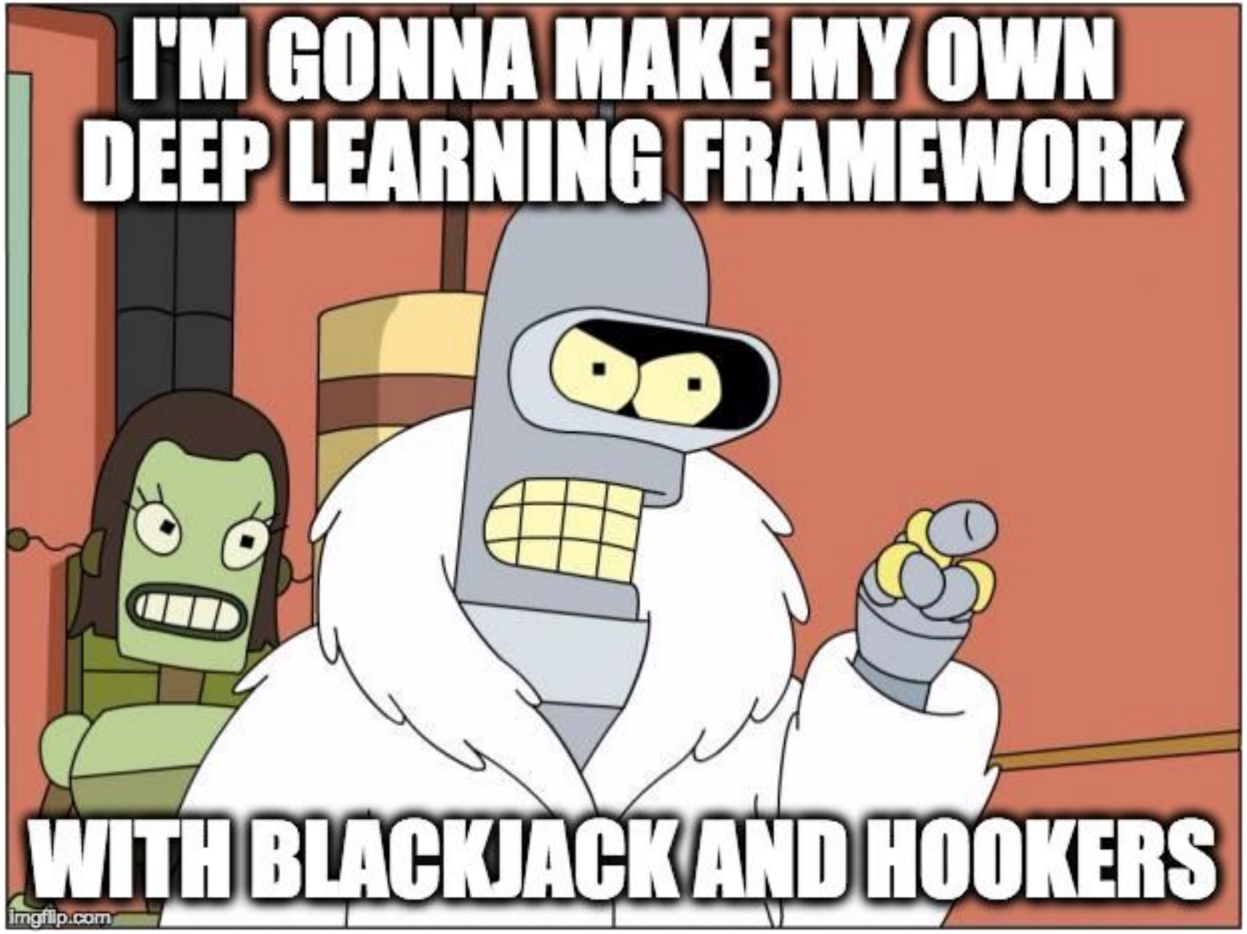
```
if '[PHD_DL2018]' in mail.subject:
    read email
else:
    ignore email
```

Let’s start :-)

DL frameworks

- Theano (Python)
- Caffe(2) (C, C++, Python, MATLAB, Command line)
- Torch (Lua, C, C++)
- MXNet (Python, R, C++, Julia)
- **PyTorch (Python, C, C++)**
- **Tensorflow (C++, Python)**

All of these frameworks have an interface (scripting) language to prototype faster and several **backend** depending on the device you use to train/deploy


I'M GONNA MAKE MY OWN DEEP LEARNING FRAMEWORK
WITH BLACKJACK AND HOOKERS
imgflip.com

Which framework should I use?

There's no winner.

It really depends what you like and what you have to do.

ONNX: open neural network exchange format

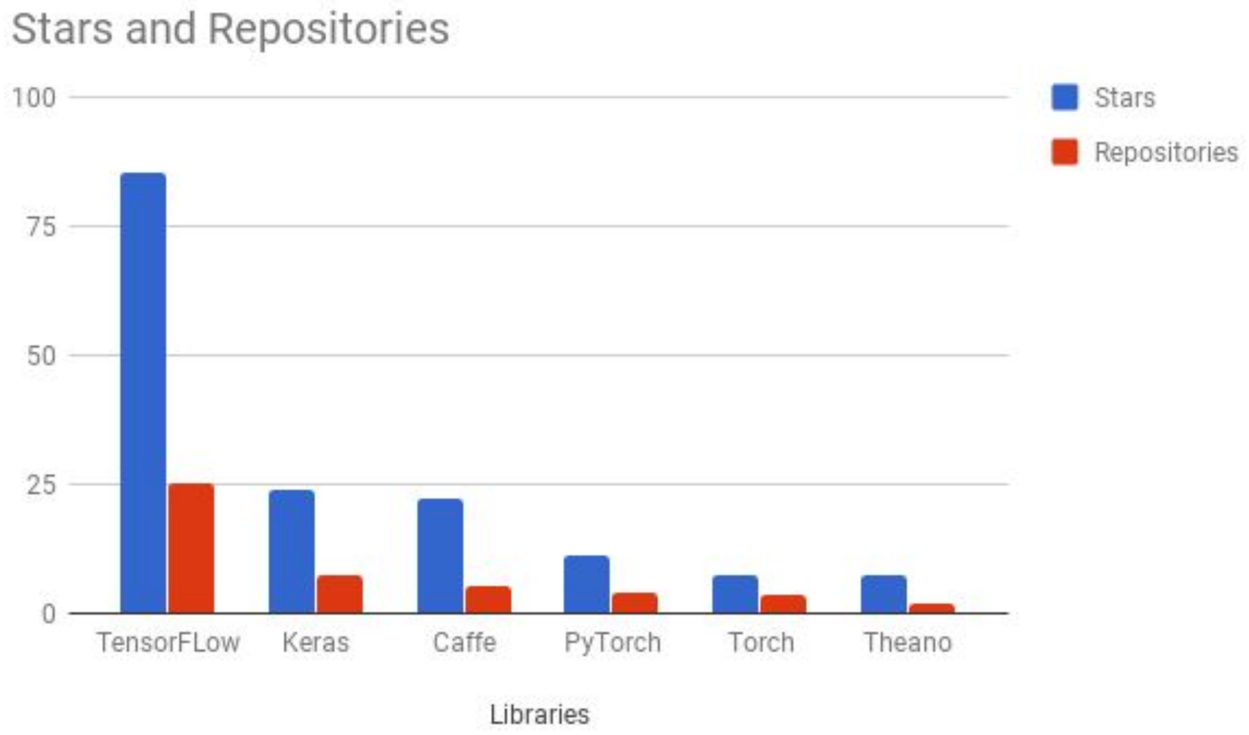
http://onnx.ai/

Train with X and deploy with Y.

CUDNN Disclaimer

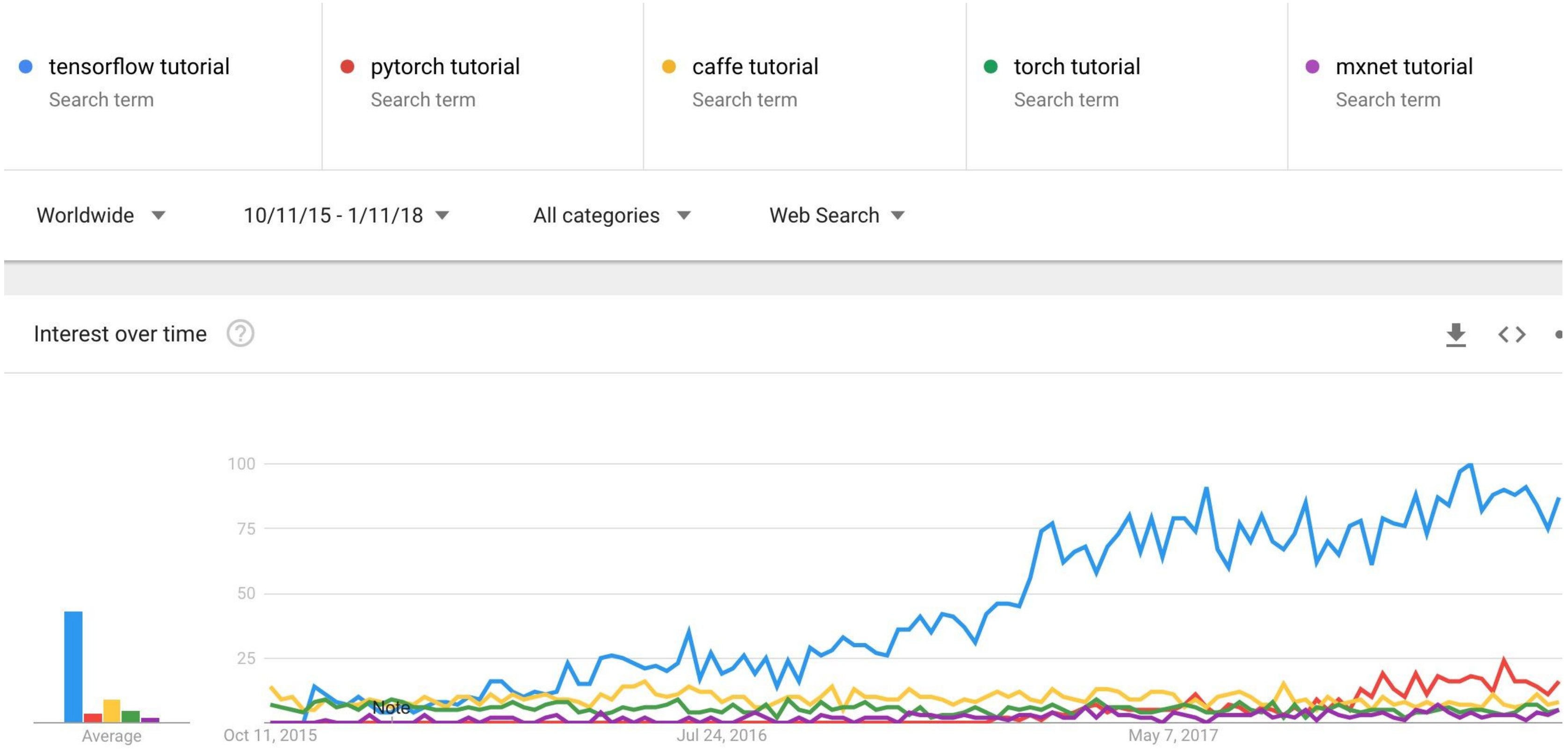
Most likely for your project you will need GPU(s) and
***CUDNN** backend for GPU acceleration.*
(optimized kernels directly provided by NVIDIA)

Github Distribution



Slide from *“Stanford TensorFlow for DL Research course”*

Demand for TensorFlow learning materials


● tensorflow tutorial
Search term

● pytorch tutorial
Search term

● caffe tutorial
Search term

● torch tutorial
Search term

● mxnet tutorial
Search term

Worldwide ▾ 10/11/15 - 1/11/18 ▾ All categories ▾ Web Search ▾

Interest over time

Slide from *"Stanford TensorFlow for DL Research course"*

TF Levels

- Basic (Keras)
- **Intermediate (custom modules)**
- Advanced (Data parallelism on Multiple Devices)
- Pro (Distributed parallelism)
- Inferno (Subgraphs on different devices)

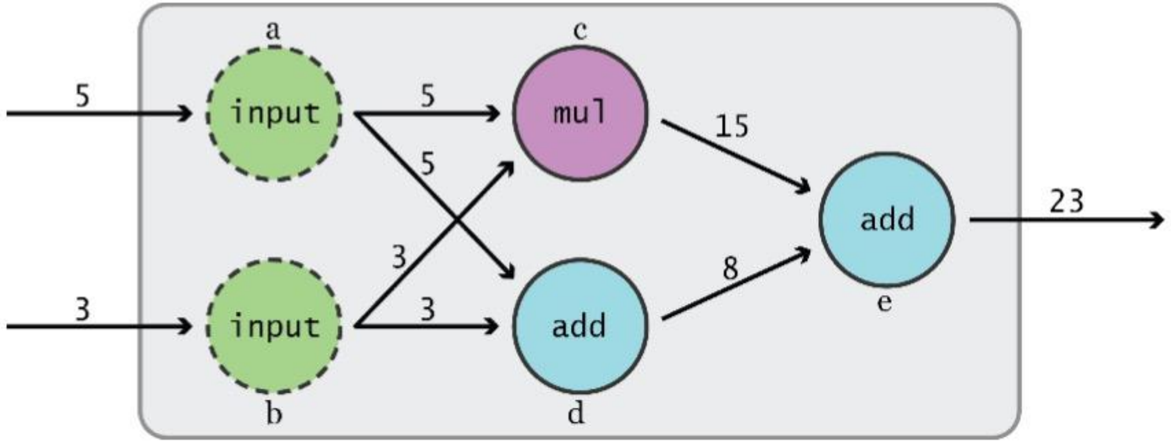
Cool, but what is TensorFlow?

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

It's not specific for Deep Learning, but it's tightly coupled with it. In principle you can use it for any tensor operation.

Graph Computation

TensorFlow decouples definition of computations from execution



Graph from *TensorFlow for Machine Intelligence and "Stanford TensorFlow for DL Research course"*

Graph Declaration

TF is ***NOT imperative*** (à la numpy) [not entirely true now… see Eager Mode]

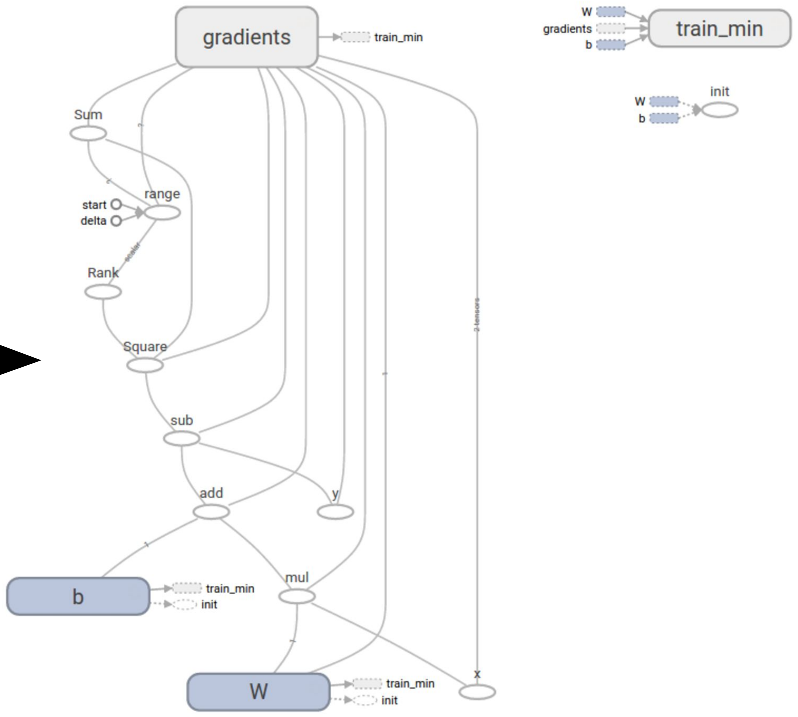
1. Define a graph of operations **(Code)**
2. Graph is built and optimized by TF **(Pray)**
3. Execute operations and feed the graph with actual data through tf.session() **(Wait & Hope for results)**

Pros and Cons:

+ The graph of operations allows to compute the gradient automagically without the need specify (code) the gradient of the operation (*Automatic Differentiation tool*).
+ Thanks to optimization techniques the resulting graph could be really fast and memory efficient.
- Debugging is a true nightmare if you don't have enough experience: you'll become *"the debbbaggher".*

Graph Declaration

```python
import numpy as np
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], tf.float32)
b = tf.Variable([-.3], tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W * x + b
y = tf.placeholder(tf.float32)
# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
# training data
x_train = [1,2,3,4]
y_train = [0,-1,-2,-3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
  sess.run(train, {x:x_train, y:y_train})

# evaluate training accuracy
curr_W, curr_b, curr_loss  = sess.run([W, b, loss], {x:x_train, y:y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
```



Slide from *“Stanford TensorFlow for DL Research course”*

Graph Pros

Optimization

- Automatic buffer reuse
- Constant folding
- Inter-op parallelism
- Automatic trade-off between compute and memory

Deployability

- Graph is an intermediate representation for models

Rewritable

- Experiment with automatic device placement or quantization

Graph Cons

Difficult to debug

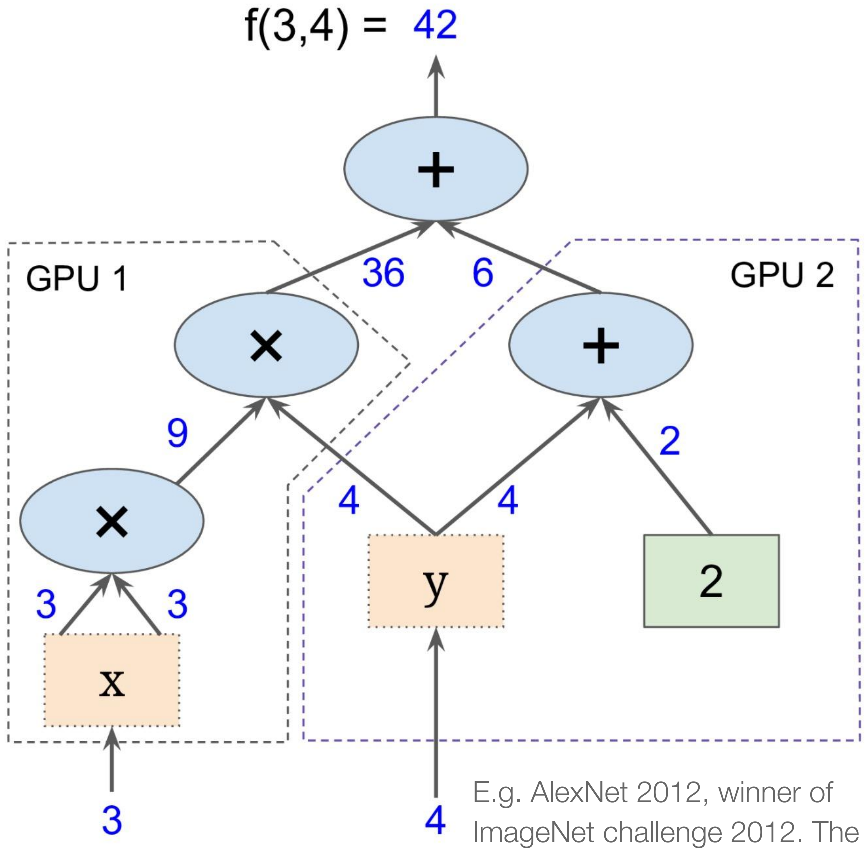
- Errors are reported long after graph construction
- Execution cannot be debugged with pdb or print statements

Un-Pythonic

- Writing a TensorFlow program is an exercise in metaprogramming
- Control flow (e.g., tf.while_loop) differs from Python
- Can't easily mix graph construction with custom data structures

Again… Why graphs?

1. **Save computation.** Only run subgraphs that lead to the values you want to fetch.
2. Break computation into small, differential pieces to facilitate **auto-differentiation.**
3. **Facilitate distributed computation.** Spread the work across multiple CPUs, GPUs, TPUs, or other devices
4. Many common machine learning models are taught and visualized as **directed graphs.**



E.g. AlexNet 2012, winner of ImageNet challenge 2012. The model was split in 2 GPUs to be able to train it.

Slide from *"Stanford TensorFlow for DL Research course"*

Before starting: Tensors

Tensors are just **n-dimensional** arrays

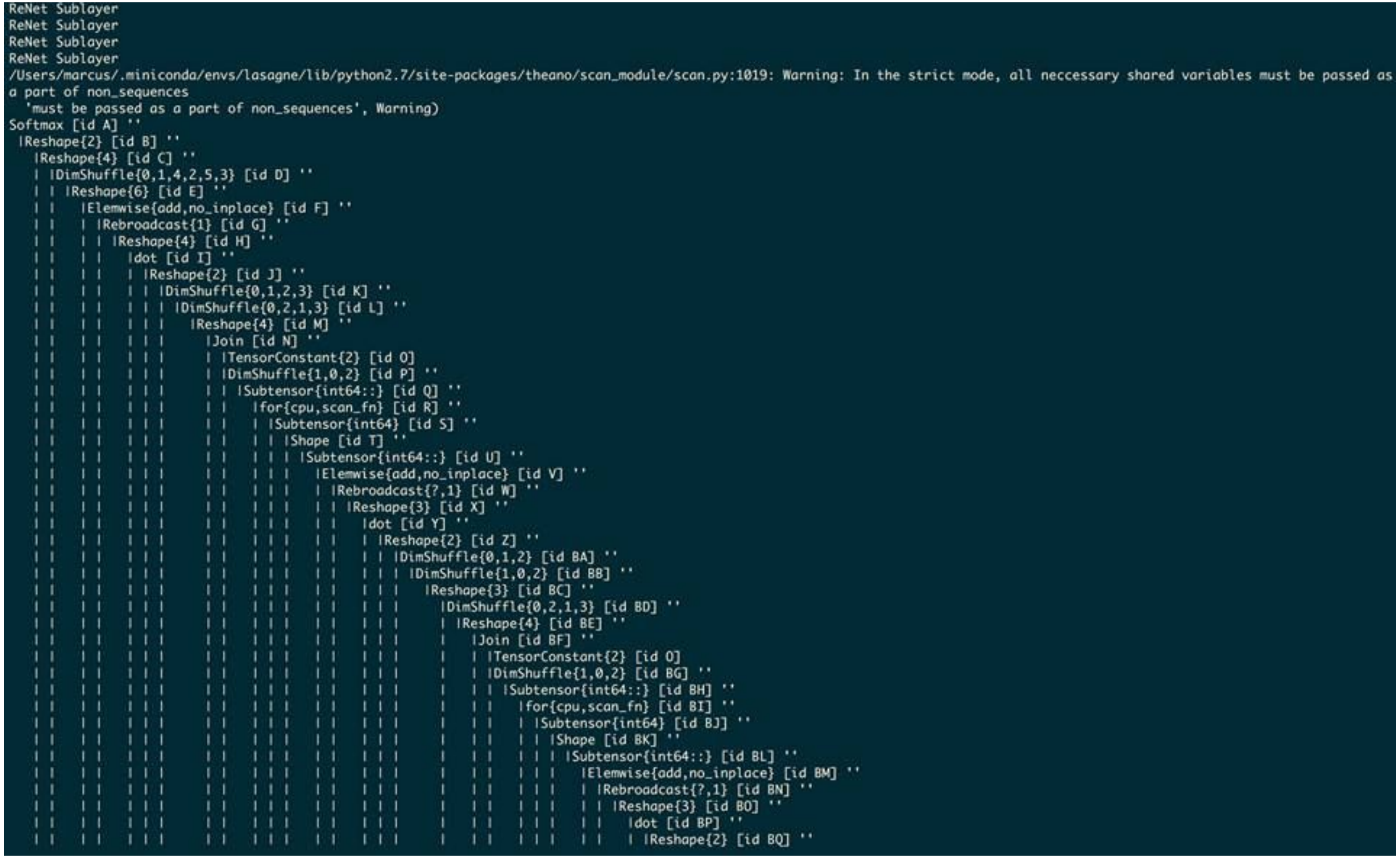
- 0-d tensor: **scalar** (number)
- 1-d tensor: **vector**
- 2-d tensor: **matrix**
- And so on …

Note that in TF you are dealing with batch of data so for instance:

- Image are 4D	batch x height x width x nchannels
- Sequences are 3D	batch x sequence_lenght x nfeatures

Shapes Disclaimer

With tensors computation, 99% of your bugs are going to be on shapes, so deal with it.

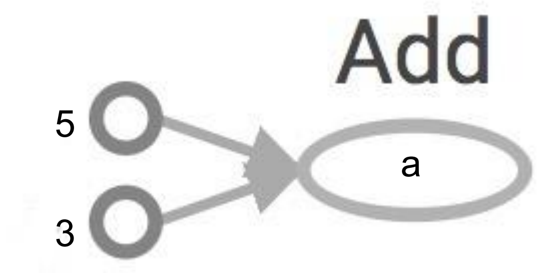


```
ReNet Sublayer
ReNet Sublayer
ReNet Sublayer
ReNet Sublayer
/Users/marcus/.miniconda/envs/lasagne/lib/python2.7/site-packages/theano/scan_module/scan.py:1019: Warning: In the strict mode, all neccessary shared variables must be passed as
a part of non_sequences
  'must be passed as a part of non_sequences', Warning)
Softmax [id A] ''
 |Reshape{2} [id B] ''
   |Reshape{4} [id C] ''
   | |DimShuffle{0,1,4,2,5,3} [id D] ''
   | | |Reshape{6} [id E] ''
   | |   |Elemwise{add,no_inplace} [id F] ''
   | |   | |Rebroadcast{1} [id G] ''
   | |   | | |Reshape{4} [id H] ''
   | |   | |   |dot [id I] ''
   | |   | |   | |Reshape{2} [id J] ''
   | |   | |   | | |DimShuffle{0,1,2,3} [id K] ''
   | |   | |   | | | |DimShuffle{0,2,1,3} [id L] ''
   | |   | |   | | |   |Reshape{4} [id M] ''
   | |   | |   | | |     |Join [id N] ''
   | |   | |   | | |     | |TensorConstant{2} [id O]
   | |   | |   | | |     | |DimShuffle{1,0,2} [id P] ''
   | |   | |   | | |     | | |Subtensor{int64::} [id Q] ''
   | |   | |   | | |     | |   |for{cpu,scan_fn} [id R] ''
   | |   | |   | | |     | |   | |Subtensor{int64} [id S] ''
   | |   | |   | | |     | |   | | |Shape [id T] ''
   | |   | |   | | |     | |   | | | |Subtensor{int64::} [id U] ''
   | |   | |   | | |     | |   | | |   |Elemwise{add,no_inplace} [id V] ''
   | |   | |   | | |     | |   | | |   | |Rebroadcast{?,1} [id W] ''
   | |   | |   | | |     | |   | | |   | | |Reshape{3} [id X] ''
   | |   | |   | | |     | |   | | |   | |   |dot [id Y] ''
   | |   | |   | | |     | |   | | |   | |   | |Reshape{2} [id Z] ''
   | |   | |   | | |     | |   | | |   | |   | | |DimShuffle{0,1,2} [id BA] ''
   | |   | |   | | |     | |   | | |   | |   | | | |DimShuffle{1,0,2} [id BB] ''
   | |   | |   | | |     | |   | | |   | |   | | |   |Reshape{3} [id BC] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |DimShuffle{0,2,1,3} [id BD] ''
   | |   | |   | | |     | |   | | |   | |   | | |     | |Reshape{4} [id BE] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   |Join [id BF] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |TensorConstant{2} [id O]
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |DimShuffle{1,0,2} [id BG] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | | |Subtensor{int64::} [id BH] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |   |for{cpu,scan_fn} [id BI] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |   | |Subtensor{int64} [id BJ] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |   | | |Shape [id BK] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |   | | | |Subtensor{int64::} [id BL] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |   | | |   |Elemwise{add,no_inplace} [id BM] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |   | | |   | |Rebroadcast{?,1} [id BN] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |   | | |   | | |Reshape{3} [id BO] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |   | | |   | |   |dot [id BP] ''
   | |   | |   | | |     | |   | | |   | |   | | |     |   | |   | | |   | |   | |Reshape{2} [id BQ] ''
```

“Desperation on terminal”, Ciccone November 2015

Hello World TF

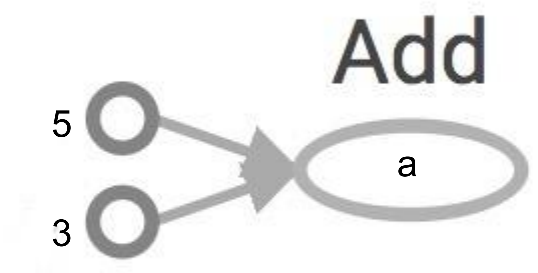
```
import tensorflow as tf

a = tf.add(3, 5)

print(a)
```



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

How to get the value of ``a``?

- Create a session,
- assign it to variable sess so we can call it later
- Within the session, evaluate the graph to fetch th value of a



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

tf.Session()

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

- Session will also **allocate memory** to store the current values of variables.

tf.Graph()

to add operators to a graph, set it as default:

```
g = tf.Graph()
with g.as_default():
    x = tf.add(3, 5)
with tf.Session(graph=g) as sess:
    sess.run(x)
```

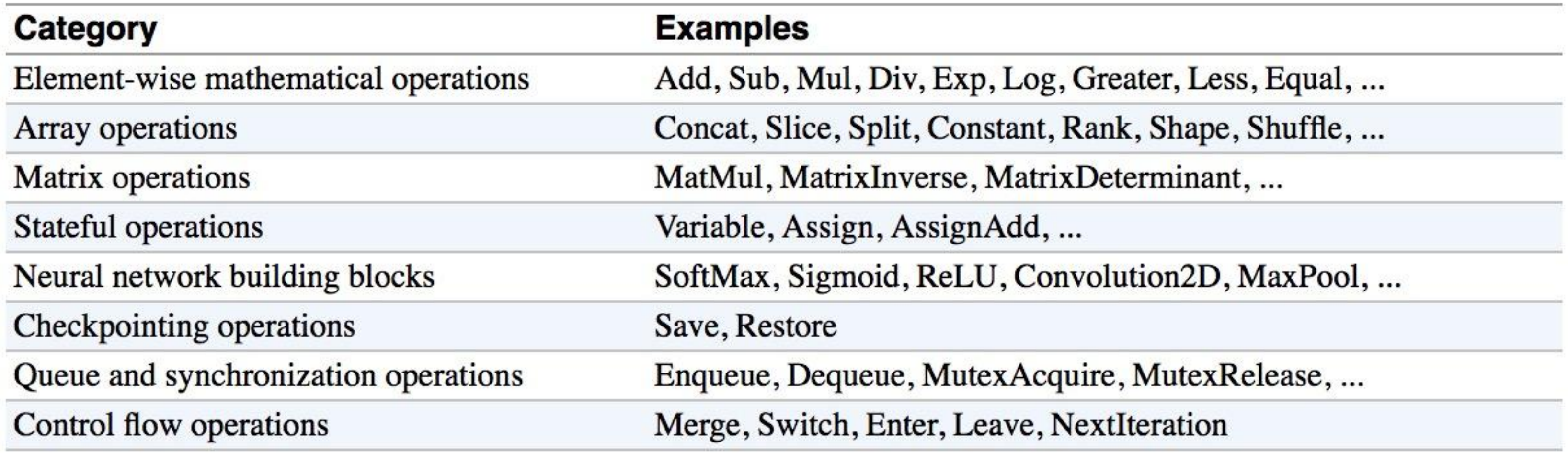
Warnings!

- DO NOT mess with graphs!
- DO NOT use more than one graph per session!

to handle the default graph:

```
g = tf.get_default_graph()
```

TF 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

Slide from *“Stanford TensorFlow for DL Research course”*

tf.constant

```python
import tensorflow as tf
my_const = tf.constant([1.0, 2.0], name="my_const")
print(tf.get_default_graph().as_graph_def())
```

```
node {
  name: "my_const"
  op: "Const"
  attr {
    key: "dtype"
    value {
      type: DT_FLOAT
    }
  }
  attr {
    key: "value"
    value {
      tensor {
        dtype: DT_FLOAT
        tensor_shape {
          dim {
            size: 2
          }
        }
        tensor_content:
"\000\000\200?\000\000\000@"
      }
    }
  }
}
versions {
  producer: 24
}
```

tf.Variable

```
x = tf.Variable(...)
x.initializer # init
x.value() # read op
x.assign(...) # write op
x.assign_add(...)
# and more
```

WARNING!

- **this old way is discouraged**
- TensorFlow recommends that we use the wrapper **tf.get_variable,** which allows for easy variable sharing

```
s = tf.Variable(2, name="scalar")
m = tf.Variable([[0, 1], [2, 3]], name="matrix")
W = tf.Variable(tf.zeros([784,10]))
```

tf.get_variable

```
tf.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
)
```

With **tf.get_variable**, we can provide

- variable's internal name,
- shape,
- type
- initializer to give the variable its initial value.

Note that when we use tf.constant as an initializer, we don't need to provide shape.

```
s = tf.get_variable("scalar", initializer=tf.constant(2))
m = tf.get_variable("matrix", initializer=tf.constant([[0, 1], [2, 3]]))
W = tf.get_variable("big_matrix", shape=(784, 10),
                    initializer=tf.zeros_initializer())
```

Variable initialization

You have to initialize a variable before using it, otherwise it will be raised:

```
>> FailedPreconditionError: Attempting to use uninitialized value.
```

To get a list of uninitialized variables, you can just print them out:

```python
print(session.run(tf.report_uninitialized_variables()))
```

The easiest way is initialize all variables at once:

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

tf.assign

We can assign a value to a variable using tf.Variable.assign()

```
W = tf.Variable(10)
W.assign(100)
with tf.Session() as sess:
     sess.run(W.initializer)
     print(W.eval()) # >> 10
```

Why 10 and not 100? W.assign(100) doesn't assign the value 100 to W, but instead create an assign op to do that. For this op to take effect, we have to run this op in session.

```
assign_op = W.assign(100)
with tf.Session() as sess:
     sess.run(assign_op)
     print(W.eval()) # >> 100
```

tf.constant vs tf.Variable

Differences between a constant and a variable:

1. A tf.constant **is an op.** A tf.Variable is a **class with multiple ops.**
2. A constant's **value is stored in the graph** and replicated wherever the graph is loaded. A variable is stored separately, and may live on a parameter server.

In other words:

- Constants are stored in the graph definition.
- When constants are memory expensive, such as a weight matrix with millions of entries, it will be slow each time you have to load the graph.

Control Dependencies

Sometimes, we have two or more independent ops and we'd like to specify which ops should be run first.
In this case, we use `tf.Graph.control_dependencies([control_inputs])`

```
# your graph g have 5 ops: a, b, c, d, e
with g.control_dependencies([a, b, c]):
      # `d` and `e` will only run after `a`, `b`, and `c` have executed.
      d = ...
      e = …
```

Example: Batch Normalization (BN)

- BN requires to update running statistics (mean, variance) after each training step.
- Unfortunately, the update_moving_averages operation is not a parent of train op (`train_step)` in the computational graph.
- Only the subgraph components relevant to `train_step` will be executed, **so we will never update the moving averages!**

To get around this, we have to explicitly tell the graph:

```
Update_ops = tf.get_collection(tf.GraphKeys.UPDATE_OPS)
with tf.control_dependencies(update_ops):
    # Ensures that we execute the update_ops before performing the train_step
    train_step = tf.train.GradientDescentOptimizer(0.01).minimize(loss)
```

Data Feeding (OLD)

Remember working with TF has 2 phases:

```
Phase 1: assemble a graph
Phase 2: use a session to execute operations and evaluate variables in the graph
```

We can assemble the graphs first without knowing the values needed for computation. This is equivalent to defining the function of x, y without knowing the values of x, y. For example: f(x, y) = 2x + y.

x, y are **placeholders** for the actual values.

With the graph assembled, we, or our clients, can later supply their own data when they need to execute the computation. To define a placeholder, we use:

```
a = tf.placeholder(tf.float32, shape=[3]) # a is placeholder for a vector of 3 elements
b = tf.constant([5, 5, 5], tf.float32)
c = a + b # use the placeholder as you would any tensor
a_value = [0,1,2] # this is numeric value, while `a` is symbolic
with tf.Session() as sess:
        print(sess.run(c), feed_dict={a: a_value})
```

Data Feeding

After few versions finally, TF has a usable dataset API interface.

The `tf.data` API enables you to build complex input pipelines from simple, reusable pieces.

It allows to create dataset iterators to:

- Load from binary datasets
- Load from numpy
- Load from TFRecords (TF data format)

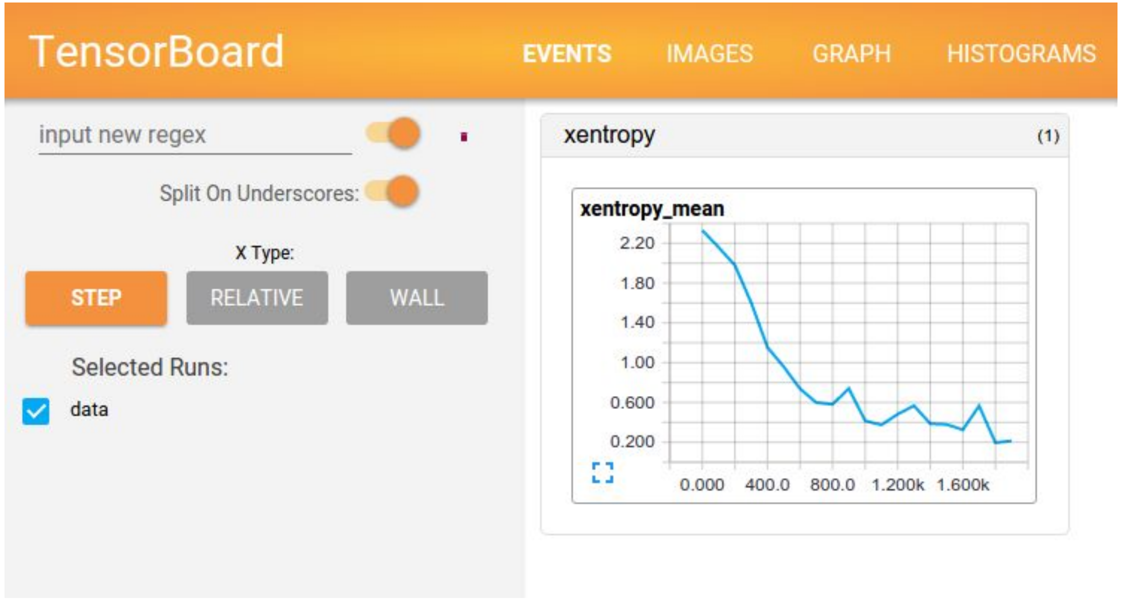
Take a look at the documentation, we'll see examples.

`https://www.tensorflow.org/programmers_guide/datasets`

Tensorboard

Tool that allows to log scalar and histogram quantities.

Helpful to track weights, gradients, losses of several experiments at the same time.



Tensorboard

```
# Create model
def multilayer_perceptron(x, weights, biases):
    # Hidden layer with RELU activation
    layer_1 = tf.add(tf.matmul(x, weights['w1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1)
    # Create a summary to visualize the first layer ReLU activation
    tf.summary.histogram("relu1", layer_1)
    # Hidden layer with RELU activation
    layer_2 = tf.add(tf.matmul(layer_1, weights['w2']), biases['b2'])
    layer_2 = tf.nn.relu(layer_2)
    # Create another summary to visualize the second layer ReLU activation
    tf.summary.histogram("relu2", layer_2)
    # Output layer
    out_layer = tf.add(tf.matmul(layer_2, weights['w3']), biases['b3'])
    return out_layer
```

Tensorboard

```
def variable_summaries(var):
  """Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""
  with tf.name_scope('summaries'):
      mean = tf.reduce_mean(var)
      tf.summary.scalar('mean', mean)
      with tf.name_scope('stddev'):
          stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
      tf.summary.scalar('stddev', stddev)
      tf.summary.scalar('max', tf.reduce_max(var))
      tf.summary.scalar('min', tf.reduce_min(var))
      tf.summary.histogram('histogram', var)
```

Tensorboard

```
# Collect summaries
merged_summaries = tf.summary.merge_all()
train_writer = tf.summary.FileWriter(FLAGS.summaries_dir + '/train', sess.graph)
(...)
for i in range(FLAGS.max_iters):
    if i % 10 == 0:  # Train and Record summaries
        summary, _ = sess.run([merged_summaries, train_op], feed_dict=val_dict)
        test_writer.add_summary(summary, i)
    else:  # Just train
        _ = sess.run([train_op], feed_dict=val_dict)
```

Let’s start with TF!
Open Jupyter!

https://codeshare.io/ayQy0o

https://goo.gl/Kki8vT

Project Recommendations

Don’t be stupid

Deep Learning could be a real PITA.

Finding bugs in a model is not always easy.

Code should be decoupled (but don’t over-engineered it):

- Data loading
- Training algorithm
- Model

Don’t be stupid II

- Don’t even think to use Windows.
- Use **Git** to version your code.
- Learn how to use VIM.
- Learn how to use ssh.
- [Respect python PEP8]

Logging and Experiments

- Use Tensorboard to inspect:
  - Losses
  - Gradients
  - Weights norm and distributions

- Use FLAGS to parametrize your scripts
- Track all the hyperparameters for each experiment (+ Loss and metrics)

“Optimization is easy when other people have found the hyper-parameter combination that works”

Acknowledgements

Slides based on https://web.stanford.edu/class/cs20si/