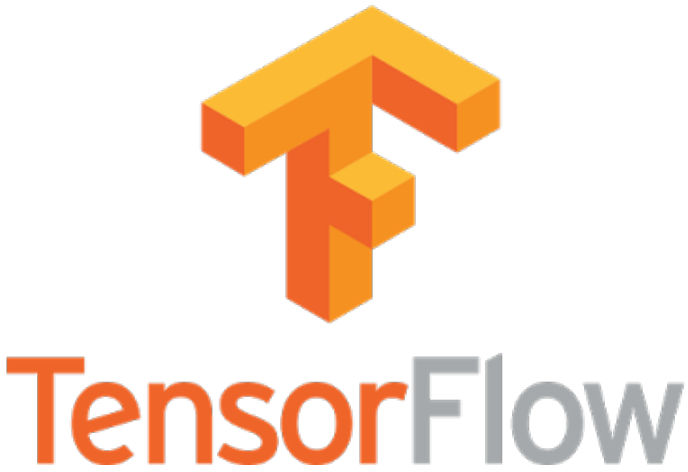
Zurich University
of Applied Sciences

zh
aw

Introduction to TensorFlow

Oliver Dürr

Datalab-Lunch Seminar Series
Winterthur, 17 Nov, 2016



Abstract

Introduction to TensorFlow

TensorFlow is a multipurpose open source software library for numerical computation using data flow graphs. It has been designed with deep learning in mind but it is applicable to a much wider range of problems.

In this tutorial I will cover the very basics of TensorFlow not going *much* into deep learning ~~at all~~. TensorFlow can be used from many programming languages. I will give simple examples, such as linear regression, showing the python API as well as the recent interface to R.

Please note that the changed room TB 534

Cheers,

Oliver

Github

- The code shown can be found in the following repositories

- R:
  - https://github.com/oduerr/tf_r
    - simple/MatrixMultiplication.Rmd
    - Linear_Regression.R

- Python:
  - https://github.com/oduerr/dl_tutorial/
    - tensorflow/simple_ops/Mandelbrot.ipynb

Some Facts about TensorFlow

- Open sourced 9th Nov. 2015
- Runs on a variety of platforms (not windows)

Automatically runs models on range of platforms:

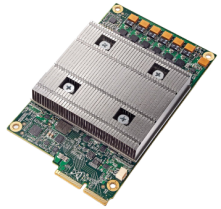
from **phones** ...

to **single machines** (CPU and/or GPUs) …

to **distributed systems** of many 100s of GPU cards

TPUs

Custom machine learning ASIC



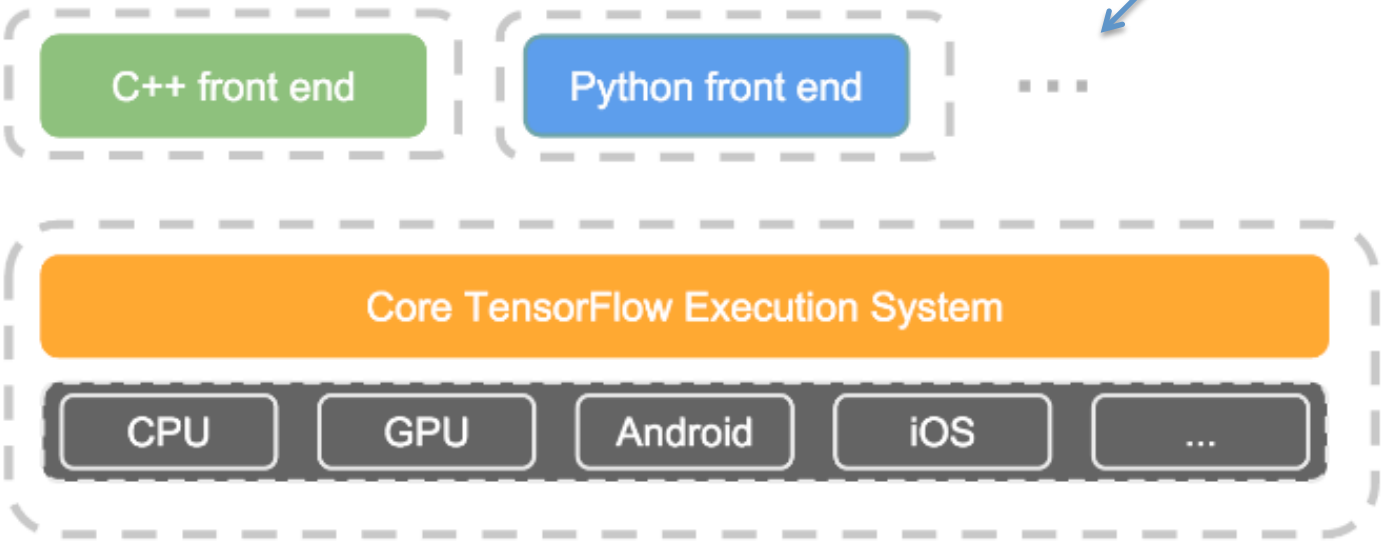
Slides from Deep Learning Day Tutorial

Some Facts about TensorFlow

Front Ends

- Different front ends for specifying/driving the computation
  - Python and C++ today, easy to add more

and **R!**



Slides from Deep Learning Day Tutorial

Tensorflow is not (only) in python!

```python
import tensorflow as tf
sess = tf.Session()
hello = tf.constant('Hello, TensorFlow')
sess.run(hello)
```

```
library(tensorflow)
sess <- tf$Session()
hello <- tf$constant('Hello, TensorFlow')
sess$run(hello)
```

[-] **PM_ME_ELLEN_PAO** 15 points 1 month ago

```
library(tensorflow)
sess = tf$Session()
hello <- tf$constant('Hello, TensorFlow!')
sess$run(hello)
W <- tf$Variable(tf$zeros(shape(784L, 10L)))
b <- tf$Variable(tf$zeros(shape(10L)))
```

My eyes! It burns!

https://www.reddit.com/r/MachineLearning/comments/54xw9e/rstudiotensorflow_tensorflow_for_r/

What is TensorFlow

- It's API about **tensors**, which flow in a computational graph

https://www.tensorflow.org/

- What are **tensors**?

What is a tensor?

In this course we only need the simple and easy accessible definition of Ricci:

Definition. A tensor of type (*p*, *q*) is an assignment of a multidimensional array

$$T^{i_1 \dots i_p}_{j_1 \dots j_q}[\mathbf{f}]$$

to each basis $\mathbf{f} = (\mathbf{e}_1, \dots, \mathbf{e}_n)$ of a fixed *n*-dimensional vector space such that, if we apply the change of basis

$$\mathbf{f} \mapsto \mathbf{f} \cdot R = \left(\mathbf{e}_i R^i_1, \dots, \right.$$

Just kidding...

then the multidimensional array obeys the transformation law

$$T^{i'_1 \dots i'_p}_{j'_1 \dots j'_q}[\mathbf{f} \cdot R] = (R^{-1})^{i'_1}_{i_1} \cdots (R^{-1})^{i'_p}_{i_p} \, T^{i_1, \dots, i_p}_{j_1, \dots, j_q}[\mathbf{f}] \, R^{j_1}_{j'_1} \cdots R^{j_q}_{j'_q}.$$

Sharpe, R. W. (1997). Differential Geometry: Cartan's Generalization of Klein's Erlangen Program. Berlin, New York: Springer-Verlag. p. 194. ISBN 978-0-387-94732-7.

What is a tensor?

For TensorFlow: A tensor is an array with several indices (like in numpy).
Order are number of indices and shape is the range.

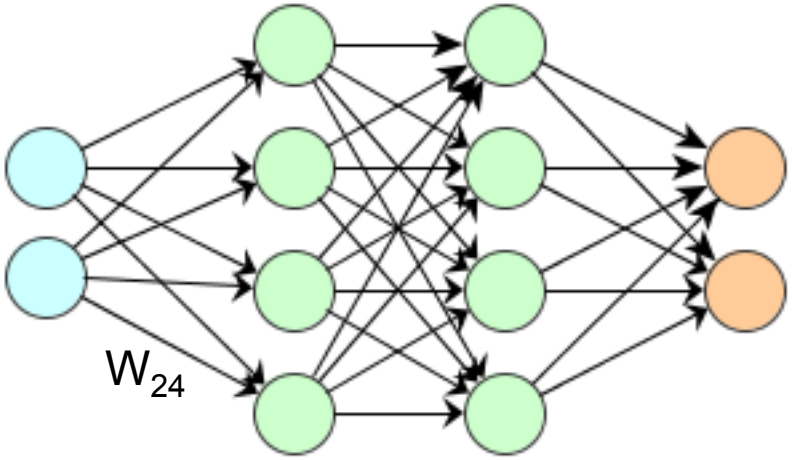
```
In [1]: import numpy as np

In [2]: T1 = np.asarray([1,2,3]) #Tensor of order 1 aka Vector
        T1

Out[2]: array([1, 2, 3])

In [3]: T2 = np.asarray([[1,2,3],[4,5,6]]) #Tensor of order 2 aka Matrix
        T2

Out[3]: array([[1, 2, 3],
               [4, 5, 6]])

In [4]: T3 = np.zeros((10,2,3)) #Tensor of order 3 (Volume like objects)

In [6]: print(T1.shape)
        print(T2.shape)
        print(T3.shape)

        (3,)
        (2, 3)
        (10, 2, 3)
```

What is a tensor (in R)

A tensor is a typed multi-dimensional array. Tensors can take the form of a single value, a vector, a matrix, or an array in many dimensions. When you initialize the value of a tensor you can use the following R data types for various tensor shapes:

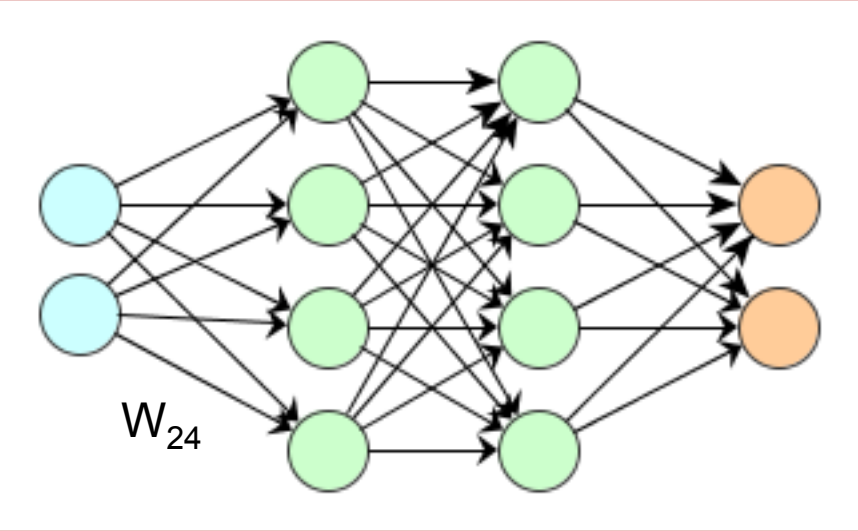
Dimensions	R Type	Example
1	vector	`c(1.0, 2.0, 3.0, 4.0)`
2	matrix	`matrix(c(1.0,2.0,3.0,4.0), nrow = 2, ncol = 2)`
3+	array	`array(rep(1, 365*5*4), dim=c(365, 5, 4))`

Typical Tensors in Deep Learning



- The input can be understood as a vector
- The weights going from e.g. Layer L_1 to Layer L_2 can be written as a matrix (often called W)
- A mini-batch of size 64 of input vectors can be understood as tensor of order 2
  - (index in batch, x_j)
- A mini-batch of size 64 images with 256,256 pixels and 3 color-channels can be understood as a tensor of order 4.

Typical Tensors in Deep Learning



- The input can be understood as a vector

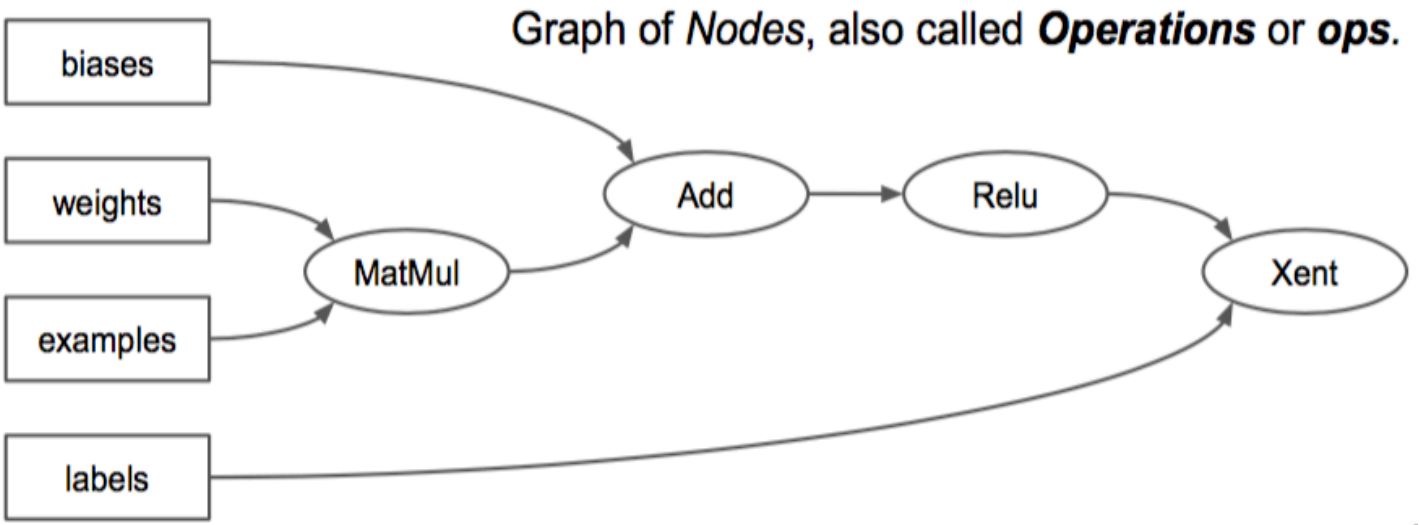
- The weights going from e.g. Layer L_1 to Layer L_2 can be written as a matrix (often called W)
- A mini-batch of size 64 of input vectors can be understood as tensor of order 2
  - (index in batch, x_j)

- A mini-batch of size 64 images with 256,256 pixels and 3 color-channels can be understood as a tensor of order 4.

- **What is the shape of the Tensors above?**

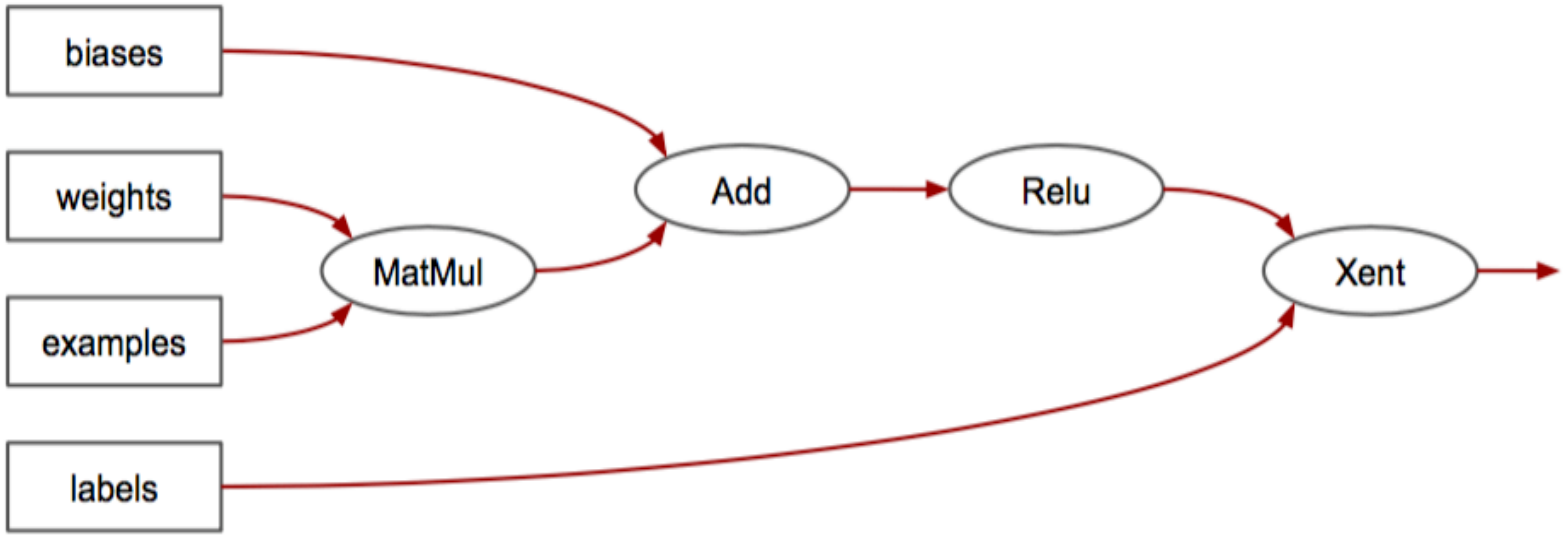
Computations in TensorFlow (and Theano)

- Computation is expressed as a dataflow graph



Computations in TensorFlow (and Theano)

- Edges are N-dimensional Arrays: Tensors



TensorFlow: Computation in 2 steps

- Computations are **done in 2 steps**
  - **First:** Build the graph
  - **Second:** Execute the graph

- Both steps can be done in many languages (python, C++, R, scala?)
- Best supported so far is python

Building the graph (python)

$$10\begin{pmatrix} 3 & 3 \end{pmatrix}\begin{pmatrix} 2 \\ 2 \end{pmatrix} = 120$$

numpy

```
In [1]:
import numpy as np
m1 = np.array([[3., 3.]])
m2 = np.array([[2.],[2.]])
10 * np.dot(m1,m2)
```

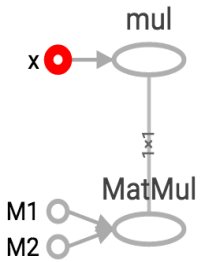
```
Out[1]:
array([[ 120.]])
```

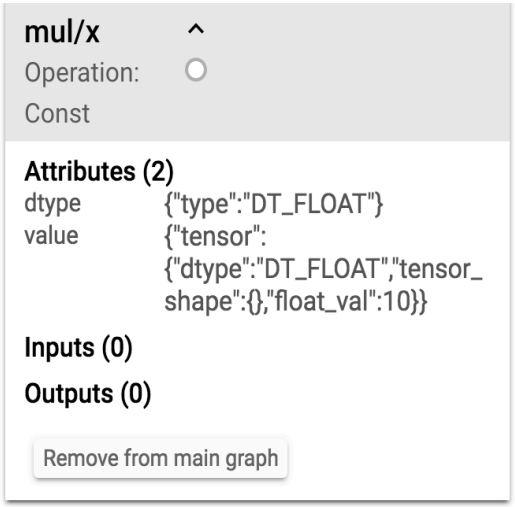
TensorFlow

```
In [3]:
import tensorflow as tf
# We construct a graph (we write to the default graph)
m1 = tf.constant([[3., 3.]], name='M1')
m2 = tf.constant([[2.],[2.]], name='M2')
product = 10*tf.matmul(m1,m2)
```

```
In [4]:
sess = tf.Session()
res = sess.run(product)
print(res)
sess.close()
```

```
[[ 120.]]
```





Have a look at the notebook: MatrixMultiplication.ipynb or MatrixMultiplication.r

Building the graph (R)

$$10\begin{pmatrix} 3 & 3 \end{pmatrix}\begin{pmatrix} 2 \\ 2 \end{pmatrix} = 120$$

In R

```
m1_values = matrix(c(3,3), nrow = 1)
m2_values = matrix(c(2,2), nrow = 2)
10 * m1_values %*% m2_values
```

```
##      [,1]
## [1,]  120
```

```
#10 * m2_values %*% m1_values
```

In Tensorflow: Building the graph

```
library(tensorflow)
tf$reset_default_graph()
m1 = tf$constant(m1_values, name='M1', dtype='float32')
m2 = tf$constant(m2_values, name='M2', dtype='float32')
product = 10*tf$matmul(m1,m2)
```

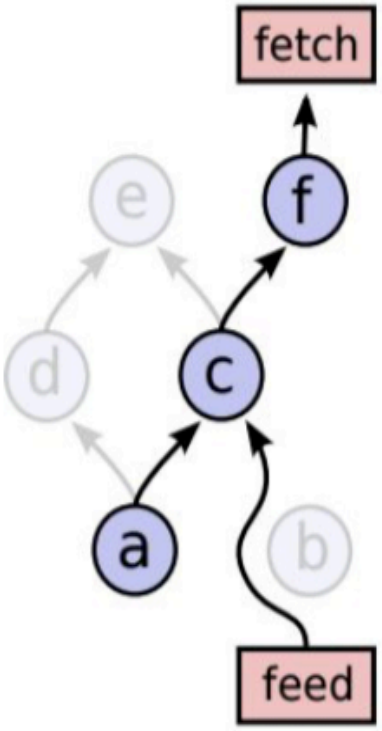
In Tensorflow: Executung the graph to the tensor "product"

```
sess = tf$Session()
res = sess$run(product)
print(res)
```

```
##      [,1]
## [1,]  120
```

```
sess$close()
```

Computations using feeding and fetching




```
res = sess.run(f, feed_dict={b:data[:,0]})
```

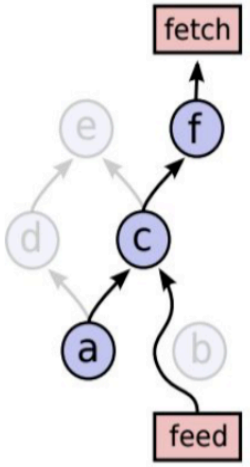
fetch
(the numeric value)

Fetch
f (symbolic)

symbolic

values

Feed and Fetch



- Fetches can be a list of tensors

- Feed (from TF docu)
  - A feed temporarily replaces the **output of an operation** with a tensor value. You supply feed data as an argument to a run() call. The feed is only used for the run call to which it is passed. The most common use case involves designating specific operations to be “feed” operations by using tf.placeholder() to create them.

```
res = sess.run(f, feed_dict={b:data[:,0]})
```

A more general example

```
x = tf.placeholder(tf.float32, shape=(1024, 1024))
res1, res2 = sess.run([loss, loss2], feed_dict={x:data[:,0], y:data[:,1]})
```

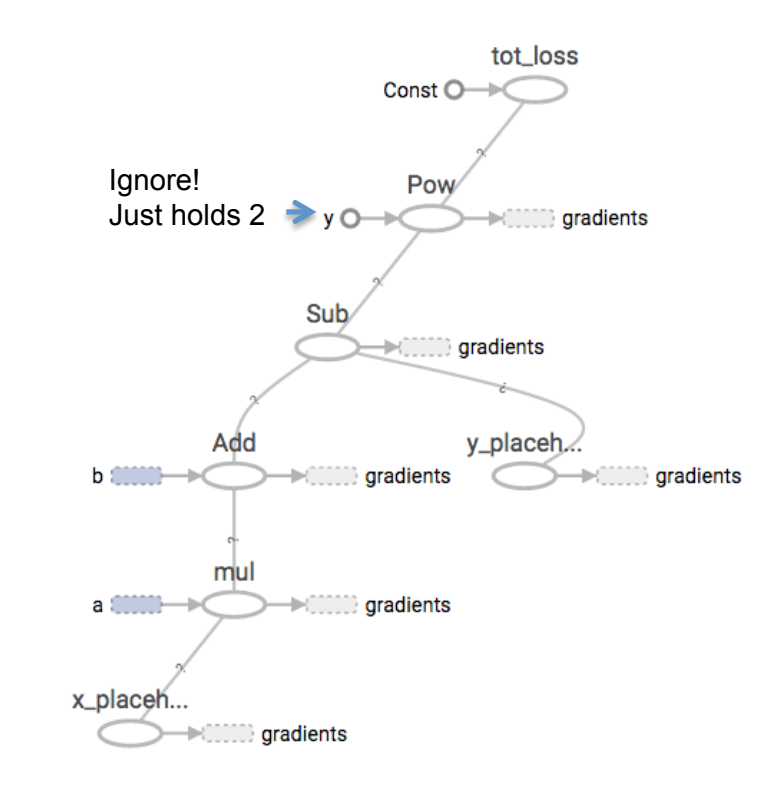
fetches
(the numeric values)

fetches
(symbolic)

two inputs (feeds)

Example: linear regression with R / Tensorflow

See: https://github.com/oduerr/tf_r/blob/master/linear_regression/Linear_Regression.R




```
x <- tf$placeholder('float32', shape(NULL), name='x_placeholder')
y <- tf$placeholder('float32', shape(NULL), name='y_placeholder')
...
loss <- tf$reduce_mean((y_hat - y) ^ 2, name='tot_loss')
...
res = sess$run(loss, feed_dict=dict(x = x_data, y = y_data))
```

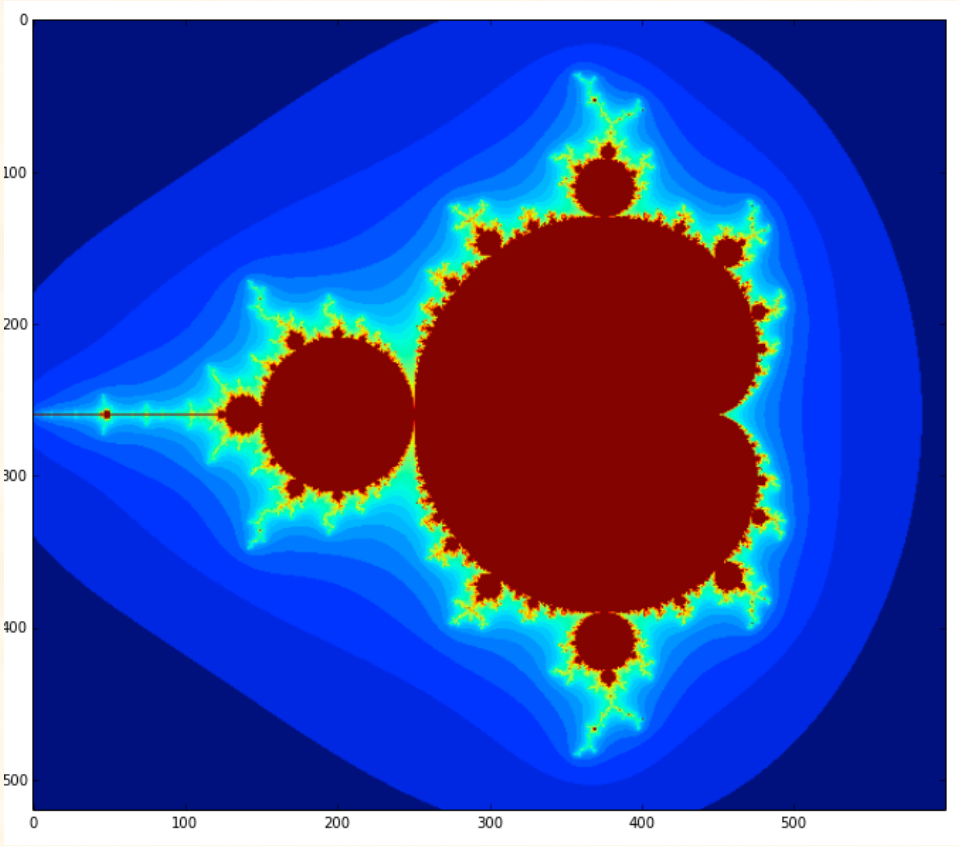
Comparing TF and numpy

Numpy	TensorFlow
`a = np.zeros((2,2)); b = np.ones((2,2))`	`a = tf.zeros((2,2)), b = tf.ones((2,2))`
`np.sum(b, axis=1)`	`tf.reduce_sum(a,reduction_indices=[1])`
`a.shape`	`a.get_shape()`
`np.reshape(a, (1,4))`	`tf.reshape(a, (1,4))`
`b * 5 + 1`	`b * 5 + 1`
`np.dot(a,b)`	`tf.matmul(a, b)`
`a[0,0], a[:,0], a[0,:]`	`a[0,0], a[:,0], a[0,:]`

https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf

Example: Mandelbrot in python

- https://github.com/oduerr/dl_tutorial/blob/master/tensorflow/simple_ops/Mandelbrot.ipynb



Specialities in R
for reference (not shown in talk)

Specialities in R

See https://rstudio.github.io/tensorflow/using_tensorflow_api.html

- "." ➔"$"
  - `tf$train$GradientDescentOptimizer(0.5)`
- Be explicit about the type
  - `tf$nn$conv2d(x, W, strides=c(1L, 1L, 1L, 1L), padding='SAME')`
- Lists (when you have NULL or single element).

```
x <- tf$placeholder(tf$float32, list(NULL, 784L))
W <- tf$Variable(tf$zeros(list(784L, 10L)))#OK with c(784L, 10L)
b <- tf$Variable(tf$zeros(list(10L)))
```

- TensorShape use List or `shape(NULL, 784L)`
- For dictionares us `dict:`
  - `feed_dict=dict(x = x_data, y = y_data)`

Specialities in R

- If you are inside TensorFlow you are 0 based.
  - `# call tf$argmax on the second dimension of the specified tensor`
  - `correct_prediction <- tf$equal(tf$argmax(y_conv, 1L), tf $argmax(y_, 1L))`

Tensor Extraction

Tensors can created by extracting elements from other tensors, either using functions like `tf$gather` and `tf$slice`, or by using `[` syntax. Extraction using `[` also expects 0-based indexes, but otherwise follows R conventions: the index range is inclusive (whereas Python omits the last element of an indexing vector) and blank indices indicate that all elements in that dimension should be retained. For example:

```
# take the elements in the first 100 rows and first 5 columns of W
W_small <- W[0:99, 0:4]

# take the last 3 columns, but all rows, of W
W_smallish <- W[, 7:9]
```

- `tf$reset_default_graph() #Also good idea in python`

Debugging with the tf.print() ops

During the construction of the graph one can include the print operation.

```
a = tf$Print(a, list(b))
```

- It adds a print function to the operation a
- It prints the values of the tensors in the list (here just b)

Options:

```
tf$Print(a, list(b, a), first_n = 5, message = 'Value of a and b’)
```

- Just print the first five calls to a

Tricks

```
a = tf$Print(a, list(b, a), first_n = 5, message = 'Value of a and b')
a = tf$Print(a, list(tf$maximum(a,b)), first_n = 5, message = 'Max')
```

You can also use a function

Further links

- Debugging: https://wookayin.github.io/TensorflowKR-2016-talk-debugging/
- Deep Learning Day: https://github.com/cuscino/tensorflow/blob/master/Introduction.ipynb
- https://cs224d.stanford.edu/lectures/CS224d-Lecture7.pdf
- Introduction on TF: http://jorditorres.org/first-contact-with-tensorflow/