How to build & run your first deep learning network in TensorFlow

Jordi Torres & Maurici Yagües

Barcelona, July 2016

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

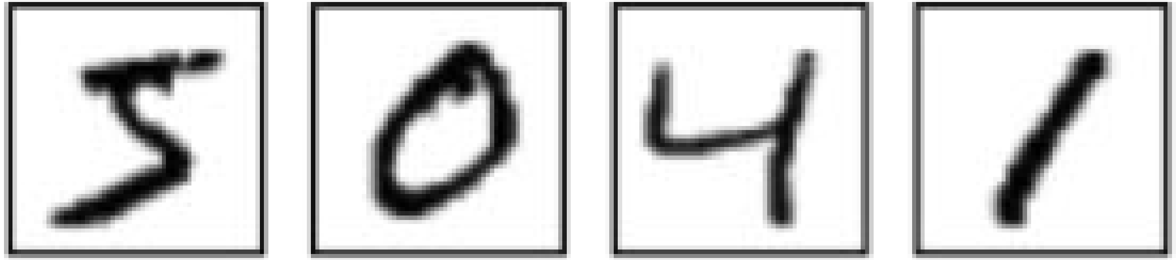
DAY 3

Single Layer Neural Network in TensorFlow

TensorBoard

Single Layer Neural Network in TensorFlow

- MNIST Data-set : “Hello World”
    - set of black and white images containing hand-written digits, containing more than 60.000 examples for training a model, and 10.000 for testing it.
    - The images have been normalized into 20x20 pixel images, preserving the aspect ratio. After that, the images are centered in 28x28 pixel frames
    - The images are labeled with the digit they represent.
    - The images are like:



Single Layer Neural Network in TensorFlow

- MNIST Data-set : “Hello World”
    - To download easily the data, you can use the script input_data.py
    - From your application you only need to import and use in the following way:

```
import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

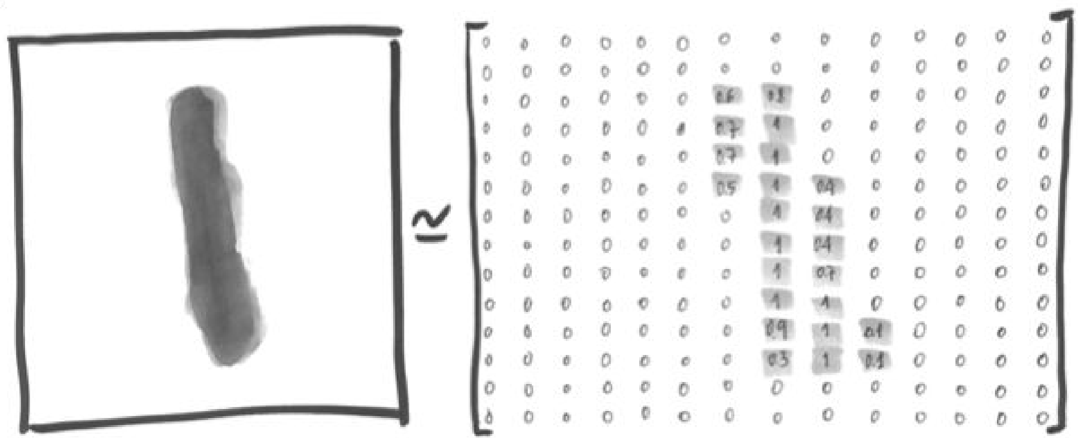
Single Layer Neural Network in TensorFlow

- MNIST Data-set : “Hello World”
  - After executing these two instructions you will have
    - the full training data-set in *mnist.train* and
    - the test data-set in *mnist.test*.
    - each element is composed by an image, referenced as “*xs*”, and its corresponding label “*ys*”

Single Layer Neural Network in TensorFlow

- MNIST Data-set : “Hello World”

  Images



  Where each position indicates the level of lackness of each pixel between 0 and 1. This matrix can be represented as an array of 28×28 = 784 numbers. Actually, the image has been transformed in a bunch of points in a vectorial space of 784 dimensions.

Single Layer Neural Network in TensorFlow

- An easy example to start: Softmax

  The *softmax* function has two main steps:

  1. Compute the “evidences” for an image belonging to a certain label

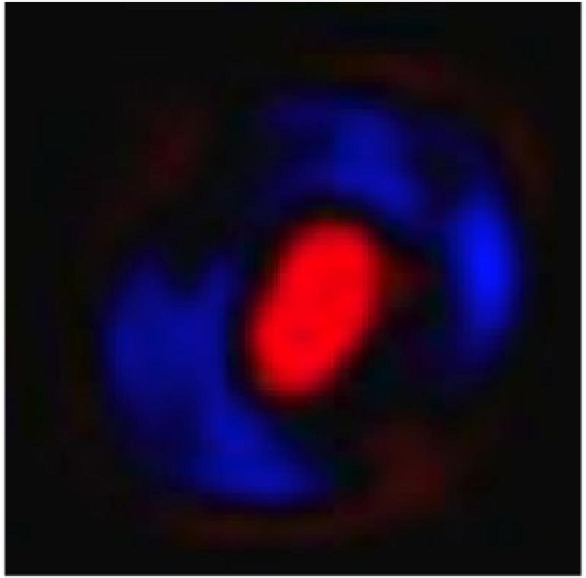
  $$\text{evidence}_i = \sum_j W_{i,j} x_j + b_i$$

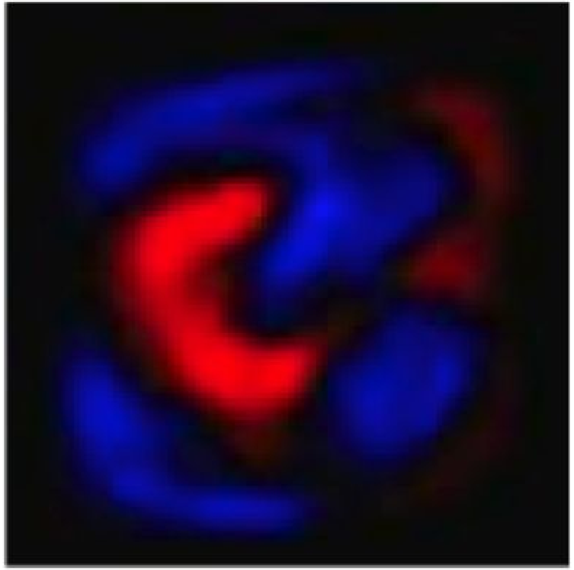
  2. Convert the evidences into probabilities for each possible label

  $$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Single Layer Neural Network in TensorFlow

1. **Compute the "evidences" for an image belonging to a certain label are**
    - a usual approximation is to compute the weighted sum of pixel intensities.
    - That weight is negative when a pixel with high intensity happens to not to be in a given class,
    - and positive if the pixel is frequent in that class.

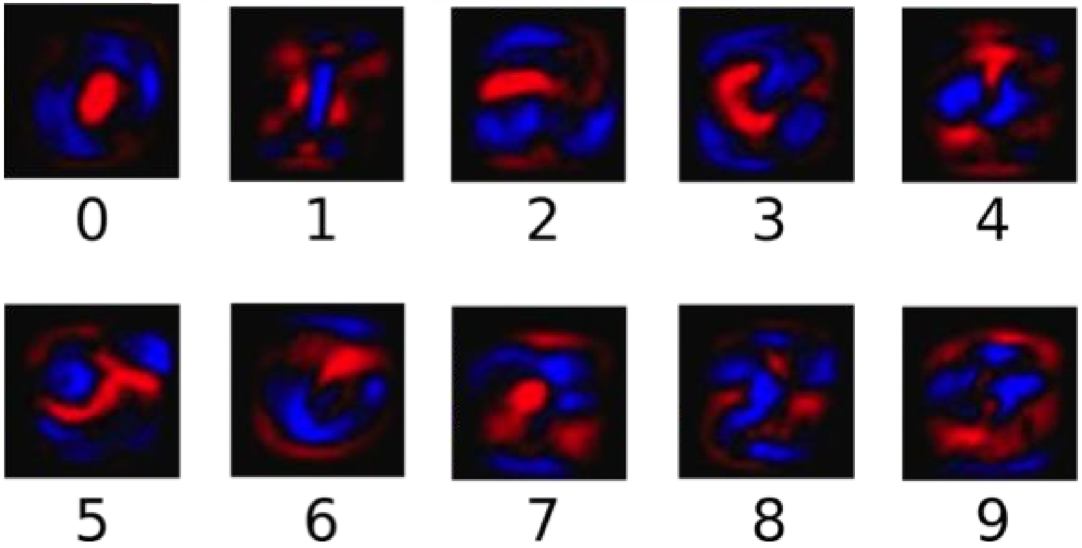




Single Layer Neural Network in TensorFlow

1. **Compute the “evidences” for an image belonging to a certain label are**
   - In a more formal way, we can say that the evidence for a class *i* given an input x is expressed as:

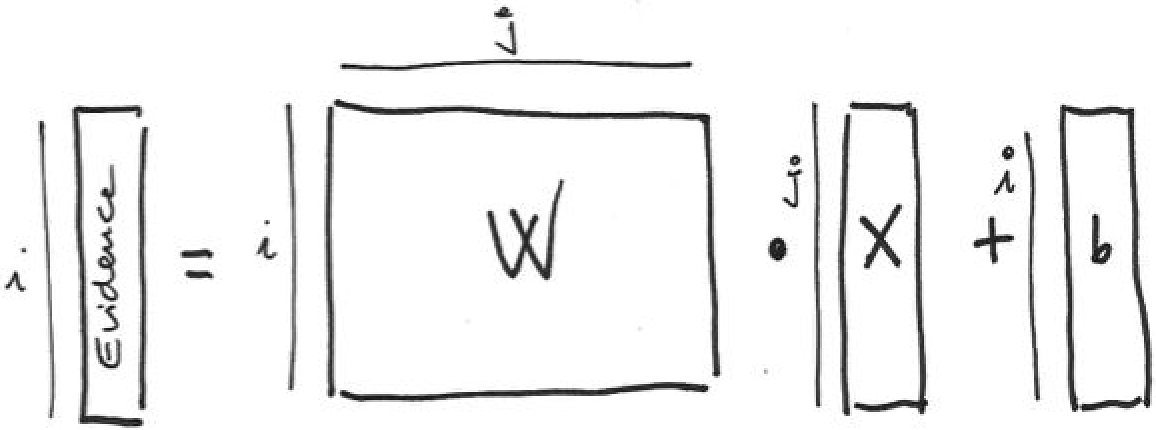
$$\text{evidence}_i = \sum_j W_{i,j} x_j$$



Single Layer Neural Network in TensorFlow

1. **Compute the “evidences” for an image belonging to a certain label are**
    - The model also include an extra parameter representing the bias (adding some base uncertainty)

$$\text{evidence}_i = \sum_j W_{i,j} x_j + b_i$$



Single Layer Neural Network in TensorFlow

2. Convert the evidences into probabilities for each possible label.

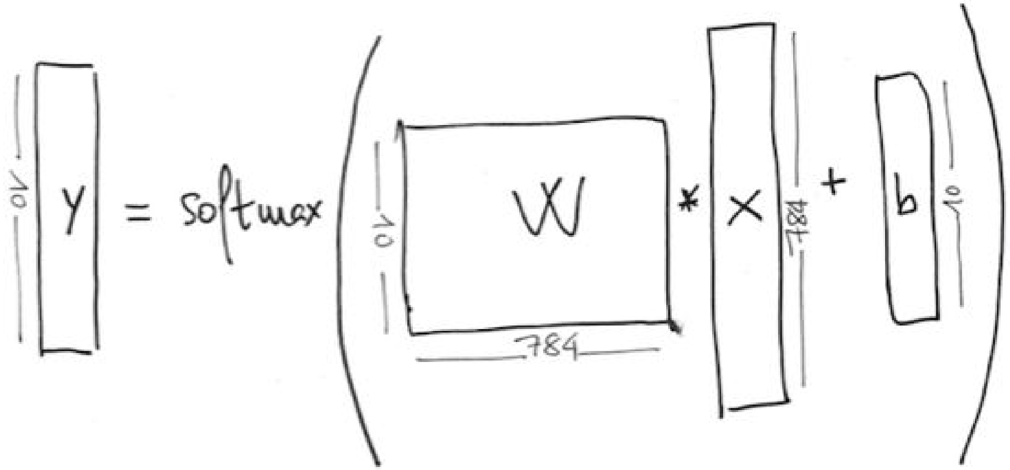
$$y = \text{softmax}(\text{evidence})$$

$$\text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

The interesting fact of such function is that a good prediction will have one output with a value near 1, while all the other outputs will be near zero; and in a weak prediction, some labels may show similar support.

Single Layer Neural Network in TensorFlow

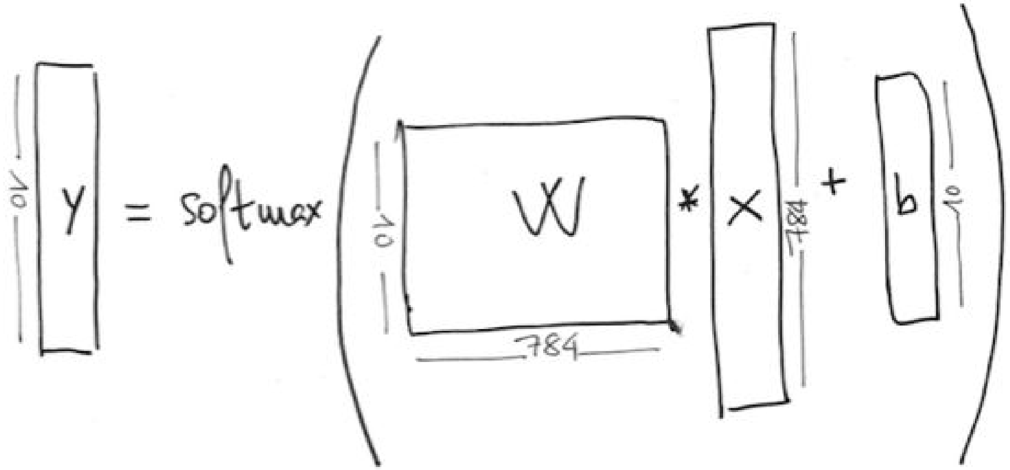
Data structure:



Two variable are created using the tf.Variable function

```
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
```

Single Layer Neural Network in TensorFlow

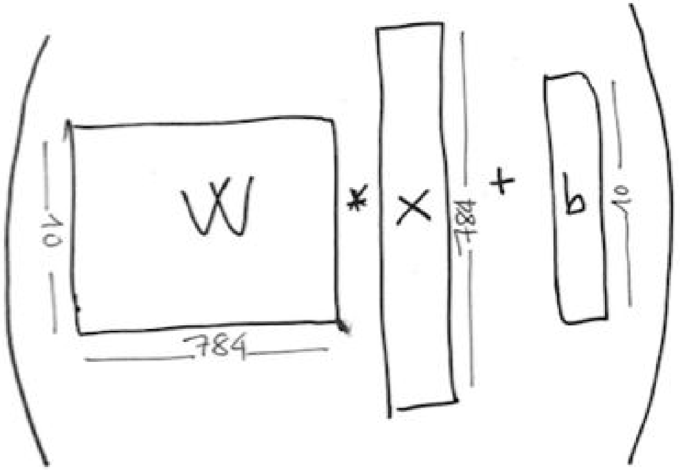


We create a tensor of two dimensions to keep the information of the x points (used to store the MNIST images as a vector of 784 floating point values (*)).

```
x = tf.placeholder("float", [None, 784])
```

(*) None indicates that the dimension can be any size

Single Layer Neural Network in TensorFlow

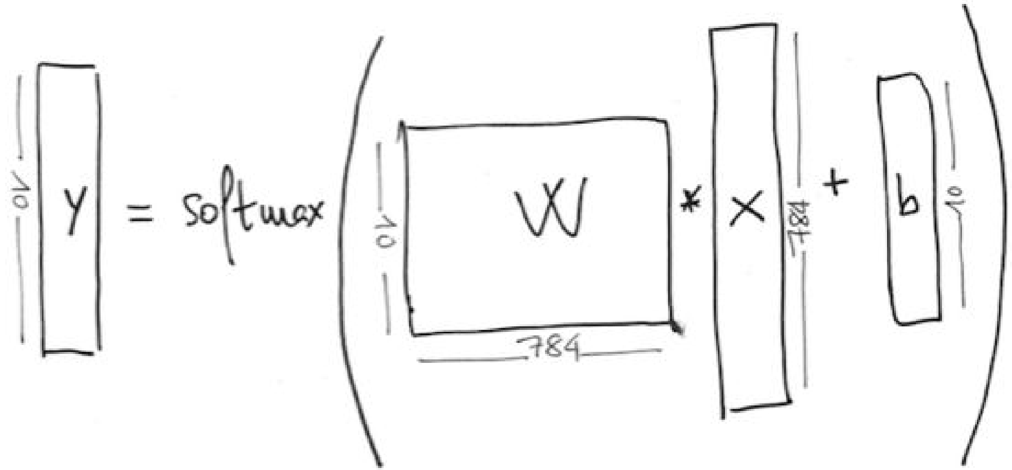



```
tf.matmul(x,W) + b
```

Single Layer Neural Network in TensorFlow

- implementing the previously described softmax function:

```
y = tf.nn.softmax(tf.matmul(x,W) + b)
```



Single Layer Neural Network in TensorFlow

- Once specified the model implementation, we can specify the necessary code to obtain the weights for W and bias b using an iterative training algorithm
    - For each iteration, the training algorithm gets the training data, applies the neural network and compares the obtained result with the expected one.
    - As a cost function : *cross entropy error*

$$-\sum_i y'_i \log(y_i)$$

Single Layer Neural Network in TensorFlow

- To implement the cross-entropy measurement we need:

```
y_ = tf.placeholder("float", [None,10])
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
```

- As a iterative minimization process: backpropagation with gradient descent method using the cross-entropy cost function.

```
train_step = tf.train.GradientDescentOptimizer(0.01).minimize
(cross_entropy)
```

(*) 0.01 is the *learning rate* hyperparameter that indicates the speed of change

Single Layer Neural Network in TensorFlow

- Start the computation by instantiating tf.Session():

```
sess = tf.Session()

sess.run(tf.initialize_all_variables())

for i in range(1000):

  batch_xs, batch_ys = mnist.train.next_batch(100)

  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

For each iteration, a bundle of 100 inputs of data, randomly sampled from the training data-set, are picked.

The returning parameter for *train_step*, when executed, will apply the gradient descent to the involved parameters. So training the model can be achieved by repeating the *train_step* execution.

Single Layer Neural Network in TensorFlow

- A **model must be evaluated** after training to see how much “good” is!
  - For example, we can compute the percentage of hits and misses in our prediction, seeing which examples were correctly predicted

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))

accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

print sess.run(accuracy, feed_dict={x: mnist.test.images, y_:
mnist.test.labels})
```

Single Layer Neural Network in TensorFlow

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

the *tf.argmax(y,1)* function returns the index of the highest value of a tensor according a given axis. In effect, *tf.argmax(y, 1)* is the label in our model with higher probability for each input, while *tf.argmax(y_,1)* is the correct label.

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

Using *tf.equal* method we can compare if our prediction coincides with the correct label.

For example, *[True, False, True, True]* will turn into *[1,0,1,1]* and the average will be 0.75 representing the percentage of *accuracy*. Now we can ask for the accuracy of our test data-set using the *mnist.test* as the *feed_dict* argument:

Single Layer Neural Network in TensorFlow

Using *tf.equal* method we can compare if our prediction coincides with the correct label:

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
```

This instruction returns a list of Booleans. To determine which fractions of predictions are correct, we can cast the values to numeric variables (floating point) and do the following operation:

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

For example, *[True, False, True, True]* will turn into *[1,0,1,1]* and the average will be 0.75 representing the percentage of *accuracy*. Now we can ask for the accuracy of our test data-set using the *mnist.test* as the *feed_dict* argument:

```
print sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels})
```

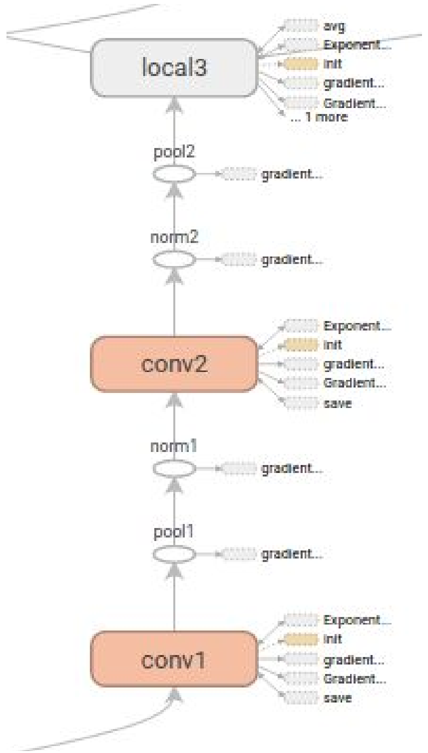
Class hands-on (or Homework)

- Download the code SingleLayerNeuralNetwork.py from github and indicates the obtained value.

TensorBoard

- Built-in graph visualization tool
- Can help you better understand the model
- Show and plot quantitative metrics about your graph
- Requires some extra work on the original code

- Some links for reference:
  - TensorBoard: Visualizing Learning
  - TensorBoard: Graph Visualization
  - Running demo (code)
  - TensorBoard README



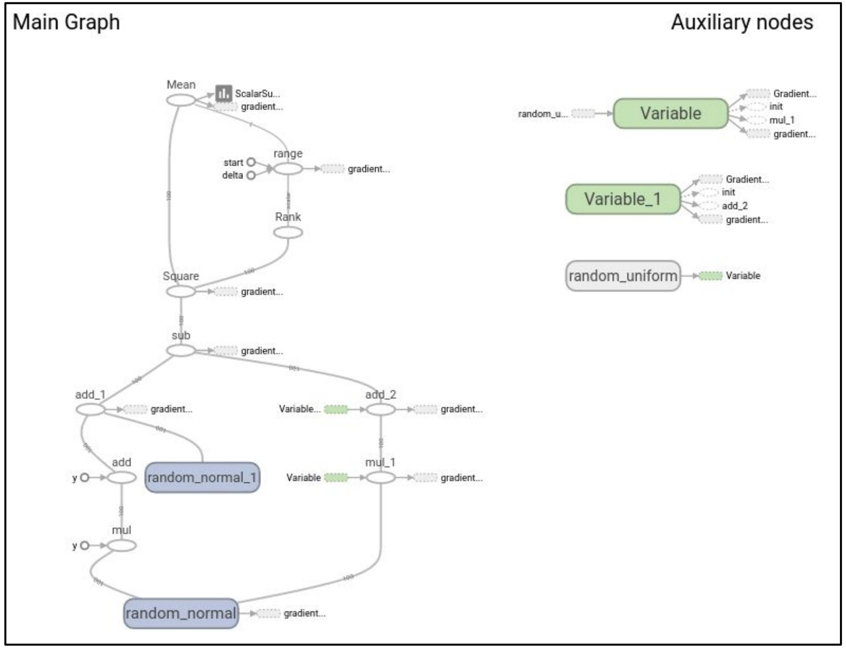
TensorBoard

- Some useful functions
  - `tf.name_scope('name')` ➙ used to define a hierarchy of the nodes in the graph
  - Summary Operations (Python API)
    - `tf.scalar_summary('name', values)` ➙ summary for scalar values
    - `tf.histogram_summary('name', values)` ➙ summary with a histogram of the values
    - `tf.merge_all_summaries()` ➙ merges all summaries collected in the graph
  - Adding Summaries to Event Files (Python API)
    - `tf.train.SummaryWriter('logdir', sess.graph)` ➙ creates event file in the given directory
    - `.add_summary(summary, step)` ➙ adds summary to the event file
- Most functions have a *name* parameter, use it for better graph organization
- You can organize different runs and compare them modifying *'logdir'*

TensorBoard

- Step 1: Name variables and make use of scopes for organizing your graph
- Step 2: Place summaries for those values you want to keep track
- Step 3: Let TF manage all summaries with tf.merge_all_summaries()
- Step 4: Create a writer pointing to the desired directory
- Step 5: Add the merged summaries to the writer
- Step 6: Launch TensorBoard

TensorBoard




```
import tensorflow as tf

tf.set_random_seed(1234)
x = tf.random_normal([100], mean=0.0, stddev=0.9)
y = x * 0.1 + 0.3 + tf.random_normal([100], mean=0.0, stddev=0.05)

W = tf.Variable(tf.random_uniform([], minval=-1.0, maxval=1.0))
b = tf.Variable(tf.zeros([]))
y_hat = W * x + b

loss = tf.reduce_mean(tf.square(y - y_hat))

train = tf.train.GradientDescentOptimizer(0.05).minimize(loss)

init = tf.initialize_all_variables()
sess = tf.Session()
writer = tf.train.SummaryWriter('/tmp/regression/', sess.graph)
sess.run(init)

for step in range(1, 101):
    _, slope, intercept, error = sess.run([train, W, b, loss])
    if step % 10 == 0:
        print('Step %.3d ; W = %.5f ; b = %.5f; loss = %.5f' %
             (step, slope, intercept, error))
```

TensorBoard

- Step 1: Name variables and make use of scopes for organizing your graph

```
with tf.name_scope('data'):
    with tf.name_scope('x'):
        x = tf.random_normal([100], mean=0.0, stddev=0.9, name='rand_x')
    with tf.name_scope('y'):
        y_true = x * tf.costant(0.1, name='real_slope') + tf.constant(0.3, name='bias') +
                 tf.random_normal([100], mean=0.0, stddev=0.05, name='rand_y')

with tf.name_scope('W'):
    W = tf.Variable(tf.random_uniform([], minval=-1.0, maxval=1.0))

with tf.name_scope('b'):
    b = tf.Variable(tf.zeros([]))

with tf.name_scope('function'):
    y_pred = W * x + b

with tf.name_scope('error'):
    loss = tf.reduce_mean(tf.square(y_pred - y_true))
```

TensorBoard

- Step 2: Place summaries for those values you want to keep track

```
with tf.name_scope('W'):
    W = tf.Variable(tf.random_uniform([], minval=-1.0, maxval=1.0))
    tf.scalar_summary('function/W', W)

with tf.name_scope('b'):
    b = tf.Variable(tf.zeros([]))
    tf.scalar_summary('function/b', b)

with tf.name_scope('error'):
    loss = tf.reduce_mean(tf.square(y_pred - y_true))
    tf.scalar_summary('error', loss)
```

TensorBoard

- Step 3: Let TF manage all summaries with tf.merge_all_summaries()

```
merged = tf.merge_all_summaries()
```

- Step 4: Create a writer pointing to the desired directory
  - Use different paths for each run to compare the performance when modifying hyperparameters

```
writer = tf.train.SummaryWriter('/tmp/regression/run1', sess.graph)
```

- Step 5: Add the merged summaries to the writer
  - Specify a global step (iteration, etc.) to be added to the summary

```
summary_str = sess.run(merged)
writer.add_summary(summary_str, step)
```

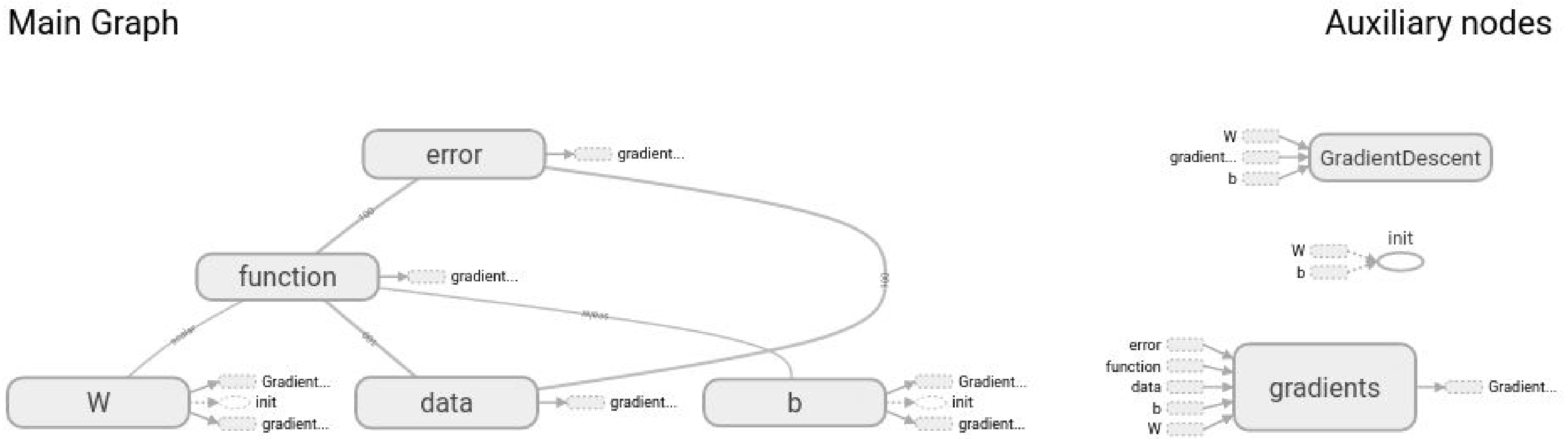
TensorBoard

- Step 6: Launch TensorBoard

  Use the command ➞ $ tensorboard --logdir=/tmp/regression/

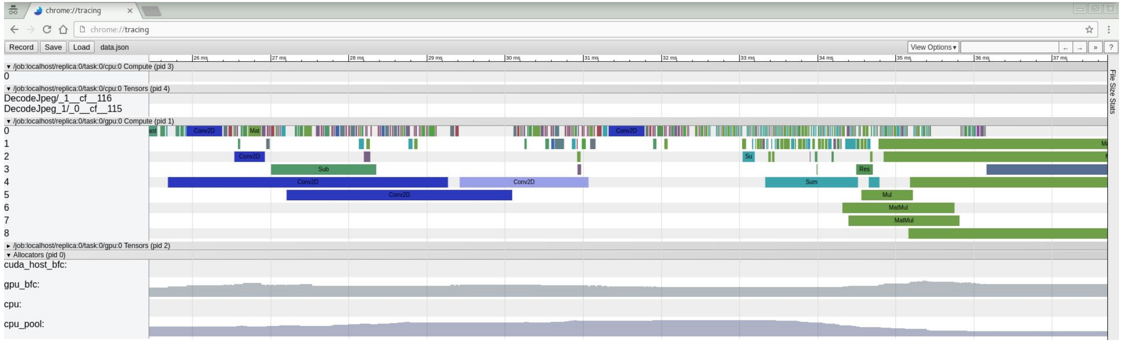
  Then browse ➞ http://localhost:6006/

  - If using Docker you may need to add the parameter -p 6006:6006



TenosorBoard

- Some runtime statistics in TensorBoard
  - See and run regression_tb_md.py for basic changes
  - Also available in [Running demo]() ([code]())
- More advanced CPU and GPU tracing
  - Steps to follow to enable it → [Link]()
  - Also, lines commented in regression_tb_md.py
  - Go to **`chrome://tracing`** and load the json file obtained



Example of a tracing with one GPU (not the regression code)

Thanks to Alberto Montes

TensorBoard

- Homework:
  - Build a TB version of the simple network build today
  - Make use of `tf.histogram_summary('name', values)` for non-scalar variables