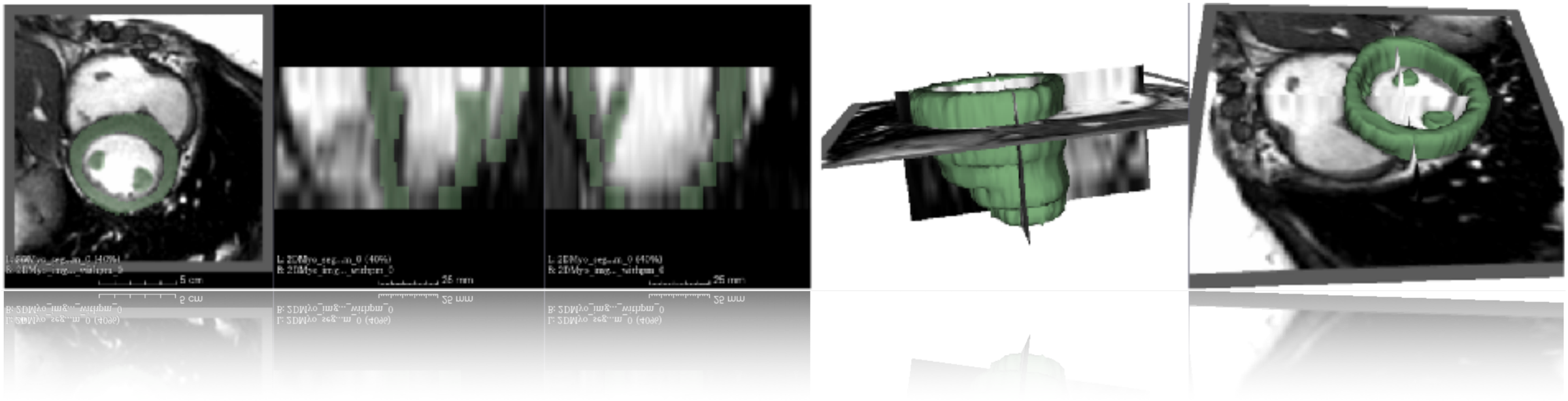


Departamento de Física Médica - Centro atómico Bariloche - IB

Deep Learning Software: TensorFlow y Keras

Ariel Hernán Curiale
ariel.curiale@cab.cnea.gov.ar

CONICET

UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO

Instituto Balseiro

CNEA

Deep learning software

Caffe
(UC Berkley)

Caffe2
(Facebook)

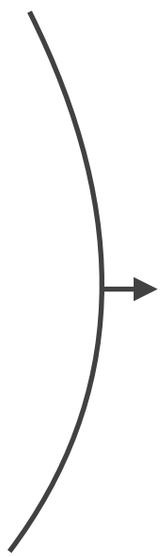
Torch
(NYU / Facebook)

PyTorch
(Facebook)

Theano
(U Montreal)

TensorFlow
(Google)

CNTK
(Microsoft)

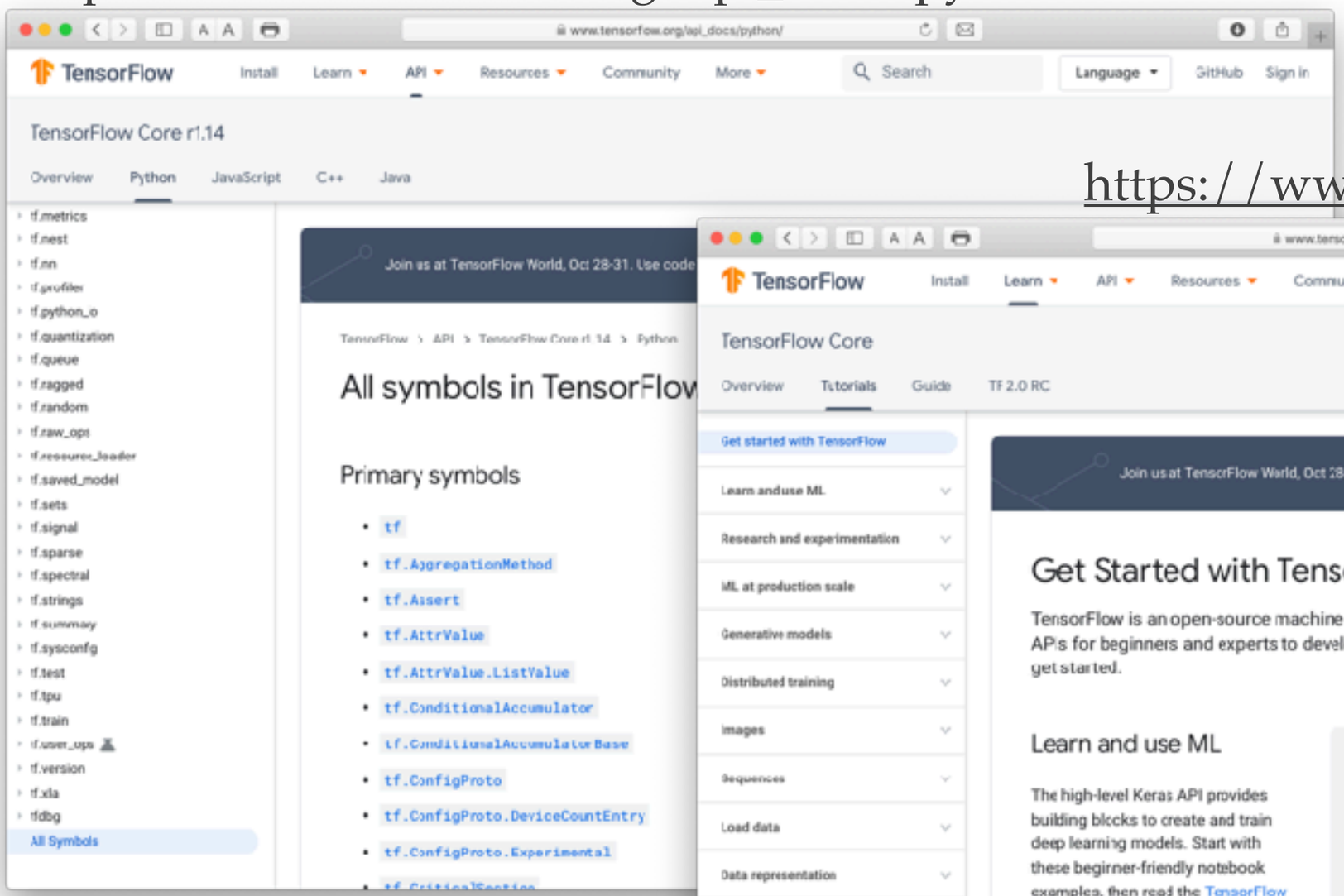


Keras
(Google)

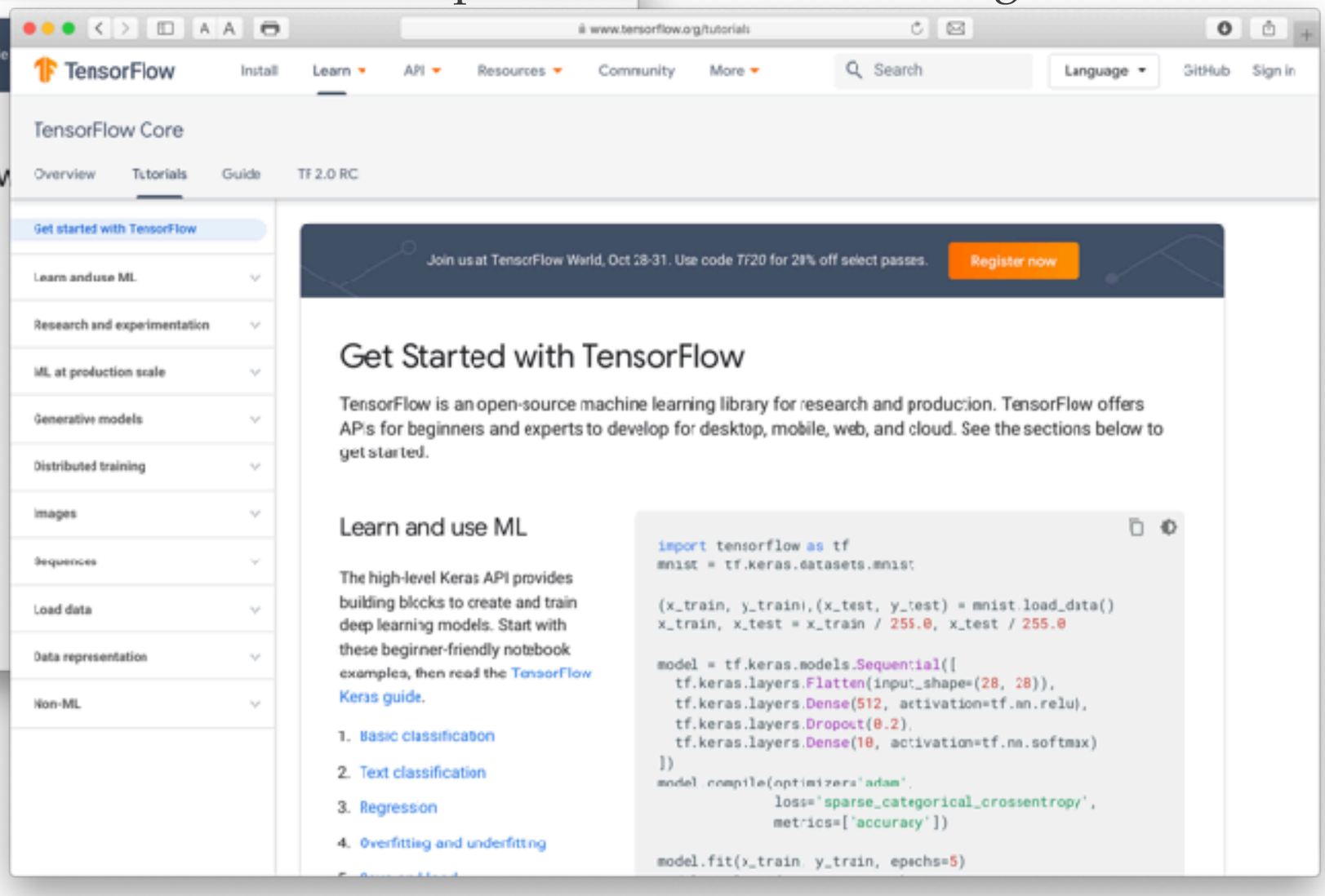
TensorFlow

TensorFlow

https://www.tensorflow.org/api_docs/python/



https://www.tensorflow.org/tutorials/



TensorFlow

- Los datos son Tensores (numpy arrays para los valores de los tensores)

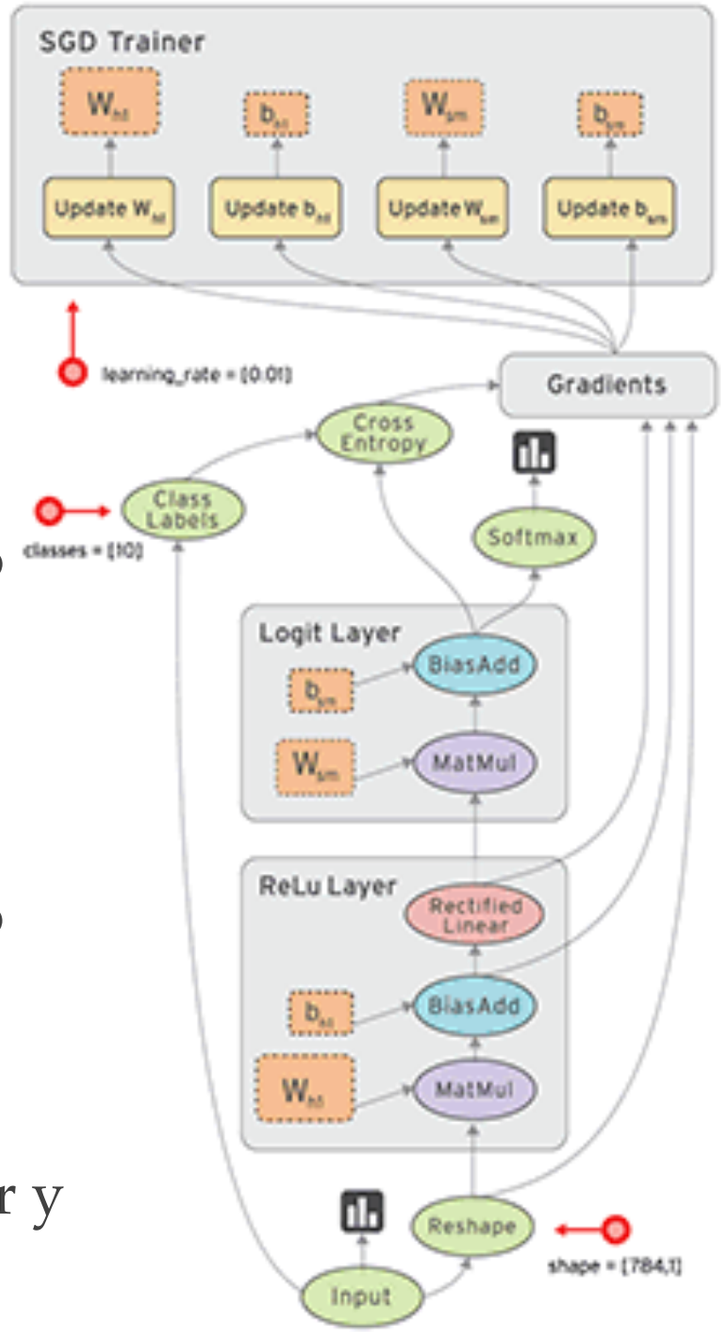
```
3. # a rank 0 tensor; a scalar with shape [],
[1., 2., 3.] # a rank 1 tensor; a vector with shape [3]
[[1., 2., 3.], [4., 5., 6.]] # a rank 2 tensor; a matrix with shape [2, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]] # a rank 3 tensor with shape [2, 1, 3]
```

- TensorFlow utiliza el flujo de datos en los grafos para representar los cálculos computacionales y las dependencias entre operaciones y variables.

- Todo programa en TF esta formado por dos grandes secciones:
  1. Creación del grafo computacional
  2. Ejecución del grafo

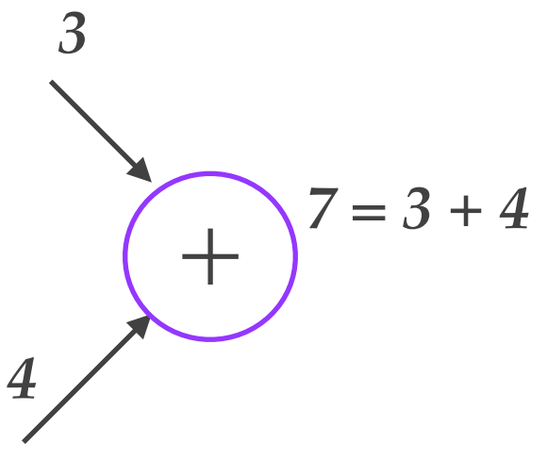
TensorFlow: Data Flow Graphs

- Data Flow Graphs es un modelo de programación muy común en computación paralela y tiene varias ventajas:
  - **Paralelismo.** Las aristas del grafo permiten identificar las dependencias entre operaciones (nodos), haciendo más simple aquellas que se pueden ejecutar en paralelo
  - **Ejecución distribuida.** Al utilizar las aristas del grafo como tensores/datos que fluyen entre las operaciones es más simple dividir las operaciones entre múltiples dispositivos (CPUs, GPUs y TPUs)
  - **Compilación.** Es más simple optimizar el código en tiempo de compilación
  - **Portabilidad.** El modelo de grafos es independiente del lenguaje. Modelos almacenados desde python se puede leer y ejecutar en C++



Source: TensorFlow

TensorFlow: Ejemplo



Ejemplo: 3 + 4 = 7

```
import numpy as np
import tensorflow as tf
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
```

Creamos el grafo

Ejecutamos el grafo

```
sess = tf.Session()
result = sess.run(total)
print(result)
```

```
7.0
```

```
print(a)
print(b)
print(total)
```

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("add:0", shape=(), dtype=float32)
```

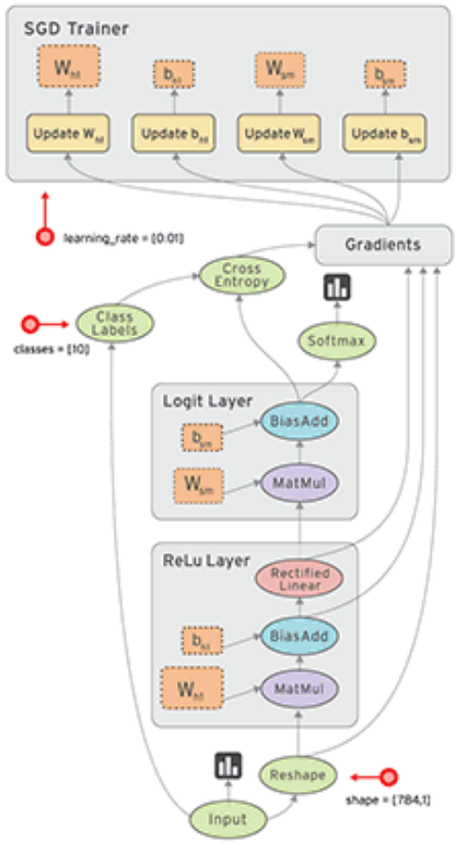
¿?

TensorFlow: Sesión

La sesión es necesaria para evaluar cualquier tensor que esta definido en el grado computacional

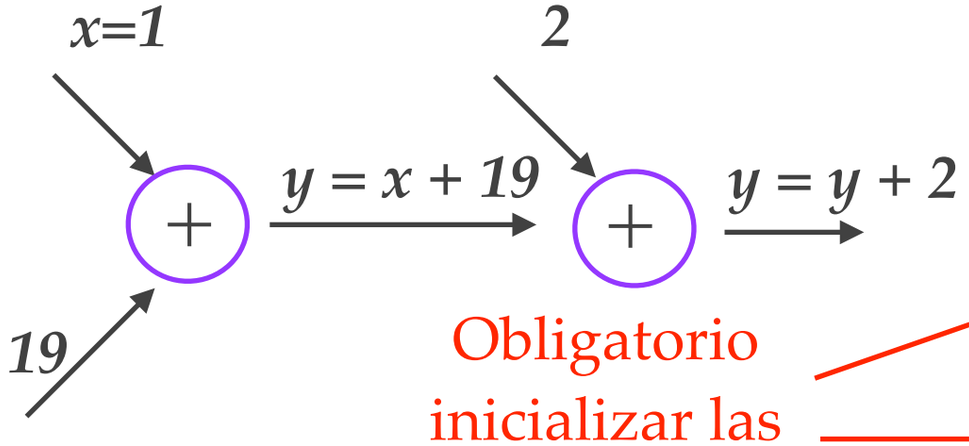
- Al ejecutar el método run Tensorflow hace un seguimiento hacia atrás del grafo para ejecutar todos los nodos

Durante la ejecución los tensores solo tienen un valor



TensorFlow: Constantes y Variables

- **Constantes:** tensores que no cambian de valor
- **Variables:** son tensores que pueden cambiar de valor que persisten a lo largo de múltiples llamadas *Session.run*()



Obligatorio inicializar las variables

```
import numpy as np
import tensorflow as tf
x = tf.constant(1, tf.float32)
y = tf.Variable(x+19, tf.float32)
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
y_val  = sess.run(y)

print(y_val)
assignment = y.assign_add(2)
sess.run(assignment)
y_val = sess.run(y)
print(y_val)
```

y = 20

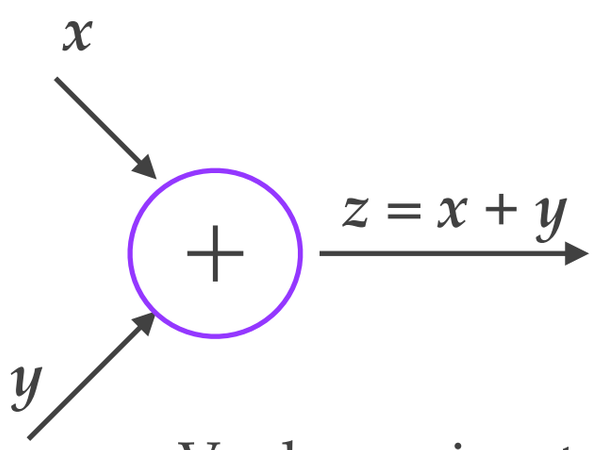
Ejecuta la asignación

Vuelve a ejecutar el grafo a partir de *y*

y = 22

TensorFlow: Placeholder

- **Placeholder:** son similares a las variables pero no es necesario proporcionar un valor inicial ya que se puede proveer en tiempo de ejecución con el argumento *feed_dict* del método *Session.run()*



Vuelve a ejecutar el grafo partir de z

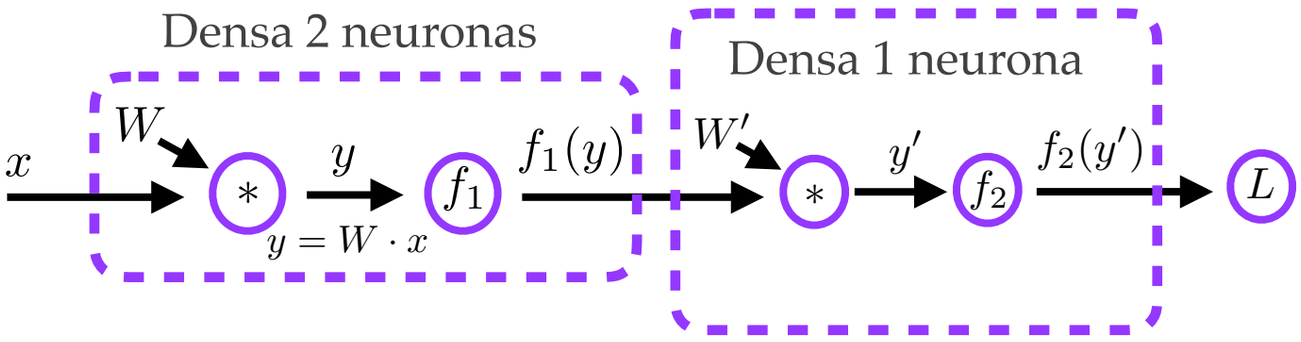
```
import numpy as np
import tensorflow as tf
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = x + y
sess = tf.Session()
z_value1 = sess.run(z, feed_dict={x: 3, y: 4.5})
z_value2 = sess.run(z, feed_dict={x: [1, 3], y: [2, 4]})

print(z_value1)
print(z_value1)
```

Feed: Diccionario de python

7.5

[3., 7.]

Problema 2.6 con TF



$$f_1(x) = \tanh(x) \qquad f_2(x) = \tanh(x)$$

$$\text{MSE} = L(y') = \frac{1}{N} \sum_i ||y_t^{(i)} - y'^{(i)}||_2^2$$

```python
# Datos
x_data = np.array([[-1,-1], [-1,1], [1,-1], [1,1]])
y_data = np.array([[1],[-1],[-1],[1]])
# Variables
x = tf.placeholder(tf.float32, shape=[None, 2])
y = tf.placeholder(tf.float32, shape=[None, 1])
W1 = tf.Variable(tf.random_normal((2,2))) # (neuronas, input)
b1 = tf.Variable(tf.random_normal((2,)))
W2 = tf.Variable(tf.random_normal((2,1))) # (neurona, input f1)
b2 = tf.Variable(tf.random_normal((1,)))
# Armamos el grafo
y1 = tf.matmul(x,W1) + b1
f1 = tf.tanh(y1)
y2 = tf.matmul(f1, W2) + b2
f2 = tf.tanh(y2)
# Ejecutamos
with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)
    out = sess.run(f2, feed_dict={x:x_data, y:y_data})
```

Definimos las variables y los placeholder
Variables: indicamos su valor inicial par

Armamos el grafo
No hay computo

Lo ejecutamos
Hay computo

Problema 2.6 en TF

```
y1 = tf.matmul(x,W1) + b1
f1 = tf.tanh(y1)
y2 = tf.matmul(f1, W2) + b2
f2 = tf.tanh(y2)
diff = y - f2
loss = tf.reduce_mean(tf.reduce_sum(diff**2, axis=1))
grad_w1, grad_b1, grad_w2, grad_b2 = tf.gradients(loss, [W1, b1, W2, b2])
lr = 1
# Actualizamos los parametros de la red
w1_update = W1.assign_sub(lr*grad_w1)
b1_update = b1.assign_sub(lr*grad_b1)
w2_update = W2.assign_sub(lr*grad_w2)
b2_update = b2.assign_sub(lr*grad_b2)
updates = tf.group(w1_update, b1_update, w2_update, b2_update)
# Ejecurtamos
with tf.Session() as sess:
    init = tf.global_variables_initializer()
    sess.run(init)
    losses = []
    for epoch in range(100):
        loss_val, _ = sess.run([loss, updates], feed_dict={x:x_data, y:y_data})
        losses.append(loss_val)
        y_pred = sess.run(f2, feed_dict={x:x_data, y:y_data})
        acc = np.isclose(y_data, y_pred, atol=1e-1).mean()
        print('epoch: {:03d}\t loss: {:.3f}\t acc:{:.3f}'.format(epoch,
            loss_val, acc))
```

Grafo: costo, gradientes y updates

No hay computo

Actualizamos

En cada época computamos el loss y actualizamos

Dispara computo gradientes

TensorBoard

Definimos una función para registrar varias cosas

```
# Para usar 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)
```

```
variable_summaries(W1)
variable_summaries(b1)
variable_summaries(W2)
variable_summaries(b2)
```

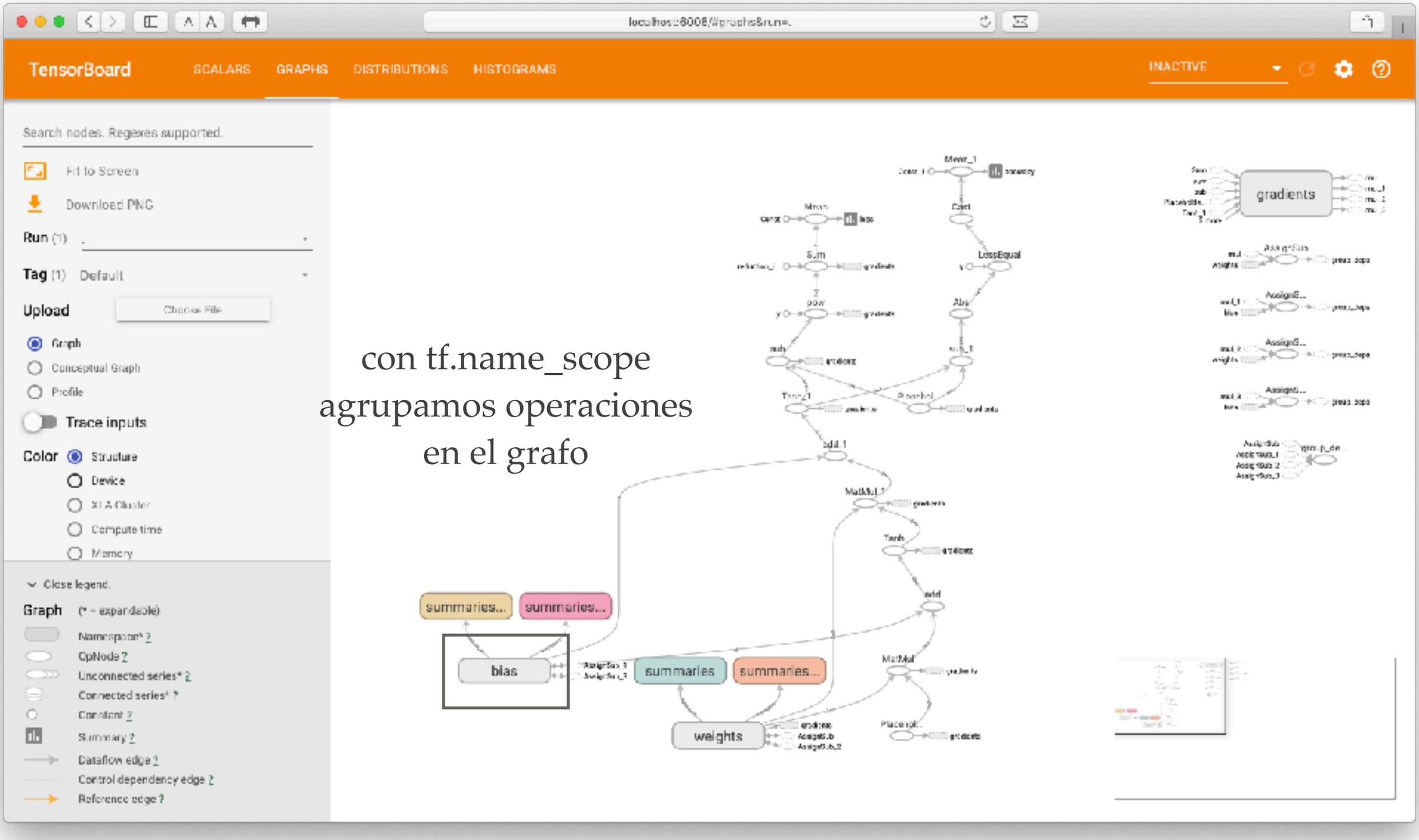
```
diff = y - f2
loss = tf.reduce_mean(tf.reduce_sum(diff**2, axis=1))
is_close = tf.less_equal(tf.abs(y-f2),0.1)
acc = tf.reduce_mean(tf.cast(is_close, tf.float32))
tf.summary.scalar('loss', loss)
tf.summary.scalar('accuracy', acc)
```

TensorBoard

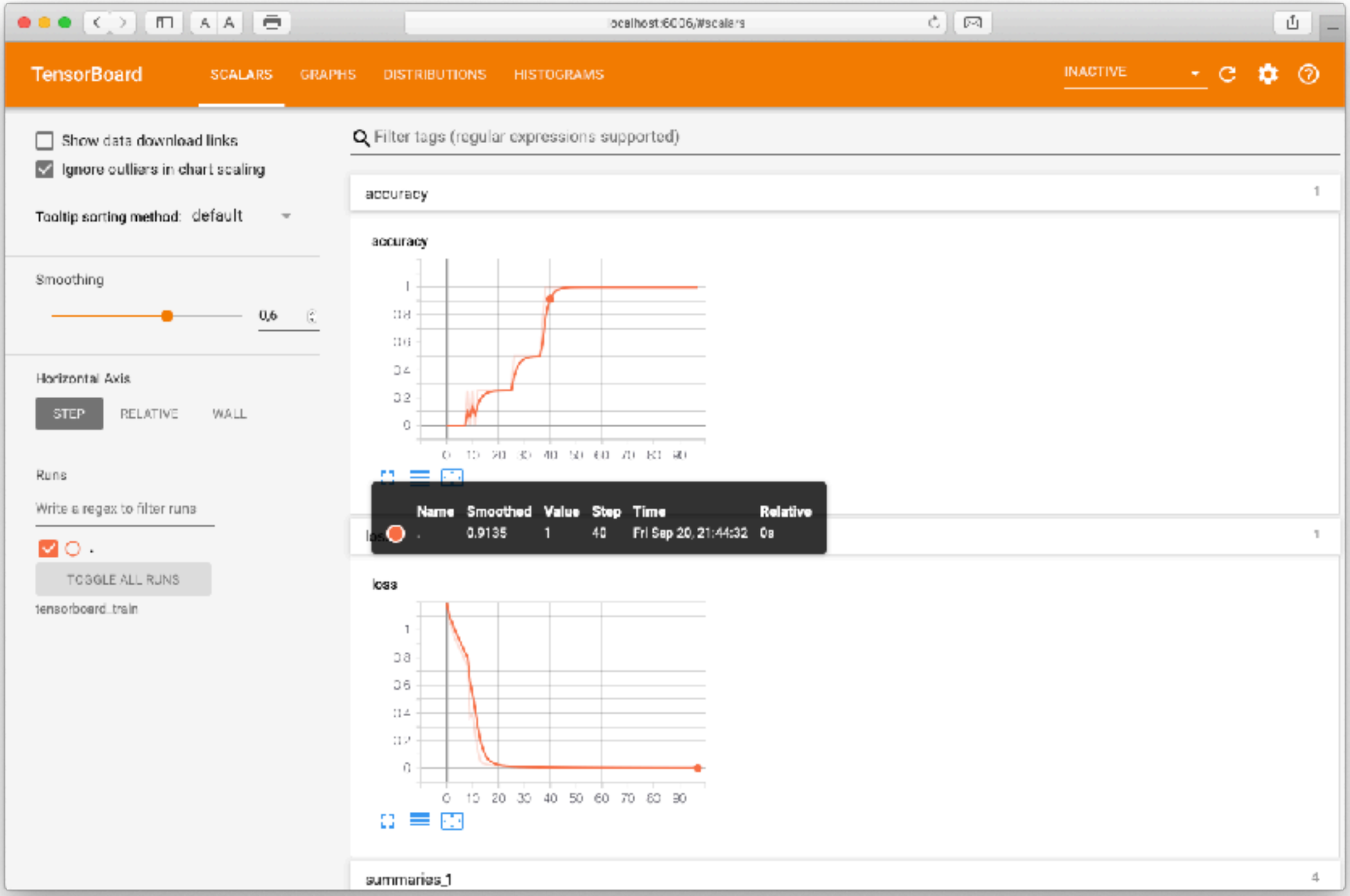
```
26 merged = tf.summary.merge_all()
29 # Ejecurtamos
30 with tf.Session() as sess:
31     train_writer = tf.summary.FileWriter('tensorboard_train', sess.graph)
32     init = tf.global_variables_initializer()
33     sess.run(init)
34     losses = []
35     for epoch in range(100):
36         summary, acc_val, loss_val, _ = sess.run([merged, acc, loss, updates],
37                 feed_dict={x:x_data, y:y_data})
38         losses.append(loss_val)
39         train_writer.add_summary(summary, epoch)
41         print('epoch: {:03d}\t loss: {:.3f}\t acc:{:.3f}'.format(epoch,loss_val, acc_val))
```

tensorboard --logdir=path/to/log-directory

TensorBoard



TensorBoard



Métricas, loss y optimizadores

```
# Armamos el grafo
x = tf.placeholder(tf.float32, shape=[None, 2], name='InputData')
y = tf.placeholder(tf.float32, shape=[None, 1], name='OutputData')
with tf.name_scope('Layer1'):
    W1 = tf.Variable(tf.random_normal((2,2)), name='W1') # (neuronas, input)
    b1 = tf.Variable(tf.zeros((2,)), name='b1')
    l1 = tf.nn.tanh(tf.matmul(x,W1) + b1)
with tf.name_scope('Layer2'):
    W2 = tf.Variable(tf.random_normal((2,1)), name='W2') # (neurona, input f1)
    b2 = tf.Variable(tf.zeros((1,)), name='b2')
    l2 = tf.nn.tanh(tf.matmul(l1,W2) + b2)
loss = tf.losses.mean_squared_error(labels=y, predictions=l2)
acc = tf.reduce_mean(tf.cast(tf.less_equal(tf.abs(y-l2),0.1), tf.float32))
acc2 = tf.metrics.mean_absolute_error(labels=y, predictions=l2)
tf.summary.scalar('Loss', loss)
tf.summary.scalar('Accuracy', acc)
merged = tf.summary.merge_all()
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1)
train = optimizer.minimize(loss)
# Ejecurtamos
with tf.Session() as sess:
    train_writer = tf.summary.FileWriter('tensorboard_train2', sess.graph)
    sess.run(tf.global_variables_initializer())
    sess.run(tf.local_variables_initializer())
    losses = []
    for epoch in range(100):
        summary, acc_val, acc2_val, loss_val, _ = sess.run([merged, acc, acc2, loss, train],
                feed_dict={x:x_data, y:y_data})
        losses.append(loss_val)
        train_writer.add_summary(summary, epoch)
        print('epoch: {:03d}\t loss: {:.3f}\t acc:{:.3f}\t acc2:{:.3f}'.format(epoch,
              loss_val, acc_val, acc2_val[0]))
```

?

loss

- Ojo cuando se utiliza la función de TF de *cross-entropy* ya que la implementación aplica la activación *softmax* por cuestiones de optimización y no hay que llamarla con *softmax* al crear la función de costo

tf.nn.softmax_cross_entropy_with_logits_v2

```
tf.nn.softmax_cross_entropy_with_logits_v2(
    _sentinel=None,
    labels=None,
    logits=None,
    dim=-1,
    name=None
)
```

Defined in `tensorflow/python/ops/nn_ops.py`.

See the guide: Neural Network > Classification

Computes softmax cross entropy between `logits` and `labels`.

Measures the probability error in discrete classification tasks in which the classes are mutually exclusive (each entry is in exactly one class). For example, each CIFAR-10 image is labeled with one and only one label: an image can be a dog or a truck, but not both.

NOTE: While the classes are mutually exclusive, their probabilities need not be. All that is required is that each row of `labels` is a valid probability distribution. If they are not, the computation of the gradient will be incorrect.

If using exclusive `labels` (wherein one and only one class is true at a time), see `sparse_softmax_cross_entropy_with_logits`.

WARNING: This op expects unscaled logits, since it performs a `softmax` on `logits` internally for efficiency. Do not call this op with the output of `softmax`, as it will produce incorrect results.

A common use case is to have logits and labels of shape `[batch_size, num_classes]`, but higher dimensions are supported, with the `dim` argument specifying the class dimension.

`logits` and `labels` must have the same dtype (either `float16`, `float32`, or `float64`).

Guardar y leer modelos

Guardar el modelo (stackoverflow)

```
# Create some variables.
v1 = tf.get_variable("v1", shape=[3], initializer = tf.zeros_initializer)
v2 = tf.get_variable("v2", shape=[5], initializer = tf.zeros_initializer)
inc_v1 = v1.assign(v1+1)
dec_v2 = v2.assign(v2-1)
# Add an op to initialize the variables.
init_op = tf.global_variables_initializer()
# Add ops to save and restore all the variables.
saver = tf.train.Saver()
# Later, launch the model, initialize the variables, do some work, and save the
# variables to disk.
with tf.Session() as sess:
  sess.run(init_op)
  # Do some work with the model.
  inc_v1.op.run()
  dec_v2.op.run()
  # Save the variables to disk.
  save_path = saver.save(sess, "/tmp/model.ckpt")
  print("Model saved in path: %s" % save_path)
```

Una **mejor** forma de crear variables

Guardar y leer modelos

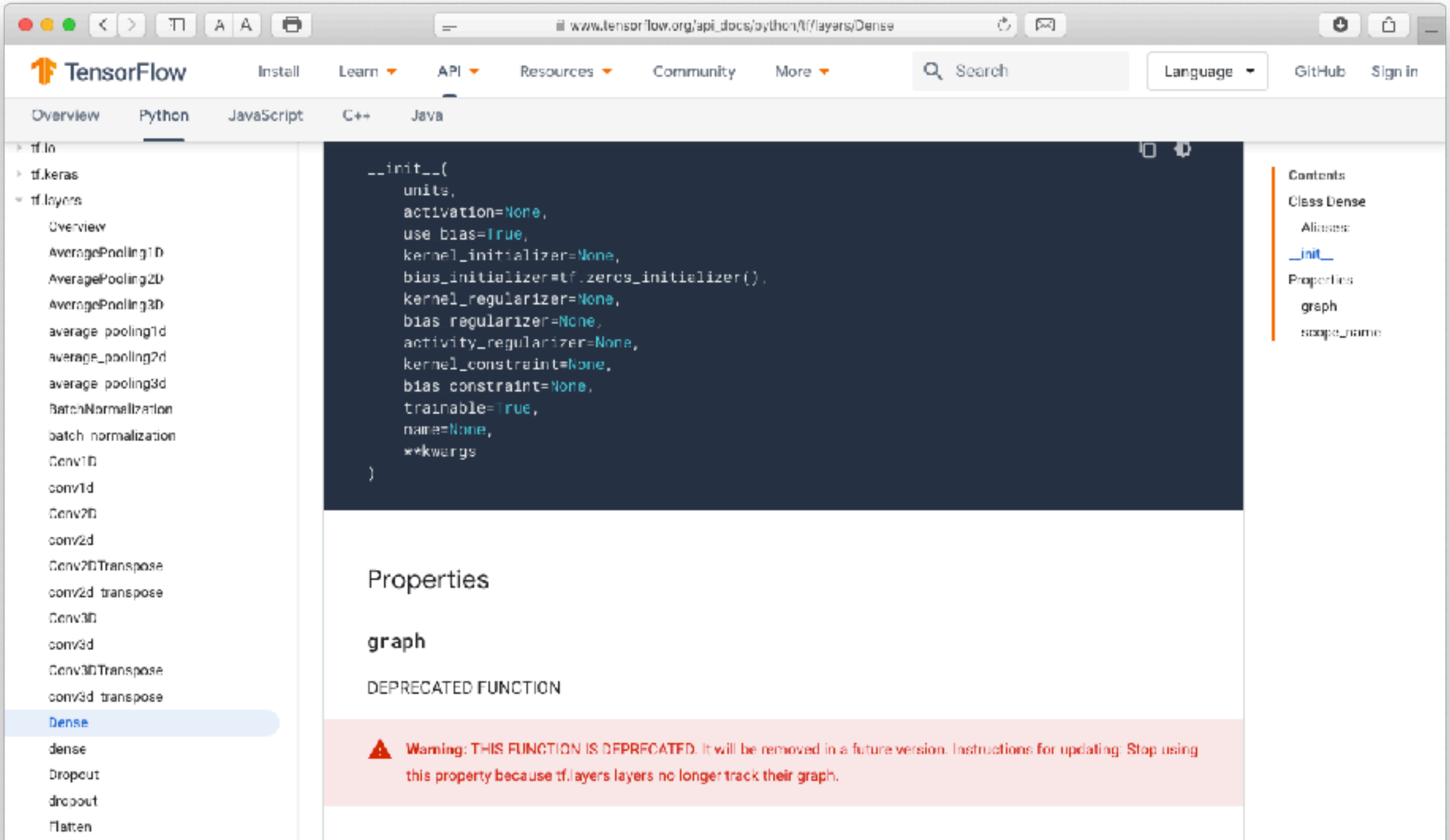
Leer el modelo (stackoverflow)

```
tf.reset_default_graph()
# Create some variables.
v1 = tf.get_variable("v1", shape=[3])
v2 = tf.get_variable("v2", shape=[5])
# Add ops to save and restore all the variables.
saver = tf.train.Saver()
# Later, launch the model, use the saver to restore variables from disk, and
# do some work with the model.
with tf.Session() as sess:
  # Restore variables from disk.
  saver.restore(sess, "/tmp/model.ckpt")
  print("Model restored.")
  # Check the values of the variables
  print("v1 : %s" % v1.eval())
  print("v2 : %s" % v2.eval())
```

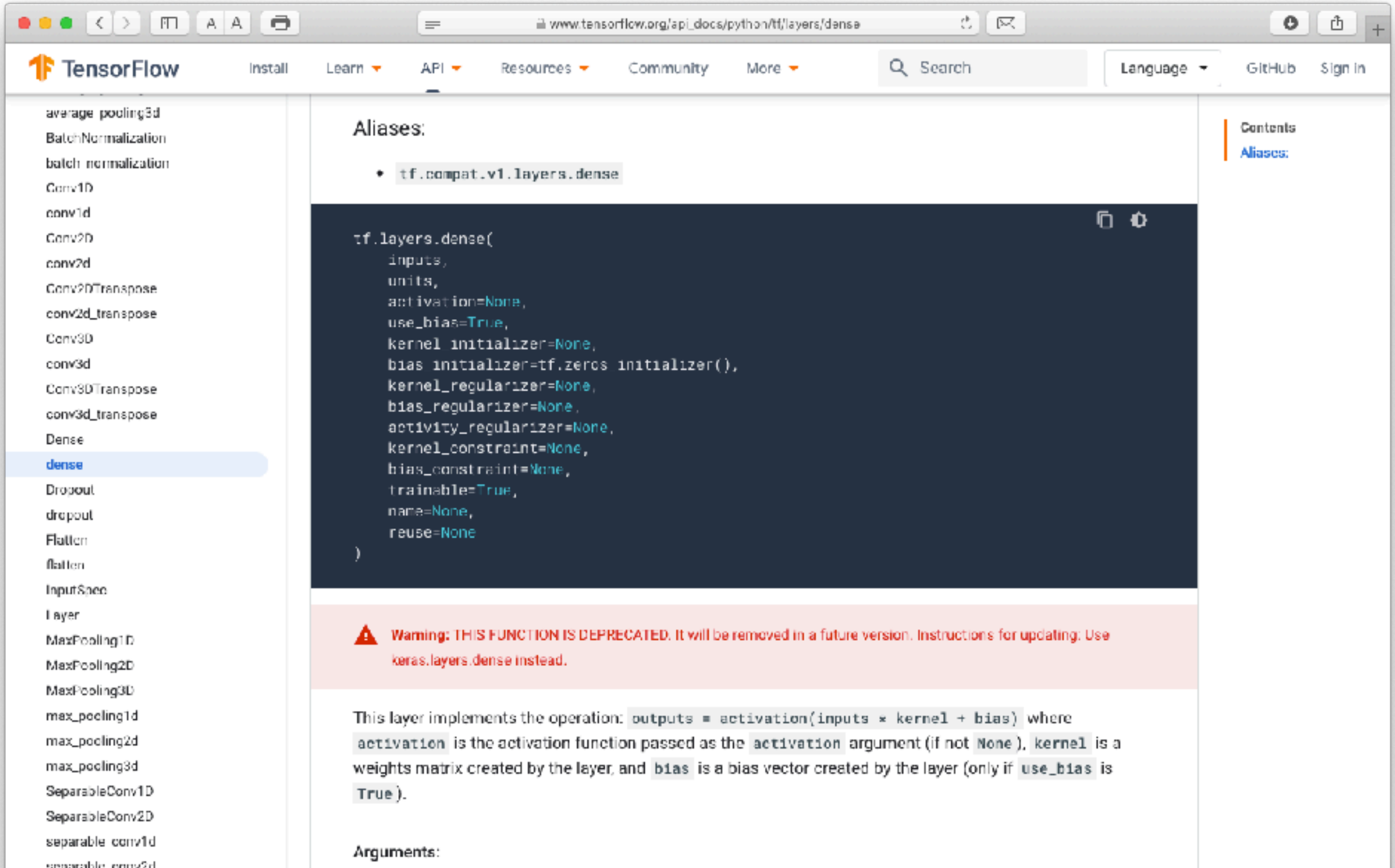
No se inicializan las variables

Capas de alto nivel

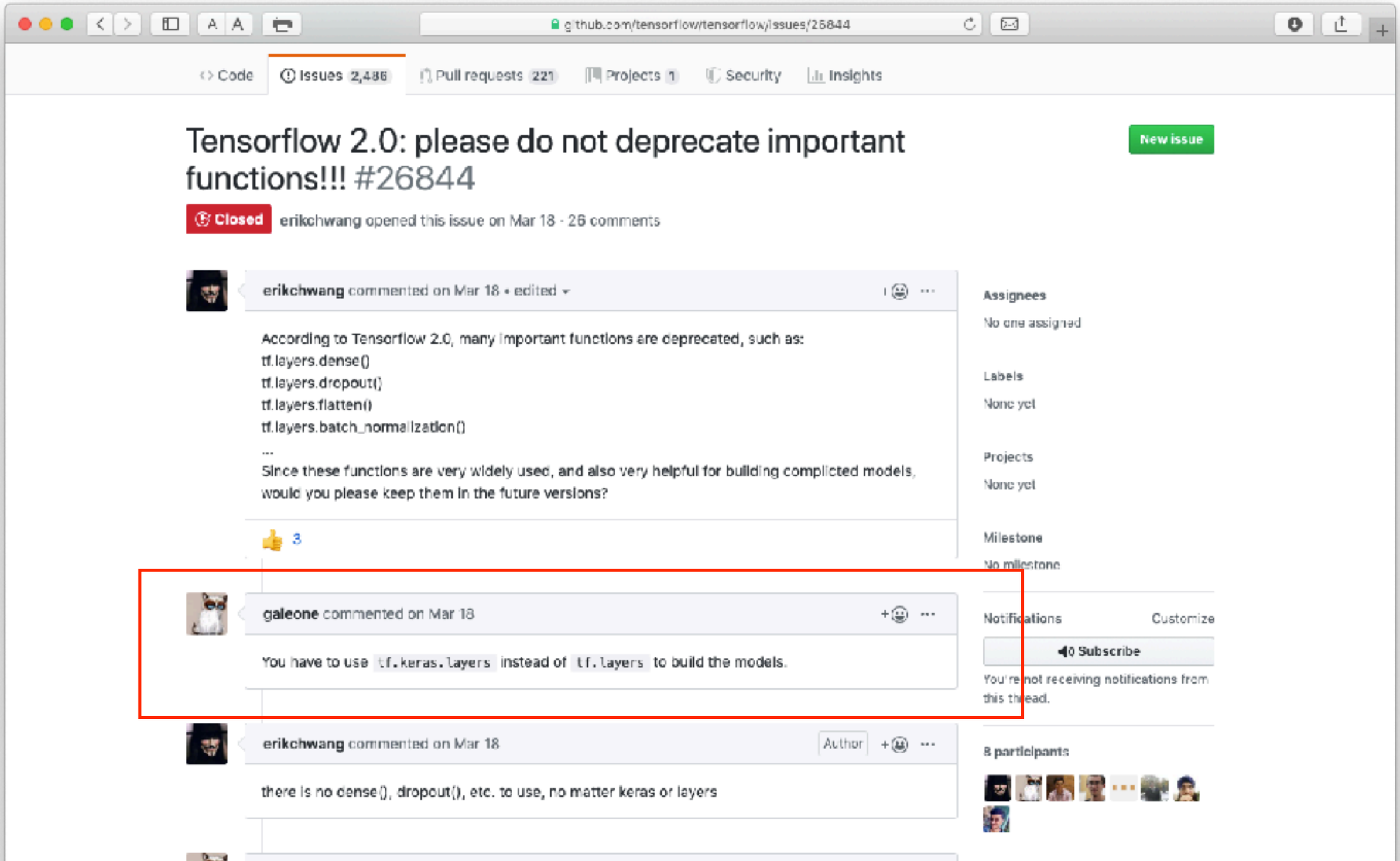
Modulo layer en TF 1.x: tiene dense, conv2d, dropout, etc..



Capas de alto nivel



Capas de alto nivel => Keras

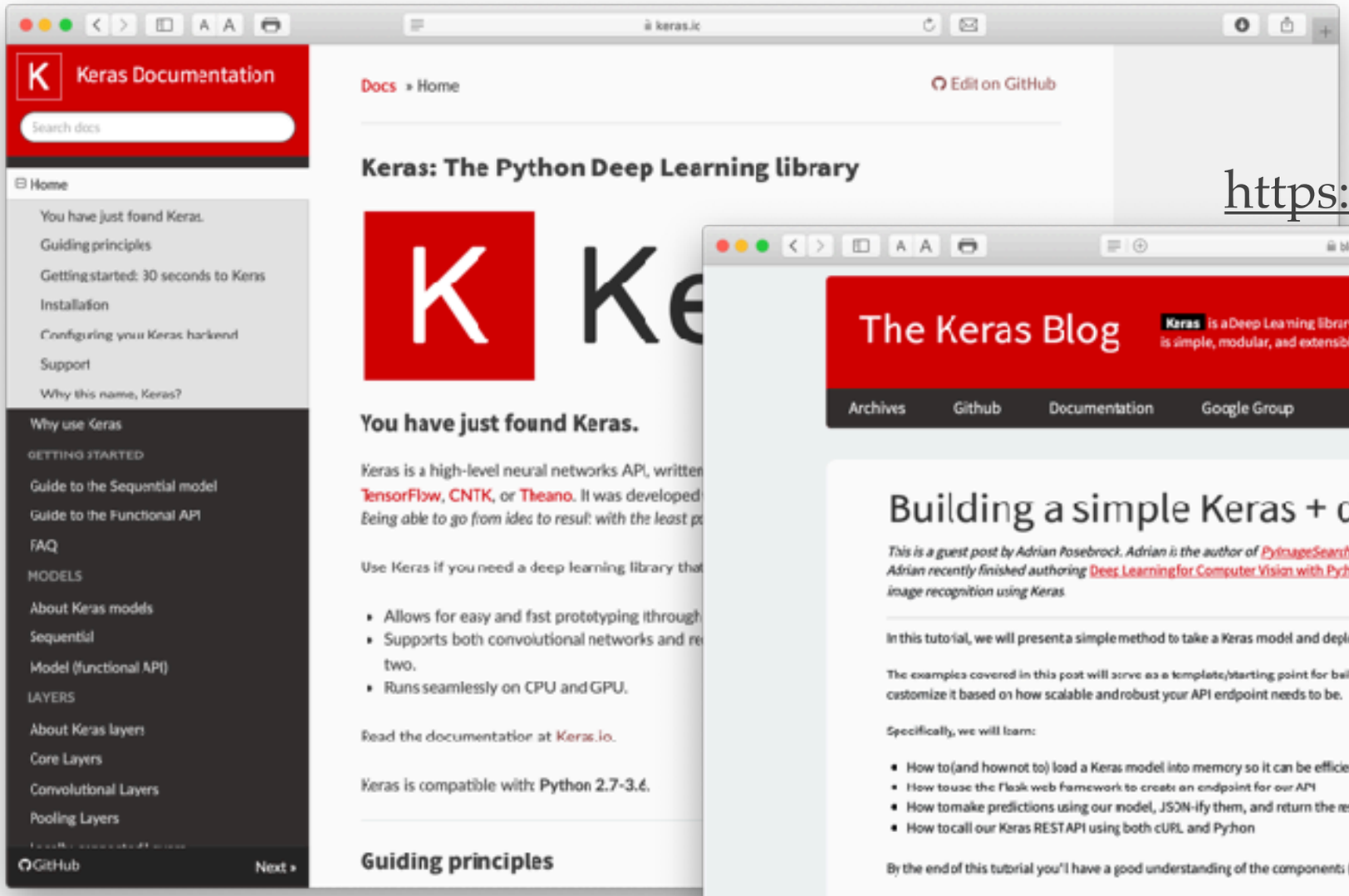


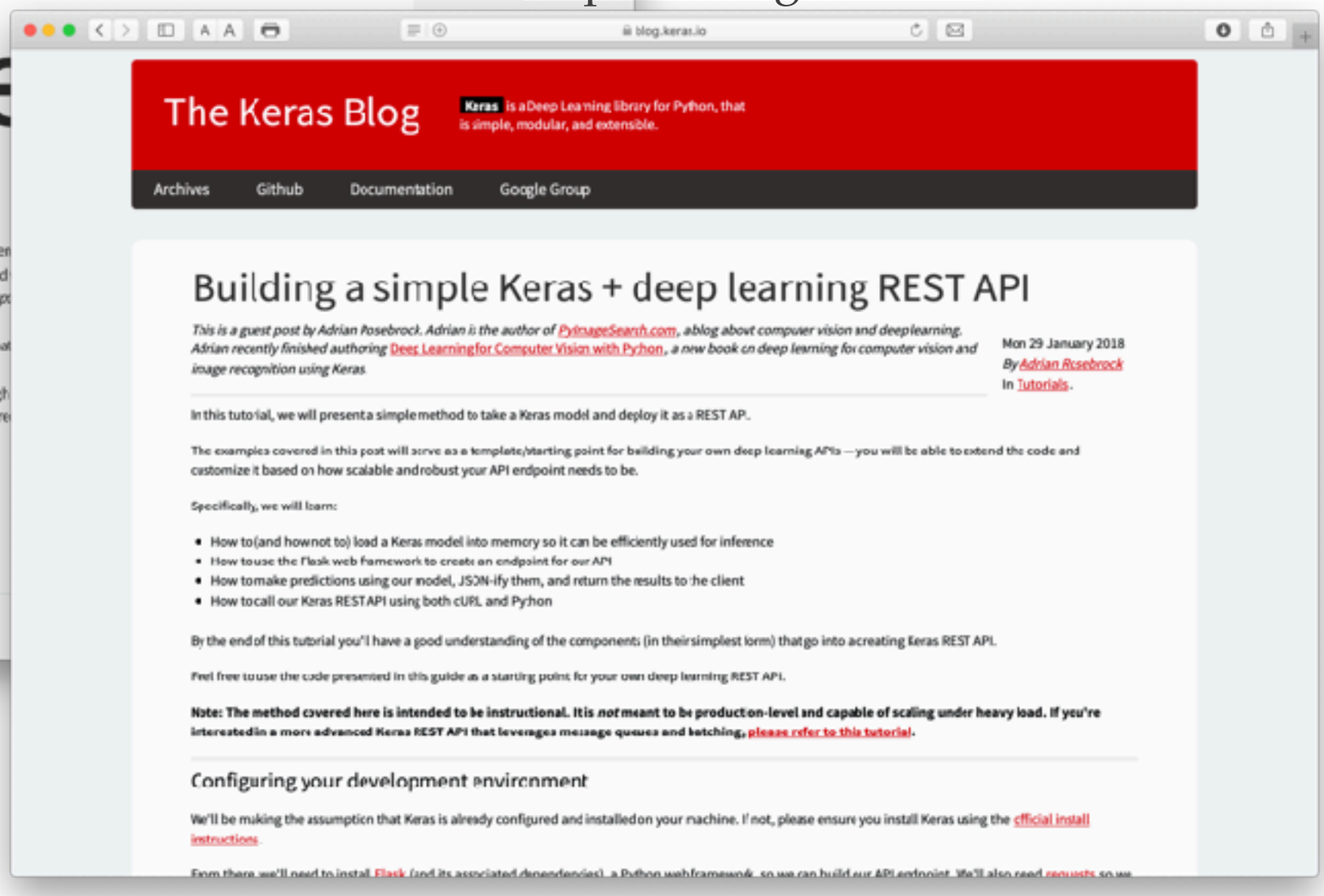
TensorFlow 2.0 with Keras

Keras

https://keras.io

https://blog.keras.io

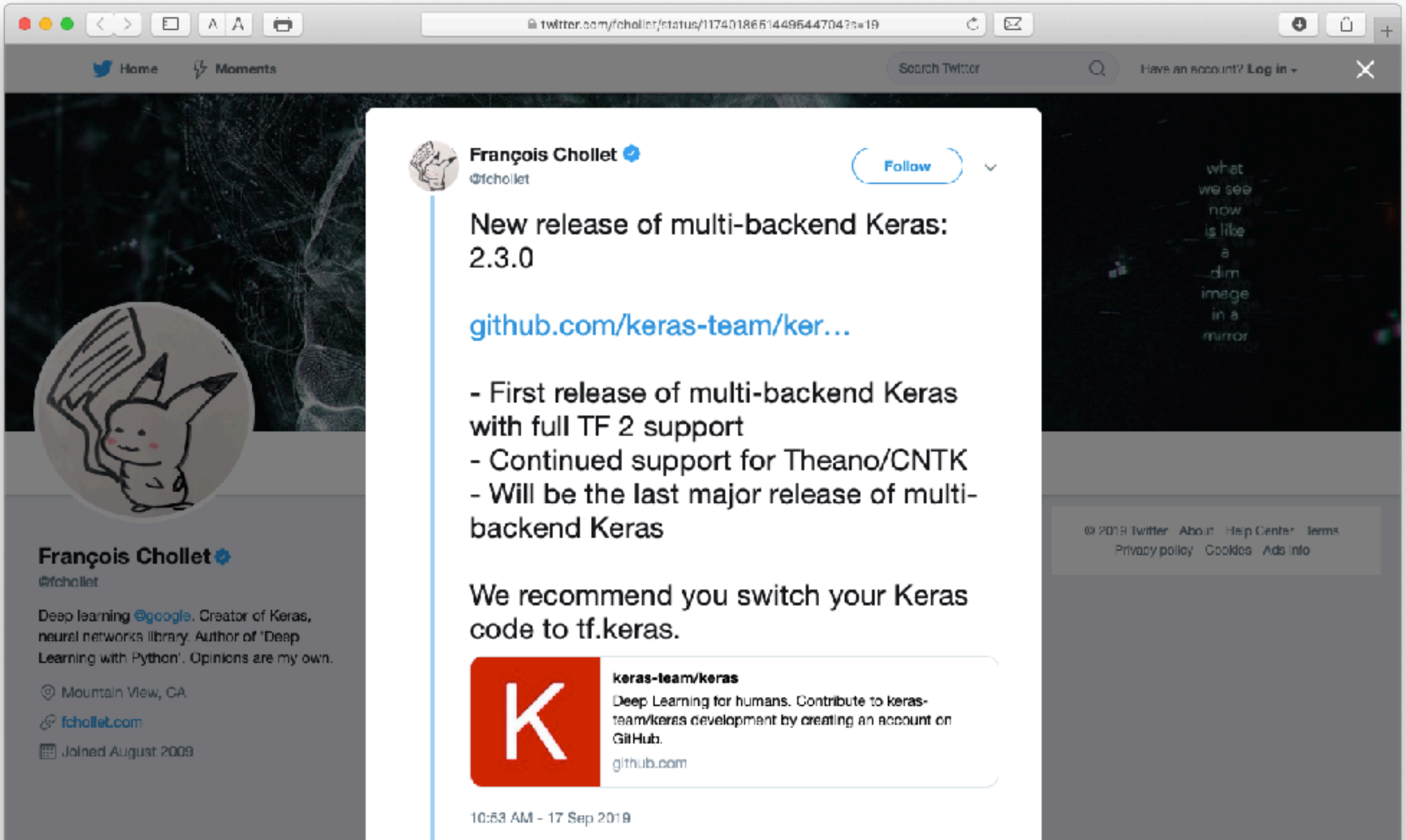




TensorFlow 2.0 and Keras

- keras en tensorflow (tf.keras) esta unido a un solo backend, tensorflow
- keras de https://keras.io es la librería completa que soporta tensorflow, theano y CNTK. Si se va a utilizar con otro backend distinto a TF entonces necesitamos la librería completa

TensorFlow 2.0 with Keras



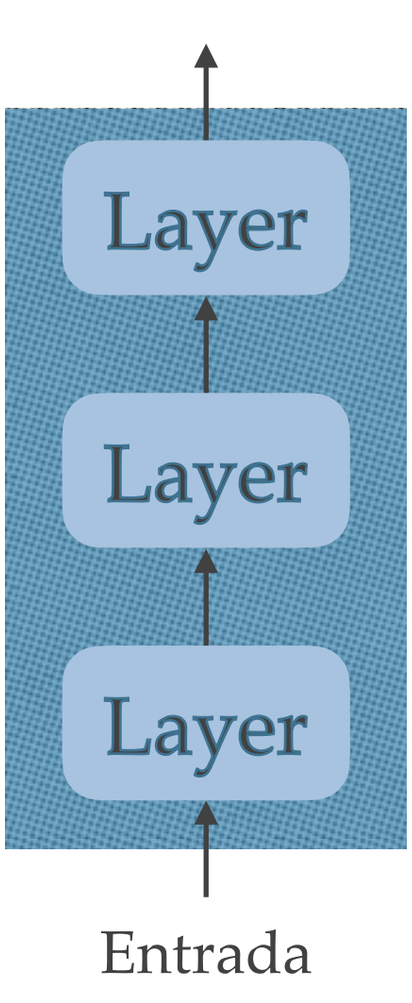
Configuración Keras

```
ariel@gpu-office:~/.keras$ cat keras.json
{
    "floatx": “float32",
    "epsilon": 1e-07,
    "backend": “tensorflow",
    "image_data_format": “channels_last"
}ariel@gpu-office:~/.keras$
```

- `image_data_format`: String, either `"channels_last"` or `"channels_first"`. It specifies which data format convention Keras will follow. (`keras.backend.image_data_format()` returns it.)
- For 2D data (e.g. image), `"channels_last"` assumes `(rows, cols, channels)` while `"channels_first"` assumes `(channels, rows, cols)`.
- For 3D data, `"channels_last"` assumes `(conv_dim1, conv_dim2, conv_dim3, channels)` while `"channels_first"` assumes `(channels, conv_dim1, conv_dim2, conv_dim3)`.
- `epsilon`: Float, a numeric fuzzing constant used to avoid dividing by zero in some operations.
- `floatx`: String, `"float16"`, `"float32"`, or `"float64"`. Default float precision.
- `backend`: String, `"tensorflow"`, `"theano"`, or `"cntk"`.

Sequential vs. Model

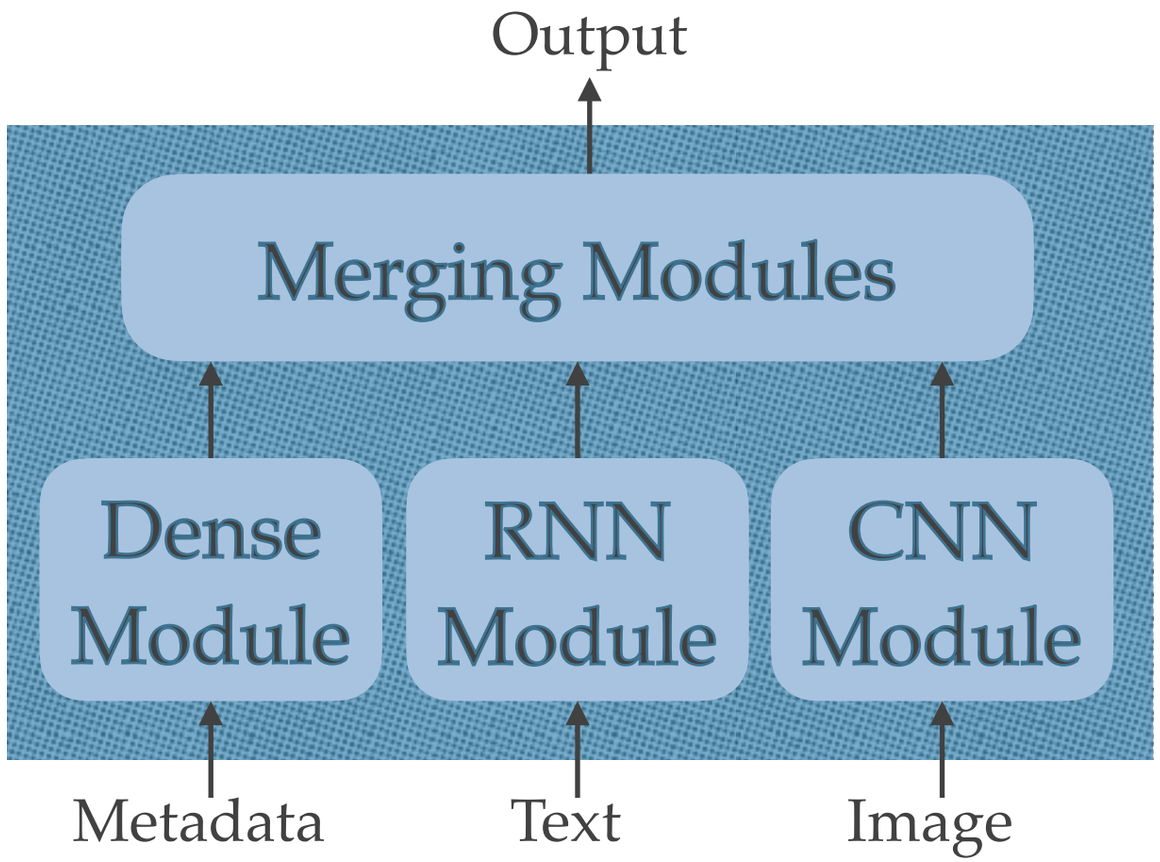
Una entrada y una salida conectada con diversas capas de forma lineal




```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(100, input_shape=(n_dim,)))
model.add(tf.keras.layers.Dense(30,
          activation='linear', use_bias=True))
```

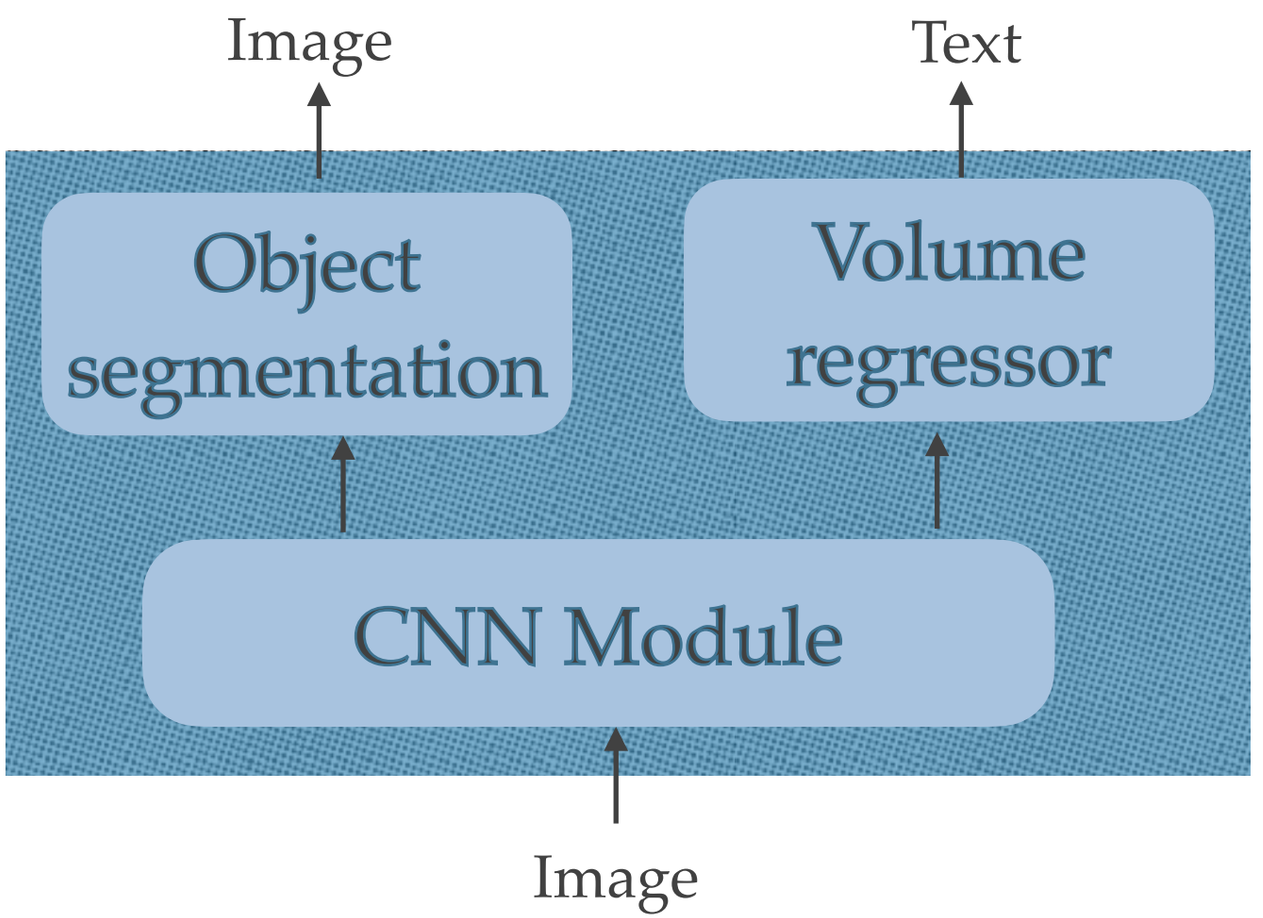
shape, en Input, o input_shape en Dense o Conv no incluyen la dim de batch_size o ejemplos. Es la dimensionalidad del problema, es decir, la de un ejemplo

Sequential vs. Model

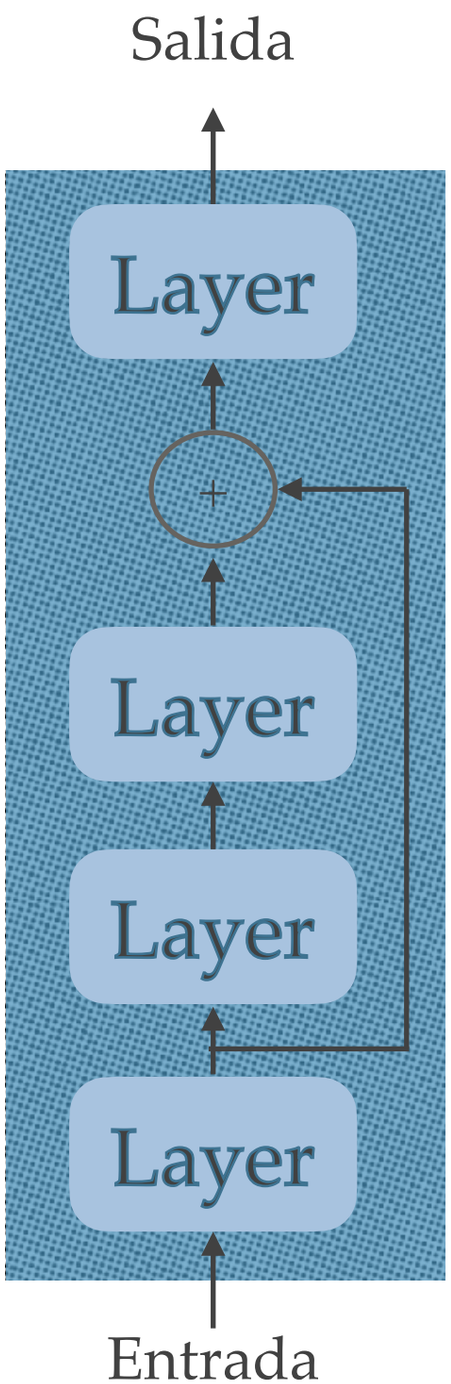


Múltiples entradas

Múltiples salidas

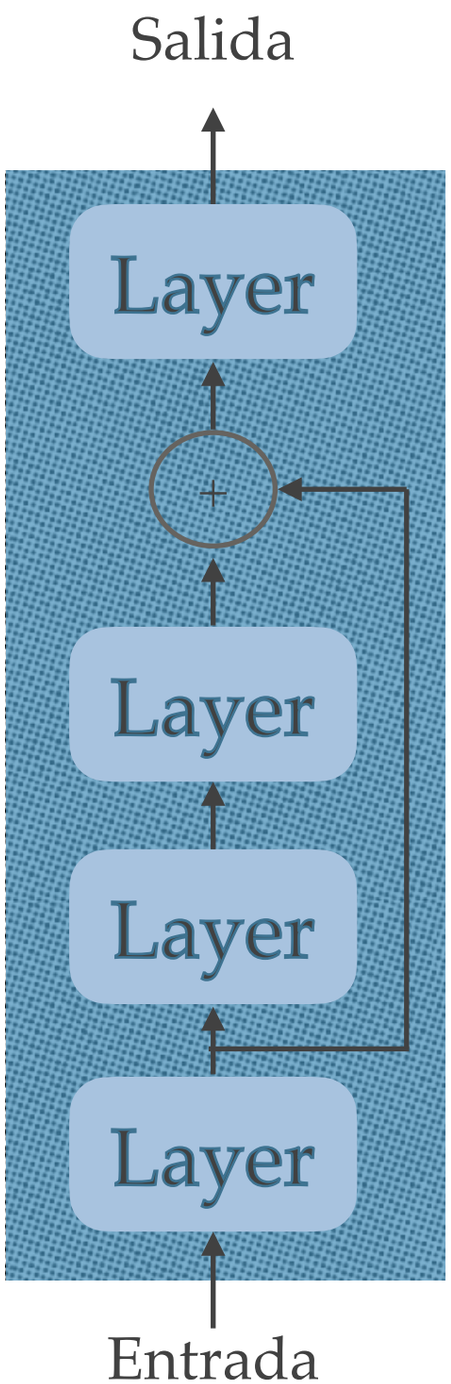


Sequential vs. Model




```
inputs = tf.keras.layers.Input(shape=(28,28,1))
l1 = tf.keras.layers.Dense(100, activation='relu')(inputs)
l2 = tf.keras.layers.Dense(100, activation='relu')(l1)
l3 = tf.keras.layers.Dense(100, activation='relu')(l2)
l4 = tf.keras.layers.Add()([l1,l3])
l5 = tf.keras.layers.Dense(200, activation='relu')(l4)
model = tf.keras.models.Model(inputs=inputs, outputs=l5)
```

model.summary()




```
1 inputs = tf.keras.layers.Input(shape=(28,28,1))
2 l1 = tf.keras.layers.Dense(100, activation='relu')(inputs)
3 l2 = tf.keras.layers.Dense(100, activation='relu')(l1)
4 l3 = tf.keras.layers.Dense(100, activation='relu')(l2)
5 l4 = tf.keras.layers.Add()([l1,l3])
6 l5 = tf.keras.layers.Dense(200, activation='relu')(l4)
7 model = tf.keras.models.Model(inputs=inputs, outputs=l5)
```

```
In [20]: model.summary()
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_3 (InputLayer)         [(None, 28, 28, 1)]       0         
_________________________________________________________________
dense_4 (Dense)              (None, 28, 28, 100)       200       
_________________________________________________________________
dense_5 (Dense)              (None, 28, 28, 100)       10100     
_________________________________________________________________
dense_6 (Dense)              (None, 28, 28, 100)       10100     
_________________________________________________________________
add_1 (Add)                  (None, 28, 28, 100)       0         
_________________________________________________________________
dense_7 (Dense)              (None, 28, 28, 200)       20200     
=================================================================
Total params: 40,600
Trainable params: 40,600
Non-trainable params: 0
_________________________________________________________________
```

tf.keras.layers

- Los layers son clases que al ser invocados como funciones, normalmente con un tensor, devuelven otro tensor
  - Input, Dense, Conv1D (2D, 3D), Activation
  - MaxPooling1D (2D, 3D), AveragePooling1D (2D, 3D)
  - UpSampling1D (2D, 3D)
  - Lambda: convierte cualquier función en un layer
  - Add, Concatenate, Maximum, Minimum

    Estos layers (mayúscula) se encuentran en su versión de función (minúscula)

Regularización

- tf.keras.layers
  - Dropout, BatchNormalization
- tf.keras.regularizers
  - l1, l2, l1_l2

www.tensorflow.org/versions/r1.14/api_docs/python/tf/keras/metrics

TensorFlow Install Learn API Resources Community Why TensorFlow Search Language GitHub Sign in

Overview Python JavaScript C++ Java

- Sequential
- activations
- applications
- backend
- callbacks
- constraints
- datasets
- estimator
- experimental
- initializers
- layers
- losses
- metrics
  - Overview
  - Accuracy
  - AUC
  - BinaryAccuracy
  - BinaryCrossentropy
  - binary_accuracy
  - CategoricalAccuracy
  - CategoricalCrossentropy
  - CategoricalHinge
  - categorical_accuracy
  - CosineSimilarity
  - deserialize
  - FalseNegatives
  - FalsePositives
  - get
  - Hinge
  - KLDivergence
  - LogCoshError
  - Mean
  - MeanAbsoluteError
  - MeanAbsolutePercentageError
  - MeanIoU
  - MeanRelativeError
  - MeanSquaredError
  - MeanSquaredLogarithmicError

Module: tf.keras.metrics

TensorFlow 2.0 version | View source on GitHub

Aliases:

- Module `tf.compat.v1.keras.metrics`

Classes

`class AUC`: Computes the approximate AUC (Area under the curve) via a Riemann sum.

`class Accuracy`: Calculates how often predictions matches labels.

`class BinaryAccuracy`: Calculates how often predictions matches labels.

`class BinaryCrossentropy`: Computes the crossentropy metric between the labels and predictions.

`class CategoricalAccuracy`: Calculates how often predictions matches labels.

`class CategoricalCrossentropy`: Computes the crossentropy metric between the labels and predictions.

`class CategoricalHinge`: Computes the categorical hinge metric between `y_true` and `y_pred`.

`class CosineSimilarity`: Computes the cosine similarity between the labels and predictions.

`class FalseNegatives`: Calculates the number of false negatives.

`class FalsePositives`: Calculates the number of false positives.

`class Hinge`: Computes the hinge metric between `y_true` and `y_pred`.

`class KLDivergence`: Computes Kullback-Leibler divergence metric between `y_true` and `y_pred`.

`class LogCoshError`: Computes the logarithm of the hyperbolic cosine of the prediction error.

`class Mean`: Computes the (weighted) mean of the given values.

Contents
- Aliases:
- Classes
- Functions

Por ser clase, necesitamos crear el objeto

MSE
MAE
Categorical cross entropy
.....

Como veremos luego es posible pasar la métrica como texto al momento de compilar el modelo
Para el caso de **acc** se determina si usar Accuracy o CategoricalAccuracy según los datos

www.tensorflow.org/versions/r1.14/api_docs/python/tf/keras/metrics

TensorFlow | Install | Learn | API | Resources | Community | Why TensorFlow | Search | Language | GitHub | Sign in

- Sequential
- activations
- applications
- backend
- callbacks
- constraints
- datasets
- estimator
- experimental
- initializers
- layers
- losses
- metrics
  - Overview
  - Accuracy
  - AUC
  - BinaryAccuracy
  - BinaryCrossentropy
  - binary_accuracy
  - CategoricalAccuracy
  - CategoricalCrossentropy
  - CategoricalHinge
  - categorical_accuracy
  - CosineSimilarity
  - deserialize
  - FalseNegatives
  - FalsePositives
  - get
  - Hinge
  - KLDivergence
  - LogCoshError
  - Mean
  - MeanAbsoluteError
  - MeanAbsolutePercentageError
  - MeanIoU
  - MeanRelativeError
  - MeanSquaredError
  - MeanSquaredLogarithmicError
  - MeanTensor
  - Metric

`class TruePositives` : Calculates the number of true positives.

Functions

`KLD(...)`

`MAE(...)`

`MAPE(...)`

`MSE(...)`

`MSLE(...)`

`binary_accuracy(...)`

`binary_crossentropy(...)`

`categorical_accuracy(...)`

`categorical_crossentropy(...)` : Computes the categorical crossentropy loss.

`cosine(...)` : Computes the cosine similarity between labels and predictions.

`cosine_proximity(...)` : Computes the cosine similarity between labels and predictions.

`deserialize(...)`

`get(...)`

`hinge(...)` : Computes the hinge loss between `y_true` and `y_pred`.

`kld(...)`

`kullback_leibler_divergence(...)`

`mae(...)`

`mape(...)`

`mean_absolute_error(...)`

`mean_absolute_percentage_error(...)`

Contents
- Aliases:
- Classes
- Functions

Se importan y usan directamente
MSE(y_true, y_pred)

La mayoría de estas funciones son alias de las funciones de costo

www.tensorflow.org/versions/r1.14/api_docs/python/tf/keras/losses

TensorFlow Install Learn API Resources Community Why TensorFlow Search Language GitHub Sign in

- tf.keras
  - Overview
  - Input
  - Model
  - Sequential
  - activations
  - applications
  - backend
  - callbacks
  - constraints
  - datasets
  - estimator
  - experimental
  - initializers
  - layers
  - losses
    - Overview
    - BinaryCrossentropy
    - binary_crossentropy
    - CategoricalCrossentropy
    - CategoricalHinge
    - categorical_crossentropy
    - categorical_hinge
    - cosine
    - CosineSimilarity
    - deserialize
    - get
    - Hinge
    - hinge
    - Huber
    - KLD
    - KLDivergence
    - LogCosh
    - logcosh
    - Loss
    - MAE
    - MAPE
    - MeanAbsoluteError
    - MeanAbsolutePercentageError

Module: tf.keras.losses

TensorFlow 2.0 version | View source on GitHub

Aliases:

- Module `tf.compat.v1.keras.losses`

Classes

Por ser clase, necesitamos crear el objeto

`class BinaryCrossentropy`: Computes the cross-entropy loss between true labels and predicted labels.

`class CategoricalCrossentropy`: Computes the crossentropy loss between the labels and predictions.

`class CategoricalHinge`: Computes the categorical hinge loss between `y_true` and `y_pred`.

`class CosineSimilarity`: Computes the cosine similarity between `y_true` and `y_pred`.

`class Hinge`: Computes the hinge loss between `y_true` and `y_pred`.

`class Huber`: Computes the Huber loss between `y_true` and `y_pred`.

`class KLDivergence`: Computes Kullback Leibler divergence loss between `y_true` and `y_pred`.

`class LogCosh`: Computes the logarithm of the hyperbolic cosine of the prediction error.

`class Loss`: Loss base class.

`class MeanAbsoluteError`: Computes the mean of absolute difference between labels and predictions.

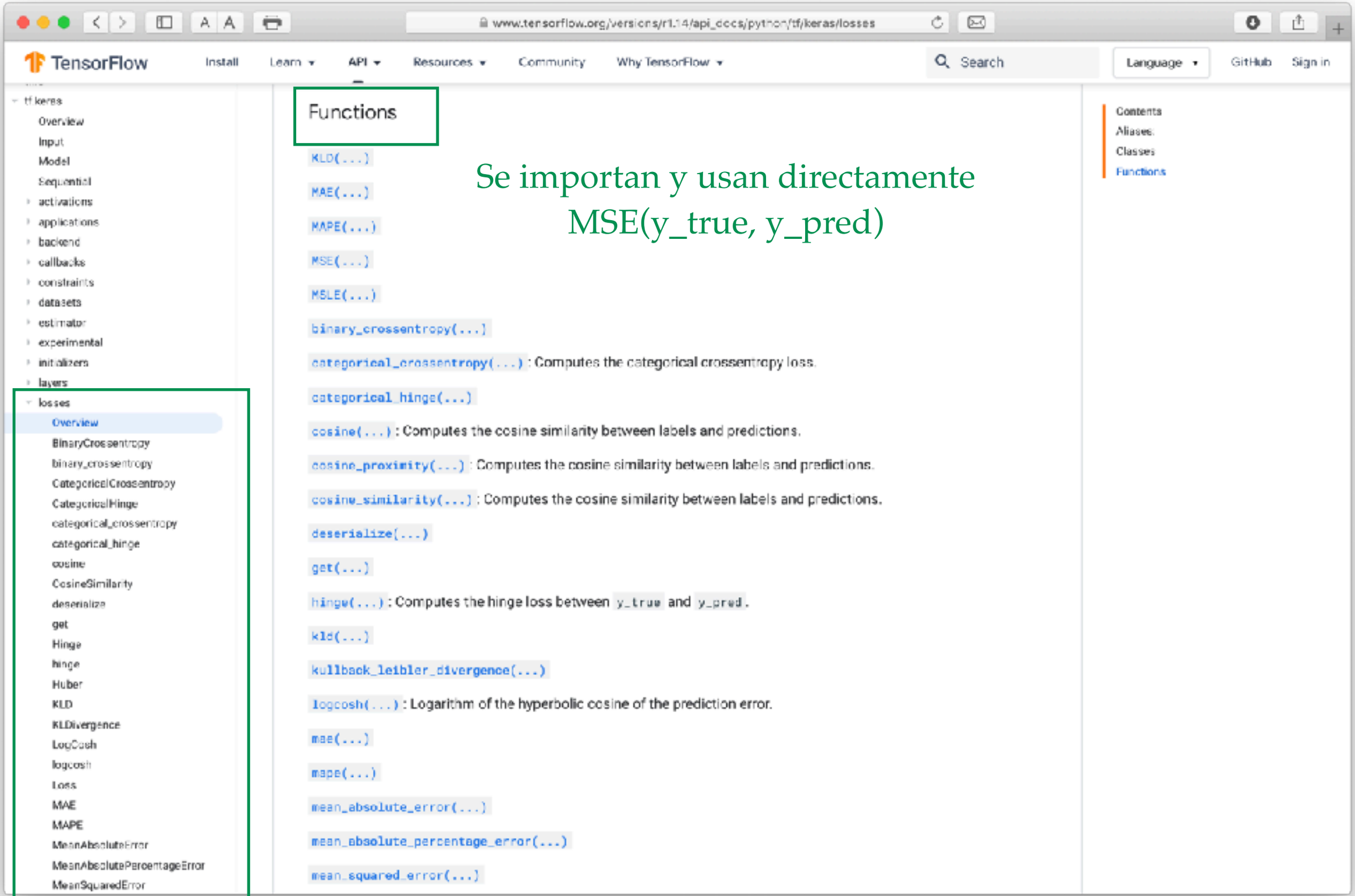
`class MeanAbsolutePercentageError`: Computes the mean absolute percentage error between `y_true` and `y_pred`.

`class MeanSquaredError`: Computes the mean of squares of errors between labels and predictions.

`class MeanSquaredLogarithmicError`: Computes the mean squared logarithmic error between `y_true` and `y_pred`.

`class Poisson`: Computes the Poisson loss between `y_true` and `y_pred`.

Contents
- Aliases
- Classes
- Functions


www.tensorflow.org/versions/r1.14/api_docs/python/tf/keras/losses
TensorFlow
Install
Learn
API
Resources
Community
Why TensorFlow
Search
Language
GitHub
Sign in
tf.keras
Overview
Input
Model
Sequential
activations
applications
backend
callbacks
constraints
datasets
estimator
experimental
initializers
layers
losses
Overview
BinaryCrossentropy
binary_crossentropy
CategoricalCrossentropy
CategoricalHinge
categorical_crossentropy
categorical_hinge
cosine
CosineSimilarity
deserialize
get
Hinge
hinge
Huber
KLD
KLDivergence
LogCosh
logcosh
Loss
MAE
MAPE
MeanAbsoluteError
MeanAbsolutePercentageError
MeanSquaredError
Functions
KLD(...)
MAE(...)
MAPE(...)
MSE(...)
MSLE(...)
binary_crossentropy(...)
categorical_crossentropy(...) : Computes the categorical crossentropy loss.
categorical_hinge(...)
cosine(...) : Computes the cosine similarity between labels and predictions.
cosine_proximity(...) : Computes the cosine similarity between labels and predictions.
cosine_similarity(...) : Computes the cosine similarity between labels and predictions.
deserialize(...)
get(...)
hinge(...) : Computes the hinge loss between y_true and y_pred.
kld(...)
kullback_leibler_divergence(...)
logcosh(...) : Logarithm of the hyperbolic cosine of the prediction error.
mae(...)
mape(...)
mean_absolute_error(...)
mean_absolute_percentage_error(...)
mean_squared_error(...)
Contents
Aliases
Classes
Functions
Se importan y usan directamente
MSE(y_true, y_pred)

Module: tf.keras.optimizers

TensorFlow 2.0 version | View source on GitHub

Aliases:

- Module `tf.compat.v1.keras.optimizers`

Modules

`schedules` module

Classes

`class Adadelta`: Optimizer that implements the Adadelta algorithm.

`class Adagrad`: Optimizer that implements the Adagrad algorithm.

`class Adam`: Optimizer that implements the Adam algorithm.

`class Adamax`: Optimizer that implements the Adamax algorithm.

`class Ftrl`: Optimizer that implements the FTRL algorithm.

`class Nadam`: Optimizer that implements the NAdam algorithm.

`class Optimizer`: Updated base class for optimizers.

`class RMSprop`: Optimizer that implements the RMSprop algorithm.

`class SGD`: Stochastic gradient descent and momentum optimizer.

Compilar, entrenar y predecir

```
13 model = tf.keras.Model(inputs=x, outputs=l2)
14 optimizer = tf.keras.optimizers.SGD(learning_rate=1)
15 model.compile(optimizer, loss=tf.keras.losses.MSE, metrics=[my_acc, 'acc', 'mse'])
16 model.fit(x_data, y_data, epochs=100, validation_data=(x_data, y_data),
17           batch_size=4, verbose=2)
18 y_pred = model.predict(x_data)
```

Multiples outputs: loss y train

- Se usa un diccionario (si definimos nombres en los layers) o una lista para definir la entrada y/o la función de costo para cada salida
- Si solo se define una función de costo, entonces se aplica la misma función para todas las salidas
- La función a minimizar será la suma de las funciones de costo
- Se puede definir la contribución de cada loss mediante el argumento loss_weights al compilar el modelo

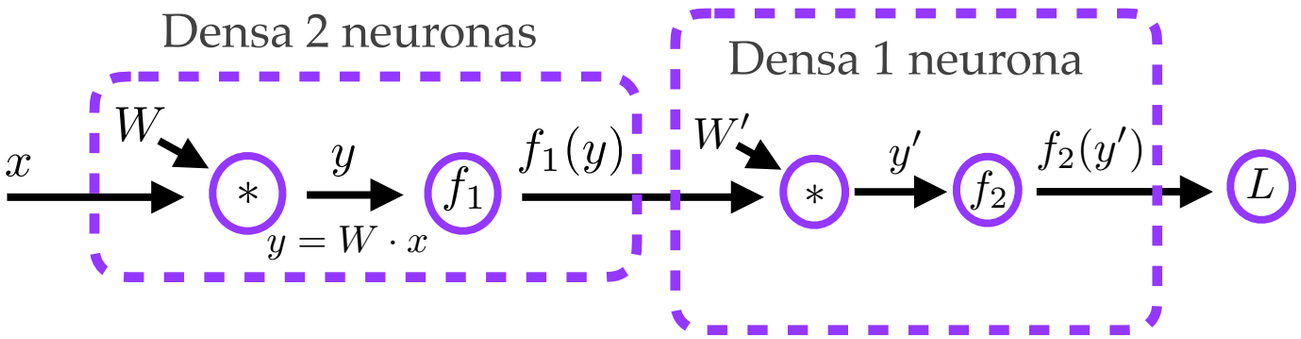
```
model.compile(optimizer='rmsprop', loss='binary_crossentropy',
              loss_weights=[1., 0.2])

model.fit([headline_data, additional_data], [headline_labels, additional_labels],
          epochs=50, batch_size=32)
```

Multiples outputs: loss y train

- Se usa un diccionario (**si definimos nombres en los layers**) o una lista para definir la entrada y/o la función de costo para cada salida
- Si solo se define una función de costo, entonces se aplica la misma función para todas las salidas
- La función a minimizar será la suma de las funciones de costo
- Se puede definir la contribución de cada loss mediante el argumento loss_weights al compilar el modelo

```
model.compile(optimizer='rmsprop',
              loss={'main_output': 'binary_crossentropy', 'aux_output': 'binary_crossentropy'},
              loss_weights={'main_output': 1., 'aux_output': 0.2})

# And trained it via:
model.fit({'main_input': headline_data, 'aux_input': additional_data},
          {'main_output': headline_labels, 'aux_output': additional_labels},
          epochs=50, batch_size=32)
```

Problema 2.6 en Keras



Armamos el grafo
No hay computo

$$f_1(x) = \tanh(x) \qquad f_2(x) = \tanh(x)$$

$$\text{MSE} = L(y') = \frac{1}{N} \sum_i ||y_t^{(i)} - y'^{(i)}||_2^2$$

Entrenamos
Ejecutamos el grafo y
modificamos las variables

```python
def my_acc(y_true, y_pred):
    acc = tf.reduce_mean(tf.cast(tf.less_equal(tf.abs(y_true-y_pred),0.1), tf.float32))
    return acc

x = tf.keras.layers.Input(shape=(2,))
l1 = tf.keras.layers.Dense(units=2, activation='tanh', use_bias=True,
        kernel_regularizer=tf.keras.regularizers.l2(1e-3))(x)
l2 = tf.keras.layers.Dense(units=1, activation='tanh', use_bias=True)(l1)
model = tf.keras.Model(inputs=x, outputs=l2)
optimizer = tf.keras.optimizers.SGD(learning_rate=1)
model.compile(optimizer, loss=tf.keras.losses.MSE, metrics=[my_acc, 'acc', 'mse'])
model.fit(x_data, y_data, epochs=100, validation_data=(x_data, y_data),
        batch_size=4, verbose=2)
y_pred = model.predict(x_data)
```

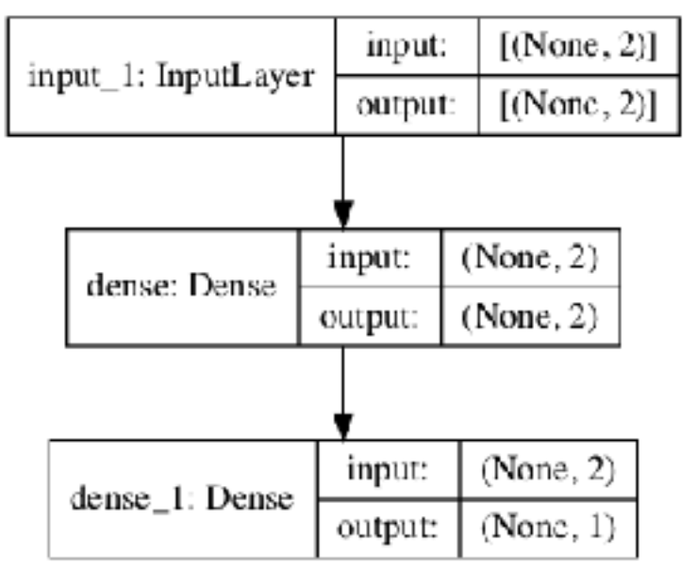
Problema 2.6 en Keras

- Observemos el modelo con el método summary()

```
In [12]: model.summary()
Model: “model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 2)]               0
_________________________________________________________________
dense (Dense)                (None, 2)                 6
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 3
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

- Exportamos el grafo

```
433 from tensorflow.keras.utils import plot_model
434 plot_model(model, show_shapes=True, to_file= ‘model.png’)
```



www.tensorflow.org/versions/r1.14/api_docs/python/tf/keras/callbacks

TensorFlow Install Learn API Resources Community Why TensorFlow Search Language GitHub Sign in

- tf.graph_util
- tf.image
- tf.initializers
- tf.io
- tf.keras
  - Overview
  - Input
  - Model
  - Sequential
  - activations
  - applications
  - backend
  - callbacks
    - Overview
    - BaseLogger
    - Callback
    - CSVLogger
    - EarlyStopping
    - History
    - LambdaCallback
    - LearningRateScheduler
    - ModelCheckpoint
    - ProgbarLogger
    - ReduceLROnPlateau
    - RemoteMonitor
    - TensorBoard
    - TerminateOnNaN
  - constraints
  - datasets
  - estimator
  - experimental
  - initializers
  - layers
  - losses
  - metrics
  - mixed_precision
  - models
  - optimizers
  - preprocessing

Module: tf.keras.callbacks

TensorFlow 2.0 version | View source on GitHub

Contents
Aliases:
Classes

Aliases:

- Module `tf.compat.v1.keras.callbacks`

Classes

`class BaseLogger`: Callback that accumulates epoch averages of metrics.

`class CSVLogger`: Callback that streams epoch results to a csv file.

`class Callback`: Abstract base class used to build new callbacks.

`class EarlyStopping`: Stop training when a monitored quantity has stopped improving.

`class History`: Callback that records events into a `History` object.

`class LambdaCallback`: Callback for creating simple, custom callbacks on-the-fly.

`class LearningRateScheduler`: Learning rate scheduler.

`class ModelCheckpoint`: Save the model after every epoch.

`class ProgbarLogger`: Callback that prints metrics to stdout.

`class ReduceLROnPlateau`: Reduce learning rate when a metric has stopped improving.

`class RemoteMonitor`: Callback used to stream events to a server.

`class TensorBoard`: Enable visualizations for TensorBoard.

`class TerminateOnNaN`: Callback that terminates training when a NaN loss is encountered.

Nos permiten interactuar con el modelo durante el entrenamiento, por ejemplo, para ajustar dinámicamente algunos parámetros como el learning rate

Callbacks are passed to the model via the callbacks argument in fit, which takes a list of callbacks. You can pass any number of callbacks.

```
import keras

callbacks_list = [
    keras.callbacks.EarlyStopping(
        monitor='acc',
        patience=1,
    ),
    keras.callbacks.ModelCheckpoint(
        filepath='my_model.h5',
        monitor='val_loss',
        save_best_only=True,
    )
]

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

model.fit(x, y,
          epochs=10,
          batch_size=32,
          callbacks=callbacks_list,
          validation_data=(x_val, y_val))
```

Interrupts training when improvement stops

Monitors the model's validation accuracy

Interrupts training when accuracy has stopped improving for more than one epoch (that is, two epochs)

Saves the current weights after every epoch

Path to the destination model file

These two arguments mean you won't overwrite the model file unless val_loss has improved, which allows you to keep the best model seen during training.

You monitor accuracy, so it should be part of the model's metrics.

Note that because the callback will monitor validation loss and validation accuracy, you need to pass validation_data to the call to fit.

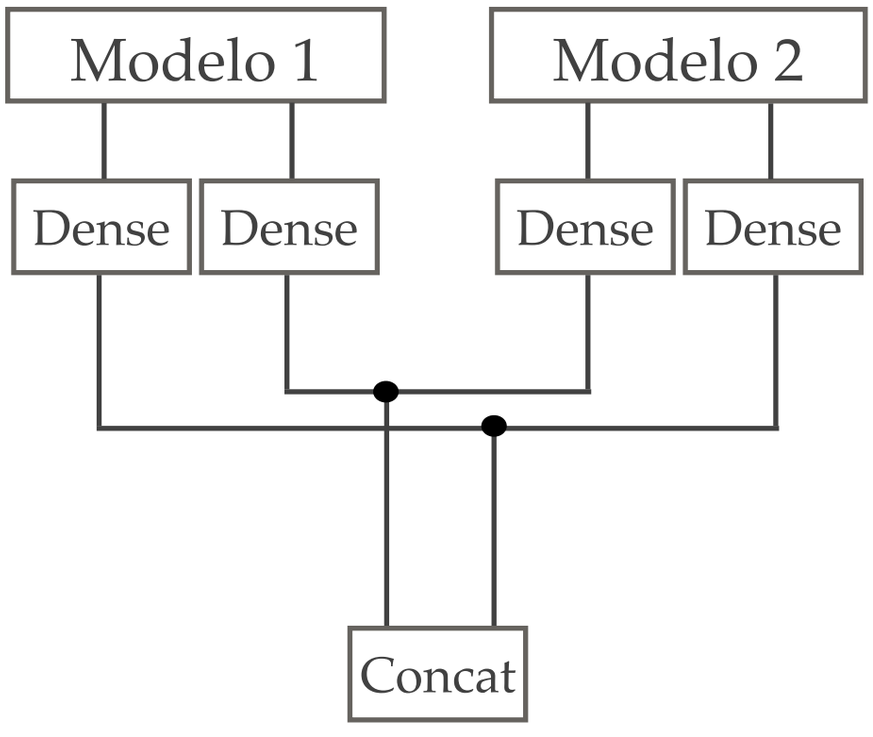
Crear un nuevo layer

- Lambda: útil para hacer operaciones aritméticas

```
import tensorflow.keras.backend as K
def antirectifier(x):
    x -= K.mean(x, axis=1, keepdims=True)
    x = K.l2_normalize(x, axis=1)
    pos = K.relu(x)
    neg = K.relu(-x)
    return K.concatenate([pos, neg], axis=1)

model.add(tf.keras.layers.Lambda(antirectifier))
l10 = tf.keras.layers.Lambda(antirectifier)(li)
```

Crear un nuevo layer

- Lambda mezclando TF




```python
def cross(l1, l2):
    return tf.linalg.cross(l1,l2)

d1 = tf.keras.layers.Dense(64)(Model_1.output)
d2 = tf.keras.layers.Dense(64)(Model_1.output)
d3 = tf.keras.layers.Dense(64)(Model_2.output)
d4 = tf.keras.layers.Dense(64)(Model_2.output)

cross1 = tf.keras.layers.Lambda(cross)([d1,d4])
cross2 = tf.keras.layers.Lambda(cross)([d2,d3])

output = tf.keras.layers.Concatenate()([cross1,cross2])
```

Crear un nuevo layer

- Clase Layer con múltiples entradas y salidas

```
 3 from tensorflow.keras import backend as K
 4 from tensorflow.keras.layers import Layer
 6 class MyLayer(Layer):
 8     def __init__(self, output_dim, **kwargs):
 9         self.output_dim = output_dim
10         super(MyLayer, self).__init__(**kwargs)
11
12     def build(self, input_shape):
13         assert isinstance(input_shape, list)
14         # Create a trainable weight variable for this layer.
15         self.kernel = self.add_weight(name='kernel',
16                                       shape=(input_shape[0][1], self.output_dim),
17                                       initializer='uniform',
18                                       trainable=True)
19         super(MyLayer, self).build(input_shape)  # Be sure to call this at the end
10
21     def call(self, x):
22         assert isinstance(x, list)
23         a, b = x
24         return [K.dot(a, self.kernel) + b, K.mean(b, axis=-1)]
25
26     def compute_output_shape(self, input_shape):
27         assert isinstance(input_shape, list)
28         shape_a, shape_b = input_shape
29         return [(shape_a[0], self.output_dim), shape_b[:-1]]
```

Se definen las variables internas del layer

se puede hacer a mano con el backend K.variable()

Se definen la lógica del layer

Si se modifica el tamaño de la entrada hay que especificarlo

Crear una métricas

y_true e y_pred son **tensores**

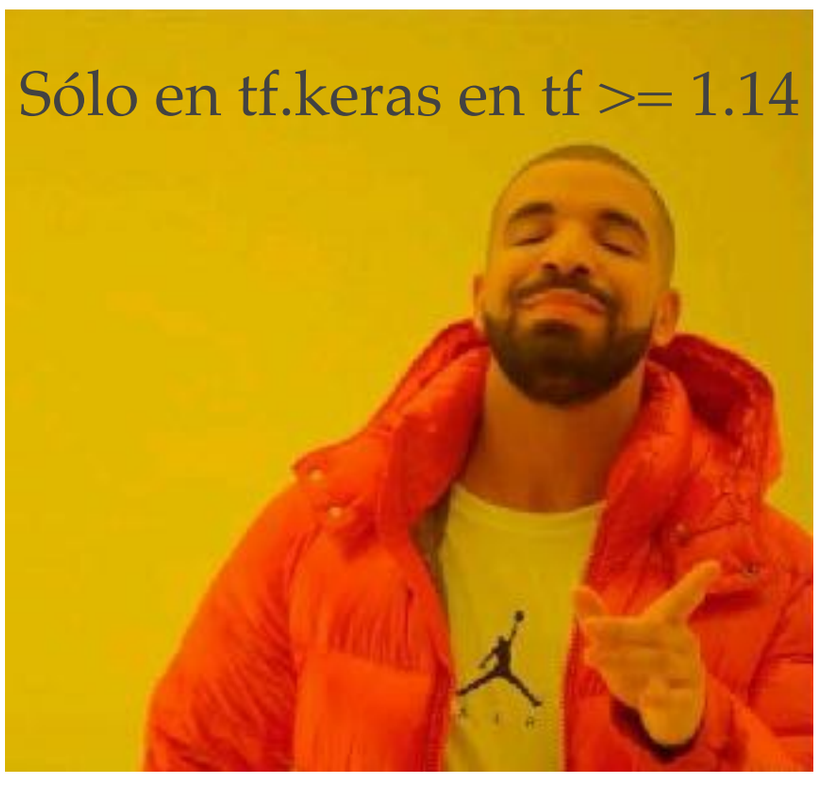
```python
def my_metric(y_true, y_pred):
    val = . . . . .
    return val
```

```python
class MyMetric(object):
    def __init__(self, a1, a2, a3, *args):
        self.a1 = a1
        self.__name__ = 'MyMetric'

        pass

    def __call__(self, y_true, y_pred):
        if self.a1 is not None:
            val = ...
            pass
        else:
            val = ...
            pass

        return val
```

Crear una métricas

y_true e y_pred son **tensores**

```
def my_metric(y_true, y_pred):
    val = . . . . .
    return val
```




```
class BinaryTruePositives(tf.keras.metrics.Metric):
  def __init__(self, name='binary_true_positives', **kwargs):
    super(BinaryTruePositives, self).__init__(name=name,
                                              **kwargs)
    self.true_positives = self.add_weight(name='tp',
                                          initializer='zeros')

  def update_state(self, y_true, y_pred, sample_weight=None):
    y_true = tf.cast(y_true, tf.bool)
    y_pred = tf.cast(y_pred, tf.bool)
    values = tf.logical_and(tf.equal(y_true, True),
                            tf.equal(y_pred, True))
    values = tf.cast(values, self.dtype)
    if sample_weight is not None:
      sample_weight = tf.cast(sample_weight, self.dtype)
      sample_weight = tf.broadcast_weights(sample_weight,
                                           values)
      values = tf.multiply(values, sample_weight)
    self.true_positives.assign_add(tf.reduce_sum(values))

  def result(self):
    return self.true_positives
```

Crear una función de costo

y_true e y_pred son **tensores**

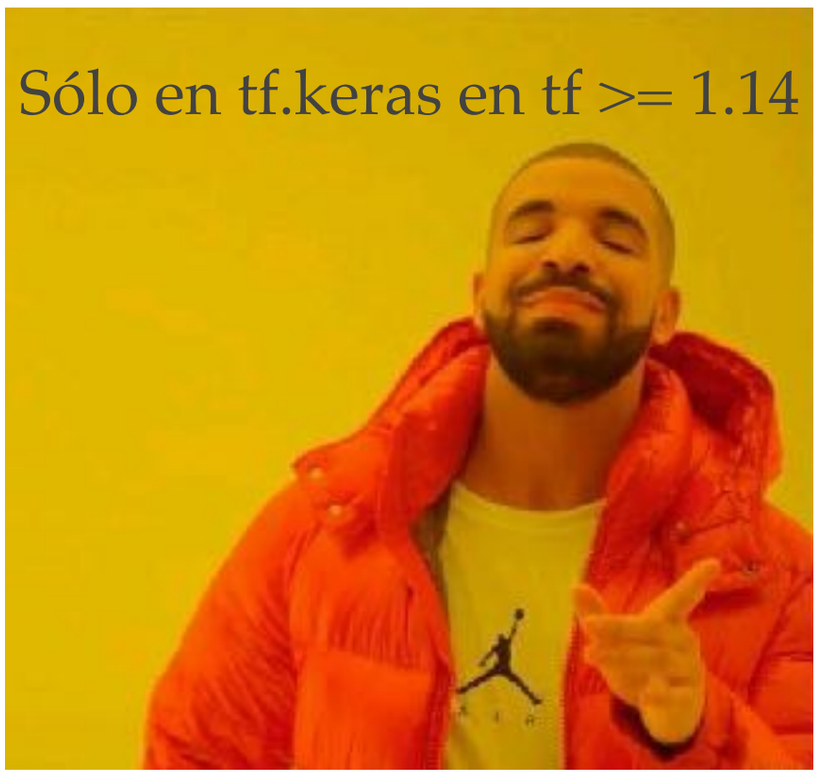
```python
def my_loss(y_true, y_pred):
    val = . . . . .
    return val
```

```python
class MyLoss(object):
    def __init__(self, a1, a2, a3, *args):
        self.a1 = a1
        self.__name__ = 'MyMetric'

        pass

    def __call__(self, y_true, y_pred):
        if self.a1 is not None:
            val = ...
            pass
        else:
            val = ...
            pass

        return val
```

Crear una función de costo

y_true e y_pred son **tensores**

```
5 def my_loss(y_true, y_pred):
6     val = . . . . .
7     return val
```



al ser llamados desde model.fit() y_true e y_pred son **tensores**

```
1 class MeanSquaredError(tf.keras.looses.Loss):
2   def call(self, y_true, y_pred):
3     y_pred = tf.convert_to_tensor(y_pred)
4     y_true = tf.cast(y_pred, y_pred.dtype)
5     values = tf.logical_and(tf.equal(y_true, True),
6                             tf.equal(y_pred, True))
7     return  K.mean(K.square(y_pred-y_true), axis=-1)
```

Crear un nuevo callback

Hay que heredar de la clase Callback e implementar (los que se necesiten) los métodos:

- set_model
- on_epoch_begin, on_epoch_end
- on_batch_begin, on_batch_end
- on_train_begin, on_train_end

```python
import os
import tensorflow.keras as keras
class ActivationLogger(keras.callbacks.Callback):

    def __init__(self, filepath, layer2save=None):
        self.filepath = filepath
        self.layer2save = layer2save

    def set_model(self, model):
        self.model = model
        if self.layer2save is not None:
            layer_outputs = model.layers[self.layer2save]
        else:
            layer_outputs = [layer.output for layer in model.layers]
        self.activation_model = keras.models.Model(model.input,
                                                   layer_outputs)

    def on_peoch_end(self, epoch, logs=None):
        if self.validation_data is None:
            raise RuntimeError('Requires validation data')

        validation_sample = self.validation_data[0][0:1]
        activations = self.activations_model.predict(validation_sample)
        fname = 'activation_at_epoch_' + str(epoch)
        if self.layer2save is not None:
            fname += '_layer_' + str(self.layer2save)
        f = open(os.path.join(self.filepath, fname + '.npy'), 'w')
        np.savez(f, activations)
        f.close()
```

Llamado antes de entrenar para informar que modelo se utiliza

Armamos un modelo para retornar la activación

Keras + TensorBoard

- Definir el path donde se va a almacenar la información de log de TensorBoard, si no existe se crea el directorio
- Usamos el callback de Keras

```
callbacks = [
        tf.keras.callbacks.TensorBoard(
            log_dir='keras_tensorboard',
            histogram_freq=1,)]

model.fit(x_data, y_data, epochs=200, validation_data=(x_data, y_data),
        batch_size=4, callbacks=callbacks, verbose=2)
```

- Iniciamos TensorBoard

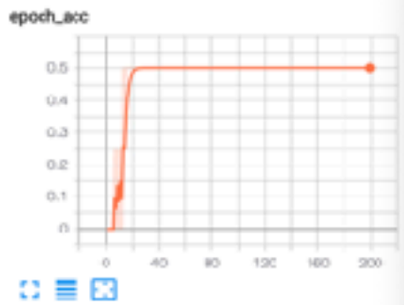
```
ariel@Goku:~/Docencia/Balseiro/Materias/CV_DeepLearning/Teoria/09$ tensorboard --logdir=keras_tensorboard/
TensorBoard 1.14.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

TensorBoard SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS PROFILE

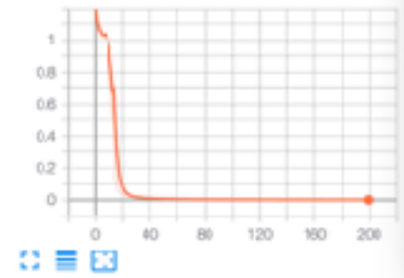
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis STEP RELATIVE WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
keras_tensorboard/

Filter tags (regular expressions supported)

epoch_acc



epoch_loss



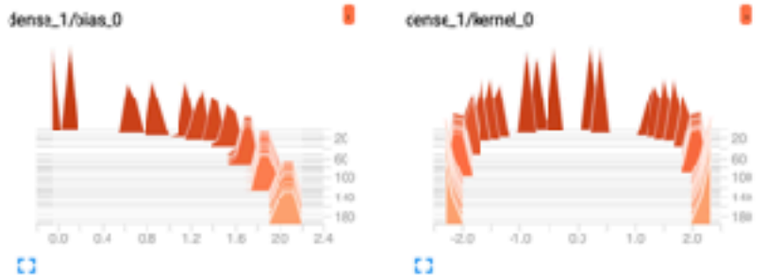
epoch_mean_squared_error

TensorBoard SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS PROFILE INACTIVE

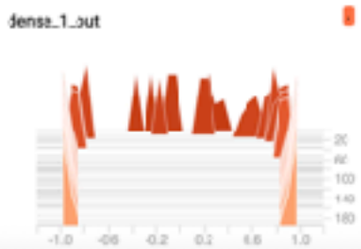
Histogram mode OVERLAY OFFSET
Offset time axis STEP RELATIVE WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
keras_tensorboard/

Filter tags (regular expressions supported)

dense_1 2

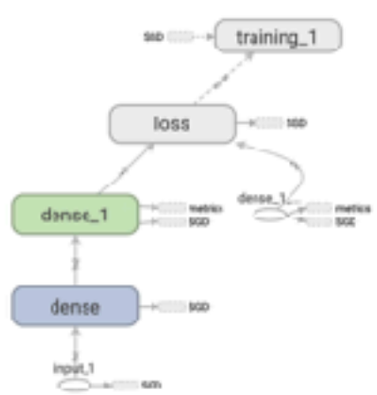


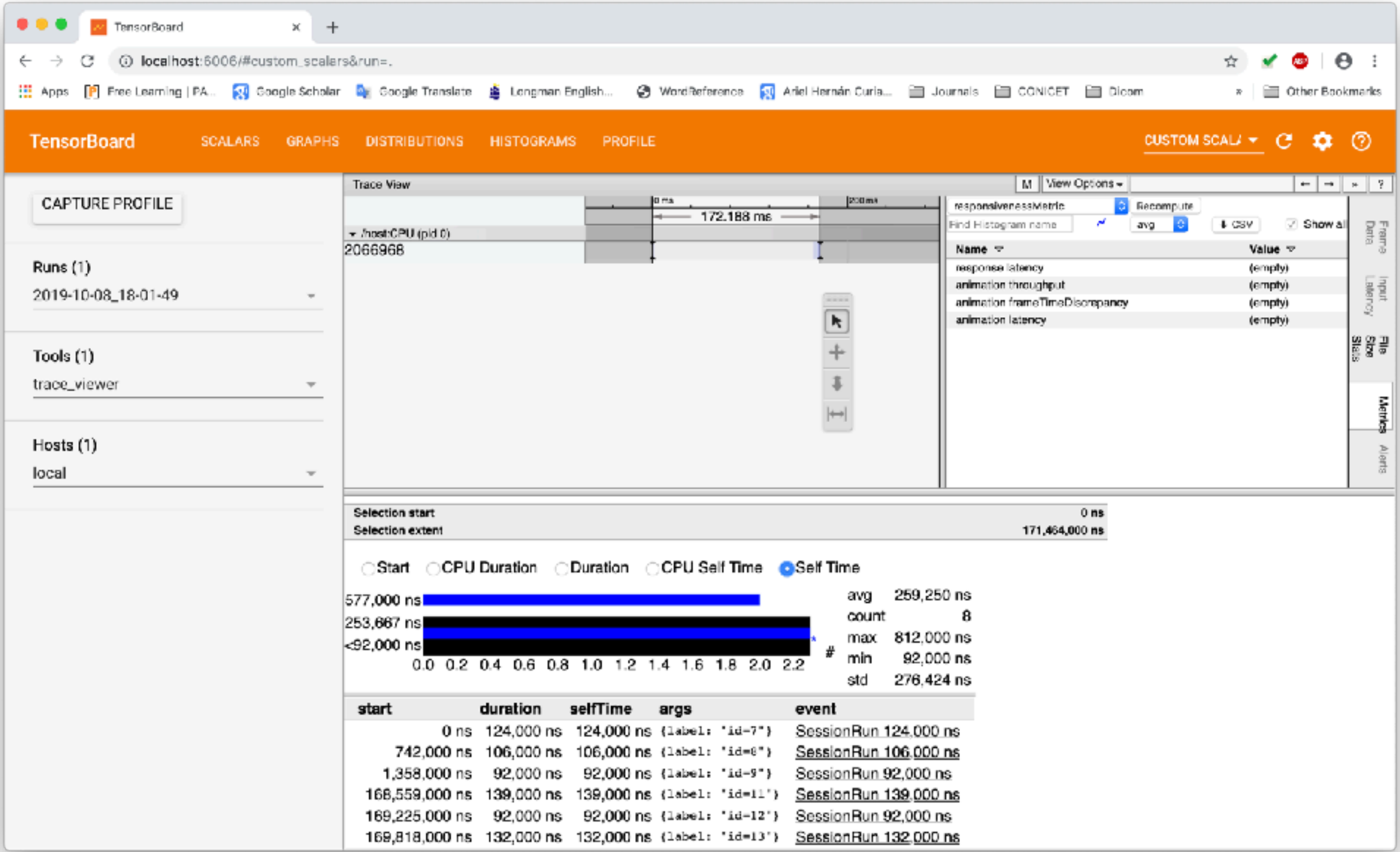
dense_1_out 1



TensorBoard SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS PROFILE INACTIVE

Search nodes. Regexes supported.
Fit to Screen
Download PNG
Run (1)
Tag (1) Default
Upload Choose File
Graph
Conceptual Graph
Profile
Trace inputs
Color Structure
Device
XLA Cluster
Compute time
Memory
Close legend.
Graph (* = expandable)
Namespace* ?
OpNode ?
Unconnected series* ?
Connected series* ?
Constant ?
Summary ?
Dataflow edge ?
Control dependency edge ?
Reference edge ?





Usando Chrome se puede obtener la información de profiling

Leer y guardar los modelos

- Todo el modelo

```
434 model.save('my_model.h5')
```

```
433 from tensorflow.keras.models import load_model
434 model = load_model('my_model.h5')
```

- Solamente los pesos

```
1 model.save_weights('my_model_weights.h5')
```

```
4 model.load_weights('my_model_weights.h5')
```

Visualización de datos intermedios

Mapa de activación

```
import tensorflow as tf
from tensorflow.keras import preprocessing
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras import backend as K

model = tf.keras.applications.VGG16(weights='imagenet', include_top='False')

img_path = 'fig/elephant.jpg'
img = preprocessing.image.load_img(img_path, target_size=(224, 224))

x = preprocessing.image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

layer_name = 'block4_conv1'
layer = model.get_layer(layer_name)
activation = layer.output

run = K.function([model.input], [activation])
I = run([x])[0]
```

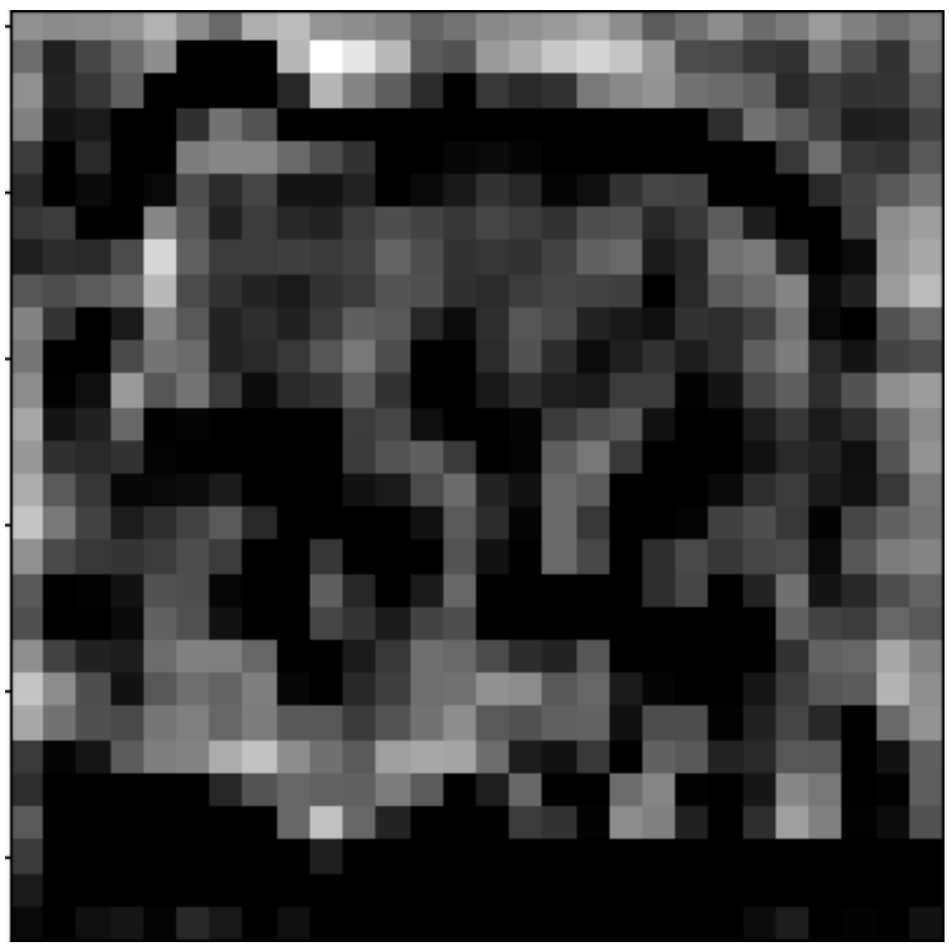
Layer

Tensor

Visualización de datos intermedios

Mapa de activación (layer block4conv1, filtro 10) dada una entrada

```
5 plt.figure()
6 plt.imshow(np.squeeze(I[...,10]), 'gray')
```



Visualización de datos intermedios

¿No había una forma más simple de hacerlo?

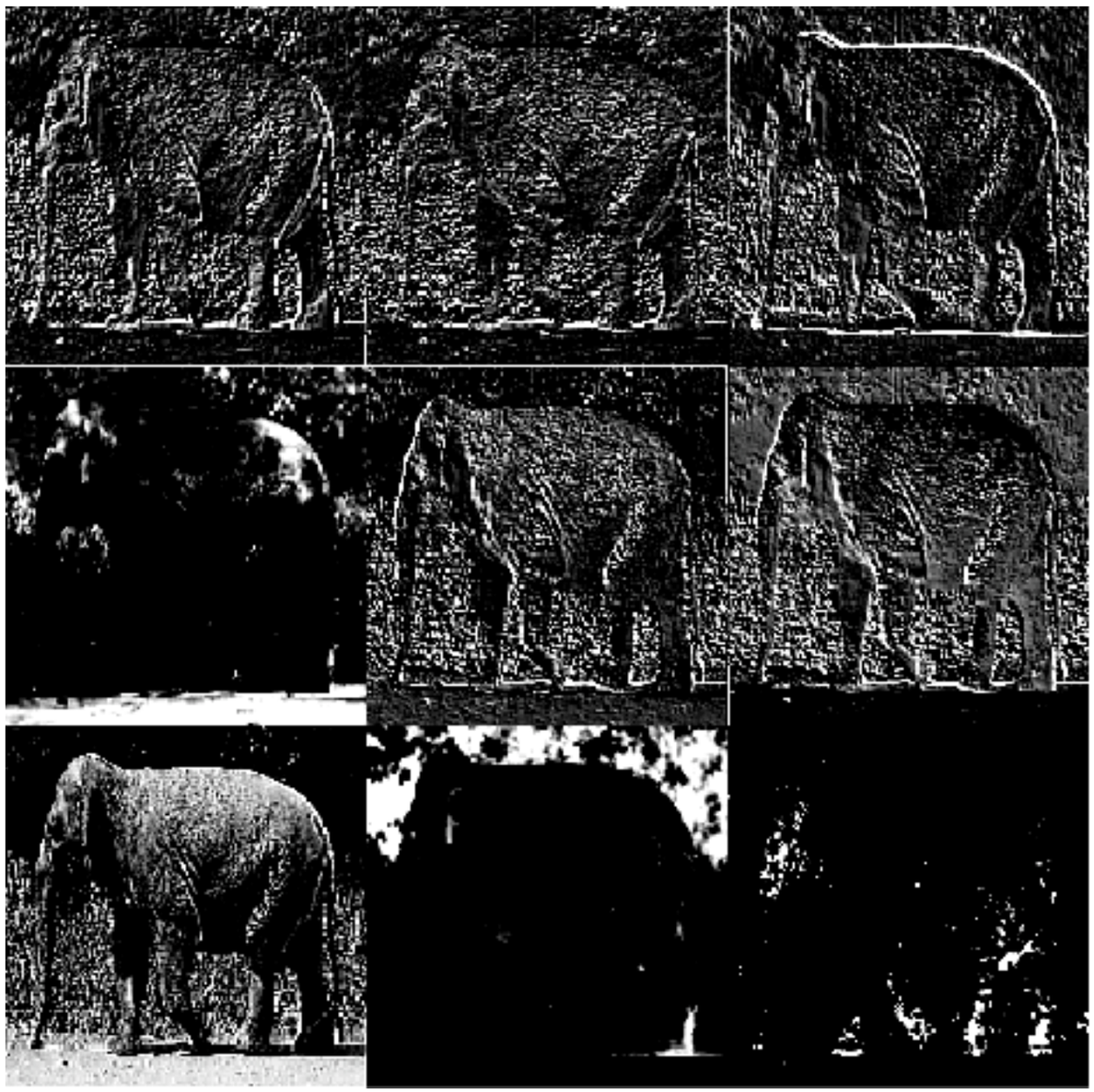
Si, pero es interesante K.function

```
478 layers = [l.output for l in model.layers[1:9] if 'conv' in l.name]
479 model_activation = tf.keras.models.Model(inputs=model.input, outputs=layers)
480 activations = model_activation.predict(x)
```

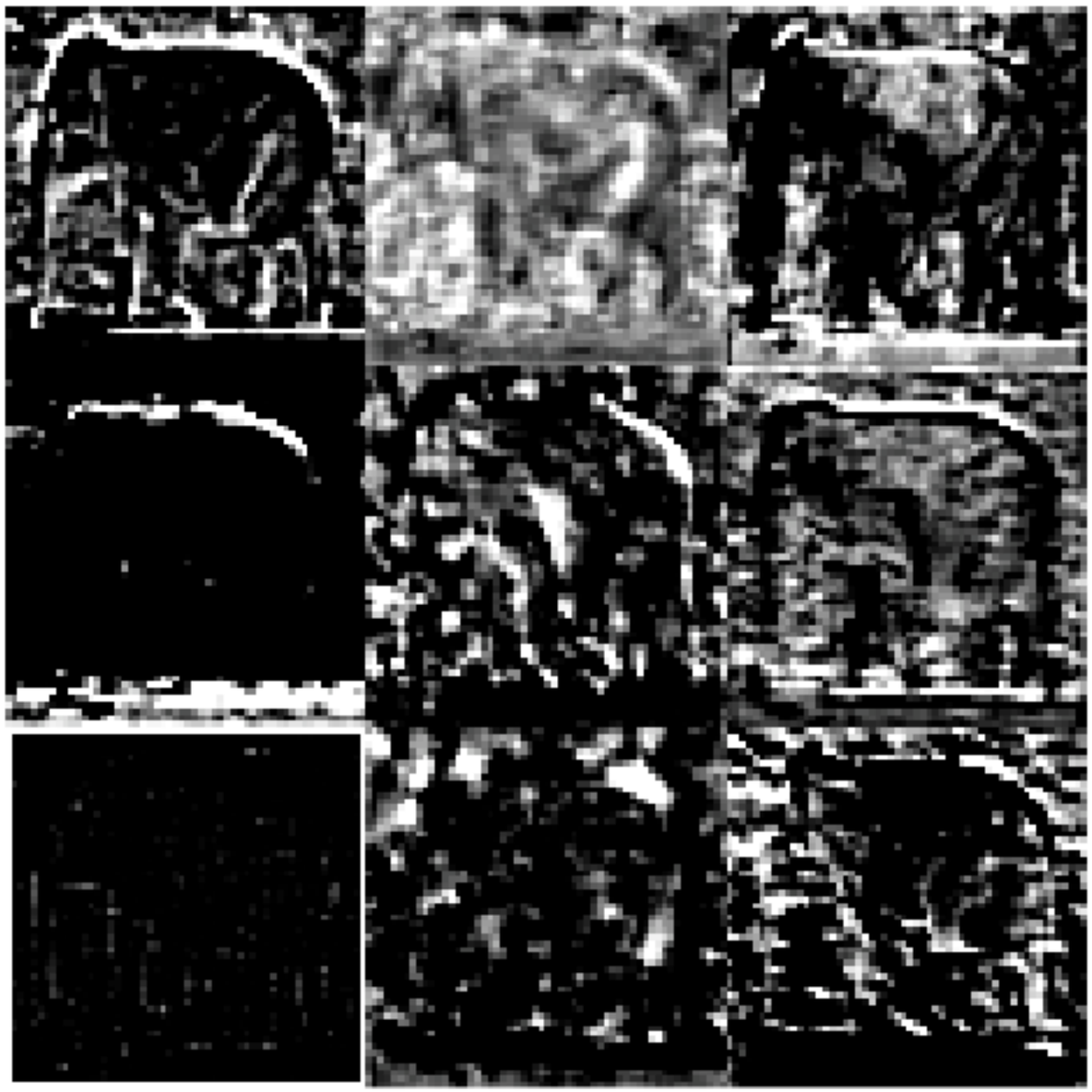
```
481 idl=4
482 nfigs=3
483 fig, ax = plt.subplots(nfigs,nfigs)
484 for i in range(nfigs):
485     for j in range(nfigs):
486         Ii = activations[idl][0,...,i+j*nfigs]
487         Ii -= Ii.mean()
488         Ii /= Ii.std()
489         Ii *= 64
490         Ii += 128
491         Ii = np.clip(Ii, 0, 255).astype('uint8')
492         ax[i,j].set_axis_off()
493         ax[i,j].imshow(Ii, 'gray')
494 fig.tight_layout()
```

Visualización de datos intermedios

layer 1 primeros 9 filtros



layer 4 primeros 9 filtros



Visualización de datos intermedios

¿Estamos viendo lo aprendido un filtro?

No, lo que prendió el filtro puede no ser ni de cerca lo que muestra su activación.

¿Qué significa lo que aprendió el filtro?

¿Qué estamos viendo entonces?

¿Cómo hacemos entonces para ver que aprendió la red?

Visualización de datos intermedios

Mapa de activación: **Gradient ascendent input space**

Armamos la imagen que maximiza la activación del filtro,
entonces, ese filtro aprendió a buscar esos detalles en la imagen

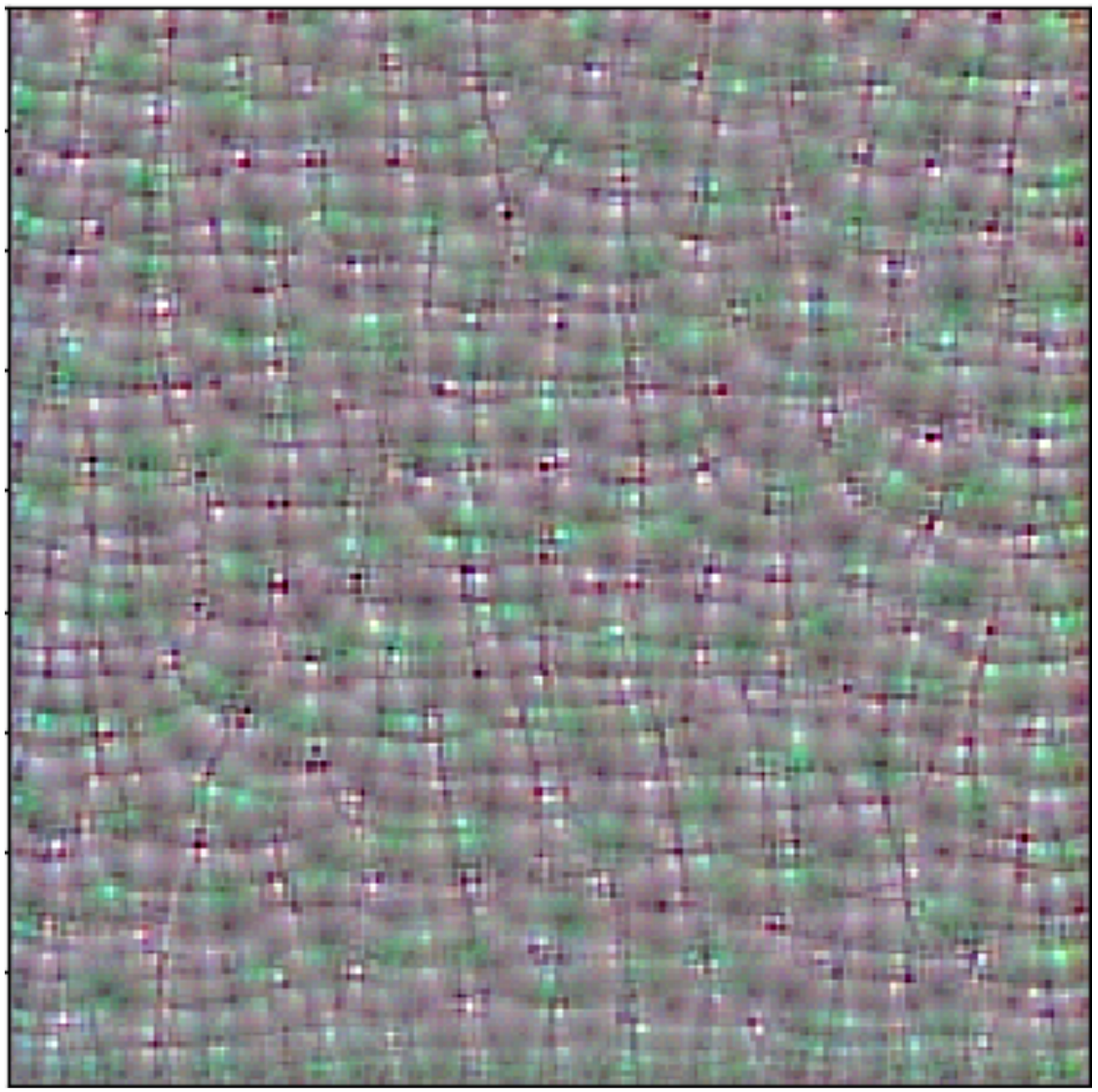
```
filt_index = 10
loss = K.mean(activation[..., filt_index])
grads = K.gradients(loss, model.input)[0]
# Truco de normalizacion
grads /= (K.sqrt(K.mean(K.square(grads))) + 1e-6)
run = K.function([model.input], [loss, grads])

img_data = np.random.random((1,224,224,3)) * 20 + 128
step = 1
for i in range(40):
    loss_val, grads_val = run([img_data])
    img_data += grads_val * step
    print('step:{:03d}\t loss:{:.3f}'.format(i, loss_val))
```

Arancamos con una imagen sintética (gris con algo de ruido)

Maximizamos con gradiente descendente

Visualización de datos intermedios

Mapa de activación: **Gradient ascent input space**



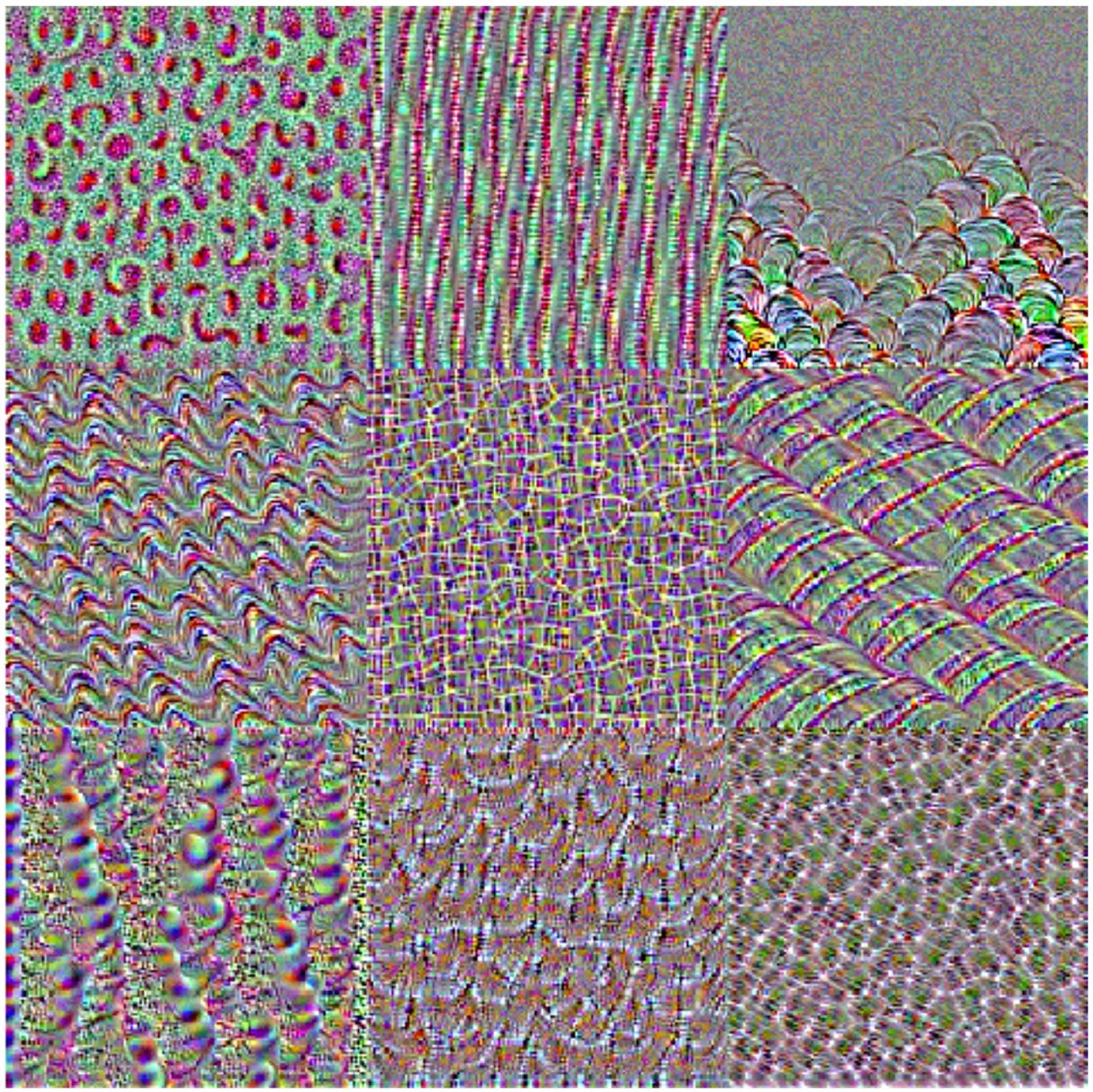
```
def deprocess_img(x):
    x -= x.mean()
    x /= (x.std() +1e-7)
    x *= .1
    x += .5
    x = np.clip(x, 0, 1)
    x *= 255
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

```
I2 = deprocess_img(img_data[0])
plt.figure()
plt.imshow(I2)
```

Visualización de datos intermedios

Mapa de activación: **Gradient ascendent input space**

```python
nfigs=3
fig, ax = plt.subplots(nfigs,nfigs)
for i in range(nfigs):
    for j in range(nfigs):
        img_data = learned_image(layer_name,
                                 i+j*nfigs)
        Ii = img_data[0]
        Ii -= Ii.mean()
        Ii /= Ii.std()
        Ii *= 64
        Ii += 128
        Ii = np.clip(Ii, 0, 255).astype('uint8')
        ax[i,j].set_axis_off()
        ax[i,j].imshow(Ii, 'gray')
fig.tight_layout()
```



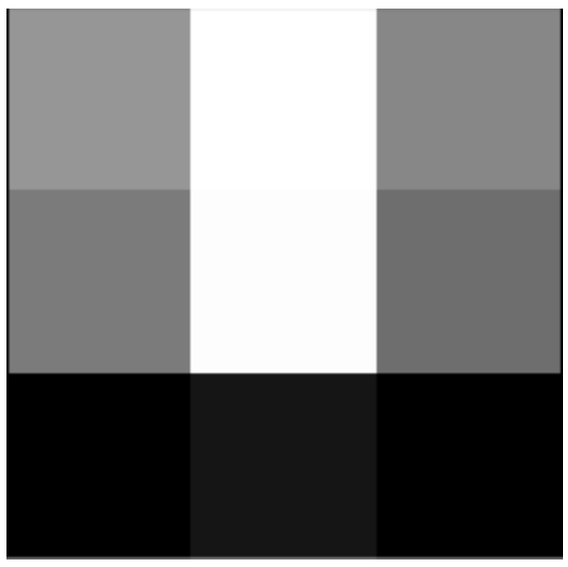
Visualización de datos intermedios

¿Qué pinta tienen los filtros?

```
506 lname1 = ‘block1_conv1'
507 l1_weights = model.get_layer(lname1).get_weights()
```

```
 3 channel_id = 0
 4 filt_id = 10
```

```
 6 plt.figure()
 7 plt.imshow(l1_weights[0][...,channel_id,filt_id], 'gray')
```



3x3 no nos aporta mucho

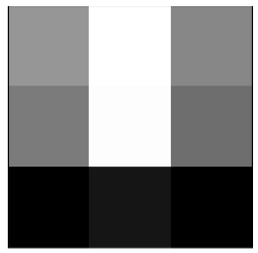
Visualización de datos intermedios

¿Qué pinta tienen los filtros?

```
lname1 = 'block1_conv1'
l1_weights = model.get_layer(lname1).get_weights()
```

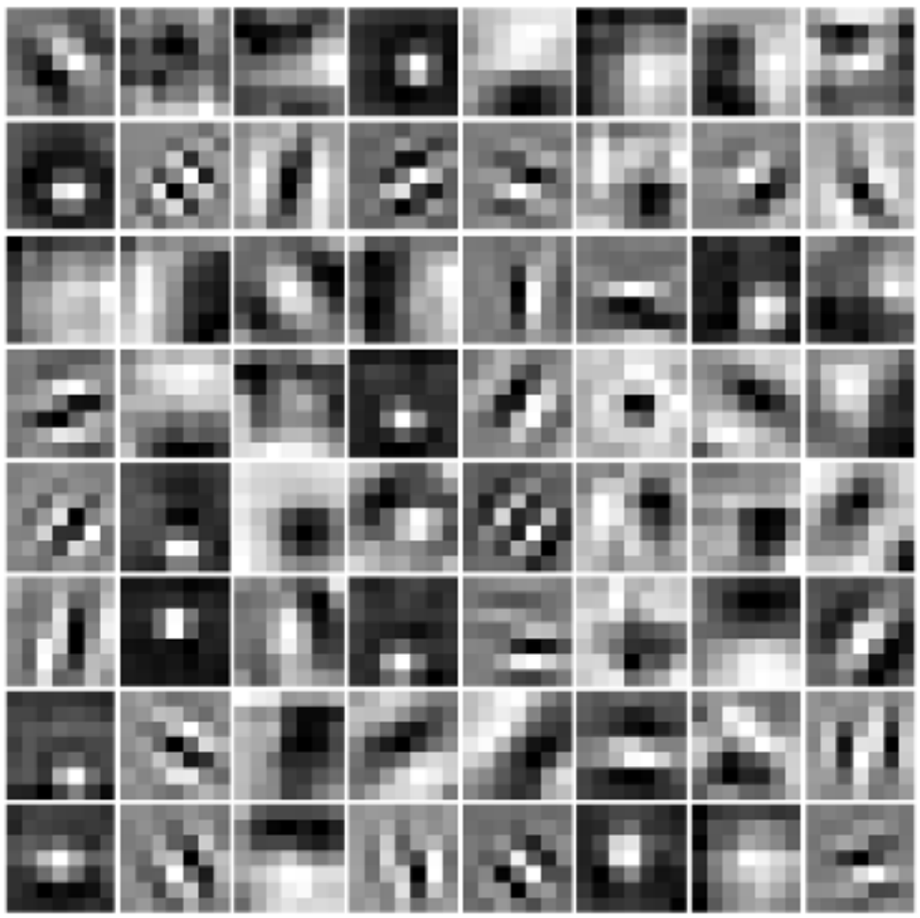
```
channel_id = 0
filt_id = 10
```

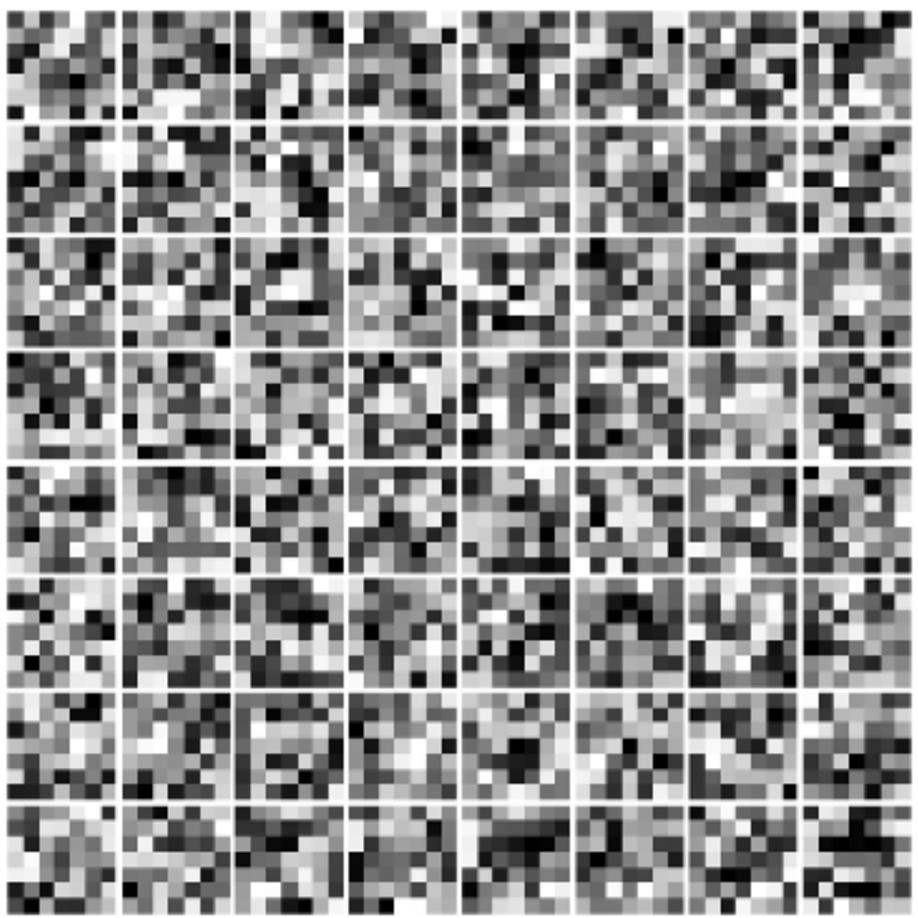
```
plt.figure()
plt.imshow(l1_weights[0][...,channel_id,filt_id], 'gray')
```



3x3 no nos aporta mucho

ResNet50 Conv 1





Unet Conv 1

¿Alguno obtuvo dos veces el mismo resultado al utilizar TF o Keras?

Reproducibilidad de los datos

- Ejecutar varias veces y usar estadísticas al momento de describir el rendimiento de nuestros modelo
- Fijar la semilla de los generadores aleatorios.
  - En GPU hay instrucciones que NO son deterministas
  - En CPU se puede lograr pero hay que fijar la variable de entorno PYTHONHASHSEED a 0 antes de ejecutar el código. Por ejemplo así:

```
CUDA_VISIBLE_DEVICES="" PYTHONHASHSEED=0 python your_program.py
```

Reproducibilidad de los datos

- Las semillas de los generadores de números aleatorios se tienen que definir antes que otra cualquier cosa

Reproducibilidad de los datos

```
#-------- Codigo para obtener reproducibilidad
import numpy as np
import tensorflow as tf
import random as rn
# The below is necessary for starting Numpy generated random numbers
# in a well-defined initial state.
np.random.seed(42)
# The below is necessary for starting core Python generated random numbers
# in a well-defined state.
rn.seed(12345)
# Force TensorFlow to use single thread.
# Multiple threads are a potential source of non-reproducible results.
# For further details, see: https://stackoverflow.com/questions/42022950/
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
                              inter_op_parallelism_threads=1)
from tensorflow.keras import backend as K
# The below tf.set_random_seed() will make random number generation
# in the TensorFlow backend have a well-defined initial state.
# For further details, see:
# https://www.tensorflow.org/api_docs/python/tf/set_random_seed
tf.set_random_seed(1234)
sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```