El dataset de MNIST

In [1]:
```
import tensorflow as tf
from skimage import io
import numpy as np
from IPython.display import Image
from skimage import io
import random
```

In [2]:
```
(images_train, labels_train), (images_test, labels_test) = tf.keras.datasets.mnist.load_data(
```

In [3]:
```
len(images_test)
```

Out[3]:
```
10000
```

In [4]:
```
len(images_train)
```

Out[4]:
```
60000
```

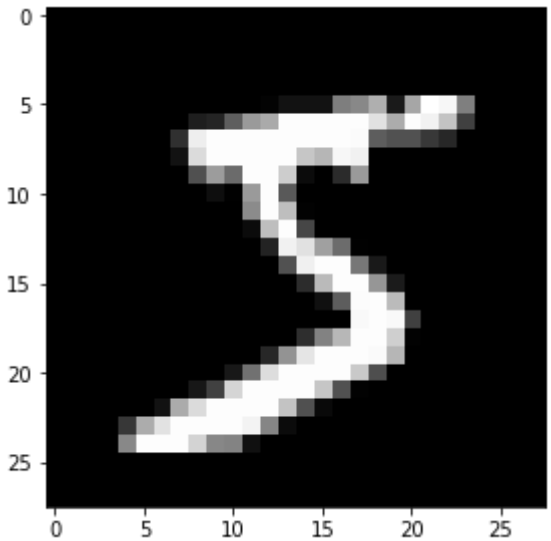
In [5]:
```
labels_train_encoded = tf.keras.utils.to_categorical(labels_train, num_classes=10)
```

In [6]:
```
labels_train_encoded[0]
```

Out[6]:
```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

In [7]:
```
io.imshow(images_train[0])
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x1af04c19b50>
```



In [8]:
```
labels_train_encoded[0] # Se trata de un 5 si contamos
```

Out[8]:
```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

In [ ]:
```

```

Una red neuronal con TensorFlow -v1

- Las imágenes de entrenamiento de MNIST viven en un espacio vectorial de dimensión 28*28=784.
- El dataset sew pudede pensar como 60000 filas y 784 columnas.
- Cada dato del dataset es un númnero entre el cero y el uno dependiendo de cuán más o menos blanco sea.

La regresión Softmax es la más natural y más común para implementar en casos de redes neuronales, puede entregrarnos una lista entre ceros y unos y pueden ser interpretados como probabilidades. **Siempre** la última capa de una red neuronal está compuesta por una capa de **SoftMax** y es que la regreseión de Softmax es muy sencilla.

SoftMax lleva a cabo dos simples cálculos, primero suma las evidencias de que la entrada de datos sea de cierta clase de cada una de las posibles y convierte esas evidencias en probabilidades. Entonces para dar una evidencia de que una imágen sea de una clase en particular, se pondera (suma de productos) de todos y cada uno de las intensidades de los píxeles de las imágenes. Existe un factor extra que se llama desviación o bias, lo que hace es sumar un poco más de evidencia. La idea es ser capaces de establecer evidencias de que un objeto se categoriza por pertenecer en la clase i-ésima.

$evidencia_i = \sum_{j \in Pixels} W_{ij} Xj + b_J$

$\begin{cases} W = \ pesos\ o\ kernel \\ b = \ desviación\ o\ bias \end{cases}$

$\hat{y} = softmax(evidencia)$

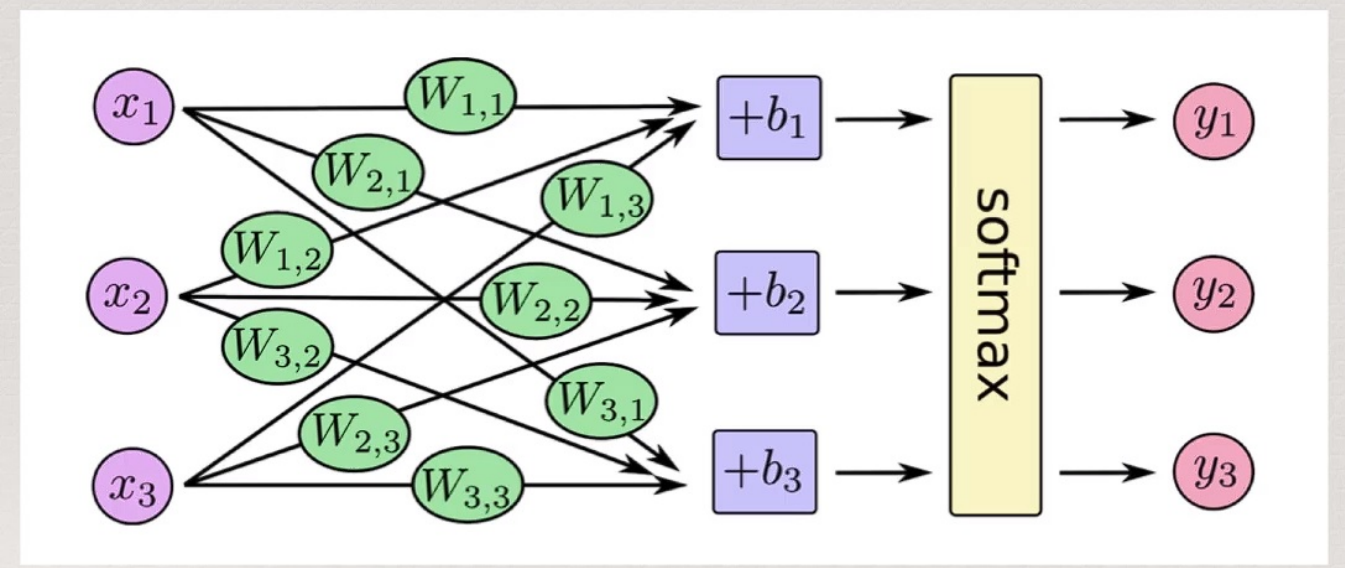
$softmax(x) = normalize(exp(x))$

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

Softmax sirve como método de activación, como link, para indicar al hecho de pertencer o nó a una clase. Se pueden utilizar diferentes funciones de activaciones, la más habitual es la sigmoide. Con la exponenciación, multiplicamos las probabilidades para sumarle peso, cuanto más cerca de uno estén entonces verdaderamente esta imágen pertenece a esta categoría o clase. Cuando tengamos pocas evidencias, la fracción será pequeña, el exponencial será pequeño y la exponencial bajará y no habrá forma de establecer que pertenezca a dicha clase.

In [9]:
```
Image(filename="C:/Users/Kevin\Desktop/Kevin/1-Cursos/1-Machine Learning/CURSO UDEMY ML/16-Re
```

Out[9]:



Con este diagrama tendremos una serie de entradas, X1, X2, X3 para cada una de las entradas computaremos una suma ponderada de las X sumándole cada una de las desviaciones, para el ejemplo sólo tenemos tres clases y cada una de las clases las pnderamos entre sí y así suscesivamente. Al final se suman los bias, una vez que tenemos todos los datos posibles aplcamos la función de softmax y luego de aplicar dicha función nos quedarán una serie de posiboles valores. En forma de ecuación nos queda:

In [10]:
```
Image(filename="C:/Users/Kevin\Desktop/Kevin/1-Cursos/1-Machine Learning/CURSO UDEMY ML/16-Re
```

Out[10]:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{matrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{matrix} \right)$$

Sería simplemente, cada una de las clases sería la función softmax, es decir la normalización de las e^(cada una de las operaciones matriciales). Incluso se puede vectorizar este proceso convirtiendo este producto de matrices con sumas dentro en un producto matricial con una suma de vectores, lo cual es múcho más eficiente desde el punto de vista computacional:

In [11]:
```
Image(filename="C:/Users/Kevin\Desktop/Kevin/1-Cursos/1-Machine Learning/CURSO UDEMY ML/16-Re
```

Out[11]:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Nosotros a la hora de escribirlo en tensorflow, python o en R al final lo que implementamos es la versión de que la estimación será la función softmax aplicado al producto de una serie de pesos por las clases **X** más la desviación, **b**.

$$\hat{y} = softmax(\omega X + b)$$

Recordar que lo que hace **Tensor Flow** es hacer que el trabajo lo realice la gpu y la cpu, fuera de lo que es python, terciariza el laburo a otros sistemas entonces Tensor Flow necesita los **placeholders** para tener una indicación de dónde están apuntados los datos que se procesarán.

In [12]:
```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
```

In [13]:
```
dim_input = 28
n_categories = 10
```

In [14]:
```
x = tf.compat.v1.placeholder(dtype = tf.float32, shape = [None, 28, 28]) #cualquier longitud
```

In [15]:
```
W = tf.Variable(tf.zeros([28,28,n_categories])) #10 columnas (una pora cada clase)
b = tf.Variable(tf.zeros([n_categories])) # el vector de suma bias será una variable.
```

Una variable es un tensor que se puede utilizar y modificar y que vive en el grafo de tensor flow para todas las operaciones interactuantes, se puede utilizar, se puede modificar por el propio algoritmo por la popia red neuronal y los algoritmos de ML utilizan las variables para guardar ahí todo lo que van aprendiendo con iteraciones, un **placeholder** NO cambia, una variable sí cambia. El cero se toma como punto de partida, fijarse las dimensiones que tienen que ser perfectamente compatibles.

In [16]:
```
softmax_args = tf.matmul(x,W) + b #multiplicación de matrices. Se pone alrevés de como lo esc
y_hat = tf.nn.softmax(softmax_args) #para lidiar con los tensores 2D
```

Fase de entrenamiento de la red neuronal

Para entrenar un modelo de red neuronal hay que definir qué entendemos porqué un modelo sea bueno, o malo. En la actualidad lo que se representa es el **cost** o **loss**, la idea es minimizar las pérdidas. El modelo siempre va a ser malo, la idea es minimizar lo malo que sea.

Lo que se suele hacer comunmente, es determinar la función de pérdeida através de la **cross entropy**, surge de:

$H_y(\hat{y}) = -\sum_i y_i log(\hat{y}_i)$

Y_i son los valores reales que ingresemos, de algún modo la entropía cruzada mide cómo de ineficientes son nuestras predicciones con respecto de descubrir la realidad. Es una forma de poder juntar ambos conceptos, entropía y comprobación cruzada.

In [17]:
```
y_ = tf.placeholder(tf.float32, [None]) #None indica que no vamos a indicar la cantidad de da
```

In [18]:
```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_*tf.log(y_hat), reduction_indices=[1]))
```

- tf.log(y_hat): Calcula el logaritmo de cada uno de los elementos de las predicciones
- se multiplica por el valor original y_
- tf.reduce_sum : sumará todos los elementos en este caso de sólo la segunda dimensión.
- tf.reduce_mean: calculará el promedio de todas las muestras del dataset

A veces para realizar funciones o algoritmos matemáticamente más estables, se utilizan datos sin normalizar porque al final cuando normalizamos, los valores varían de 0 a 1 y cuando tenemos el

logaritmo de un valor muy muy chico cercano a cero, quiere decir que el logaritmo es extremadamamente negativo entonces se dice que esos algoritmos son numéricamente **inestables** con lo que se recomienda utilizar es :

tf.nn.softmax_cross_entropy_with_logits(softmax*args, y*)

In [19]:
```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

In [20]:
```
session = tf.InteractiveSession()
```

In [21]:
```
tf.global_variables_initializer().run()
```

In [22]:
```
for _ in range(5000):
    rand = random.sample(range(0, len(images_train)), 100)
    batch_x = images_train[rand]
    batch_y = labels_train[rand]
    session.run(train_step, feed_dict={x:batch_x, y_:batch_y})
```

```
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\client\session.py in _do_call(self, fn, *args)
   1376     try:
-> 1377       return fn(*args)
   1378     except errors.OpError as e:

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\client\session.py in _run_fn(feed_dict, fetch_list, target_list, options, run_metadata)
   1359       self._extend_graph()
-> 1360       return self._call_tf_sessionrun(options, feed_dict, fetch_list,
   1361                                       target_list, run_metadata)

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\client\session.py in _call_tf_sessionrun(self, options, feed_dict, fetch_list, target_list, run_metadata)
   1452                               run_metadata):
-> 1453     return tf_session.TF_SessionRun_wrapper(self._session, options, feed_dict,
   1454                                             fetch_list, target_list,

InvalidArgumentError: In[0] and In[1] must have compatible batch dimensions: [100,28,28] vs. [28,28,10]
        [[{{node MatMul}}]]

During handling of the above exception, another exception occurred:

InvalidArgumentError                      Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_16928/3838139002.py in <module>
      3     batch_x = images_train[rand]
      4     batch_y = labels_train[rand]
----> 5     session.run(train_step, feed_dict={x:batch_x, y_:batch_y})

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\client\session.py in run(self, fetches, feed_dict, options, run_metadata)
    965
    966     try:
--> 967       result = self._run(None, fetches, feed_dict, options_ptr,
    968                          run_metadata_ptr)
    969       if run_metadata:

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\client\session.py in _run(self, handle, fetches, feed_dict, options, run_metadata)
```

```
   1188     # or if the call is a partial run that specifies feeds.
   1189     if final_fetches or final_targets or (handle and feed_dict_tensor):
-> 1190       results = self._do_run(handle, final_targets, final_fetches,
   1191                              feed_dict_tensor, options, run_metadata)
   1192     else:

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\client\session.py in _do_run(sel
f, handle, target_list, fetch_list, feed_dict, options, run_metadata)
   1368
   1369     if handle is None:
-> 1370       return self._do_call(_run_fn, feeds, fetches, targets, options,
   1371                            run_metadata)
   1372     else:

C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\client\session.py in _do_call(se
lf, fn, *args)
   1394                     '\nsession_config.graph_options.rewrite_options.'
   1395                     'disable_meta_optimizer = True')
-> 1396       raise type(e)(node_def, op, message)  # pylint: disable=no-value-for-parameter
   1397
   1398   def _extend_graph(self):

InvalidArgumentError: Graph execution error:

Detected at node 'MatMul' defined at (most recent call last):
    File "C:\ProgramData\Anaconda3\lib\runpy.py", line 197, in _run_module_as_main
      return _run_code(code, main_globals, None,
    File "C:\ProgramData\Anaconda3\lib\runpy.py", line 87, in _run_code
      exec(code, run_globals)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py", line 16, in <mod
ule>
      app.launch_new_instance()
    File "C:\ProgramData\Anaconda3\lib\site-packages\traitlets\config\application.py", line 8
46, in launch_instance
      app.start()
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelapp.py", line 677, in st
art
      self.io_loop.start()
    File "C:\ProgramData\Anaconda3\lib\site-packages\tornado\platform\asyncio.py", line 199,
 in start
      self.asyncio_loop.run_forever()
    File "C:\ProgramData\Anaconda3\lib\asyncio\base_events.py", line 596, in run_forever
      self._run_once()
    File "C:\ProgramData\Anaconda3\lib\asyncio\base_events.py", line 1890, in _run_once
      handle._run()
    File "C:\ProgramData\Anaconda3\lib\asyncio\events.py", line 80, in _run
      self._context.run(self._callback, *self._args)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 457, in d
ispatch_queue
      await self.process_one()
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 446, in p
rocess_one
      await dispatch(*args)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 353, in d
ispatch_shell
      await result
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 648, in e
xecute_request
      reply_content = await reply_content
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py", line 353, in do_
execute
      res = shell.run_cell(code, store_history=store_history, silent=silent)
    File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\zmqshell.py", line 533, in run
_cell
      return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line
```

```
 2901, in run_cell
      result = self._run_cell(
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line
 2947, in _run_cell
      return runner(coro)
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\async_helpers.py", line 68,
in _pseudo_sync_runner
      coro.send(None)
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line
 3172, in run_cell_async
      has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line
 3364, in run_ast_nodes
      if (await self.run_code(code, result,  async_=asy)):
    File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line
 3444, in run_code
      exec(code_obj, self.user_global_ns, self.user_ns)
    File "C:\Users\Kevin\AppData\Local\Temp/ipykernel_16928/2730511702.py", line 1, in <modul
e>
      softmax_args = tf.matmul(x,W) + b #multiplicación de matrices. Se pone alrevés de como
 lo escribimos
Node: 'MatMul'
In[0] and In[1] must have compatible batch dimensions: [100,28,28] vs. [28,28,10]
         [[{{node MatMul}}]]

Original stack trace for 'MatMul':
  File "C:\ProgramData\Anaconda3\lib\runpy.py", line 197, in _run_module_as_main
    return _run_code(code, main_globals, None,
  File "C:\ProgramData\Anaconda3\lib\runpy.py", line 87, in _run_code
    exec(code, run_globals)
  File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py", line 16, in <modul
e>
    app.launch_new_instance()
  File "C:\ProgramData\Anaconda3\lib\site-packages\traitlets\config\application.py", line 84
6, in launch_instance
    app.start()
  File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelapp.py", line 677, in star
t
    self.io_loop.start()
  File "C:\ProgramData\Anaconda3\lib\site-packages\tornado\platform\asyncio.py", line 199, in
start
    self.asyncio_loop.run_forever()
  File "C:\ProgramData\Anaconda3\lib\asyncio\base_events.py", line 596, in run_forever
    self._run_once()
  File "C:\ProgramData\Anaconda3\lib\asyncio\base_events.py", line 1890, in _run_once
    handle._run()
  File "C:\ProgramData\Anaconda3\lib\asyncio\events.py", line 80, in _run
    self._context.run(self._callback, *self._args)
  File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 457, in dis
patch_queue
    await self.process_one()
  File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 446, in pro
cess_one
    await dispatch(*args)
  File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 353, in dis
patch_shell
    await result
  File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 648, in exe
cute_request
    reply_content = await reply_content
  File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\ipkernel.py", line 353, in do_ex
ecute
    res = shell.run_cell(code, store_history=store_history, silent=silent)
  File "C:\ProgramData\Anaconda3\lib\site-packages\ipykernel\zmqshell.py", line 533, in run_c
ell
    return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)
```

```
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 29
01, in run_cell
    result = self._run_cell(
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 29
47, in _run_cell
    return runner(coro)
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\async_helpers.py", line 68, i
n _pseudo_sync_runner
    coro.send(None)
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 31
72, in run_cell_async
    has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 33
64, in run_ast_nodes
    if (await self.run_code(code, result,  async_=asy)):
  File "C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 34
44, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "C:\Users\Kevin\AppData\Local\Temp/ipykernel_16928/2730511702.py", line 1, in <module>
    softmax_args = tf.matmul(x,W) + b #multiplicación de matrices. Se pone alrevés de como lo
escribimos
  File "C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\util\traceback_utils.p
y", line 150, in error_handler
    return fn(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\util\dispatch.py", line
 1082, in op_dispatch_handler
    return dispatch_target(*args, **kwargs)
  File "C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py", line 3
666, in matmul
    return gen_math_ops.batch_mat_mul_v2(
  File "C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\ops\gen_math_ops.py", li
ne 1589, in batch_mat_mul_v2
    _, _, _op, _outputs = _op_def_library._apply_op_helper(
  File "C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\op_def_librar
y.py", line 797, in _apply_op_helper
    op = g._create_op_internal(op_type_name, inputs, dtypes=None,
  File "C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\ops.py", line
 3754, in _create_op_internal
    ret = Operation(
  File "C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\python\framework\ops.py", line
 2133, in __init__
    self._traceback = tf_stack.extract_stack_for_node(self._c_op)
```

Evaluando la red neuronal

In [ ]:
```
correct_predictions = tf.equal(tf.argmax(labels_train,1), tf.argmax(y_, 1))
```

In [ ]:
```
accuracy = tf.reduce_mean(tf.cast(correct_predictions, tf.float32))
```

In [ ]:
```
print(session.run(accuracy, feed_dict={x:mnist.test.images, y_: mnist.test.labels}))
#No va a funcionar porque Juan Gabriel utiliza un Tensor Flow tan viejo que no sirven estas t
#de hecho es totalmente diferente el funcionamiento actual.
```