Introducción

A finales de noviembre de 2015 google liberó **TensorFlow**, una librería orientada a la construcción de modelos usando redes neuronales y usando todo el potencial del bicho que tengamos.

Las cinco nociones básicas sobre TensorFlow que nos describe el propio Google:

- El diagrama de nuestro modelo está contenido dentro de la clase **Graph**
- Se ejectuta en el contexto de **Sessions**
- Los datos son tensores
- Para mantener el estado durante las ejecuciones del diagrama, usaremos **Variables**
- Para obtener los resultados y alimentar al sistema con variables externas, usaremos respectivamente ***fetches*** y ***feeds*** al correr la sesión

Flujo de trabajo

Un grafo es un conjunto de operaciones animadas que representan nuestro modelo.

Al iniciar cualquier operación, se añadirá al grafo(`Graph`) que está por defecto. Si no creamos ninguno, será el que TensorFlow no provee.

Después, solo falta ejecutar el grafo a través de una sesión

Graphs, construir el diagrama

Lo primero que tenemos que hacer, es generar los nodos o operaciones que intervienen en nuestro modelo. Si no instanciamos ningún grafo, la libreria TensorFlow lo hará por nosotros. A partir de ahora, para dejarlo mas claro, vamos a instanciar explícitamente los grafos por medio de los **bloques with**.

Grafo Implicito/Explicito

En ese ejemplo se puede apreciar que se ha creado dos grafos, uno implicito al crear la operacion ***const1*** y otro explicito al crear el ***grafo1***. **Las operaciones o nodos se asignarán al grafo por defecto**

```
import tensorflow as tf

# Se genera un grafo por defecto
const1 = tf.constant(4.0,name='const1')
grafo_por_defecto=tf.get_default_graph()
assert const1.graph is grafo_por_defecto

# En este caso ya no se genera el grafo, unicamente se añade la operación
const2 = tf.constant(30.0,name='const2')
assert const2.graph is grafo_por_defecto

# Explicitamente creamos un nuevo grafo
grafo1 = tf.Graph()
with grafo1.as_default():
    conts3 = tf.constant(30.0)
    assert conts3.graph is grafo1

const3 = tf.constant(20)
assert const3.graph is grafo_por_defecto
```

Sessions, ejecutar el modelo

Lo próximo es ejecutar el grafo, para ello vamos a crear una sesión para luego ejecutarlo por medio del método `run`. Si instanciamos el objeto sin argumentos, lanzará el grafo por defecto.

Lanzar Grafo

En este caso, como no se ha generado ningún grafo por defecto, tenemos que pasarle por argumentos al constructor de la sesión el grafo que queremos lanzar

```
import tensorflow as tf

# Generamos el grafo
grafo1 = tf.Graph()
with grafo1.as_default():
    matrix1 = tf.constant([[3., 3.], [3., 3.]], name='Matriz1')
    matrix2 = tf.constant([[2., 2.], [2., 2.]], name='Matriz2')

    producto_grafo1 = tf.matmul(matrix1, matrix2, name='Producto')

# Ejecutamos el grafo1
with tf.Session(graph=grafo1) as sess:
    result = sess.run([producto_grafo1])
    print (result)
```

Del mismo modo que se construyen los grafos por medio de bloques `with`, también se va a realizar con las sesiones. En este caso nos ayuda a liberar los recursos automáticamente. Sino deberíamos indicarlo explícitamente

Variables y entrada(feeds) /salida(fetches)

Ya sabemos crear y ejecutar nuestro modelo, pero poco podemos tirar de nuestra imaginación sin presentar algo tan básico como las variables. Las **variables** permiten mantener el estado de nuestros datos a través de las distintas ejecuciones. **Siempre hay que inicializarlas**, lo mas fácil: `tf.initialize_all_variables().run()`
También es básico saber el estado de dichas variables, para ello, utilizaremos el primer argumento (**`fetch`**) del método `run` de la sesión. Como se puede ver en el ejemplo, se puede pasar tantas variables como deseemos, **Por cada ejecución se evaluará las variables o tensores, no al reves**.

TensorFlow también nos permite proveer (**feed**) datos directamente a cualquier nodo o operación del **Graph.** Lo usual es crear parametros a través del método `tf.placeholder()`, de esta manera se reemplaza el contenido por lo introducido a través del argumento `feed_dict` del método `run`de la sesión.

Variable, feed y fetch

Se guarda el resultado del cuadrado del contador en la variable sumatorio

```
import tensorflow as tf

grafo1=tf.Graph()
with grafo1.as_default():
    # Variable que guardará el sumatorio en las distintas ejecuciones
    sumatorio = tf.Variable(0, name="counter")
    # Variable de entrada (feed) = contador
    input1 = tf.placeholder(tf.int32)

    # Logica de cada paso
    valor_operacion = tf.mul(input1, input1)
    flujo=tf.assign_add(sumatorio, valor_operacion)


with tf.Session(graph=grafo1) as sess:
    # Las variables deben inicializarse explicitamente
    tf.initialize_all_variables().run()

    for contador in range(3):
        estado=
sess.run([flujo,valor_operacion],feed_dict={input1:contador})
        print ("Iteracion {}: {}".format(contador,estado))
```

Tensor Flow en modo interactivo

La librería de Google también nos blinda una forma fácil de utilizar Tensor Flow de forma interactiva en Ipython. Unicamente en vez de crear una `Session`, utilizaremos el

método `tf.InteractiveSession()` que automáticamente nos creará una sesión por defecto, por lo que no tenemos que llamar a `Tensor.eval()` o `Operation.run()` indicando la sesión.

Ejemplo de como correr interactivamente la libreria TensorFlow en IPython

In [1]:

```
import tensorflow as tf
```

De esta manera lanzar una sesion interactiva, util cuando queremos probar metodos

In [2]:

```
sess = tf.InteractiveSession()
```

In [19]:

```
x = tf.Variable([[2.0, 3.0],[4.0, 12.0]])
```

Probamos la funcion que nos reduce un tensor por medio de medias

In [20]:

```
x.initializer.run()
tf.reduce_mean(x).eval()
```

Out[20]:

```
5.25
```

In [21]:

```
tf.reduce_mean(x,1).eval()
```

Out[21]:

```
array([ 2.5,  8. ], dtype=float32)
```

In [22]:

```
tf.reduce_mean(x,0).eval()
```

Out[22]:

```
array([ 3. ,  7.5], dtype=float32)
```

Cerramos la session para liberar recursos

In [24]:

```
sess.close()
```

TensorBoard, reprentación del grafo

TensorFlow nos ofrece la posibilidad de visualizar nuestro modelo y muchas cosas mas a traves de la herramienta **`tensorboard.`**

Unicamente tenemos que generar un informe de la sesión y ejecutar en nuestra terminal `tensorboard`.

TensorBoard

Añadimos al ejemplo anterior, las dos ultimas lineas encargadas de guardar el informe de la sesion

```
import tensorflow as tf

grafo1=tf.Graph()
with grafo1.as_default():
    # Variable que guardará el sumatorio en las distintas ejecuciones
    sumatorio = tf.Variable(0, name="counter")
    # Variable de entrada (feed) = contador
    input1 = tf.placeholder(tf.int32)

    # Logica de cada paso
    valor_operacion = tf.mul(input1, input1)
    flujo=tf.assign_add(sumatorio, valor_operacion)


with tf.Session(graph=grafo1) as sess:
    # Las variables deben inicializarse explicitamente
    tf.initialize_all_variables().run()

    for contador in range(3):
        estado=
sess.run([flujo,valor_operacion],feed_dict={input1:contador})
        print ("Iteracion {}: {}".format(contador,estado))

        # Las siguientes sentencias, nos crearan
    merged = tf.merge_all_summaries()
    writer =
tf.train.SummaryWriter("/tmp/logs_tensor_flow/ejemplo_variable",
sess.graph_def)
```

Terminal TensorBoard

En la terminal ejecutamos tensorboard especificando a través del argumento logdir la carpeta donde hemos generado el informe

```
[pedro@localhost ~]$ tensorboard --logdir
/tmp/logs_tensor_flow/ejemplo_variable
```

Y ya lo tenemos en nuestro navegador `http://localhost:6006/`