

Introducción a TensorFlow

Introducción	2
“Hola Mundo” con TensorFlow	4
Elementos de TensorFlow	5
Tensores	5
Variables	7
Placeholders	7
Operaciones	9
Ejemplos	11
Serie de Fibonacci	11
Regresión lineal o ajuste lineal	13
Regresión lineal con Tensor Flow	13
Regresión Logística	16
Regresión logística con TensorFlow	18
Perceptrón multicapa con Tensor Flow	23
Referencias	28

Introducción

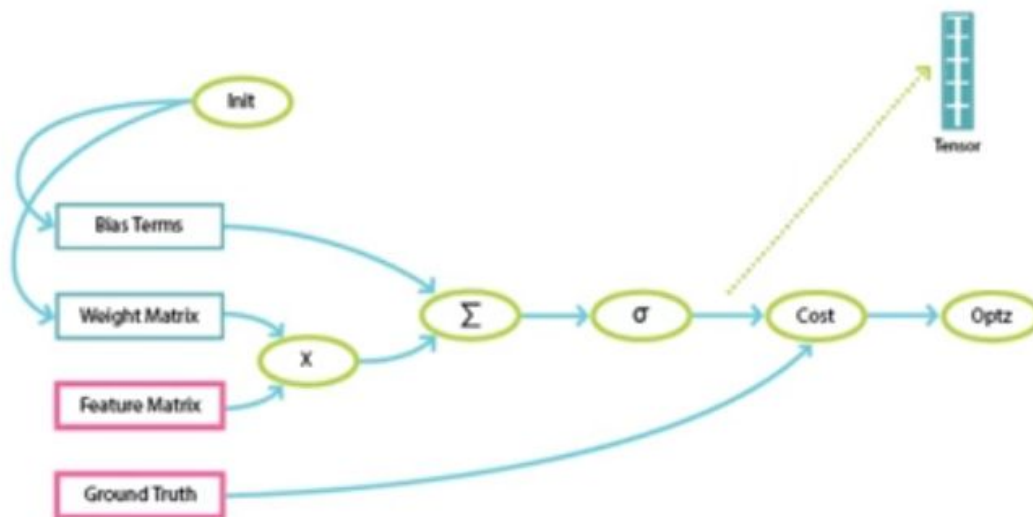
TensorFlow es una librería de código abierto desarrollada por un equipo de Google conocido como "Brain Team". Es una librería extremadamente versátil, pensada para atacar problemas generales de aprendizaje automático (machine learning) y aprendizaje profundo.

¿Por qué usarla?

- API Python y C++
- Tiempos más rápidos de compilación y entrenamiento que otras bibliotecas.
- Soporte de CPUs, GPUs y procesamiento distribuido
- Arquitectura flexible (Desktop, mobile, servidor)
- No requiere hardware especial

Una aplicación de TensorFlow usa una estructura conocida como grafo de conocimiento. Un grafo de conocimiento de TensorFlow tiene dos unidades básicas:

- **nodos**: representan operaciones matemáticas
- **aristas**: representan arrays multi-dimensionales, conocidos como tensores



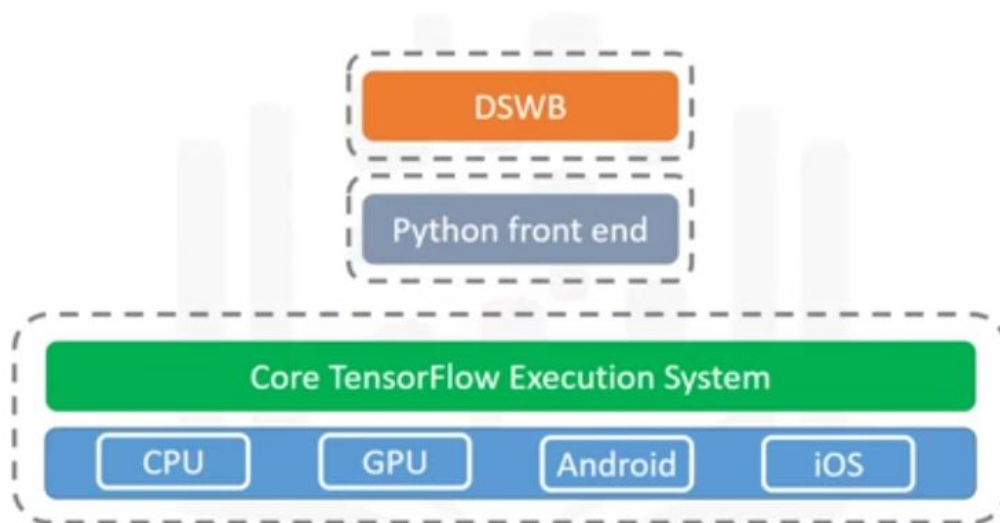
El funcionamiento básico consiste en construir un grafo y ejecutarlo, después de crear una sesión. Para ello hay dos operaciones: "run" y "eval"

Una vez que el grafo es construido, un loop interno es escrito para conducir la computación. las entradas (inputs) se envían a los nodos a través de variables.

En Tensor Flow un grafo solo va a correr después de haber sido creada una sesión.

Además de la API Python, incluye un módulo llamado Data Scientist Workbench (DSWB).

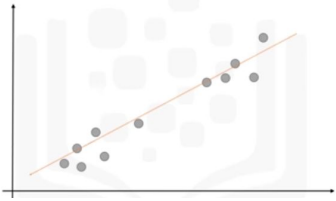
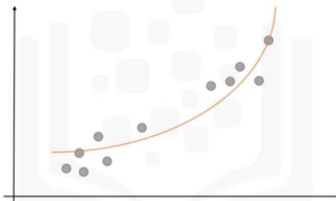

Arquitectura:



Contiene además una colección de funciones matemáticas entrenables y herramientas de aprendizaje automático, que son muy útiles para trabajar con redes neuronales.

Además de definir redes neuronales (superficiales o profundas), TensorFlow nos permite definir distintas funciones de activación, como la función escalón, la función sigma o la tangente hiperbólica.

Entre otras cosas, Tensor Flow puede usado también para aplicar regresión lineal sobre una serie de puntos, modelos no lineales o regresión logística. Como se muestra más abajo:

		
Regresión lineal	Regresión no lineal	Regresión logística

“Hola Mundo” con TensorFlow

Como se dijo, TensorFlow define las computaciones y cálculos como grafos, estas son llamadas operaciones u “ops”, así que para poder operar con TensorFlow primero hay que crear un grafo de operaciones.

Luego hay que crear una sesión, la sesión le pasa el grafo al dispositivo que la ejecutará (un CPU, un GPU, etc.)

Una suma de dos constantes sería un grafo como el que sigue:



Vamos a definir un grafo así y a ejecutarlo:

1. Instalar TensorFlow: <https://www.tensorflow.org/install/> (Esto no está cubierto en esta guía)
2. Una vez que lo instalaron correctamente. Creamos un archivo “**helloWorld.py**”
3. Importamos la librería:

```
import tensorflow as tf
```
4. Creamos dos constantes:

```
a = tf.constant([2])  
b = tf.constant([3])
```
5. Creamos una operación de suma:

```
c = tf.add(a,b) #c = a + b también está permitido
```
6. Creamos la sesión:

```
session = tf.Session()
```
7. Corremos el grafo e imprimimos los resultados:

```
result = session.run(c)  
print(result)
```
8. Cerramos la sesión:

```
session.close()
```
9. Desde una consola ejecutamos el archivo de python:

```
python helloWorld.py
```

Debería imprimir lo siguiente: [5]

Si imprime un mensaje adicional, un mensaje que comienza con una fecha y la letra **W**, es un **Warning**. Para que no nos muestre los mensajes de Warning, debemos añadir estas líneas al principio del archivo:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
```

Otra forma de utilizar una sesión sin tener que específicamente pedirle que la cierre es utilizando el siguiente código:

```
with tf.Session() as session:
    result = session.run(c)
    print(result)
```

Elementos de TensorFlow

Se describen a continuación los elementos típicos del lenguaje: Tensores, variables, constantes, operaciones y placeholders.

Tensores

En TensorFlow todo dato pasado entre operaciones en un grafo es pasado en la forma de tensores. La palabra Tensor viene del Latín y significa “que se estira”, es un objeto matemático y se lo llamó así ya que originalmente se utilizó para el estudio del estiramiento de los materiales bajo presión. Pero un significado más contemporáneo, más cerca de nuestro uso es el de array multidimensional. O sea que un tensor no es ni más ni menos que un array como lo conocemos de cualquier lenguaje de programación.

Ejemplo de tensores en TensorFlow:

```
import tensorflow as tf
Scalar = tf.constant([2])
Vector = tf.constant([5,6,2])
Matrix = tf.constant([[1,2,3],[2,3,4],[3,4,5]])
Tensor = tf.constant( [ [[1,2,3],[2,3,4],[3,4,5]] , [[4,5,6],[5,6,7],[6,7,8]] ,
[[7,8,9],[8,9,10],[9,10,11]] ] )
with tf.Session() as session:
    result = session.run(Scalar)
    print "Scalar (1 entry):\n %s \n" % result
    result = session.run(Vector)
    print "Vector (3 entries) :\n %s \n" % result
    result = session.run(Matrix)
    print "Matrix (3x3 entries):\n %s \n" % result
    result = session.run(Tensor)
    print "Tensor (3x3x3 entries) :\n %s \n" % result
```

La salida de este código será:

```

Scalar (1 entry):
[2]

Vector (3 entries) :
[5 6 2]

Matrix (3x3 entries):
[[1 2 3]
 [2 3 4]
 [3 4 5]]

Tensor (3x3x3 entries) :
[[[ 1  2  3]
   [ 2  3  4]
   [ 3  4  5]]

 [[ 4  5  6]
   [ 5  6  7]
   [ 6  7  8]]

 [[ 7  8  9]
   [ 8  9 10]
   [ 9 10 11]]]

```

Operaciones entre Matrices, es decir, entre tensores :-)

```

import tensorflow as tf
Matrix_one = tf.constant([[1,2,3],[2,3,4],[3,4,5]])
Matrix_two = tf.constant([[2,2,2],[2,2,2],[2,2,2]])

#SUMA DE MATRICES
first_operation = tf.add(Matrix_one, Matrix_two)
#SUMA DE MATRICES USANDO EL OPERADOR +
second_operation = Matrix_one + Matrix_two
#MULTIPLICACIÓN DE MATRICES
third_operation = tf.matmul(Matrix_one, Matrix_two)

with tf.Session() as session:
    result = session.run(first_operation)
    print "Defined using tensorflow function :"
    print(result)
    result = session.run(second_operation)
    print "Defined using normal expressions :"
    print(result)
    result = session.run(third_operation)
    print "Defined using tensorflow function :"
    print(result)

```

Variables

Para declarar una variable se usa el comando: `"tf.Variable()"`. Pero además hay que inicializar las variables antes de ejecutar el grafo, y se hace con otro comando:

`"tf.global_variables_initializer()"`

Veamos el ejemplo de un contador muy sencillo:

```
state = tf.Variable(0)    #defino la variable state y le asigno el valor 0
one   = tf.constant(1)    #defino la constante one y le asigno el valor 1
#defino un grafo, es decir una operación, de adición: 1+state
new_value = tf.add(state, one)
#defino una segunda operación, de asignación: state=state+1
update = tf.assign(state, new_value)
#inicializamos las variables (es también una operación que
#debe ser ejecutada en la sesión)
init_op = tf.global_variables_initializer()

with tf.Session() as session:
    session.run(init_op) #Se ejecuta la inicialización de las variables
    #se ejecuta state, que es solo una variable.
    print("originalmente state es:"+str(session.run(state)))
    for _ in range(3):
        session.run(update) #se ejecuta UPDATE 3 veces
        print(session.run(state)) #se imprime el valor de state 3 veces
```

Esto produce la siguiente salida:

```
originalmente state es:0
1
2
3
```

Placeholders

Sirven para tomar datos desde el exterior y ponerlos en variables.

Para ello hay que usar el comando: `"tf.placeholder(datatype)"` en donde datatype es el tipo de dato. Puede ser: entero, flotante, strings o booleanos. También se debe indicar la precisión.

A continuación se muestran los tipos de datos y su descripción

Data type	Python type	Description
DT_FLOAT	tf.float32	32 bits floating point.
DT_DOUBLE	tf.float64	64 bits floating point.
DT_INT8	tf.int8	8 bits signed integer.
DT_INT16	tf.int16	16 bits signed integer.
DT_INT32	tf.int32	32 bits signed integer.
DT_INT64	tf.int64	64 bits signed integer.
DT_UINT8	tf.uint8	8 bits unsigned integer.
DT_STRING	tf.string	Variable length byte arrays. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32 bits floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64 bits floating points: real and imaginary parts.
DT_QINT8	tf.qint8	8 bits signed integer used in quantized Ops.
DT_QINT32	tf.qint32	32 bits signed integer used in quantized Ops.
DT_QUINT8	tf.quint8	8 bits unsigned integer used in quantized Ops.

Ejemplo:

```
a=tf.placeholder(tf.float32) #creamos un placeholder
b=a*2 #definimos una multiplicación

#Se llama a la función run de la sesión como siempre, pero se le pasa un
#argumento extra: feed_dict, en el cual se le pasará un diccionario,
#en donde cada placeholder es seguido de sus datos:
with tf.Session() as sess:
    result = sess.run(b,feed_dict={a:3.5})
    print result
```

Como los datos en TensorFlow son pasados como arrays multidimensionales, se puede pasar cualquier tipo de tensor usando los placeholders


```
dictionary={a: [ [ [1,2,3],[4,5,6],[7,8,9],[10,11,12] ] , [
[13,14,15],[16,17,18],[19,20,21],[22,23,24] ] ] }
```

```
with tf.Session() as sess:
    result = sess.run(b,feed_dict=dictionary)
    print result
```

Operaciones

Las operaciones son nodos en un grafo de TensorFlow, representan las operaciones matemáticas entre tensores. Pueden ser cualquier tipo de operaciones como suma, resta, funciones de activación, etc. Por ejemplo:

- `tf.add(x, y)` (o operador +): ejemplos:
 - `c=a+b`
 - `c=tf.add(a,b)`
- `tf.multiply(x, y)` (o operador *): multiplica escalares, o entre dos matrices multiplica cada elemento en una con su par en la segunda, ejemplo:
 - `c=a*b`
 - `c=tf.multiply(a,b)`
 - `matrixA = tf.constant([[2,3],[3,4]])`
`matrixB = tf.constant([[2,3],[3,4]])`
`C = tf.multiply(matrixA,matrixB)`
`with tf.Session() as session:`
`print(session.run(C))`

```
[[ 4  9]
 [ 9 16]]
```
- `tf.matmul(x, y)` : multiplica matrices, usando las mismas matrices del ejemplo anterior:
 - `matrixA = tf.constant([[2,3],[3,4]])`
`matrixB = tf.constant([[2,3],[3,4]])`
`C = tf.matmul(matrixA,matrixB)`
`with tf.Session() as session:`
`print(session.run(C))`

```
[[13 18]
 [18 25]]
```
- `tf.nn.sigmoid(x)`: funcion Sigma
- `tf.div(x, y)`: división
- `tf.square(x)`: cuadrado
- `tf.sqrt(x)`: raiz cuadrada
- `tf.pow(x, y)`: potencia
- `tf.exp(x)`: exponente
- `tf.log(x)`: logaritmo

- `tf.cos(x)`: coseno
- `tf.sin(x)`: seno

Son como las funciones ordinarias en Python, solo que operan directamente sobre tensores.

Más operaciones matemáticas: [más operaciones](#)

Más funciones y símbolos de TF: https://www.tensorflow.org/api_docs/python/

Ejemplos

A continuación una serie de ejemplos sencillos y no tanto para implementar desde una serie de Fibonacci hasta un perceptrón multicapa tradicional.

Serie de Fibonacci

La serie de Fibonacci, definida como:

$$F_n = F_{n-1} + F_{n-2}$$

Y comienza, con: 1,1,2,3,5,8,13,21...

Hay dos formas, en principio de implementarla con TensorFlow. La primera, más tradicional, más acorde a la forma de programar tradicional:

```
a=tf.Variable(0)
b=tf.Variable(1)
temp=tf.Variable(0)
c=a+b

update1=tf.assign(temp,c)
update2=tf.assign(a,b)
update3=tf.assign(b,temp)

init_op = tf.initialize_all_variables()
with tf.Session() as s:
    s.run(init_op)
    for _ in range(15):
        print(s.run(a))
        s.run(update1)
        s.run(update2)
        s.run(update3)
f = [tf.constant(1),tf.constant(1)]

for i in range(2,10):
    temp = f[i-1] + f[i-2]
    f.append(temp)

with tf.Session() as sess:
    result = sess.run(f)
    print result
```

Y una segunda, pensando un poco más con Tensores, en donde no imprimimos cada número en cada ciclo, sino que simplemente añadimos un elemento más a un Tensor. Cada elemento es un número en la serie de Fibonacci:

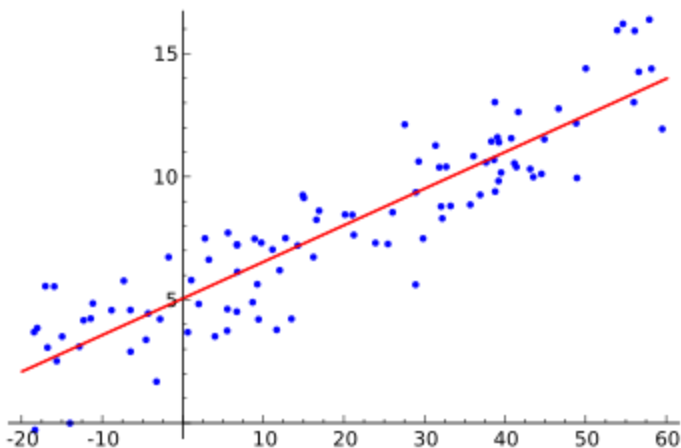
```
f = [tf.constant(1),tf.constant(1)]

for i in range(2,10):
    temp = f[i-1] + f[i-2]
    f.append(temp)

with tf.Session() as sess:
    result = sess.run(f)
    print result
```

Regresión lineal o ajuste lineal

En estadística la regresión lineal o ajuste lineal es un modelo matemático que consiste en utilizar una recta para aproximar un conjunto de observaciones realizadas, para ello cada observación tiene que poder ser representada como un punto en el plano o en bien en un espacio de dimensión N, la recta que mejor se “ajusta” a todos los puntos será aquella cuya distancia a cada uno de estos puntos u observaciones sea mínima.



En la figura se observa un problema tipo, se tiene una serie de puntos azules (observaciones obtenidas) y se necesita saber cómo estos pueden ser aproximados, (también predichos si hablamos de futuras observaciones) utilizando una recta. La recta se calcula tal que la distancia euclidiana desde ella a cada uno de los puntos sea mínima. Como la distancia puede ser negativa o positiva, según si un punto quede debajo de la línea o por encima de esta, se toma directamente el cuadrado de la misma. Hay que entender que la distancia de un punto a la recta es en realidad el error que se está cometiendo al aproximar las observaciones con una línea, por ello minimizar todas las distancias es minimizar el error. Y como utilizamos el cuadrado de la distancia, para que los errores siempre se sumen (ya que sino podrían sumarse y restarse dando como resultado un error igual a cero, cuando no fuese así) el método de aproximación se llama de mínimos cuadrados.

Regresión lineal con Tensor Flow

Primeramente instalamos el módulo **numpy** de python si no lo tenemos instalado:

```
pip install numpy
```

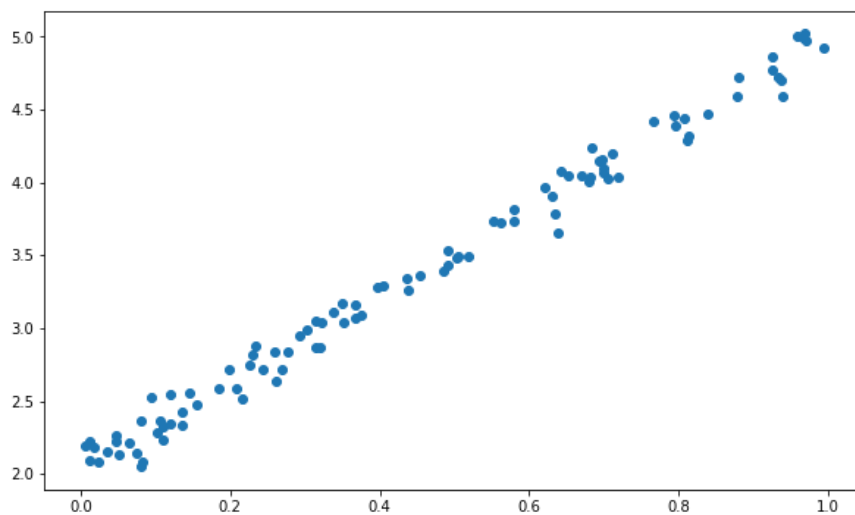
Luego creamos un archivo llamado: “**regresionLineal.py**” y agregamos el siguiente código:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#Las líneas anteriores nos permiten trabajar y poner comentarios en UTF-8
import numpy as np
import tensorflow as tf
import matplotlib.patches as mpatches
```

```
import matplotlib.pyplot as plt

#generamos aleatoriamente 100 números entre 0 y 1, este vector tendrá los números en X
x_data = np.random.rand(100).astype(np.float32)
#definimos un nuevo vector que son los números en Y (dados por la ecuación y=x*3+2)
y_data = x_data * 3 + 2
#Esto agrega ruido a los números en X
y_data = np.vectorize(lambda y: y + np.random.normal(loc=0.0, scale=0.1))(y_data)
#si quieren graficar los puntos, ejecuten las siguientes dos líneas:
plt.scatter(x_data,y_data)
plt.show()
```

Guardamos el archivo y ejecutamos con: “python regresionLineal.py”. Obtendremos un gráfico similar al siguiente:



Sigamos editando el archivo para obtener la regresión lineal:

```
#definimos dos variables a y b de TensorFlow y las inicializamos con un valor
#cualquiera
a = tf.Variable(1.0)
b = tf.Variable(0.2)

#definimos y como la ecuación entre estas variables y todos nuestros X
y = a * x_data + b

#definimos ahora el error (loss) como el error cuadrático medio de la diferencia.
#tf.reduce_mean() básicamente calcula el promedio de todo el tensor y devuelve un
valor #escalar, un tensor de dimensión 1 que es el promedio de todos los valores
loss = tf.reduce_mean(tf.square(y - y_data))

#ahora viene la magia, tenemos que definir un método de optimización, un método que
#optimice el error (la variable loss), usamos descenso por gradiente con una tasa de
#aprendizaje de 0.5
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss) #le decimos al método que minimice los
```

```

#inicializamos variables
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

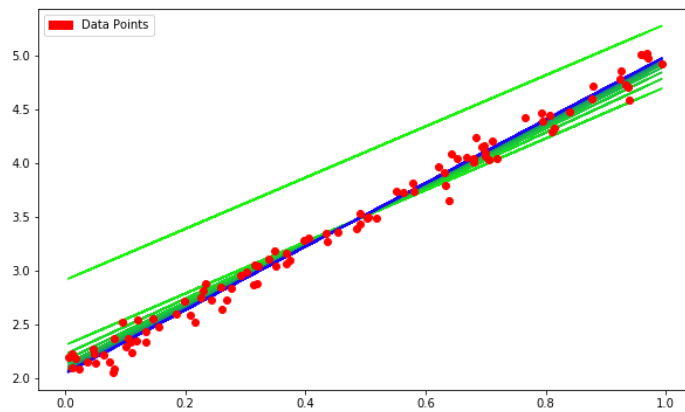
#corremos el algoritmos de entrenamiento
train_data = []
for step in range(100):
    evals = sess.run([train,a,b])[1:]
    if step % 5 == 0:
        print(step, evals)
        train_data.append(evals)

#graficamos las rectas intermedias (que guardamos en train_data )y la recta final
#hallada junto con los puntos
converter = plt.colors
cr, cg, cb = (0.1, 1.0, 0.0)
for f in train_data:
    cb += 1.0 / len(train_data)
    cg -= 1.0 / len(train_data)
    if cb > 1.0: cb = 1.0
    if cg < 0.0: cg = 0.0
    [a, b] = f
    f_y = np.vectorize(lambda x: a*x + b)(x_data)
    line = plt.plot(x_data, f_y)
    plt.setp(line, color=(cr,cg,cb))

plt.plot(x_data, y_data, 'ro')
green_line = mpatches.Patch(color='red', label='Data Points')
plt.legend(handles=[green_line])
plt.show()

```

Guardamos el archivo y ejecutamos. Debería mostrar finalmente un gráfico como el siguiente:



Regresión Logística

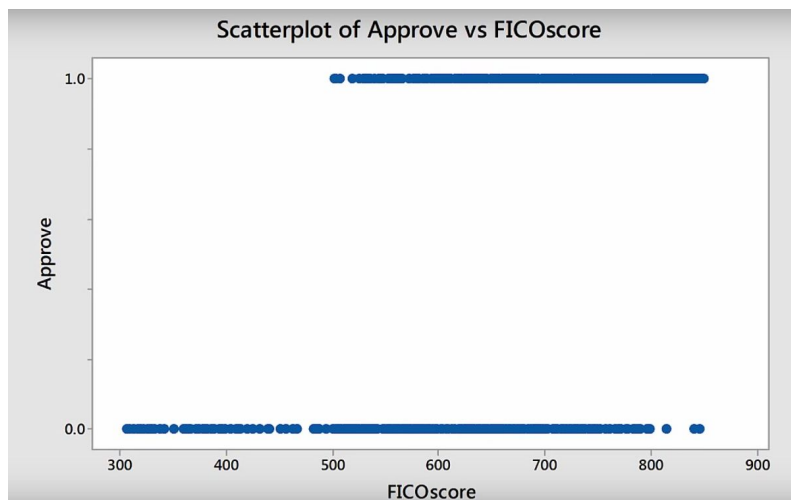
La regresión logística, es otro estimador. A diferencia de la regresión lineal, en donde genero una recta en base a una serie de puntos y por lo tanto puedo realizar predicciones: es decir que para un valor X cualquiera puedo predecir su valor en Y simplemente usando la ecuación encontrada: $y = x \cdot m + b$. En la regresión logística lo que quiero es categorizar, clasificar. Es decir que dado una serie de puntos quiero encontrar una función (no es una recta en este caso) que separe los puntos en dos, en dos conjuntos. Y una vez que la encontré puedo determinar para cualquier valor X futuro, el conjunto al cual pertenecerá. Está asociado a problemas de probabilidad. Veamos un ejemplo típico para ilustrar el estimador:

Una persona quiere comprar una casa y para ello necesita pedir un préstamo hipotecario. Esta persona quiere saber si se lo van a otorgar o no. Pero el único dato fehaciente que tiene es su puntaje crediticio, el cual es de 720. El puntaje crediticio de una persona es un número entre 300 y 850 que mide a grandes rasgos la solvencia de la persona. Sin embargo, cuando alguien va al banco a pedir un préstamo hay otros factores, además del puntaje crediticio, que son tenidos en cuenta por los bancos para determinar si el préstamo debe o no ser otorgado. Sin embargo esos otros factores no son siempre conocidos y las reglas aplicadas solo las conoce el banco.

La única información que se tiene es una lista de puntajes crediticios de otras 1000 personas con el resultado del otorgamiento, es decir si el crédito fue otorgado: 1 o bien si no fue otorgado: 0.

creditScore	approved
655	0
692	0
681	0
663	1
688	1
693	1
699	0
699	1
683	1
698	0
655	1
703	0
704	1
745	1
702	1

Si graficamos los puntajes en el eje X y los resultados en el eje Y, quedara algo como lo siguiente:



Cómo puede observarse, los puntos azules en el sector inferior corresponden a los puntajes que no obtuvieron préstamos y los puntos azules en la parte superior aquellos a los cuales se les otorgó un crédito.

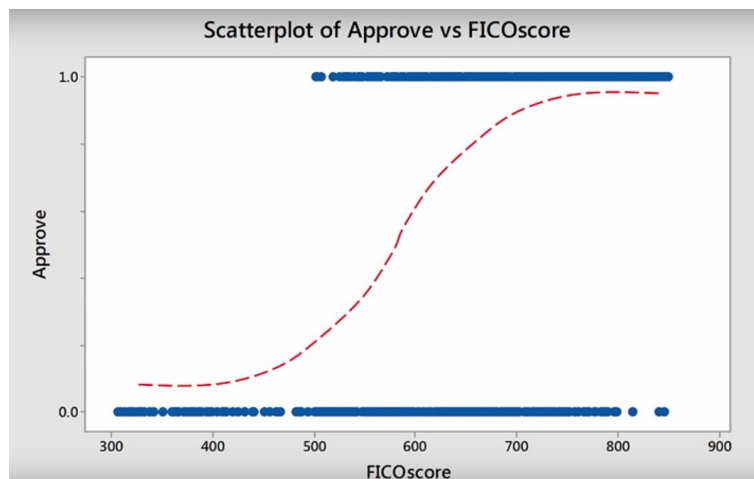
Claramente no es un gráfico que pueda ser aproximado por una línea, como en el caso de la regresión lineal.

En este caso, se buscará encontrar una función de probabilidad, tal que para un puntaje dado me diga cual es la probabilidad de que se otorgue un préstamo.

Para ello se utiliza una función llamada función logística o curva logística. Esta función tiene la forma de una S y está dada por la siguiente ecuación:

$$ProbabilityOfaClass = \theta(y) = \frac{1}{1 + e^{-x}}$$

Si la dibujamos sobre el gráfico anterior quedaría algo así como:



Acá podemos ver que la curva, para cada valor de X, asigna un valor entre 0 y 1 que indica la probabilidad de que el préstamo sea otorgado. Como la curva es creciente, la probabilidad será más alta cuanto el score esté más cerca de 850 y será más baja cuando el score esté cerca de 300. Encontrar los parametros optimos para esta curva consiste en construir un estimador de regresión logística.

A diferencia de la regresión lineal, la regresión logística es un método de clasificación, mientras que la regresión lineal es un método de regresión o predicción. Es decir, la regresión logística se utiliza para saber si una observación cae dentro de una categoría dada, en este caso: “préstamo otorgado”, “préstamo no-otorgado” mientras que la regresión lineal devuelve un valor continuo, es decir un número real.

Regresión logística con TensorFlow

En este ejercicio vamos a trabajar con el conjunto de datos: “iris.csv” que viene con el módulo Pandas de python. Pueden ver el conjunto de datos aquí:

<https://raw.githubusercontent.com/pandas-dev/pandas/master/pandas/tests/data/iris.csv>

Pero no es necesario que lo descarguen ya que viene incluido dentro de la librería de python: Pandas.

Este conjunto de datos tiene 4 atributos de entrada: SepalLength, SepalWidth, PetalLength, PetalWidth, todos ellos propiedades medibles de ciertas flores llamadas Iris. Según estos valores la flor puede ser clasificada en una de estas tres subespecies: “Iris-setosa”, “Iris-versicolor”, “Iris-virginica”.

Instalamos el paquete de python “Pandas” para la manipulación de datos: `pip install pandas`. También los paquetes “sklearn” y “matplotlib” de la misma forma. Puede ser complicado instalar “sklearn”, ya que depende de otras librerías. Pero la instalación de esos paquetes excede a este tutorial.

Luego creamos un archivo llamado: “regresionLogistica.py” y copiamos el siguiente código:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#Las líneas anteriores nos permiten trabajar y poner comentarios en UTF-8

#Importamos las librerías que vamos a usar
import tensorflow as tf #TensorFlow
import pandas as pd # Pandas, para manipulación de datos
#numpy, biblioteca para manejo de números y operaciones matemáticas en python
import numpy as np
import time
from sklearn.datasets import load_iris #Esta función permitirá importar directamente
```

```

# el CSV en un par de arrays llamados "data" y "target"
from sklearn.model_selection import train_test_split #función que permite partir un
# array en dos, para armar conjuntos de entrenamiento y prueba
import matplotlib.pyplot as plt #librería para graficar funciones en python

#Primero que nada cargamos los datos en el CSV, lo almacenamos en una variable iris
iris = load_iris()

#Las variables linguisticas:"Iris-setosa", "Iris-versicolor", "Iris-virginica"
# las importa como números: 0,1,2
iris_X, iris_y = iris.data[:-1,:], iris.target[:-1]

#convertimos los datos de salida, en vectores, así 0 será [1,0,0],
#1 será [0,1,0] y 2 será [0,0,1]
iris_y= pd.get_dummies(iris_y).values

#Finalmente partimos el conjunto en 0.66 para entrenamiento y 0.33 para testing.
# Además añadimos aleatoriedad a la muestra
trainX, testX, trainY, testY = train_test_split(iris_X, iris_y, test_size=0.33,
random_state=42)

#cantidad de valores de entrada, que son 4
numFeatures = trainX.shape[1]

#cantidad de valores de salida, en este caso 3. ya que convertimos las etiquetas
#linguisticas en vectores de 3 dimensiones
numLabels = trainY.shape[1]

# Creamos los Placeholders
#####

# 'None' significa que tensorflow no esperará un número fijo en esa dimensión.
# X representará a los valores de entrada
X = tf.placeholder(tf.float32, [None, numFeatures])

#yGold es la respuesta esperada.
yGold = tf.placeholder(tf.float32, [None, numLabels])

# Pesos y umbrales
#####

# Creamos la matriz de pesos, de 4x3 ya que 4 son los valores de entrada
# y 3 los de salida. Es una variable ya que es la que se irá ajustando en cada ciclo.

weights = tf.Variable(tf.random_normal([numFeatures,numLabels],
                                         mean=0,
                                         stddev=0.01,
                                         name="weights"))

```

```

# Creamos la variable para los umbrales (bias). Esta de 3 dimensiones

bias = tf.Variable(tf.random_normal([1,numLabels],
                                   mean=0,
                                   stddev=0.01,
                                   name="bias"))

#Grafo de tensorflow
#####
#Ahora sí creamos el grafo de Tensor flow que hará lo siguiente:
#  $y = \sigma(WX + b)$  en donde W es la matriz de pesos (4x3), X son los valores de
# entrada (4) y b los umbrales (3)

#multiplicación de matrices
apply_weights_OP = tf.matmul(X, weights, name="apply_weights")

#suma de umbrales
add_bias_OP = tf.add(apply_weights_OP, bias, name="add_bias")

#función SIGMA
activation_OP = tf.nn.sigmoid(add_bias_OP, name="activation")

##Definimos parámetros de entrenamiento
#####

# ciclos de entrenamiento
numEpochs = 700

# definimos el decaimiento de la tasa de aprendizaje, será una exponencial
learningRate = tf.train.exponential_decay(learning_rate=0.0008,
                                           global_step= 1,
                                           decay_steps=trainX.shape[0],
                                           decay_rate= 0.95,
                                           staircase=True)

#El error o costo será la salida de "activation_OP"
#menos lo que definimos como yGold
cost_OP = tf.nn.l2_loss(activation_OP-yGold, name="squared_error_cost")

#Definimos método de entrenamiento: Descenso por gradiente
training_OP = tf.train.GradientDescentOptimizer(learningRate).minimize(cost_OP)

#Sesion de TensorFlow
#####

# Creamos session

```

```

sess = tf.Session()

#Inicializar variables
init_OP = tf.global_variables_initializer()
sess.run(init_OP)

# argmax(activation_OP, 1) devuelve la etiqueta con más probabilidad
# argmax(yGold, 1) devuelve la etiqueta correcta siempre
correct_predictions_OP = tf.equal(tf.argmax(activation_OP,1),tf.argmax(yGold,1))

# calculamos la precisión de nuestro estimador como el promedio
# entre las precisiones correctas. 1 es verdadero y 0 falso según lo anterior,
# ya que es 1 si la precisión fue igual al valor esperado
accuracy_OP = tf.reduce_mean(tf.cast(correct_predictions_OP, "float"))

##### CICLO ENTRENAMIENTO #####
#####

# Inicialización de las variables que vamos a utilizar para ir armando un reporte
cost = 0
diff = 1
epoch_values = []
accuracy_values = []
cost_values = []

# Entrenamos el grafo de TensorFlow
for i in range(numEpochs):
    if i > 1 and diff < .0001:
        print("El cambio en el costo es %g; => hay convergencia."%diff)
        # se hace un break ya que la diferencia entre los últimos
        # errores calculados es muy chica, es decir converge
        break
    else:
        # Corremos el grafo que es: "training_OP" y lo alimentamos
        # con los valores de entrenamiento del CSV
        step = sess.run(training_OP, feed_dict={X: trainX, yGold: trainY})
        # cada 10 ciclos reportamos algo
        if i % 10 == 0:
            # epoc actual
            epoch_values.append(i)
            # obtenemos precisión y costo
            train_accuracy, newCost = sess.run([accuracy_OP, cost_OP], feed_dict={X:
trainX, yGold: trainY})
            # añadimos la precisión a la lista para graficar luego
            accuracy_values.append(train_accuracy)
            # añadimos el costo a la lista de costos para graficar luego
            cost_values.append(newCost)
            # asignamos el último costo y la diferencia con el anterior
            diff = abs(newCost - cost)

```

```

cost = newCost

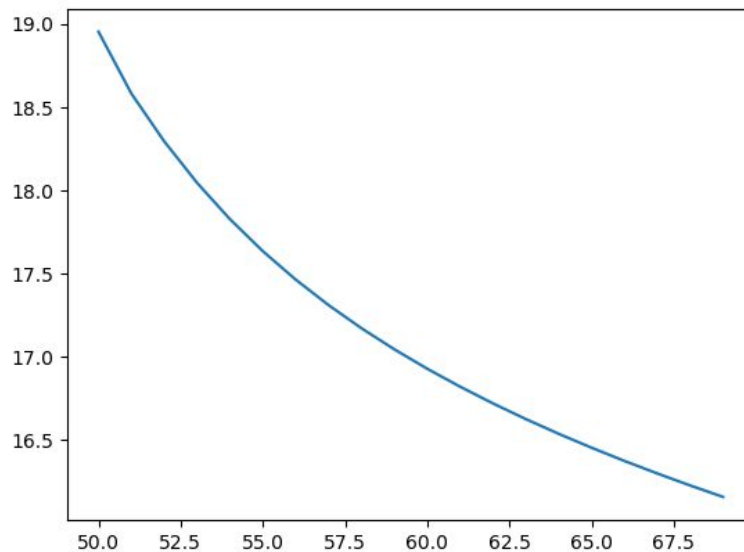
#Imprimimos el reporte
print("paso %d, precisión de entrenamiento %g, costo %g, cambio en el
costo %g"%(i, train_accuracy, newCost, diff))

# ahora, lo probamos con el conjunto de prueba
print("prueba final de precisión en el conjunto de prueba: %s"
%str(sess.run(accuracy_OP,feed_dict={X: testX,yGold: testY})))

#Graficamos como fue cayendo el error
plt.plot([np.mean(cost_values[i-50:i]) for i in range(len(cost_values))])
plt.show()

```

Debería mostrar un gráfico como el siguiente:



Verán que les indica que la precisión es de 0.9, es decir 90%. Y el error fue decreciendo en cada ciclo de entrenamiento como se observa en el gráfico anterior.

Perceptr3n multicapa con Tensor Flow

En el siguiente ejemplo, ver3n un perceptr3n multicapa sencillo, de 4 capas de neuronas. No es una red profunda. La primera es la capa de neuronas de entrada, luego le siguen dos capas ocultas y por 3ltimo una capa de salida.

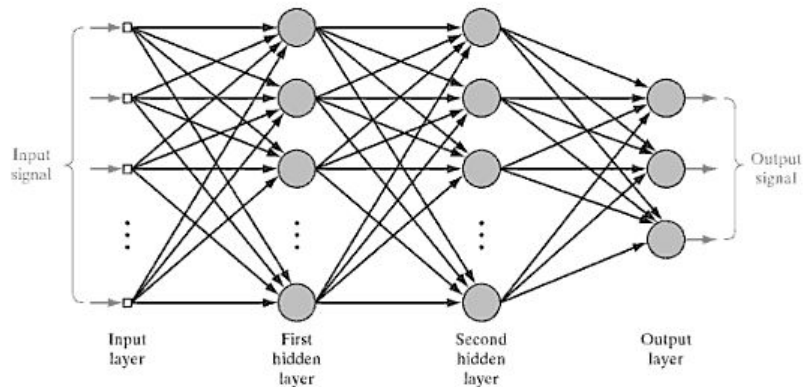
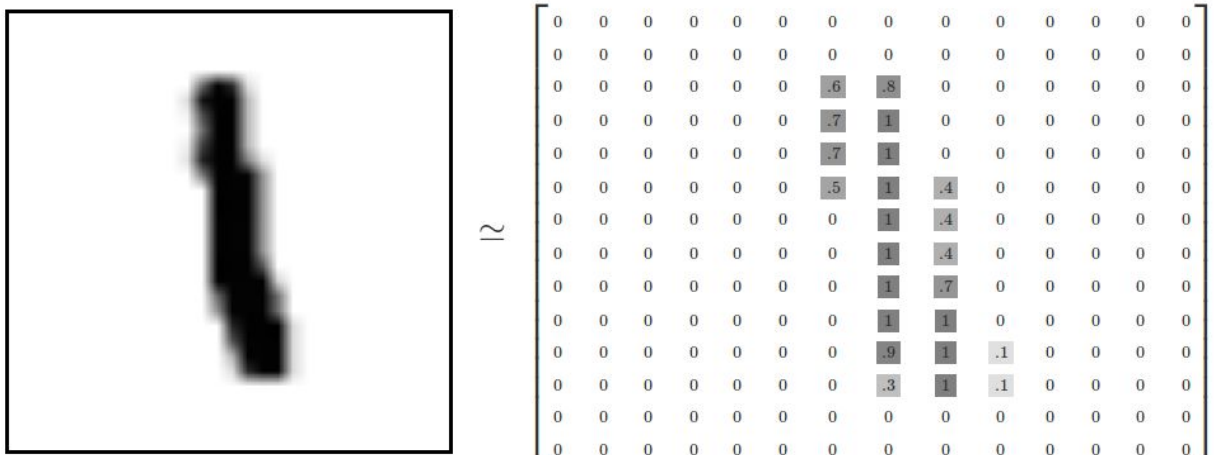


FIGURE 4.1 Architectural graph of a multilayer perceptron with two hidden layers.

Este perceptr3n sirve para reconocer d3gitos manuscritos. Como los que se ven a continuaci3n:



Es decir que los datos de entrada son im3genes. M3s espec3ficamente im3genes en blanco y negro (y gris) de 28 x 28 p3xeles. Si lo vemos m3s de cerca ver3amos algo as3:



Para que se den una idea, los datos de entrada est3n en un archivo de texto y tienen esta forma:

```

0000 0803 0000 ea60 0000 001c 0000 001c
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0312 1212 7e88 af1a
a6ff f77f 0000 0000 0000 0000 0000 0000
1e24 5e9a aafd fd fd fd fd e1ac fd fd c340
0000 0000 0000 0000 0000 0031 eefd fd fd

```

Sin embargo, la misma biblioteca incluye un método: “input_data.read_data_sets” que sabe como leer estos archivos y los devuelve como un objeto con dos atributos: “train” y “test” y a su vez cada uno tiene dos atributos más “images” y “labels”. El atributo “Images” es un array de vectores, y cada vector tiene 784 valores. En cambio “Labels” es un array de escalares, cada escalar es un número de 0 a 9 y presenta el valor que se quiso dibujar. En un pseudo json sería algo así:

```

{
  train:{
    images: [[0,1,0,0....0], [1,1,1,0....0], [0.5,1,0,0....0],.....[0.4,0,0,0....0]],
    labels: [1,3,6,3,9.....1],
  },
  test:{
    images: [[0.5,1,0,0....0], [1,1,1,0....0],
[0.5,1,1,0....0],.....[0.4,0,0,0....0.4]],
    labels: [7,5,6,3,9.....8],
  }
}

```

En definitiva, una red neuronal solo entiende números, así que si bien para nosotros son pequeñas imágenes de 28x28, la red entiende a cada imagen como un vector de 784 neuronas de entrada.

El conjunto de datos es parte de una biblioteca de Python, con lo cual no hay que bajarlo, se descarga solo al ejecutar el ejemplo y se guarda en el directorio: /tmp/data/

El código fuente que copio a continuación pueden obtenerlo de la siguiente URL:

https://github.com/floydhub/tensorflow-examples/blob/master/3_NeuralNetworks/multilayer_perceptron.py


```

""" Neural Network.
A 2-Hidden Layers Fully Connected Neural Network (a.k.a Multilayer Perceptron)
implementation with TensorFlow. This example is using the MNIST database
of handwritten digits (http://yann.lecun.com/exdb/mnist/).
This example is using TensorFlow layers, see 'neural_network_raw' example for
a raw implementation with variables.
Links:
    [MNIST Dataset](http://yann.lecun.com/exdb/mnist/).
Author: Aymeric Damien
Project: https://github.com/aymericdamien/TensorFlow-Examples/
"""

from __future__ import print_function

# Importamos los datos de ejemplo de la biblioteca MNIST
from tensorflow.examples.tutorials.mnist import input_data
#le decimos que los descargue en /tmp/data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=False)

import tensorflow as tf

#parámetros usados para entrenar la red
learning_rate = 0.1 #tasa de aprendizaje
num_steps = 1000    #cantidad de pasos de entrenamiento
batch_size = 128    #cantidad de ejemplos por paso
display_step = 100  #cada cuánto imprime algo por pantalla

# Parámetros para la construcción de la red
n_hidden_1 = 256 # número de neuronas en la capa oculta 1
n_hidden_2 = 256 # número de neuronas en la capa oculta 2
num_input = 784  # MNIST data input (img shape: 28*28)
num_classes = 10 # MNIST clases: dígitos (0-9 dígitos)

# Definimos la red neuronal
def neural_net(x_dict):
    # x_dic es un diccionario con los valores de entrada
    x = x_dict['images'] #en particular vendrán en el campo "image"
    # Conectamos x (la entrada) con la capa oculta 1: Conexión full
    layer_1 = tf.layers.dense(x, n_hidden_1)
    # Conectamos la capa oculta 1, con la capa oculta 2: Conexión full
    layer_2 = tf.layers.dense(layer_1, n_hidden_2)
    # Conectamos la capa oculta 2 con la capa de salida
    out_layer = tf.layers.dense(layer_2, num_classes)
    return out_layer

# Usamos la clase "TF Estimator Template", para definir cómo será el entrenamiento
def model_fn(features, labels, mode):
    # Llamamos a la función anterior para construir la red

```

```

logits = neural_net(features)

# Predicciones
pred_classes = tf.argmax(logits, axis=1)
pred_probab = tf.nn.softmax(logits)

# Si es de predicción devolvemos directamente un EstimatorSpec
if mode == tf.estimator.ModeKeys.PREDICT:
    return tf.estimator.EstimatorSpec(mode, predictions=pred_classes)

# Definimos nuestro error
loss_op = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
    logits=logits, labels=tf.cast(labels, dtype=tf.int32)))
# sparse_softmax_cross_entropy_with_logits: Mide el error de probabilidad
# en tareas de clasificación discretas en las que las clases son mutuamente
# excluyentes (cada entrada está en exactamente una clase)
# reduce_mean: Calcula la media de los elementos a través de las dimensiones
# de un tensor

#Definimos un optimizador, que trabaja por el método de descenso por gradiente
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op,
                              global_step=tf.train.get_global_step())

# Definimos cómo se evaluará la precisión del modelo
acc_op = tf.metrics.accuracy(labels=labels, predictions=pred_classes)

# Finalmente devolvemos un objeto: "EstimatorSpec", indicando todo lo que
# calculamos para el entrenamiento: modo, predicción, error (loss), método de
# entrenamiento y métricas
estim_specs = tf.estimator.EstimatorSpec(
    mode=mode,
    predictions=pred_classes,
    loss=loss_op,
    train_op=train_op,
    eval_metric_ops={'accuracy': acc_op})

return estim_specs

# Construimos un estimador, le decimos que use la función antes definida
model = tf.estimator.Estimator(model_fn)

#pasamos ahora todos los parámetros que necesita la función definida
input_fn = tf.estimator.inputs.numpy_input_fn(
    x={'images': mnist.train.images}, y=mnist.train.labels,
    batch_size=batch_size, num_epochs=None, shuffle=True)
#Entrenamos el modelo
model.train(input_fn, steps=num_steps)

# Evaluamos el modelo

```

```
# Definimos la entrada para evaluar
input_fn = tf.estimator.inputs.numpy_input_fn(
    x={'images': mnist.test.images}, y=mnist.test.labels,
    batch_size=batch_size, shuffle=False)
# Usamos el método 'evaluate' del modelo
e = model.evaluate(input_fn)

print("Precisión en el conjunto de prueba:", e['accuracy'])
```

Referencias

- <https://cognitiveclass.ai/> (Cursos gratuitos online)
- https://www.tensorflow.org/versions/r0.9/get_started/index.html
- <http://jrmeyer.github.io/tutorial/2016/02/01/TensorFlow-Tutorial.html>
- https://www.tensorflow.org/versions/r0.9/api_docs/python/index.html
- https://www.tensorflow.org/versions/r0.9/resources/dims_types.html
- <https://en.wikipedia.org/wiki/Dimension>
- <https://book.mql4.com/variables/arrays>
- [https://msdn.microsoft.com/en-us/library/windows/desktop/dn424131\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/dn424131(v=vs.85).aspx)
- https://github.com/floydhub/tensorflow-examples/blob/master/3_NeuralNetworks/multilayer_perceptron.py (ejemplos en python de TensorFlow)
- https://www.tensorflow.org/versions/r1.2/get_started/mnist/beginners (información sobre los dataset de ejemplo)