Una neurona artificial con TensorFlow

Después de programar nuestra neurona artificial de manera «artesanal», ahora vamos a ver cómo utilizar el módulo TensorFlow para crear esta misma neurona artificial.

1. Breve resumen de TensorFlow

Actualmente, cuando les pedimos a los desarrolladores que nos recomienden una herramienta de Machine Learning y de Deep Learning, suelen respondernos con el framework «TensorFlow» desarrollado por Google.

En 2011, el proyecto Google Brain vio la luz con el objetivo de mejorar los rendimientos comerciales y las experiencias de los usuarios de Google, con ayuda de Deep Learning. En efecto, teniendo en cuenta la cantidad de datos acumulados por Google a través de sus productos Gmail, Google Fotos y su motor de búsqueda, Deep Learning parecía lo más adecuado para mejorar los rendimientos comerciales de Google a partir de estos datos... sin olvidar la faceta de investigación científica. De esta actividad nació el producto interno de Google llamado DistBielef, que fue modificado para hacerlo más rápido y más sólido bajo el nombre de TensorFlow. En 2015, Google decidió hacer pública la biblioteca TensorFlow, con una versión reconocida como estable en 2017.

TensorFlow se articula alrededor de una arquitectura compuesta por tres componentes, que son:

* El pretratamiento de los datos.
* La construcción de los modelos de predicciones.
* La evaluación de los modelos.

TensorFlow toma su nombre del término matemático Tensor, que expresa la facultad de tratar tablas en múltiples dimensiones, y de Flow (Proceso), que permite representar todas las etapas necesarias en la creación de un módulo.

Ahora que sabemos un poco más sobre el módulo TensorFlow, podemos añadirlo a nuestro proyecto para descubrir cómo usarlo en nuestro caso práctico.

La instalación del módulo TensorFlow en la herramienta de desarrollo PyCharm es idéntica a la de los otros módulos que ya está acostumbrado a instalar.

2. Datos de aprendizaje y de pruebas

Una vez instalado el módulo TensorFlow, le invitamos a crear un archivo de script de Python nuevo en su proyecto y a escribir algunas líneas:

import tensorflow as tf

#-------------------------------------

#    DATOS DE APRENDIZAJE

#-------------------------------------

valores\_entradas\_X = [[1., 0.], [1., 1.], [0., 1.], [0., 0.]]

valores\_a\_predecir\_Y = [[0.], [1.], [0.], [0.]]

Después de haber importado el módulo TensorFlow, creamos tablas encargadas de contener las observaciones. Cada dato se especifica como formato decimal para facilitar el tratamiento.

3. Parametrización de la neurona

Ahora vamos a parametrizar la neurona de la siguiente manera:

#-------------------------------------

#    PARÁMETROS DE LA RED

#-------------------------------------

#Variable TensorFLow correspondiente a los valores neuronas

#de entrada

tf\_neuronas\_entradas\_X = tf.placeholder(tf.float32, [None, 2])

#Variable TensorFlow correspondiente a la neurona de salida

(predicción real)

tf\_valores\_reales\_Y = tf.placeholder(tf.float32, [None, 1])

#-- Pesos --

#Creación de una variable TensorFlow del tipo tabla

#que contiene 3 líneas con valores de tipo decimal

#Estos valores se inicializan de manera aleatoria

pesos = tf.Variable(tf.random\_normal([2, 1]), tf.float32)

#-- Sesgo inicializado a 0 --

sesgo = tf.Variable(tf.zeros([1, 1]), tf.float32)

#La suma ponderada es en la práctica una multiplicación de matrices

#entre los valores en la entrada X y los distintos pesos

#la función matmul se encarga de hacer esta multiplicación

sumaponderada = tf.matmul(tf\_neuronas\_entradas\_X,pesos)

#Adición del sesgo a la suma ponderada

sumaponderada = tf.add(sumaponderada,sesgo)

#Función de activación de tipo sigmoide que permite calcular

#la predicción

prediccion = tf.sigmoid(sumaponderada)

#Función de error de media cuadrática MSE

funcion\_error = tf.reduce\_sum(tf.pow(tf\_valores\_reales\_Y-

prediccion,2))

#Descenso de gradiente con una tasa de aprendizaje fijada en 0,1

optimizador =

tf.train.GradientDescentOptimizer(learning\_rate=0.1).minimize

(funcion\_error)

Dentro de los parámetros de TensorFlow hay variables que se pueden inicializar inmediatamente y otras más tarde, es decir, durante las distintas fases del aprendizaje. Este es el caso de los datos en la entrada de cada neurona y de las predicciones esperadas porque se cargan durante el aprendizaje con la finalidad de alimentar (feed) el modelo de aprendizaje. Para este tipo de datos utilizaremos variables del tipo tf.Placeholder.

En cuanto a los pesos y al sesgo, se inicializan desde el principio y de manera aleatoria (random), y serán calculados y modificados durante todo el aprendizaje; por eso se utilizan variables de tipo tf.Variable.

A continuación, en esta sucesión de líneas de programa encontramos la creación de la función de cálculo de la suma ponderada que se hace en dos tiempos.

En el primer tiempo se hace la suma ponderada de los pesos de los valores de entrada mediante una multiplicación de las matrices de los valores de entrada y de los pesos (tf.matmul).

Después, en un segundo tiempo se añade (add) el sesgo (tf.add).

A continuación, viene la definición de la función de activación de tipo sigmoide (tf.sigmoid); luego, la función de cálculo de error establecido calculando la media (tf.reduce\_mean) de los errores cuadráticos. La función tf.pow significa «elevar a la potencia»; entonces podemos deducir que la instrucción:

tf.reduce\_sum(tf.pow(tf\_valores\_reales\_Y-prediccion,2))

permite calcular la diferencia entre el valor predicho y el valor real esperado. Esta diferencia está elevada al cuadrado (concepto cuadrático visto anteriormente en este capítulo).

Finalmente, se añade una línea que permite definir el optimizador, es decir, en nuestro caso la búsqueda de una convergencia minimizando la función de error  mediante el descenso de gradiente y utilizando una tasa de aprendizaje de 0,1.\*.

Ahora nuestra neurona formal está configurada y vamos a poder realizar el aprendizaje.

4. Aprendizaje

A continuación puede ver el programa que permite realizar el aprendizaje con TensorFlow:

#-------------------------------------

#    APRENDIZAJE

#-------------------------------------

#Cantidad de epochs

epochs = 10000

#Inicialización de las variables

init = tf.global\_variables\_initializer()

#Inicio de una sesión de aprendizaje

sesion = tf.Session()

sesion.run(init)

#Para la realización de la gráfica para la MSE

Grafica\_MSE=[]

#Para cada epoch

for i in range(epochs):

  #Realización del aprendizaje con actualización de los pesos

  sesion.run(optimizador, feed\_dict = {tf\_neuronas\_entradas\_X:

valores\_entradas\_X, tf\_valores\_reales\_Y:valores\_a\_predecir\_Y})

  #Calcular el error

  MSE = sesion.run(funcion\_error, feed\_dict =

{tf\_neuronas\_entradas\_X: valores\_entradas\_X,

tf\_valores\_reales\_Y:valores\_a\_predecir\_Y})

  #Visualización de las informaciones

  Grafica\_MSE.append(MSE)

  print("EPOCH (" + str(i) + "/" + str(epochs) + ") -  MSE: "+

str(MSE))

#Visualización gráfica

import matplotlib.pyplot as plt

plt.plot(Grafica\_MSE)

plt.ylabel('MSE')

plt.show()

Como podemos ver, las primeras etapas consisten en inicializar las variables y crear una sesión. A continuación, está el aprendizaje propiamente dicho realizando varias epochs (10 000 en nuestro caso).

Allí donde habíamos escrito varios métodos para la actualización de los pesos, esta se realiza un poco bajo la forma de una caja negra dentro de la función:

sesion.run(optimizador, feed\_dict = {tf\_neuronas\_entradas\_X:

valores\_entradas\_X, tf\_valores\_reales\_Y:valores\_a\_predecir\_Y})

Esta línea le indica a TensorFlow que utilice el conjunto de los datos en la entrada, los almacene en su variable tf\_neuronas\_entradas\_X y haga lo mismo con los valores que hay que predecir almacenándolos dentro de su variable tf\_valores\_reales, después de utilizar el optimizador que realiza la predicción y busca minimizar la función de error. Tenemos derecho a preguntar cómo procede TensorFlow para la actualización de los pesos. Buscando minimizar la función de error que se calcula en función de la suma ponderada, que a su vez se calcula en función de los pesos, TensorFlow realiza operaciones de optimización y actualización de los pesos.

Si ejecutamos ahora el programa, podemos ver que el aprendizaje se realiza correctamente y que la función de error se minimiza con mayor rapidez. Esto se debe a la función de optimización, que indudablemente es más robusta que el programa que hemos escrito antes.

Imagen que contiene Gráfico

Descripción generada automáticamente

*Gráfica de la minimización de la función de error MSE con TensorFlow*

5. Pruebas de predicciones

Ahora vamos a probar el aprendizaje correcto de nuestra neurona formal.

Después del aprendizaje, los distintos pesos están actualizados y guardados en memoria. Para probar el aprendizaje correcto de nuestra neurona formal, vamos a utilizar la neurona de salida encargada de realizar la predicción pasándole cada observación una a una mediante el parámetro feed\_dict.

El código siguiente muestra que pasamos como parámetro la observación para la que deseamos una predicción, almacenándola en la variable de TensorFlow tf\_neuronas\_entradas\_X. Luego realizamos la predicción con ayuda de la neurona de salida llamada prediccion y llamando a la función run de la sesión TensorFlow.

print("--- VERIFICACIONES ----")

for i in range(0,4):

   print("Observación:"+str(valores\_entradas\_X[i])+ " - Esperado:

"+str(valores\_a\_predecir\_Y[i])+" - Predicción:

"+str(**session.run(prediccion, feed\_dict={tf\_neuronas\_entradas\_X:**

**[valores\_entradas\_X[i]]})**))

Estos son los resultados obtenidos correspondientes a las clasificaciones esperadas:

--- VERIFICACIONES ----

Observación:[1.0, 0.0] - Esperado: [**0.0**] - Predicción: [[**0.038227**]]

Observación:[1.0, 1.0] - Esperado: [**1.0**] - Predicción: [[**0.9545917**]]

Observación:[0.0, 1.0] - Esperado: [**0.0**] - Predicción: [[**0.038227**]]

Observación:[0.0, 0.0] - Esperado: [**0.0**] - Predicción: [[**0.00007514158**]]