OR exclusivo (XOR)

Haga memoria: hemos visto que, cuando utilizamos la función lógica OR exclusivo, también llamada XOR, los datos no son linealmente separables. En efecto, hay dos rectas de separación y no una sola. Por lo tanto, una sola neurona no puede tener éxito clasificando los datos.

Gráfico

Descripción generada automáticamente con confianza media

*Representación gráfica de la función lógica OR Exclusivo*

1. ¿Cuántas capas y neuronas necesitamos?

Sabemos que una neurona formal tiene la facultad de clasificar los datos en dos clases. En el caso de XOR, tenemos que hacer dos separaciones de dos clases (dos rectas). Por eso necesitaremos una capa adicional como mínimo.

Diagrama

Descripción generada automáticamente

*Cantidad de capas de neuronas necesarias*

Ahora podemos modelizar nuestra red de neuronas como se indica en la figura siguiente, mostrando las neuronas de entrada, la capa oculta y la neurona de salida, sin olvidar los sesgos. Todas las neuronas están conectadas unas con otras; por eso decimos que nuestra red está «completamente conectada» (*Full connected*).

Imagen que contiene Gráfico

Descripción generada automáticamente

*Redes de neuronas multicapa*

2. Un ejemplo numérico

La programación de estas redes no es muy compleja; las etapas que hay que seguir se corresponden con las de la neurona formal:

* Initialización de los pesos.

Cálculo de la preactivación.

Activación.

* Cálculo del error.

Retropropagación (actualización de los pesos de cada capa).

**a. Datos de aprendizaje**

La tabla que aparece aquí debajo retoma los distintos resultados relacionados con la lógica de OR Exclusivo respetando las siguientes normas:

* El resultado es VERDAD si uno y solo uno de los operandos A y B es VERDAD.
* El resultado es VERDAD si los dos operandos A y B tienen valores distintos.
* El resultado es VERDAD si una cantidad impar de entradas es verdad.

| **OR EXCLUSIVO (XOR)** | | |
| --- | --- | --- |
| A | B | A XOR B |
| 0 | 0 | FALSO |
| 0 | 1 | VERDAD |
| 1 | 0 | VERDAD |
| 1 | 1 | FALSO |

**b. Inicialización de los pesos**

Ahora vamos a inicializar de manera aleatoria los pesos en el intervalo [-1 y 1] y dar el valor 0 a los sesgos:

W11 = -0,165955990594852,

W21 = 0,4406489868843162,

W31 = -0,9997712503653102,

W41 = -0,39533485473632046,

Capa oculta

W12 = -0,7064882183657739,

W22 = -0,8153228104624044,

Sesgo

W51 = 0,

W61 = 0,

W32 = 0,

**c. Carga de los datos de entrada**

Ahora cargamos los datos de entrada correspondientes al primer caso de aprendizaje:

X1 = 0

X2 = 0

ESPERADO = 0

**d. Cálculo de la preactivación de la neurona de salida**

Ahora calculamos el valor de la neurona H1 realizando una preactivación y una activación de la neurona con ayuda de una función de activación de tipo sigmoide:

Pre\_activacion\_H1 = (X1\*W11 + X2\*W31) + (1\*W51)

H1 = sigmoide(Pre\_activacion\_H1)

H1 = sigmoide(0)

H1 = 1 / (1+EXP(0))

H1= 0,5

Luego procedemos de la misma manera para el valor de la neurona H2.

Pre\_activacion\_H2 = (X1\*W21 + X2\*W41) + (1\*W51)

H2 = sigmoide(Pre\_activacion\_H2)

H2 = sigmoide(0)

H2 = 1 / (1+EXP-0))

H2 = 0,5

Después pasamos a la preactivación de la neurona de salida:

Pre\_activacion = (H1\*W12 + H2\*W22) + (1\*W31)

Pre\_activacion = -0,760905514

**e. Cálculo de la activación**

Como solemos hacer, ahora vamos a calcular la activación de la neurona de salida con la ayuda de una función de activación de tipo sigmoide.

Y = sigmoide(-0,760905514)

Y = 1 / (1+EXP(-0,760905514))

Y = 0,318449702

**f. Cálculo del error**

El error se calcula obteniendo la diferencia entre el valor esperado y la predicción realizada:

Error = 0 - 0,318449702

Error = -0,318449702

**g. Actualización de los pesos**

La actualización de los pesos se efectúa tal y como lo hemos hecho para la red formal: calculando el gradiente, el valor de ajuste y partiendo de la neurona de predicción hacia las neuronas de entrada. Hay que destacar que hemos elegido una tasa de aprendizaje de 0,1.

Este es el cálculo para la actualización del peso W12:

Gradiente = -1 \* error \* Y \* (1-Y) \* H1

Gradiente = 0,03455808

Valor\_Ajuste = Tasa de aprendizaje \* Gradiente

Valor\_Ajuste = 0,1 \* Gradiente

Valor\_Ajuste = 0,00345581

Nuevo W12 = W12 - Valor\_Ajuste

Nuevo W12 = -0,70994403

Hemos procedido de la misma manera para actualización de los distintos pesos y obtenemos los siguientes resultados:

| **W12** | |  | **W22** | |
| --- | --- | --- | --- | --- |
| H1 | 0,5 |  | H2 | 0,5 |
| Y | 0,3184497 |  | Y | 0,3184497 |
| Error | -0,3184497 |  | Error | -0,3184497 |
| Gradiente | 0,03455808 |  | Gradiente | 0,03455808 |
| tasa aprendizaje | 0,1 |  | tasa aprendizaje | 0,1 |
| valor ajuste | 0,00345581 |  | valor ajuste | 0,00345581 |
| W12 | -0,70648822 |  | W22 | -0,81532281 |
| Nuevo W12 | -0,70994403 |  | Nuevo W22 | -0,81877862 |
| **W32** | |  | **W61** | |
| SESGO | 1 |  | SESGO | 1 |
| Y | 0,3184497 |  | Y | 0,3184497 |
| Error | -0,3184497 |  | Error | -0,3184497 |
| Gradiente | 0,06911616 |  | Gradiente | 0,06911616 |
| tasa aprendizaje | 0,1 |  | tasa aprendizaje | 0,1 |
| valor ajuste | 0,00691162 |  | valor ajuste | 0,00691162 |
| W32 | 0 |  | W61 | 0 |
| Nuevo W32 | -0,00691162 |  | Nuevo W61 | -0,00691162 |

| **W51** | | |  | | **W41** | |
| --- | --- | --- | --- | --- | --- | --- |
| SESGO | 1 | |  | | X2 | 0 |
| Y | 0,3184497 | |  | | Y | 0,3184497 |
| Error | -0,3184497 | |  | | Error | -0,3184497 |
| Gradiente | 0,06911616 | |  | | Gradiente | 0 |
| tasa aprendizaje | 0,1 | |  | | tasa aprendizaje | 0,1 |
| valor ajuste | 0,00691162 | |  | | valor ajuste | 0 |
| W51 | 0 | |  | | W41 | -0,39533485 |
| Nuevo W51 | -0,00691162 | |  | | Nuevo W41 | -0,39533485 |
| **W31** | | | |  | **W21** | |
| X2 | | 0 | |  | X1 | 0 |
| Y | | 0,3184497 | |  | Y | 0,3184497 |
| Error | | -0,3184497 | |  | Error | -0,3184497 |
| Gradiente | | 0 | |  | Gradiente | 0 |
| tasa aprendizaje | | 0,1 | |  | tasa aprendizaje | 0,1 |
| valor ajuste | | 0 | |  | valor ajuste | 0 |
| W31 | | -0,99977125 | |  | W21 | 0,44064899 |
| Nuevo W31 | | -0,99977125 | |  | Nuevo W21 | 0,44064899 |

| **W11** | |
| --- | --- |
| X1 | 0 |
| Y | 0,3184497 |
| Error | -0,3184497 |
| Gradiente | 0 |
| tasa aprendizaje | 0,1 |
| valor ajuste | 0 |
| W11 | -0,16595599 |
| Nuevo W11 | -0,16595599 |

Una vez actualizados los pesos, podemos cargar el siguiente caso de aprendizaje.

Como hemos comprobado, el hecho de añadir una capa complementaria no ha aportado complejidad a los cálculos. Sin embargo, hay que considerar adecuadamente la activación de cada neurona usando una función de activación.

3. Programar con TensorFlow

Ahora vamos a programar esta red neuronal con TensorFlow.

Para hacerlo, le invitamos a retomar el proyecto realizado en el capítulo anterior y a añadirle un archivo nuevo de script de Python al que llamaremos Perceptron\_Multicapa.py.

Este es el código que debe insertar:

import tensorflow as tf

import numpy as np

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

#    DATOS DE APRENDIZAJE

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

#Los datos se transforman en decimales

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

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

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

#    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])

**#Cantidad de neuronas en la capa oculta**

**nbr\_neuronas\_capa\_oculta = 2**

#PESOS

#Los primeros están 4 : 2 en la entrada (X1 y X2) y 2

pesos por entrada

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

**#los pesos de la capa oculta están 2 : 2 en la entrada**

**(H1 y H2) y 1 peso por entrada**

**pesos\_capa\_oculta = tf.Variable(tf.random\_normal([2, 1]),**

**tf.float32)**

#El primer sesgo contiene 2 pesos

sesgo = tf.Variable(tf.zeros([2]))

**#El segundo sesgo contiene 1 peso**

**sesgo\_capa\_oculta = tf.Variable(tf.zeros([1]))**

#Cálculo de la activación de la primera capa

#cálculo de la suma ponderada (tf.matmul) con ayuda de los datos

X1, X2, W11,W12,W31,W41 y del sesgo

#después aplicación de la función sigmoide (tf.sigmoid)

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

+ sesgo)

**#Cálculo de la activación de la capa oculta**

**#cálculo de la suma ponderada (tf.matmul) con ayuda de los datos**

**H1, H2, W12,W21 y del sesgo**

**#después aplicación de la función sigmoide (tf.sigmoid)**

**activacion\_capa\_oculta = tf.sigmoid(tf.matmul(activacion,**

**peso\_capa\_oculta) + sesgo\_capa\_oculta)**

#Función del error de media cuadrática MSE

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

**activacion\_capa\_oculta,2))**

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

optimizador =

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

(funcion\_error)

#Cantidad de epochs

= 100000

#Inicialización de las variables

init = tf.global\_variables\_initializer()

#Inicio de una sesión de aprendizaje

session = tf.Session()

session.run(init)

#Para la realización del gráfico para la MSE

Grafica\_MSE=[]

#Para cada epoch

for i in range(epochs):

  #Realización del aprendizaje con actualización de los pesos

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

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

  #Calcular el error

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

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

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

  #Visualización de la información

  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()

session.close()

Hemos destacado en negrita las partes que corresponden a la implantación de la capa oculta en nuestra red neuronal. Afectan especialmente a las siguientes funciones:

* La creación de los pesos de la capa oculta.
* La creación del sesgo relacionado con la capa oculta.
* La consideración de la capa oculta en la función de error.

Dejando a un lado algunas adiciones y modificaciones, el programa es idéntico al que hemos utilizado para la programación de la neurona formal. Según estas líneas, también podemos apreciar la facilidad de creación de una red neuronal con TensorFlow.

La figura que aparece a continuación muestra la evolución de la optimización del error que disminuye a lo largo de los aprendizajes.

Forma

Descripción generada automáticamente con confianza baja

*Gráfica de la curva de error obtenida con TensorFlow*

Ahora podemos realizar algunas predicciones para verificar el aprendizaje de nuestro algoritmo añadiendo las siguientes líneas de programa:

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(activacion\_capa\_oculta,

feed\_dict={tf\_neuronas\_entradas\_X: [valores\_entradas\_X[i]]})))

Realizamos una predicción para los cuatro casos de nuestro conjunto de datos recurriendo a la última capa de nuestra red: le pasamos el caso de prueba en forma de parámetros:

session.run(activacion\_capa\_oculta,

feed\_dict={tf\_neuronas\_entradas\_X: [valores\_entradas\_X[i]]})

Entonces obtenemos los siguientes resultados, que muestran un buen aprendizaje de nuestra red neuronal:

--- VERIFICACIONES ----

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

[[**0.02842014**]]

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

[[**0.9734256**]]

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

[[**0.9678324**]]

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

[[**0.02508838**]]