Programamos nuestra primera neurona formal «From Scratch»

Es muy probable que tenga que leer los párrafos anteriores varias veces para comprenderlos totalmente. Por eso vamos a programar nuestra primera neurona formal como complemento del ejemplo numérico que hemos realizado antes. El objetivo es comprender bien toda la mecánica del aprendizaje mediante un ejemplo concreto.

Como ya tiene práctica, dejamos que cree un proyecto nuevo en el editor Pycharm usted solo y que añada un archivo de script nuevo al que llamaremos perceptron.py.

Para realizar el proyecto necesitaremos las bibliotecas matplotlib y numpy.

1. Datos de aprendizaje

Lo primero que vamos a hacer es crear nuestros datos de aprendizaje y definir la tasa de aprendizaje:

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

#    OBSERVACIONES Y PREDICCIONES

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

observaciones\_entradas = array([

                             [1, 0],

                             [1, 1],

                             [0, 1],

                             [1, 0]])

predicciones = array([[0],[1], [0],[0]])

2. Definición de los pesos

Ahora pasamos a la generación de los pesos de forma aleatoria, contenidos en el intervalo de valor [-1,1]:

#Generación de los pesos en el intervalo [-1;1]

random.seed(1)

limiteMin = -1

limiteMax = 1

w11 = (limiteMax-limiteMin) \* random.random() + limiteMin

w21 = (limiteMax-limiteMin) \* random.random() + limiteMin

w31 = (limiteMax-limiteMin) \* random.random() + limiteMin

No olvidamos el sesgo, que tendrá valor 1 y peso igual a 0.

#El sesgo

sesgo = 1

wb = 0

3. Gestión de los hiperparámetros

Ahora vamos a definir los hiperparámetros, que representan la cantidad de epoch y la tasa de aprendizaje.

#Tasa de aprendizaje

tsAprendizaje = 0.1

#Cantidad de epoch

epochs = 300

4. Programación de funciones útiles

El programa que aparece más abajo retoma las distintas funciones que necesitaremos y que se enumeran a continuación:

* El cálculo de la suma ponderada.
* El cálculo de la función de activación de tipo sigmoide.
* El cálculo del error lineal.
* El cálculo del gradiente.
* El cálculo del valor de ajustes del peso.
* El cálculo del valor del peso.
* El cálculo de la función de error media cuadrática (MSE).

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

#       FUNCIONES ÚTILES

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

def suma\_ponderada(X1,W11,X2,W21,B,WB):

   return (B\*WB+( X1\*W11 + X2\*W21))

def funcion\_activacion\_sigmoide(valor\_suma\_ponderada):

   return (1 / (1 + exp(-valor\_suma\_ponderada)))

def funcion\_activacion\_relu(valor\_suma\_ponderada):

   return (max(0,valor\_suma\_ponderada))

def error\_lineal(valor\_esperado, valor\_predicho):

   return (valor\_esperado-valor\_predicho)

def calculo\_gradiente(valor\_entrada,prediccion,error):

   return (-1 \* error \* prediccion \* (1-prediccion) \*

valor\_entrada)

def calculo\_valor\_ajuste(valor\_gradiente,

tasa\_aprendizaje):

   return (valor\_gradiente\*tasa\_aprendizaje)

def calculo\_nuevo\_peso (valor\_peso, valor\_ajuste):

   return (valor\_peso - valor\_ajuste)

def calculo\_MSE(predicciones\_realizadas,

predicciones\_esperadas):

   i=0;

   for prediccion in predicciones\_esperadas:

       diferencia = predicciones\_esperadas [i] -

predicciones\_realizadas[i]

       cuadradoDiferencia = diferencia \* diferencia

       suma = suma + cuadradoDiferencia

   media\_cuadratica = 1 / (len(predicciones\_esperadas)) \*

suma

   return media\_cuadratica

5. Pasamos al aprendizaje

Ahora que disponemos de todo lo que necesitamos, podemos pasar a la fase de aprendizaje.

Para llevar a cabo esta fase de aprendizaje, vamos a realizar varias épocas (epoch), es decir, varios pasajes completos del conjunto de las observaciones contenidas en nuestro conjunto de datos mediante nuestra neurona formal. Para cada observación, vamos a hacer una predicción y calcular el error para a continuación proceder a la actualización de los pesos sinápticos.

Este es el código vinculado con el aprendizaje:

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

#    APRENDIZAJE

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

for epoch in range(0,epochs):

   print("EPOCH ("+str(epoch)+"/"+str(epochs)+")")

   prediccions\_realizadas\_durante\_epoch = [];

   predicciones\_esperadas = [];

   numObservacion = 0

   for observacion in observaciones\_entradas:

       #Carga de la capa de entrada

       x1 = observacion[0];

       x2 = observacion[1];

       #Valor de predicción esperado

       valor\_esperado = predicciones[numObservacion][0]

       #Etapa 1: Cálculo de la suma ponderada

       valor\_suma\_ponderada =

suma\_ponderada(x1,w11,x2,w21,sesgo,wb)

       #Etapa 2: Aplicación de la función de activación

       valor\_predicho =

funcion\_activacion\_sigmoide(valor\_suma\_ponderada)

       #Etapa 3: Cálculo del error

       valor\_error =

error\_lineal(valor\_esperado,valor\_predicho)

       #Actualización del peso 1

       #Cálculo del gradiente del valor de ajuste y del

peso nuevo

       gradiente\_W11 =

calculo\_gradiente(x1,valor\_predicho,valor\_error)

       valor\_ajuste\_W11 = calculo\_valor\_ajuste(gradiente\_W11,tsAprendizaje)

       w11 = calculo\_nuevo\_peso(w11,valor\_ajuste\_W11)

       # Actualización del peso 2

       gradiente\_W21 = calculo\_gradiente(x2, valor\_predicho,

valor\_error)

       valor\_ajuste\_W21 =

calculo\_valor\_ajuste(gradiente\_W21, tsAprendizaje)

       w21 = calculo\_nuevo\_peso(w21, valor\_ajuste\_W21)

       # Actualización del peso del sesgo

       gradiente\_Wb = calculo\_gradiente(sesgo, valor\_predicho,

valor\_error)

       valor\_ajuste\_Wb =

calculo\_valor\_ajuste(gradiente\_Wb, tsAprendizaje)

       wb = calculo\_nuevo\_peso(wb, valor\_ajuste\_Wb)

       print("     EPOCH (" + str(epoch) + "/" + str(epochs) + ") -

Observacion: " + str(numObservacion+1) + "/" +

str(len(observaciones\_entradas)))

       #Almacenamiento de la predicción realizada:

predicciones\_realizadas\_durante\_epoch.append(valor\_predicho)

predicciones\_esperadas.append(predicciones[numObservacion][0])

       #Paso a la observación siguiente

       numObservacion = numObservacion+1

   MSE = calculo\_MSE(predicciones\_realizadas\_durante\_epoch,

predicciones)

   Grafica\_MSE.append(MSE[0])

   print("MSE: "+str(MSE))

Estas líneas no son muy difíciles de entender. Si se ejecuta este programa, podemos ver que la función de error disminuye con el paso del tiempo, lo que demuestra que nuestra neurona está aprendiendo correctamente.

EPOCH (297/300)

    EPOCH (297/300) -  Observacion: 1/4

    EPOCH (297/300) -  Observacion: 2/4

    EPOCH (297/300) -  Observacion: 3/4

    EPOCH (297/300) -  Observacion: 4/4

**MSE : [0.08400651]**

EPOCH (298/300)

    EPOCH (298/300) -  Observacion: 1/4

    EPOCH (298/300) -  Observacion: 2/4

    EPOCH (298/300) -  Observacion: 3/4

    EPOCH (298/300) -  Observacion: 4/4

**MSE : [0.08390084]**

EPOCH (299/300)

    EPOCH (299/300) -  Observacion: 1/4

    EPOCH (299/300) -  Observacion: 2/4

    EPOCH (299/300) -  Observacion: 3/4

    EPOCH (299/300) -  Observacion: 4/4

**MSE : [0.08379514]**

¿Hemos llegado al punto de convergencia?

6. Buscando el punto de convergencia

Para ver si hemos alcanzado el punto de convergencia, primero vamos a mostrar la curva de la función de error.

Para conseguirlo, antes de la función de aprendizaje añadimos una lista que nos permitirá almacenar los distintos valores de la función de errores MSE durante toda la fase de aprendizaje:

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

#       GRÁFICA

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

Grafica\_MSE=[]

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

#    APRENDIZAJE

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

A continuación, al final del aprendizaje (después del bucle For epoch…), creamos y mostramos la gráfica:

import matplotlib.pyplot as plt

plt.plot(Grafica\_MSE)

plt.ylabel('MSE')

plt.show()

Así conseguimos la curva representada por la figura siguiente, donde se constata una disminución del error y luego un ligero escalón para a continuación retomar el descenso. Pero eso no nos indica si hemos alcanzado la convergencia, solo podemos deducir que, con una función de error que muestra un índice del 8 % y según la apariencia de la curva, todavía podemos esperar resultados mejores.

Gráfico

Descripción generada automáticamente

*Gráfica de la función de error MSE*

Le invitamos a modificar la cantidad de epochs aumentando hasta 1 000 000 y a volver a lanzar el aprendizaje (cuidado, puede llevar bastante tiempo). Hemos elegido este valor enorme de manera voluntaria para mostrarle que nuestro algoritmo sigue mejorando su aprendizaje, aunque la mejora es muy pequeña porque la curva de la función de error parece estabilizarse.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

*Gráfica de la función de error con 1 000 000 epochs*

Si utilizamos la función zoom de la ventana de visualización de la gráfica y vamos a la época 300 000, constatamos que estamos cerca del 0 % de errores.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

*Gráfica de la función de error cercana a cero con 300 000 epochs*

7. Pruebas de predicciones

Ahora probamos a hacer algunas predicciones para verificar el correcto funcionamiento de nuestra neurona artificial.

Primero vamos a cambiar la cantidad de epochs para ponerla en 300 000.

#Cantidad de épocas

epochs = 300000

Añadimos algunas líneas que nos permitirán conocer los pesos procedentes del aprendizaje:

print ("Pesos finales: " )

print ("W11 = "+str(w11))

print ("W21 = "+str(w21))

print ("Wb = "+str(wb))

Luego hacemos una predicción:

print()

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

print ("PREDICCIÓN ")

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

x1 = 0

x2 = 1

#Etapa 1: Cálculo de la suma ponderada

valor\_suma\_ponderada = suma\_ponderada(x1,w11,x2,w21,sesgo,wb)

#Etapa 2: Aplicación de la función de activación

valor\_predicho = funcion\_activacion\_sigmoide(valor\_suma\_ponderada)

print("Predicción del [" + str(x1) + "," + str(x2) + "]")

print("Predicción = " + str(valor\_predicho))

Pesos finales:

W11 = 9.163546255436527

W21 = 9.163526933525727

Wb = -13.830737198803797

--------------------------

PREDICCIÓN

--------------------------

Prediccion del [0,1]

Prediccion = 0.00931094371781672

Según nuestra tabla de admisión en la universidad, la admisión no es posible cuando no se supera uno de los dos exámenes. Por lo tanto, el valor esperado es 0 y la neurona nos ha predicho 0,0093. ¡Eso está muy bien!

Ahora probamos con otros valores para X1 y X2.

print()

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

print ("PREDICCIÓN ")

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

x1 = 1

x2 = 1

Pesos finales:

W11 = 9.163546255436527

W21 = 9.163526933525727

Wb = -13.830737198803797

--------------------------

PREDICCIÓN

--------------------------

Prediccion del [1,1]

Prediccion = 0.9889731719767871

Esta vez la predicción está muy cerca de 1, como se esperaba, porque le hemos indicado que se han superado los dos exámenes.

¡Nuestra neurona formal es funcional!

Como podemos comprobar, con unas líneas de código hemos conseguido programar una neurona formal capaz de realizar predicciones sobre datos linealmente separables (insistimos en esta característica).

¡Tenga cuidado con el sobreajuste! Si aumentamos la cantidad de epochs, La neurona cometerá menos errores, pero no podrá ser generalizable. Le invitamos a consultar el capítulo Machine Learning y los Pokémon: segunda parte, para obtener más información sobre este tema.