Trabajo Práctico 1 De las Neuronas artificiales al Deep Learning

Objetivo: Conocer y programar la estructura básica de una neurona artificial, y un perceptron multicapa. Resolver con pequeñas redes neuronales problemas de regresión y clasificación simples. Fortalecer los conceptos de forward y backward pass, y de backprogation. Realizar un ciclo de aprendizaje y de análisis de resultados.

1.1 La Neurona Artificial

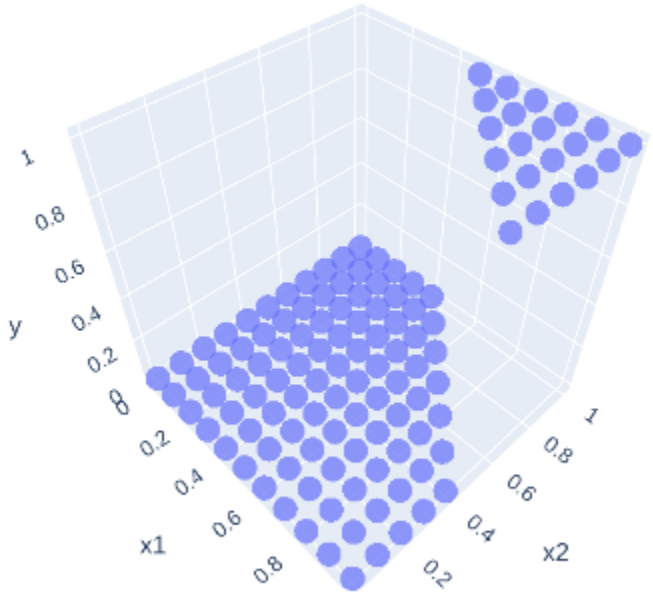
Comenzaremos creando una subrutina que cree una neurona artificial. La misa estará definida de el número de inputs (xi, i=1 hasta a n), sus pesos correspondientes (wi), el bias (b), y la función de activación.

```
#Codigo para 1 Neurona Artificial
import numpy as np

# Inicializa la neurona con pesos al azar
def initialize_neuron(input_size):
    weights = np.random.rand(input_size)
    bias = np.random.rand(1)
    return weights, bias

# Sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

#ReLu
def relu(x):
    return np.maximum(0, x)

# Define the step function
def step(x):
    return np.where(x >= 0, 1, 0)

# Forward pass through the neuron
def forward_pass(inputs, weights, bias):
    z = np.dot(inputs, weights) + bias

#permite elegir que tipo de funcion utilizar
    #return step(z)
    #return relu(z)
    return sigmoid(z)
```

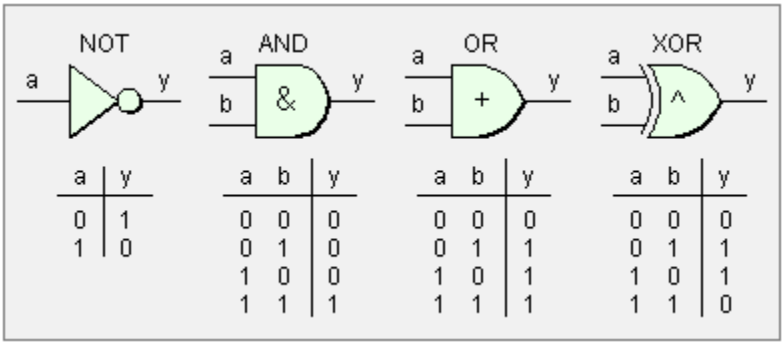
```
# ejemplo de uso ***********************************


# seteo de valores a mano
weights = np.array([1, 1])
bias = np.array([-1.5])
output=forward_pass([0.5,1],weights,bias)[0]
print(output)
```

Restringiendo los valores de entrada de cada neurona entre 0 y 1 analice brevemente como se comporta la neurona con inputs de 1, 2 y 3 dimensiones (¿que pasa en dimensiones superiores?).
Pruebe cambiar (y programar) a otras funciones de activacion como ReLu, tanh etc.

A modo de ejemplo mostramos la salida para una neurona con una step-function y 2 inputs.



Con una neurona, especialmente si se usa la función escalón como función de activación. es posible diseñar los circuitos lógicos AND, OR, NOT.



a	y
0	1
1	0

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

Verifique que con los siguientes paràmetros (w1=1, w2=1, b=−1.5) la neurona resuelve correctamente la función AND, por ejemplo: para el input [1,1] el output debería ser 1. ¿Se anima a programar y verificar las funciones OR, o NOT?.
¿Que pasa si en lugar de utilizar como activación “step” utiliza la sigmoidea o ReLu?

1.1.2 Enseñandole (o entrenando) una neurona.

El proceso de aprendizaje o entrenamiento consiste en consiste en ajustar los pesos (w) y sesgos (b) para minimizar el error en las predicciones. Este proceso utiliza un conjunto de datos de entrenamiento donde cada entrada tiene una salida esperada conocida. Primero, las entradas se procesan hacia adelante (forward pass) para generar predicciones. Luego, se calcula la diferencia entre las predicciones y los valores reales mediante la “loss function”. Este error se propaga hacia atrás (*backpropagation*) y luego se ajustan iterativamente los pesos y sesgos para reducir el error. Este ciclo de propagación hacia adelante, cálculo de pérdida y ajuste continúa hasta que el modelo alcanza un nivel aceptable de precisión o se cumplen los criterios de parada.

Entonces para entrenar una neurona debemos a) tener el código que càlcule los gradientes respecto de cada parametro (wi, b), para un ejemplo (o un batch) y b) tener el còdigo que dados los gradientes actualice los parametros. Para calcular los gradientes debemos ademas definir la loss function (LF).
Como ejemplo vamos a entrenar nuestra neurona para que a partir de 2 inputs binarios realice la operación lógica AND. Còmo el output es binario utilizaremos como LF la entropía cruzada binaria:

$$LF = -(y \cdot \log(yp) + (1-y) \cdot \log(1-yp))$$

Donde “y” es el valor “esperado” del output, e yp es el valor predicho.

A continuaciòn se presenta el código que permite calcular el gradiente para nuestra neurona con la LF correspondiente. Es importante notar que para este caso NO podemos usar la step function como función ya que no es derivable, por eso uitilizamos la sigmoidea.

Codigo

```
#Gradiente para una neurona simple con activacion sigmoidea

def compute_gradients(inputs, weights, bias, y_true):
    # Forward pass
    z = np.dot(weights, inputs) + bias
    y_pred = sigmoid(z)

    # Compute loss
    loss = binary_cross_entropy_loss(y_pred, y_true)

    # Backward pass
    d_loss = binary_cross_entropy_derivative(y_pred, y_true)  # Gradient of loss w.r.t. y_pred
    d_activation = sigmoid_derivative(z)  # Gradient of sigmoid w.r.t. z

    # Gradients for weights and bias
    d_weights = d_loss * d_activation * inputs  # Chain rule: dL/dw
    d_bias = d_loss * d_activation              # Chain rule: dL/db

    return d_weights, d_bias, loss, y_pred


# Sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Derivative of the sigmoid function
def sigmoid_derivative(x):
    s = sigmoid(x)
    return s * (1 - s)

# Binary cross-entropy loss function
def binary_cross_entropy_loss(y_pred, y_true):
    return -(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

# LF derivative
def binary_cross_entropy_derivative(y_pred, y_true):
    return y_pred - y_true
```

Un breve análisis del código muestra que este tiene funciones auxiliares para calcular la LF y su derivada, y la función de activación y su derivada. Luego para calcular el gradiente primero realiza las predicciones (forward pass), luego calcula la LF, y luego combina las derivadas y el output para calcular los gradientes.

Elija parámetros cercanos a los correctos (por ejemplo w1=5, w2=3, b=-7) y calcule para cada uno de los cuatro input posibles ([0,0][1,0][0,1][1,1]) el output de la neurona, el valor de la LF y los 3 gradientes.
Repita la operación pero cambiando los parámetros (por ejemplo eligiendolos al azar).
Analice y comente los resultados.

Una vez obtenidos los gradientes dL/dw1, dl/dw2 y dL/db1 recordemos debemos utilizar una función de optimización para actualizar los parámetros. Básicamente la actualizaciòn de parámetros responde a la siguiente fórmula:

w1_new = w1_old - dL/dw1 * LR

Donde LR es el “Learning Rate” el parámetro que controla la velocidad a la que la neurona aprende (un valor positivo pequeño)

Elija unos valores de w1,w2,b de partida y uno de los casos testigo. Determine la LF y los gradientes. Luego aplique la formula para actualizar los pesos (puede probar diferentes LR). Verifique que cuando los parámetros son actualizados en la dirección correcta la LF baja y el valor predicho se acerca al real.

Le dejamos unos resultados de “ejemplo”

params w, b: [1 1],[1]
Inputs: [0 1] output: [0.88079708]
Loss: [2.12692801]
Gradients: [0. 0.09247804], [0.09247804]

LR=1
params w, b: [1. 0.9],[0.9]
Inputs: [0 1] output: [0.85814894]
Loss: [1.95297761]
Gradients: [0. 0.1044619], [0.1044619]

LR=5
params w, b: [1. 0.5],[0.5]
Inputs: [0 1] output: [0.73105858]
Loss: [1.31326169]
Gradients: [0. 0.14373484], [0.14373484]

Analice ahora qué pasa para los gradientes obtenidos con los “mismos” parámetros pero para cada una de las 4 posibles entradas-salidas del AND. ¿Que le parece?

Como es de esperarse cada dato de entrada genera valores de los gradientes diferentes, e incluso estos pueden ser contradictorios. Es por ello que las optimizaciones de parámetros se

hacen utilizando el promedio de los gradientes obtenidos para todo (batch) un pequeño conjunto (mini batch(de datos de entrenamiento, y este valor promedio es el que se utiliza para actualizar los parámetros.

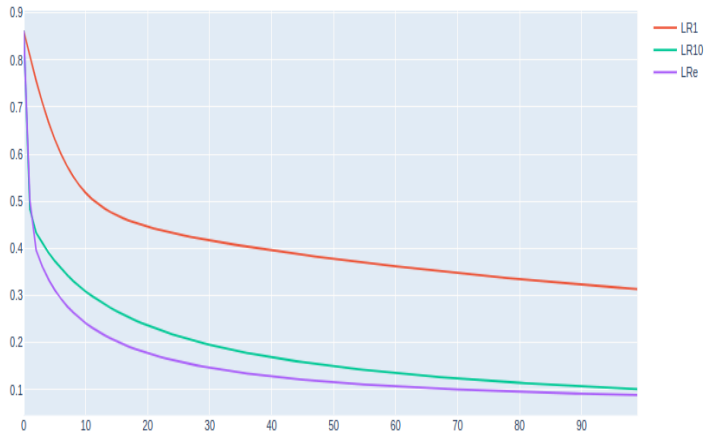
Con toda esta información ya estamos listos para entrenar nuestra neurona para que pueda resolver el problema **AND**.

Ejercicio 1.1: a) Utilizando una neurona de dos entradas (x1, x2) y la función de activacion sigmoidea, programe un código que tomando valores de w1,w2 y b de entrada al azar, determine los gradientes y los vaya actualizando hasta encontrar un valor optimo, de modo que la neurona ejecute el AND.
Controle del mismo la cantidad de ciclos de entrenamiento, y el LR.
Consejo: pruebe diferentes valores de LR, e incluso puede utilizar una funciòn donde el LR decrece a medida que avanza el entrenamiento. Ademas utilice los gradientes promedio (batch) para la actualizaciòn de los parámetros
Para monitorear el entrenamiento calcule (y grafique) el avance de la LF en función del número de ciclo y calcule y muestre los valores finales que obtiene la neurona para cada una de las 4 condiciones de entrada del AND.
Analice los resultados

b) Que pasa si en lugar de una sigmoidea utiliza ReLu?
c) Se anima a repetir el proceso pero para la función lógica “OR”

d) ¿que pasa si intenta resolver el “XOR”?

A modo de ejemplo a continuaciòn le mostramos un plot de la LF vs el paso de entrenamiento para 3 LR diferentes, y al final los valores obtenidos para las 4 opcones del AND.



valores finales :w1=4, w2= 4, b=-6
LF=0.08862609

input: 0 0 out: 0.002
input: 0 1 out: 0.10
input: 1 0 out: 0.10
input: 1 1 out: 0.87