UNIVERSIDAD AUSTRAL DE CHILE

FACULTAD DE CIENCIAS DE LA INGENIERÍA

INGENIERÍA CIVIL ELECTRÓNICA

ELEP 233 - VISIÓN ARTIFICIAL Y REDES NEURONALES

DESARROLLO DE TAREA N°6

PROFESOR:
Gustavo Schleyer.

INTEGRANTES:
Ángel Andrade
Jorge Palavecino.

12 de junio de 2023

Índice general

El presente documento desarrolla la tarea 6 del ramo ELEL233. El repositorio en GitHub de la actividad puede encontrarse en `https://github.com/Atrabilis/UACH/tree/main/Trabajos/Vision%20artificial/Tarea%206`.

Pregunta 1

El propósito de este apartado es representar la función lógica AND e IMPLICA utilizando la implementación del perceptrón con NumPy, Esquematizar el perceptrón obtenido, graficar la recta de decisión y ver qué ocurre si se modifica los valores de entrada para ver si son bien clasificados.

La esquematización de los perceptrones obtenidos es la siguiente:

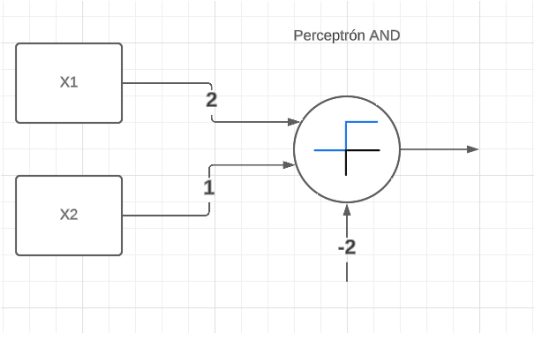

Figura 1: Esquematización del perceptrón AND

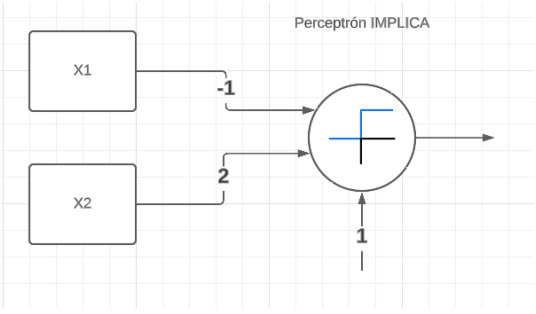

Figura 2: Esquematización perceptrón Implica

Figura 3: Esquematización de los perceptrones

Las rectas de decisión se muestran a continuación:

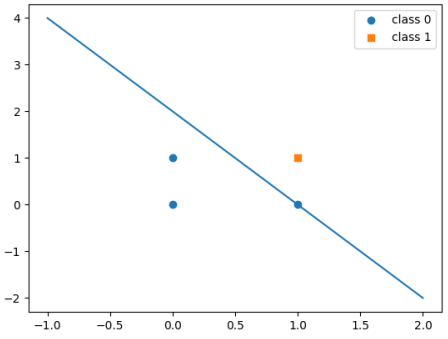

Figura 4: Linea de decisión del perceptrón AND

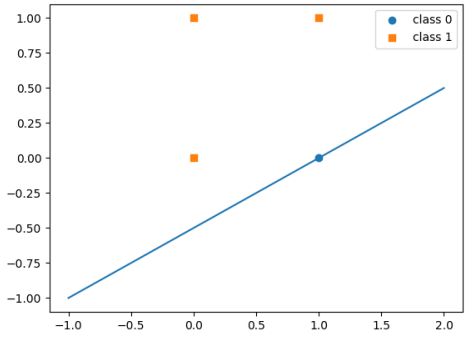

Figura 5: Linea de decisión del perceptrón Implica

Figura 6: Rectas de decisión de los perceptrones

Una vez que los perceptrones están entrenados, tomarán cualquier punto que esté a un lado de la recta o hiper plano y le asignara el valor correspondiente a su entrenamiento, por lo que es difícil contestar a si son clasificados de buena o mala manera. Por ejemplo, el punto (.5,.5) no tendría sentido en el contexto de una puerta lógica, pero según la recta de decisión AND seria clasificada como 0 y como 1 por la recta de decisión implica, esto no es necesariamente malo ya que el perceptrón solo hace la tarea para la que ha sido entrenada. Por otro lado, si las entradas no están debidamente ordenadas con respecto a sus labels, el perceptrón aprenderá de manera errónea, ya que la tabla de verdad que le estaremos pasando no es la correcta.

Pregunta 2

el objetivo de este apartado es comparar losa resultados obtenidos en el apartado anterior con los de la implementación en PyTorch: las rectas obtenidas son las siguientes.

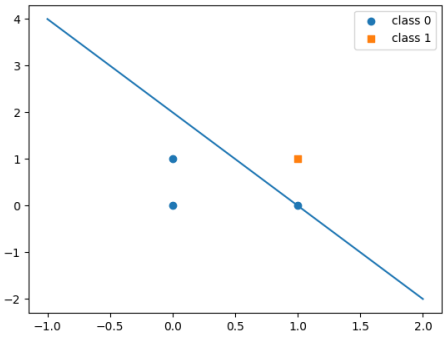

Figura 7: Linea de decisión del perceptrón AND con PyTorch

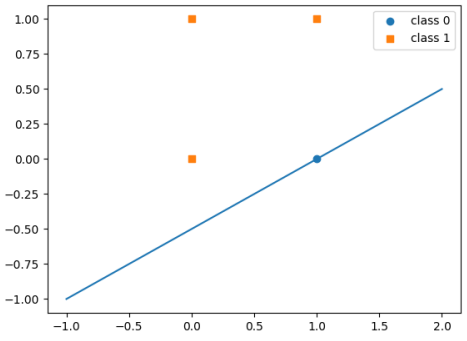

Figura 8: Linea de decisión del perceptrón Implica con PyTorch

Figura 9: Rectas de decisión de los perceptrones implementados con Torch

Los resultados son idénticos, incluyendo los pesos y bias obtenidos.

Pregunta 3

El objetivo de este apartado es representar la función lógica $(A \vee B) \rightarrow (C \wedge D)$ con un perceptrón implementado en numpy. Se entrenó el perceptrón con las primeras 12 entradas de la función lógica y se probó con las ultimas 4. Los resultados fueron los siguientes:

```
Precisión (A ∨ B) -> (C ∧ D) = 75.0%

Pesos de (A ∨ B) -> (C ∧ D) = [[-1.]
 [-1.]
 [ 1.]
 [ 2.]]

bias del perceptron:
[1.]

Outputs del perceptrón con las entradas restantes:
[[0]
 [1]
 [0]
 [1]]
```

Figura 10: Resultados obtenidos después del entrenamiento

Cabe observar que el vector de salida debería ser (0,0,0,1), es decir, el perceptrón devolvió la salida indicada con un $75\,\%$ de precisión. por lo que se puede intuir que no existe un hiper plano que separe linealmente los datos.

la esquematización del perceptrón es la siguiente:

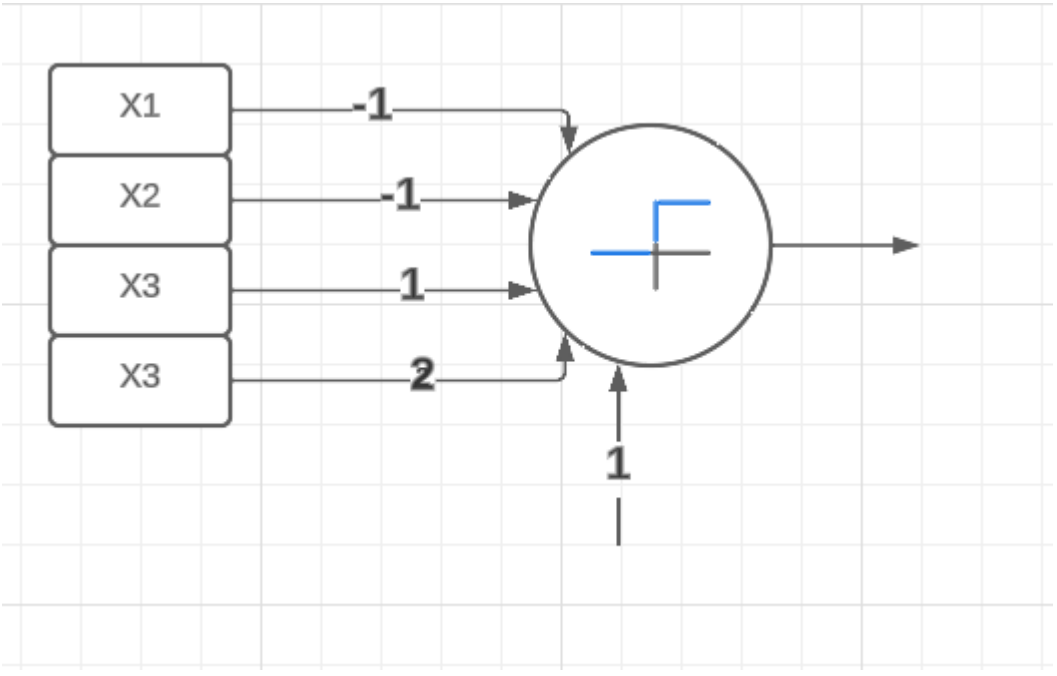

Figura 11: Esquematización del perceptrón obtenido.

Pregunta 4

El objetivo de este apartado es implementar el mismo perceptrón con PyTorch. Los resultados son idénticos:

```
Precisión (A ∨ B) -> (C ∧ D) = 75.0%

Pesos de (A ∨ B) -> (C ∧ D) = tensor([[-1.],
        [-1.],
        [ 1.],
        [ 2.]])

bias del perceptron:
tensor([1.])

Outputs del perceptrón
tensor([[0.],
        [1.],
        [0.],
        [1.]])
```

Figura 12: Resultados obtenidos después del entrenamiento

Cabe destacar que para ambos entrenamientos se utilizaron 1000 iteraciones. Si comparamos el desempeño de los perceptrones implementados en numpy y pytorch utilizando todos los datos y relativos al número de veces que se les entrena, obtenemos lo siguiente (se obtuvo la misma gráfica para ambas implementaciones):

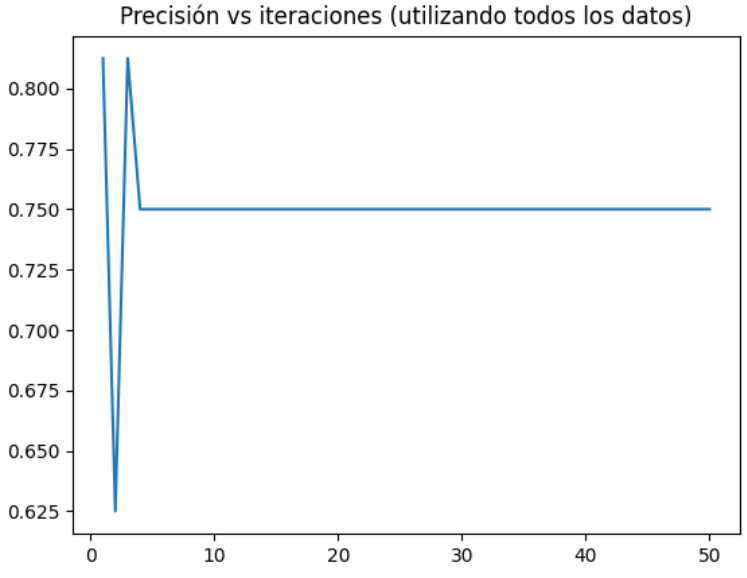

Figura 13: Precisión obtenida en función de las iteraciones de aprendizaje.

Se puede observar que antes de las 10 iteraciones ambos modelos convergen a un 75 % de precisión.

Pregunta 5

El objetivo de este apartado es ver si es posible implementar la puerta lógica XOR o la equivalencia. La respuesta es que no, debido a que los datos no son linealmente separables, por lo que no existe una linea o hiperplano que pueda separar los datos, por ejemplo estos son los datos de la puerta lógica XOR:

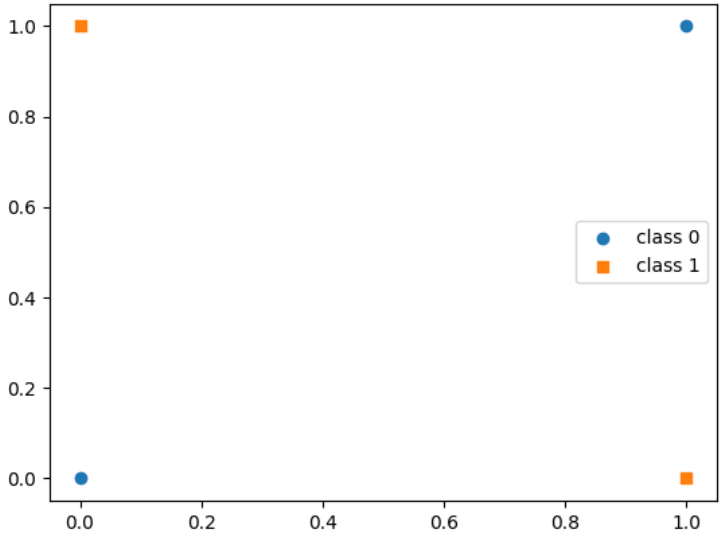

Figura 14: Datos y clases de la puerta lógica XOR, no existe linea que pueda separar una clase de otra.

Sin embargo es posible implementar estas funciones utilizando multicapas en la red neuronal para crear distintas regiones de decisión, de manera análoga a los circuitos digitales, donde para crear una puerta lógica XOR se pueden utilizar una combinación de puertas OR y AND por etapas.

Pregunta 6

En el contexto de las puertas lógicas, necesitamos salidas que sean ceros o unos. La función signo devolverá -1 cuando la salida sea menor o igual a cero y 1 cuando sea positiva, el único cambio en la implementación seria verificar si la salida obtenida es -1 convertirla a 0, lo cual puede realizarse en la clase Perceptron si se desea o en el programa de ejecución.