1 Resolviendo el problema XOR utilizando una red neuronal multicapa (DNN)

Ahora que ya hemos metido un poquito mano a Keras, vamos a demostrar que Vapnik estaba equivocado, vamos a resolver el problema XOR mediante el uso de redes neuronales. Sabemos que una sola capa no podría resolver este problema, sin embargo, lo podremos resolver fácilmente utilizando más de una capa.

In [1]:
```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

print(tf.__version__)
```

executed in 10.1s, finished 18:29:03 2021-10-18

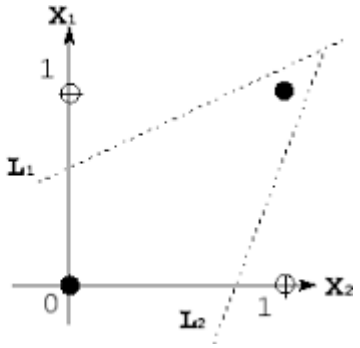
```
2.6.0
```

Vamos a definir el problema XOR con 4 ejemplos de entrada. Nótese que a pesar de que utilizamos una entrada binaria la salida de la red va a ser un número real.

X va a ser nuestro input bidimensional (x1, x2) e Y va a estar formado por las etiquetas correspondientes, que obtendremos siguiendo la función XOR.

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

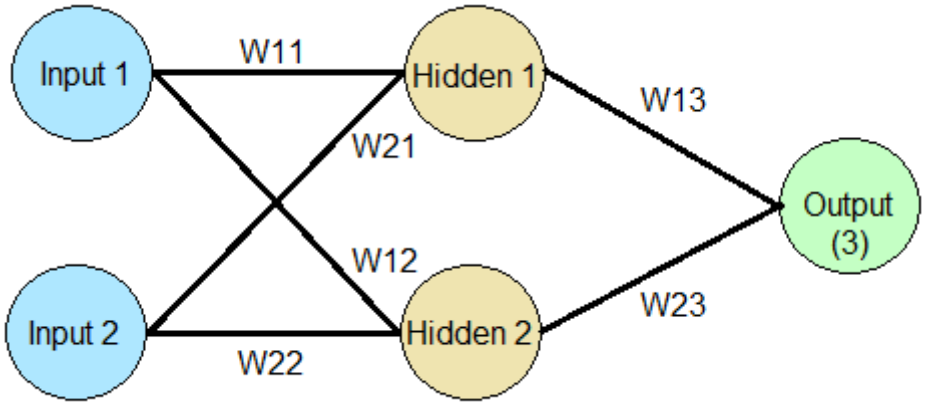
$Y = X_1 \oplus X_2$



In [2]:
```
train_data = np.array([[0,0],[0,1],[1,0],[1,1]])
train_labels = np.array([[0],[1],[1],[0]])
```

executed in 24ms, finished 18:29:04 2021-10-18

Definimos ahora la topología de nuestra DNN. En este caso 2 - 2 - 1



In [3]:
```
n_input_nodes = 2
n_hidden_nodes = 2
n_output_nodes = 1
```

executed in 10ms, finished 18:29:04 2021-10-18

Ha llegado el momento de construir la red neuronal que resuelva nuestro problema!

TODO 1: define una red neuronal con la topología 2 - 2 - 1. Utiliza "sigmoid" como la función de activación no lineal de la capa oculta [1 pto]:

- ***Respuesta:***
  - Creamos el modelo con un Sequential y creamos dos capasa Dense o densas, con lo valores indicados y la función de activación sigmoide como se nos indica en el enunciado. Notese que la primera capa de input o entrada no se decalara, solo se le dice a la capa interna cuantas features o propiedades entran.
  - En la capa de salida al ser un ejemplo de regresión, solo existe una neurona de salida. Como la función de activación no puede ser una softmax, ya que esta función es para casos de clasificación, elegimos una sigmoid ya que el rango de valores que maneja es entre (0,1) y se ajusta a lo que esperamos.

In [4]:
```
# model = keras.Sequential([


# ])

# Importamos las capas y modelos que vamos a necesitar para este worksheet
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Creamos el modelo con un Sequential y creamos dos capasa Dense o densas, con lo valores
# indicados y la función de activación sigmoide como se nos indica en el enunciado.
# Notese que la primera capa de input o entrada no se decalara, solo se le dice a la capa interna
#cuantas features o propiedades entran.

model = Sequential()
model.add(Dense(n_hidden_nodes, activation='sigmoid',input_dim=n_input_nodes,name='Hidden'))

# En la capa de salida al ser un ejemplo de regresión, solo existe una neurona de salida.
# Como la función de activación no puede ser una softmax, ya que esta función es para casos de
# clasificación, elegimos una sigmoid ya que el rango de valores que maneja es entre (0,1) y se ajusta a lo
# que esperamos.

model.add(Dense(n_output_nodes, activation='sigmoid',name='Output'))

```

executed in 1.12s, finished 18:29:05 2021-10-18

TODO 2: muestra qué pinta tiene nuestra red neuronal (pista, ver summary en la doc. de keras) [1 pto]

In [5]:
```
# Con la función summary mostramos la arquitectura de la red neuronal.
model.summary()
```

executed in 26ms, finished 18:29:05 2021-10-18

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
Hidden (Dense)               (None, 2)                 6         
_________________________________________________________________
Output (Dense)               (None, 1)                 3         
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

- ***Respuesta:*** Vemos que la red tiene una complejidad o dimensión de 9 parámteros y se divide en:
  - **dense**: Contiene 2 neuronas correspondientes según el gráfico a Hidden1 y Hidden2. Y 6 parametros correspondientes a los 4 pesos W11,W12,W13,W14 y dos Bias el Bias_Hidden1 y el Bias_Hidden2.
  - **dense_1 (salida)**: Contiene 1 neurona y 3 parámtetros, correspondientes a 2 pesos W13,W23 y al Bias_Ouput3.

TODO 3: ahora tenemos que compilar el modelo. Vamos a utilizar sgd como nuestro optimizador, mean squared error como nuestra función de coste, y utilizaremos como métrica accuracy. Además, pon el learning rate de sgd a 0.1. (Pista, mira cómo hemos compilado el modelo en el ejemplo anterior) [1 pto]

- ***Respuesta:*** Creamos el optimizador importando la clase SGD. Uno de los parámetros del contructor del optimizador es el learning_rate, que según se nos indica en el enunciado es 0.1

In [6]:
```
from tensorflow.keras.optimizers import SGD
opt = SGD(learning_rate=0.1)

model.compile(loss='mean_squared_error',
                    optimizer=opt,
                    metrics=['accuracy'])
```

executed in 26ms, finished 18:29:05 2021-10-18

TODO 4: Toca entrenar el modelo, utiliza la función fit con 1000 épocas. [1 pto]

- ***Respuesta:*** Entrenamos el módelo, previamente compilado en el apartado anterior, parametrizando 1000 epochs. Observamos que el accuracy o precisión es relativamente bajo, solo nos da un 75% en algunas ocasiones y baja al 50% de precisión. Esto indica que posblemente no se capaz de calcular bien todas las combianaciones de XOR.

In [7]:
```
my_epochs=1000
my_verbose = 1
model.fit(train_data, train_labels, epochs=my_epochs, verbose=my_verbose)
```

executed in 15.5s, finished 18:29:20 2021-10-18

```
1/1 [==============================] - 0s 10ms/step - loss: 0.2558 - accuracy: 0.7500
Epoch 16/1000
1/1 [==============================] - 0s 8ms/step - loss: 0.2557 - accuracy: 0.7500
Epoch 17/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.2557 - accuracy: 0.7500
Epoch 18/1000
1/1 [==============================] - 0s 9ms/step - loss: 0.2556 - accuracy: 0.7500
Epoch 19/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.2555 - accuracy: 0.7500
Epoch 20/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.2554 - accuracy: 0.7500
Epoch 21/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.2553 - accuracy: 0.7500
Epoch 22/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.2552 - accuracy: 0.7500
Epoch 23/1000
1/1 [==============================] - 0s 6ms/step - loss: 0.2552 - accuracy: 0.7500
Epoch 24/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.2551 - accuracy: 0.7500
Epoch 25/1000
```

TODO 5: Por último, imprime las salidas que obtendríamos para nuestros datos de entrenamiento (pista, ver predict) [1 pto]

- ***Respuesta :*** Como podiamos avanzar vemos que no ha acertado todos los resultados.

In [8]:
```
print (model.predict(train_data).round())
```

executed in 438ms, finished 18:29:21 2021-10-18

```
[[0.]
 [0.]
 [1.]
 [1.]]
```

- Seguramente estos resultados se podrían mejorar jugando con los hiperparámetros, subiendo el número de neuronas en la capa hidden,

cambiando el optimizador, bien con el learning_rate, poniendo un valor más bajo o subiendo el número de epochs o ciclos de entrenamiento.

In [9]:

```
from tensorflow.keras.optimizers import Adam

def exec_NN(my_train_data, my_train_labels,my_hidden_nodes=2,my_opt='SGD',my_epochs=1000,my_lr=0.1,my_verbose = 1)

    # Creamos el modelo con un Sequential y creamos dos capasa Dense o densas, con lo valores
    # indicados y la función de activación sigmoide como se nos indica en el enunciado.
    # Notese que la primera capa de input o entrada no se decalara, solo se le dice a la capa interna
    #cuantas features o propiedades entran.
    print(f'Parametros:')
    print(f'Neuronas capa Hidden: {my_hidden_nodes}')
    print(f'Optimizador: {my_opt}')
    print(f'Epochs: {my_epochs}')
    print(f'lr: {my_lr}')

    model_nn = Sequential()
    model_nn.add(Dense(my_hidden_nodes, activation='sigmoid',input_dim=n_input_nodes,name='Hidden'))

    # En la capa de salida al ser un ejemplo de regresión, solo existe una neurona de salida.
    # Como la función de activación no puede ser una softmax, ya que esta función es para casos de
    # clasificación, elegimos una sigmoid ya que el rango de valores que maneja es entre (0,1) y se ajusta a lo
    # que esperamos.

    model_nn.add(Dense(n_output_nodes, activation='sigmoid',name='Ouput'))

    # Con la función summary mostramos la arquitectura de la red neuronal.
    model_nn.summary()

    if my_opt=='Adam':
        opt = Adam(learning_rate=my_lr)
    else:
        opt = SGD(learning_rate=my_lr)

    # Compilamos el modelo.

    model_nn.compile(loss='mean_squared_error', optimizer=opt, metrics=['accuracy'])

    #Entrenamos el modelo con los parámetros indicados.

    model_nn.fit(my_train_data, my_train_labels, epochs=my_epochs, verbose=my_verbose)

    # evaluamos el modelo
```

```
    scores = model_nn.evaluate(my_train_data, my_train_labels)

    print(f"{model_nn.metrics_names[1]} : {scores[1]*100:.2f}")

    print (model_nn.predict(my_train_data).round())

    return None

```

executed in 38ms, finished 18:29:21 2021-10-18

Ajustando 4 neuronas en la capa oculta, optimizador Adam y 250 epochs:

In [10]:
```
exec_NN(train_data, train_labels,my_hidden_nodes=4,my_opt='Adam',my_epochs=250,my_verbose=0)
```

executed in 2.69s, finished 18:29:23 2021-10-18

```
Parametros:
Neuronas capa Hidden: 4
Optimizador: Adam
Epochs: 250
lr: 0.1
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Hidden (Dense)               (None, 4)                 12
_________________________________________________________________
Ouput (Dense)                (None, 1)                 5
=================================================================
Total params: 17
Trainable params: 17
Non-trainable params: 0
_________________________________________________________________
1/1 [==============================] - 0s 279ms/step - loss: 5.1758e-04 - accuracy: 1.0000
accuracy : 100.00
[[0.]
 [1.]
 [1.]
 [0.]]
```

[Avanzado/Opcional] TODO 6: Por último, imprime los pesos entrenados del modelo e intenta explicar con tus palabras cómo funcionan. [0.5 pto]

In [11]:
```
# Para obtener los pesos recurrimos a la propiedad wights del modelo,
# donde se va guardando las matrices de pesos y bias (sesgo)

for weight in model.weights:
    print(f'CAPA: {weight.name.split ("/")[0]} TIPO MATRIZ: {weight.name.split ("/")[1]}')
    print(weight.value())
    print('\n')
```

executed in 25ms, finished 18:29:23 2021-10-18

```
CAPA: Hidden TIPO MATRIZ: kernel:0
tf.Tensor(
[[-0.7803771   0.735316  ]
 [ 0.43463126 -0.3693844 ]], shape=(2, 2), dtype=float32)


CAPA: Hidden TIPO MATRIZ: bias:0
tf.Tensor([ 0.17193617 -0.04442494], shape=(2,), dtype=float32)


CAPA: Output TIPO MATRIZ: kernel:0
tf.Tensor(
[[-0.725258  ]
 [ 0.24629456]], shape=(2, 1), dtype=float32)


CAPA: Output TIPO MATRIZ: bias:0
tf.Tensor([0.22297463], shape=(1,), dtype=float32)
```

- ***Respuesta :*** Evaluando las matrices de pesos y bias de cada capa o Dense, podemos determinar que tomando como referencia el dibujo de la red al inicio del notebook ha asignado:
  - Los siguientes *pesos** :

$$\begin{pmatrix} W11 & W12 \\ W21 & W22 \end{pmatrix} = \begin{pmatrix} -0.7803771 & 0.735316 \\ 0.43463126 & -0.3693844 \end{pmatrix} \quad (1)$$

$$\begin{pmatrix} W13 \\ W23 \end{pmatrix} = \begin{pmatrix} 0.17193617 \\ -0.04442494 \end{pmatrix} \tag{2}$$

- Los siguientes Bias\sesgos* :

$$\begin{pmatrix} BiasHidden1 \\ BiasHidden2 \end{pmatrix} = \begin{pmatrix} -0.725258 \\ 0.24629456 \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} BiasOuput3 \end{pmatrix} = \begin{pmatrix} 0.22297463 \end{pmatrix} \tag{4}$$

- ***(salvo que se guarde el modelo y los pesos y los bias cambian en cada ejecución)***

- El funcionamiento a nivel general sería el siguiente:
  - Cada una de las neuronas recogen el peso x el valor de cada feature en la primera capa o por el valor de salida de la anterior neurona, de cada una de las entradas y lo suma. En este caso la neurona Hidden1, recoge:

$$(Input1.W11 + Input2.W21) \tag{5}$$

  - Posteriormente le suma el Byas o sesgo:

$$(Input1.W11 + Input2.W21) + BiasHidden1 \tag{6}$$

    - Resultando esto la función de mapeo de la Hidden 1 y que será el valor que utilice la función de activación de la neuorona Hidden 1, en este caso la función de activación que hemos configurado es una sigmoide.
    - De igual forma sucedería para la Hidden 2 pero con los pesos W12 y W22 y con el Bias Hidden 2
    - Una vez ha sucedido esto pasariamos a la capa de salida en el que la neurona Output 3 tendría como funciín de mapeo:

$$(SalidaHidden1.W13 + SalidaHidden2.W23) + BiasOutput3 \tag{7}$$

    - En la valor de esta función de mapeo se le aplicara la función de activación, que en este caso al ser una sola neurona de salida, lo hemos identificado como un caso de regresión y no podemos usar softmax, por lo que hemos usado otra vez una sigmoide. Con esto sacaraemos un valor que se aproximara más a 0 o a 1