UNIVERSIDAD SANTO TOMÁS
PRIMER CLAUSTRO UNIVERSITARIO DE COLOMBIA
SECCIONAL TUNJA
VIGILADA MINEDUCACIÓN - SNIES 1732

REDES NEURONALES

RESUMEN ALGORITMOS DE ML

- Ningún algoritmo es el mejor para todos los problemas. Cada uno tiene unas ventajas y otras desventajas.
- El rendimiento de los algoritmos de ML depende mucho del tamaño y de la estructura de los datos.

RESUMEN ALGORITMOS DE ML

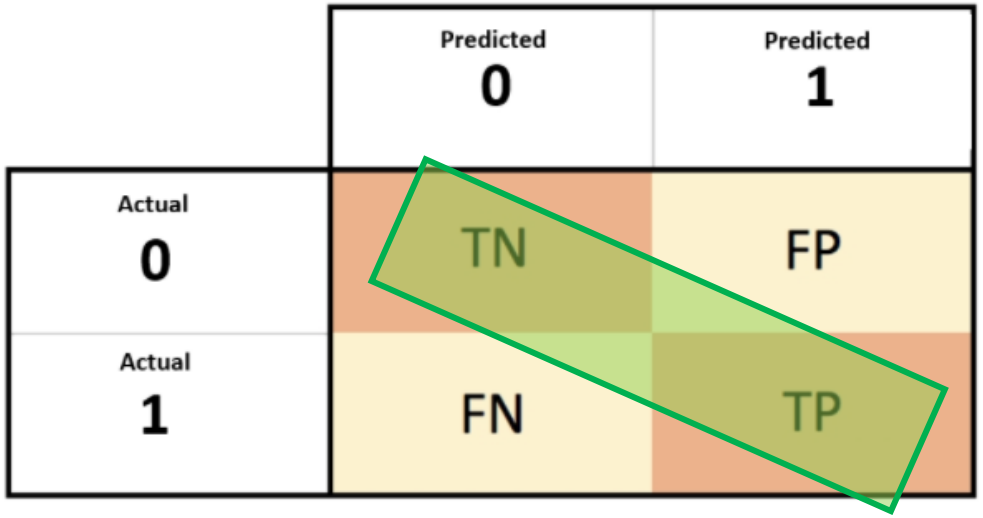
ALGORITMO	VENTAJAS	DESVENTAJAS
Regresión lineal	• Fácil de entender y aplicar. • Rápido de modelar. • Útil cuando el sistema no es complejo y se posee poca información. • Menos propenso al sobreajuste. • Útil para predicciones de economía y de márketing.	• No se puede aplicar a relaciones complejas. • Puede presentar valores atípicos.
Vectores de soporte	• Se pueden modelar relaciones complejas, no lineales. • Robusto al ruido. • Es útil para clasificación de texto e imágenes y para reconocimiento de escritura.	• Requiere memoria significativa y poder de procesamiento (si hay muchos datos toma demasiado tiempo en el entrenamiento. • Los parámetros del modelo son difíciles de interpretar.

RESUMEN ALGORITMOS DE ML

ALGORITMO	VENTAJAS	DESVENTAJAS
Árboles de decisión	• Fácil de interpretar • Rápido • Robusto al ruido • Preciso. • Es útil para diagnóstico médico y análisis de riesgo crediticio.	• Árboles complejos son difíciles de interpretar. • Puede presentar valores duplicados en el mismo subárbol.
Vecino más cercano.	• Es simple y fácil de interpretar. • Robusto. • Versátil (clasificación y regresión) • Preciso. • Útil en reconocimiento de escritura a mano, sistema de recomendación, detección de anomalías y clasificación.	• Alto costo computacional cuando el tamaño de los datos aumenta. • Funciona mejor en datasets pequeños y con pocas características.

EVALUACIÓN DEL ERROR

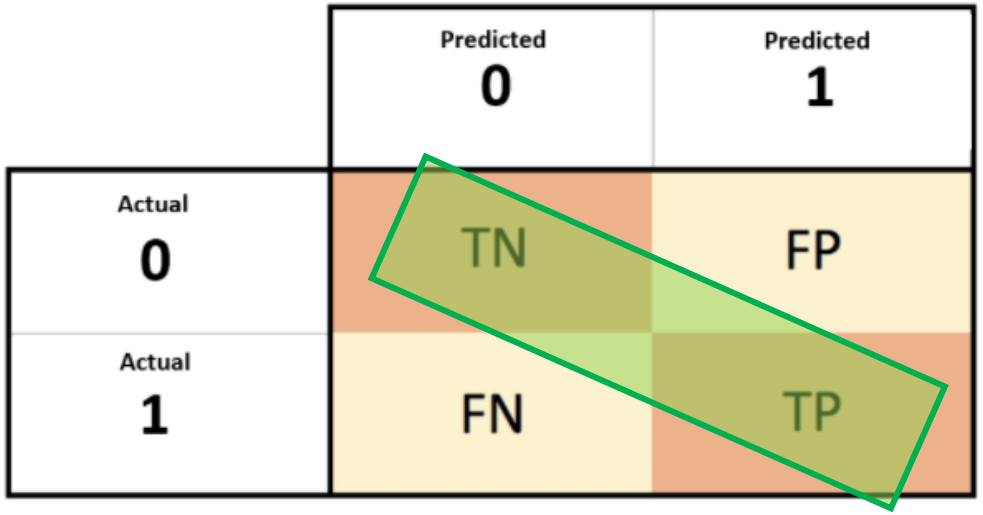
- **EXACTITUD**: número de predicciones correctas realizadas por el modelo por el número total de registros.



$$Exactitud = \frac{TN + TP}{TN + FP + FN + TP}$$

EVALUACIÓN DEL ERROR

- **EXACTITUD**: número de predicciones correctas realizadas por el modelo por el número total de registros.

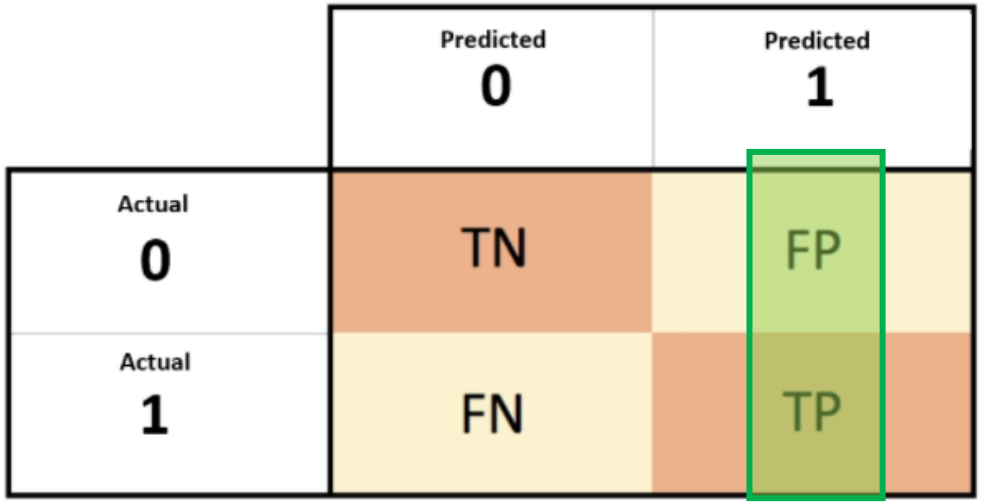


$$Exactitud = \frac{TN + TP}{TN + FP + FN + TP}$$

La mejor exactitud es el 100%, sin embargo, si los datos están desbalanceados (hay dos categorías y una tiene una alta representación y la otra muy baja) no es de fiar esta métrica.

EVALUACIÓN DEL ERROR

- **PRECISIÓN**: se evalúan los datos según su desempeño de predicciones positivas.

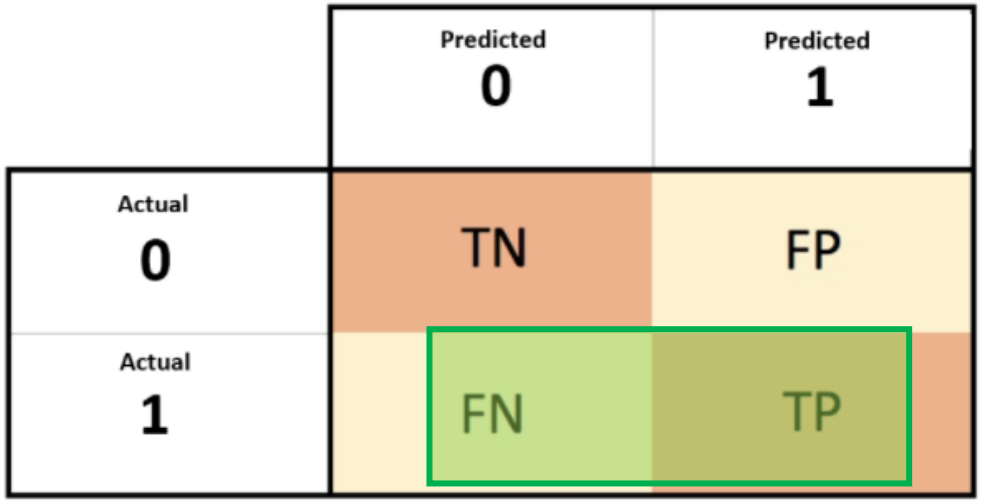


$$precisión = \frac{TP}{TP + FP}$$

Si los datos están desbalanceados (hay dos categorías y una tiene una alta representación y la otra muy baja) no es de fiar esta métrica.

EVALUACIÓN DEL ERROR

- **SENSIBILIDAD**: evalúa el número de predicciones positivas correctas por el total de positivos.

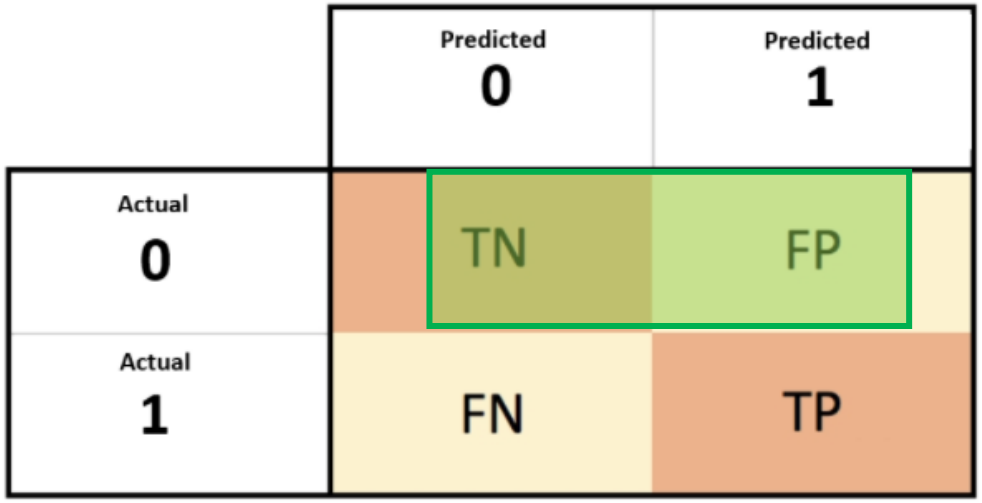


$$Sensibilidad = \frac{TP}{FN + TP}$$

Si los datos están desbalanceados (hay dos categorías y una tiene una alta representación y la otra muy baja) no es de fiar esta métrica.

EVALUACIÓN DEL ERROR

- **ESPECIFICIDAD**: evalúa el número de predicciones negativas correctas por el número total de negativos.



$$Especificidad = \frac{TN}{TN + FP}$$

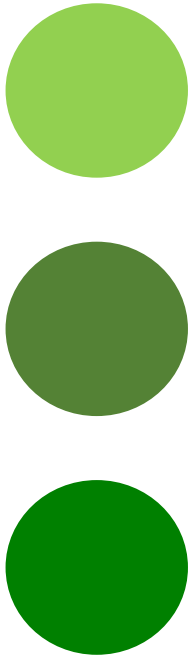
Si los datos están desbalanceados (hay dos categorías y una tiene una alta representación y la otra muy baja) no es de fiar esta métrica.

REDES NEURONALES

REDES NEURONALES

En la programación tradicional:

Entradas



Reglas y lógica computacional

Salidas

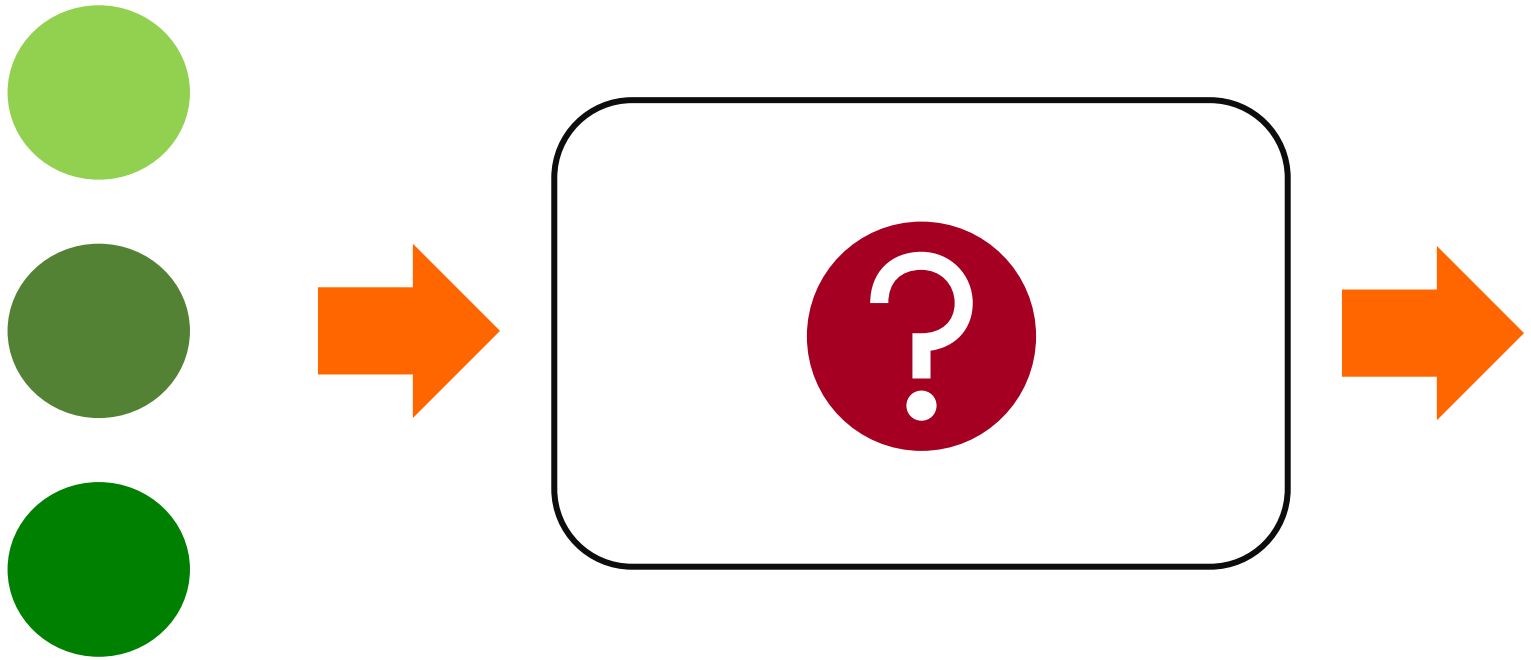
¡Siempre hacia lo alto!

REDES NEURONALES

En la Aprendizaje automático

Entradas

Salidas



¡Siempre hacia lo alto!

REDES NEURONALES

Una red neuronal se compone de neuronas artificiales que se comportan de forma similar a una neurona cerebral.

Al igual que las naturales, las neuronas artificiales contienen **ramificaciones** y un **núcleo** o **nodo**, que corresponden a las entradas de la neurona que provienen de otras neuronas. La comunicación entre neuronas se realiza a través de las ramificaciones.

REDES NEURONALES

Una red neuronal se compone de neuronas artificiales que se comportan de forma similar a una neurona cerebral.

Al igual que las naturales, las neuronas artificiales contienen **ramificaciones** y un **núcleo** o **nodo**, que corresponden a las entradas de la neurona que provienen de otras neuronas. La comunicación entre neuronas se realiza a través de las ramificaciones.

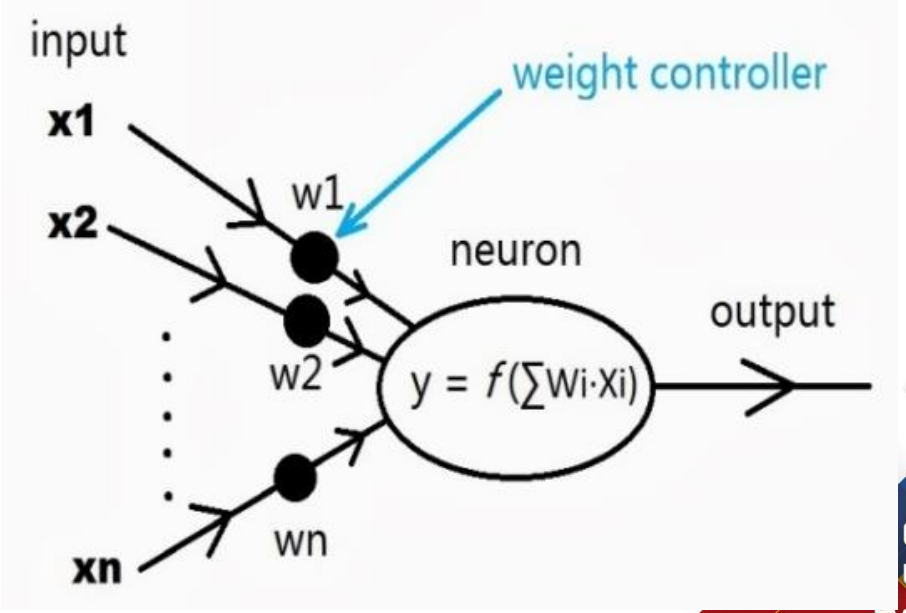


Imagen tomada de: https://quantdare.com/wp-content/uploads/2014/03/neurona-artificial1.png

REDES NEURONALES

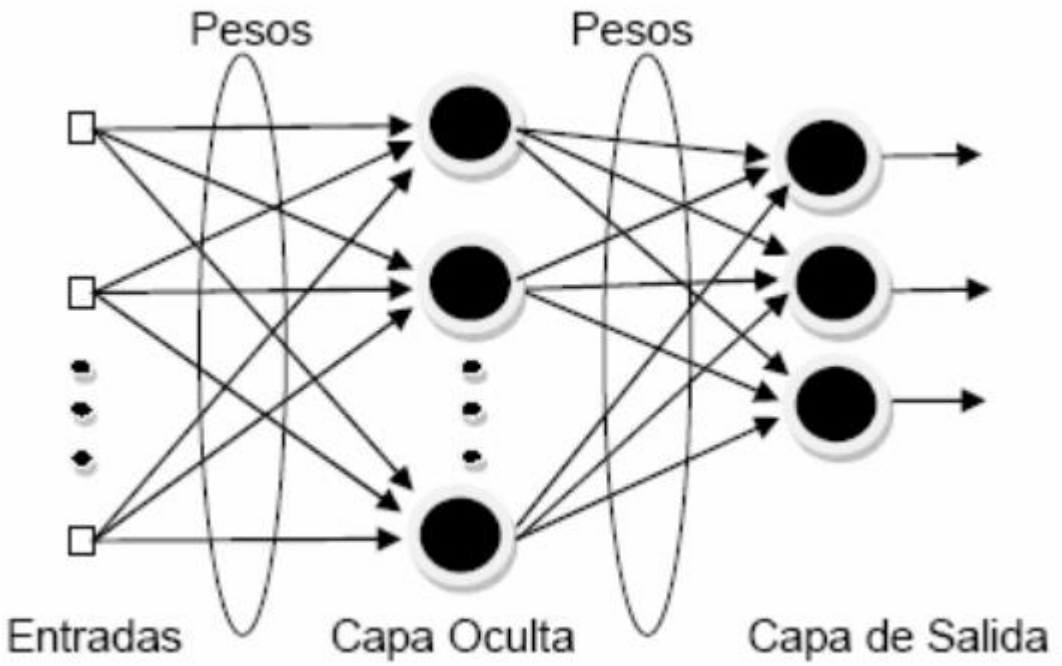


Imagen tomada de: Alves, Marcelo & Ferreira, Bruno & Leta, Fabiana. (2010). Evaluación de Parámetros de Rugosidad usando Análisis de Imágenes de Diferentes Microscopios Ópticos y Electrónicos. Información tecnológica. Vol 22. Pág 129-146. Doi: 10.4067/S0718-07642011000400014.

REDES NEURONALES

- Las redes neuronales se componen de **capas**, y cada capa puede tener una o más neuronas.
- Toda red neuronal tiene al menos una capa de entrada y una de salida.
- Las capas intermedias se conocen como capas ocultas.
- Las neuronas se comunican a través de **conexiones** y cada conexión tiene un **peso** asignado. El peso es un valor numérico que hace referencia a la importancia de la conexión entre las neuronas.
- Cada neurona, excepto las de entrada, tienen un sesgo, que también es un valor numérico.

REDES NEURONALES

Por ejemplo, si se tiene un sistema que quiere convertir el valor de grados Fahrenheit conociendo los grados Celcius.
Para ello, se pueden ingresar un conjunto de datos de °C y sus respectivos °F.
El sistema aprenderá la relación que existe y ajustará automáticamente el peso y el sesgo, para hacer las predicciones.

EJEMPLO REDES NEURONALES

REDES NEURONALES: TALLER

Para ello tener en cuenta que la fórmula para calcular los grados Fahrenheit a partir de grados Celcius:

$$°F = 1{,}8 * °C + 32$$

```
#importar librerías
import tensorflow as tf
import numpy as np
```

TensorFlow es una biblioteca de Python para computación numérica rápida, creada y lanzada por Google.

REDES NEURONALES: TALLER

Para ello tener en cuenta que la fórmula para calcular los grados Fahrenheit a partir de grados Celcius:

$$°F = 1{,}8 * °C + 32$$

```
#importar librerías
import tensorflow as tf
import numpy as np
```

TensorFlow es una biblioteca de Python para computación numérica rápida, creada y lanzada por Google.

```
#arreglos de datos de entrada y salida
celcius = np.array([-40,-20,-10,0,10,15,20,25,30,35], dtype=float)
fahrenheit = np.array([-40,-4,14,32,50,59,68,77,86,95], dtype=float)
```

Estos datos se calcularon con la ecuación de conversión.

REDES NEURONALES: TALLER

```
#crear modelo de red neuronal
#capas densas son las que tienen conexiones de todas las neuronas a todas las demás
#units = unidades o neuronas de la capa de salida
#input_shape = unidades o neuronas de la capa de entrada
capa = tf.keras.layers.Dense(units=1,input_shape=[1])
modelo = tf.keras.Sequential([capa])
```

REDES NEURONALES: TALLER

```
#crear modelo de red neuronal
#capas densas son las que tienen conexiones de todas las neuronas a todas las demás
#units = unidades o neuronas de la capa de salida
#input_shape = unidades o neuronas de la capa de entrada
capa = tf.keras.layers.Dense(units=1,input_shape=[1])
modelo = tf.keras.Sequential([capa])
```

```
#Compilar el modelo:
#El modelo Adam permite ajustar pesos y sesgos de manera eficiente
#de manera progresiva (vaya mejorando)
#El parámetro del modelo Adam es la tasa de aprendizaje

#loss= función de pérdida
modelo.compile(optimizer=tf.keras.optimizers.Adam(0.1), loss = 'mean_squared_error')
```

¡Siempre hacia lo alto!

REDES NEURONALES: TALLER

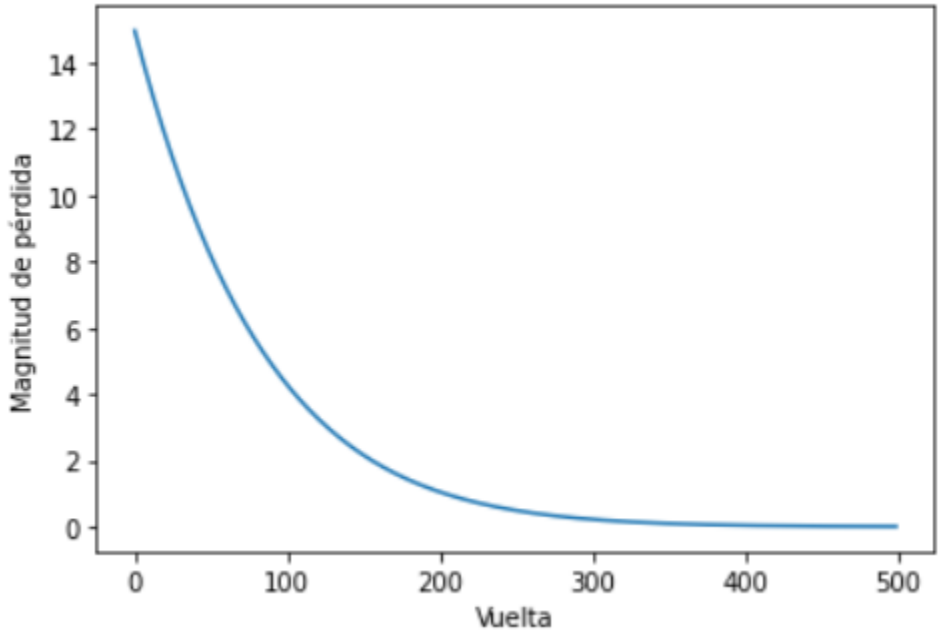
```
#Entrenar el modelo
#epochs = vueltas que dará la red (a prueba y error)
#verbose=false: no imprime tanta información
Data_train = modelo.fit(celcius, fahrenheit, epochs=500, verbose=False)
print("Modelo entrenado")  #para saber cuándo acaba
```

```
Modelo entrenado
```

REDES NEURONALES: TALLER

```
#Imprimir función de pérdida
#Permite saber qué tan mal se encuentra en cada vuelta que dió
import matplotlib.pyplot as plt
plt.xlabel("Vuelta")
plt.ylabel("Magnitud de pérdida")
plt.plot(Data_train.history["loss"])
```

```
[<matplotlib.lines.Line2D at 0x7ff1e71300d0>]
```



Se aprecia que la pérdida de datos disminuye en cada vuelta (va mejorando) y se estabiliza a partir de las 350 vueltas aproximadamente.

REDES NEURONALES: TALLER

```
#predicciones
print("********* Predicciones **********")
y_pred=modelo.predict([100])
print("El resultado es" , str(y_pred), " °F")
```

```
********* Predicciones **********
El resultado es [[212.0158]]  °F
```

Se predice el resultado con una entrada de 100°C. Este valor no está en los datos de entrenamiento. El modelo obtuvo el valor de 212,0158.
Al contrastar con un conversor o aplicando la fórmula, se tiene que 100°C equivalen a 212 °F, lo cual demuestra que el modelo funciona muy bien y proporciona predicciones muy aproximadas.

REDES NEURONALES: TALLER

```
#Imprimir variables
print("Peso y sesgo: ", capa.get_weights())
```

```
Peso y sesgo:  [array([[1.8009497]], dtype=float32), array([31.920828], dtype=float32)]
```

Acá se obtiene el valor del peso 1,8009497 y el sesgo es 31,920828.

Si comparamos esta información con la fórmula de conversión, apreciamos que tienen mucho en común:

$$°F = 1{,}8 * °C + 32$$

Esta fórmula la predijo el modelo, aprendiendo con el entrenamiento.

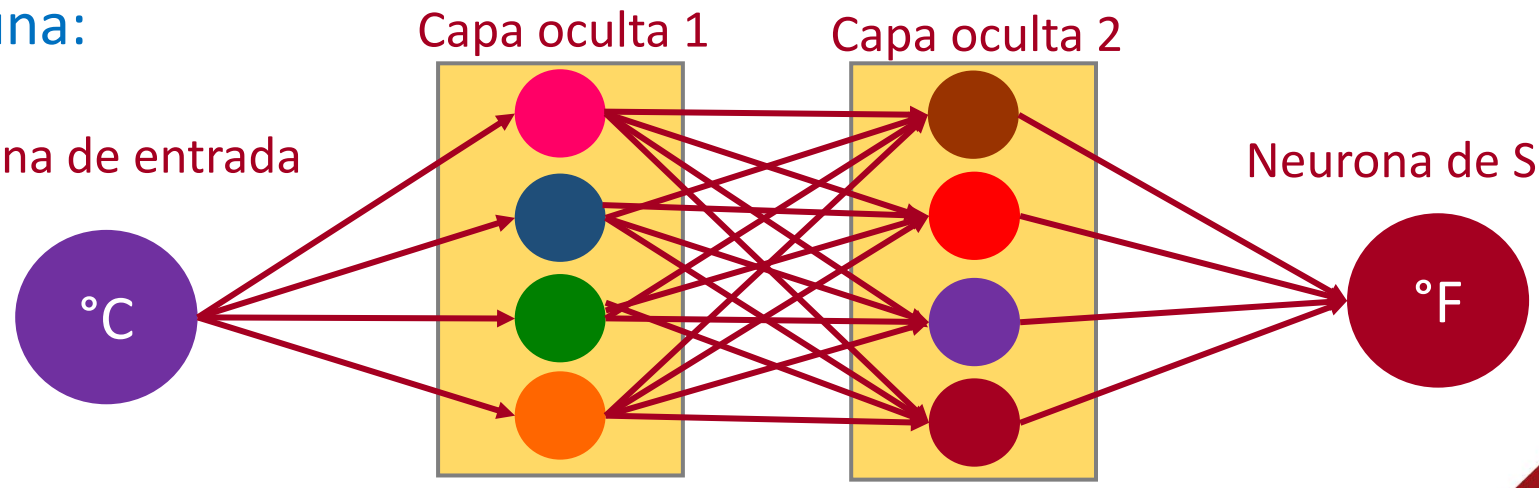
REDES NEURONALES: TALLER

Agreguemos más capas y veamos qué sucede:

```
#agregamos más capas y más neuronas
oculta1 = tf.keras.layers.Dense(units=4,input_shape=[1])
oculta2 = tf.keras.layers.Dense(units=4)
salida = tf.keras.layers.Dense(units=1)
modelo = tf.keras.Sequential([oculta1, oculta2, salida])
```

```
modelo.compile(optimizer=tf.keras.optimizers.Adam(0.1), loss = 'mean_squared_error')
```

En este caso agregamos 2 capas ocultas (intermedias) con 4 neuronas cada una:



¡Siempre hacia lo alto!

REDES NEURONALES: TALLER

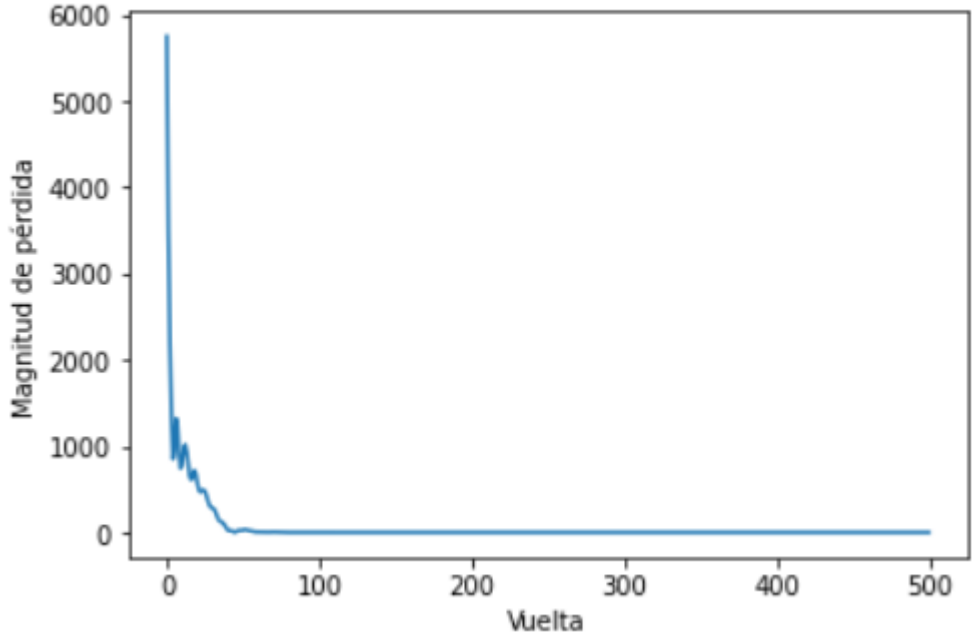
```
#Entrenar el modelo (tarda un poco más)
Data_train = modelo.fit(celcius, fahrenheit, epochs=500, verbose=False)
print("Modelo entrenado")  #para saber cuándo acaba
```

```
Modelo entrenado
```

REDES NEURONALES: TALLER

```
#Imprimir función de pérdida
#Permite saber qué tan mal se encuentra en cada vuelta que dió
import matplotlib.pyplot as plt
plt.xlabel("Vuelta")
plt.ylabel("Magnitud de pérdida")
plt.plot(Data_train.history["loss"])
```

```
[<matplotlib.lines.Line2D at 0x7ff1e7d39e90>]
```



Se aprecia que la pérdida se estabilizó antes de las 100 vueltas. Esto indica que con más capas se logra una eficiencia más rápido.

REDES NEURONALES: TALLER

```
#Aprendizaje más rápido
```

```
#predicciones
print("********* Predicciones **********")
y_pred=modelo.predict([100])
print("El resultado es" , str(y_pred), " °F")
```

```
********* Predicciones **********
El resultado es [[212.]]  °F
```

La predicción lograda es más precisa en este caso.

REDES NEURONALES: TALLER

```
#Imprimir variables
print("Peso y sesgo: ", salida.get_weights())
```

```
Peso y sesgo:  [array([[ 0.691582  ],
       [-0.73682606],
       [-0.0782465 ],
       [-0.6059677 ]], dtype=float32), array([3.7902393], dtype=float32)]
```

En este caso los valores que se obtienen de peso (4 neuronas) y sesgo no son tan fáciles de entender, y tampoco hace falta entenderlos. Basta con conocer que la predicción es muy precisa y no hace falta entender cómo se consiguió.

REFERENCIAS

- https://www.iartificial.net/redes-neuronales-desde-cero-i-introduccion/#Redes_neuronales
- https://www.youtube.com/watch?v=iX_on3VxZzk