Ejemplo de redes neuronales

Revise un ejemplo sencillo con Python haciendo uso de las redes neuronales, para tal fin imagine que no conoce la fórmula de conversión de grados Celsius a Fahrenheit, solo conoce los datos Celsius y sus respectivos datos Fahrenheit que se muestran a continuación:

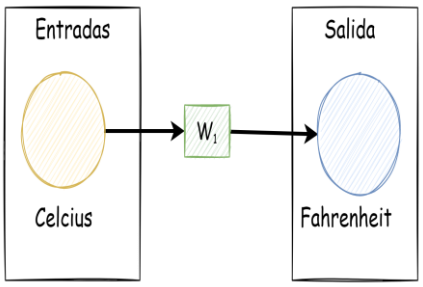
Tabla 1. *Datos Celsius y Fahrenheit*

Celsius	Fahrenheit
-20	-4
-5	23
0	32
5	41
10	50
20	68
38	100

Con este ejemplo se realizará la red neuronal simple, en la cual se tiene solo una capa de entrada, una capa de salida y una sola conexión, cada conexión tiene un peso asignado y cada neurona diferente a las de la capa de entrada tiene un sesgo. Suponga que el peso es W1 = 1.8 y el sesgo es 4.5 escogidos aleatoriamente.

Si se desea convertir 20 grados Celsius a Fahrenheit, estos 20 grados Celsius se colocan en la primera neurona, estos se multiplican por el peso W1 = 1.8, este valor llega a la siguiente neurona, aquí se le suma el sesgo y este será el resultado como muestra la fórmula:

Figura1. *Representación de una red neuronal simple.*



$$\text{Celsius} * \text{W1} + \text{sesgo} = 20 * 1.8 + 4.5 = 40.5$$

El peso W1 y el sesgo inicialmente se inician en forma aleatoria, en el caso de los 20 grados Celsius esta red neuronal predice un resultado de 40.5, el cual es completamente incorrecto, puesto que el verdadero valor es 68, pero la verdadera potencia de la red neuronal es que esta aprenderá con los datos que conoce, y ajustará los valores de los pesos y sesgos con los datos conocidos y así predecirá con mucha más exactitud los valores de salida.

Para continuar con el proceso se debe hacer la importación de las librerías necesarias, en este caso TensorFlow.

```
import tensorflow as tf
```

TensorFlow es una librería que contiene algoritmos de inteligencia artificial con Google y con Numpy, se trabajan arreglos numéricos *import* numpy *as* np.

Posteriormente, se debe declarar en Python los arreglos conocidos que son grados Celsius y sus respectivos grados Fahrenheit. Estos datos son usados por la red neuronal para aprender, para tal fin se utiliza el siguiente comando:

```
celcius    = np.array ( [-20, -5, 0, 5, 10, 20, 38], dtype =float )
fahrenheit = np.array ( [-4, 23, 32, 41, 50, 68,100], dtype =float )
```

- **Construcción de la red neuronal**

Para construir la red neuronal en este ejemplo se usará *framework keras*, la cual hace esta construcción de manera simple y hace que se evite crear líneas de código. Con *keras* se puede especificar las capas de entrada y salida por separado, en este caso con *keras* solo se crea la capa de salida.

Se crea una capa de tipo densa, esto significa que cada neurona de una capa se conecta con absolutamente todas las neuronas de la siguiente capa. En el caso del ejemplo solo se cuenta con dos neuronas, la conexión es una sola, units = 1, significa que la capa de salida tiene una neurona y la variable *input_shape* dice que la red tiene una capa de entrada con una sola neurona.

```
capa = tf.keras.layers.Dense(units = 1, input_shape =[1] )
modelo = tf.keras.Sequential([capa])
```

Con esa capa se puede construir el modelo y en este caso se crea un modelo de tipo secuencial que permite unir las capas en secuencia.

- **Compilación del modelo**

En este paso se prepara el modelo para ser entrenado, en ese caso se utiliza un optimizador llamado Adam, que es un algoritmo que ajusta pesos y sesgos de la manera más eficiente posible de acuerdo con el argumento, en este caso es 0.1 que define la ruta de aprendizaje. Si este número es pequeño ajusta los pesos y el sesgo es un poco lento, de lo contrario si es muy grande puede generar errores al fijar esos pesos y sesgos. El otro parámetro es la función de pérdida, generalmente se escoge el error cuadrático medio.

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

- **Entrenamiento del modelo**

Se usa la función ***fit*** con argumentos de datos de entrada y salida, en este caso datos Celsius y datos Fahrenheit, respectivamente, y la cantidad de vueltas en este caso puede ser 1000, una vuelta significa que revisa los datos de entrada una sola vez y fija datos de peso y sesgo. Para llegar a un valor correcto de los pesos, se debe realizar esto varias veces.

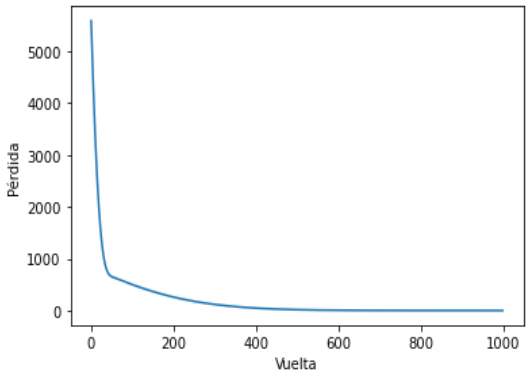
```
print ("Inicio de entrenamiento")
modeloNeuronal =  modelo.fit (celcius,fahrenheit, epochs=1000, verbose = False)
print ("Fin de entrenamiento")
```

- **Función de pérdida**

Esta función permite determinar qué tan alejados están los resultados en cada vuelta que dio al revisar los datos.

```
import matplotlib.pyplot as plt
plt.xlabel ("Vuelta")
plt.ylabel ("Pérdida")
plt.plot (modeloNeuronal.history ["loss"])
```

Figura 2. *Gráfico que representa la función de pérdida*



Se observa que entre más vueltas se ejecutaron el error fue disminuyendo, se parametrizan 1000 vueltas, pero en la vuelta 500 la pérdida se estabilizó.

- **Realizar predicciones con el modelo red neuronal**

Con el modelo construido ya se pueden realizar predicciones de forma muy sencilla como, por ejemplo, con esa red neuronal convertir 80 grados Centígrados a Fahrenheit.

```
print ("Prediccion ")
resultado = modelo.predict ([80])
print ("resultado en grados fahrenheit " + str(resultado) )

Predicción
1/1 [==============================] - 0s 44ms/step
resultado en grados Fahrenheit [[175.52505]]
```

El resultado es 175.52505 grados Fahrenheit

Con cálculos reales y usando la función de conversión de grados Centígrados a Fahrenheit el resultado es 176, el cual es muy cercano al valor calculado usando la red neuronal, esto significa que el modelo está muy bien entrenado.

- **Obtención de parámetros definitivos de peso y sesgo del modelo**

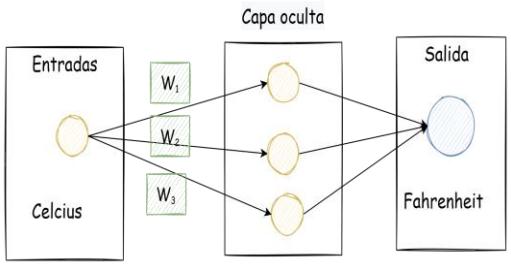
Después del entrenamiento, los datos definitivos son peso = 1.79 y sesgo 31.95 que son datos muy similares a la función de conversión de grados Centígrados a Fahrenheit. La red neuronal por tanto obtuvo la fórmula de conversión de grados Centígrados a Fahrenheit.

```
print ("variables peso y sesgo")
print (capa.get_weights())
variables peso y sesgo
[array([[1.7947289]], dtype=float32),
array([31.946745], dtype=float32)].
```

- **Incremento de otras capas**

Ahora se incrementa una capa intermedia entre la entrada y salida con tres neuronas cada una, estas capas son llamadas capas ocultas.

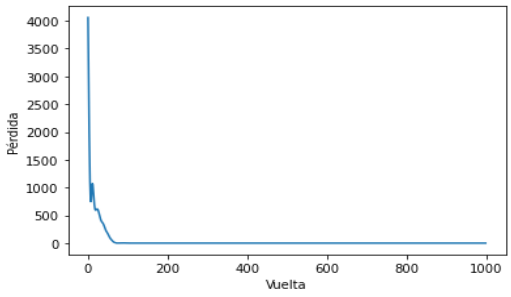
Figura 5. *Diagrama del incremento de otras capas*



El código a utilizar es:

```
oculta1 = tf.keras.layers.Dense(units = 3,
input_shape =[1] )
salida = tf.keras.layers.Dense(units = 1)
modelo = tf.keras.Sequential([oculta1,
salida])modelo.compile  (
   optimizer = tf.keras.optimizers.Adam(0.1),
   loss = 'mean_squared_error'
)print ("Inicio de entrenamiento")
modeloNeuronal =  modelo.fit (celcius,fahrenheit,
epochs=1000, verbose = False)
print ("Fin de entrenamiento")import
matplotlib.pyplot as plt
plt.xlabel ("Vuelta")
plt.ylabel ("Pérdida")
plt.plot (modeloNeuronal.history ["loss"])
```

Figura 4. *Representación del resultado del incremento de otras capas*



Con esta capa intermedia se observa que desde la vuelta 50 el algoritmo ya no aprendió más, en conclusión, aprendió más rápidamente que cuando se usó una sola capa,