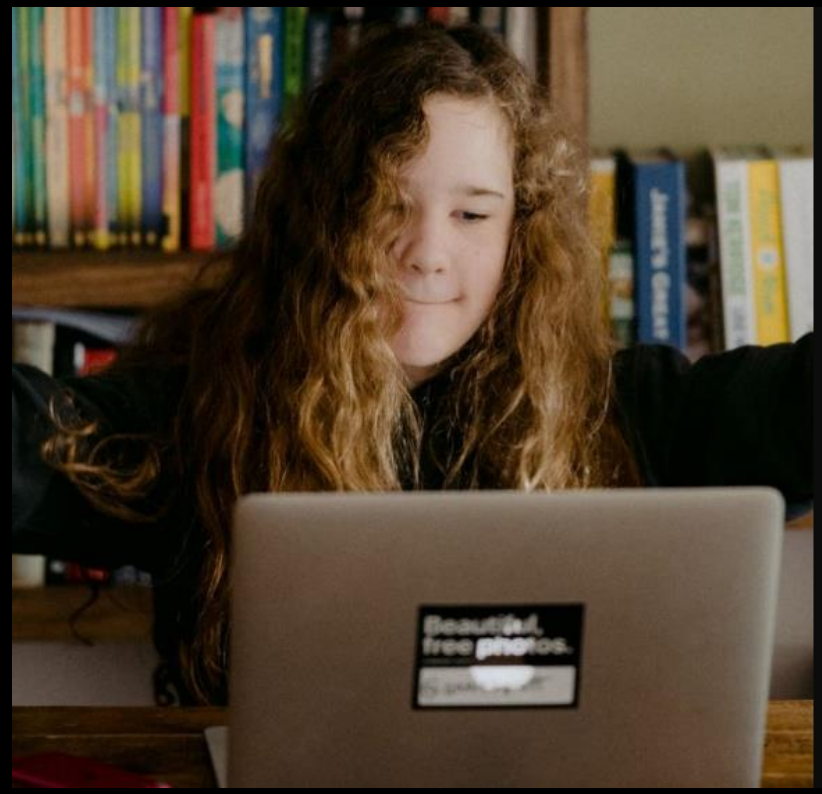
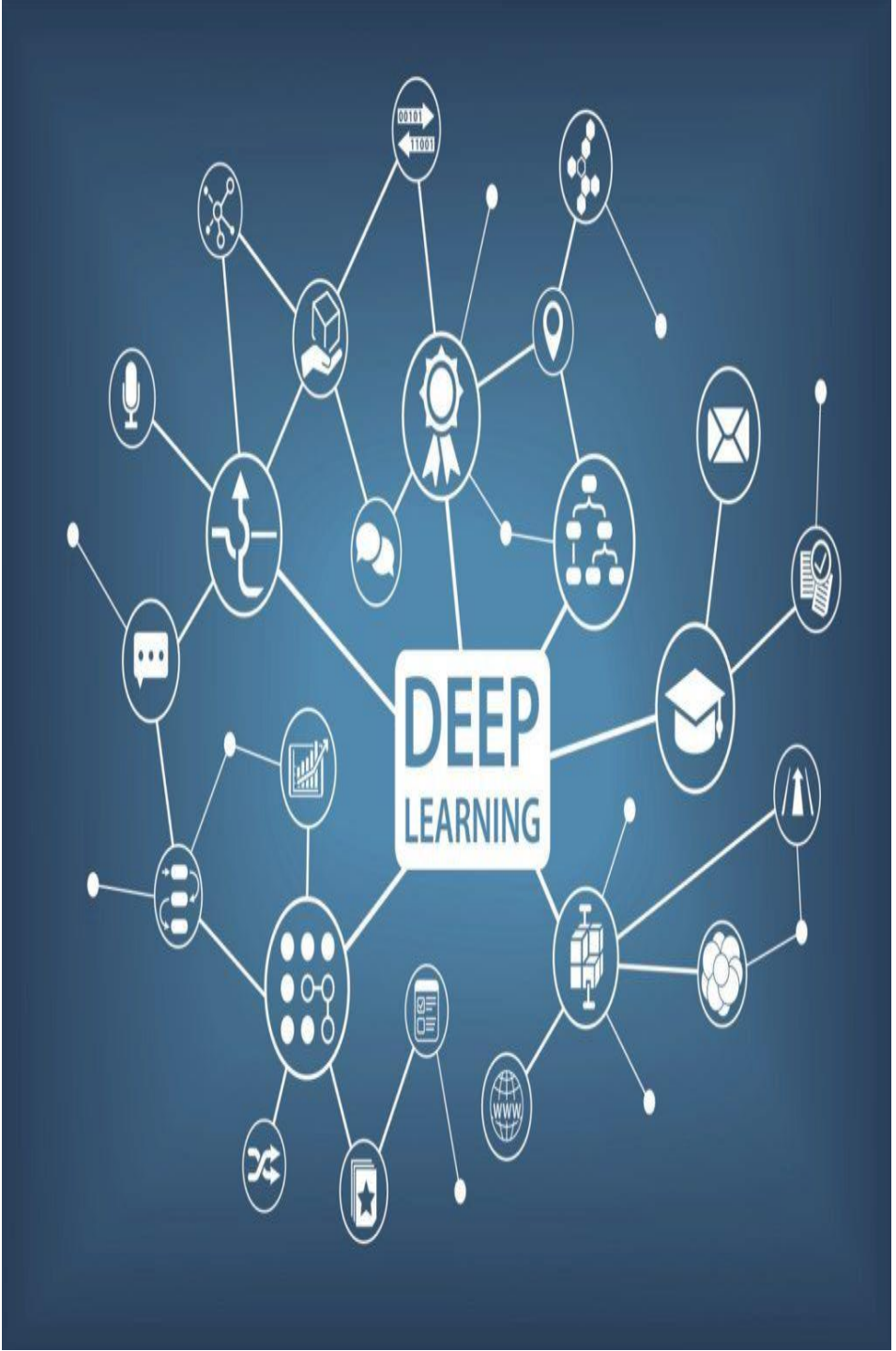
Ruta de Aprendizaje



COMPUTER VISION

- **6 lecciones** de Deep Learning

A single Neuron

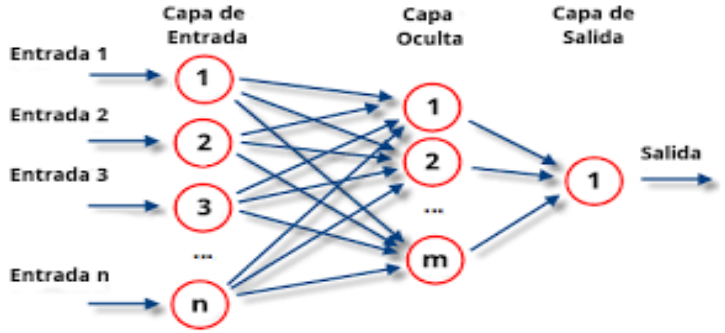


Lesson 1.1: Intro to “Deep Learning”

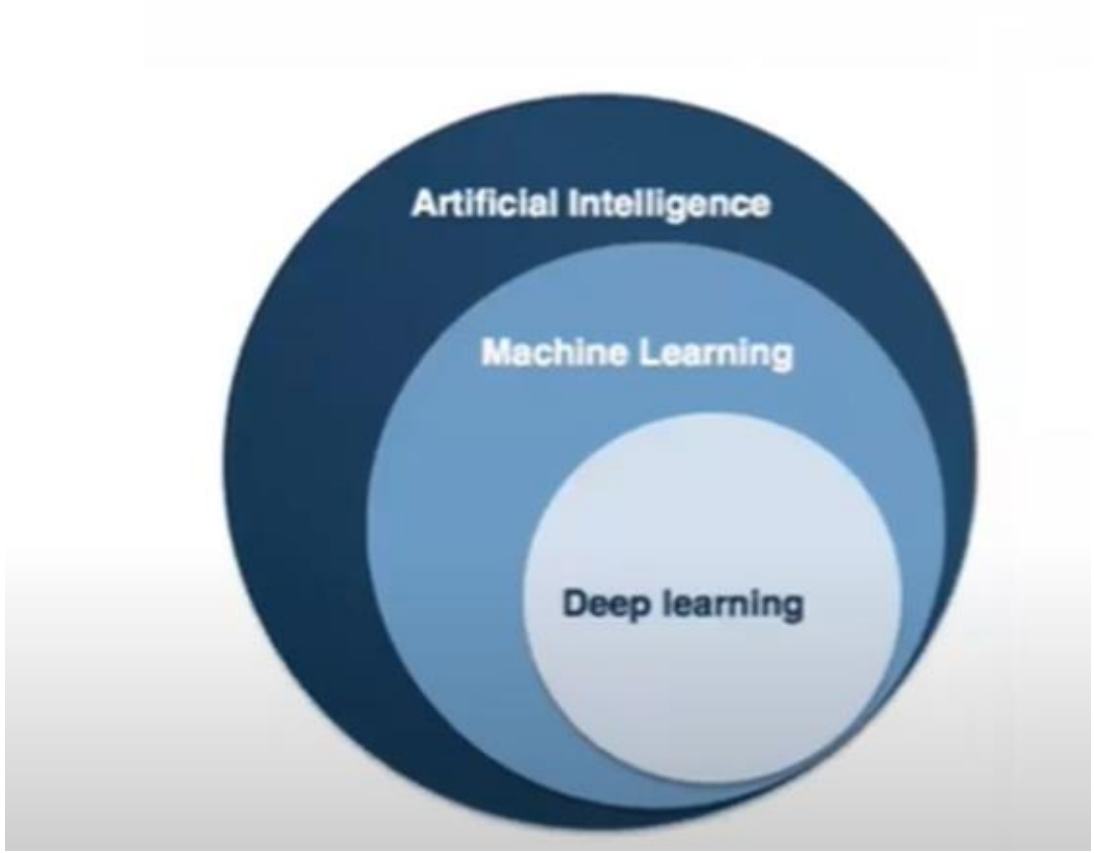
Deep Learning

El Deep learning es un tipo de Machine Learning que entrena a una computadora para que realice tareas como las hacemos los seres humanos, como el reconocimiento del habla, la identificación de imágenes o hacer predicciones.

En lugar de organizar datos para que se ejecuten a través de ecuaciones predefinidas, el deep learning configura parámetros básicos acerca de los datos y entrena a la computadora para que aprenda por cuenta propia reconociendo patrones mediante el uso de muchas capas de procesamiento.



Lesson 1.1: Intro to “Deep Learning”



La Inteligencia Artificial (IA) es la **combinación de algoritmos, la simulación de procesos de inteligencia humana por parte de máquinas**, en especial sistemas informáticos.

Inteligencia Artificial

Máquinas simulando el comportamiento y razonamiento de los humanos. Para ello, usan diferentes técnicas, entre ellas, Machine Learning.

Machine Learning

Es la capacidad de las computadoras para aprender por sí mismas a partir de datos y experiencia. En su uso más complejo utiliza Deep Learning.

Deep Learning

Algoritmos que permiten clasificar y relacionar grandes volúmenes de información imitando las redes neuronales.

Lesson 1.1: Intro to “Deep Learning”

Librerías populares de Python

pandas

Contiene módulos de análisis y estructuras de datos de alto rendimiento

NumPy

Se ocupa de la computación científica y matemática con Python

learn

Biblioteca de software Machine Learning para análisis de datos, minería y Machine Learning

TensorFlow

Ayuda a crear módulos y funciones de Deep Learning, como detección / reconocimiento de imágenes

matplotlib

Matplotlib es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy.

Keras

A deep learning library

Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. Es capaz de ejecutarse sobre TensorFlow.

Herramientas de Ciencia de Datos

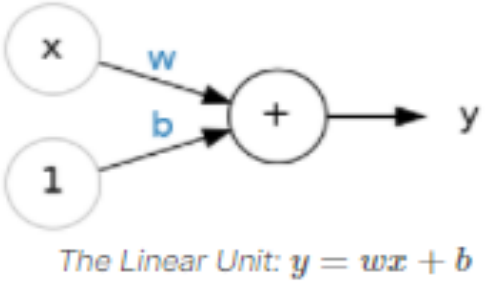
jupyter

SPYDER

colab

scikit learn

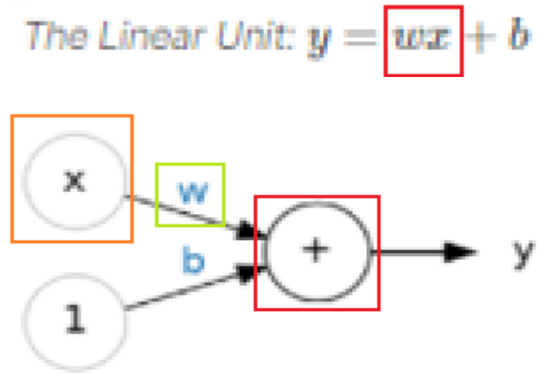
Lesson 1.2: The linear unit



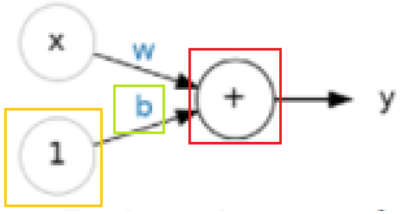
The Linear Unit: $y = wx + b$

x = entrada
w = weights (peso)
y = salida
b = bias (sesgo)

La entrada es **x**. Su conexión con la neurona tiene un peso que es **w**. Siempre que un valor fluya a través de una conexión, multiplique el valor por el peso de la conexión. Para la entrada **x**, lo que llega a la neurona es **w * x.** Una red neuronal "aprende" modificando sus pesos.



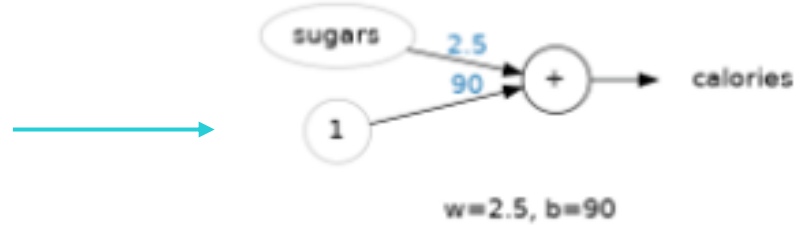
La **b** es un tipo especial de ponderación que llamamos **bias** (sesgo). El sesgo no tiene ningún dato de entrada asociado; en cambio ponemos un **1** en el diagrama para que el valor que llegue a la neurona sea simplemente b **(1 * b = b).**



La **y** es el valor que finalmente produce la neurona. Para obtener la salida, la neurona suma todos los valores que recibe a través de sus conexiones. La activación de esta neurona es **y = w * x + b**, o como fórmula **y = wx + b.**

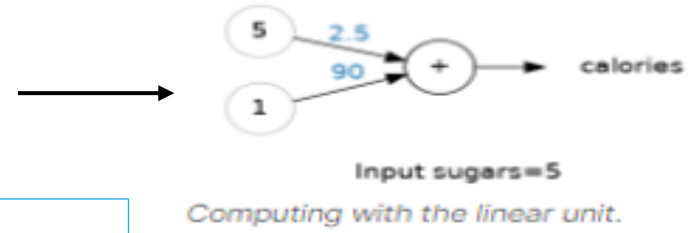
Lesson 1.3: Example – The linear unit as a model

Pensemos en cómo podría funcionar eso en un conjunto de datos de 80 cereales. Al entrenar un modelo con '**sugars**' (gramos de azúcares por porción) como entrada y '**calories**' (calorías por porción) como salida,



podríamos encontrar que el sesgo es **b = 90** y el peso es **w = 2.5**.

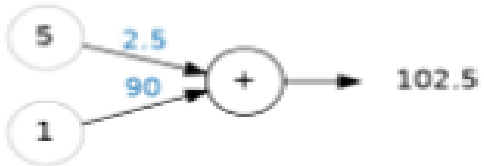
Podríamos estimar el contenido calórico de un cereal con 5 gramos de azúcar por ración así:



Computing with the linear unit.

Y, comparando nuestra fórmula, tenemos
calories = 2.5 × 5 + 90 = 102.5, tal como esperamos.

The Linear Unit: $y = wx + b$



Multiply inputs by weights and sum.

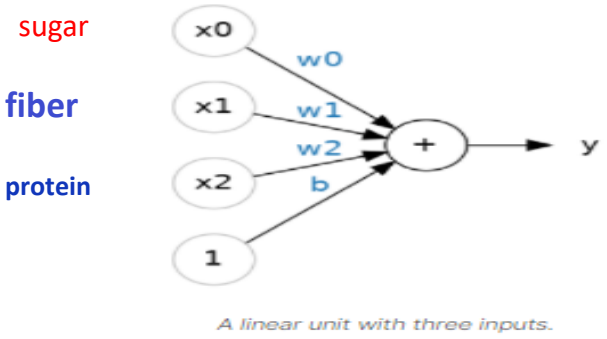
x = 5
w = 2.5
y = calories
b = 90

Lesson 1.4: Multiple inputs

El conjunto de datos de 80 cereales tiene muchas más características que solo ‘sugars'. ¿Qué pasaría si quisiéramos expandir nuestro modelo para incluir elementos como el contenido de ‘fiber’ o ‘protein’?

Podemos simplemente agregar más conexiones de entrada a la neurona, una para cada característica adicional.

Para encontrar la salida, multiplicaríamos cada entrada por su peso de conexión y luego las sumaríamos todas.



A linear unit with three inputs.

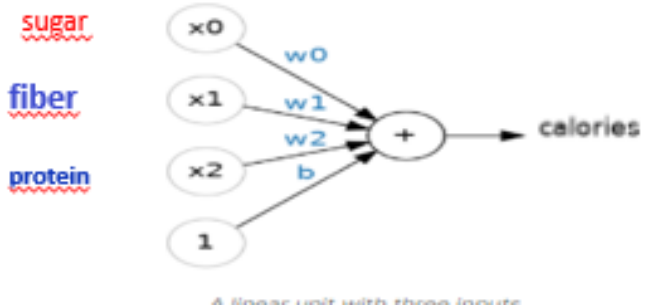
La fórmula para esta neurona sería **y = w0x0 + w1x1 + w2x2 + b**.

Una unidad lineal con dos entradas encajará en un plano, y una unidad con más entradas encajará en un hiperplano

Lesson 1.5: Linear units in Keras

La forma más sencilla de crear un modelo en Keras es a través de **keras.Sequential**, que crea una red neuronal como una pila de capas.

Podríamos definir un modelo lineal aceptando tres características de entrada (‘sugars', ‘fiber' y 'protein’) y produciendo una única salida ('caloríes') así:




```
from tensorflow import keras
from tensorflow.keras import layers

# Create a network with 1 linear unit
model = keras.Sequential([
    layers.Dense(units=1, input_shape=[3])
])
```

input_shape = representa las entradas ‘sugars’, ‘fiber’ y ‘protein’
units = representa la salida ‘calories’

Ejercicio:

Si tienes 6 entradas: ‘dolor de cabeza’, ‘fiebre’, ‘tos’, ‘erupción cutánea’, ‘conjuntivitis’ y ‘náuseas’
1 salida: ‘diagnóstico’

input_shape=[**?**]
units=**?**

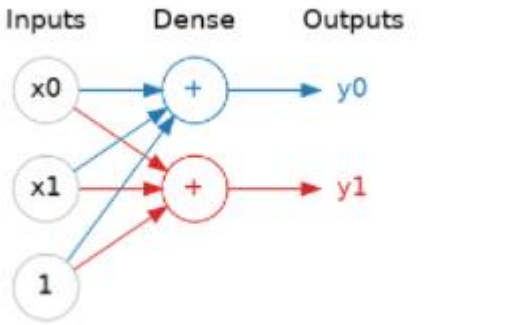
LABORATORIO 1

Lesson
#2

Deep Neural Networks

Lesson 2.1: Layers

Las redes neuronales suelen organizar sus neuronas en **layers** (capas). Cuando reunimos unidades lineales que tienen un conjunto común de entradas, obtenemos una capa densa.



A dense layer of two linear units receiving two inputs and a bias.

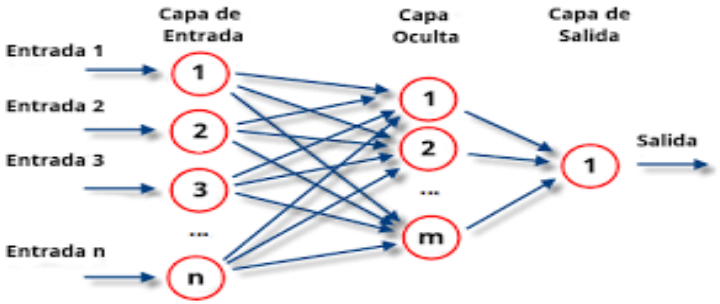
Código anterior

```
from tensorflow import keras
from tensorflow.keras import layers

# Create a network with 1 linear unit
model = keras.Sequential([
    layers.Dense(units=1, input_shape=[3])
])
```

Las capas **Dense** son la capas de cálculo de que conectan cada neurona en una capa con todas las salidas de la capa anterior.

A través de una pila profunda de capas, una red neuronal puede transformar sus entradas de formas cada vez más complejas. En una red neuronal bien entrenada, cada capa es una transformación que nos acerca un poco más a una solución.



Lesson 2.2: The activation function

La función de activación se encarga de devolver una salida a partir de un valor de entrada, normalmente el conjunto de valores de salida en un rango determinado como (0,1) o (-1,1).

- Función de activación Tangente hiperbólica
- Función de activación ReLU
- Función de activación Sigmoide
- Función de activación Softmax

Lesson 2.2.1: Types of activation functions

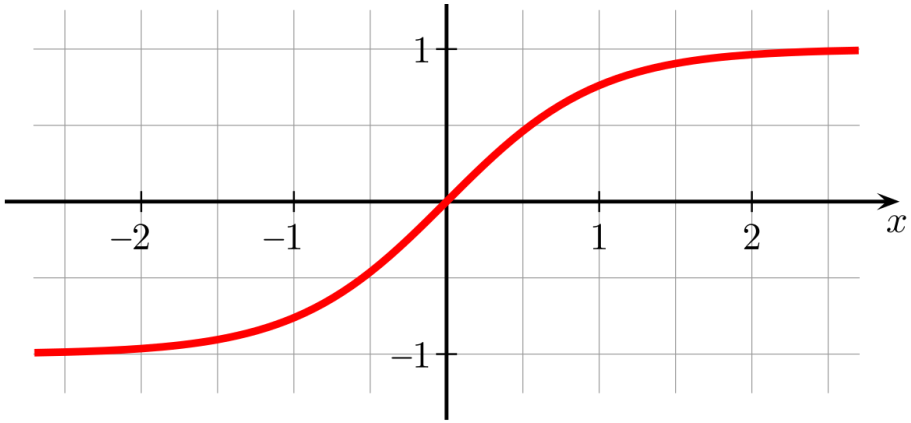
Función de activación “Tangente hiperbólica”

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Función tangente hiperbólica

La función tangente hiperbólica transforma los valores introducidos a una escala (-1, 1), donde los valores altos tienen de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a -1.

Características:



- Muy similar a la sigmoide
- Satura y mata el gradiente
- Lenta convergencia
- Centrada en 0
- Esta acotada entre -1 y 1
- Se utiliza para decidir entre una opción y la contraria
- Buen desempeño en redes recurrentes

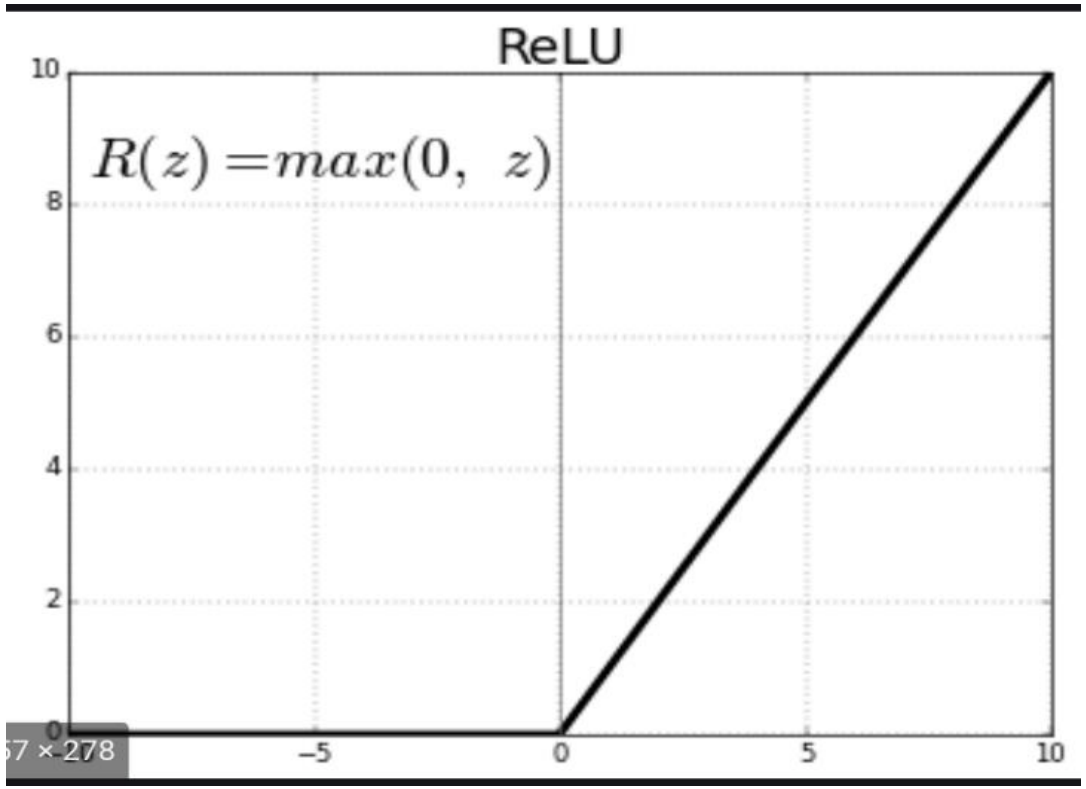
Lesson 2.2.1: Types of activation functions

$$f(x) = \max(0, x) = \begin{cases} 0 \ for \ x < 0 \\ x \ for \ x \geq 0 \end{cases}$$

Función ReLU

Función de activación “ReLU – Rectified Lineal Unit”

La función ReLU transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran.



Características:

- Activación Sparse – solo se activa si son positivos
- No está acotada
- Se pueden morir demasiadas neuronas
- Se comportan bien con imágenes
- Buen desempeño en redes convolucionales

Note: Tiene variantes de activación cómo ‘elu’, ‘selu’, ‘switch’ entre otras las cuáles se pueden usar en Keras

Lesson 2.2.1: Types of activation functions

Función de activación “Sigmoide”

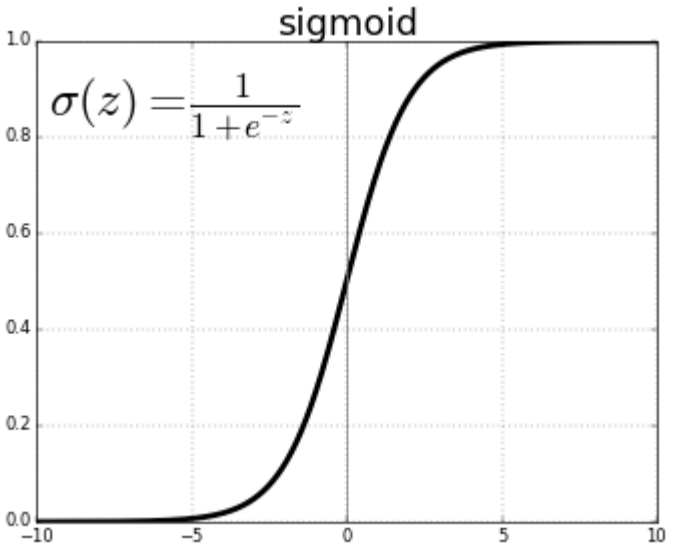
$$f(x) = \frac{1}{1 - e^{-x}}$$

Función Sigmoide

La función de activación sigmoide transforma los valores introducidos a una escala (0,1), donde los valores altos tienen de manera asintótica a 1 y los valores muy bajos tienden de manera asintótica a 0.

Características:

- Satura y mata el gradiente
- Lenta convergencia
- No esta centrada en el cero
- Esta acotada entre 0 y 1
- Buen rendimiento en la última capa



Lesson 2.2.1: Types of activation functions

Función de activación “Softmax”

$$f(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Función Softmax

La función Softmax transforma las salidas a una representación en forma de probabilidades, de tal manera que el sumatorio de todas las probabilidades de las salidas de 1.

Características:

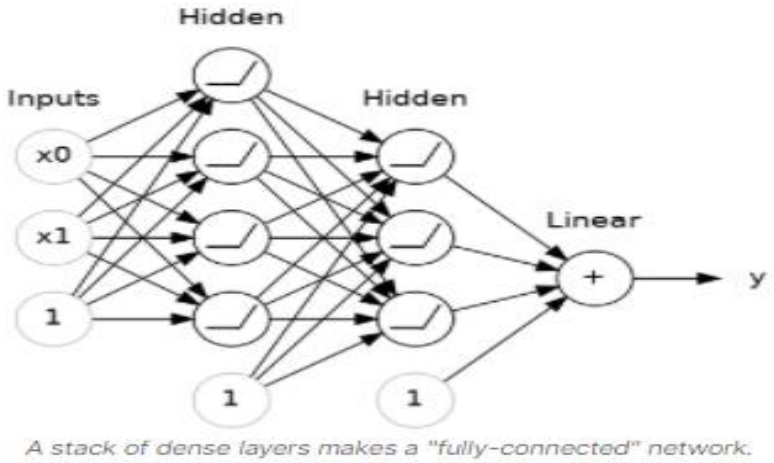
- Se utiliza cuando queremos tener una representación en forma de probabilidades
- Esta acotada entre 0 y 1
- Muy diferenciable
- Se utiliza para normalizar tipo multiclase
- Buen rendimiento en las últimas capas

Note: Utilice las funciones de activación sigmoide y softmax para realizar una clasificación de clases múltiples y una clasificación de etiquetas múltiples. Generalmente, softmax se utiliza para la clasificación de clases múltiples. sigmoide se utiliza para la clasificación de dos o múltiples etiquetas.

Lesson 2.3: Stacking Dense Layers

Las capas antes de la capa de salida a veces se llaman **hidden** (ocultas), ya que nunca vemos sus salidas directamente.

Ahora, observe que la capa final (salida) es una unidad lineal (es decir, sin función de activación). Eso hace que esta red sea apropiada para una tarea de regresión, donde estamos tratando de predecir algún valor numérico arbitrario.



A stack of dense layers makes a "fully-connected" network.

```
#crear una red neuronal perceptrón multicapa

model = keras.Sequential([

#crear capa de entrada
    layers.Dense(units=1, activation = 'relu', input_shape=[15]),

#crear capa oculta
     layers.Dense(units=1, activation = 'relu'),

#crear capa de salida

     layers.Dense(units=1, activation='sigmoid'),

   ])
```

Otras tareas (como la clasificación) pueden requerir una función de activación en la salida.

Lesson 2.4: Building Sequential Models

El modelo secuencial que hemos estado usando conectará una lista de capas en orden de la primera a la última: la primera capa obtiene la entrada, la última capa produce la salida. Esto crea el modelo en la figura anterior:

In [1]:

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    # the hidden ReLU layers
    layers.Dense(units=4, activation='relu', input_shape=[2]),
    layers.Dense(units=3, activation='relu'),
    # the linear output layer
    layers.Dense(units=1),
])
```

Note: Ahora, observe que la capa final (salida) es una unidad lineal (es decir, sin función de activación). Eso hace que esta red sea apropiada para una tarea de regresión, donde estamos tratando de predecir algún valor numérico arbitrario.

Note: Otras tareas (como la clasificación) pueden requerir una función de activación en la salida.

Diferencias entre algoritmos de Clasificación y Regresión

- Predecir si una persona está infectada con dengue

- Predecir el precio de una casa
- Eficiencia de combustible de un automóvil

Ejercicios

Si una nueva película que está saliendo en los cines te va a gustar o no

---

En caso de que si te guste la nueva película, ¿cuántas veces la verás?

LABORATORIO 2